Assignment

❖ **Extra Lab Practice 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.

Answer:

- **Create database and books table**

  Step 1: Create database

  ```
  CREATE DATABASE library_db;
  ```

  Step 2: Use the database

  ```
  USE library_db;
  ```

  Step 3: Create books table

  ```
  CREATE TABLE books (
     book_id INT PRIMARY KEY,
     title VARCHAR(100),
     author VARCHAR(50),
     publisher VARCHAR(50),
     year_of_publication YEAR,
     price DECIMAL(6,2)
  );
  ```

  Step 4: Insert 5 records into books table

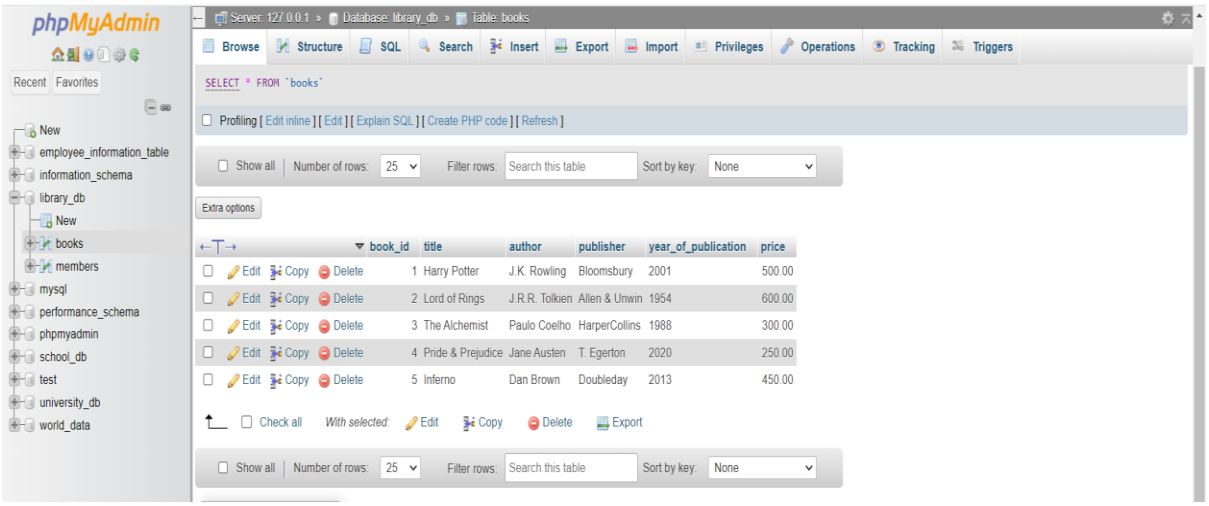
  ```
  INSERT INTO books (book_id, title, author, publisher, year_of_publication, price)
  VALUES
  (1, 'Harry Potter', 'J.K. Rowling', 'Bloomsbury', 2001, 500.00),
  (2, 'Lord of Rings', 'J.R.R. Tolkien', 'Allen & Unwin', 1954, 600.00),
  (3, 'The Alchemist', 'Paulo Coelho', 'HarperCollins', 1988, 300.00),
  ```

(4, 'Pride & Prejudice', 'Jane Austen', 'T. Egerton', 2020, 250.00),

(5, 'Inferno', 'Dan Brown', 'Doubleday', 2013, 450.00);

Step 5: View records

SELECT * FROM books;



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.

Answer:

- **Create members table**

  Step 1: Create members table

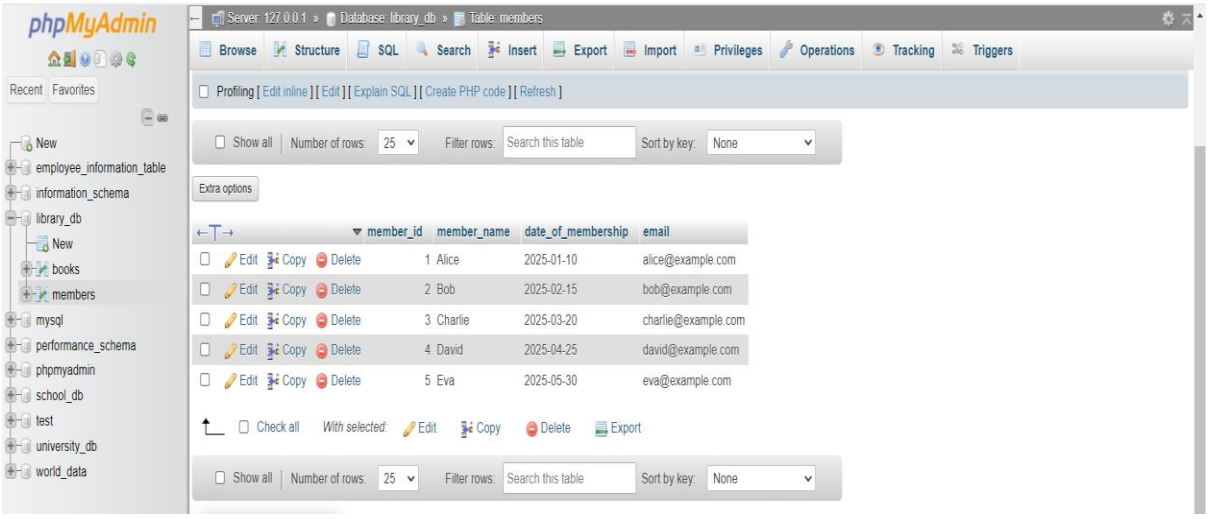
  ```
  CREATE TABLE members (
      member_id INT PRIMARY KEY,
      member_name VARCHAR(50),
      date_of_membership DATE,
      email VARCHAR(50)
  );
  ```

  Step 2: Insert 5 records into members table

  ```
  INSERT INTO members (member_id, member_name, date_of_membership, email)
  VALUES
  (1, 'Alice', '2025-01-10', 'alice@example.com'),
  (2, 'Bob', '2025-02-15', 'bob@example.com'),
  (3, 'Charlie', '2025-03-20', 'charlie@example.com'),
  (4, 'David', '2025-04-25', 'david@example.com'),
  (5, 'Eva', '2025-05-30', 'eva@example.com');
  ```

  Step 3: View records

  ```
  SELECT * FROM members;
  ```



2. SQL Syntax

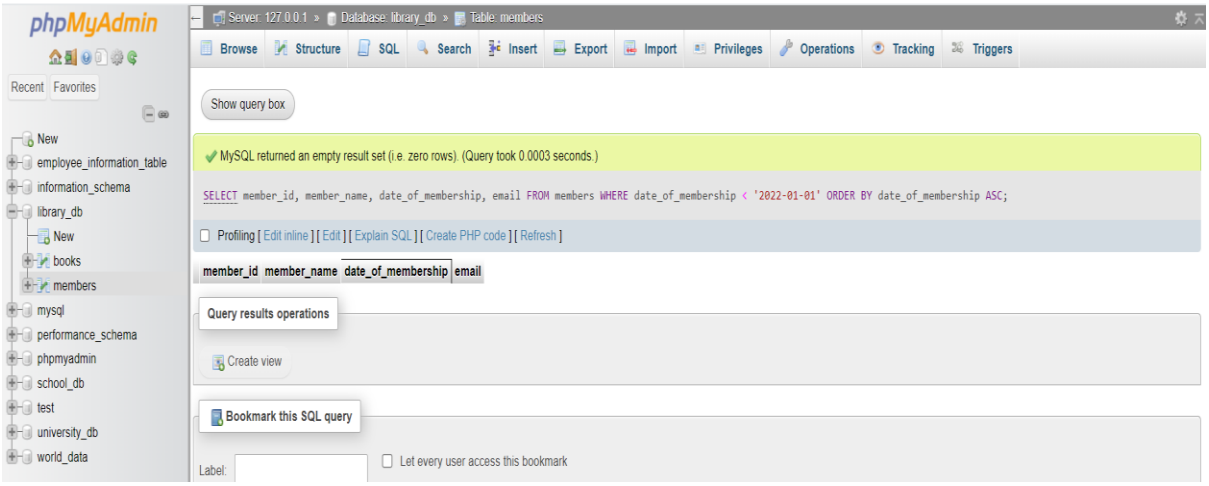
Lab 3: Retrieve all members who joined the library before 2022. Use appropriate SQL syntax with WHERE and ORDER BY.

Answer:

Query: Retrieve all members who joined the library before 2022

SELECT member_id, member_name, date_of_membership, email FROM members

WHERE date_of_membership < '2022-01-01' ORDER BY date_of_membership ASC;



- These data have membership dates only in 2025, so no records – because no one joined before 2022.

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.

Answer:

Query: Display the titles of books published by a specific author (sorted DESC)

Example: Author – J.K. Rowling

```
SELECT title, author, year_of_publication
FROM books
WHERE author = 'J.K. Rowling'
ORDER BY year_of_publication DESC;
```

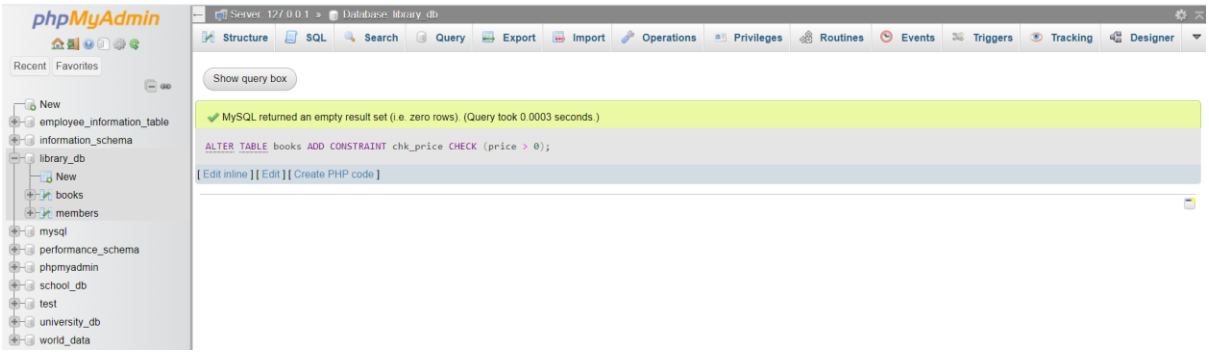
3. SQL Constraints

Lab 3: Add a CHECK constraint to ensure that the price of books in the books table is greater than 0.

Answer:

Query: Add a Check constraint to ensure price > 0

```
ALTER TABLE books ADD CONSTRAINT chk_price CHECK (price > 0);
```



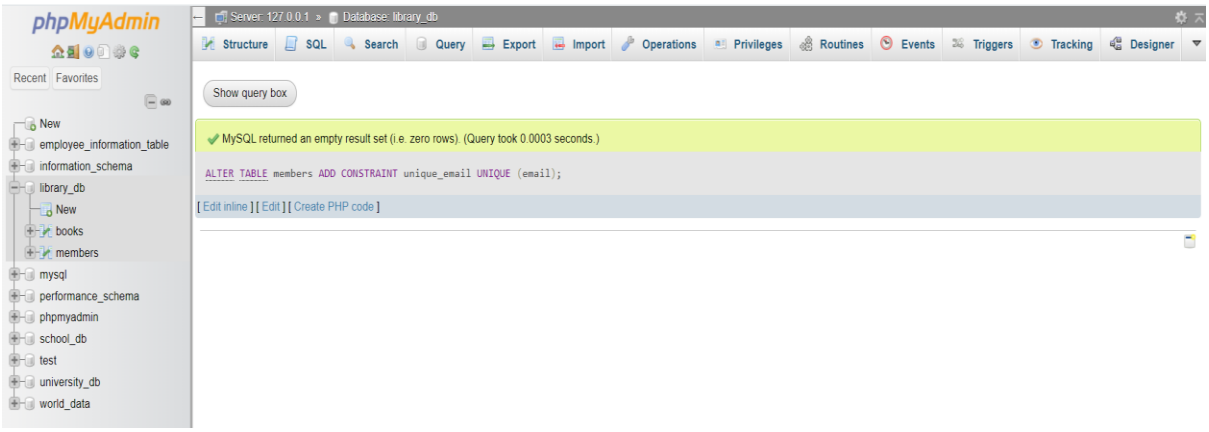
- This ensures price cannot be 0 or negative.

Lab 4: Modify the members table to add a UNIQUE constraint on the email column, ensuring that each member has a unique email address.

Answer:

Query: Add UNIQUE constraint on email column in members table

ALTER TABLE members ADD CONSTRAINT unique_email UNIQUE (email);



- This ensures no two members can have the same email.

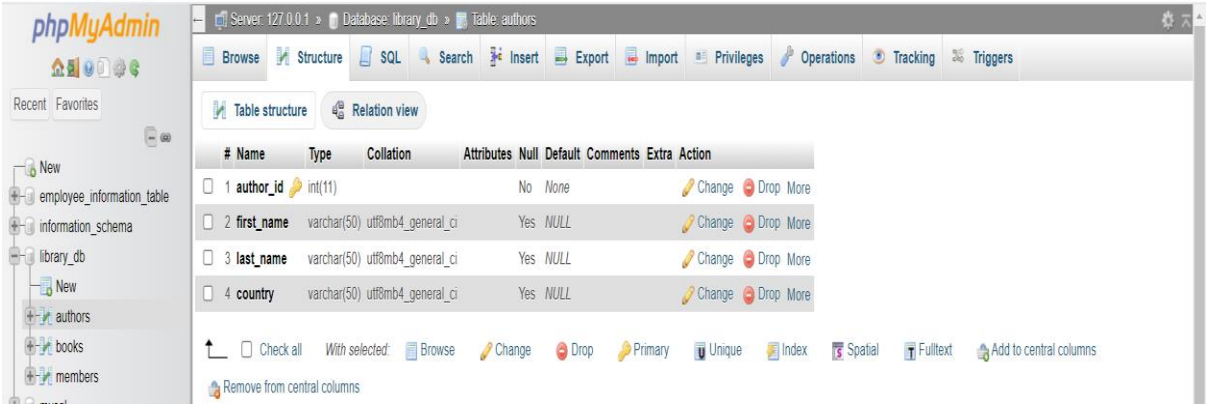
4. Main SQL Commands and Sub-commands (DDL)

Lab 3: Create a table authors with the following columns: author_id, first_name, last_name, and country. Set author_id as the primary key.

Answer:

Query: Create a Table

```
CREATE TABLE authors (
    author_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    country VARCHAR(50)
);
```

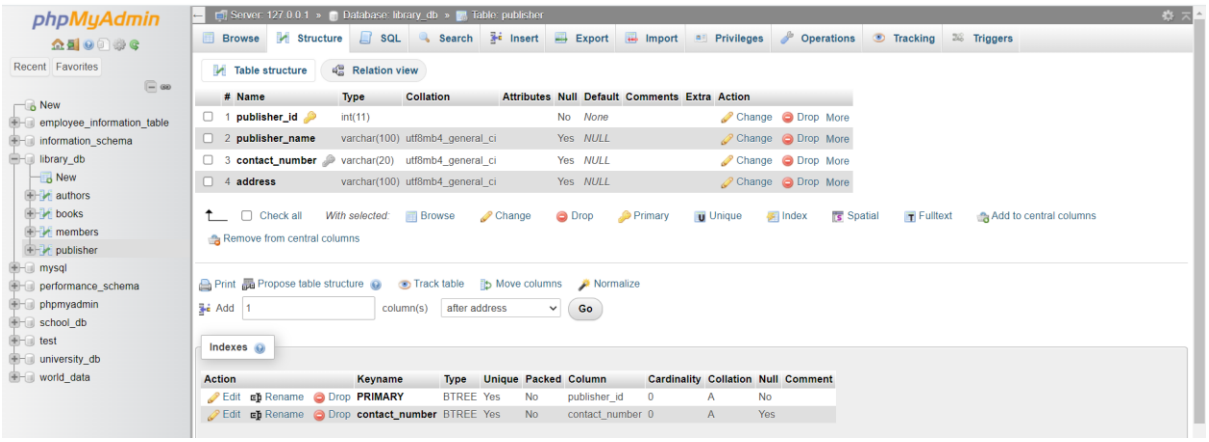


Lab 4: Create a table publishers with columns: publisher_id, publisher_name, contact_number, and address. Set publisher_id as the primary key and contact_number as unique.

Answer:

Query: Create a Table publisher **with UNIQUE contact_number**

```
CREATE TABLE publishers (
    publisher_id INT PRIMARY KEY,
    publisher_name VARCHAR(100),
    contact_number VARCHAR(20) UNIQUE,
    address VARCHAR(100)
);
```



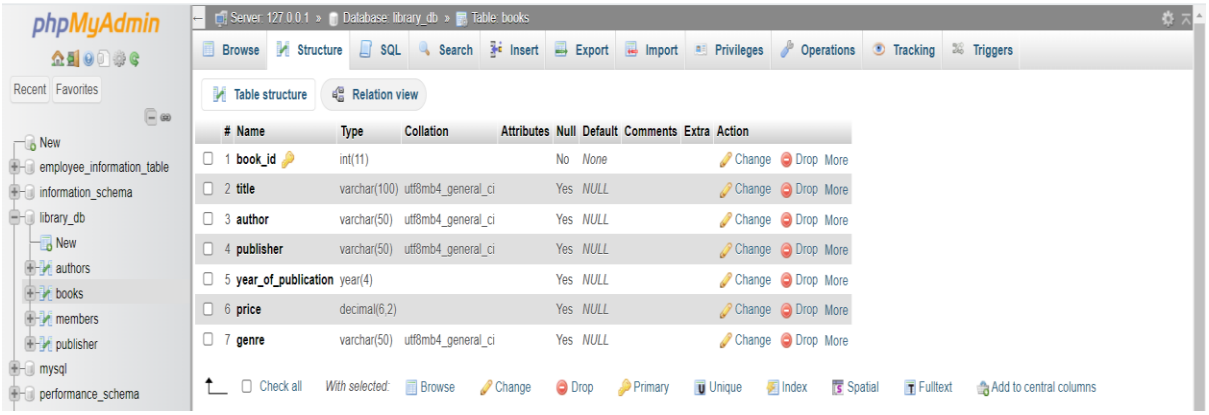
5. ALTER Command

Lab 3: Add a new column genre to the books table. Update the genre for all existing records.

Answer:

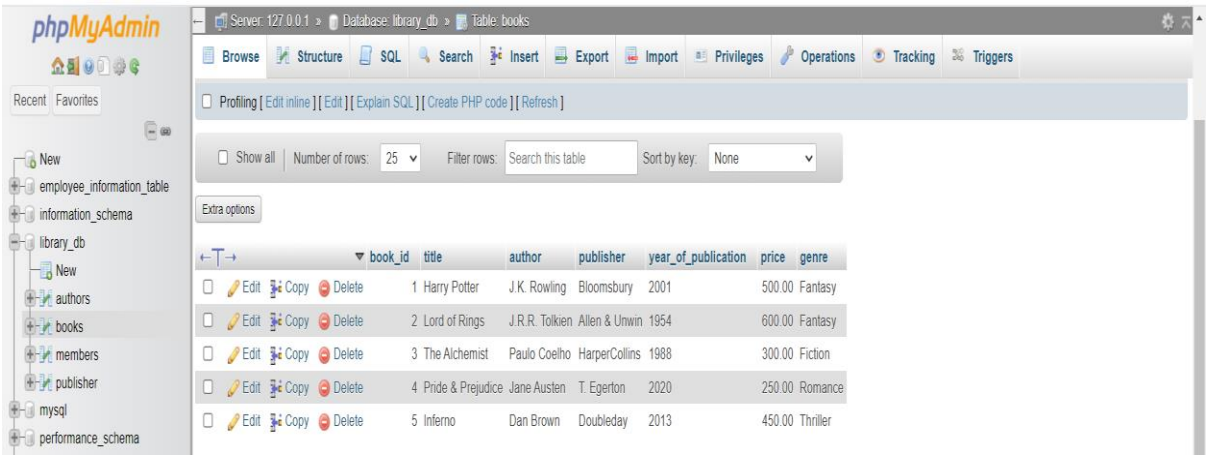
Query 1: Add a new column genre to the books table

```
ALTER TABLE books ADD genre VARCHAR(50);
```



Query 2: Update genre for all existing books

```
UPDATE books SET genre = 'Fantasy' WHERE book_id = 1;

UPDATE books SET genre = 'Fantasy' WHERE book_id = 2;

UPDATE books SET genre = 'Fiction' WHERE book_id = 3;

UPDATE books SET genre = 'Romance' WHERE book_id = 4;

UPDATE books SET genre = 'Thriller' WHERE book_id = 5;
```



Lab 4: Modify the members table to increase the length of the email column to 100 characters.

Answer:

Query: Modify email length in members table to 100 characters

ALTER TABLE members MODIFY email VARCHAR(100);

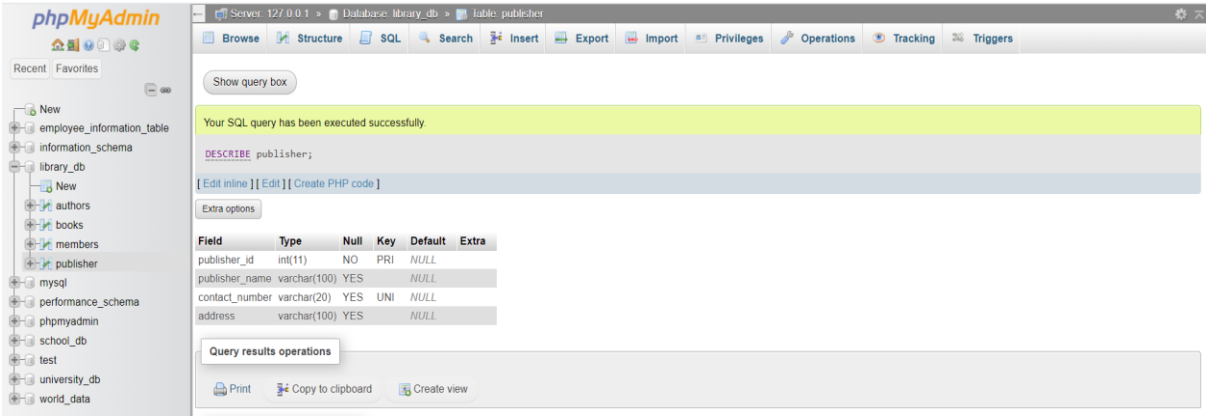
6. DROP Command

Lab 3: Drop the publishers table from the database after verifying its structure.

Answer:

Query 1: Verify table structure

DESCRIBE publisher;



Query 2: Drop the table

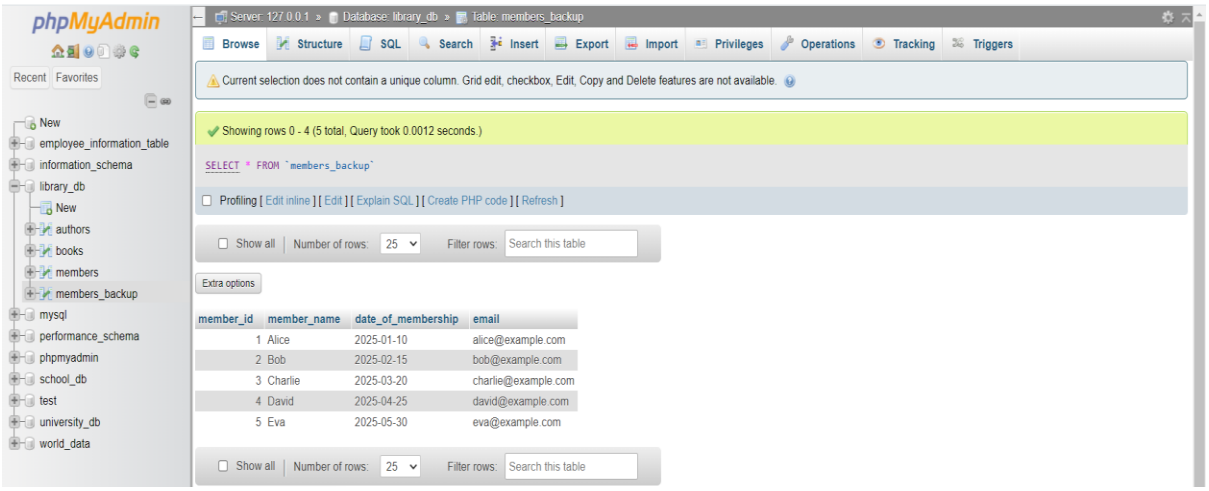
DROP TABLE publisher;

Lab 4: Create a backup of the members table and then drop the original members table.

Answer:

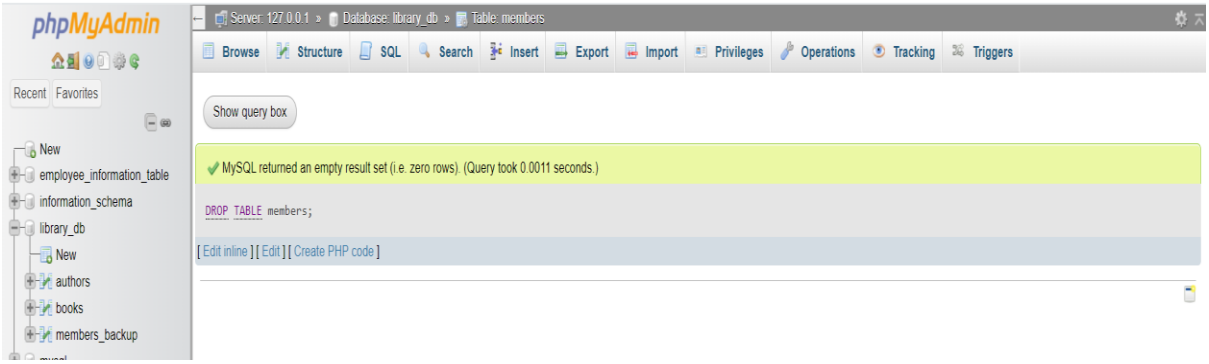
Query 1: Create backup table

CREATE TABLE members_backup AS SELECT * FROM members;



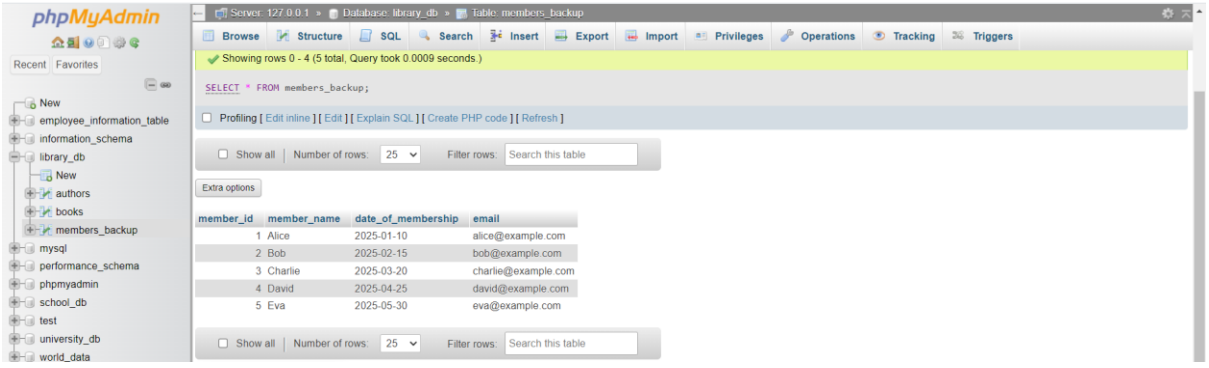
Query 2: Drop the Original Members table

DROP TABLE members;



Query 3: View backup table (optional)

SELECT * FROM members_backup;



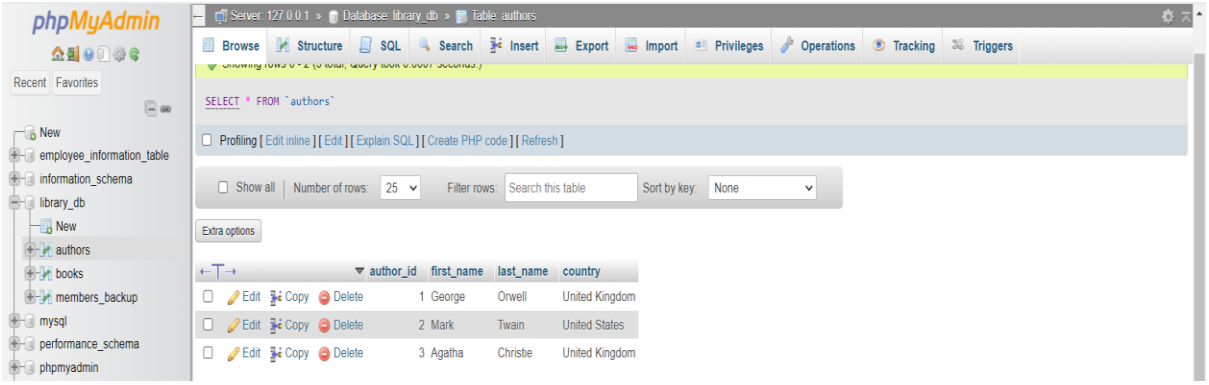
7. Data Manipulation Language (DML)

Lab 4: Insert three new authors into the authors table, then update the last name of one of the authors.

Answer:

Query 1: Insert 3 authors

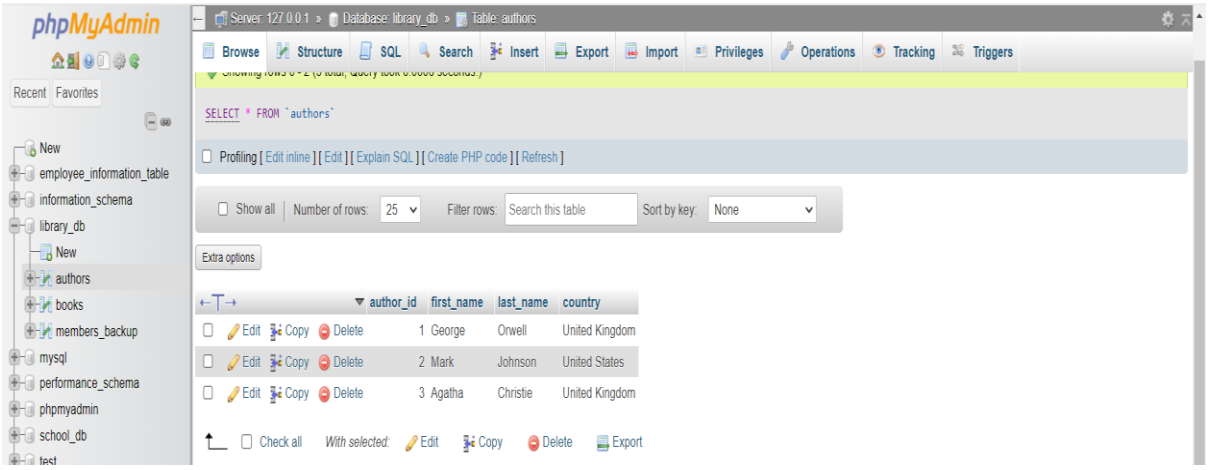
```
INSERT INTO authors (author_id, first_name, last_name, country)
VALUES
(1, 'George', 'Orwell', 'United Kingdom'),
(2, 'Mark', 'Twain', 'United States'),
(3, 'Agatha', 'Christie', 'United Kingdom');
```



Query 2: Update last name of one author

Example: Change Mark Twain -> Mark Johnson

```
UPDATE authors SET last_name = 'Johnson' WHERE author_id = 2;
```



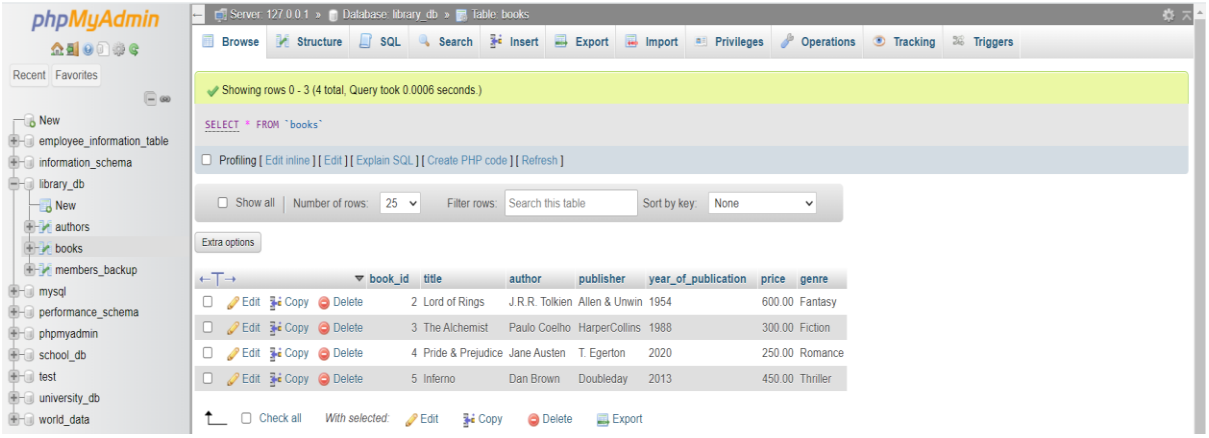
Lab 5: Delete a book from the books table where the price is higher than $100.

Answer:

Query: Delete a book where price > 100

DELETE FROM books WHERE price > 100 LIMIT 1;

- Based on data, all books have price > 100, so deleting any one book is valid.



- Deletes only one book, even though many match.

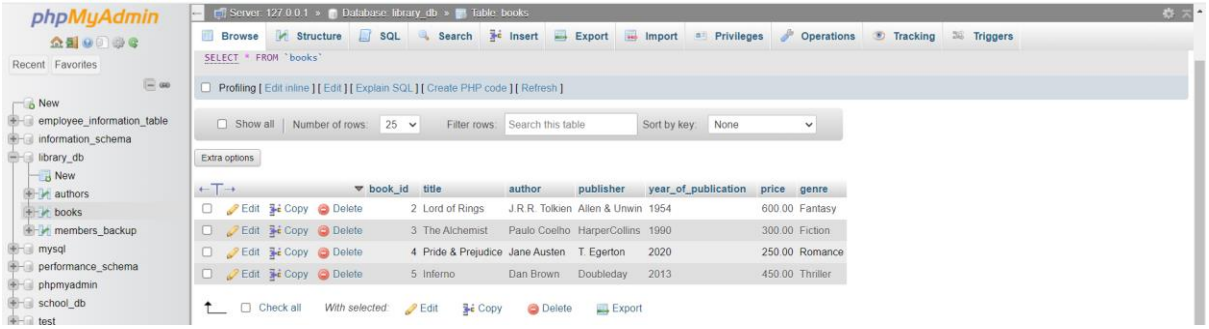
8. UPDATE Command

Lab 3: Update the year_of_publication of a book with a specific book_id.

Answer:

Query: Update the year of publication of book_id = 3 (The Alchemist**)**

UPDATE books SET year_of_publication = 1990 WHERE book_id = 3;

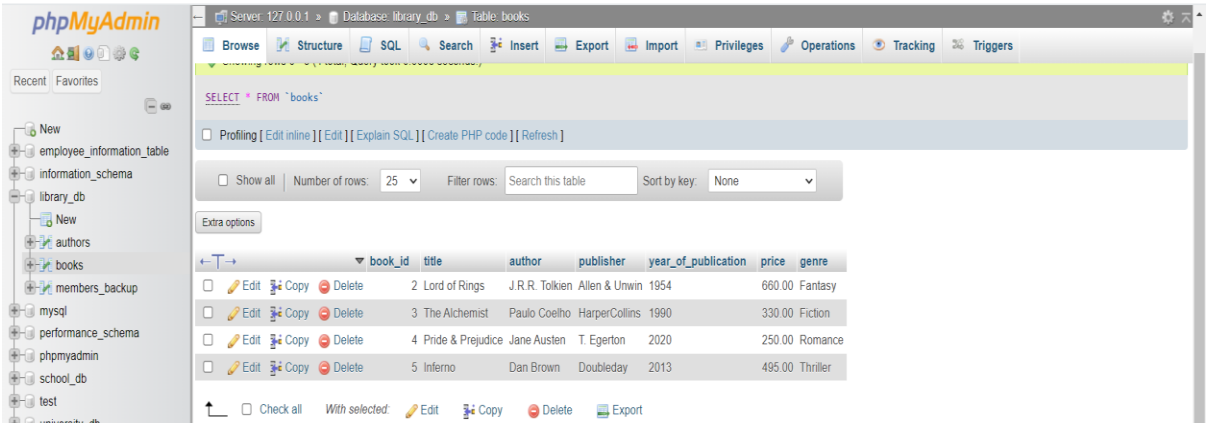


Lab 4: Increase the price of all books published before 2015 by 10%.

Answer:

Query: increase price of all books

UPDATE books SET price = price * 1.10 WHERE year_of_publication < 2015;



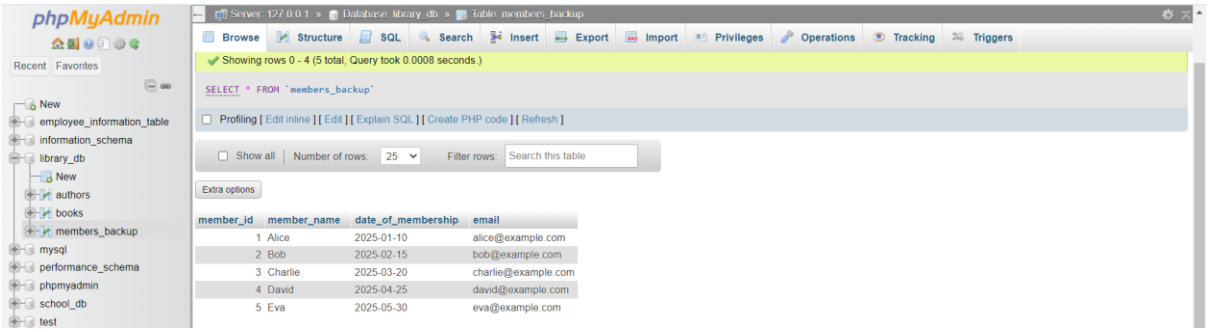
9. DELETE Command

Lab 3: Remove all members who joined before 2020 from the members table.

Answer:

Query: Remove all members who joined before 2020

DELETE FROM members_backup WHERE date_of_membership < '2020-01-01';



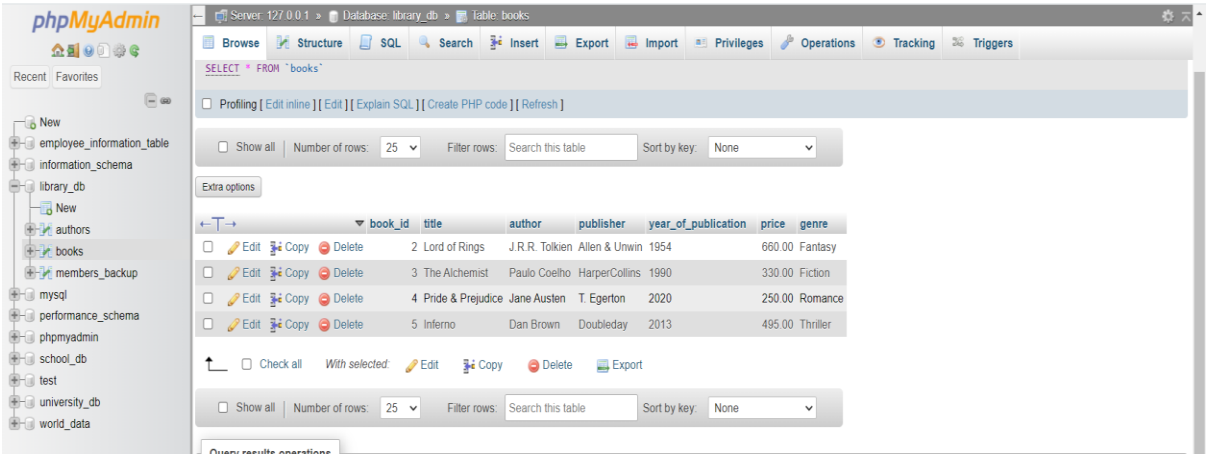
- In members table, all members are joined on 2025.
- All dates are after 2020, so no record will be deleted.

Lab 4: Delete all books that have a NULL value in the author column.

Answer:

Query: Delete all books with NULL author

DELETE FROM books WHERE author IS NULL;



- In books table, all books have authors so no rows are deleted.

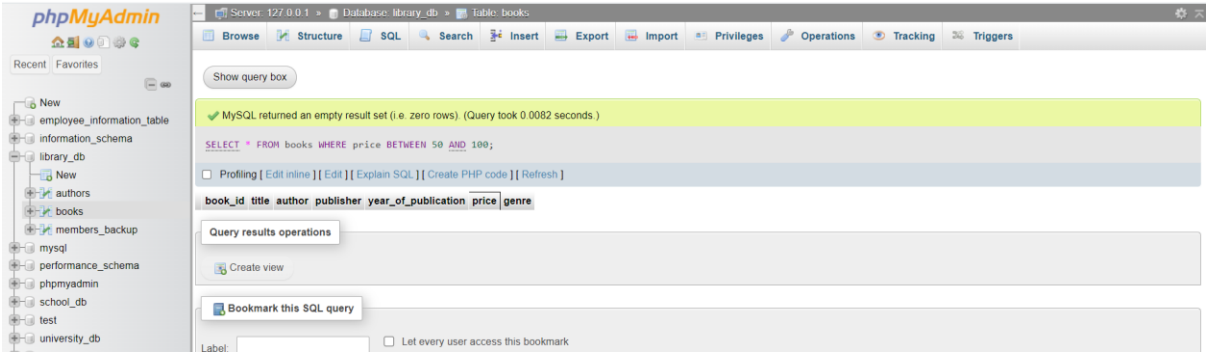
10. Data Query Language (DQL)

Lab 4: Write a query to retrieve all books with price between $50 and $100.

Answer:

Query: retrieve all books with price between $50 and $100

SELECT * FROM books WHERE price BETWEEN 50 AND 100;



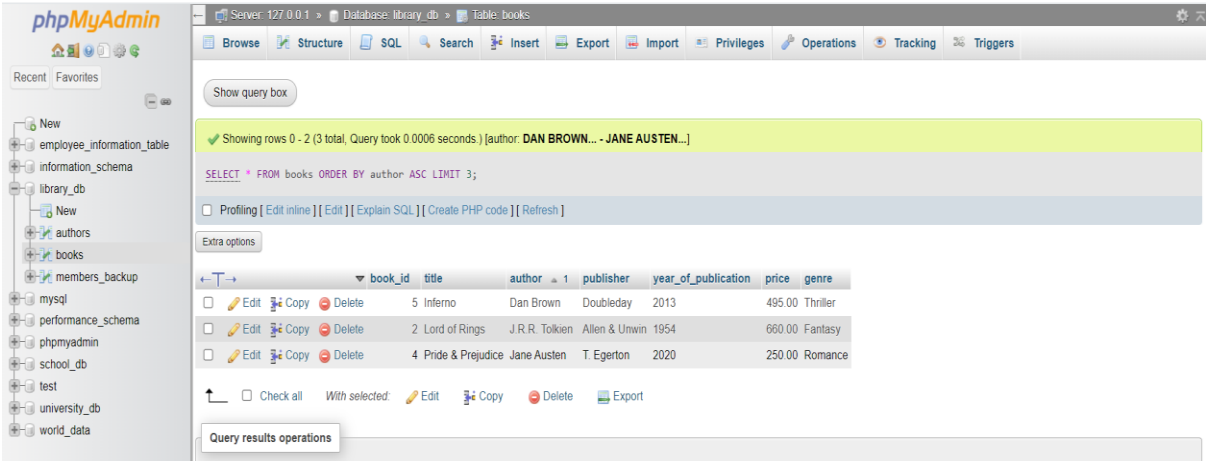
- None are between 50 and 100, so result is empty.

Lab 5: Retrieve the list of books sorted by author in ascending order and limit the results to the top 3 entries.

Answer:

Query: Retrieve books sorted by author (ASC) and show top 3

SELECT * FROM books ORDER BY author ASC LIMIT 3;



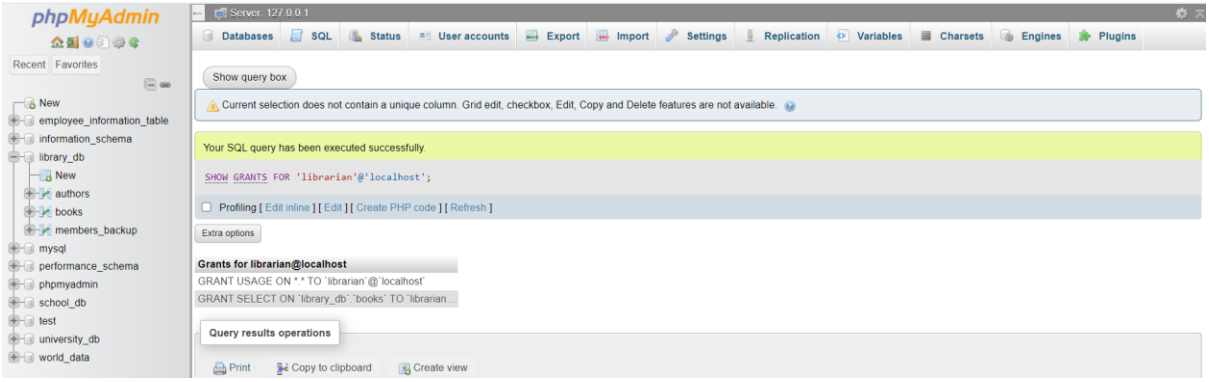
11. Data Control Language (DCL)

Lab 3: Grant SELECT permission to a user named librarian on the books table.

Answer:

Query: Grant SELECT permission to user librarian **on books table**

GRANT SELECT ON library_db.books TO 'librarian'@'localhost';

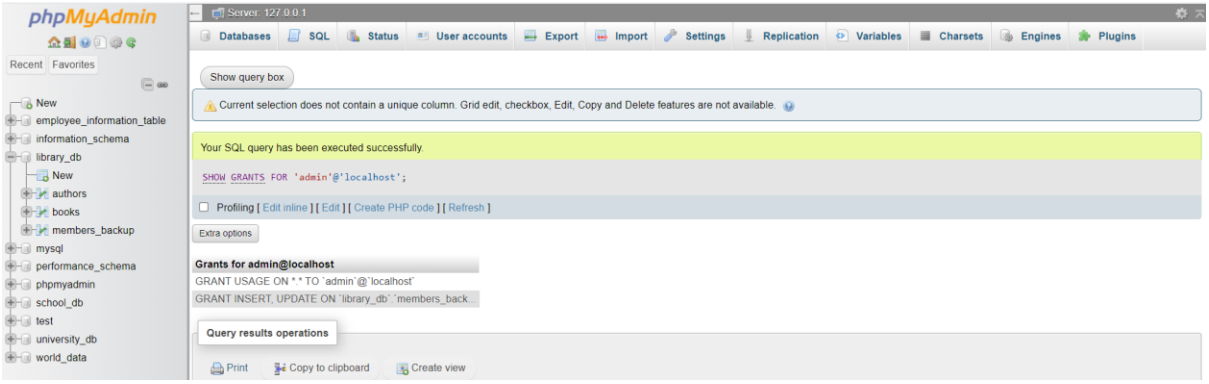


Lab 4: Grant INSERT and UPDATE permissions to the user admin on the members table.

Answer:

Query: Grant INSERT and UPDATE permissions to user admin **on members table**

GRANT INSERT, UPDATE ON library_db.members_backup TO 'admin'@'localhost';



12. REVOKE Command

Lab 3: Revoke the INSERT privilege from the user librarian on the books table.

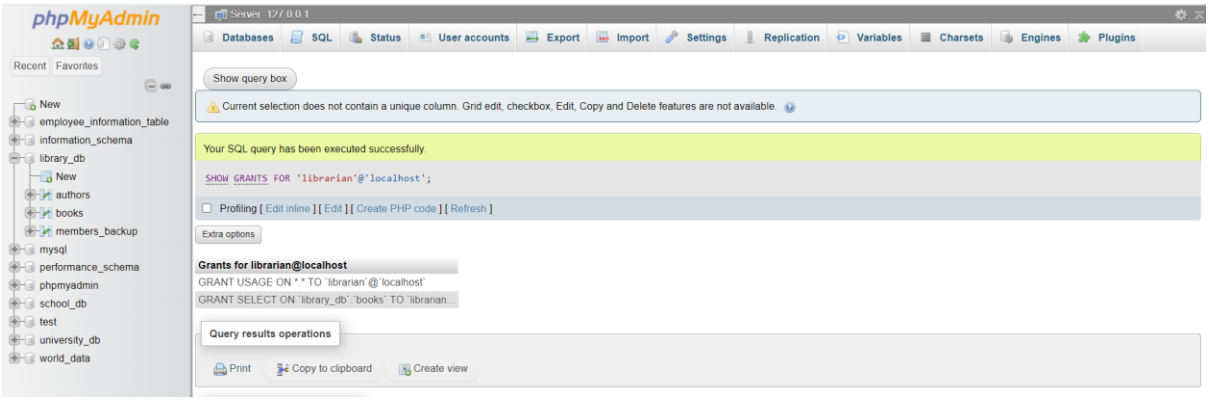
Answer:

Query: 1

REVOKE INSERT ON library_db.books FROM 'librarian'@'localhost';

Query 2: run

SHOW GRANTS FOR 'librarian'@'localhost';



- You will no longer see INSERT privilege for library_db.book.

Lab 4: Revoke all permissions from user admin on the members table.

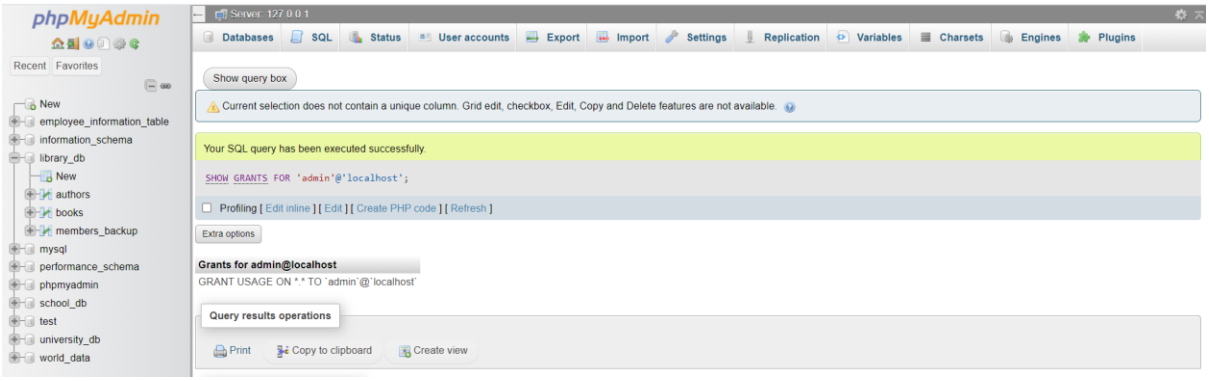
Answer:

Query: 1

REVOKE ALL PRIVILEGES ON library_db.members_backup FROM 'admin'@'localhost';

Query 2: run

SHOW GRANTS FOR 'admin'@'localhost';



- You should see no privileges for the members_backup table.

13. Transaction Control Language (TCL)

Lab 3: Use COMMIT after inserting multiple records into the books table, then make another insertion and perform a ROLLBACK.

Answer:

Query:

Step 1: Start transaction

START TRANSACTION;

Step 2: Insert multiple records

INSERT INTO books (book_id, title, author, publisher, year_of_publication, price)

VALUES

(201, 'Book A', 'Author A', 'Publisher A', 2018, 250),

(202, 'Book B', 'Author B', 'Publisher B', 2020, 300);

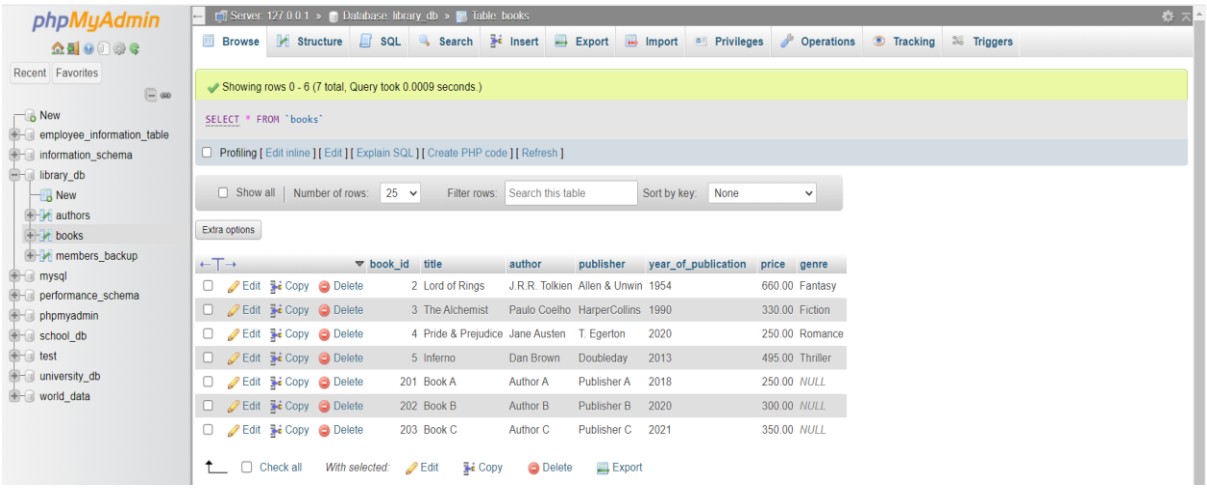
Step 3: Commit the transaction

COMMIT;

Step 4: Insert another record

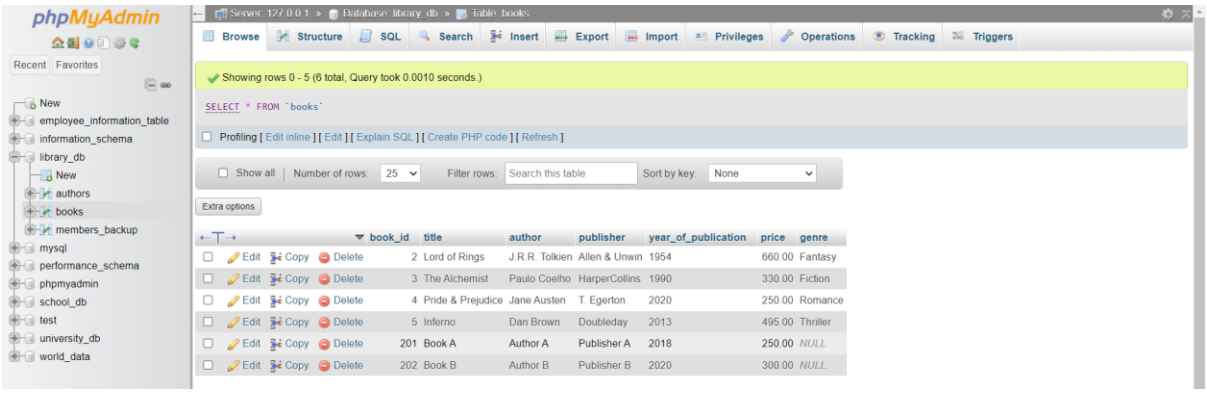
INSERT INTO books (book_id, title, author, publisher, year_of_publication, price)

VALUES (203, 'Book C', 'Author C', 'Publisher C', 2021, 350);



Step 5: Rollback

ROLLBACK;



- Records 201 and 202 -> saved permanently.
- Records 203 -> not saved (because of rollback).

Lab 4: Set a SAVEPOINT before making updates to the members table, perform some updates, and then roll back to the SAVEPOINT.

Answer:

Query:

Step 1: Start transaction

START TRANSACTION;

Step 2: Set a SAVEPOINT

SAVEPOINT sp_before_update;

Step 3: Perform updates

UPDATE members SET email = 'newemail1@example.com' WHERE member_id = 1;

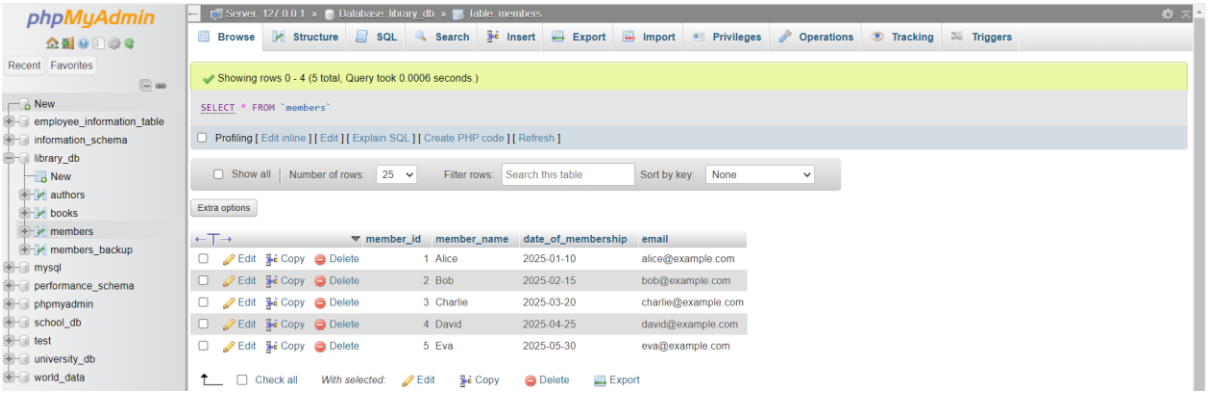
UPDATE members SET member_name = 'Updated Name' WHERE member_id = 2;

Step 4: Roll back to SAVEPOINT

ROLLBACK TO sp_before_update;

Step 5: Commit final state

COMMIT;



- Updates on member 1 & 2 -> Undone
- Database returns to the states at SAVEPOINT sp_before_update

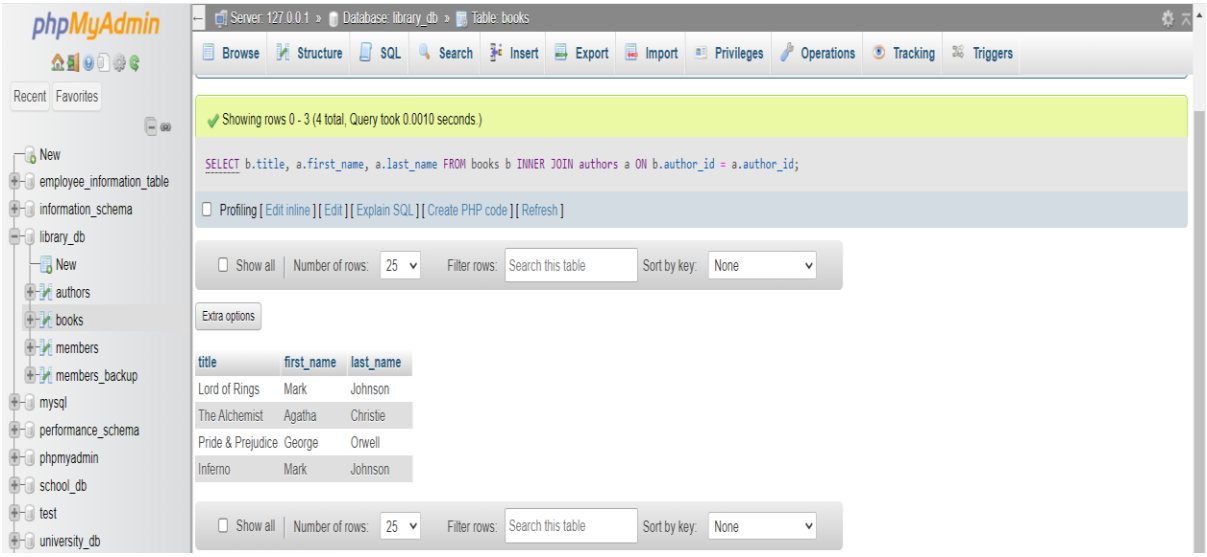
14. SQL Joins

Lab 3: Perform an INNER JOIN between books and authors tables to display the title of books and their respective authors' names.

Answer:

Query:

SELECT b.title, a.first_name, a.last_name FROM books b INNER JOIN authors a ON b.author_id = a.author_id;



Lab 4: Use a FULL OUTER JOIN to retrieve all records from the books and authors tables, including those with no matching entries in the other table.

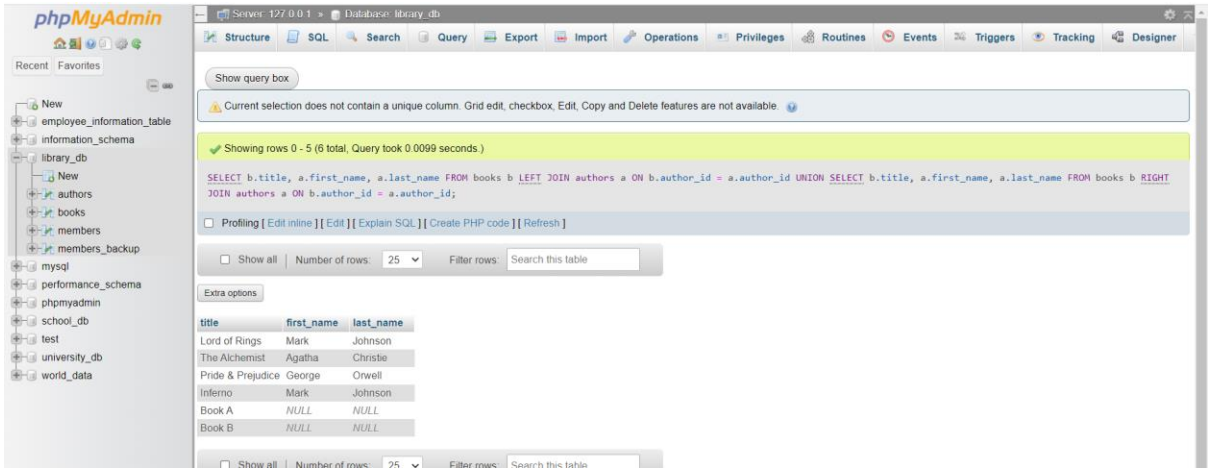
Answer:

Query:

SELECT b.title, a.first_name, a.last_name FROM books b LEFT JOIN authors a ON b.author_id = a.author_id

UNION

SELECT b.title, a.first_name, a.last_name FROM books b RIGHT JOIN authors a ON b.author_id = a.author_id;


phpMyAdmin
Recent
Favorites
New
employee_information_table
information_schema
library_db
New
authors
books
members
members_backup
mysql
performance_schema
phpmyadmin
school_db
test
university_db
world_data
Server: 127.0.0.1 » Database: library_db
Structure
SQL
Search
Query
Export
Import
Operations
Privileges
Routines
Events
Triggers
Tracking
Designer
Show query box
Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.
Showing rows 0 - 5 (6 total, Query took 0.0099 seconds.)
SELECT b.title, a.first_name, a.last_name FROM books b LEFT JOIN authors a ON b.author_id = a.author_id UNION SELECT b.title, a.first_name, a.last_name FROM books b RIGHT JOIN authors a ON b.author_id = a.author_id;
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all
Number of rows: 25
Filter rows: Search this table
Extra options
title
first_name
last_name
Lord of Rings
Mark
Johnson
The Alchemist
Agatha
Christie
Pride & Prejudice
George
Orwell
Inferno
Mark
Johnson
Book A
NULL
NULL
Book B
NULL
NULL

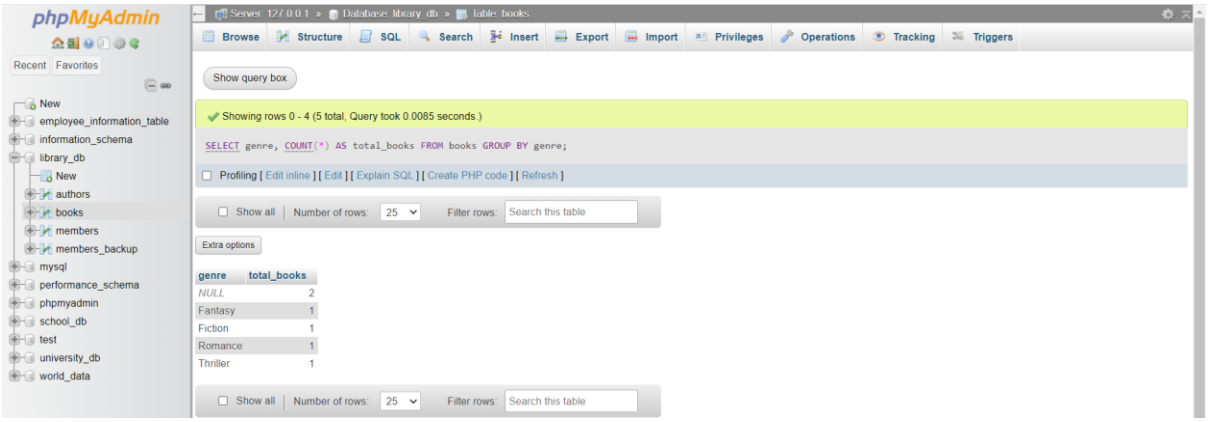
15. SQL Group By

Lab 3: Group books by genre and display the total number of books in each genre.

Answer:

Query:

```
SELECT
    genre,
    COUNT(*) AS total_books
FROM books
GROUP BY genre;
```

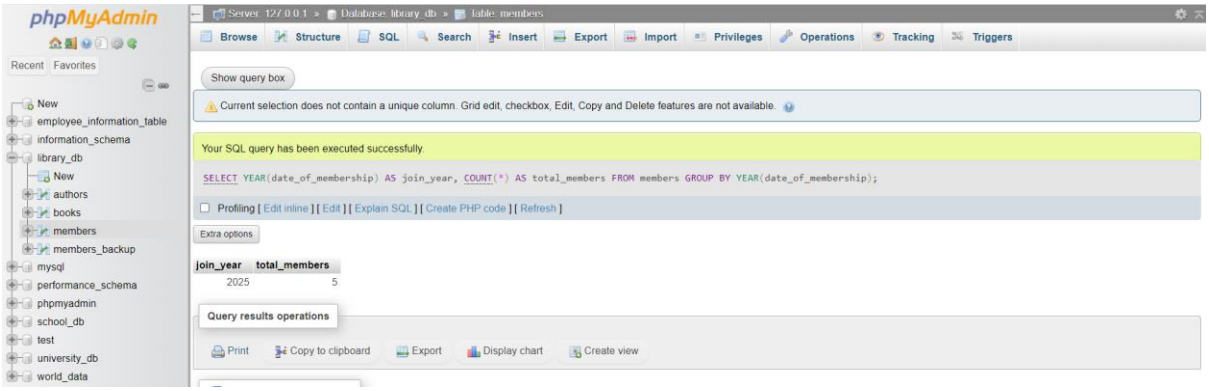


Lab 4: Group members by the year they joined and find the number of members who joined each year.

Answer:

Query:

```
SELECT
    YEAR(date_of_membership) AS join_year,
    COUNT(*) AS total_members
FROM members
GROUP BY YEAR(date_of_membership);
```



16. SQL Stored Procedure

Lab 3: Write a stored procedure to retrieve all books by a particular author.

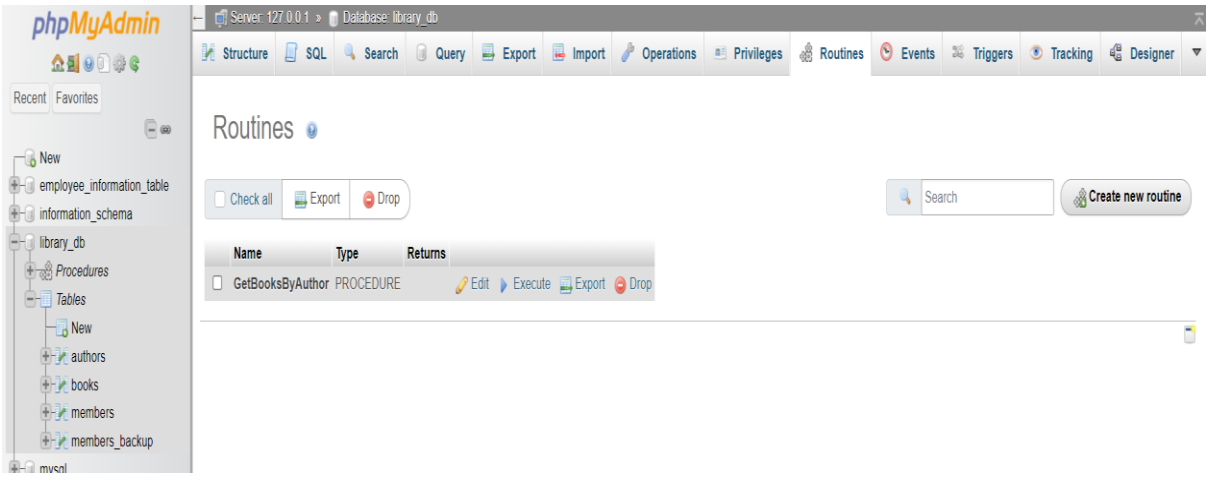
Answer:

Query 1: Create the stored procedure

```
DELIMITER $$

CREATE PROCEDURE GetBooksByAuthor(IN authorName VARCHAR(100))
BEGIN
    SELECT book_id, title, author, publisher, year_of_publication, price FROM books
    WHERE author = authorName;
END $$

DELIMITER ;
```

Query 2: call (run) the stored procedure

```
CALL GetBooksByAuthor('J.K. Rowling');
```



Lab 4: Write a stored procedure that takes book_id as an argument and returns the price of the book.

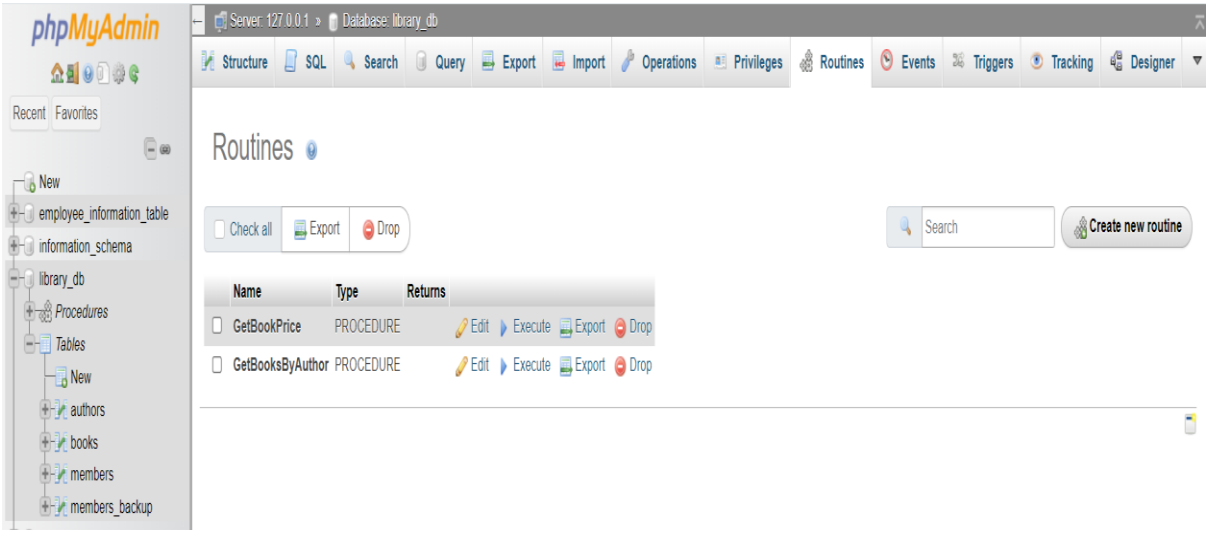
Answer:

Query 1: stored procedure – getbookprice

```
DELIMITER $$

CREATE PROCEDURE GetBookPrice(IN p_book_id INT)
BEGIN
    SELECT price
    FROM books
    WHERE book_id = p_book_id;
END $$

DELIMITER ;
```

Query 2: call (run) the stored procedure

```
CALL GetBookPrice(1);
```



17. SQL View

Lab 3: Create a view to show only the title, author, and price of books from the books table.

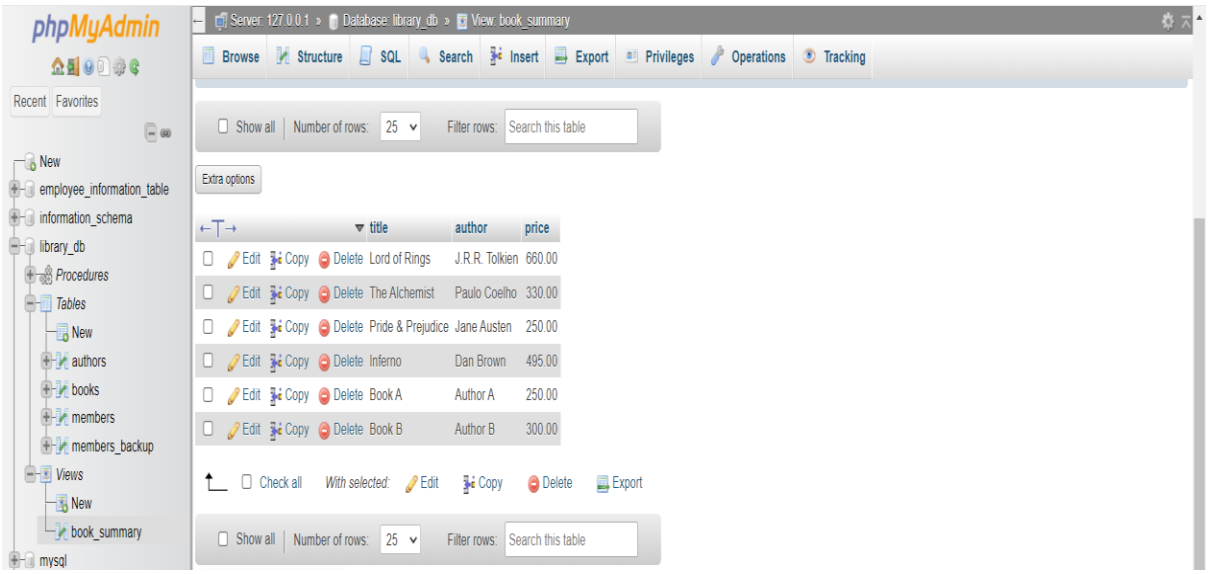
Answer:

Query 1: Create the View

```
CREATE VIEW book_summary AS SELECT title, author, price FROM books;
```

Query 2: To see the view

```
SELECT * FROM book_summary;
```



Lab 4: Create a view to display members who joined before 2020.

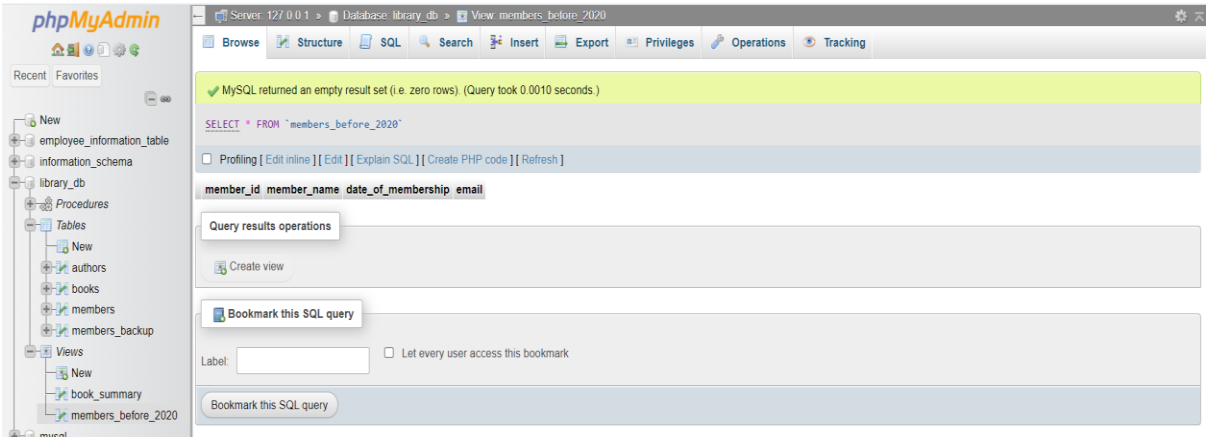
Answer:

Query 1: Create the View

```
CREATE VIEW members_before_2020 AS
SELECT member_id, member_name, date_of_membership, email FROM members
WHERE YEAR(date_of_membership) < 2020;
```

Query 2: To see the view

```
SELECT * FROM members_before_2020;
```



- No member joined before 2020 so rows are empty. All members joined on 2025.

18. SQL Trigger

Lab 3: Create a trigger to automatically update the last_modified timestamp of the books table whenever a record is updated.

Answer:

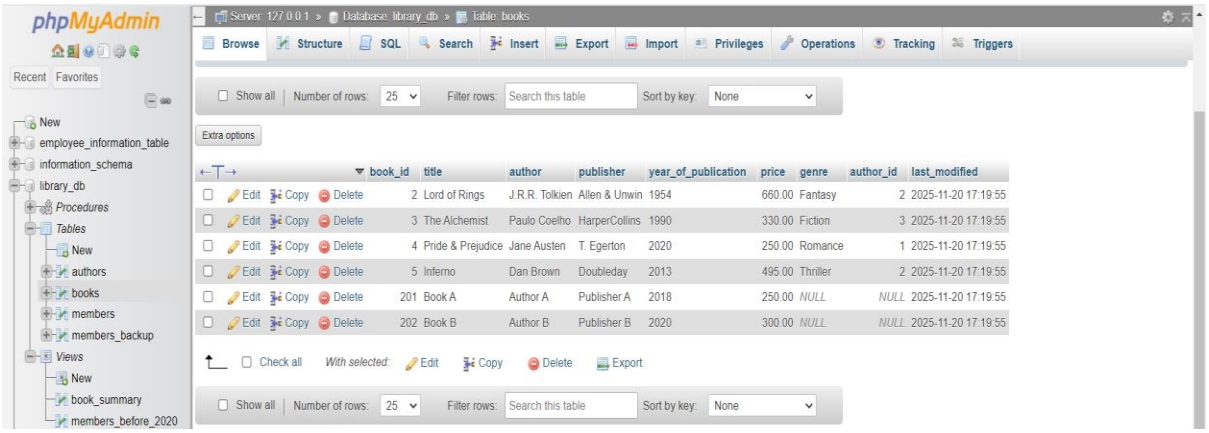
Query:

Step 1: Make sure books table has last_modified column

ALTER TABLE books

ADD COLUMN last_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP;

- This ensures last_modified is updated automatically on UPDATE, but we can also do it via trigger if needed.



Step 2: Trigger to update last_modified on UPDATE

DELIMITER $$

CREATE TRIGGER update_books_timestamp

BEFORE UPDATE ON books

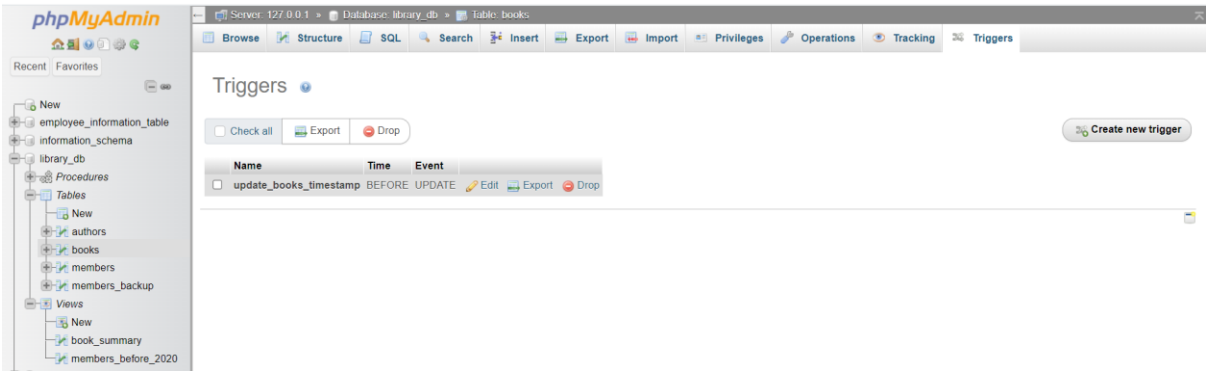
FOR EACH ROW

BEGIN

  SET NEW.last_modified = NOW();

END$$

DELIMITER ;



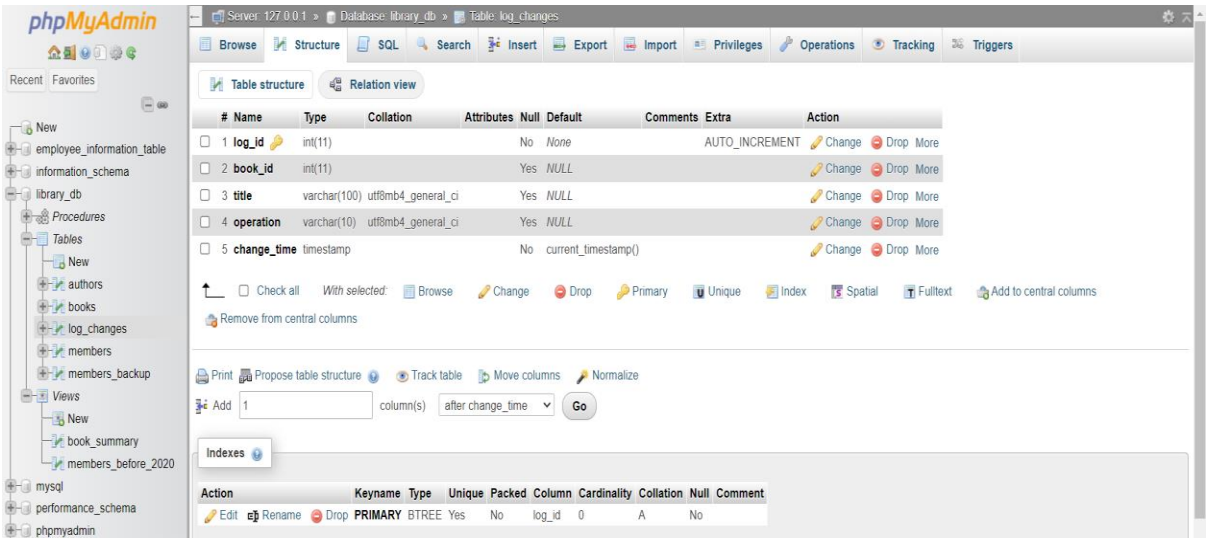
Lab 4: Create a trigger that inserts a log entry into a log_changes table whenever a DELETE operation is performed on the books table.

Answer:

Query:

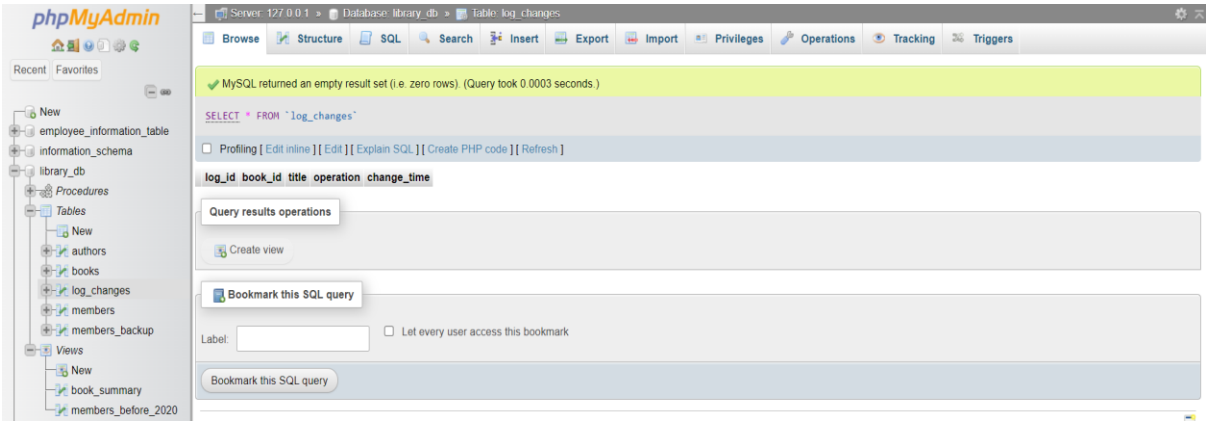
Step 1: Create log_changes table

```
CREATE TABLE log_changes (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    book_id INT,
    title VARCHAR(100),
    operation VARCHAR(10),
    change_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```



Step 2: Trigger to insert a log on DELETE

```
DELIMITER $$

CREATE TRIGGER log_books_deletion
AFTER DELETE ON books
FOR EACH ROW
BEGIN
    INSERT INTO log_changes(book_id, title, operation)
    VALUES (OLD.book_id, OLD.title, 'DELETE');
END$$

DELIMITER ;
```



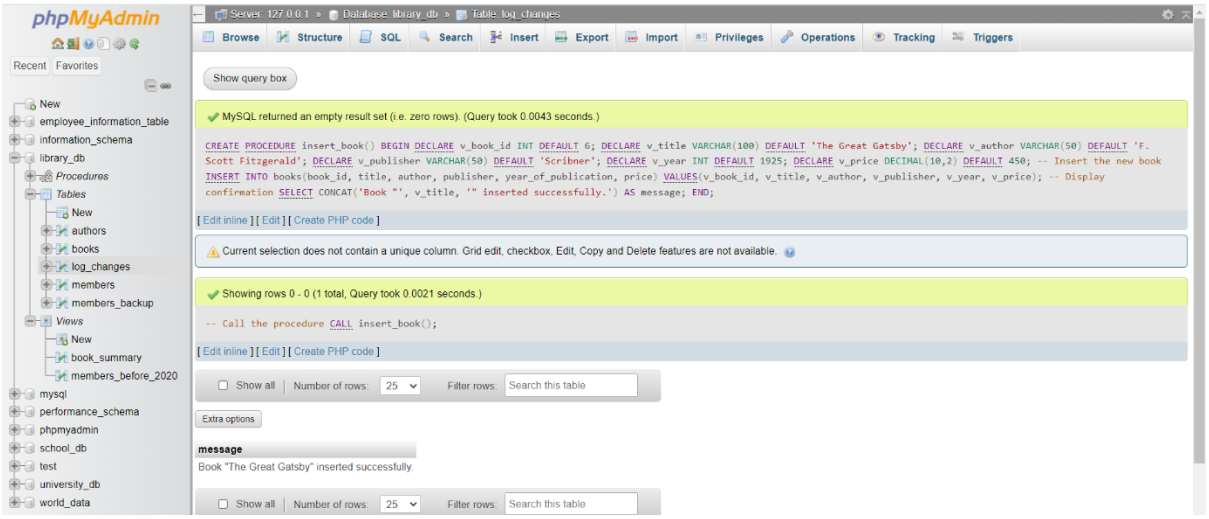
19. Introduction to PL/SQL

Lab 3: Write a PL/SQL block to insert a new book into the books table and display a confirmation message.

Answer:

Query: Insert a new book and display confirmation

```
DELIMITER $$

CREATE PROCEDURE insert_book()
BEGIN
    DECLARE v_book_id INT DEFAULT 6;
    DECLARE v_title VARCHAR(100) DEFAULT 'The Great Gatsby';
    DECLARE v_author VARCHAR(50) DEFAULT 'F. Scott Fitzgerald';
    DECLARE v_publisher VARCHAR(50) DEFAULT 'Scribner';
    DECLARE v_year INT DEFAULT 1925;
    DECLARE v_price DECIMAL(10,2) DEFAULT 450;

    -- Insert the new book
    INSERT INTO books(book_id, title, author, publisher, year_of_publication, price)
    VALUES(v_book_id, v_title, v_author, v_publisher, v_year, v_price);

    -- Display confirmation
    SELECT CONCAT('Book "', v_title, '" inserted successfully.') AS message;
END$$

DELIMITER ;

-- Call the procedure
CALL insert_book();
```

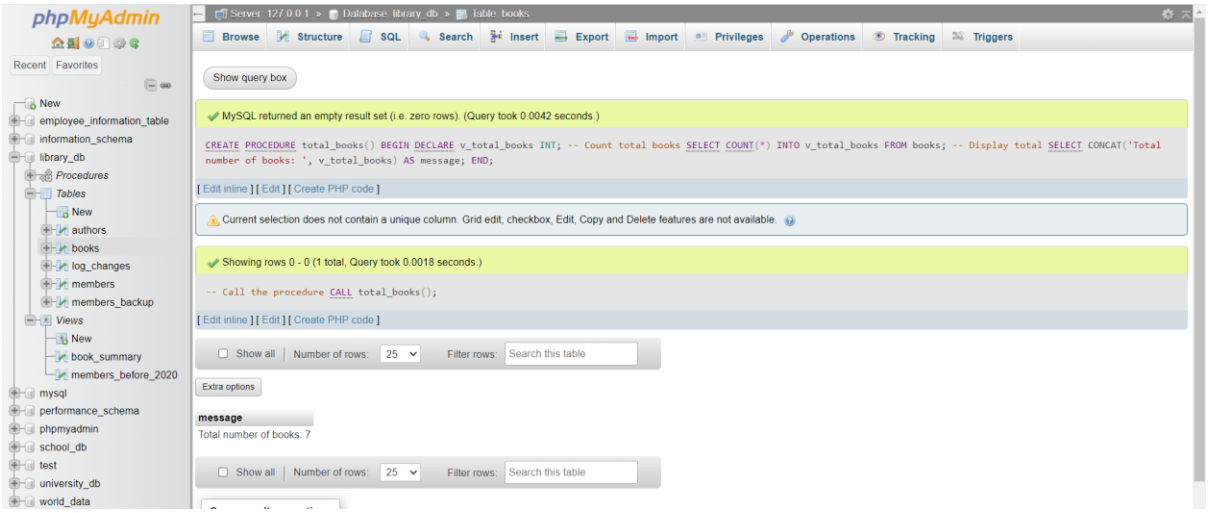


Lab 4: Write a PL/SQL block to display the total number of books in the books table.

Answer:

Query: Display total number of books

```
DELIMITER $$

CREATE PROCEDURE total_books()
BEGIN
    DECLARE v_total_books INT;

    -- Count total books
    SELECT COUNT(*) INTO v_total_books FROM books;

    -- Display total
    SELECT CONCAT('Total number of books: ', v_total_books) AS message;
END$$

DELIMITER ;

-- Call the procedure
CALL total_books();
```


phpMyAdmin
Recent
Favorites
New
employee_information_table
information_schema
library_db
Procedures
Tables
New
authors
books
log_changes
members
members_backup
Views
New
book_summary
members_before_2020
mysql
performance_schema
phpmyadmin
school_db
test
university_db
world_data
Server: 127.0.0.1 » Database: library_db » Table: books
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Tracking
Triggers
Show query box
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0042 seconds.)
CREATE PROCEDURE total_books() BEGIN DECLARE v_total_books INT; -- Count total books SELECT COUNT(*) INTO v_total_books FROM books; -- Display total SELECT CONCAT('Total number of books: ', v_total_books) AS message; END;
[Edit inline] [Edit] [Create PHP code]
Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.
Showing rows 0 - 0 (1 total, Query took 0.0018 seconds.)
-- Call the procedure CALL total_books();
[Edit inline] [Edit] [Create PHP code]
Show all
Number of rows: 25
Filter rows: Search this table
Extra options
message
Total number of books: 7
Show all
Number of rows: 25
Filter rows: Search this table

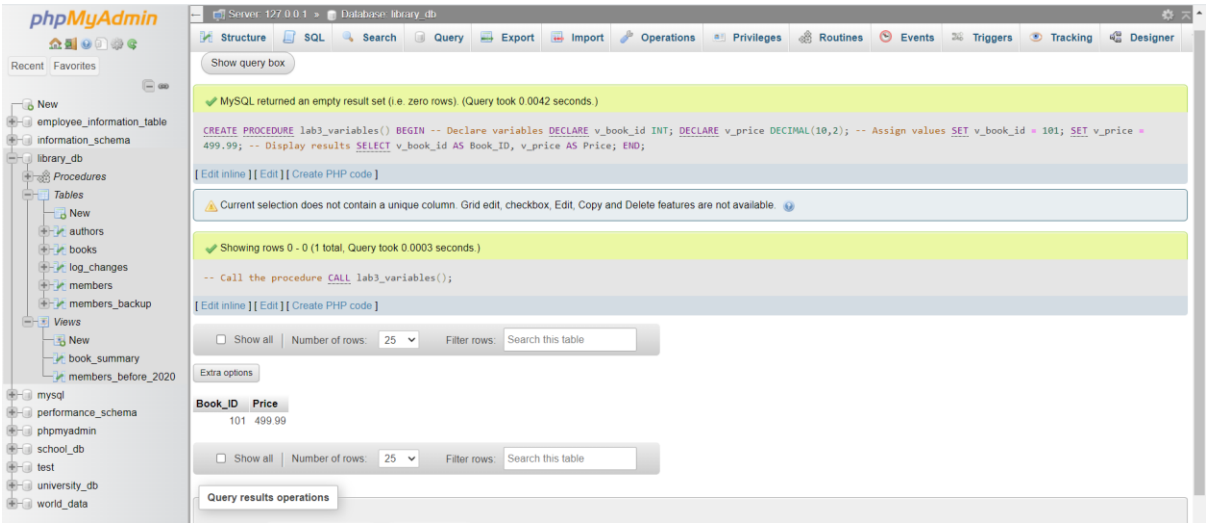
20. PL/SQL Syntax

Lab 3: Write a PL/SQL block to declare variables for book_id and price, assign values, and display the results.

Answer:

Query: Declare variables for book_id and price, assign values, display results

```
DELIMITER $$

CREATE PROCEDURE lab3_variables()
BEGIN
   -- Declare variables
   DECLARE v_book_id INT;
   DECLARE v_price DECIMAL(10,2);

   -- Assign values
   SET v_book_id = 101;
   SET v_price = 499.99;

   -- Display results
   SELECT v_book_id AS Book_ID, v_price AS Price;
END$$

DELIMITER ;

-- Call the procedure
CALL lab3_variables();
```



Lab 4: Write a PL/SQL block using constants and perform arithmetic operations on book prices.

Answer:

Query: Using constants and arithmetic operations on book prices

```
DELIMITER $$

CREATE PROCEDURE lab4_constants()
BEGIN
    -- Declare constants using user-defined variables (MySQL doesn't have true constants in procedures)
    DECLARE price1 DECIMAL(10,2) DEFAULT 300;
    DECLARE price2 DECIMAL(10,2) DEFAULT 450;

    -- Perform arithmetic operations
    DECLARE total_price DECIMAL(10,2);
    DECLARE price_difference DECIMAL(10,2);

    SET total_price = price1 + price2;
    SET price_difference = price2 - price1;
```

-- Display results

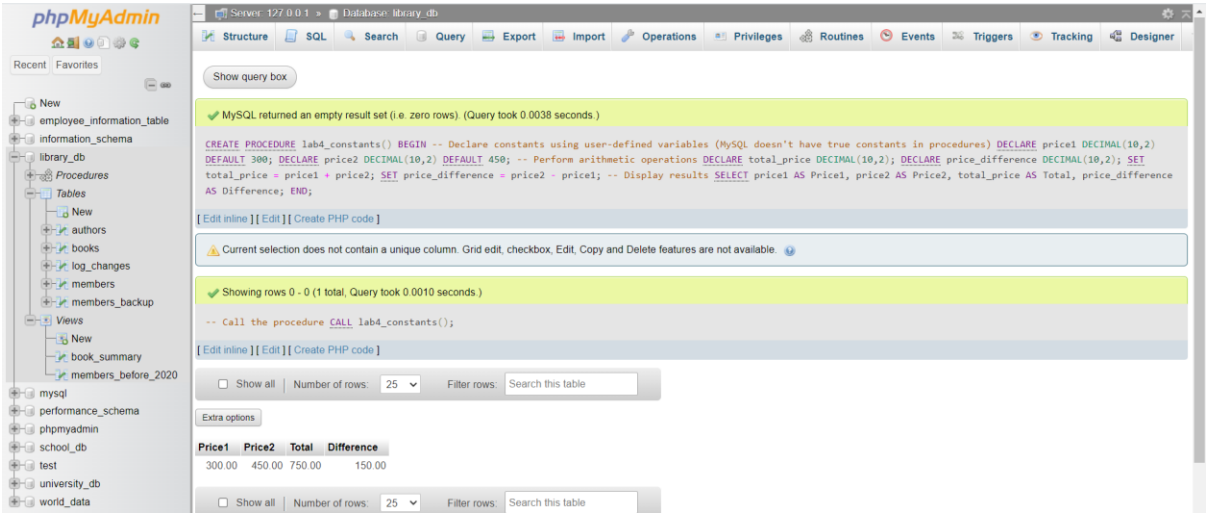
SELECT price1 AS Price1, price2 AS Price2, total_price AS Total, price_difference AS Difference;

END$$

DELIMITER ;

-- Call the procedure

CALL lab4_constants();



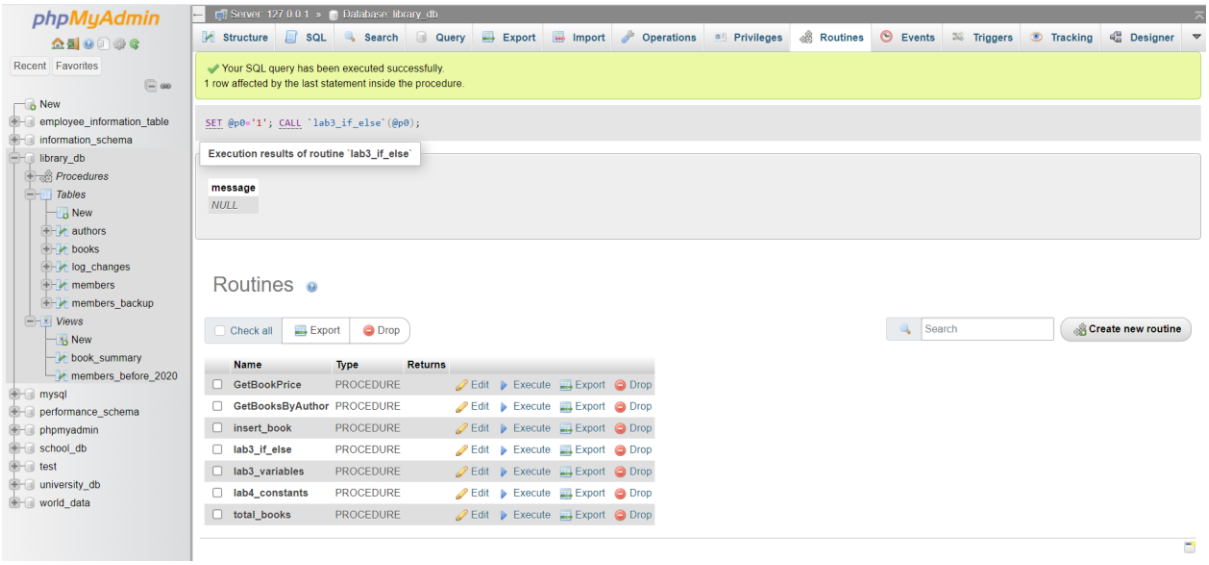
21. PL/SQL Control Structures

Lab 3: Write a PL/SQL block using IF-THEN-ELSE to check if a book's price is above $100 and print a message accordingly.

Answer:

Query: IF-THEN-ELSE to check book price

```
DELIMITER $$

CREATE PROCEDURE lab3_if_else(IN p_book_id INT)
BEGIN
   DECLARE v_price DECIMAL(10,2);

   -- Get the price of the book
   SELECT price INTO v_price FROM books WHERE book_id = p_book_id;

   -- IF-THEN-ELSE logic
   IF v_price > 100 THEN
      SELECT CONCAT('Book ID ', p_book_id, ' has a price above $100: $', v_price) AS message;
   ELSE
      SELECT CONCAT('Book ID ', p_book_id, ' has a price below or equal $100: $', v_price) AS message;
   END IF;
END$$

DELIMITER ;

-- Example call
CALL lab3_if_else(1);
```



Lab 4: Use a FOR LOOP in PL/SQL to display the details of all books one by one.

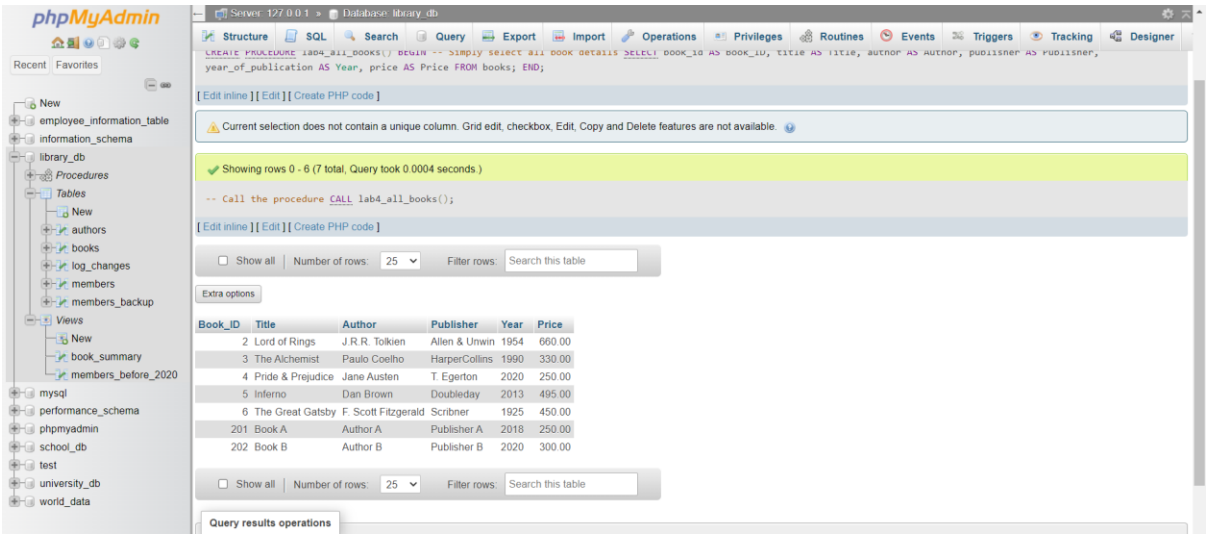
Answer:

Query: FOR LOOP to display all book details

```
DELIMITER $$

CREATE PROCEDURE lab4_all_books()
BEGIN
    -- Simply select all book details
    SELECT book_id AS Book_ID,
           title AS Title,
           author AS Author,
           publisher AS Publisher,
           year_of_publication AS Year,
           price AS Price
    FROM books;
END$$

DELIMITER ;
```

-- Call the procedure

CALL lab4_all_books();



22. SQL Cursors

Lab 3: Write a PL/SQL block using an explicit cursor to fetch and display all records from the members table.

Answer:

Query: Fetch and display all members using a cursor

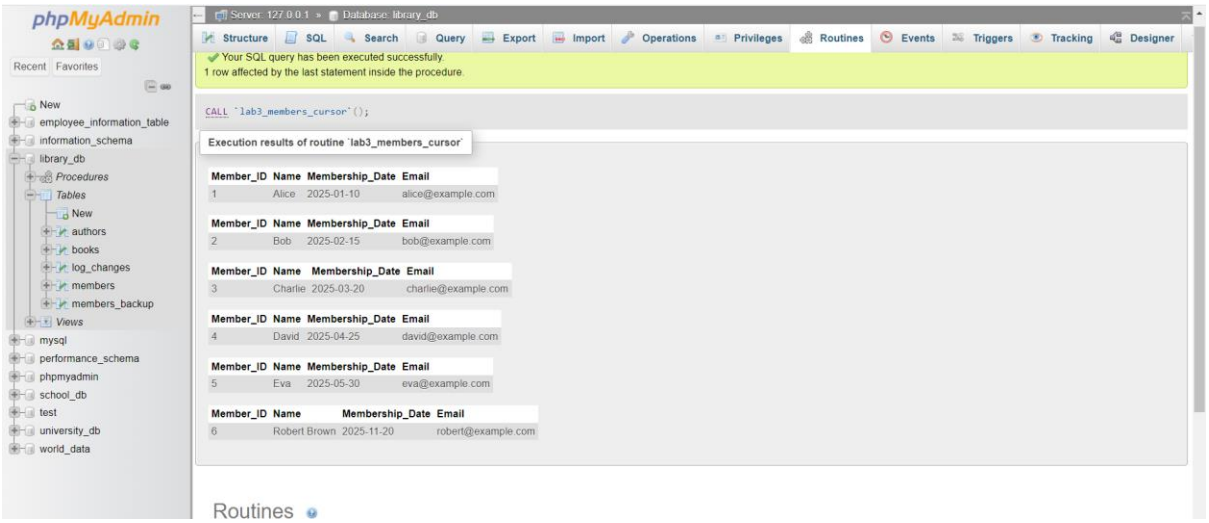
Step 1: Stored Procedure

```
DELIMITER $$

CREATE PROCEDURE lab3_members_cursor()
BEGIN
   -- Variables to hold member details
   DECLARE v_member_id INT;
   DECLARE v_member_name VARCHAR(50);
   DECLARE v_date_of_membership DATE;
   DECLARE v_email VARCHAR(100);
   DECLARE done INT DEFAULT 0;

   -- Cursor for all members
   DECLARE cur_members CURSOR FOR
      SELECT member_id, member_name, date_of_membership, email FROM members;

   -- Handler to exit loop
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

   OPEN cur_members;

   read_loop: LOOP
      FETCH cur_members INTO v_member_id, v_member_name, v_date_of_membership, v_email;
```

```
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Display member details
        SELECT v_member_id AS Member_ID,
               v_member_name AS Name,
               v_date_of_membership AS Membership_Date,
               v_email AS Email;
    END LOOP;

    CLOSE cur_members;
END$$

DELIMITER ;
```

Step 2: call the procedure

```
CALL lab3_members_cursor();
```



Lab 4: Create a cursor to retrieve books by a particular author and display their titles.

Answer:

Query: Retrieve books by a particular author using a cursor

Step 1: Stored Procedure

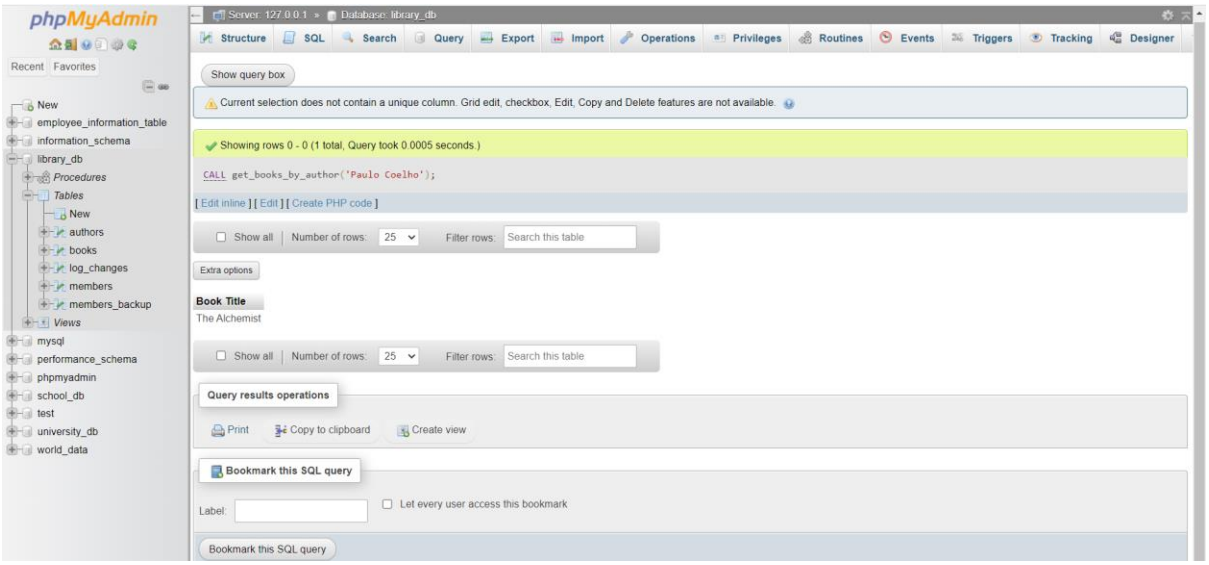
```
DELIMITER $$

CREATE PROCEDURE get_books_by_author(IN auth_name VARCHAR(100))
BEGIN
   DECLARE done INT DEFAULT 0;
   DECLARE b_title VARCHAR(100);

   -- Cursor to select books by author
   DECLARE book_cursor CURSOR FOR
      SELECT title FROM books WHERE author = auth_name;

   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

   OPEN book_cursor;

   read_loop: LOOP
      FETCH book_cursor INTO b_title;
      IF done THEN
         LEAVE read_loop;
      END IF;
      SELECT b_title AS 'Book Title';
   END LOOP;

   CLOSE book_cursor;
```

END$$

DELIMITER ;

Step 2: Call the procedure

CALL get_books_by_author('Paulo Coelho');



23. Rollback and Commit Savepoint

Lab 3: Perform a transaction that includes inserting a new member, setting a SAVEPOINT, and rolling back to the savepoint after making updates.

Answer:

Query: Transaction with SAVEPOINT and ROLLBACK

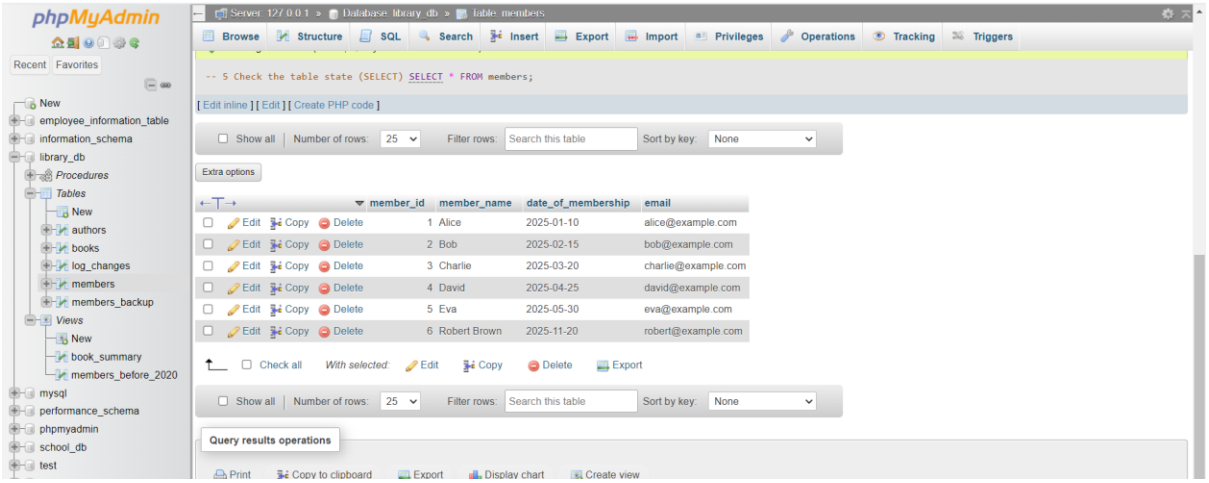
Suppose we have a members table with columns:

member_id, member_name, date_of_membership, email

```
-- Start the transaction
START TRANSACTION;

-- 1 Insert a new member
INSERT INTO members(member_id, member_name, date_of_membership, email)
VALUES (6, 'Robert Brown', '2025-11-20', 'robert@example.com');

-- 2 Set a SAVEPOINT
SAVEPOINT before_update;

-- 3 Update some records (example)
UPDATE members
SET member_name = 'Robert B.'
WHERE member_id = 6;

-- 4 Rollback to the savepoint
ROLLBACK TO SAVEPOINT before_update;

-- 5 Check the table state (SELECT)
SELECT * FROM members;
```

-- Optionally commit if everything is fine

COMMIT;



Lab 4: Use COMMIT after successfully inserting multiple books into the books table, then use ROLLBACK to undo a set of changes made after a savepoint.

Answer:

Query: COMMIT and ROLLBACK with multiple books

Suppose we have a books table:

book_id, title, author, publisher, year_of_publication, price

-- Start the transaction

START TRANSACTION;

-- 1 Insert multiple books

INSERT INTO books(book_id, title, author, publisher, year_of_publication, price)

VALUES

(7, 'Pride and Prejudice', 'Jane Austen', 'T. Egerton', 1813, 350),

(8, '1984', 'George Orwell', 'Secker & Warburg', 1949, 400);

-- 2 Commit the transaction (permanent)

COMMIT;

-- 3 Start a new transaction

START TRANSACTION;

-- Make some changes

UPDATE books SET price = price + 50 WHERE book_id = 7;

UPDATE books SET price = price + 100 WHERE book_id = 8;

-- 4 Set a savepoint

SAVEPOINT before_rollback;

-- Make additional changes

UPDATE books SET price = price * 2 WHERE book_id = 7;

-- 5 Rollback to savepoint (undo last update only)

ROLLBACK TO SAVEPOINT before_rollback;

-- 6 Check table state

SELECT * FROM books;

-- Commit final changes

COMMIT;