1. Introduction to SQL

**LAB EXERCISES**: Create a database called library\_db and a table books with columns: b

 **Lab 3**: book\_id, title, author, publisher, year\_of\_publication, and price. Insert five records into the table.

* CREATE DATABASE library\_db;
* Create the books table
  + CREATE TABLE books (
  + book\_id INT AUTO\_INCREMENT PRIMARY KEY,
  + title VARCHAR (255),
  + author VARCHAR(255),
  + publisher VARCHAR (255),
  + year\_of\_publication INT,
  + price DECIMAL (10, 2)

);

* INSERT INTO books (title, author, publisher, year\_of\_publication, price)

VALUES

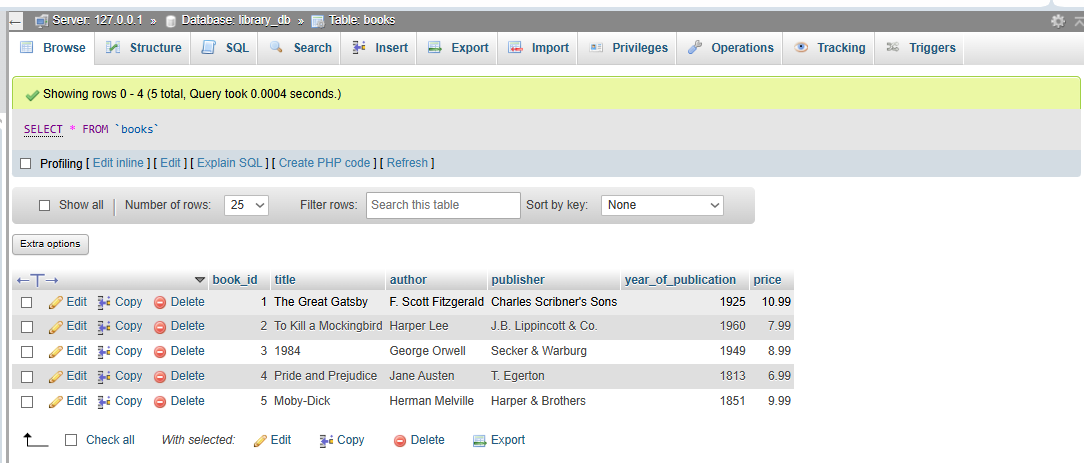
('The Great Gatsby', 'F. Scott Fitzgerald', 'Charles Scribner\'s Sons', 1925, 10.99),

('To Kill a Mockingbird', 'Harper Lee', 'J.B. Lippincott & Co.', 1960, 7.99),

('1984', 'George Orwell', 'Secker & Warburg', 1949, 8.99),

('Pride and Prejudice', 'Jane Austen', 'T. Egerton', 1813, 6.99),

('Moby-Dick', 'Herman Melville', 'Harper & Brothers', 1851, 9.99);



 **Lab 4**: Create a table member in library\_db with columns: member, member name,

date\_of\_membership, and email. Insert five records into this table.

* CREATE TABLE members (

member\_id INT AUTO\_INCREMENT PRIMARY KEY,

member\_name VARCHAR(100),

date\_of\_membership DATE,

email VARCHAR(100)

);

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

VALUES

('Harshani Patil', '2023-01-15', 'harshani.patil@example.com'),

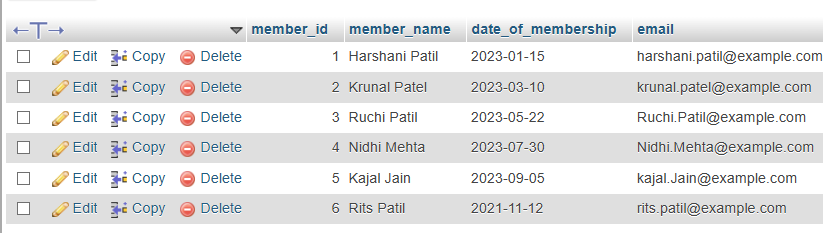
('Krunal Patel', '2023-03-10', 'krunal.patel@example.com'),

('Ruchi Patil', '2023-05-22', 'Ruchi.Patil@example.com'),

('Nidhi Mehta', '2023-07-30', 'Nidhi.Mehta@example.com'),

('Kajal Jain', '2023-09-05', 'kajal.Jain@example.com');

('Rits Patil', '2021-11-12', 'rits.patil@example.com');

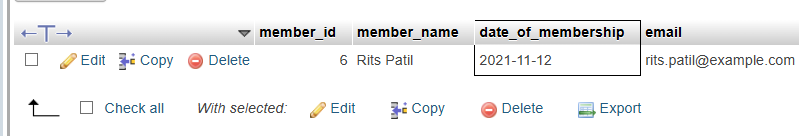


2. SQL Syntax

**LAB EXERCISES**:

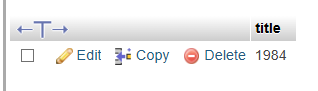
 **Lab 3**: Retrieve all members who joined the library before 2022. Use appropriate SQL syntax with WHERE and ORDER BY.

* 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.

* SELECT title FROM books WHERE author = 'George Orwell’ ORDER BY year\_of\_publication DESC;



3. SQL Constraints

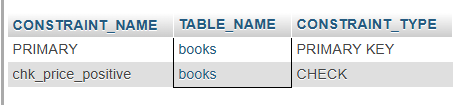
**LAB EXERCISES**:

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

* ALTER TABLE books ADD CONSTRAINT chk\_price\_positive CHECK (price > 0);

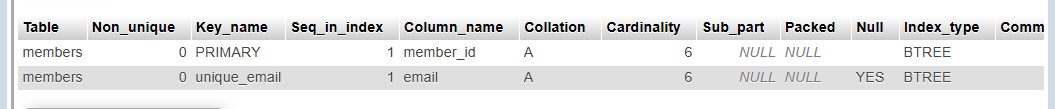
**Show Constraints in MySQL**

* SELECT CONSTRAINT\_NAME, TABLE\_NAME, CONSTRAINT\_TYPE FROM information\_schema.TABLE\_CONSTRAINTS WHERE TABLE\_SCHEMA = 'library\_db' AND TABLE\_NAME = 'books';



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

* ALTER TABLE members ADD CONSTRAINT unique\_email UNIQUE (email);
* SHOW INDEX FROM members;



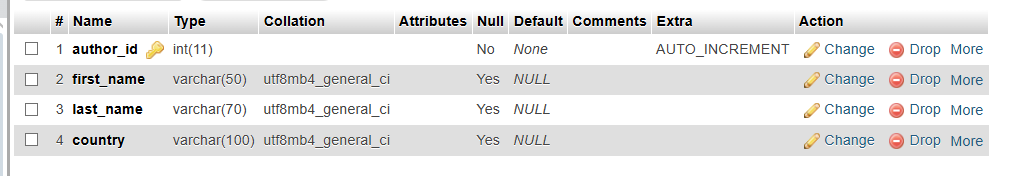
4. Main SQL Commands and Sub-commands (DDL)

**LAB EXERCISES**:

 **Lab 3**: Create a table authors with the following columns: author\_id, first\_name,

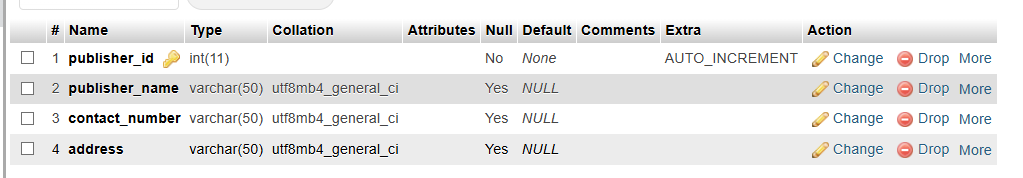
last\_name, and country. Set author\_id as the primary key.

* CREATE TABLE authors ( author\_id INT AUTO\_INCREMENT PRIMARY KEY, first\_name VARCHAR(50), last\_name VARCHAR(50),country VARCHAR(70));



 **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.

* CREATE table publishers (publisher\_id int AUTO\_INCREMENT PRIMARY KEY, publisher\_name varchar (50), contact\_number varchar (50), address varchar (50));



5. ALTER Command

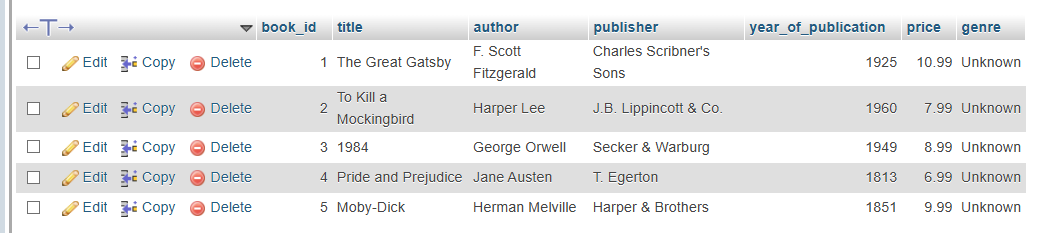
**LAB EXERCISES**:

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

* ALTER TABLE books ADD COLUMN genre VARCHAR(100);
* UPDATE books SET genre = 'Fiction' WHERE title = 'To Kill a Mockingbird';
* UPDATE books SET genre = 'Science Fiction' WHERE title = 'Dune';
* UPDATE books SET genre = 'Fantasy' WHERE title = 'Harry Potter and the Sorcerer''s Stone';
* UPDATE books SET genre = 'Historical' WHERE title = 'The Book Thief';

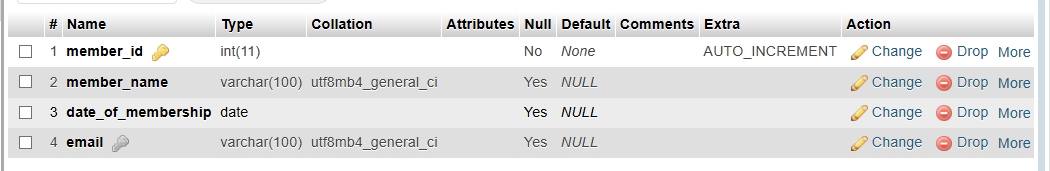
-- Add more UPDATE statements based on the actual titles in your table

* UPDATE books SET genre = 'Unknown';



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

* ALTER TABLE members MODIFY COLUMN email VARCHAR (100);

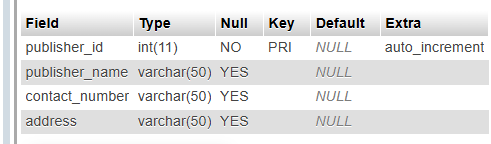


6. DROP Command

**LAB EXERCISES**:

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

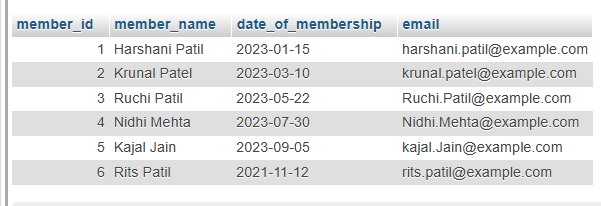
* DESCRIBE publishers;



* [DROP](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/drop-table.html) [TABLE](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/drop-table.html) publishers;

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

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



* DROP TABLE members;

7. Data Manipulation Language (DML)

**LAB EXERCISES**:

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

* INSERT INTO authors (first\_name, last\_name, country) VALUES ('Anita', 'Desai', 'India'), ('Chetan', 'Bhagat', 'India'), ('Jhumpa', 'Lahiri', 'USA');



* UPDATE authors SET last\_name = 'Mehta' WHERE first\_name = 'Chetan' AND last\_name = 'Bhagat';



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

* DELETE FROM books WHERE price > 100;

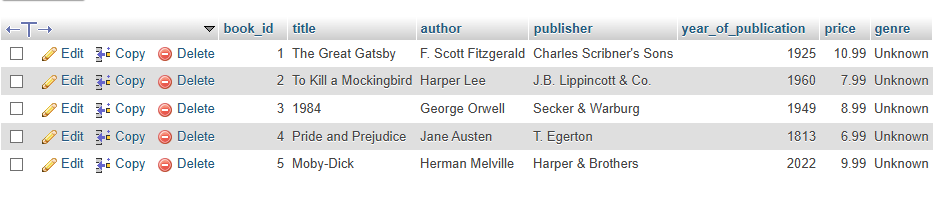


8. UPDATE Command

**LAB EXERCISES**:

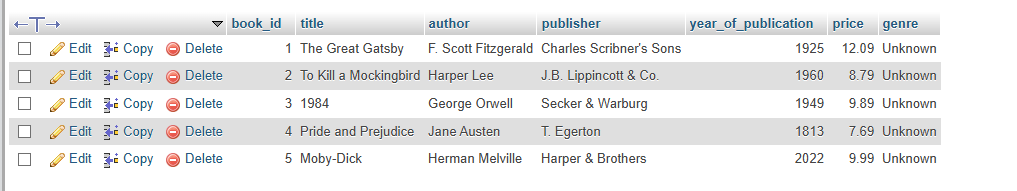
 **Lab 3**: Update the year\_of\_publication of a book with a specific book\_id.

* UPDATE books SET year\_of\_publication = 2022 WHERE book\_id = 5;



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

* UPDATE books SET price = price \* 1.10 WHERE year\_of\_publication < 2015;



9. DELETE Command

**LAB EXERCISES**:

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

* DELETE FROM members WHERE date\_of\_membership < '2020-01-01';
* SELECT \* FROM members WHERE date\_of\_membership < '2020-01-01';

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

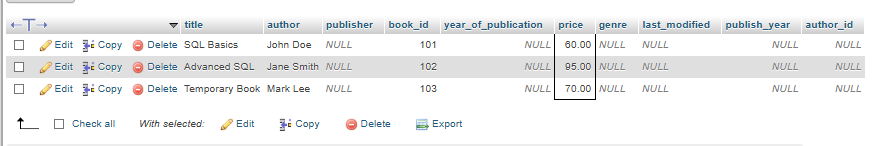
* DELETE FROM books WHERE author IS NULL;
* SELECT \* FROM books WHERE author IS NULL;

10. Data Query Language (DQL)

**LAB EXERCISES**:

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

* SELECT \* FROM books WHERE price BETWEEN 50 AND 100;



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

* SELECT \* FROM books ORDER BY author ASC LIMIT 3;

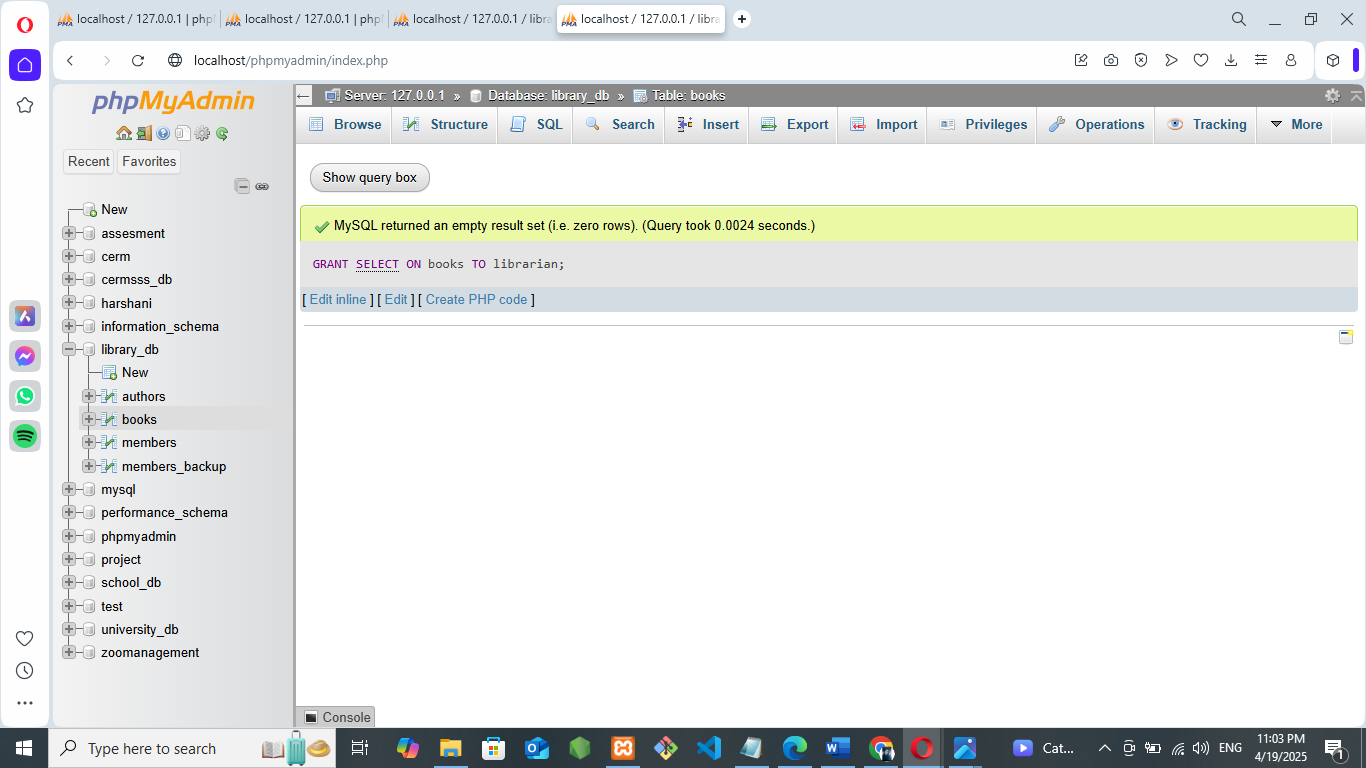


11. Data Control Language (DCL)

**LAB EXERCISES**:

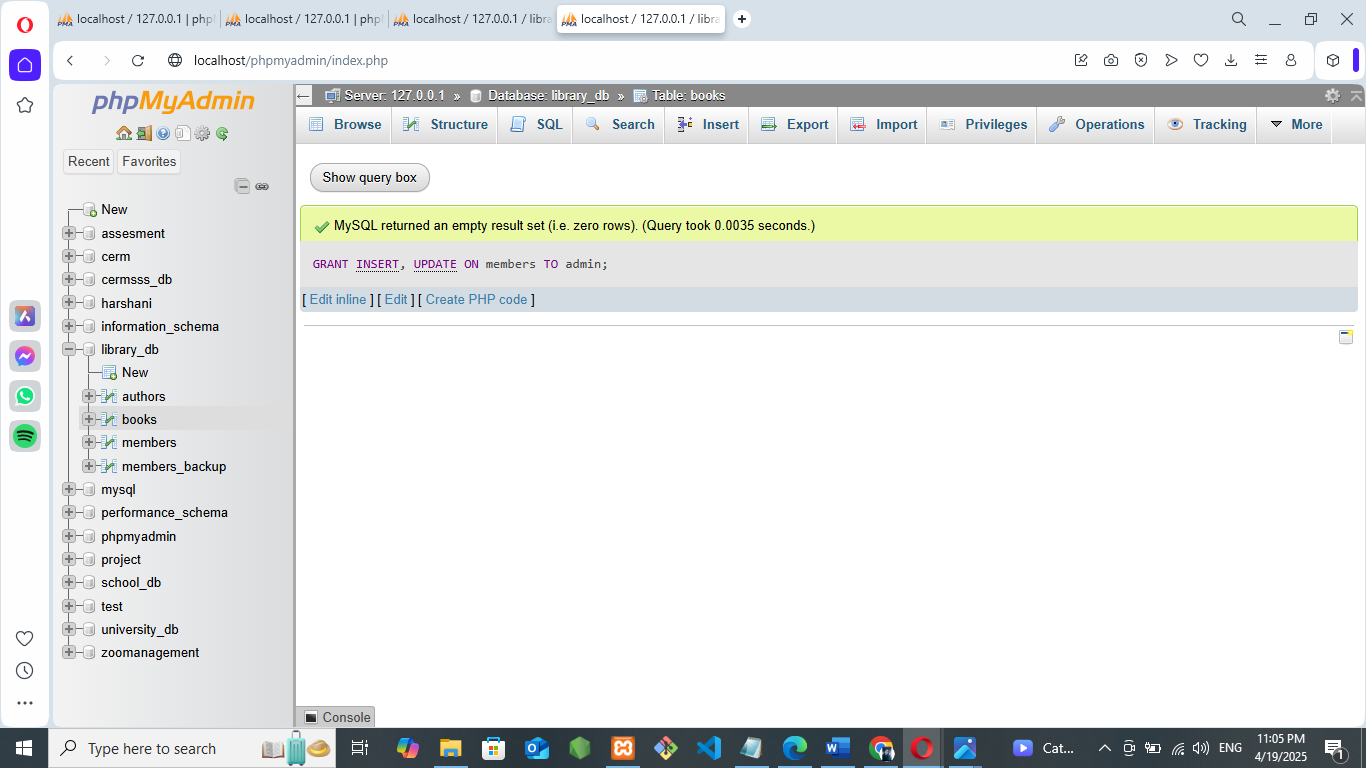
 **Lab 3**: Grant SELECT permission to a user named librarian on the books table.

* GRANT SELECT ON books TO librarian;



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

* GRANT INSERT, UPDATE ON members TO admin;

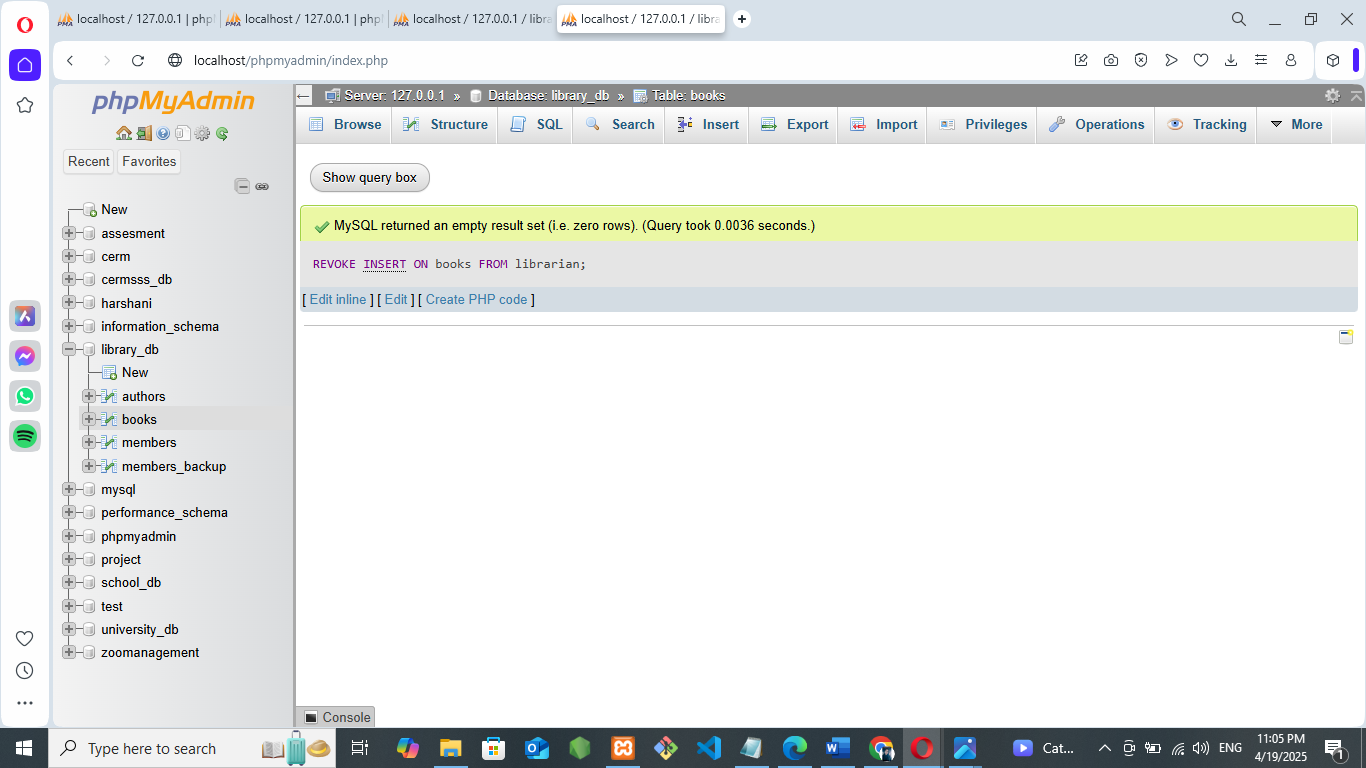


12. REVOKE Command

**LAB EXERCISES**:

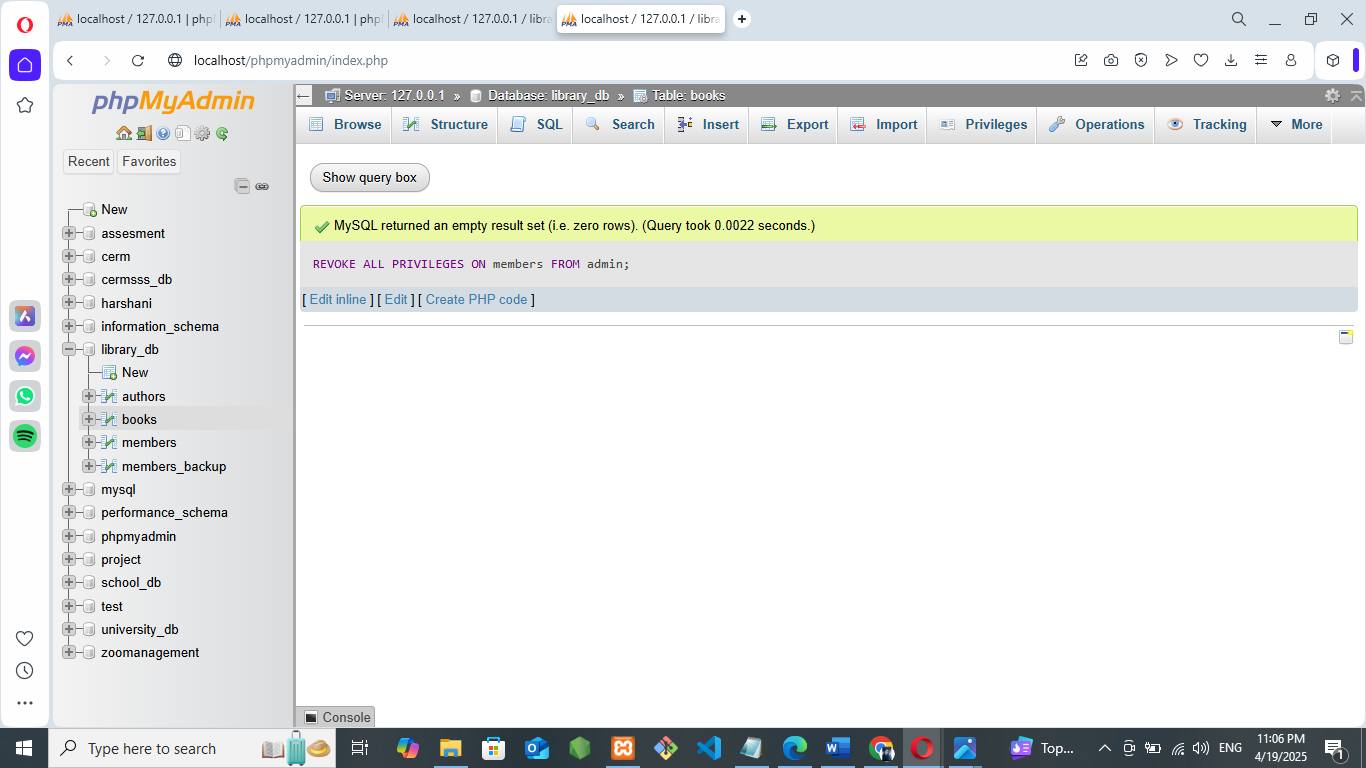
 **Lab 3**: Revoke the INSERT privilege from the user librarian on the books table.

* REVOKE INSERT ON books FROM librarian;



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

* REVOKE ALL PRIVILEGES ON members FROM admin;



13. Transaction Control Language (TCL)

**LAB EXERCISES**:

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

Start a transaction (optional in some DBs)

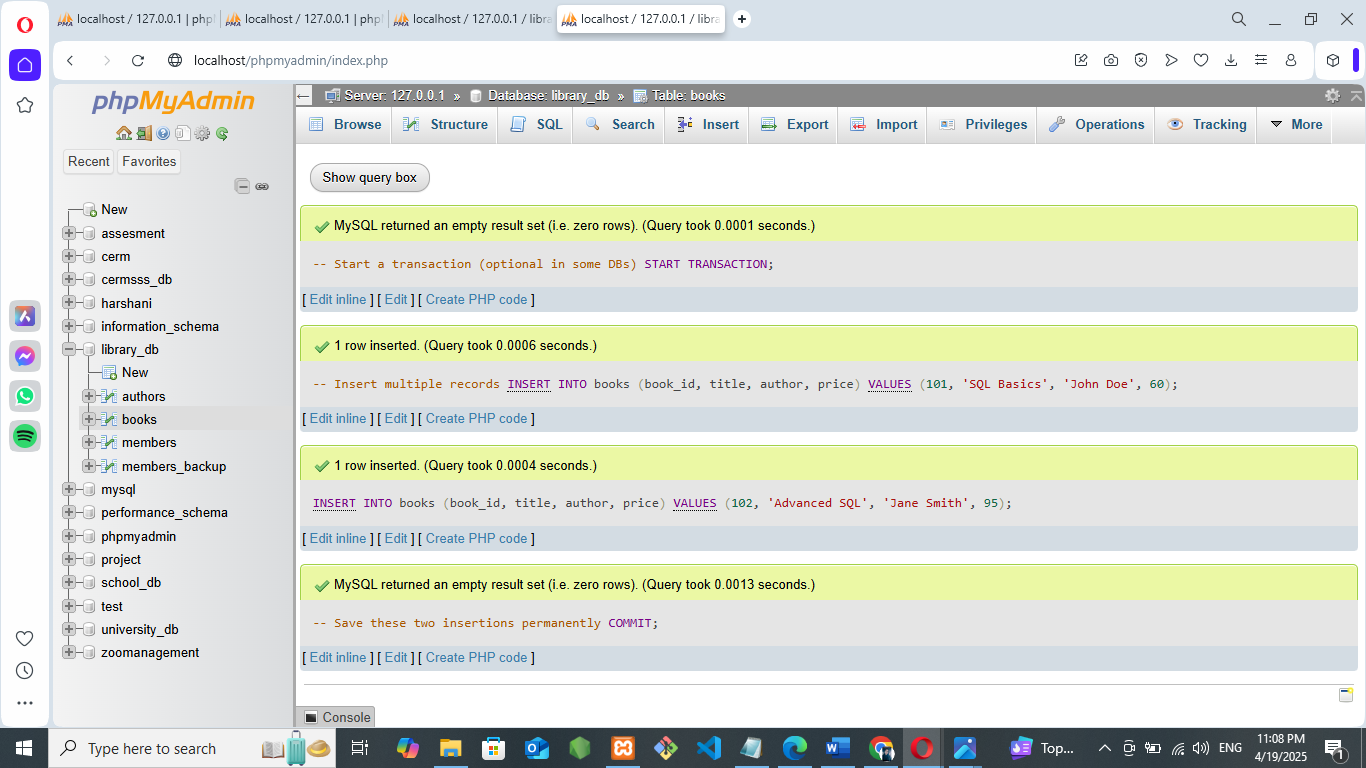
* START TRANSACTION;

Insert multiple records

* INSERT INTO books (book\_id, title, author, price) VALUES (101, 'SQL Basics', 'John Doe', 60);
* INSERT INTO books (book\_id, title, author, price) VALUES (102, 'Advanced SQL', 'Jane Smith', 95);

Save these two insertions permanently

* COMMIT;

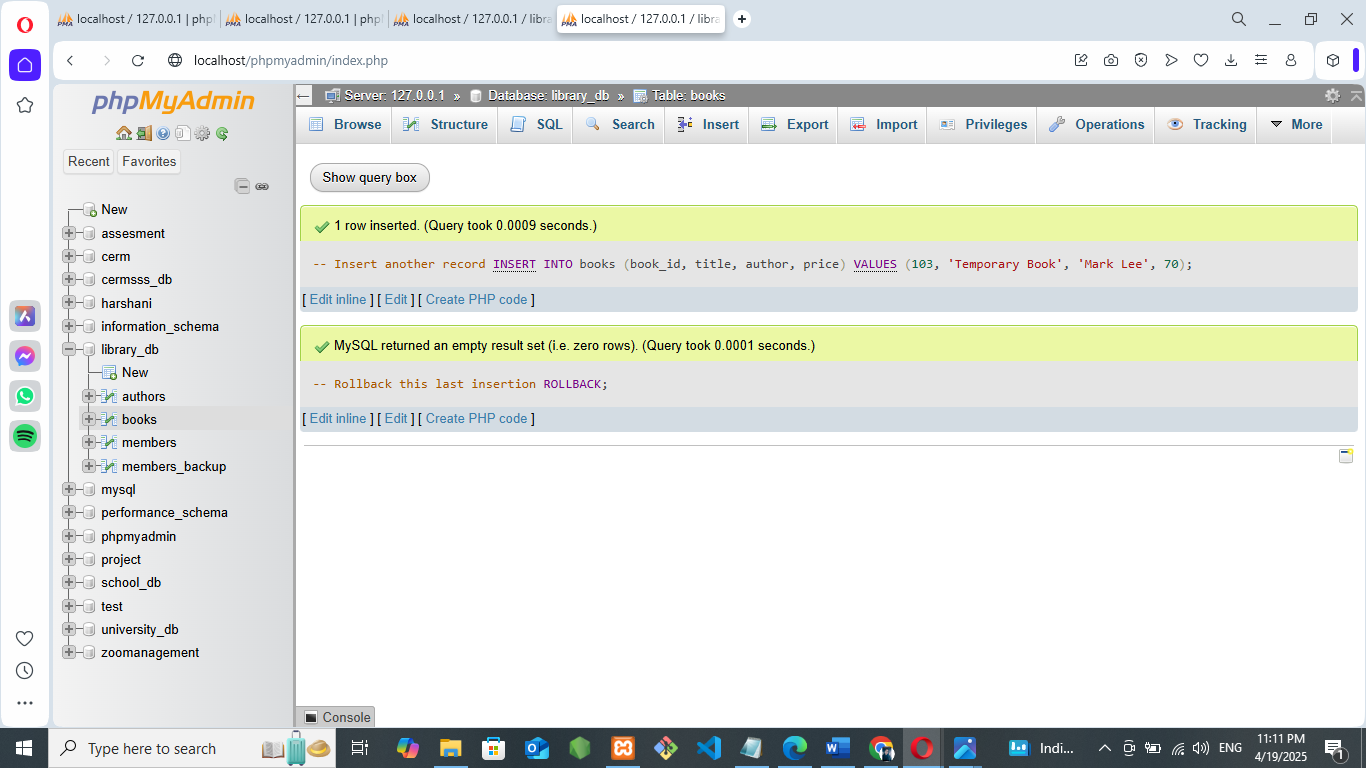


Insert another record

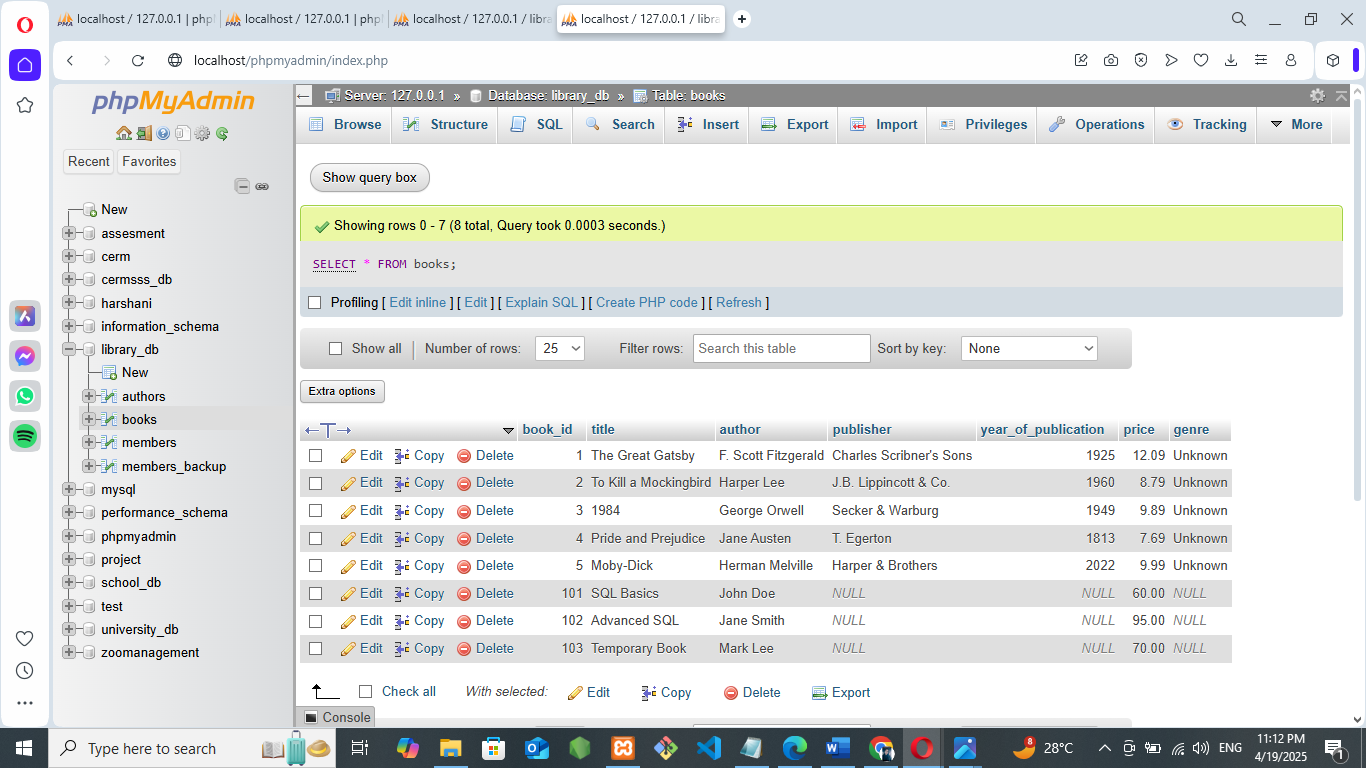
* INSERT INTO books (book\_id, title, author, price) VALUES (103, 'Temporary Book', 'MarkLee',70);

Rollback this last insertion

* ROLLBACK;



* SELECT \* FROM books;



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

14. SQL Joins

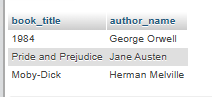
**LAB EXERCISES**:

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

* SELECT books.title AS book\_title, CONCAT(authors.first\_name, ' ', authors.last\_name) AS author\_name FROM books

INNER JOIN

authors ON books.author\_id = authors.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.

* SELECT books.title AS book\_title,CONCAT(authors.first\_name, ' ', authors.last\_name) AS author\_name,books.author\_id AS book\_author\_id, authors.author\_id AS author\_table\_id

FROM books

LEFT JOIN authors ON books. author\_id = authors.author\_id

UNION

SELECT books.title AS book\_title,

CONCAT(authors.first\_name, ' ', authors.last\_name) AS author\_name,

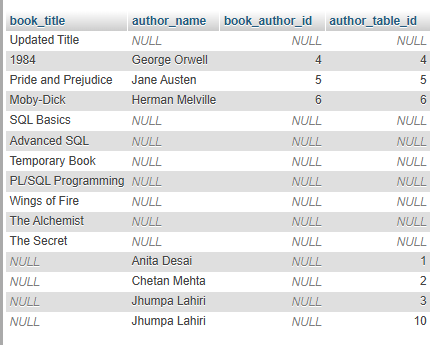
books.author\_id AS book\_author\_id,

authors.author\_id AS author\_table\_id

FROM

authors

LEFT JOIN books ON books.author\_id = authors.author\_id;



15. SQL Group By

**LAB EXERCISES**:

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

* 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.

* SELECT YEAR (date\_of\_membership) AS join\_year,COUNT(\*) AS member\_count FROM members

GROUP BY YEAR(date\_of\_membership)

ORDER BY join\_year LIMIT 0, 25;



16. SQL Stored Procedure

**LAB EXERCISES**:

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

* DELIMITER $$

CREATE PROCEDURE GetBooksByAuthor(IN author\_name VARCHAR(255))

BEGIN

SELECT \* FROM books WHERE author = author\_name;

END $$

DELIMITER ;

* CALL GetBooksByAuthor('John Doe');



 **Lab 4**: Write a stored procedure that takes book\_id as an argument and returns the price of the book.

* DELIMITER $$

CREATE PROCEDURE GetBookPriceById(IN book\_id INT)

BEGIN

SELECT price FROM books WHERE book\_id = book\_id;

END $$

DELIMITER ;

* CALL GetBookPriceById(101);

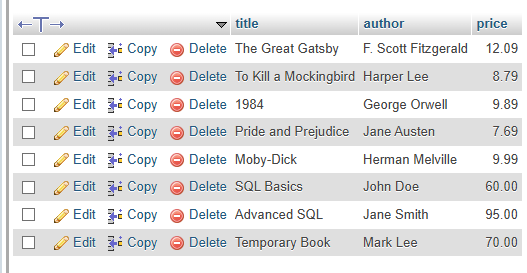


17. SQL View

**LAB EXERCISES**:

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

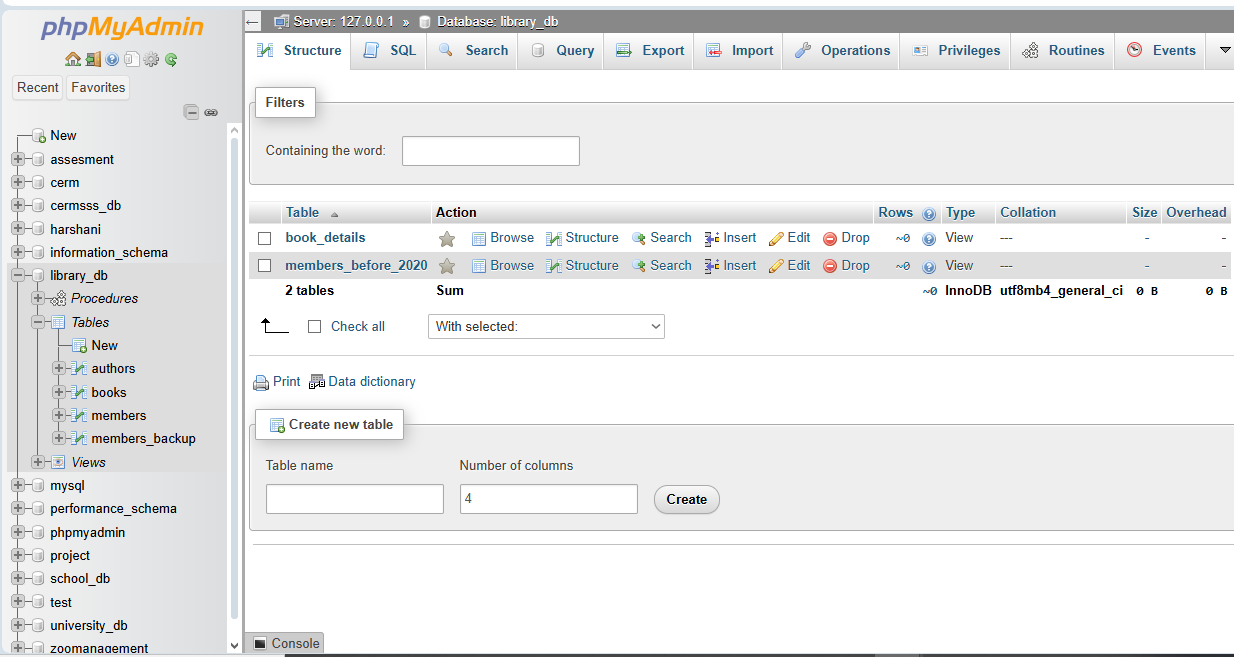
* CREATE VIEW book\_details AS SELECT title, author, price FROM books;
* SELECT \* FROM book\_details;



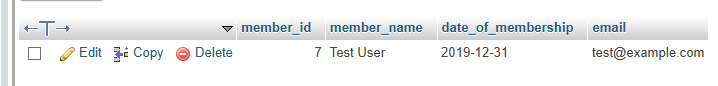
 **Lab 4**: Create a view to display members who joined before 2020.

* CREATE VIEW members\_before\_2020 AS

SELECT \*FROM members WHERE date\_of\_membership < '2020-01-01';



* INSERT INTO members (member\_id, member\_name, date\_of\_membership, email)VALUES (7, 'Test User', '2019-12-31', 'test@example.com');
* SELECT \* FROM members\_before\_2020;



18. SQL Trigger

**LAB EXERCISES**:

 **Lab 3**: Create a trigger to automatically update the last\_modified timestamp of the

books table whenever a record is updated.

* ALTER TABLE books ADD last\_modified DATETIME;

Create the trigger:

* DELIMITER $$

CREATE TRIGGER update\_books\_last\_modified

BEFORE UPDATE ON books

FOR EACH ROW

BEGIN

SET NEW.last\_modified = NOW();

END$$

DELIMITER ;

Test the Trigger:

* UPDATE books SET title = 'Updated Title' WHERE book\_id = 1;

Then check:

* SELECT \* FROM books WHERE book\_id = 1;



 **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.

Create the log\_changes table:

* CREATE TABLE log\_changes (log\_id INT AUTO\_INCREMENT PRIMARY KEY,book\_id INT,title VARCHAR(255), author VARCHAR(255),deleted\_at DATETIME);

Create the AFTER DELETE trigger:

* DELIMITER $$

CREATE TRIGGER after\_book\_delete

AFTER DELETE ON books

FOR EACH ROW

BEGIN

INSERT INTO log\_changes (book\_id, title, author, deleted\_at)

VALUES (OLD.book\_id, OLD.title, OLD.author, NOW ());

END$$

DELIMITER ;

### Delete a book:

### DELETE FROM books WHERE book\_id = 2;

### SELECT \* FROM log\_changes;



19. Introduction to PL/SQL

**LAB EXERCISES**:

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

-- Change delimiter to $$ to allow semicolons inside the procedure body

* DELIMITER $$

Create the procedure

CREATE PROCEDURE insert\_book(

IN book\_id INT,

IN title VARCHAR(255),

IN author VARCHAR(255),

IN publish\_year INT

)

BEGIN

Insert a new book into the books table

INSERT INTO books (book\_id, title, author, publish\_year) VALUES (book\_id, title, author, publish\_year);

Display a confirmation message

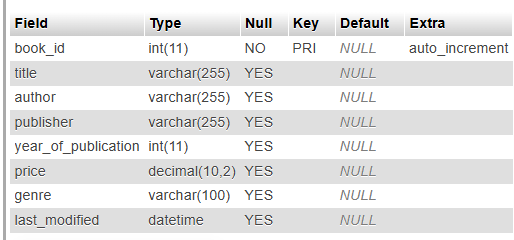
SELECT CONCAT('Book "', title, '" by ', author, ' has been successfully added.') AS confirmation\_message;

END $$

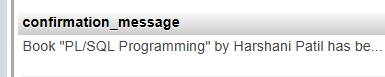
Reset the delimiter back to semicolon

DELIMITER ;

* DESCRIBE books;



* ALTER TABLE books ADD COLUMN publish\_year INT;
* CALL insert\_book(201, 'PL/SQL Programming', 'Harshani Patil', 2025);



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

* DELIMITER $$

CREATE PROCEDURE get\_total\_books()

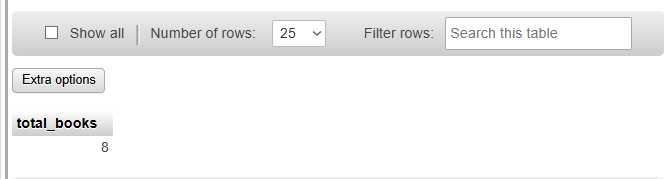
BEGIN

SELECT COUNT(\*) AS total\_books FROM books;

END $$

DELIMITER ;

* CALL get\_total\_books();



20. PL/SQL Syntax

**LAB EXERCISES**:

 **Lab 3**: Write a PL/SQL block to declare variables for book\_id and price, assign values, and display the results.

* DELIMITER $$

CREATE PROCEDURE display\_book\_details()

BEGIN

Declare variables

DECLARE book\_id INT DEFAULT 101;

DECLARE price DECIMAL(8,2) DEFAULT 499.99;

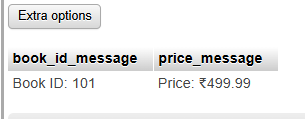
Display results

SELECT CONCAT ('Book ID: ', book\_id) AS book\_id\_message, CONCAT ('Price: ₹', price) AS price\_message;

END $$

DELIMITER;

* CALL display\_book\_details ();



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

* DELIMITER $$

CREATE PROCEDURE book\_price\_arithmetic ()

BEGIN

Declare constants using variables (MySQL doesn't support true constants in procedures)

DECLARE book\_price1 DECIMAL (8,2) DEFAULT 350.75;

DECLARE book\_price2 DECIMAL (8,2) DEFAULT 499.25;

Declare result variables

DECLARE total\_price DECIMAL (8,2);

DECLARE price\_difference DECIMAL (8,2);

DECLARE average price DECIMAL (8,2);

Perform arithmetic operations

SET total\_price = book\_price1 + book\_price2;

SET price\_difference = ABS (book\_price1 - book\_price2);

SET average\_price = (book\_price1 + book\_price2) / 2;

Display results

SELECT

book\_price1 AS 'Book Price 1',

book\_price2 AS 'Book Price 2',

total\_price AS 'Total Price',

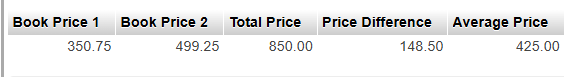
price\_difference AS 'Price Difference',

average\_price AS 'Average Price';

END $$

DELIMITER;

* CALL book\_price\_arithmetic ();



21. PL/SQL Control Structures

**LAB EXERCISES**:

 **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.

* DELIMITER $$

CREATE PROCEDURE check\_book\_price()

BEGIN

DECLARE book\_price DECIMAL(8,2) DEFAULT 120.50; -- You can change this value

IF book\_price > 100 THEN

SELECT 'The book price is above $100.' AS message;

ELSE

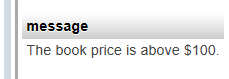
SELECT 'The book price is $100 or below.' AS message;

END IF;

END $$

DELIMITER;

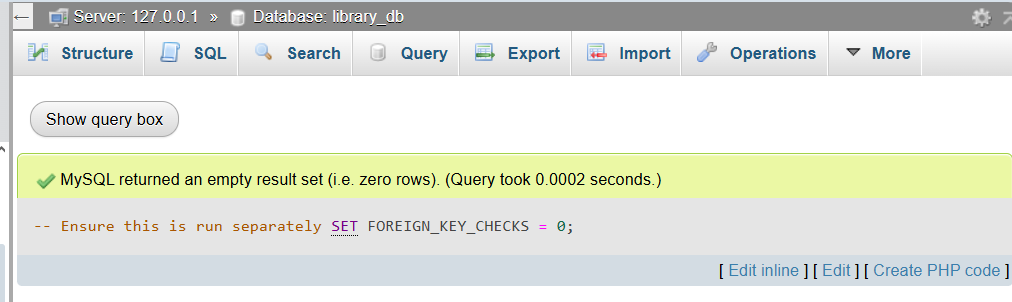
* CALL check\_book\_price();



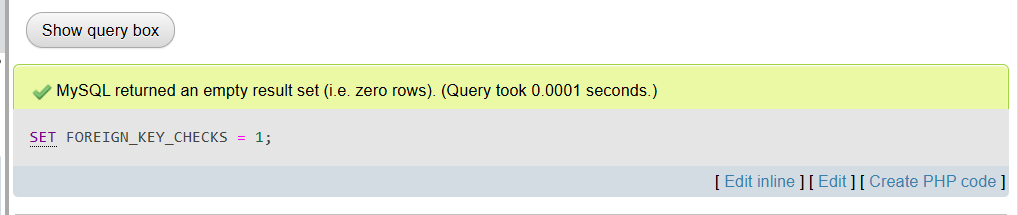
 **Lab 4**: Use a FOR LOOP in PL/SQL to display the details of all books one by one.

-- Ensure this is run separately

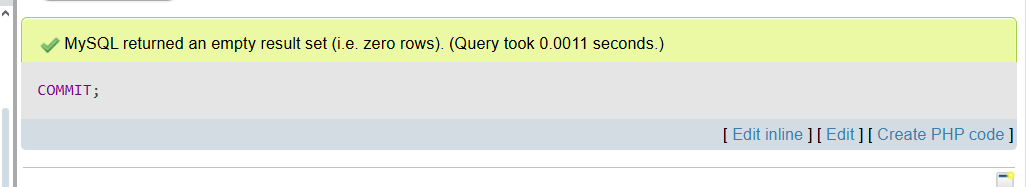
* SET FOREIGN\_KEY\_CHECKS = 0;



* SET FOREIGN\_KEY\_CHECKS = 1;



* COMMIT;



22. SQL Cursors

**LAB EXERCISES**:

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

members table.

* DELIMITER $$

CREATE PROCEDURE FetchAllMembers()

BEGIN

Declare variables to hold column data

DECLARE done INT DEFAULT FALSE;

DECLARE v\_id INT;

DECLARE v\_name VARCHAR(100);

DECLARE v\_join\_date DATE;

DECLARE v\_email VARCHAR(100);

Declare the cursor

DECLARE member\_cursor CURSOR FOR

SELECT member\_id, member\_name, date\_of\_membership, email FROM members;

Declare handler for when there are no more rows

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

Open the cursor

OPEN member\_cursor;

read\_loop: LOOP

FETCH member\_cursor INTO v\_id, v\_name, v\_join\_date, v\_email;

IF done THEN

LEAVE read\_loop;

END IF;

Since MySQL does not support DBMS\_OUTPUT, we'll use SELECT to display

SELECT v\_id AS 'ID', v\_name AS 'Name', v\_join\_date AS 'Joined', v\_email AS 'Email';

END LOOP;

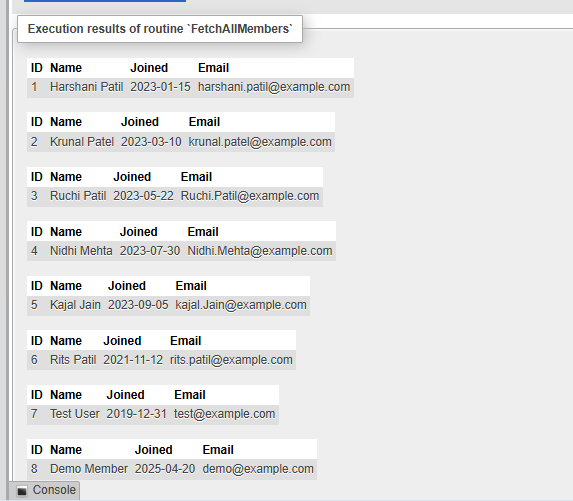
Close the cursor

CLOSE member\_cursor;

END$$

DELIMITER ;

* [CALL](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/call.html) `FetchAllMembers`();



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

Drop existing procedure

* DROP PROCEDURE IF EXISTS get\_books\_by\_author;

Create the procedure

* DELIMITER $$

CREATE PROCEDURE get\_books\_by\_author(IN author\_name VARCHAR(100))

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE book\_title VARCHAR(255);

Declare cursor

DECLARE book\_cursor CURSOR FOR

SELECT title FROM books WHERE author = author\_name;

Exit handler

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

Open cursor

OPEN book\_cursor;

read\_loop: LOOP

FETCH book\_cursor INTO book\_title;

IF done THEN

LEAVE read\_loop;

END IF;

Display the book title

SELECT CONCAT('Book Title: ', book\_title) AS result;

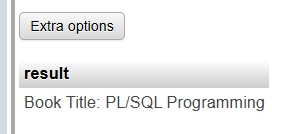
END LOOP;

CLOSE book\_cursor;

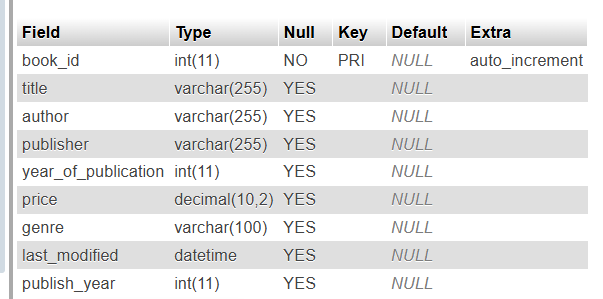
END $$

DELIMITER ;

* CALL get\_books\_by\_author('Harshani Patil');



* DESCRIBE books;



23. Rollback and Commit Savepoint

**LAB EXERCISES**:

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

-- Start the transaction

START TRANSACTION;

-- Insert a new member

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

VALUES ('Demo Member', '2025-04-20', 'demo@example.com');

-- Set a savepoint

SAVEPOINT after\_insert;

-- Update the newly inserted member's name (this will be undone)

UPDATE members

SET member\_name = 'Updated Demo'

WHERE email = 'demo@example.com';

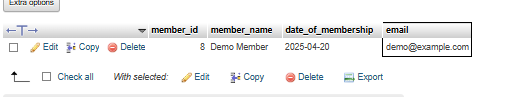
-- Rollback to the savepoint (undo the update)

ROLLBACK TO SAVEPOINT after\_insert;

-- Commit the transaction (only the insert remains)

COMMIT;

* SELECT \* FROM members WHERE email = 'demo@example.com';



 **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.

-- Step 1: Start a transaction

START TRANSACTION;

-- Step 2: Insert multiple new books

INSERT INTO books (title, author, publish\_year)

VALUES

('Wings of Fire', 'A.P.J. Abdul Kalam', 1999),

('The Alchemist', 'Paulo Coelho', 1988),

('The Secret', 'Rhonda Byrne', 2006);

-- Step 3: Commit the successful inserts

COMMIT;

-- Step 4: Start a new transaction to make changes

START TRANSACTION;

-- Step 5: Set a savepoint

SAVEPOINT before\_update;

-- Step 6: Perform updates (which we will undo)

UPDATE books SET author = 'Unknown Author' WHERE title = 'The Secret';

UPDATE books SET publish\_year = 2025 WHERE title = 'The Alchemist';

-- Step 7: Rollback the updates (back to savepoint)

ROLLBACK TO SAVEPOINT before\_update;

-- Step 8: Final commit to save only what came before the savepoint

COMMIT;

* SELECT \* FROM books WHERE title IN ('Wings of Fire', 'The Alchemist', 'The Secret');

