।। શ્રી રામ ।।

Abhay Gajjar

Module – 4 DBMS

Theory Questions

1.) Introduction to SQL

1.) What is SQL, and why is it essential in database management?

Answer : SQL is a Structured Query language for storing, manipulating and retrieving data in databases.

* **It is Essential in Database Management** :

1. **Data Integrity and Accuracy**: Ensures data consistency and correctness through validation and constraints, preventing anomalies.

2. **Data Security**: Provides access control and encryption to protect sensitive information from unauthorized access.
3. **Efficient Data Retrieval**: Optimizes query performance and uses indexing to allow quick access to needed information.
4. **Data Redundancy Management**: Minimizes duplication through normalization and centralizes data storage to avoid inconsistencies.
5. **Backup and Recovery**: Offers tools for regular backups and recovery mechanisms to restore data in case of failures.
6. **Multi-User Access**: Supports concurrent access by multiple users while managing transactions to maintain data integrity.
7. **Scalability**: Adapts to growing data volumes and complex queries, ensuring continued performance as organizations expand.
8. **Data Analysis and Reporting**: Facilitates data aggregation and analysis, enabling informed decision-making through business intelligence.
9. **Support for Data Relationships**: Organizes data in relational formats, allowing complex queries and relationships between data entities.

2.) Explain the difference between DBMS and RDBMS ?

Answer :

No.	DBMS	RDBMS
1.	DBMS stores data as file.	RDBMS stores data in tabular form.
2.	Individual access of data elements.	Multiple Data Elements are accessible together.
3.	Normalization is not present.	Normalization is present.
4.	No Support for distributed database.	support for distributed database.
5.	Data is stored in a Small Quantity.	Data stored is a Large Amount.
6.	DBMS supports Single users.	RDBMS supports multiple users.
7.	The software and hardware requirements are low.	The software and hardware requirements are higher.
8.	Example: Microsoft Access , XML , Forxpro etc.	Example : Oracle , SQL , Microsoft SQL Server etc.
9.	Security is less.	More Security measures provided.

3.) Describe the role of SQL in managing relational databases ?

Answer :

1) Data Definition Language (DDL)

SQL includes commands for defining and modifying the structure of databases and tables. This includes creating, altering, and dropping database objects such as tables, indexes, views, and constraints.

➔ Operation = create , alter , drop , truncate , rename.

2) Data Manipulation Language (DML)

SQL allows users to manipulate and retrieve data stored in tables. This includes inserting, updating, deleting, and querying data.

➔ Operation = select

3) Data Query Language (DQL)

SQL is renowned for its querying capabilities, allowing users to retrieve specific data from tables based on various criteria.

➔ Operation = select , insert , update & delete.

4) Data Control Language (DCL)

SQL includes commands for controlling access to data within the database.

➔ Operation = grant & revoke permission to users.

5) Transaction Control Language (TCL)

TCL used to manage transactions and ensure data integrity in relational databases.

➔ Operation = start transaction , commit , rollback etc.

4.) What are the key features of SQL ?

Answer :

1. Data Definition Language (DDL): SQL provides a set of commands to define and modify the structure of a database, including creating tables, modifying table structure, and dropping tables.
2. Data Manipulation Language (DML): SQL provides a set of commands to manipulate data within a

database, including adding, modifying, and deleting data.

3. Data Query Language: SQL provides a rich set of commands for querying a database to retrieve data, including the ability to filter, sort, group, and join data from multiple tables.
4. Transaction Control Language: SQL supports transaction processing, which allows users to group a set of database operations into a single transaction that can be rolled back in case of failure.
5. Data Integrity: SQL includes features to enforce data integrity, such as the ability to specify constraints on the values that can be inserted or updated in a table, and to enforce referential integrity between tables.
6. User Access Control: SQL provides mechanisms to control user access to a database, including the ability to grant and revoke privileges to perform certain operations on the database.
7. Portability: SQL is a standardized language, meaning that SQL code written for one database management system can be used on another system with minimal modification.

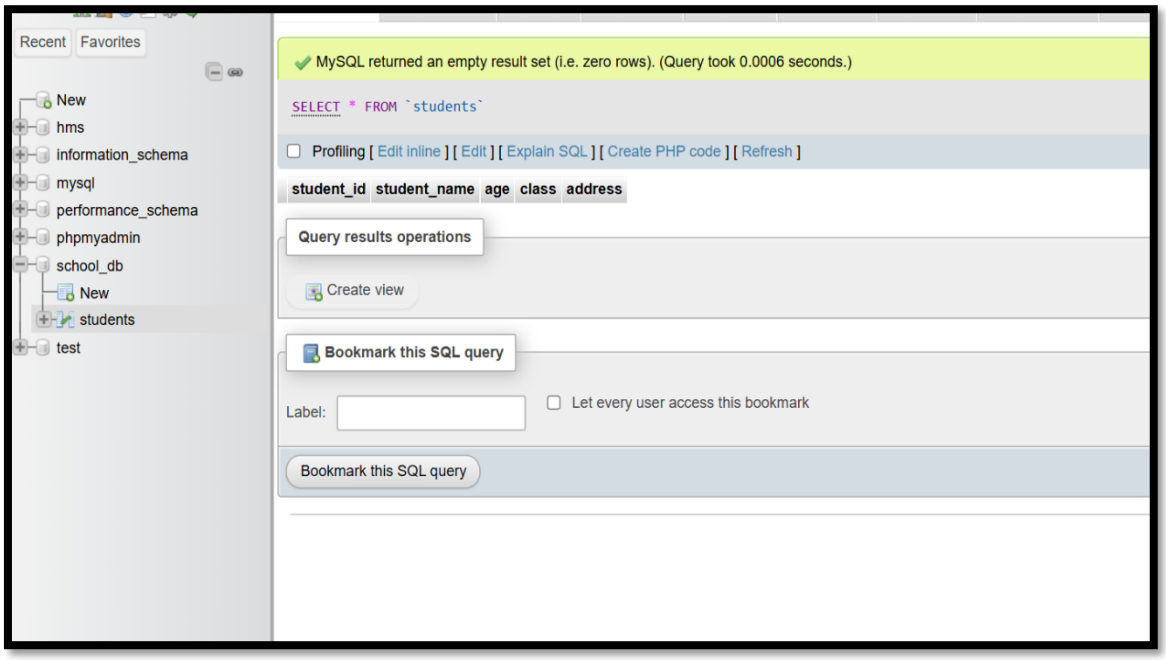
LAB EXERCISES:-

Lab 1: Create a new database named school_db and a table called students with the following columns: student_id, student_name, age, class, and address.

Answer :

1) CREATE DATABASE school_db;
2) CREATE TABLE students (student_id int PRIMARY KEY, student_name varchar(30), age int, class varch ar(5), address varchar(20));

Output :



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

Answer : INSERT INTO students (student_id,student_name,age,class,address) VALUES

(1,"Abhay",21,"A","Hanuman Nagar") ,

(2,"Raj",22,"B","Shree Ram Nagar") ,

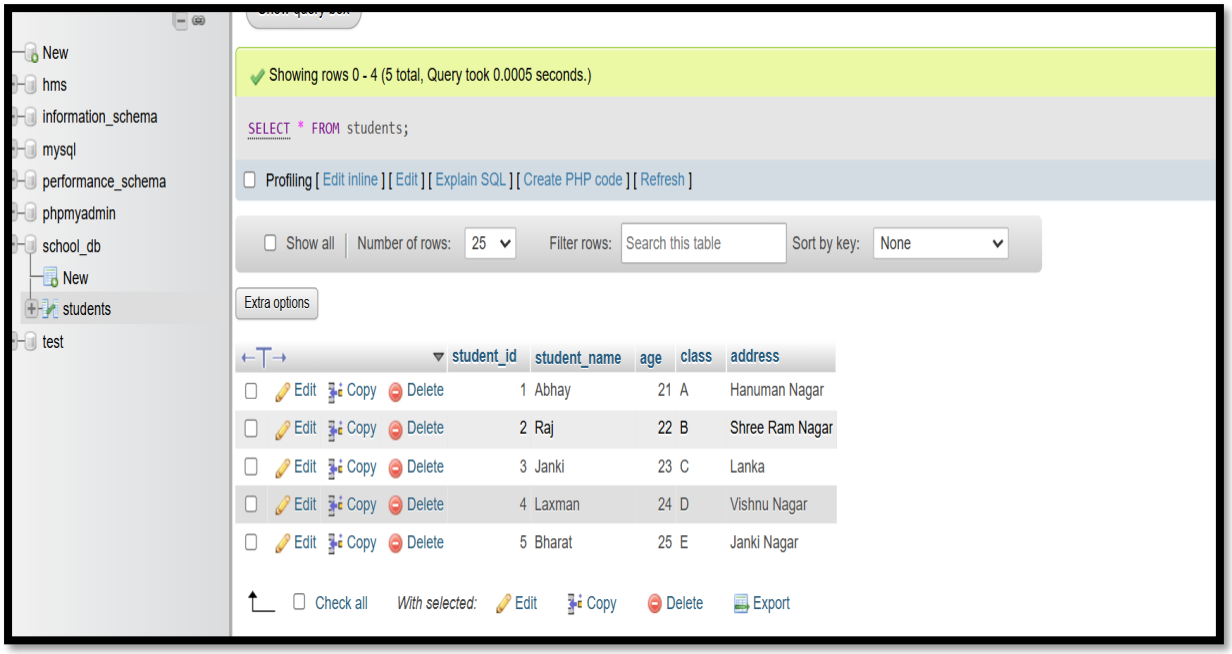
(3,"Janki",23,"C","Lanka") ,

(4,"Laxman",24,"D","Vishnu Nagar") ,

(5,"Bharat",25,"E","Janki Nagar");

SELECT * FROM students;

Output :



2.) SQL Syntax

1.) What are the basic components of SQL syntax ?

Answer :

1) SELECT: Retrieves data from a database.
2) INSERT: Adds new rows to a table.
3) UPDATE: Modifies existing data in a table.
4) DELETE: Removes rows from a table.
5) WHERE: Filters records based on a condition.
6) JOIN: Combines rows from two or more tables based on a related column.
7) CREATE: Creates a new table or database.
8) DROP: Deletes a table or database.

2.) Write the general structure of an SQL SELECT statement ?

Answer :

SELECT column1, column2, ... FROM table_name WHERE condition GROUP BY column HAVING condition ORDER BY column;

3.) Explain the role of clauses in SQL statements ?

Answer :

Key SQL Clauses and Their Roles :

1. **SELECT**: Specifies the columns to retrieve from the database.
2. **FROM**: Indicates the table from which to retrieve the data.
3. **WHERE**: Filters records based on a specified condition, returning only those that meet the criteria.
4. **GROUP BY**: Groups rows that have the same values in specified columns into summary rows, like totals or counts.
5. **HAVING**: Filters groups based on a condition, similar to WHERE but used for groups.
6. **ORDER BY**: Sorts the result set in either ascending or descending order based on one or more columns.
7. **JOIN**: Combines rows from two or more tables based on a related column between them.
8. **LIMIT**: Specifies the maximum number of records to return in the result set.

➔ **Example :**

```
SELECT name, age FROM employees WHERE
 age > 30 GROUP BY department HAVING
COUNT(*) > 5 ORDER BY age DESC LIMIT 10;
```

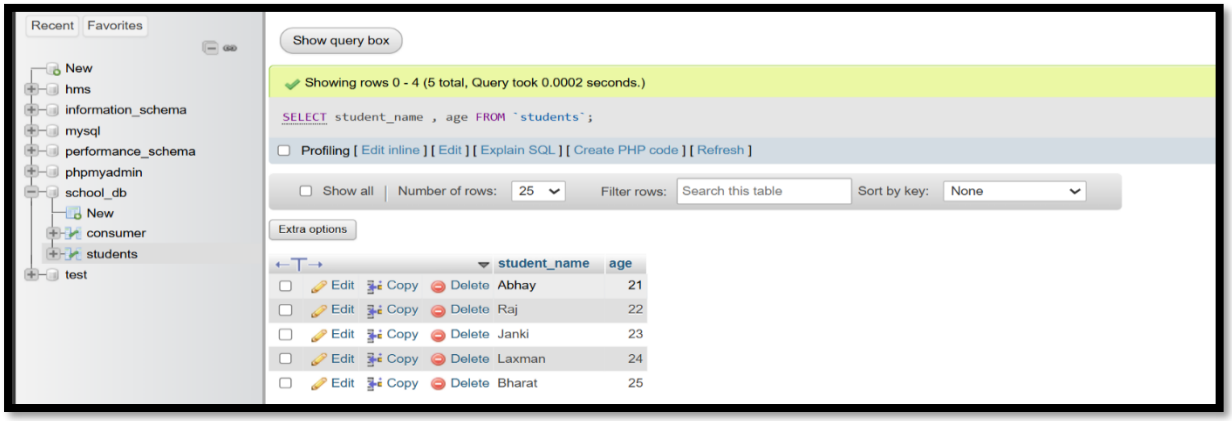
LAB EXERCISES :

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

Answer :

SELECT student_name , age FROM `students`;

Output :

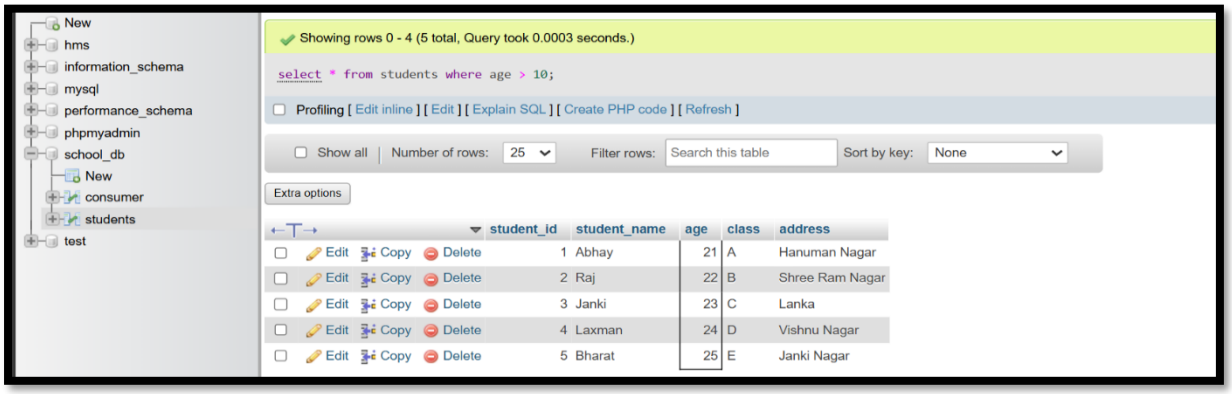


Lab 2: Write SQL queries to retrieve all students whose age is greater than 10.

Answer :

SELECT * FROM `students` where age > 10;

Output :



3.) SQL Constraints

1.) What are constraints in SQL? List and explain the different types of constraints.

Answer :

Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.

The available constraints in SQL are:

- **NOT NULL:** This constraint tells that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more**.**
- **UNIQUE**: This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.
- **PRIMARY KEY:** A primary key is a field which can uniquely identify each row in a table. And this

constraint is used to specify a field in a table as primary key.

- **FOREIGN KEY:** A Foreign key is a field which can uniquely identify each row in a another table. And this constraint is used to specify a field as Foreign key.
- **CHECK**: This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.
- **DEFAULT:** This constraint specifies a default value for the column when no value is specified by the user.

2.) How do PRIMARY KEY and FOREIGN KEY constraints differ ?

Answer :

No.	Primary Key	Foreign Key
1	A primary key is used to ensure data in the specific column is unique.	A foreign key is a column or group of columns in a relational database table that provides a link

		between data in two tables.
2	It uniquely identifies a record in the relational database table.	It refers to the field in a table which is the primary key of another table.
3	Only one primary key is allowed in a table.	Whereas more than one foreign key is allowed in a table.
4	It is a combination of UNIQUE and Not Null constraints.	It can contain duplicate values and a table in a relational database.
5	It does not allow NULL values .	It can also contain NULL values.
6	Its value cannot be deleted from the parent table.	Its value can be deleted from the child table.
7	It constraint can be implicitly defined on the temporary tables.	It constraint cannot be defined on the local or global temporary tables.

3.) What is the role of NOT NULL and UNIQUE constraints?

Answer :

(1) NOT NULL :

This constraint can’t store a null value in a column. If a column is specified as NOT NULL then we will not be able to store null in this particular column anymore.

When NOT NULL constraint applied to a column ensures that the particular or specific column always contains a value, i.e., we can’t insert a new record, or update a record while not providing a value for that field.

(2) UNIQUE :

UNIQUE constraint in SQL ensures that all the values stored in a column are totally different from each other. Furthermore, the UNIQUE Constraint prevents two records from having similar values in a column, which means that the values must not be repeated.

We can the UNIQUE SQL constraint in a single column or multiple columns.

This constraint helps us to uniquely identify each row in the table means that for a particular column, all the rows should have unique values. Furthermore, We can have more than one UNIQUE column in a table.

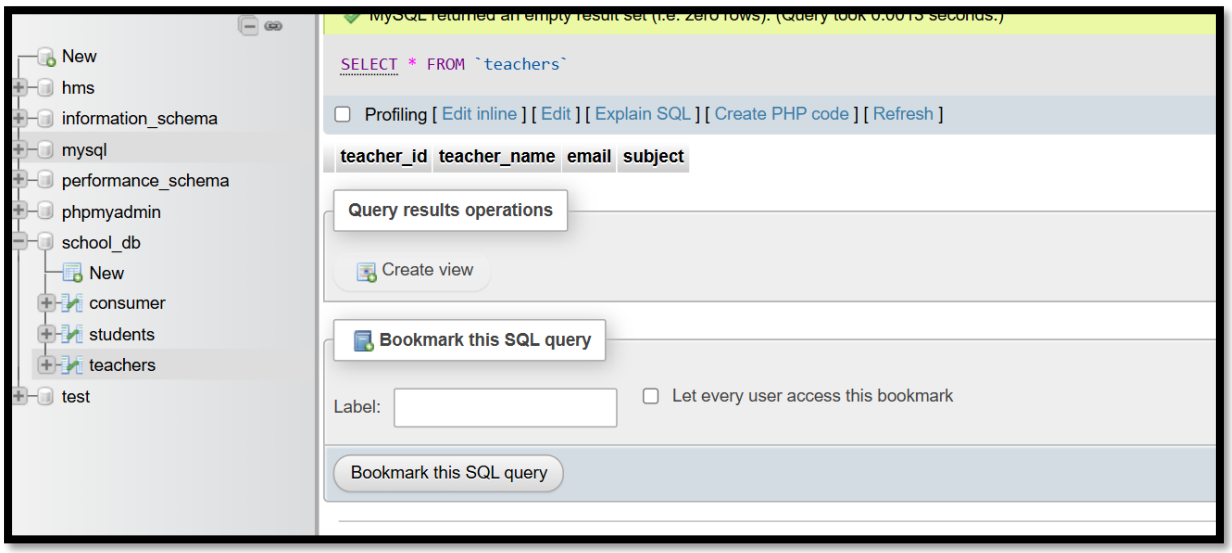
LAB EXERCISES :

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

Answer :

```
CREATE TABLE teachers
(
   teacher_id int,
   teacher_name varchar(30) not null,
   email varchar(50) UNIQUE ,
   subject varchar(40) not null
);
```

Output :



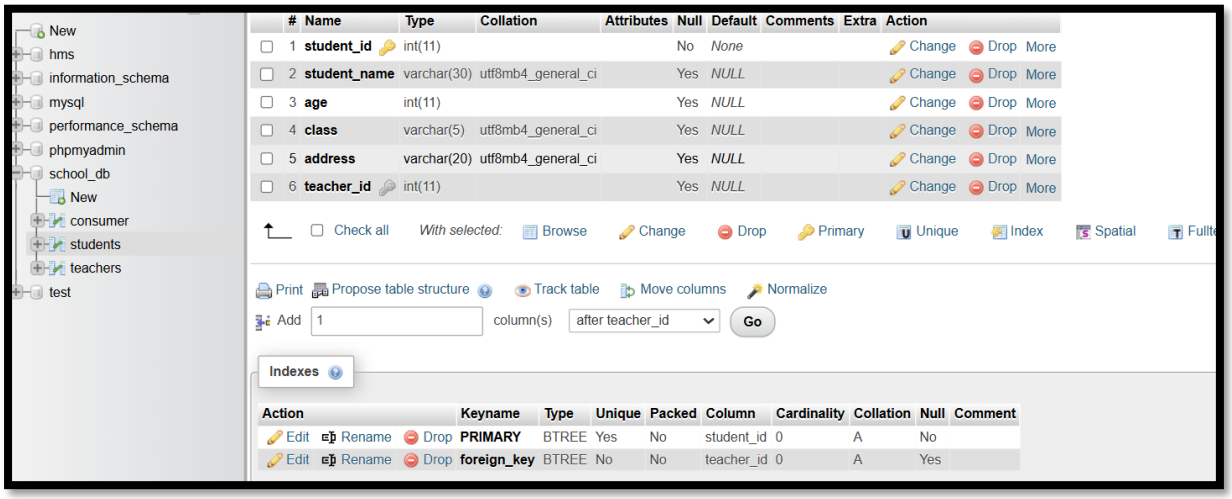
Lab 2: Implement a FOREIGN KEY constraint to relate the teacher_id from the teachers table with the students table.

Answer :

ALTER TABLE students ADD CONSTRAINT foreign_key

FOREIGN KEY(teacher_id) REFERENCES teachers (teacher_id);

Output :



4.) Main SQL Commands and Sub

Commands (DDL)

1.) Define the SQL Data Definition Language (DDL)?

Answer :

Data Definition Language (DDL) :- SQL provides a set of commands to define and modify the structure of a database, including creating tables, modifying table structure, and dropping tables.

➔ **Operation :** Create , Alter , Drop , Truncate , Rename.

2.) Explain the CREATE command and its syntax ?

Answer :

The CREATE command in SQL is used to create new objects in the database, such as tables, views, indexes, and more. It is fundamental to defining the structure of the database.

Syntax :

```
CREATE TABLE table_name (
    column1 datatype [constraint],
    column2 datatype [constraint],
```

```
    ...
);
```

3.) What is the purpose of specifying data types and constraints during table creation ?

Answer :

The Purpose of specifying data types and constraints during table creation is :

- It ensures data integrity and accuracy.
- It protects the table from malicious attacks.
- Constraints limit the type of data that can be stored in a table.
- Constraints include primary keys, foreign keys, and unique values

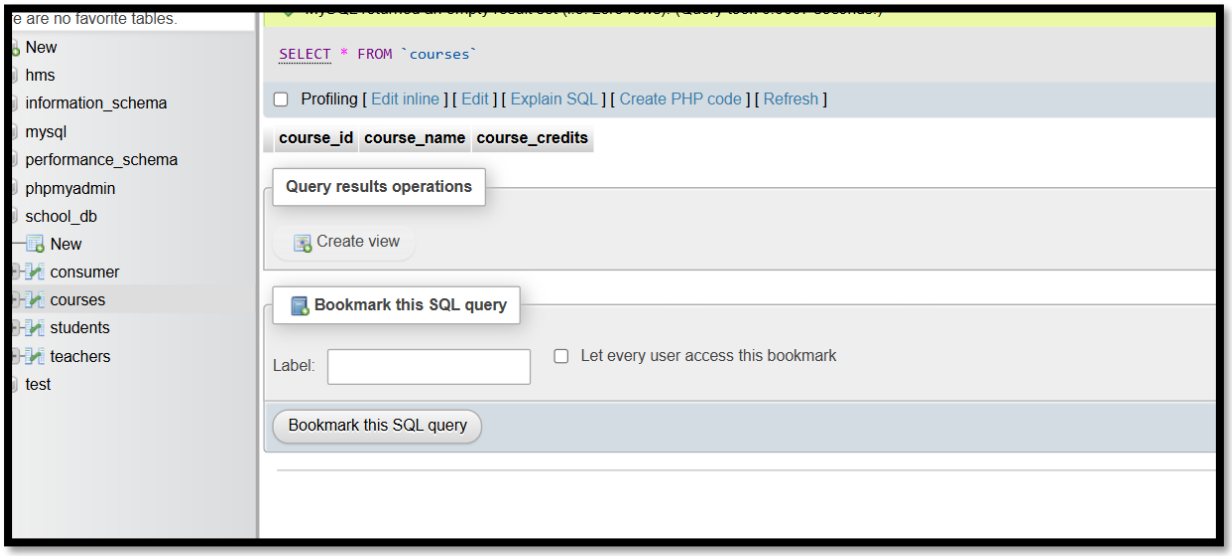
LAB EXERCISES :

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

Answer :

```
CREATE table courses
(
    course_id int PRIMARY key,
    course_name varchar(30),
    course_credits char(20)
);
```

Output :

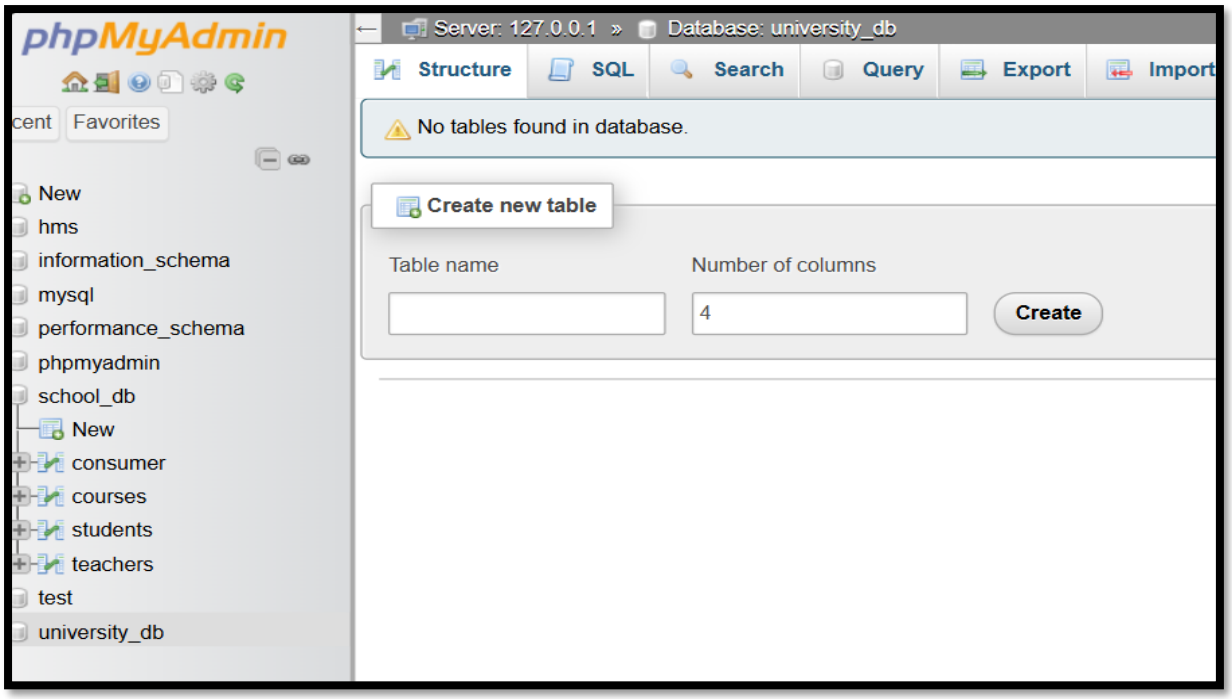


Lab 2: Use the CREATE command to create a database university_db.

Answer :

```
Create Database university_db;
```

Output :



5.) ALTER Command

1.) What is the use of the ALTER command in SQL ?

Answer :

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

2.) How can you add, modify, and drop columns from a table using ALTER ?

Answer :

1. Adding a Column:

To add a new column to an existing table.

Example = ALTER TABLE students ADD address VARCHAR(255);

2. Modifying a Column:

To change the data type or properties of an existing column.

Example = ALTER TABLE students

MODIFY address VARCHAR(500);

3. Dropping a Column:

To remove a column from an existing table.

Example = ALTER TABLE students

DROP COLUMN address;

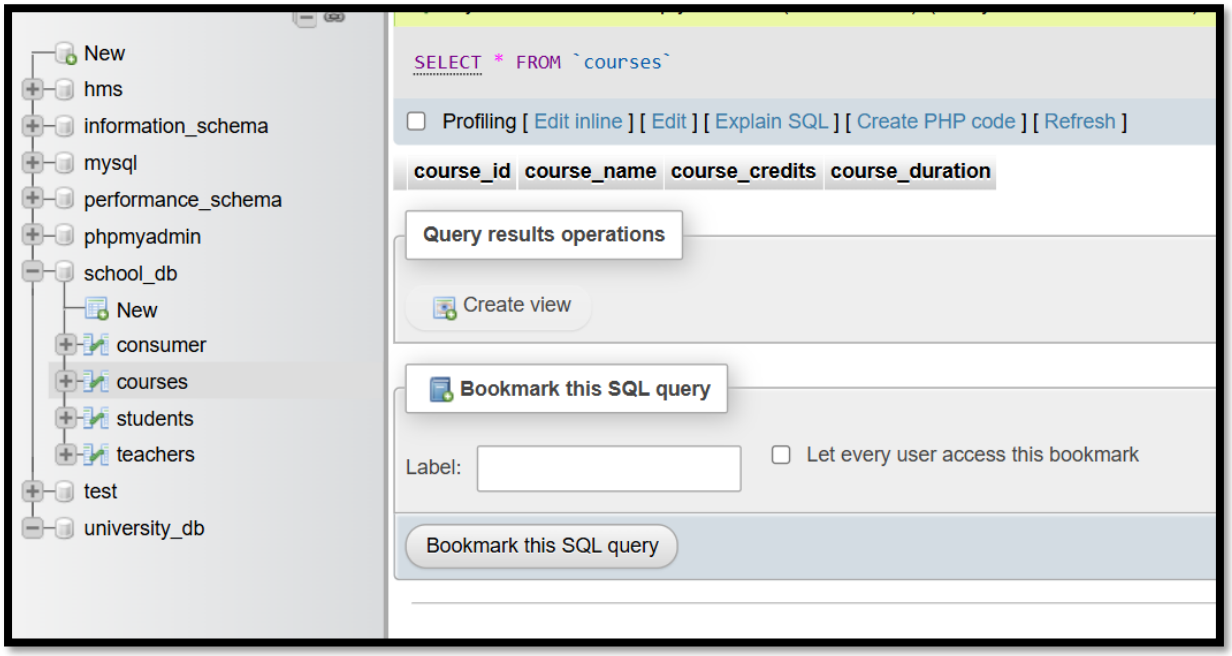
LAB EXERCISES :

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

Answer :

alter table courses add COLUMN course_duration varchar(40) not null;

Output :

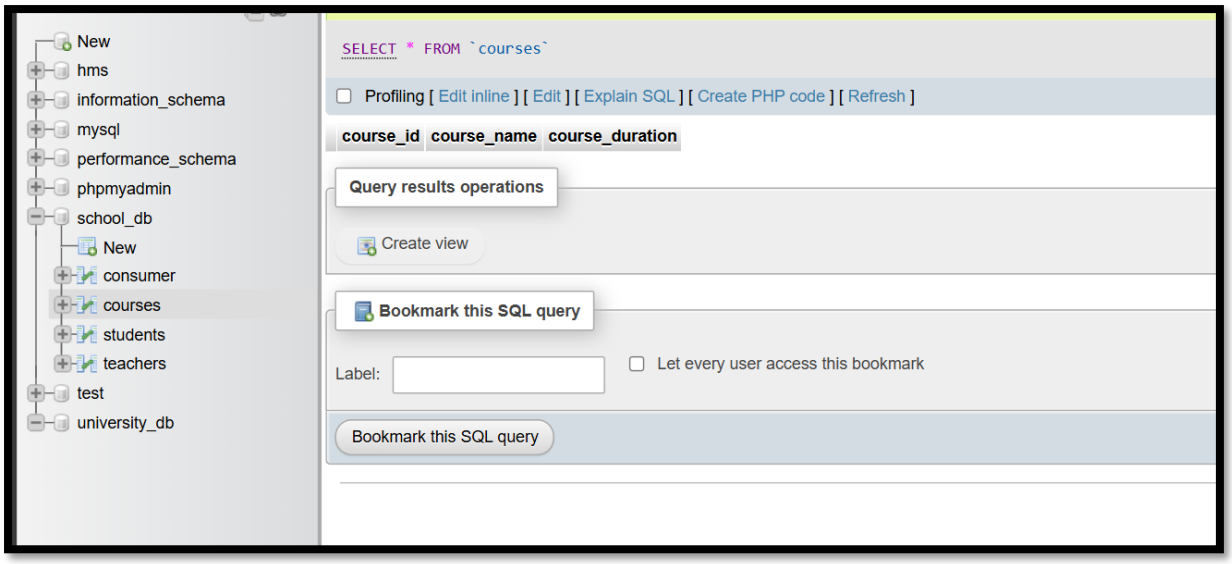


Lab 2: Drop the course_credits column from the courses table.

Answer :

ALTER TABLE courses DROP COLUMN course_credits;

Output :



6.) DROP Command

1.) What is the function of the DROP command in SQL?

Answer :

The DROP COLUMN command is used to delete a column in an existing table.

The DROP INDEX command is used to delete an index in a table.

The DROP DATABASE command is used is to delete an existing SQL database.

The DROP TABLE command deletes a table in the database.

Example = ALTER TABLE Customers
DROP COLUMN ContactName;

2.) What are the implications of dropping a table from a database?

Answer :

The Implications of dropping a table from a database are :

- Permanently removing the table structure and all associated data.
- Loss of information in the table unless a backup is available.
- Removal of indexes, statistics, permissions, triggers, and constraints.
- Stored procedures and views referencing the table need to be dropped explicitly.

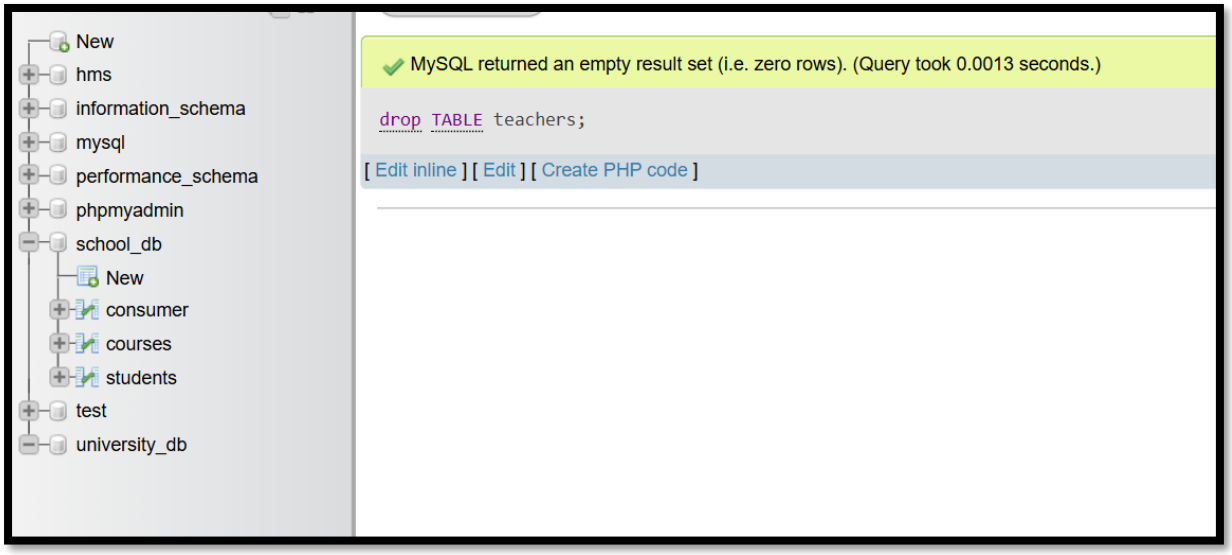
LAB EXERCISES :

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

Answer :

drop TABLE teachers;

Output :

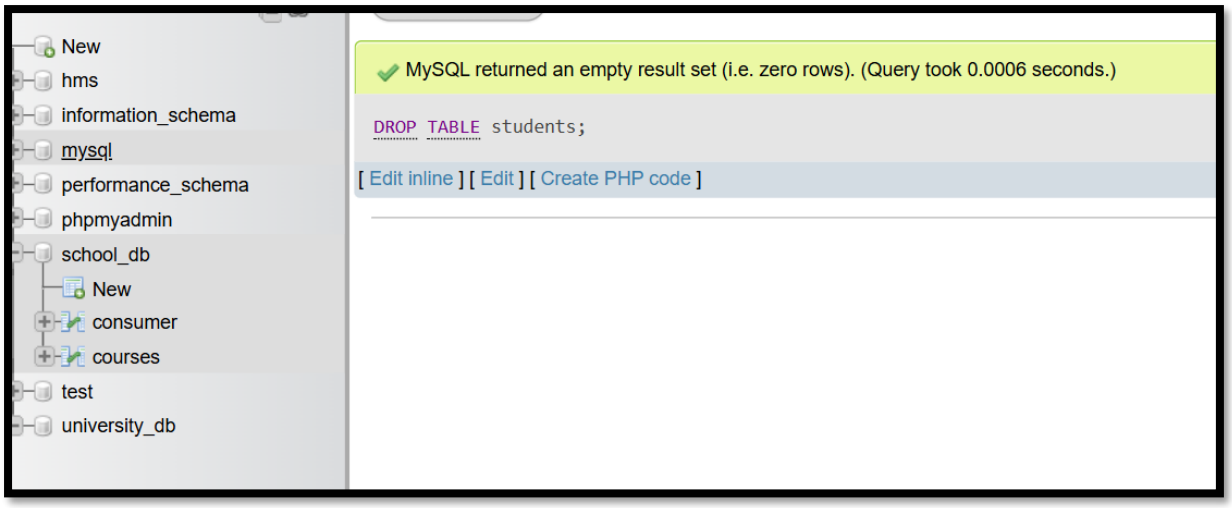


Lab 2: Drop the students table from the school_db database and verify that the table has been removed.

Answer :

DROP TABLE students;

Output :



7.) Data Manipulation Language (DML)

1.) Define the INSERT, UPDATE, and DELETE commands in SQL ?

Answer :

INSERT	Adds new data to a table.	INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);

UPDATE	Modifies existing data in a table.	UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;
DELETE	Removes data from a table.	DELETE from table_name WHERE condition;

2.) What is the importance of the WHERE clause in UPDATE and DELETE operations ?

Answer :

- **WHERE Clause with UPDATE Statement :**

The UPDATE statement is used to modify the existing records in a table. Using the SQL WHERE clause with the UPDATE statement, we can update particular records. If the WHERE clause is not used, the UPDATE statement would affect all the records of a table.

- **WHERE Clause with DELETE Statement :**

The DELETE statement is used to delete existing records in a table.

The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted.

LAB EXERCISES :

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

Answer :

```
insert into courses VALUES(1, "java developer", "2 months"), (2,"python developer","5 months"),(3,"Ai Architect","6 months");
```

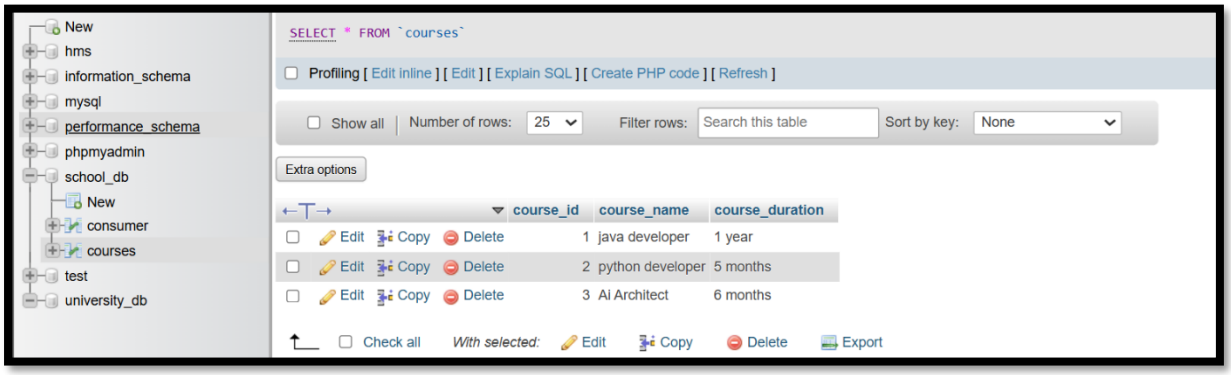
Output :

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

Answer :

UPDATE courses set course_duration="1 year" where course_id=1;

Output :

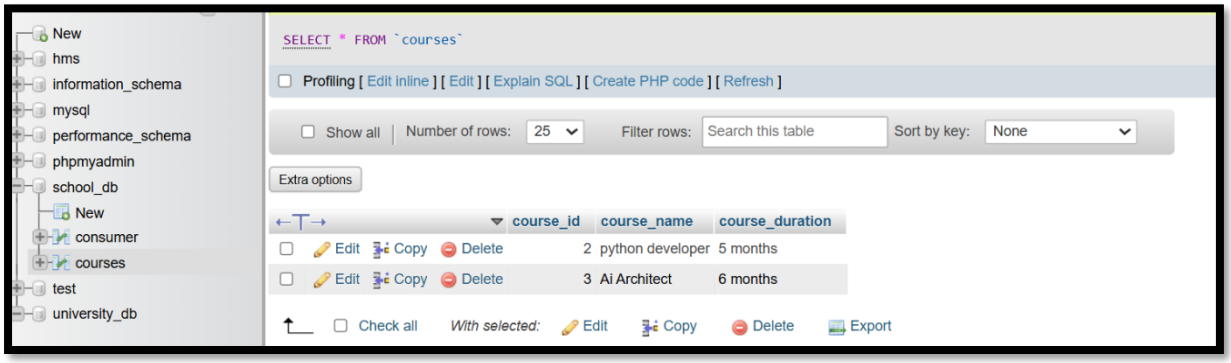


Lab 3: Delete a course with a specific course_id from the courses table using the DELETE command.

Answer :

delete from courses where course_id=1;

Output :



8.) Data Query Language (DQL)

1.) What is the SELECT statement, and how is it used to query data?

Answer :

The SELECT statement is one of the most fundamental and widely used commands in SQL. It is used to query and retrieve data from one or more tables in a database.

2.) Explain the use of the ORDER BY and WHERE clauses in SQL queries.

Answer :

(1) WHERE Clauses :

WHERE clause is used to filter records and extract only those that meet a specified condition.

It restricts the rows returned by the query to only those that fulfill the criteria set in the condition.

It can be combined with operators such as =, >, <, >=, <=, <>, BETWEEN, LIKE, and IN.

(2) ORDER BY :

ORDER clause is used to sort the result set of a query in either ascending or descending order based on one or more columns.

By default, the sorting order is ascending (ASC). To sort in descending order, the DESC keyword is used.

Multiple columns can be sorted by listing them in the ORDER BY clause, separated by commas.

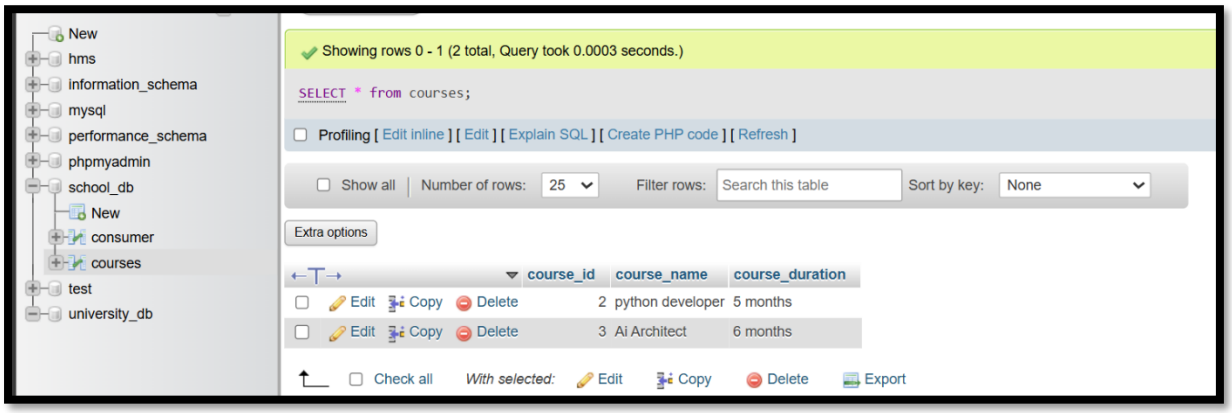
LAB EXERCISES :

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

Answer :

SELECT * from courses;

Output :

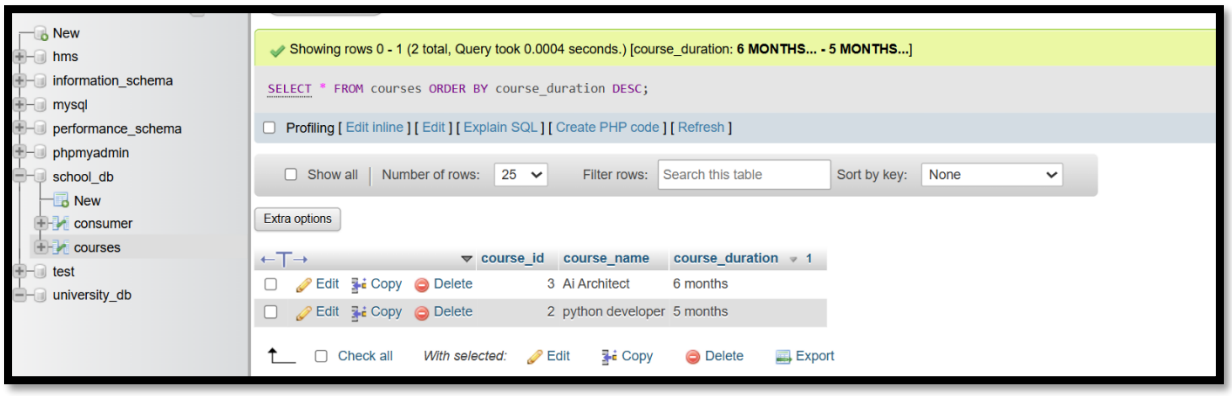


Lab 2: Sort the courses based on course_duration in descending order using ORDER BY.

Answer :

SELECT * FROM courses ORDER BY course_duration DESC;

Output :



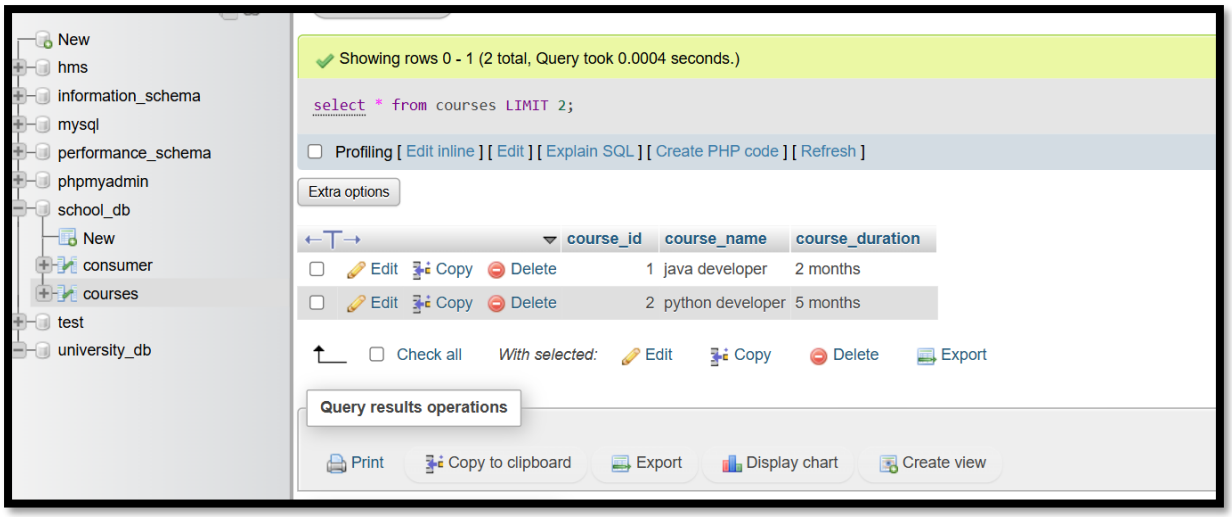
Lab 3: Limit the results of the SELECT query to show only the top two courses using LIMIT.

Answer :

select * from courses

LIMIT 2;

Output :



9.) Data Control Language (DCL)

1.) What is the purpose of GRANT and REVOKE in SQL?

Answer :

GRANT Command:

Purpose: The GRANT command is used to give specific privileges to users or roles on database objects like tables, views, or procedures.

REVOKE Command:

Purpose: The REVOKE command is used to remove previously granted privileges from users or roles on database objects.

2.) How do you manage privileges using these commands?

Answer :

Managing privileges using the **GRANT and REVOKE** commands is essential for maintaining database security and access control. The **GRANT** command is used to assign specific privileges to users or roles, determining what actions they can perform on database objects like tables, views, or procedures. These privileges can include operations such as selecting data, inserting new records, updating existing records, and deleting records.

The **REVOKE** command is used to withdraw previously granted privileges, ensuring that users or roles no longer have access to perform certain actions on database objects. This command helps in tightening security and controlling access, especially when a user's role changes or when they no longer require certain permissions.

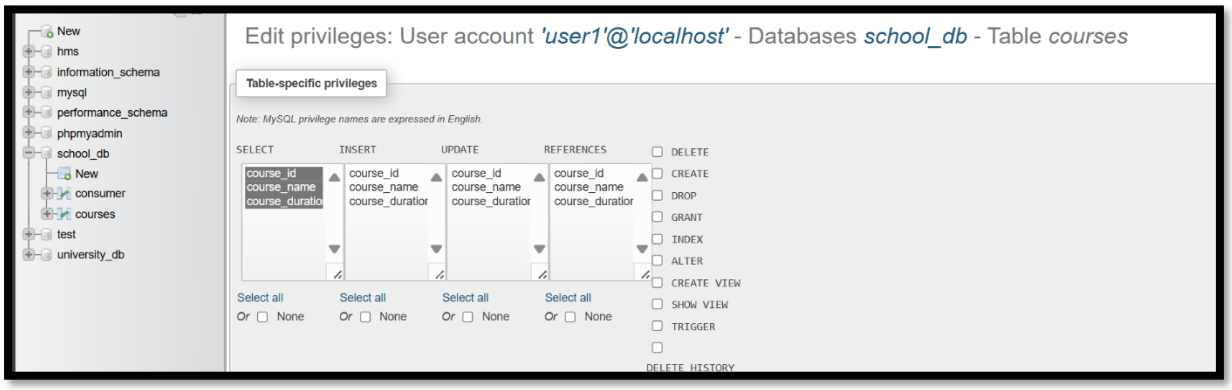
LAB EXERCISES :

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

Answer :

```
CREATE USER user1 IDENTIFIED BY 'password1';

CREATE USER user2 IDENTIFIED BY 'password2';


GRANT SELECT ON courses TO user1;
```

Output :

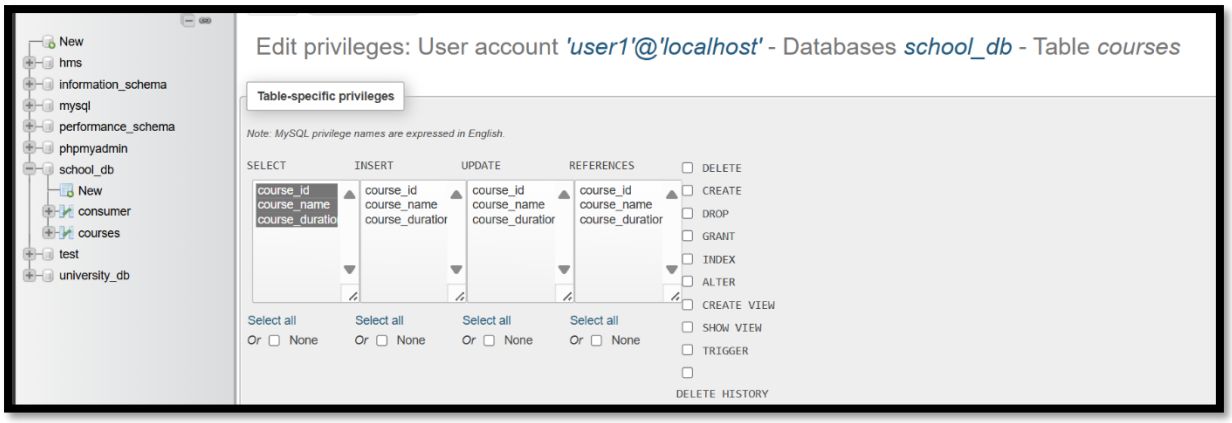


Lab 2: Revoke the INSERT permission from user1 and give it to user2.

Answer :

```
REVOKE INSERT ON courses FROM 'user1'@'localhost';

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

Output :



10.) Transaction Control Language (TCL)

1.) What is the purpose of the COMMIT and ROLLBACK commands in SQL?

Answer :

Purpose of COMMIT and ROLLBACK Commands in SQL:

COMMIT Command :

- **Purpose:** Saves all changes made during the current transaction permanently in the database. Ensures data integrity by marking the successful end of a transaction.

ROLLBACK Command :

- **Purpose:** Reverts all changes made during the current transaction, restoring the database to its previous state. Ensures data integrity by undoing partial or faulty transactions.

2.) Explain how transactions are managed in SQL databases.

Answer : Transactions are managed in SQL database.

- Transactions are units that contain work for the database.
- You start a transaction, define the work the database needs to do, and then finish by committing or rolling back the changes.
- Transactions follow the principles of Atomicity, Consistency, Isolation, and Durability (ACID).
- If a transaction encounters errors and must be canceled or rolled back, all data modifications are erased.

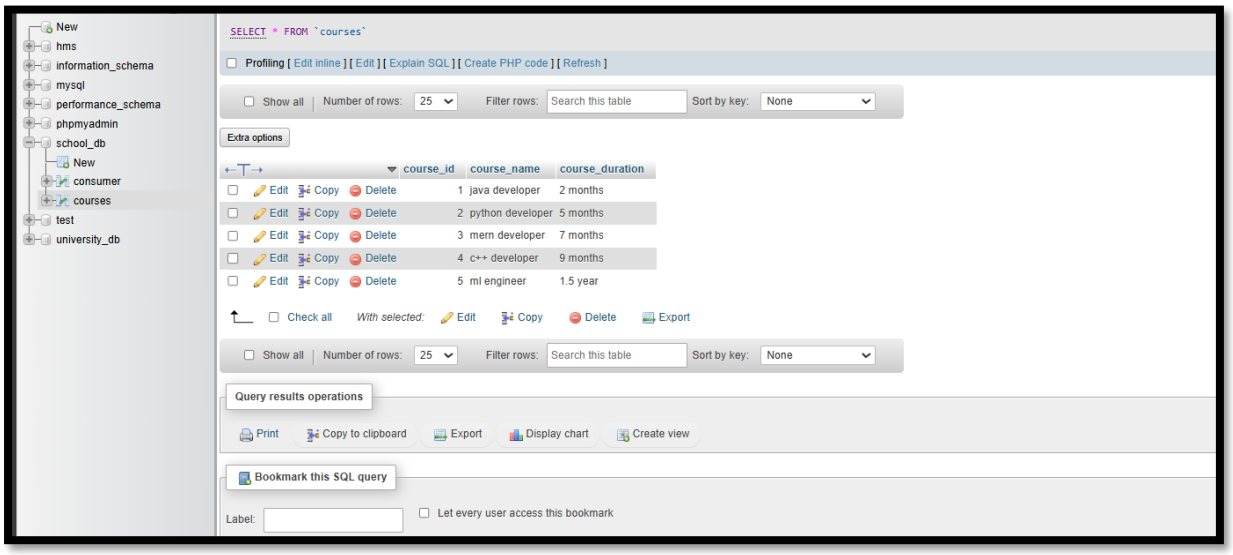
LAB EXERCISES :

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

Answer :

```
insert into courses values (5,"ml engineer","1.3 year");

commit;
```

Output :

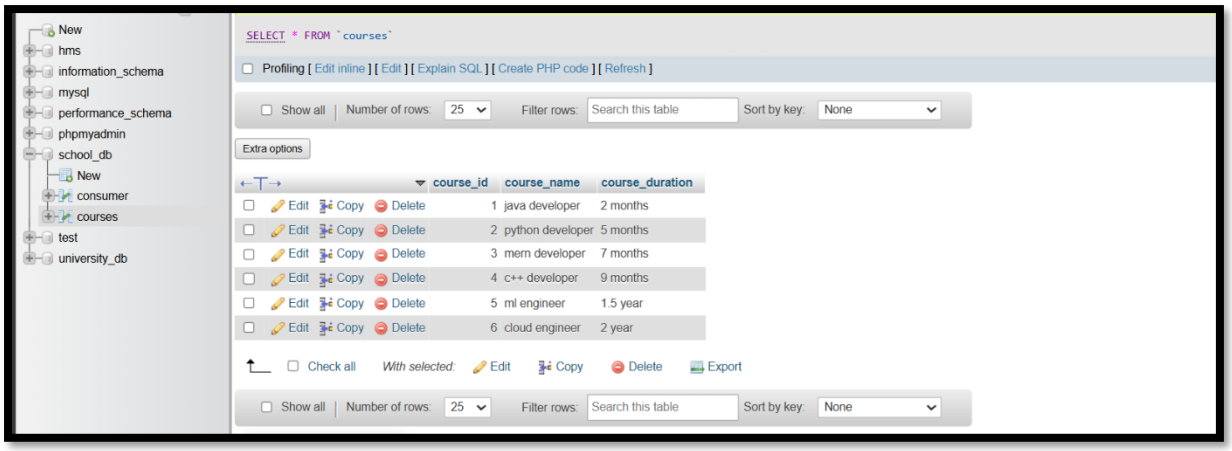


Lab 2: Insert additional rows, then use ROLLBACK to undo the last insert operation.

Answer :

```
insert into courses VALUES(6,"cloud engineer","2 year");

Rollback;
```

Output :



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

Answer :

START TRANSACTION;

SAVEPOINT before_update;

UPDATE courses SET course_name = 'Updated SQL Basics' WHERE course_id = 1;

UPDATE courses SET course_name = 'Updated Advanced SQL' WHERE course_id = 2;

ROLLBACK TO before_update;

COMMIT;

Output :

11.) SQL Joins

1.) Explain the concept of JOIN in SQL. What is the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN?

Answer : A JOIN used to combine rows from two or more tables based on a related column between them. It allows querying data from multiple tables as if it were a single table, which is essential for relational databases.

Types of JOIN :

1. INNER JOIN:

- Retrieves records that have matching values in both tables.

- Only rows with matching keys in both tables are included in the result set.

2. LEFT JOIN (or LEFT OUTER JOIN):

- Retrieves all records from the left table and the matched records from the right table.
- If there is no match, the result is NULL on the side of the right table.

3. RIGHT JOIN (or RIGHT OUTER JOIN):

- Retrieves all records from the right table and the matched records from the left table.
- If there is no match, the result is NULL on the side of the left table.

4. FULL OUTER JOIN:

- Retrieves all records when there is a match in either left or right table.
- Combines the result of both LEFT JOIN and RIGHT JOIN, showing NULLs where there are no matches.

2.) How are joins used to combine data from multiple tables?

Answer :

Joins are SQL that allow you to combine data from two or more tables based on related columns. They enable you to query and retrieve data as if it were stored in a single table, which is crucial for relational database management.

LAB EXERCISES :

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

Answer :

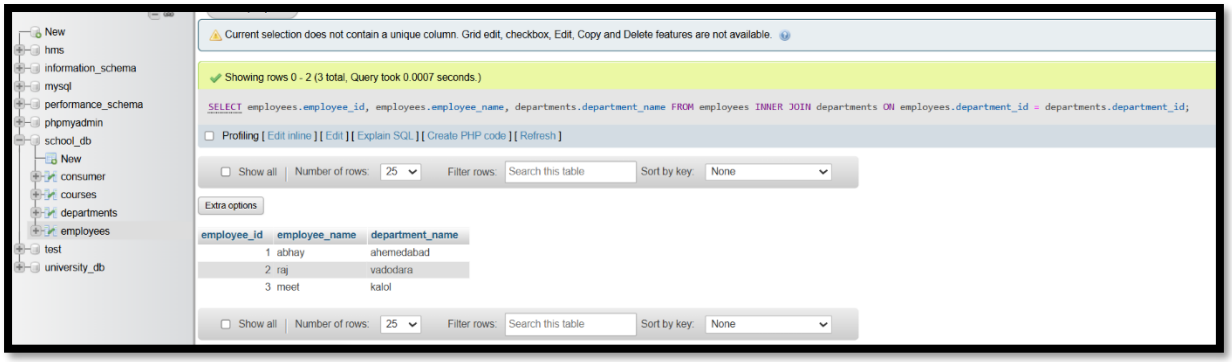
```
CREATE TABLE departments
(
   department_id INT PRIMARY KEY,
   department_name VARCHAR(50) NOT NULL
);
CREATE TABLE employees
 (
   employee_id INT PRIMARY KEY,
   employee_name VARCHAR(50) NOT NULL,
   department_id INT,
```

FOREIGN KEY (department_id) REFERENCES departments(department_id)

);

SELECT employees.employee_id, employees.employee_name,departments.department_name FROM employees INNER JOIN departments ON employees.department_id=departments.department_id;

Output :



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

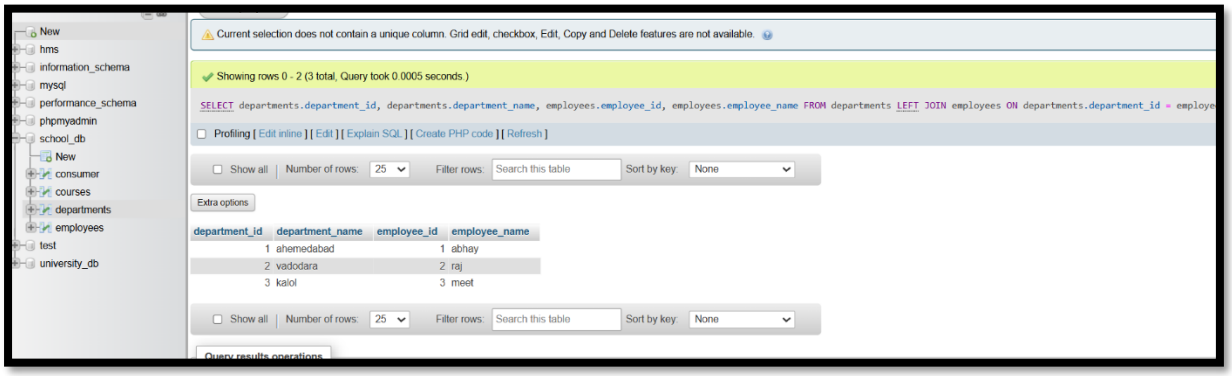
Answer :

SELECT departments.department_id, departments.department_name, employees.employee_id, employees.employee_name

FROM departments

LEFT JOIN employees ON departments.department_id = employees.department_id;

Output :



12.) SQL Group By

1.) What is the GROUP BY clause in SQL? How is it used with aggregate functions?

Answer : The Group clause in SQL is used to arrange identical data into groups. This is particularly useful for aggregate functions, which perform a calculation on a set of values and return a single value.

Usage with Aggregate Functions:

- Aggregation: The GROUP BY clause works alongside aggregate functions such as COUNT, SUM, AVG,

MAX, and MIN to perform calculations on each group of data.

- Data Grouping: It groups the rows that have the same values in specified columns into summary rows, like finding the total sales per region.
- Result Customization: By using the GROUP BY clause, you can generate reports and summarize large datasets effectively.

2.) Explain the difference between GROUP BY and ORDER BY.

Answer :

GROUP BY	ORDER BY
Group by statement is used to group the rows that have the same value.	Whereas Order by statement sort the result-set either in ascending or descending order.
It may be allowed in CREATE VIEW statement.	While it does not use in CREATE VIEW statement.
In select statements, it is always used before the order by keyword.	While in the select statement, it is always used after the group by keyword.

An attribute cannot be in the group by a statement under the aggregate function.	Whereas in order by statement, the attribute can be under aggregate function.
Group by controls the presentation of tuples(rows).	While order by clause controls the presentation of columns.
In group by clause, the tuples are grouped based on the similarity between the attribute values of tuples.	Whereas in order by clause, the result set is sorted based on ascending or descending order.

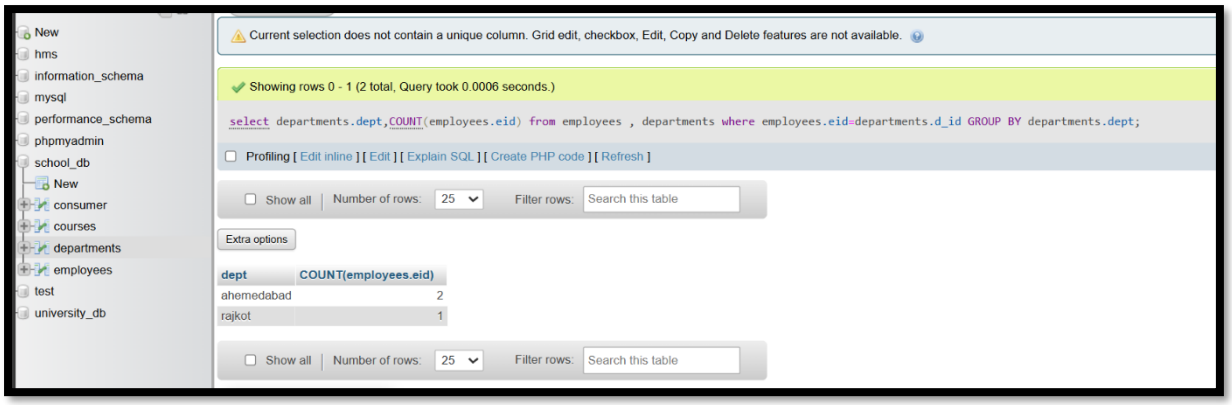
LAB EXERCISES :

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

Answer :

select departments.dept,COUNT(employees.eid) from employees , departments where employees.eid=departments.d_id GROUP BY departments.dept;

Output :

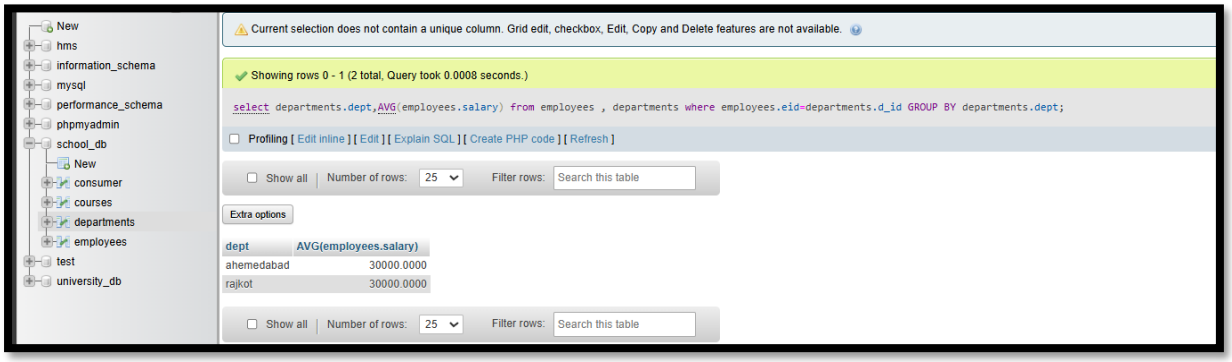


Lab 2: Use the AVG aggregate function to find the average salary of employees in each department.

Answer :

```
select departments.dept,AVG(employees.salary) from e
mployees , departments where employees.eid=depart
ments.d_id GROUP BY departments.dept;
```

Output :



13.) SQL Stored Procedure

1.) What is a stored procedure in SQL, and how does it differ from a standard SQL query?

Answer :

A stored procedure in SQL is a precompiled set of SQL statements that can be executed as a single unit. Stored procedures are used to encapsulate repetitive or complex operations, providing a structured way to manage database tasks.

how does it differ from a standard SQL query? short form to write

Standard SQL Query:

- Single statement.
- Executed each time.
- Not stored.
- Less secure.

2.) Explain the advantages of using stored procedures.

Answer :

Advantages of using stored Procedures :

Performance : Precompiled, reducing execution time.

Reusability : Can be reused across multiple applications.

Security : Enhances data security and prevents SQL injection.

Maintainability : Easier to manage and update business logic.

Modularity : Breaks down complex tasks into manageable units.

Efficiency : Reduces network traffic by executing multiple statements in one call.

Consistency : Ensures consistent application of business rules.

LAB EXERCISES :

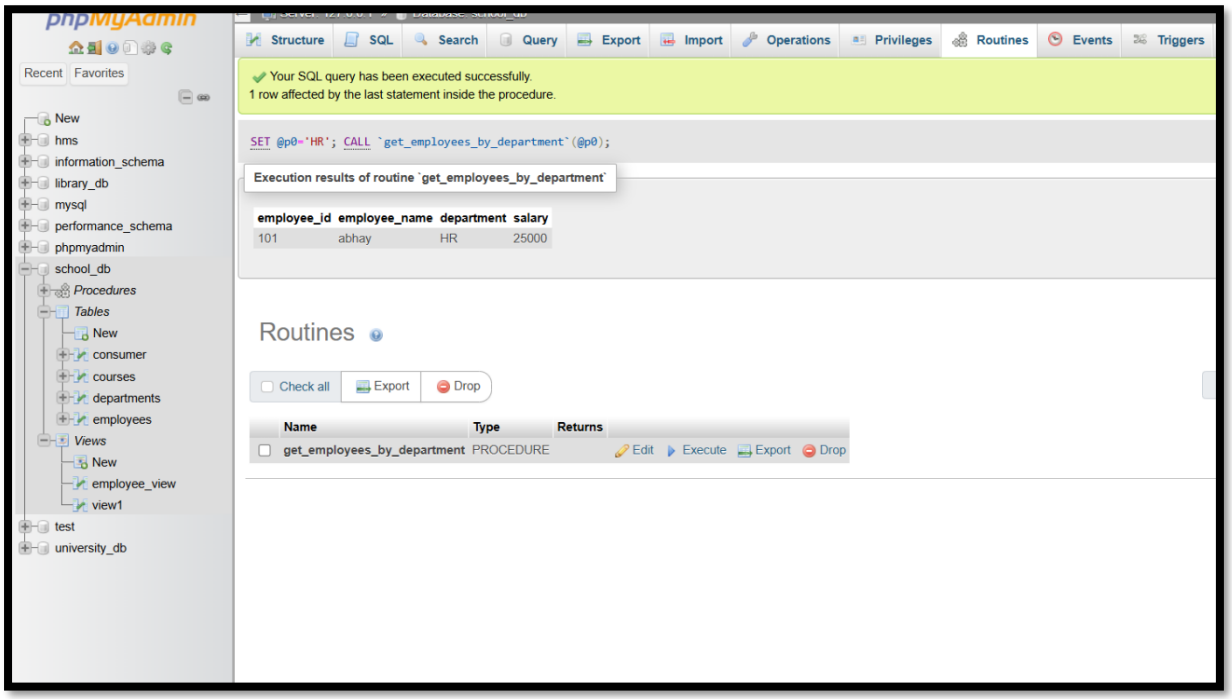
Lab 1: Write a stored procedure to retrieve all employees from the employees table based on department.

Answer :

```
DELIMITER //

CREATE PROCEDURE get_employees_by_department(IN dept_name VARCHAR(100))
```

```
BEGIN
    DROP TEMPORARY TABLE IF EXISTS temp_employees;
    CREATE TEMPORARY TABLE temp_employees AS
    SELECT employee_id, employee_name, department, salary
    FROM employees
    WHERE department = dept_name;

    SELECT * FROM temp_employees;
END //
DELIMITER ;
```

Output :

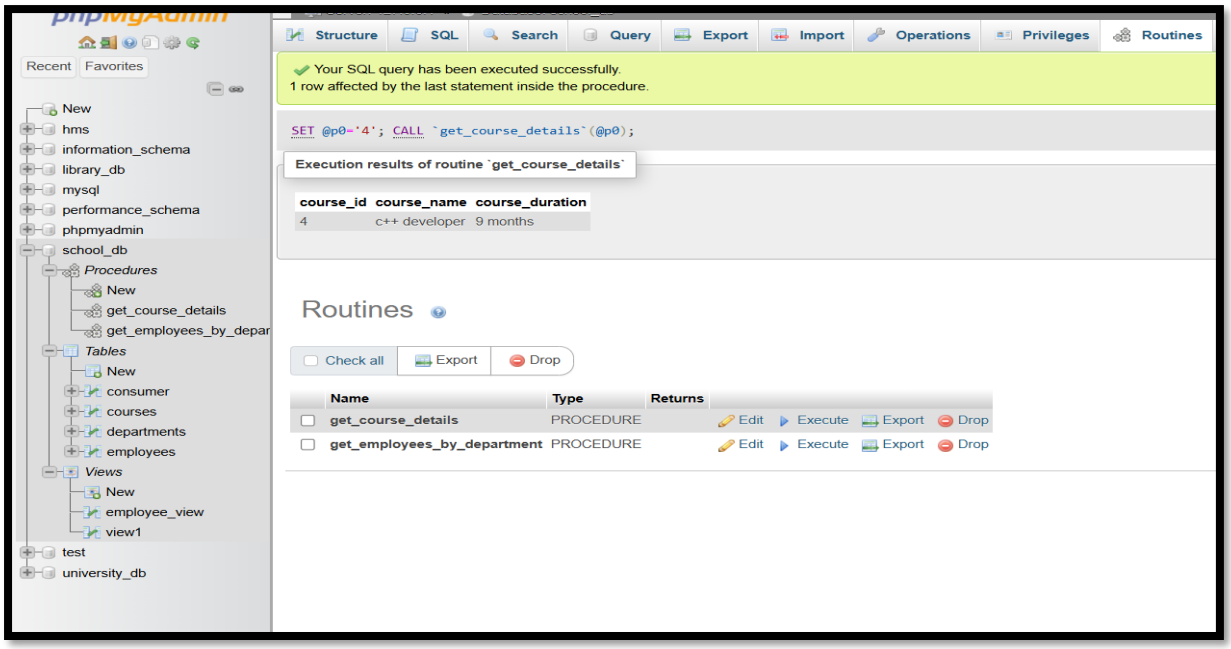


Lab 2: Write a stored procedure that accepts course_id as input and returns the course details.

Answer :

```
DELIMITER //
CREATE PROCEDURE get_course_details(IN coursee_id INT)
BEGIN
    SELECT course_id, course_name, course_duration
    FROM courses
    WHERE course_id = coursee_id;
END //
DELIMITER ;
```

Output :



14.) SQL View

1.) What is a view in SQL, and how is it different from a table?

Answer :

View in SQL

A view in SQL is a virtual table that provides a way to simplify complex queries by representing the result set of a SELECT query as a table. Unlike physical tables, views do not store data permanently. Instead, they

dynamically generate results from the underlying tables whenever they are queried.

Difference from Tables:

1. **Data Storage**:
    - **Tables**: Store data physically in the database.
    - **Views**: Do not store data physically. They generate data dynamically from underlying tables.
2. **Persistence**:
    - **Tables**: Data in tables is persisted and physically stored.
    - **Views**: Views are virtual and do not store data. They provide a dynamic window into the data based on the defined query.
3. **Creation**:
    - **Tables**: Created to store data directly.
    - **Views**: Created to simplify query complexity and improve data abstraction.
4. **Maintenance**:
    - **Tables**: Require direct data management and maintenance.

- **Views**: Automatically reflect changes in the underlying tables, reducing maintenance effort.

2.) Explain the advantages of using views in SQL databases.

Answer :

Advantages of Using Views in SQL Databases

1. Simplification of Complex Queries
2. Data Abstraction
3. Enhanced Security
4. Data Consistency and Integrity
5. Reusability
6. Maintenance
7. Improved Performance

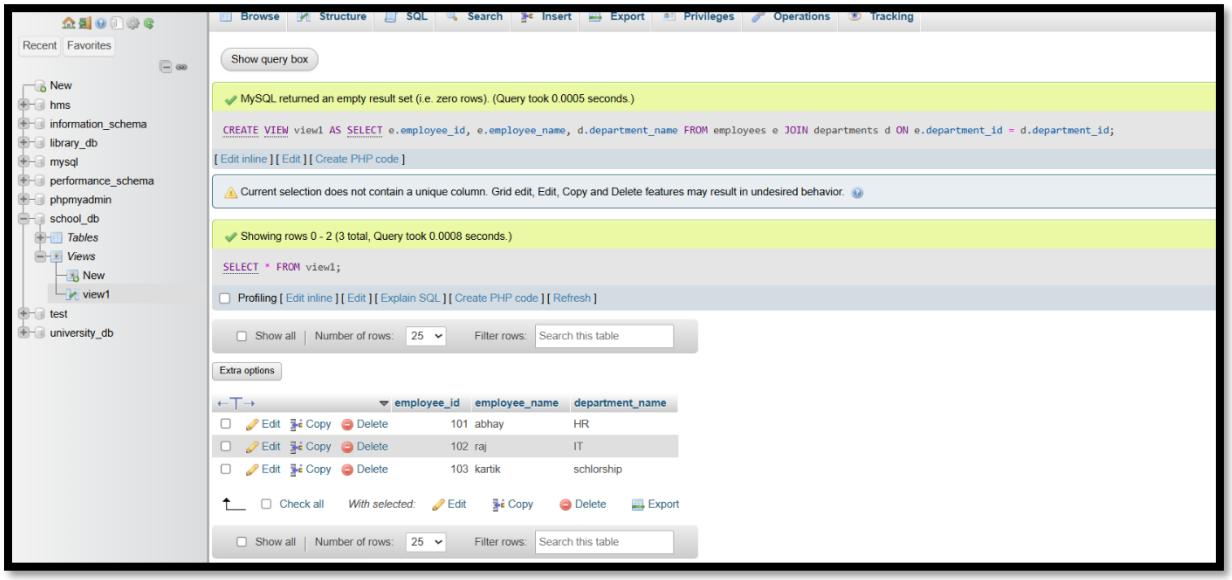
LAB EXERCISES :

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

Answer :

CREATE VIEW view1 AS SELECT e.employee_id, e.employee_name, d.department_name FROM employees e JOIN departments d ON e.department_id = d.department_id;

Output :



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

Answer :

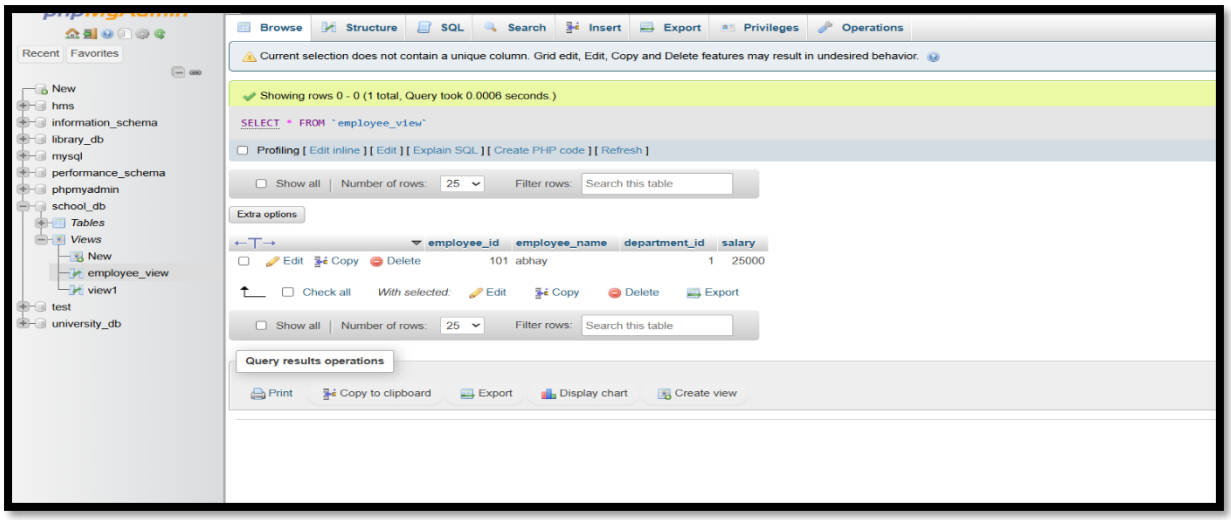
CREATE OR REPLACE VIEW employee_view AS

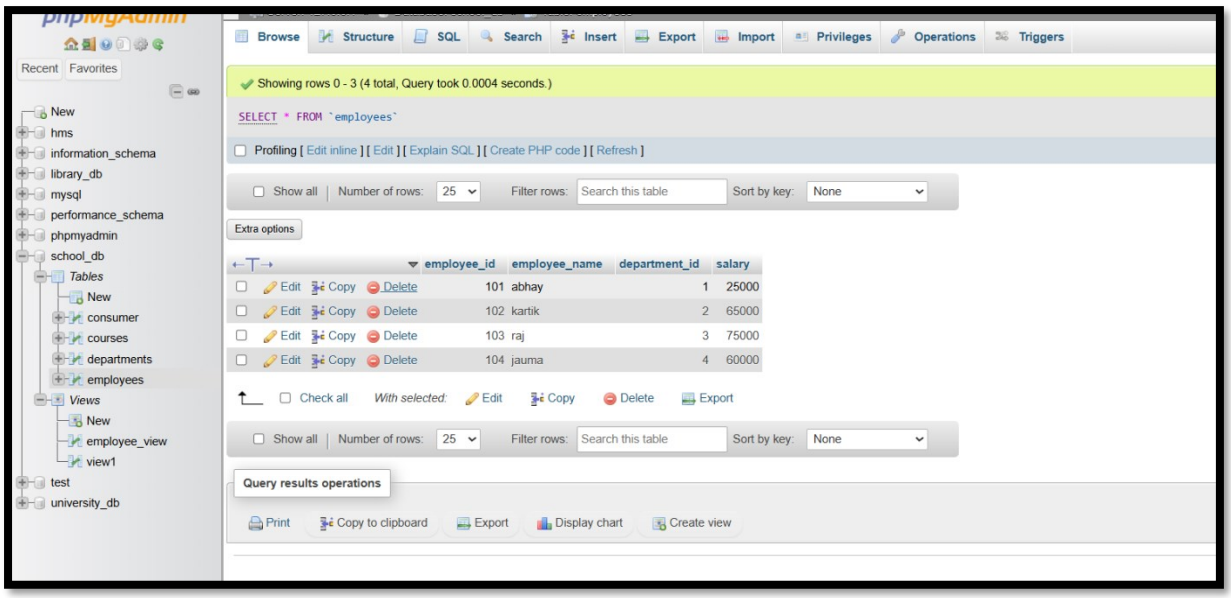
SELECT employee_id, employee_name, salary

FROM employees

WHERE salary <= 50000;

Output :





15.) SQL Triggers

1.) What is a trigger in SQL? Describe its types and when they are used ?

Answer :

Trigger is a statement that a system executes automatically when there is any modification to the database.

Types of Triggers :

1. **AFTER INSERT**: activated after data is inserted into the table.
2. **AFTER UPDATE:** activated after data in the table is modified.
3. **AFTER DELETE:** activated after data is deleted/removed from the table.
4. **BEFORE INSERT:** activated before data is inserted into the table.
5. **BEFORE UPDATE:** activated before data in the table is modified.
6. **BEFORE DELETE:** activated before data is deleted/removed from the table.

Triggers are used to specify certain integrity constraints and referential constraints that cannot be specified using the constraint mechanism of SQL.

2.) Explain the difference between INSERT, UPDATE, and DELETE triggers ?

Answer :

INSERT Trigger :

- An INSERT trigger activates when a new row is added to a table. It allows for automated actions or enforcement of business rules at the moment data is inserted.
- Commonly used for logging insertions, initializing values, or ensuring data integrity by validating or modifying the data before it is added to the table.

UPDATE Trigger :

- An UPDATE trigger is fired whenever an existing row in a table is modified. It enables the execution of specific actions or validation checks when data is updated.
- Often used to audit changes, maintain data consistency, enforce business rules, or trigger updates in related tables whenever a table's data is altered.

DELETE Trigger:

- A DELETE trigger is executed when a row is removed from a table. It allows for actions or constraints to be enforced when data is deleted.
- Frequently used to maintain referential integrity by cascading deletes, log deletion activities, or prevent deletions based on certain conditions.

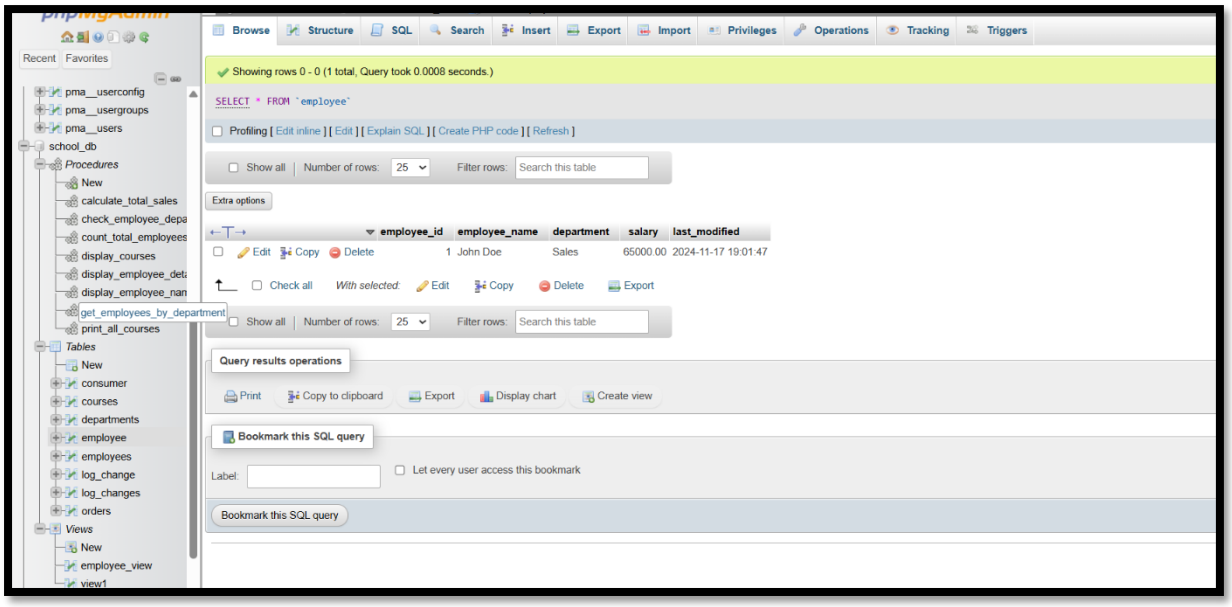
LAB EXERCISES :

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

Answer :

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

Output :



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

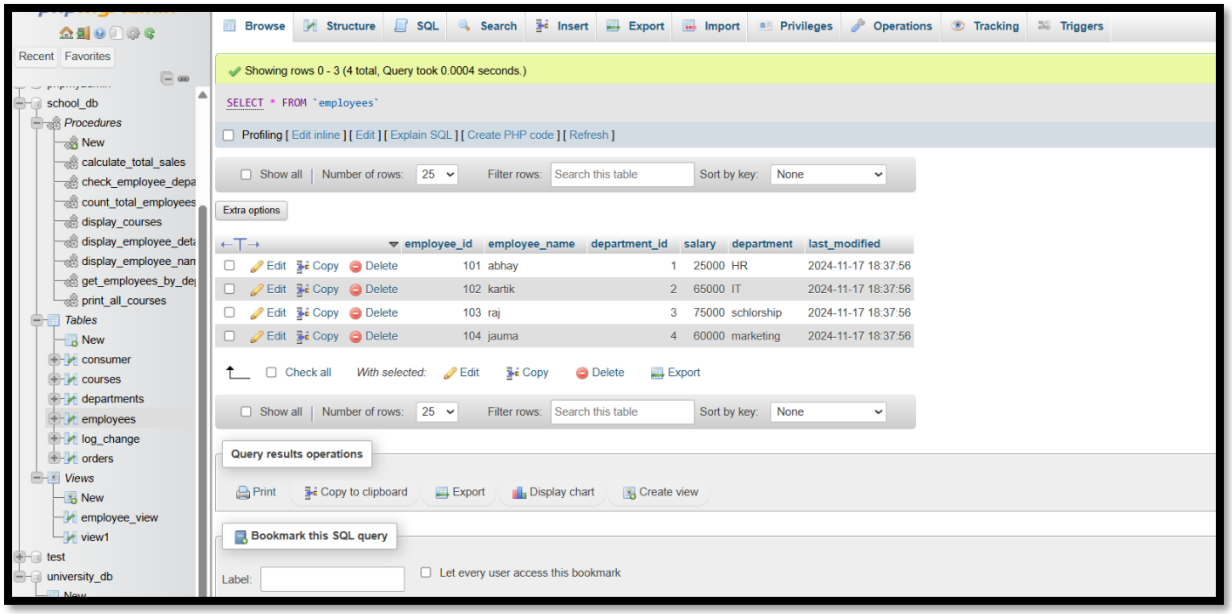
Answer :

```
ALTER TABLE employees

ADD COLUMN last_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP;

DELIMITER //

CREATE TRIGGER update_last_modified

BEFORE UPDATE ON employees

FOR EACH ROW

BEGIN

    SET NEW.last_modified = CURRENT_TIMESTAMP;
```

END //

DELIMITER ;

Output :



16.) Introduction PL/SQL

1.) What is PL/SQL, and how does it extend SQL's capabilities?

Answer :

PL/SQL (Procedural Language/Structured Query Language) extends SQL by integrating procedural programming features such as variables, loops, and

error handling. It enables more complex operations within the database, supports triggers and stored procedures, and improves performance by bundling multiple SQL statements into a single execution.

Key Benefits:

- Procedural Constructs : Adds programming capabilities to SQL.
- Error Handling : Advanced error management.
- Performance : Reduces communication overhead.
- Security : Enhances control and integrity.

2.) List and explain the benefits of using PL/SQL.

Answer :

1. High-Level of Execution
2. Increased Performance
3. Compatibility with SQL
4. Supports Portability
5. Better Productivity
6. Provides Diverse Support
7. Time-Saver

8. Integration with Other Programming Languages

9. Error Management

10. Reusability of Code

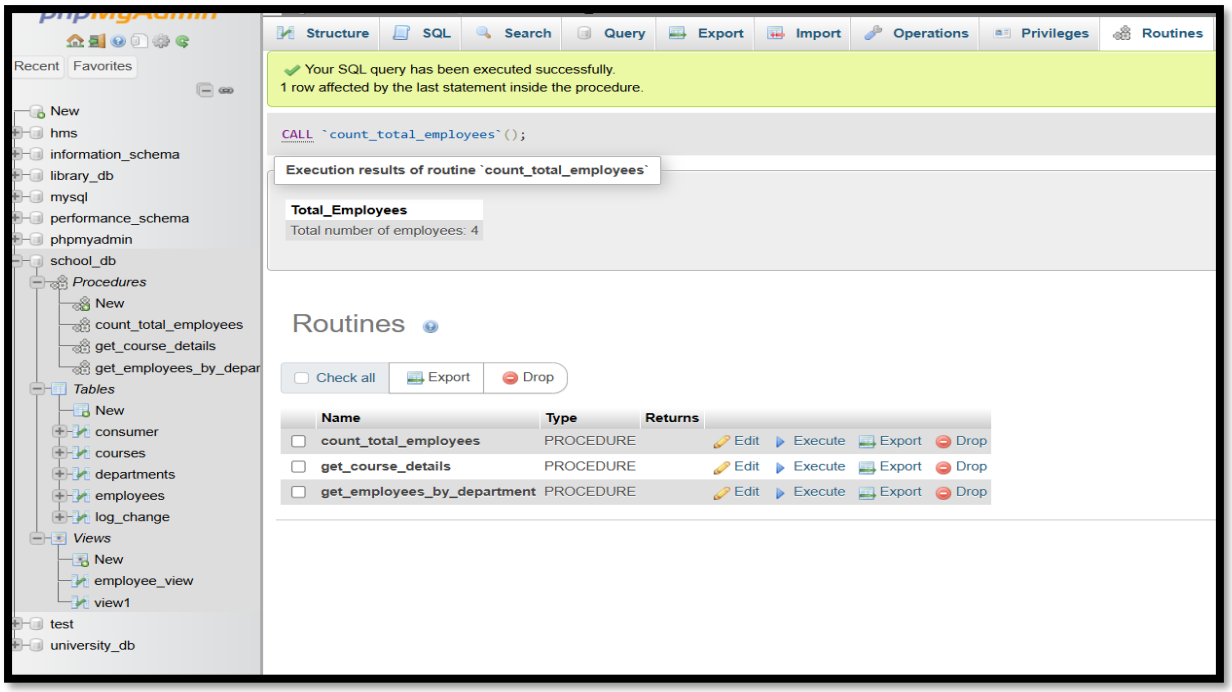
LAB EXERCISES :

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

Answer :

```
DELIMITER //
CREATE PROCEDURE count_total_employees()
BEGIN
   DECLARE v_total_employees INT;
   SELECT COUNT(employee_id) INTO v_total_employees FROM employees;
   SELECT CONCAT('Total number of employees: ', v_total_employees) AS Total_Employees;
END //
DELIMITER ;
```

Output :



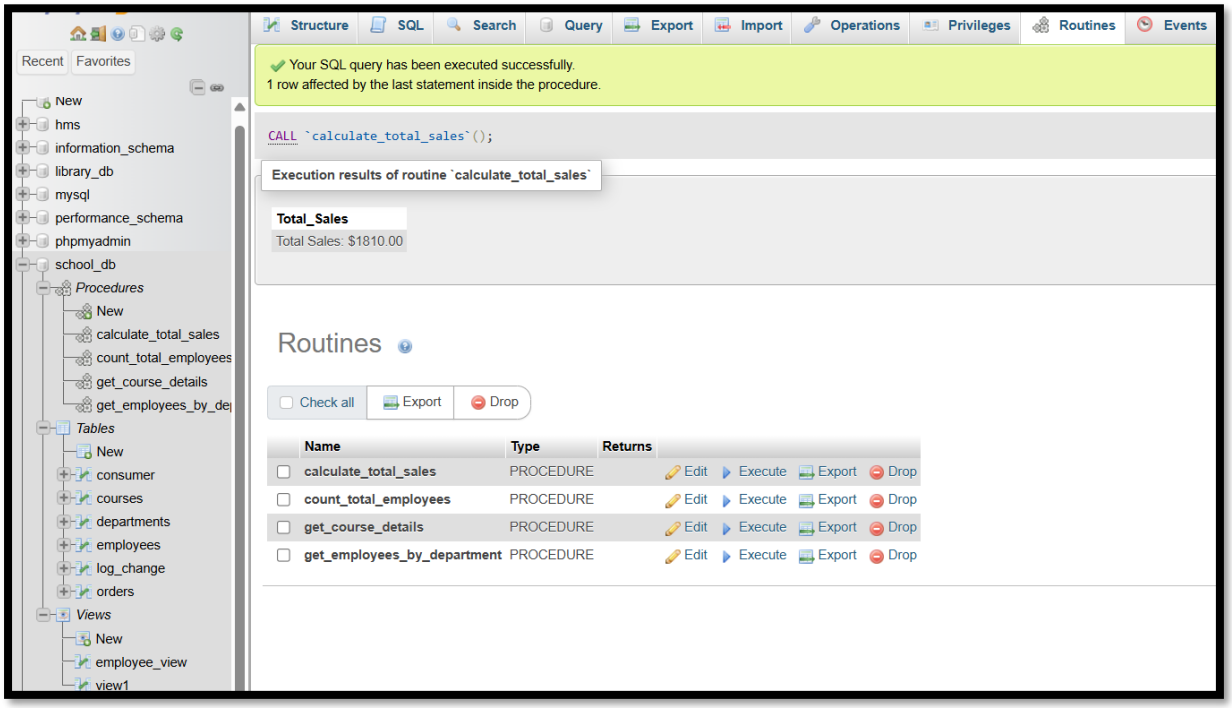
Lab 2: Create a PL/SQL block that calculates the total sales from an orders table.

Answer :

```
DELIMITER //

CREATE PROCEDURE calculate_total_sales()

BEGIN

    DECLARE v_total_sales DECIMAL(10, 2);

    SELECT SUM(order_amount) INTO v_total_sales FROM orders;

    SELECT CONCAT('Total Sales: $', v_total_sales) AS Total_Sales;

END //
```

DELIMITER ;

Output :



17.) PL/SQL Control Structures

1.) What are control structures in PL/SQL? Explain the IF-THEN and LOOP control structures ?

Answer :

Control structures in PL/SQL enable developers to dictate the flow of execution within their programs. These constructs allow for decision-making, iteration, and branching, making PL/SQL a powerful and flexible programming language.

IF-THEN Control Structure:

- The IF-THEN structure is used for conditional execution of statements. It allows a program to choose different paths of execution based on the evaluation of a condition.

How It Works:

- The condition is evaluated, and if it is true, the subsequent statements within THEN block are executed. If the condition is false, the statements are skipped.

LOOP Control Structure:

- The LOOP structure allows for repeated execution of a block of statements. This is useful for performing iterative tasks.

Types:

- Basic LOOP: Repeats the block of code indefinitely until an exit condition is met.

- WHILE LOOP: Repeats the block of code as long as a specified condition remains true.
- FOR LOOP: Repeats the block of code a specific number of times, iterating over a range of values.

2.) How do control structures in PL/SQL help in writing complex queries?

Answer :

Control structures in PL/SQL provide the ability to introduce logic and flow control into SQL queries, enabling the creation of more sophisticated and dynamic database operations.

Conditional Execution:

- **IF-THEN-ELSE Statements**:
    - Allow for conditional logic within queries. This means actions can be taken based on specific conditions, enabling complex decision-making processes within SQL.

Iterative Processing:

- **LOOP Constructs**:
    - Enable repetitive execution of a block of statements. This is crucial for performing operations that need to be repeated multiple

times, such as processing each row of a result set individually.

Exception Handling:

- **EXCEPTION Blocks**:
  - Provide a robust mechanism to handle errors and exceptions. This ensures that the application can gracefully handle unexpected situations and maintain data integrity.

Procedural Logic:

- **Procedural Blocks**:
  - PL/SQL allows the use of procedural logic with the integration of loops, conditionals, and nested blocks. This modular approach helps in organizing code into understandable and maintainable units.

LAB EXERCISES :

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

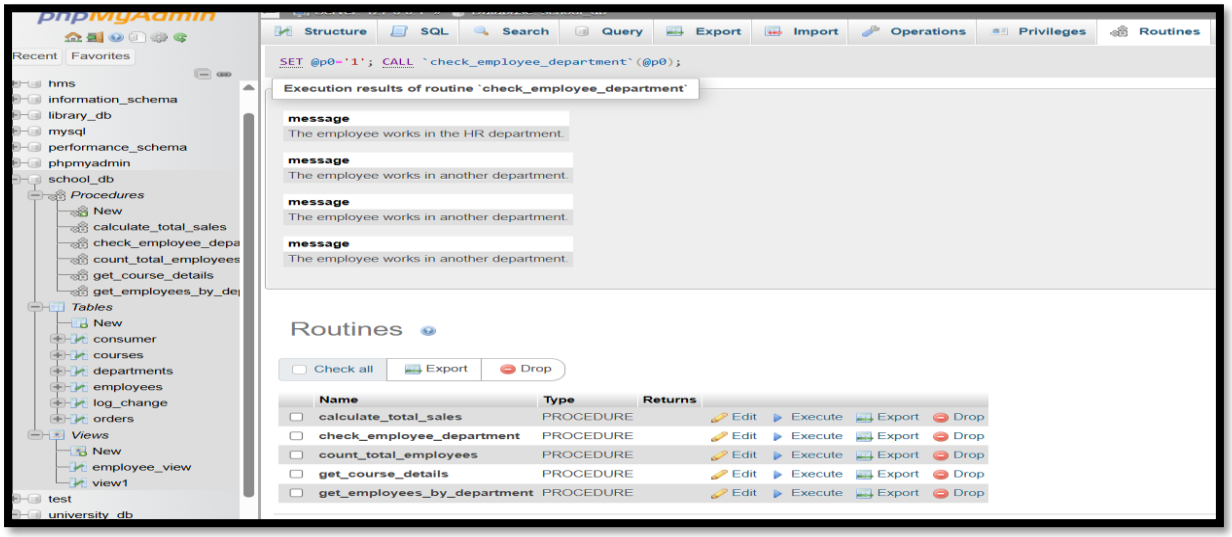
Answer :

```
DELIMITER //
```

```
CREATE PROCEDURE check_employee_department(IN employee_id INT)
BEGIN
    DECLARE v_department VARCHAR(100);
    DECLARE done INT DEFAULT FALSE;
    DECLARE cur CURSOR FOR
        SELECT department FROM employees WHERE employee_id = employee_id;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO v_department;
        IF done THEN
            LEAVE read_loop;
        END IF;
        IF v_department = 'Sales' THEN
            SELECT 'The employee works in the Sales department.' AS message;
        ELSEIF v_department = 'HR' THEN
```

```
        SELECT 'The employee works in the HR department.' AS message;
    ELSE
        SELECT 'The employee works in another department.' AS message;
    END IF;
  END LOOP;
  CLOSE cur;
END //
DELIMITER ;
```

Output :



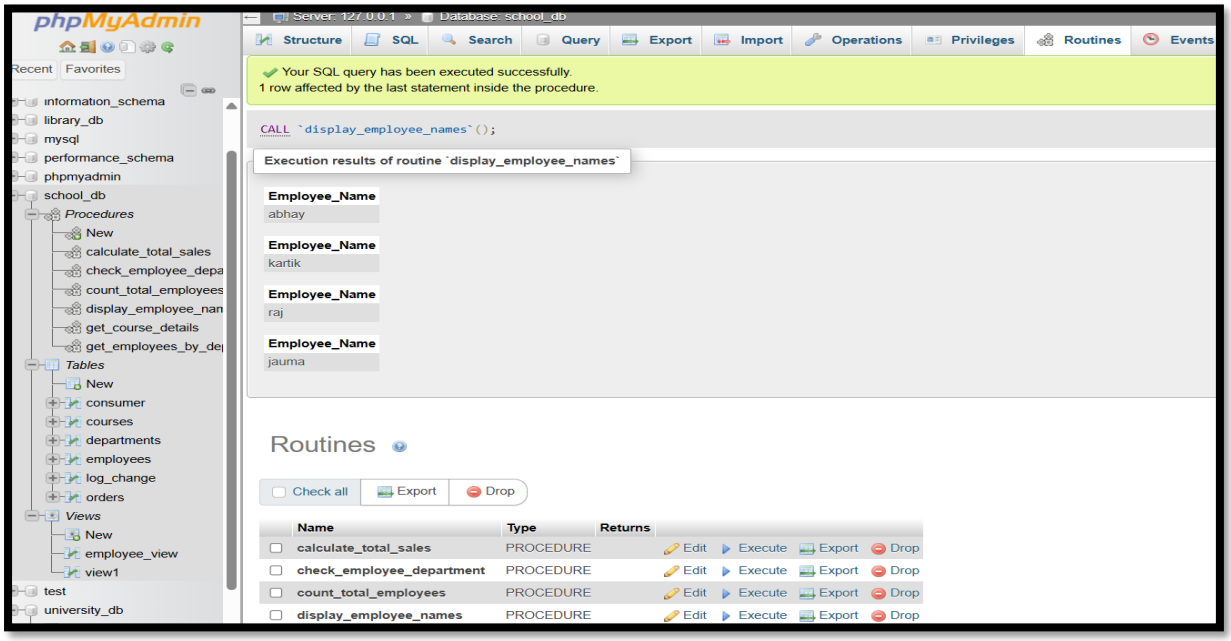
Lab 2: Use a FOR LOOP to iterate through employee records and display their names.

Answer :

```
DELIMITER //
CREATE PROCEDURE display_employee_names()
BEGIN
    DECLARE v_employee_name VARCHAR(100);
    DECLARE done INT DEFAULT 0;

    DECLARE emp_cursor CURSOR FOR
        SELECT employee_name FROM employees;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET
done = 1;

    OPEN emp_cursor;

    read_loop: LOOP
        FETCH emp_cursor INTO v_employee_name;
        IF done THEN
            LEAVE read_loop;
        END IF;
```

```
    SELECT v_employee_name AS Employee_Name;
  END LOOP;

  CLOSE emp_cursor;
END //
DELIMITER ;
```

Output :



18.) SQL Cursors

1.) What is a cursor in PL/SQL? Explain the difference between implicit and explicit cursors.

Answer :

cursors are essential tools that allow developers to retrieve and manipulate data row by row.

NO.	Implicit Cursors	Explicit Cursors
1	Implicit cursors are automatically created when select statements are executed.	Explicit cursors needs to be defined explicitly by the user by providing a name.
2	They are capable of fetching a single row at a time.	Explicit cursors can fetch multiple rows.
3	Closes automatically after execution.	Need to close after execution.
4	They are more vulnerable to errors such as Data errors, etc.	They are less vulnerable to errors.
5	Provides less programmatic control to the users	User/Programmer has the entire control.
6	Implicit cursors requires anonymous buffer memory for storage purpose.	Explicit cursors use user-defined memory space for storage purpose

2.) When would you use an explicit cursor over an implicit one?

Answer :

Explicit Cursors :

- Use explicit cursors when you need detailed control over the processing of multi-row queries. They allow you to fetch and manipulate each row individually, making them ideal for complex data manipulation and iterative processing tasks.
- They are preferable in scenarios where you need to handle large result sets, perform row-by-row operations, or implement custom logic for each row fetched from a query.

Implicit Cursors :

- Implicit cursors are suitable for simple, single-row queries or straightforward multi-row operations where detailed control over each row is not necessary. They are automatically managed by the PL/SQL runtime engine, making them easier and less verbose to use for basic operations.

Explicit cursors provide more flexibility and control, making them the better choice for complex, multi-row operations in PL/SQL.

LAB EXERCISES :

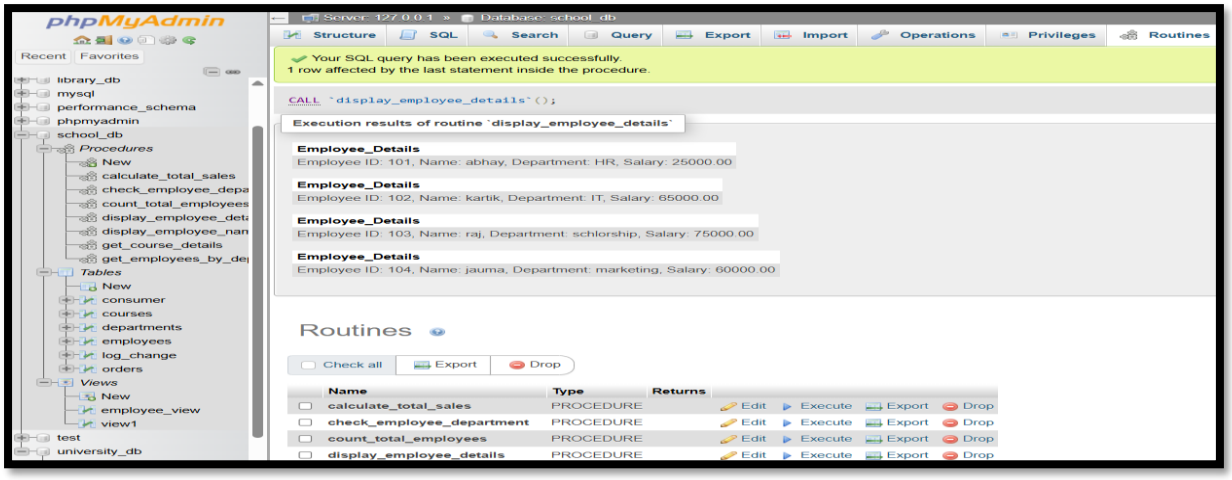
Lab 1: Write a PL/SQL block using an explicit cursor to retrieve and display employee details.

Answer :

```
DELIMITER //
CREATE PROCEDURE display_employee_details()
BEGIN
    DECLARE v_employee_id INT;
    DECLARE v_employee_name VARCHAR(100);
    DECLARE v_department VARCHAR(100);
    DECLARE v_salary DECIMAL(10, 2);
    DECLARE done INT DEFAULT FALSE;
    DECLARE emp_cursor CURSOR FOR
        SELECT employee_id, employee_name, department, salary FROM employees;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN emp_cursor;
    read_loop: LOOP
```

```
        FETCH emp_cursor INTO v_employee_id, v_employee_name, v_department, v_salary;

        IF done THEN

            LEAVE read_loop;

        END IF;

        SELECT CONCAT('Employee ID: ', v_employee_id, ', Name: ', v_employee_name, ', Department: ', v_department, ', Salary: ', v_salary) AS Employee_Details;

    END LOOP;


    CLOSE emp_cursor;

END //

DELIMITER ;
```

Output :



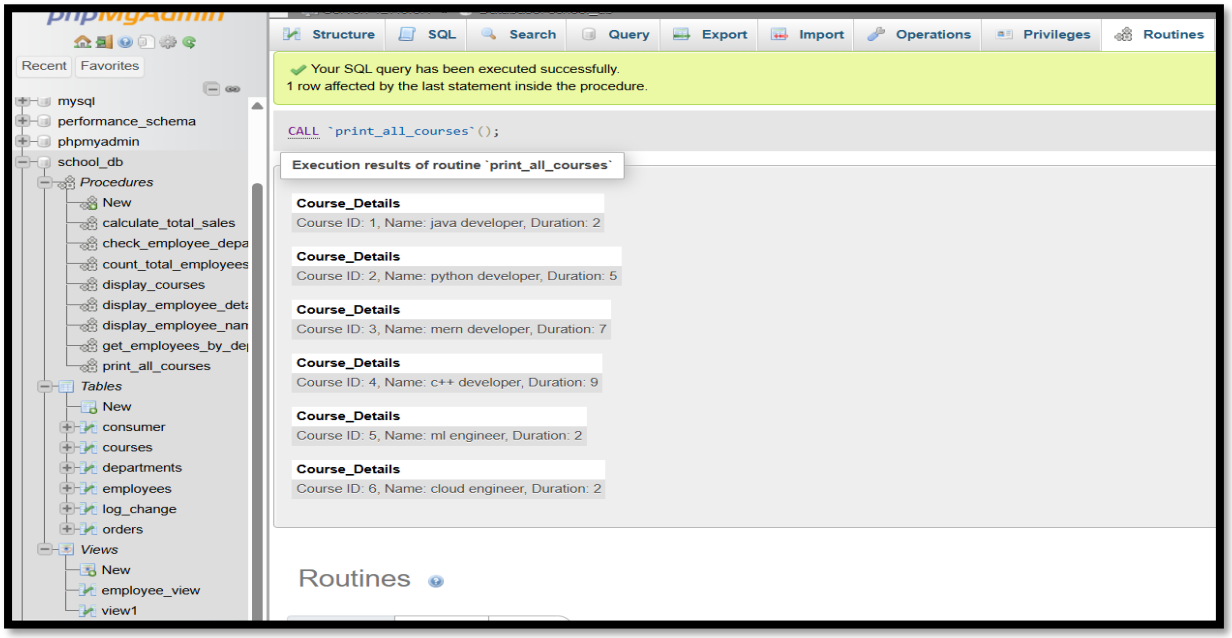
Lab 2: **Create a cursor to retrieve all courses and display them one by one.**

Answer :

```
DELIMITER //


CREATE PROCEDURE print_all_courses()
BEGIN
   DECLARE v_course_id INT;
   DECLARE v_course_name VARCHAR(100);
   DECLARE v_duration INT;
   DECLARE done INT DEFAULT FALSE;
   DECLARE course_cursor CURSOR FOR
      SELECT course_id, course_name,  course_duration
FROM courses;
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET
done = TRUE;
   OPEN course_cursor;
   read_loop: LOOP
      FETCH course_cursor INTO v_course_id,
v_course_name,v_duration;
```

```
        IF done THEN
            LEAVE read_loop;
        END IF;
        SELECT CONCAT('Course ID: ', v_course_id, ', Name: ', v_course_name, ', Duration: ', v_duration) AS Course_Details;
    END LOOP;
    CLOSE course_cursor;
END //
DELIMITER ;
```

Output :



19.) Rollback and Commit Savepoint

1.) Explain the concept of SAVEPOINT in transaction management. How do ROLLBACK and COMMIT interact with savepoints?

Answer :

SAVEPOINT in Transaction Management

SAVEPOINT is a command in SQL that allows you to set a marker within a transaction. This marker can be used to roll back part of a transaction without affecting the entire transaction.

Interaction with ROLLBACK and COMMIT:

- **ROLLBACK :** You can use the ROLLBACK TO SAVEPOINT command to revert the transaction to a specific savepoint, undoing all changes made after that savepoint while preserving the changes made before it.
- **COMMIT :** When you issue a COMMIT command, it saves all changes made in the transaction, including those before and after the savepoint, making them permanent in the database.

2.) When is it useful to use savepoints in a database transaction?

Answer :

Savepoints are useful in long, complex transactions where you need granular control to revert parts of the transaction without rolling back the entire operation. They allow you to manage partial rollbacks, enhance error handling, and improve transaction management by providing checkpoints that safeguard the progress up to certain points within the transaction.

Savepoints enhance flexibility, control, and error recovery in transaction management, making them valuable in scenarios involving multiple steps or critical operations.

LAB EXERCISES :

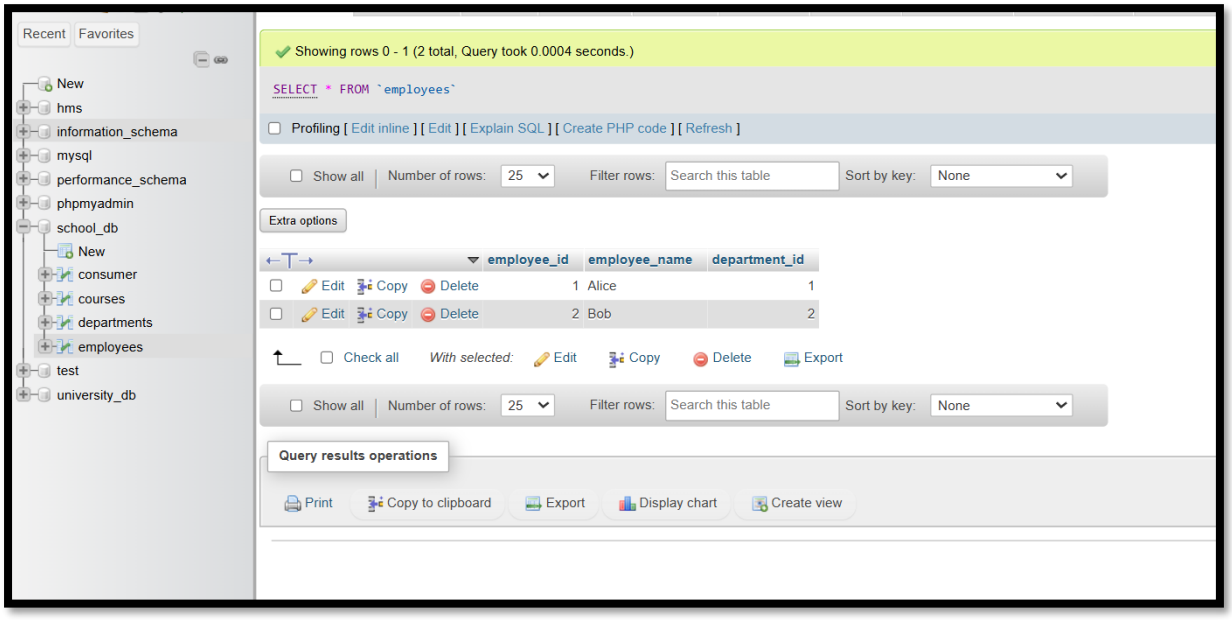
Lab 1: **Perform a transaction where you create a savepoint, insert records, then rollback to the savepoint.**

Answer :

START TRANSACTION;

```
INSERT INTO employees (employee_id, employee_name, department_id) VALUES (1, 'Alice', 1);

INSERT INTO employees (employee_id, employee_name, department_id) VALUES (2, 'Bob', 2);

SAVEPOINT before_addition;

INSERT INTO employees (employee_id, employee_name, department_id) VALUES (3, 'Charlie', 1);

INSERT INTO employees (employee_id, employee_name, department_id) VALUES (4, 'David', 3);


ROLLBACK TO SAVEPOINT before_addition;

COMMIT;
```

Output :

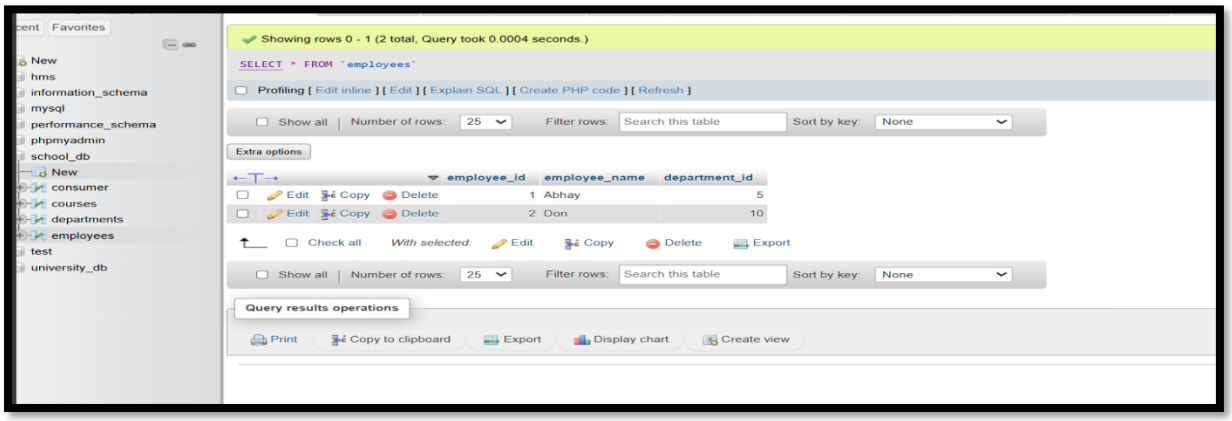


Lab 2: Commit part of a transaction after using a savepoint and then rollback the remaining changes.

Answer :

```
START TRANSACTION;

INSERT INTO employees (employee_id, employee_name, department_id) VALUES (1, 'Abhay', 5);

INSERT INTO employees (employee_id, employee_name, department_id) VALUES (2, 'Don', 10);

SAVEPOINT savepoint1;

INSERT INTO employees (employee_id, employee_name, department_id) VALUES (3, 'Che', 15);

ROLLBACK TO savepoint1;

COMMIT;
```

Output :



EXTRA LAB PRACTISE FOR DATABASE CONCEPTS :

1.) Introduction to SQL

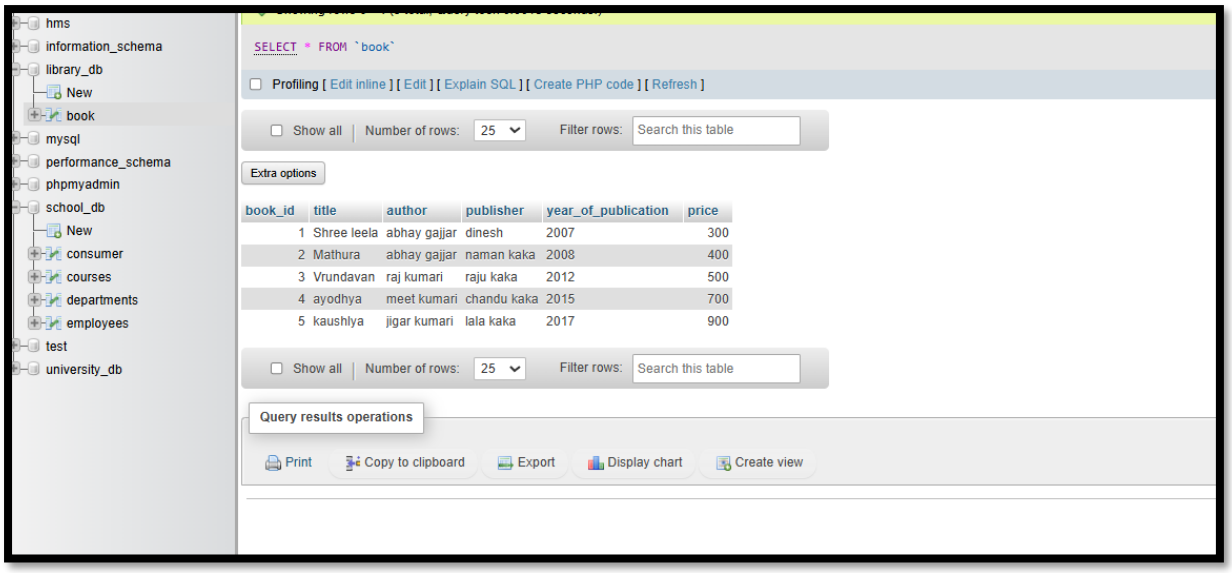
LAB EXERCISES:

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 :

insert into book VALUES (1, "Shree leela","abhay gajjar","dinesh",2007,300), (2, "Mathura","abhay gajjar","naman kaka",2008,400), (3, "Vrundavan","raj kumari","raju kaka",2012,500), (4, "ayodhya","meet kumari","chandu kaka",2015,700), (5, "kaushlya","jigar kumari","lala kaka",2017,900);

Output :



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 :

insert into members VALUES(1,"abhay","2014-01-20","abhay@gmail.com"),

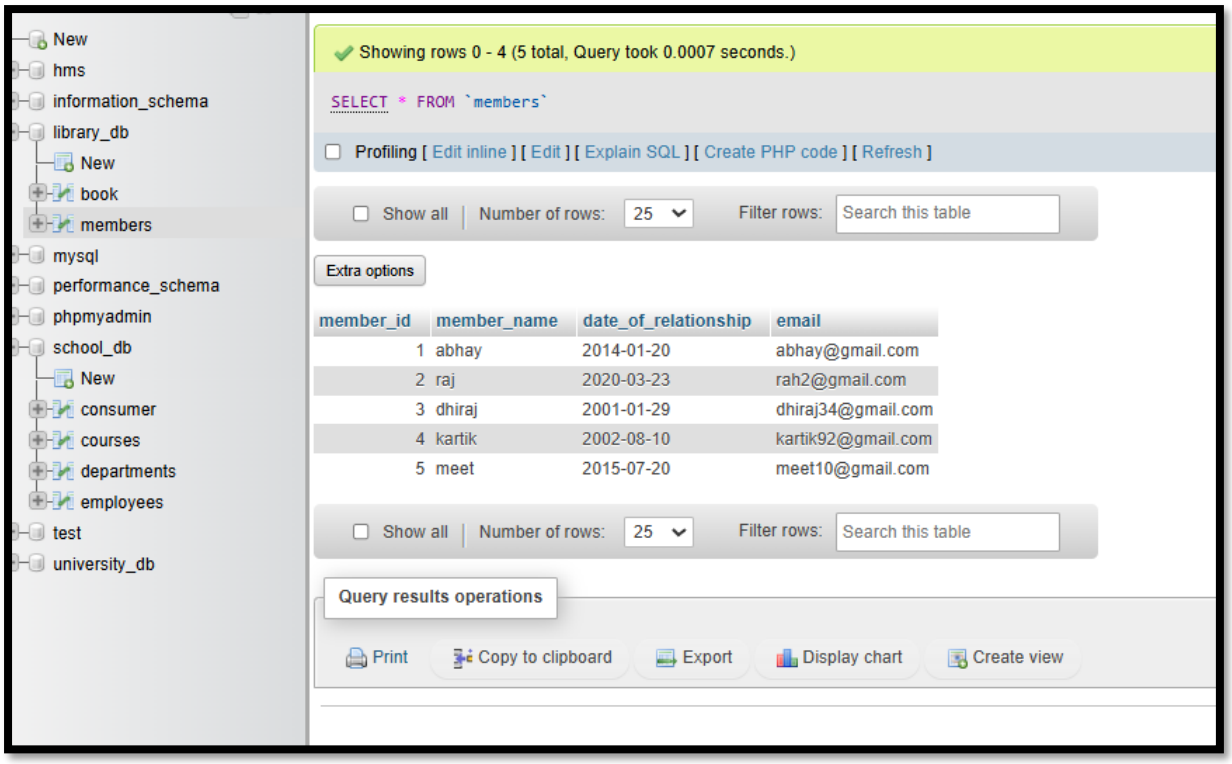
(2,"raj","2020-03-23","rah2@gmail.com"),

(3,"dhiraj","2001-01-29","dhiraj34@gmail.com"),

(4,"kartik","2002-08-10","kartik92@gmail.com"),

(5,"meet","2015-07-20","meet10@gmail.com");

Output :



2.) SQL Syntax

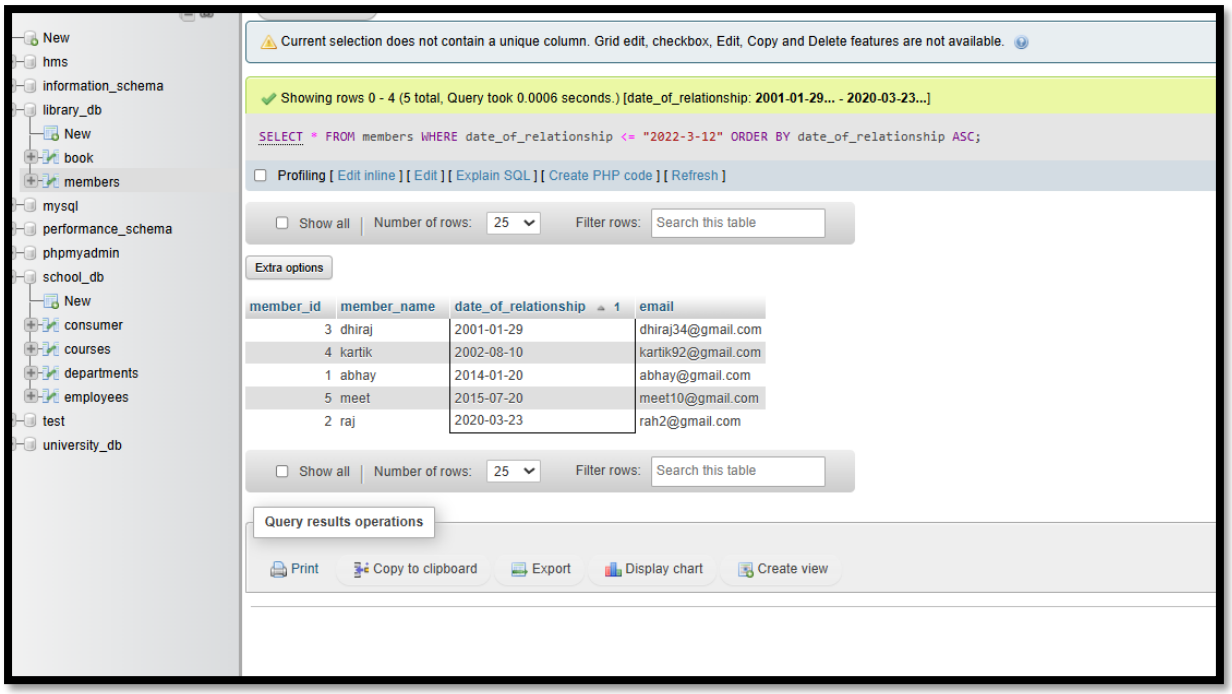
LAB EXERCISES:

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

Answer :

SELECT * FROM members WHERE date_of_relationship <= "2022-3-12" ORDER BY date_of_relationship ASC;

Output :

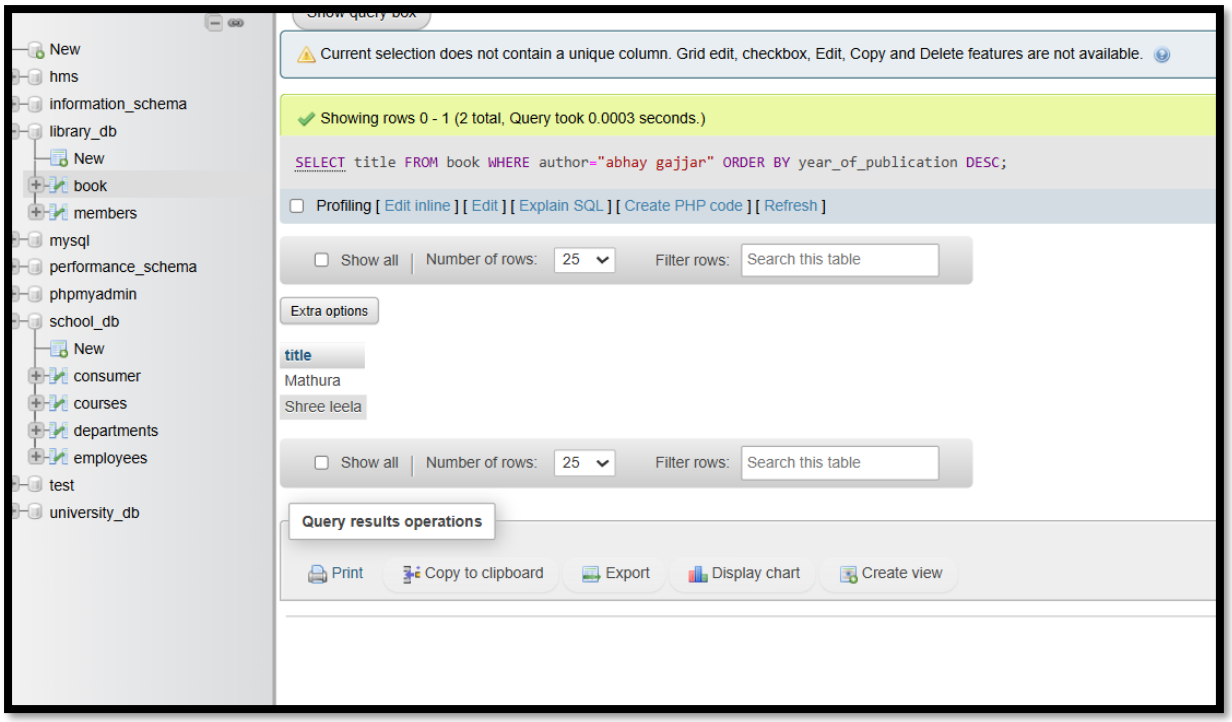


Lab 4: Write SQL queries to display the titles of books published by a specific author. Sort the results by year_of_publication in descending order.

Answer :

SELECT title FROM book WHERE author="abhay gajjar" ORDER BY year_of_publication DESC;

Output :



3.) SQL Constraints

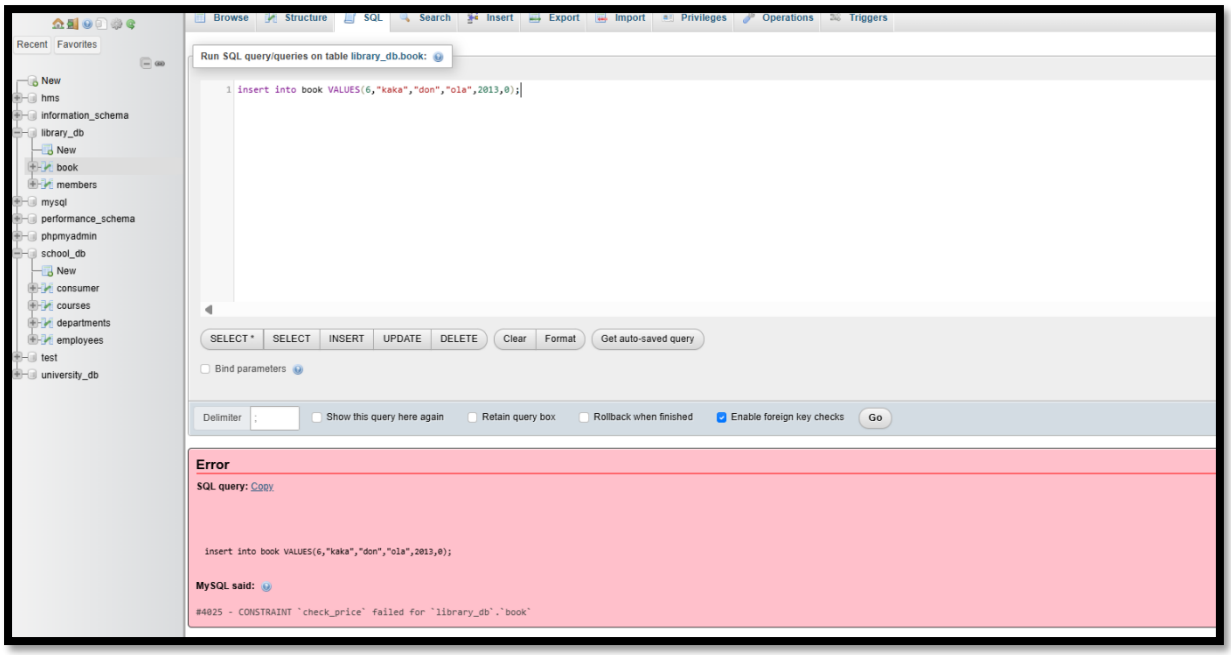
LAB EXERCISES:

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

Answer :

ALTER TABLE book ADD CONSTRAINT price CHECK (price > 0);

Output :

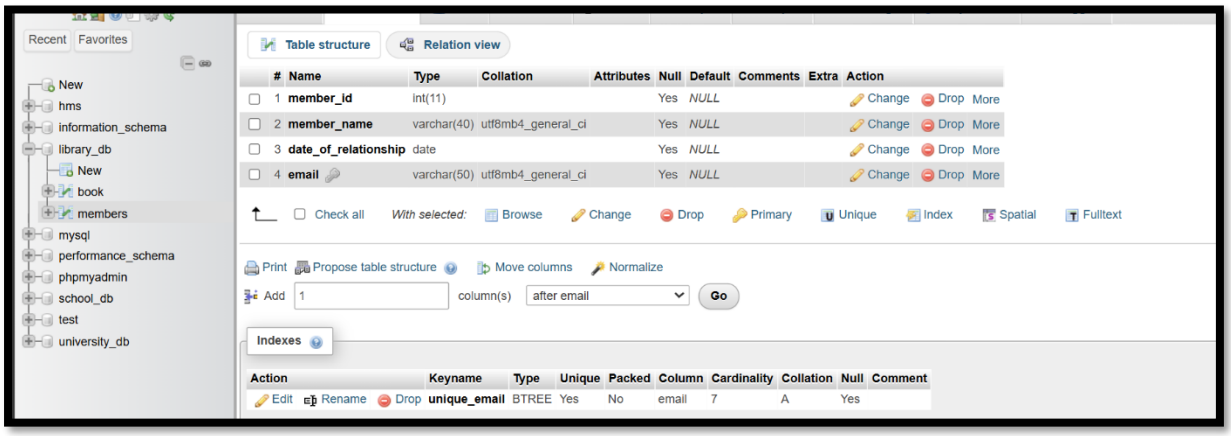


Lab 4: Modify the members table to add a UNIQUE constraint on the email column, ensuring that each member has a unique email address.

Answer :

ALTER TABLE members ADD CONSTRAINT unique_email UNIQUE(email);

Output :



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

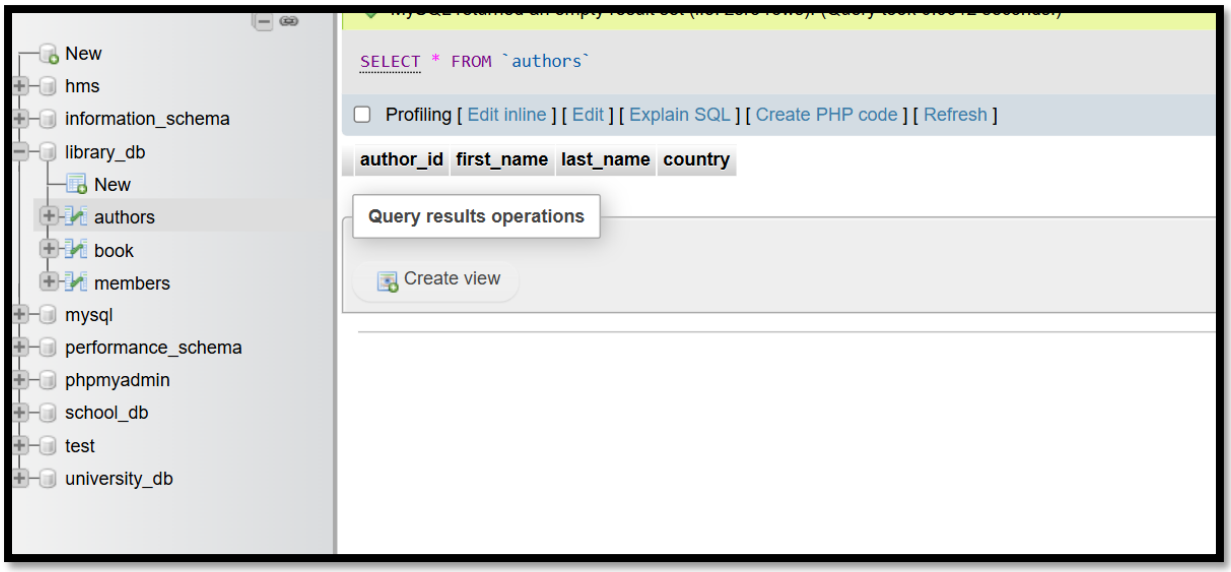
LAB EXERCISES:

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.

Answer :

```
CREATE TABLE authors
(
    author_id int PRIMARY KEY,
       first_name varchar(40),
    last_name varchar(50),
    country varchar(60)
);
```

Output :

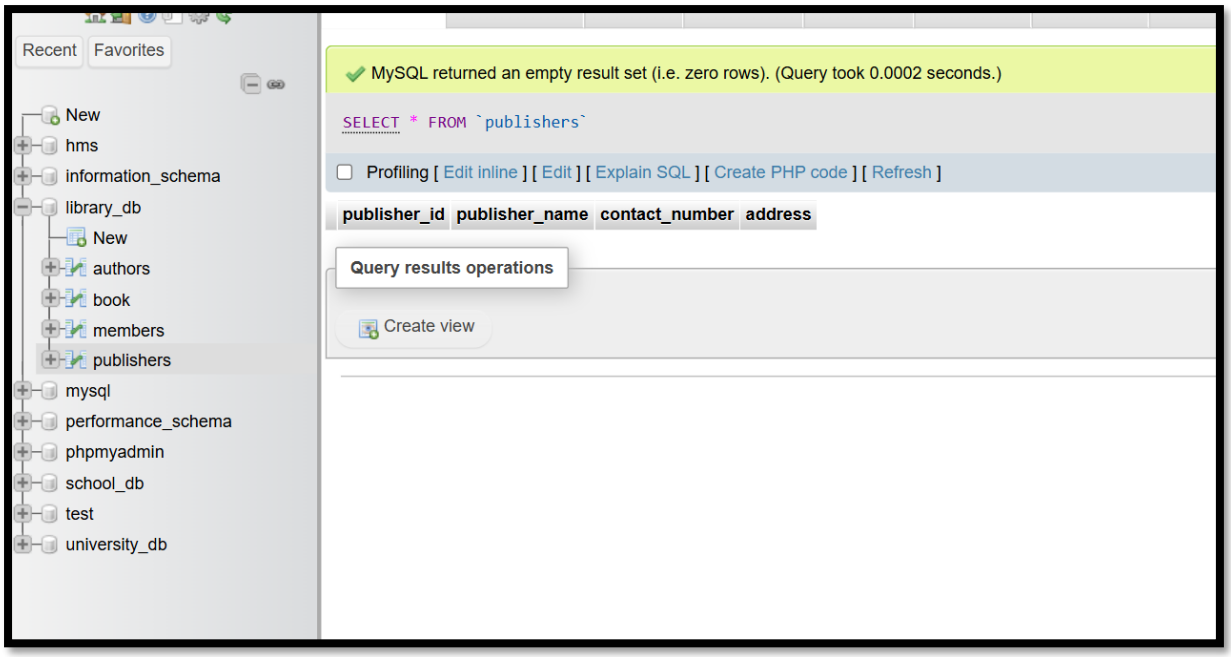


Lab 4: Create a table publishers with columns: publisher_id, publisher_name, contact_number, and address. Set publisher_id as the primary key and contact_number as unique.

Answer :

```
CREATE TABLE publishers
(
   publisher_id int PRIMARY KEY,
   publisher_name varchar(50),
   contact_number int UNIQUE,
   address varchar(20)
);
```

Output :



5.) ALTER Command

LAB EXERCISES:

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

Answer :

ALTER TABLE book add COLUMN genre int;

UPDATE book SET genre=5 WHERE book_id=1;

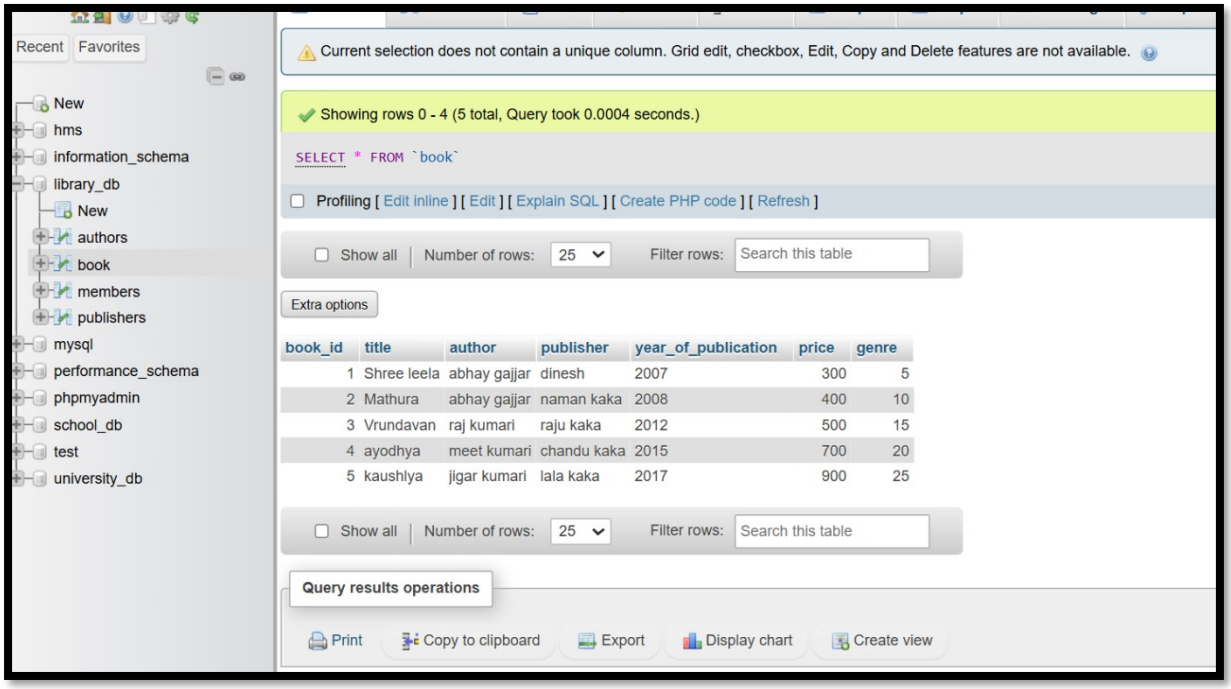
UPDATE book SET genre=10 WHERE book_id=2;

UPDATE book SET genre=15 WHERE book_id=3;

UPDATE book SET genre=20 WHERE book_id=4;

UPDATE book SET genre=25 WHERE book_id=5;

Output :

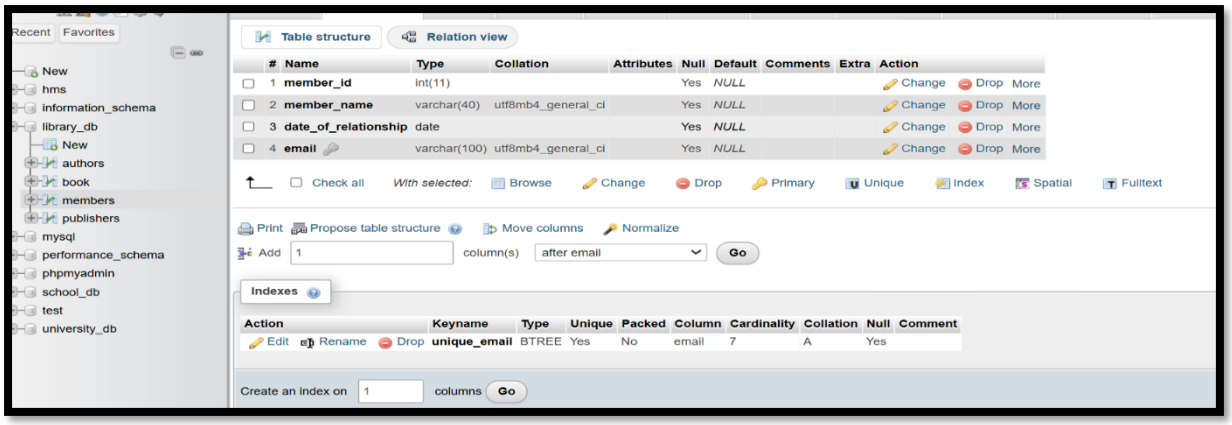


Lab 4: Modify the members table to increase the length of the email column to 100 characters.

Answer :

ALTER TABLE members MODIFY COLUMN email varchar (100);

Output :



6.) DROP Command

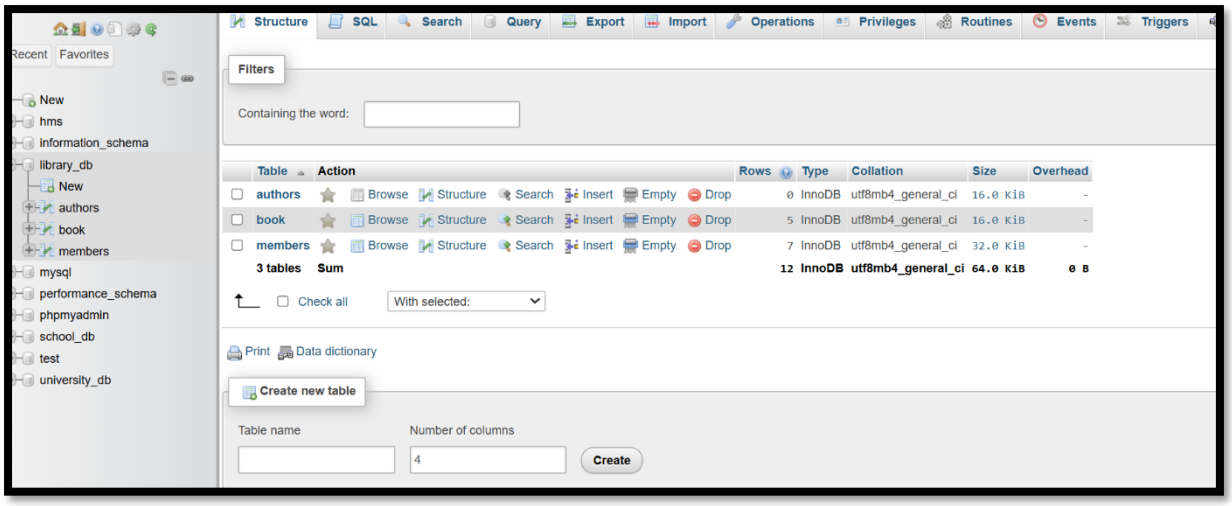
LAB EXERCISES:

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

Answer :

drop table publishers;

Output :



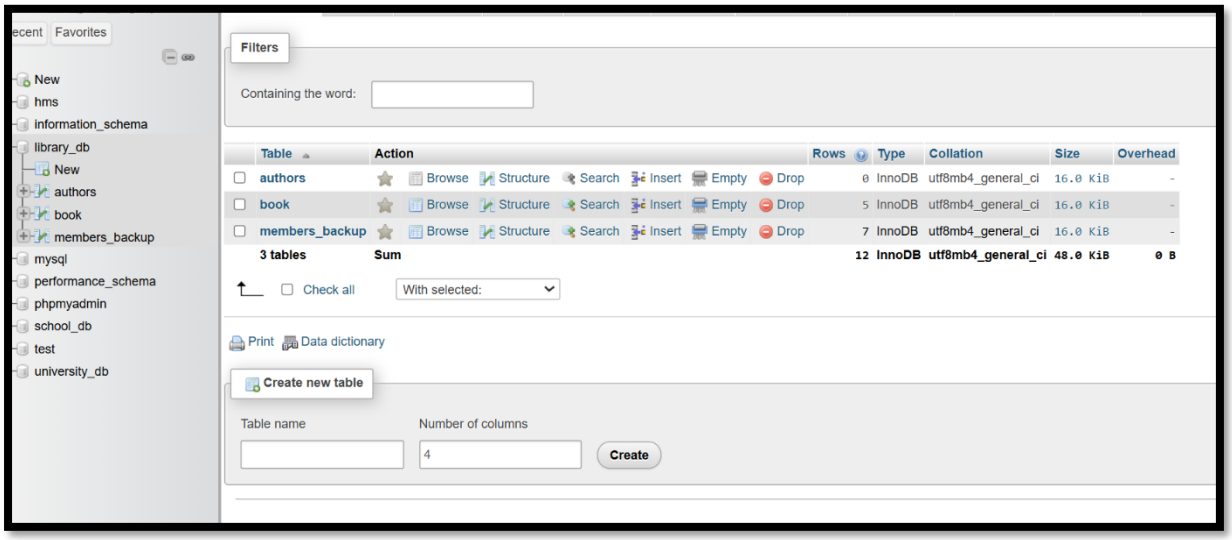
Lab 4: Create a backup of the members table and then drop the original members table.

Answer :

CREATE TABLE members_backup AS SELECT * FROM members;

DROP TABLE members;

Output :



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.**

Answer :

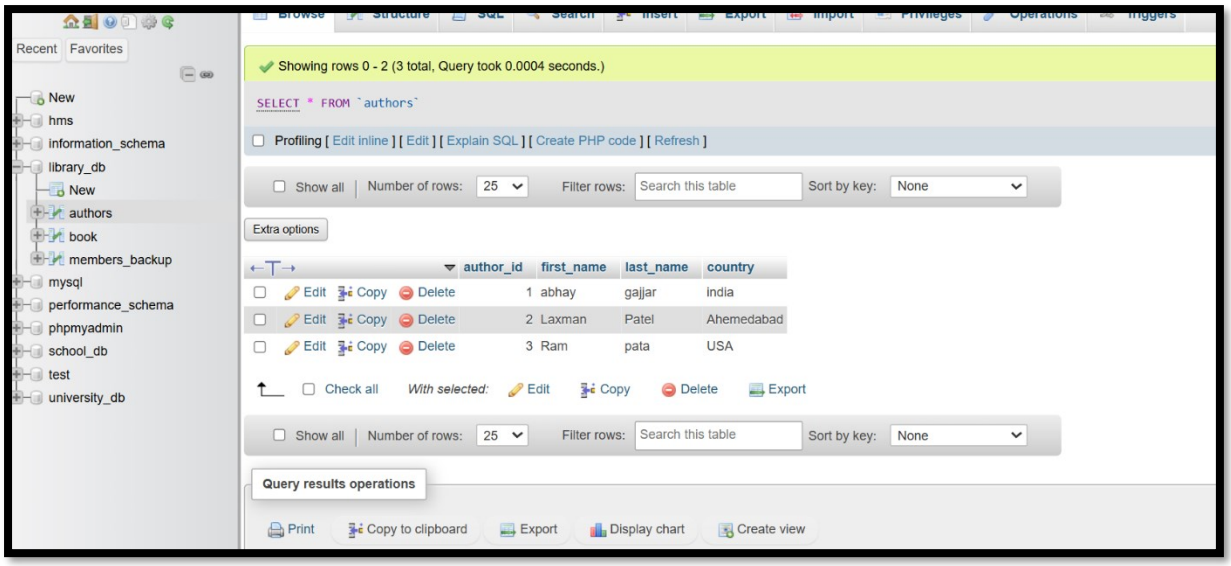
INSERT INTO authors VALUES (1,"abhay","gajjar","india"),

(2,"Laxman","Patel","Ahemedabad"),

(3,"Ram","gajjar","USA");

UPDATE authors set last_name="pata" WHERE author_id=3;

Output :

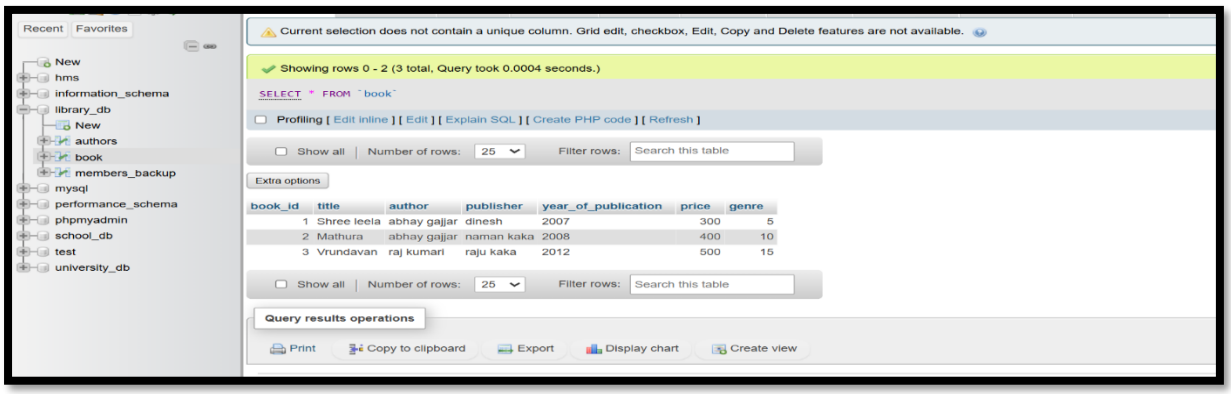


Lab 5: Delete a book from the books table where the price is higher than $100.

Answer :

```
DELETE FROM book WHERE price >= 600;
```

Output :



8.) UPDATE Command

LAB EXERCISES:

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

Answer :

UPDATE book SET year_of_publication=2023 WHERE book_id=3;

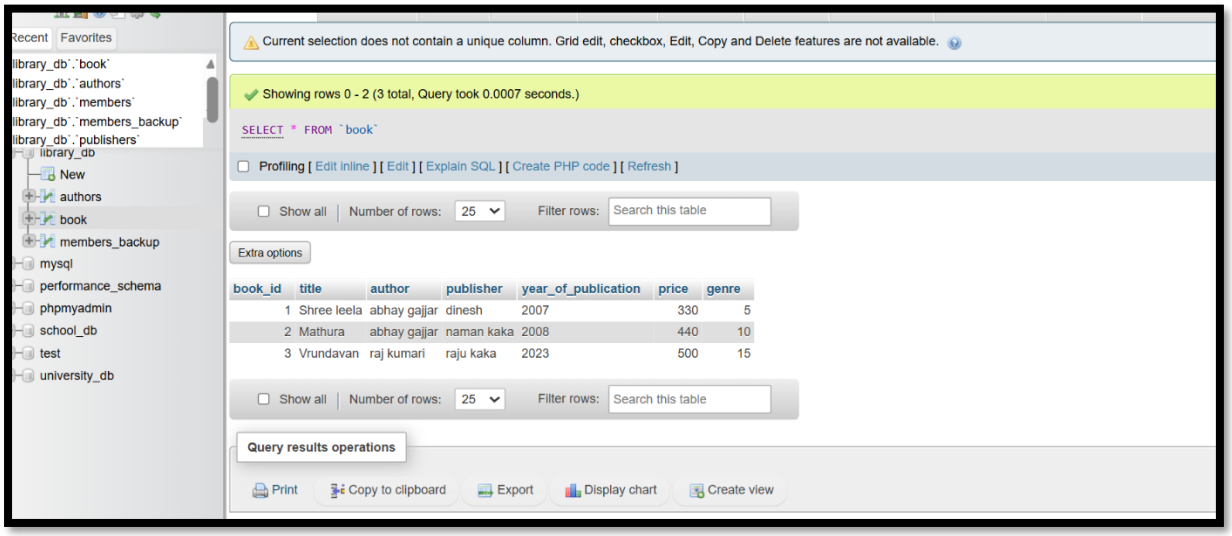
Output :

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

Answer :

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

Output :



9.) DELETE Command

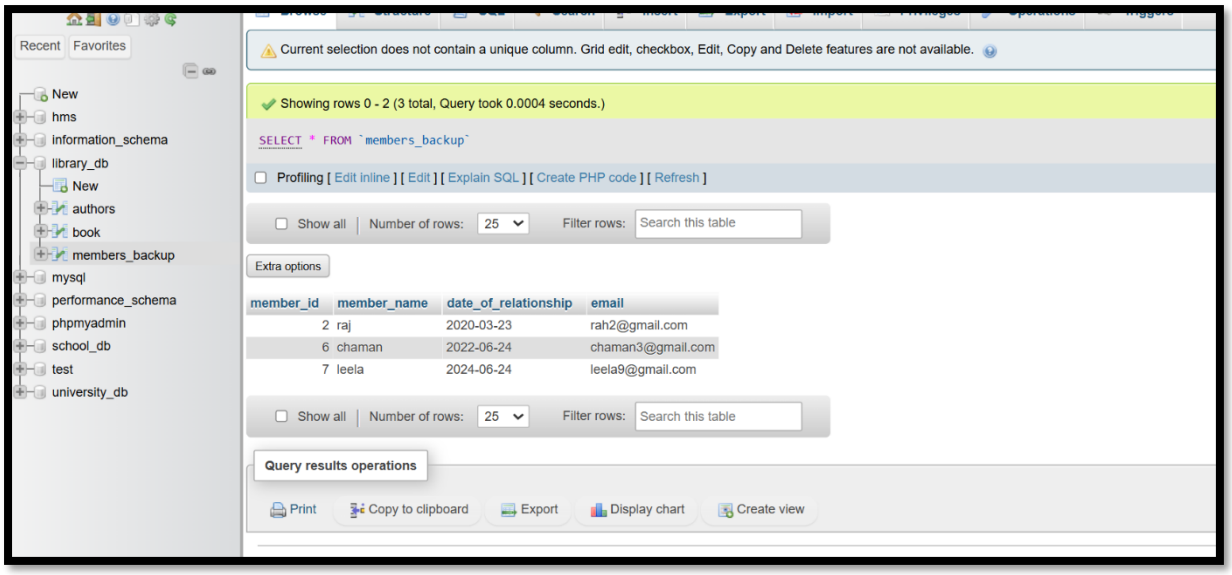
LAB EXERCISES:

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

Answer :

DELETE FROM members_backup WHERE date_of_relationship <” 2018-2-20”;

Output :



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

Answer :

DELETE FROM book WHERE author="null";

Output :

10.) Data Query Language (DQL)

LAB EXERCISES:

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

Answer :

SELECT * FROM book WHERE price BETWEEN 200 and 350;

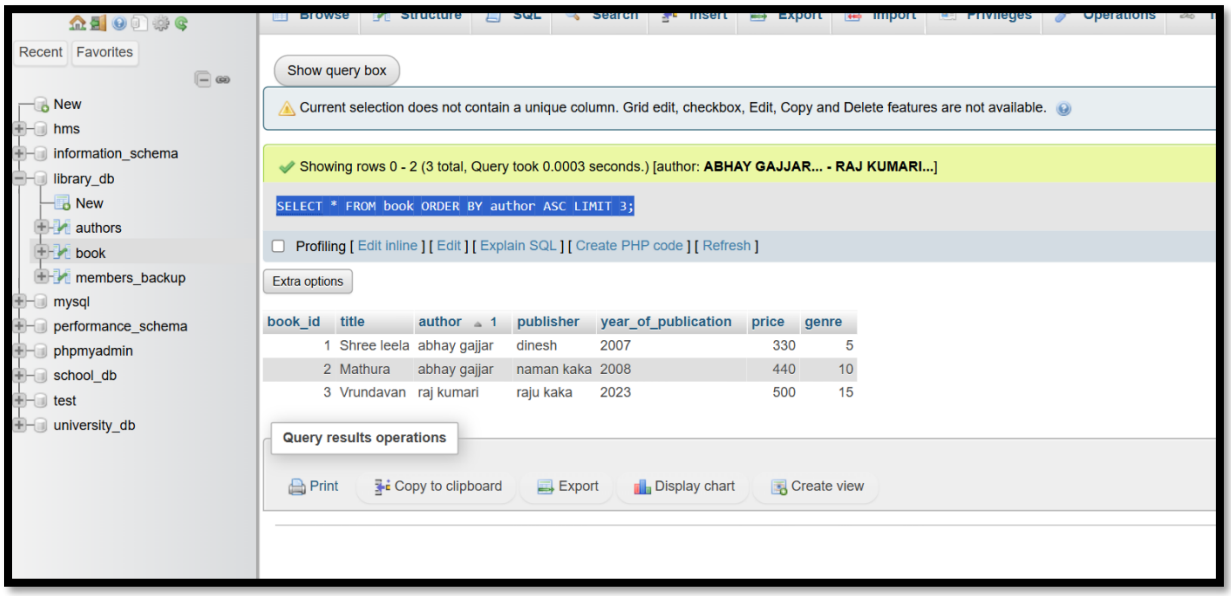
Output :

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

Answer :

SELECT * FROM book ORDER BY author ASC LIMIT 3;

Output :



11.) Data Control Language (DCL)

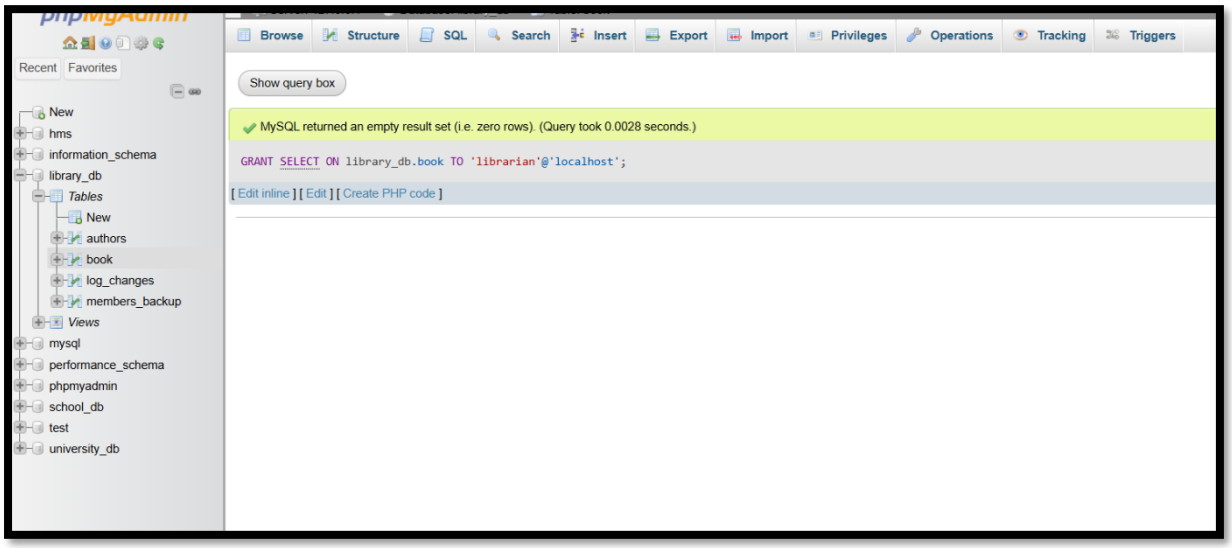
LAB EXERCISES:

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

Answer :

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

Output :

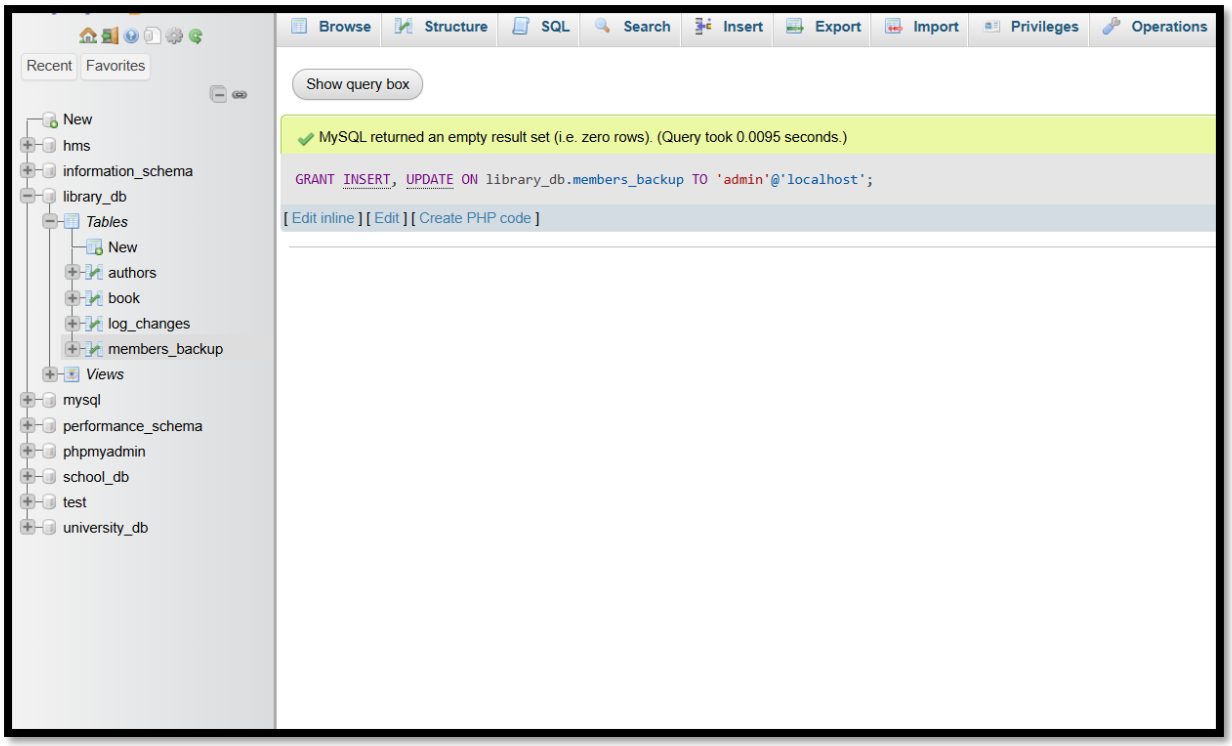


Lab 4: Grant INSERT and UPDATE permissions to the user admin on the members table.

Answer :

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

Output :



12.) REVOKE Command

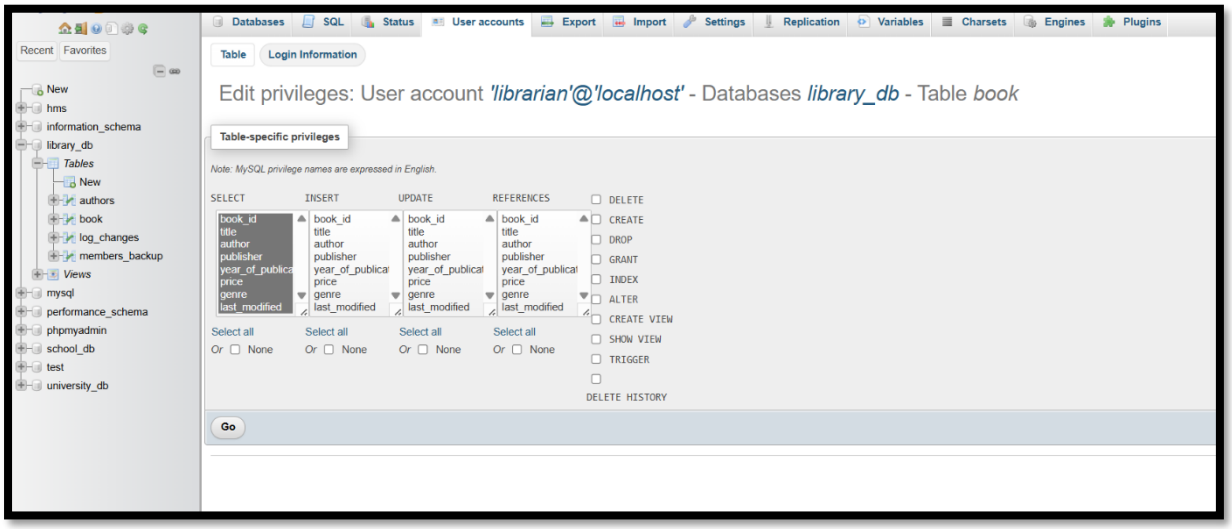
LAB EXERCISES:

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

Answer :

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

Output :

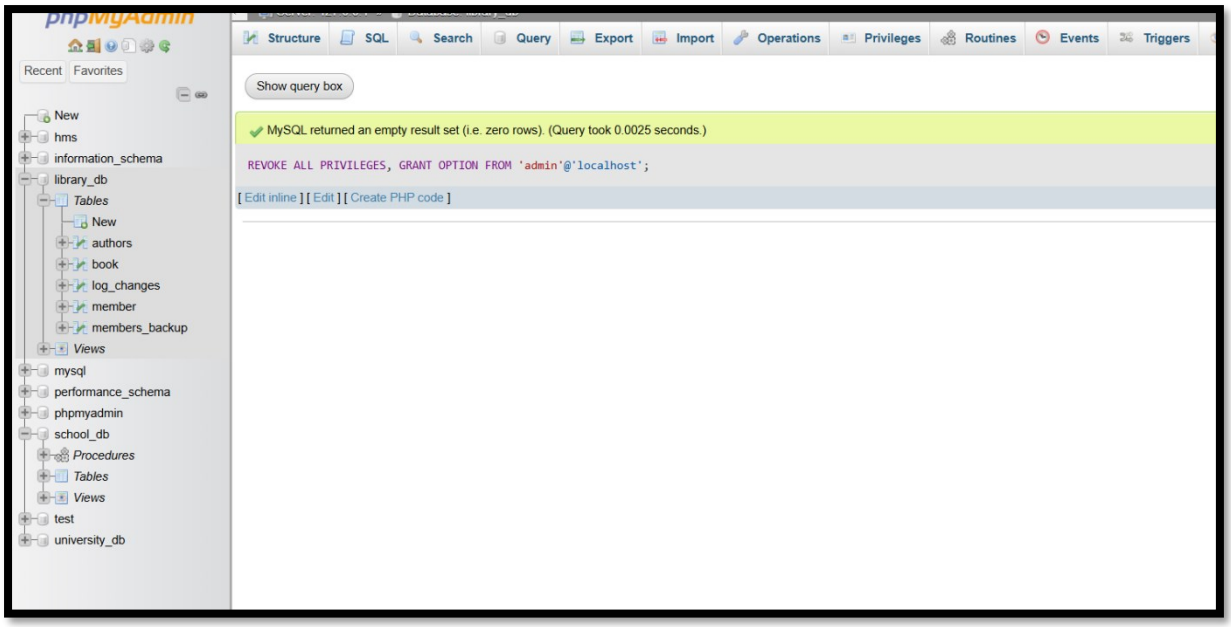


Lab 4: Revoke all permissions from user admin on the members table.

Answer :

REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'admin' @'localhost';

Output :



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.

Answer :

START TRANSACTION;

INSERT INTO book VALUES(4,"Har Har MAHADEV","yellow","ula",2024,4450,20);

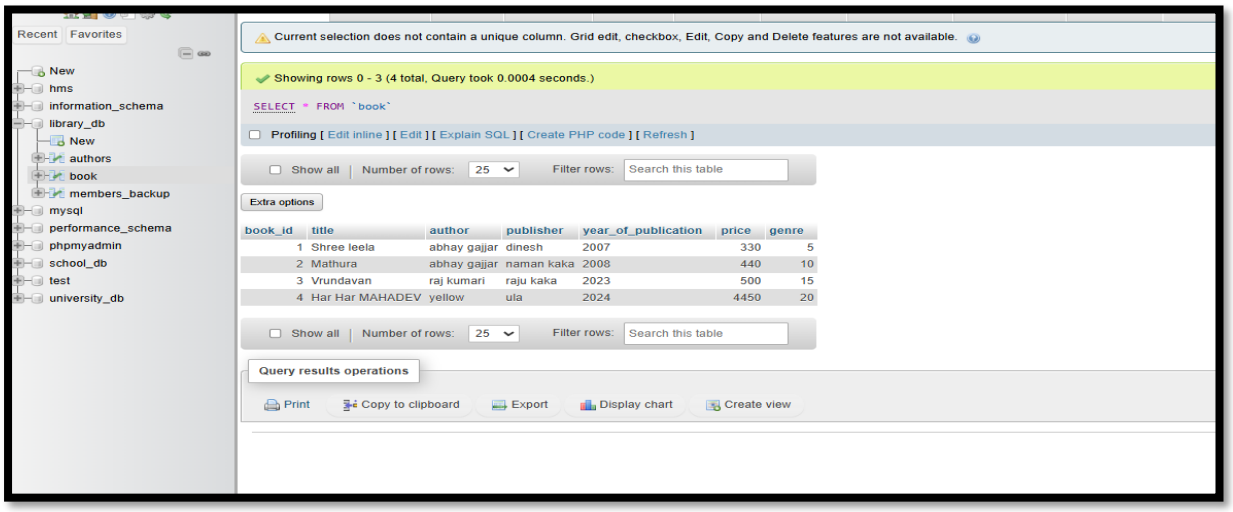
COMMIT;

START TRANSACTION;

INSERT INTO book VALUES(5,"hajir moj","shivalay","lola ji",2018,980,80);

ROLLBACK;

Output :



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

Answer :

START TRANSACTION;

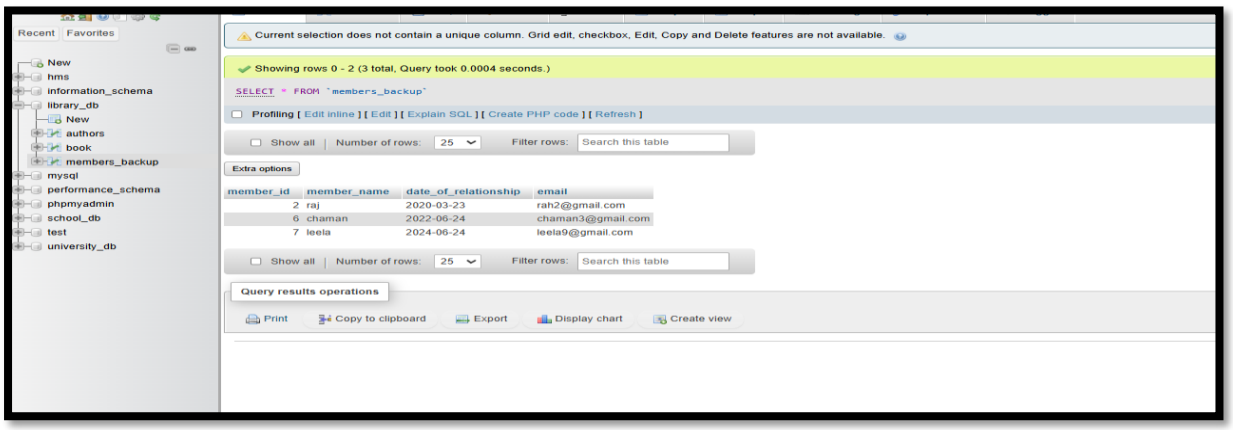
SAVEPOINT before_update;

UPDATE members_backup

SET email = 'yela1@gmail.com'

```
WHERE member_id = 6;

UPDATE members_backup

SET email = 'loal2@gmail.com'

WHERE member_id = 7;

ROLLBACK TO SAVEPOINT before_update;

COMMIT;
```

Output :



14.) SQL Joins

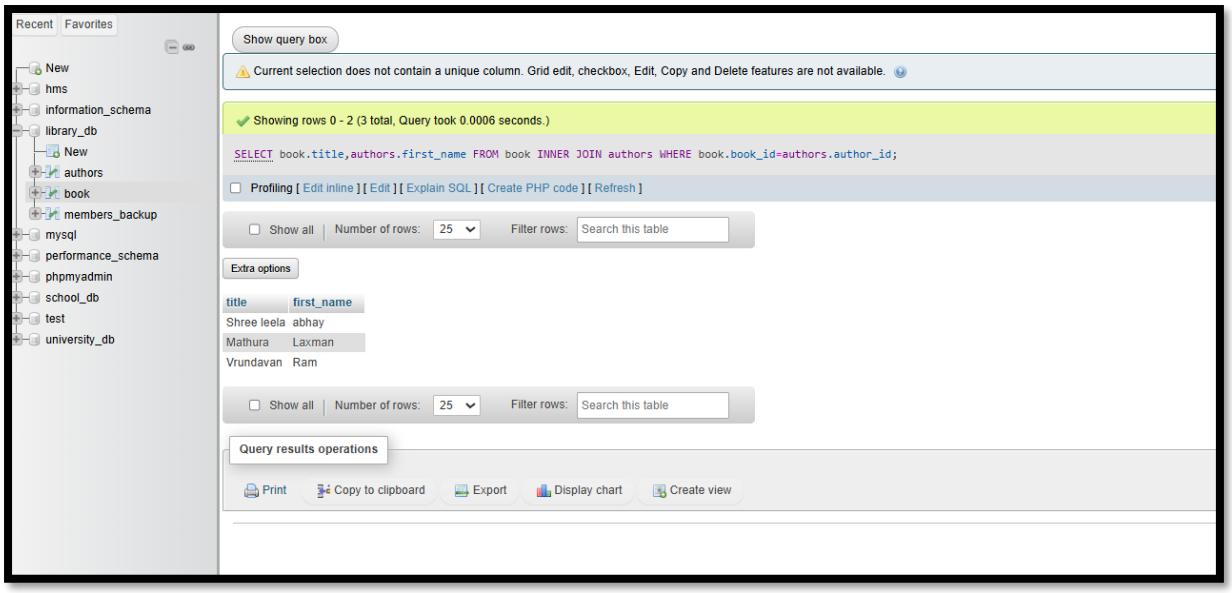
LAB EXERCISES:

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

Answer :

```
SELECT book.title,authors.first_name
FROM book
INNER JOIN authors
WHERE book.book_id=authors.author_id;
```

Output :

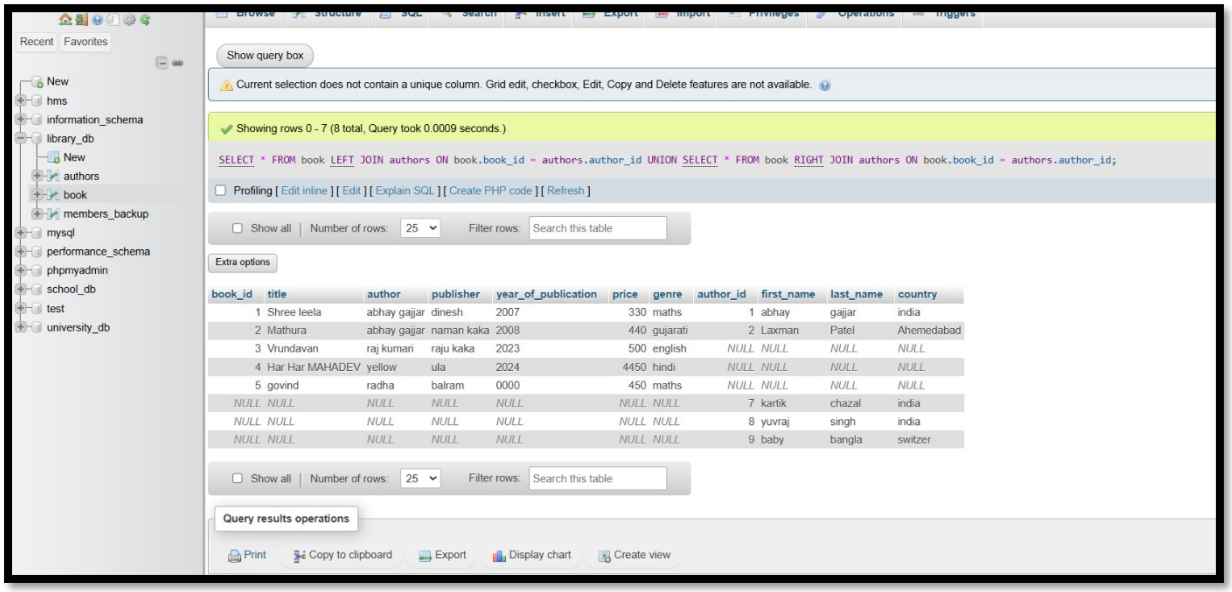


Lab 4: Use a FULL OUTER JOIN to retrieve all records from the books and authors tables, including those with no matching entries in the other table.

Answer :

SELECT * FROM book LEFT JOIN authors ON book.book_id = authors.author_id UNION SELECT * FROM book RIGHT JOIN authors ON book.book_id = authors.author_id;

Output :



15.) SQL Group By

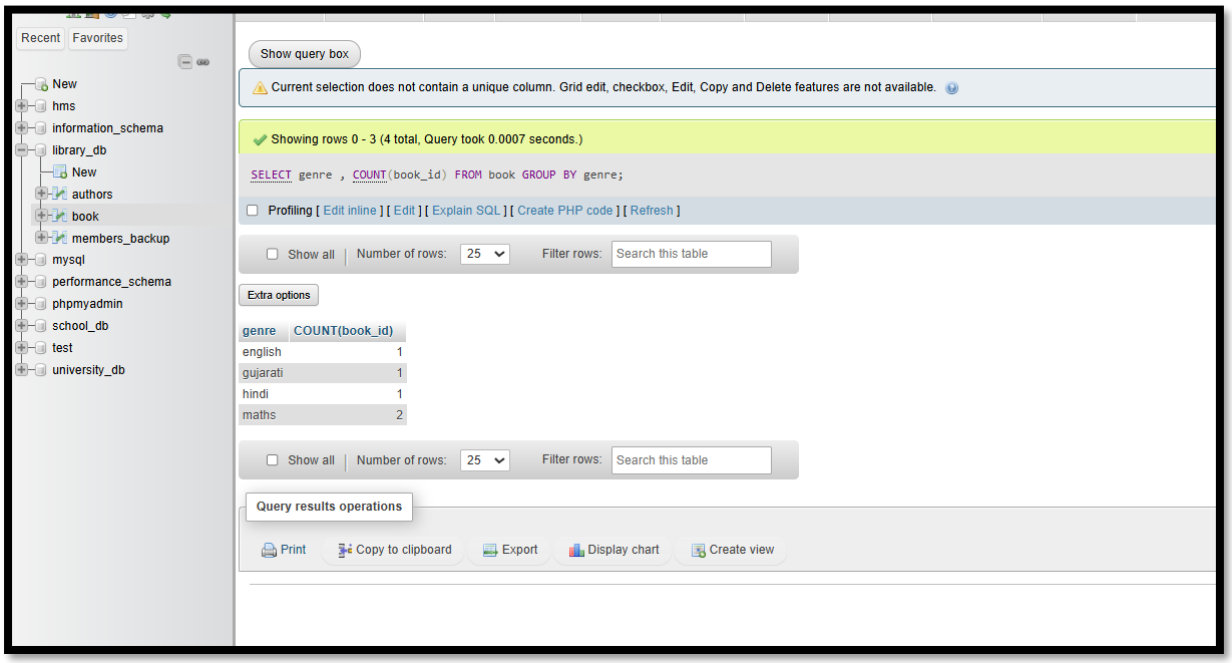
LAB EXERCISES:

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

Answer :

```
SELECT genre , COUNT(book_id)
FROM book
GROUP BY genre;
```

Output :

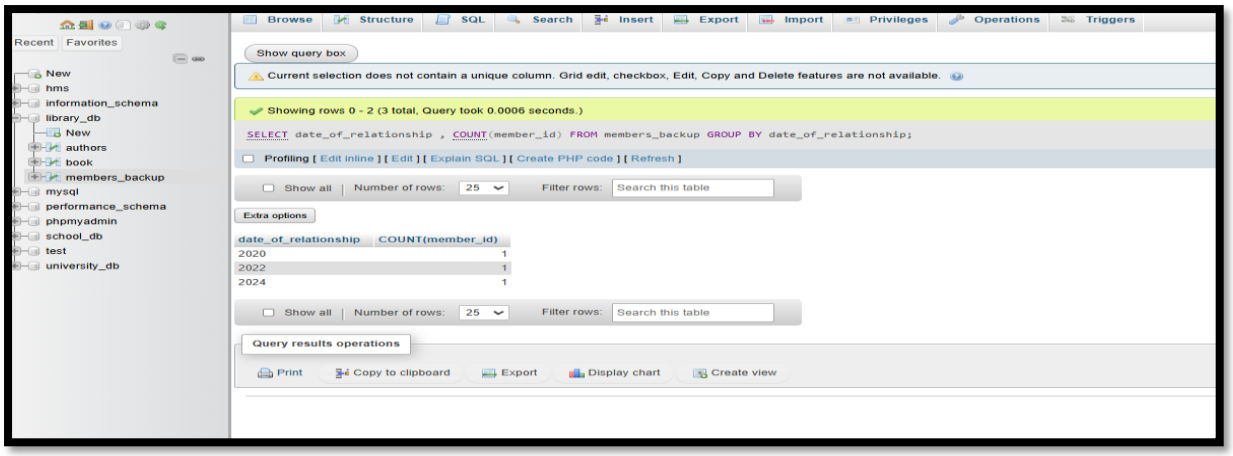


Lab 4: Group members by the year they joined and find the number of members who joined each year.

Answer :

```
SELECT date_of_relationship , COUNT(member_id)
FROM members_backup
GROUP BY date_of_relationship;
```

Output :



16.) SQL Stored Procedure

LAB EXERCISES:

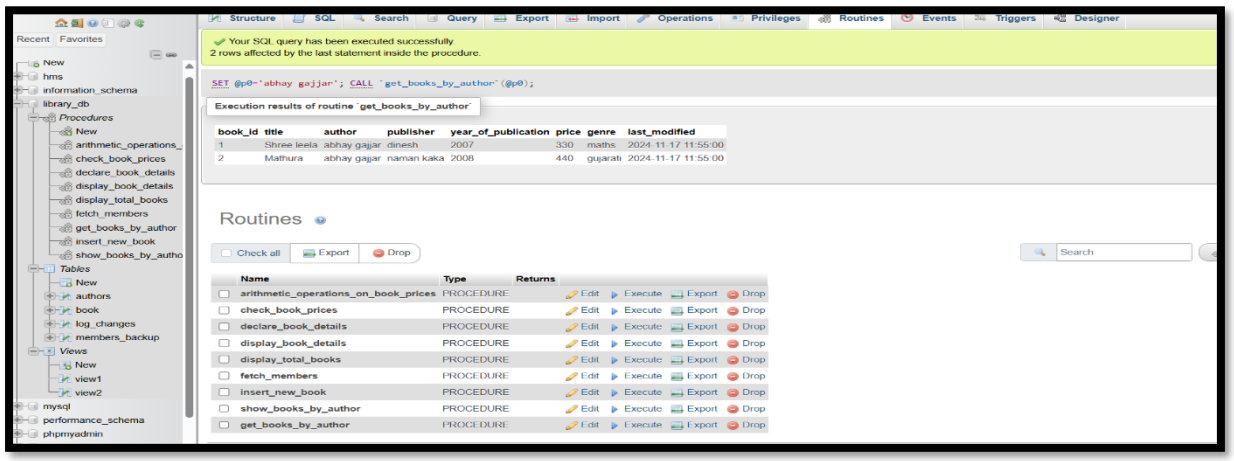
Lab 3: **Write a stored procedure to retrieve all books by a particular author.**

Answer :

```
DELIMITER //

CREATE PROCEDURE get_books_by_author(IN author_name VARCHAR(100))

BEGIN

   SELECT * FROM book

   WHERE author = author_name;

END //
```

DELIMITER ;

Output :



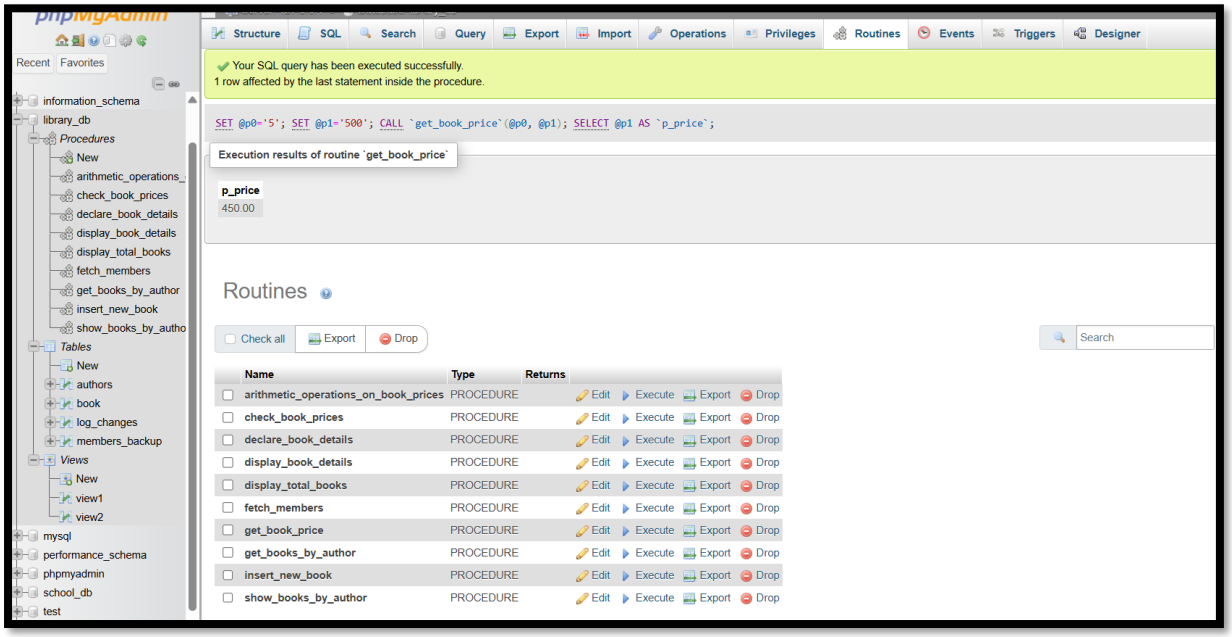
Lab 4: Write a stored procedure that takes book_id as an argument and returns the price of the book.

Answer :

```
DELIMITER //

CREATE PROCEDURE get_book_price(IN p_book_id INT,
OUT p_price DECIMAL(10, 2))

BEGIN

    SELECT price INTO p_price

    FROM book

    WHERE book_id = p_book_id;

END //
```

DELIMITER ;

Output :



17.) SQL View

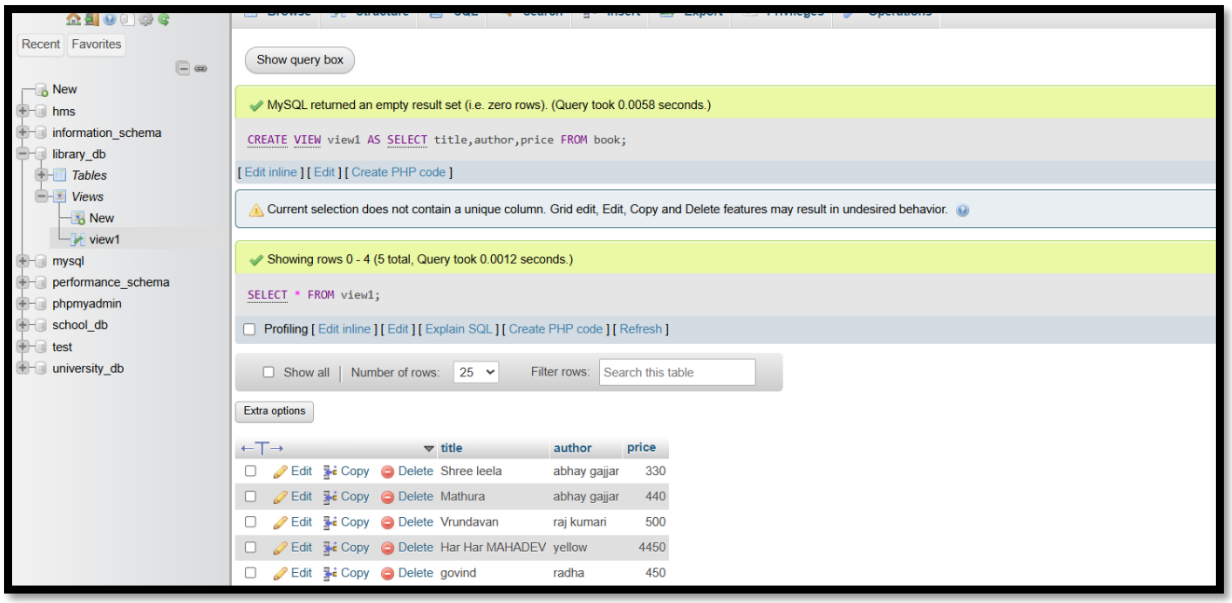
LAB EXERCISES:

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

Answer :

CREATE VIEW view1 AS SELECT title,author,price FROM book;

Output :

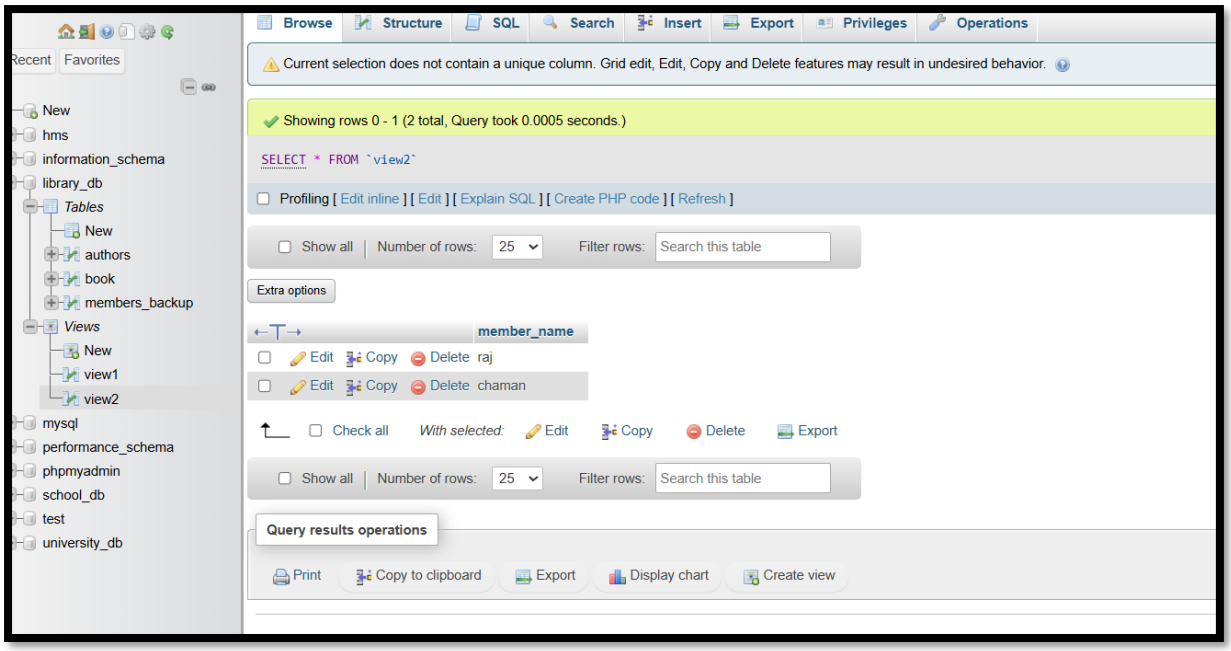


Lab 4: Create a view to display members who joined before 2020.

Answer :

CREATE VIEW view2 AS SELECT member_name FROM members_backup WHERE date_of_relationship <= 2022;

Output :



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.

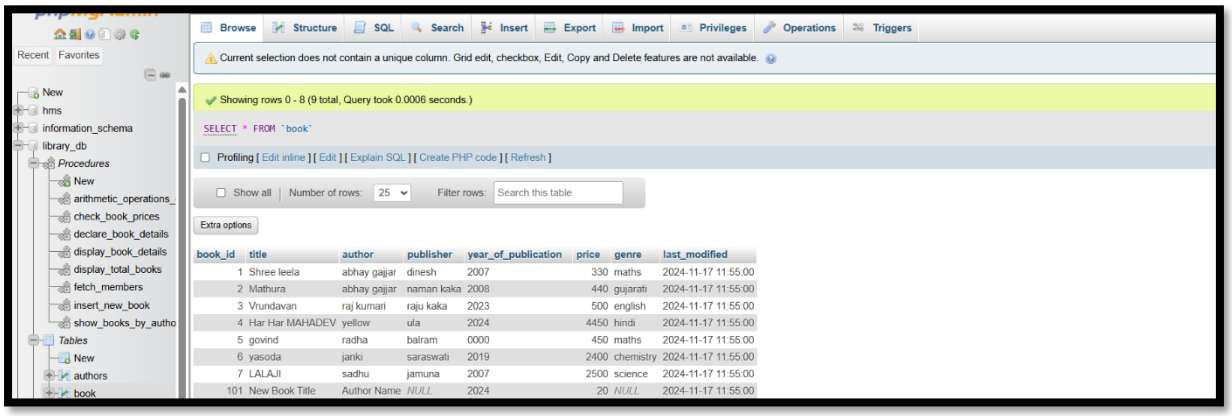
Answer :

ALTER TABLE book

ADD COLUMN last_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP;

```
DELIMITER //

CREATE TRIGGER before_books_update
BEFORE UPDATE ON book
FOR EACH ROW
BEGIN
    SET NEW.last_modified = CURRENT_TIMESTAMP;
END //
DELIMITER ;
```

Output :



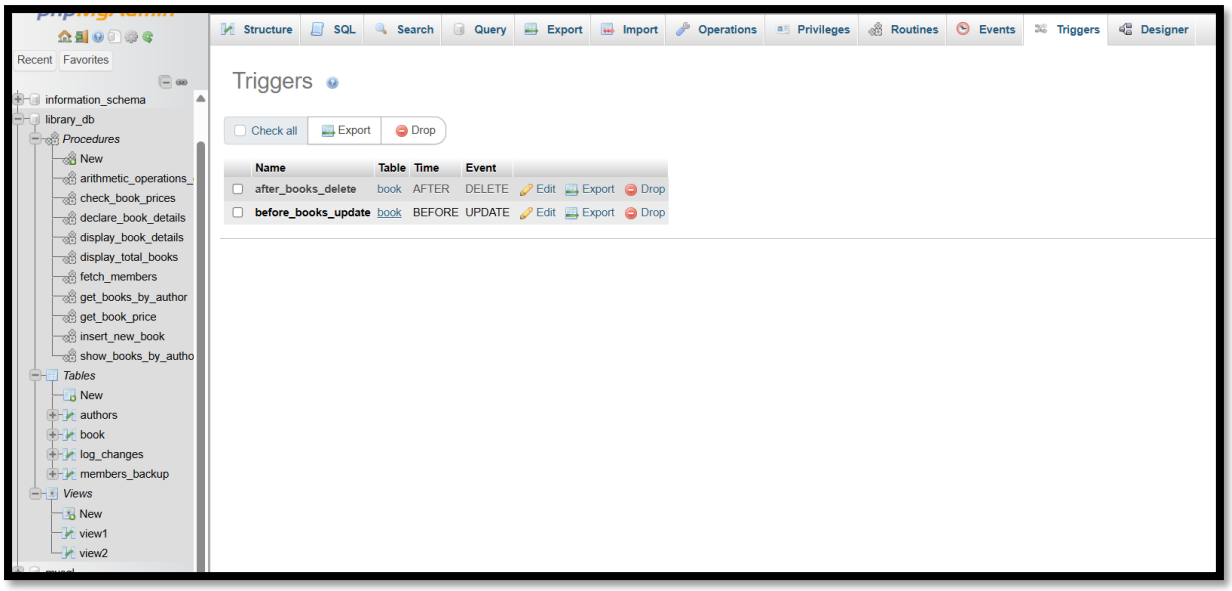
Lab 4: Create a trigger that inserts a log entry into a log_changes table whenever a DELETE operation is performed on the books table.

Answer :

```
CREATE TABLE IF NOT EXISTS log_changes (
```

```
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    book_id INT,
    title VARCHAR(100),
    author VARCHAR(100),
    deleted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
DELIMITER //

CREATE TRIGGER after_books_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, CURRENT_TIMESTAMP);
END //
DELIMITER ;
```

Output :



19.) Introduction to PL/SQL

LAB EXERCISES:

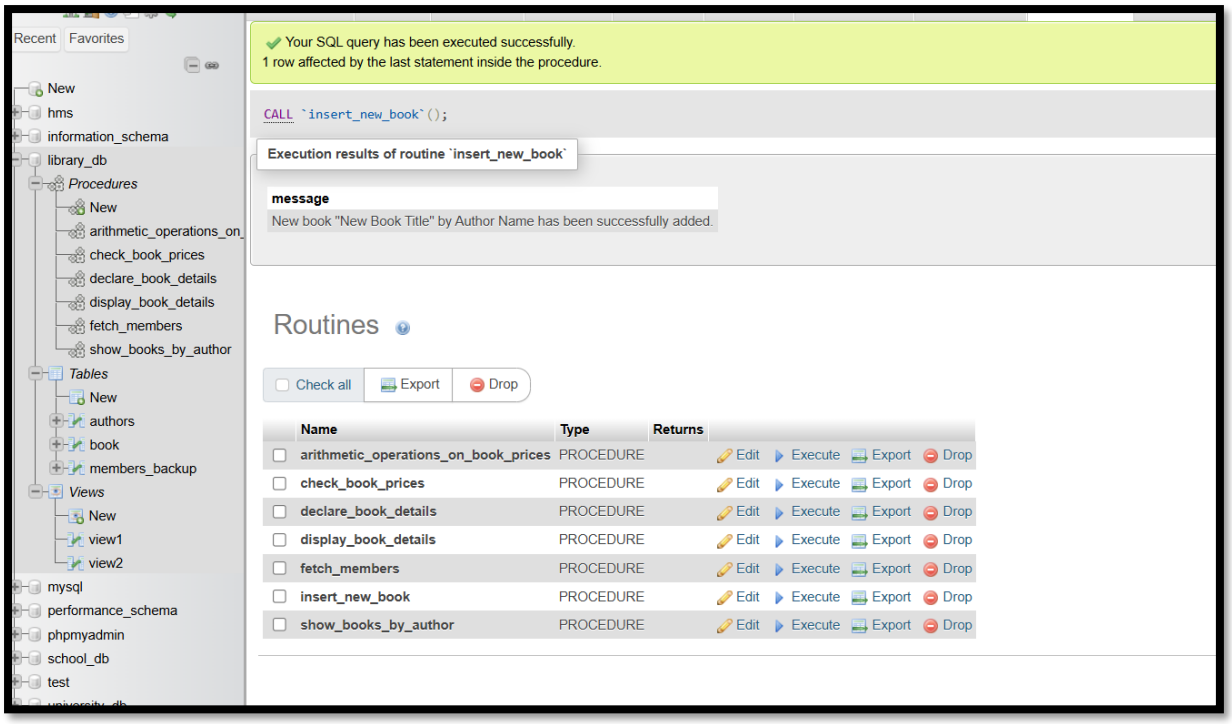
Lab 3: Write a PL/SQL block to insert a new book into the books table and display a confirmation message.

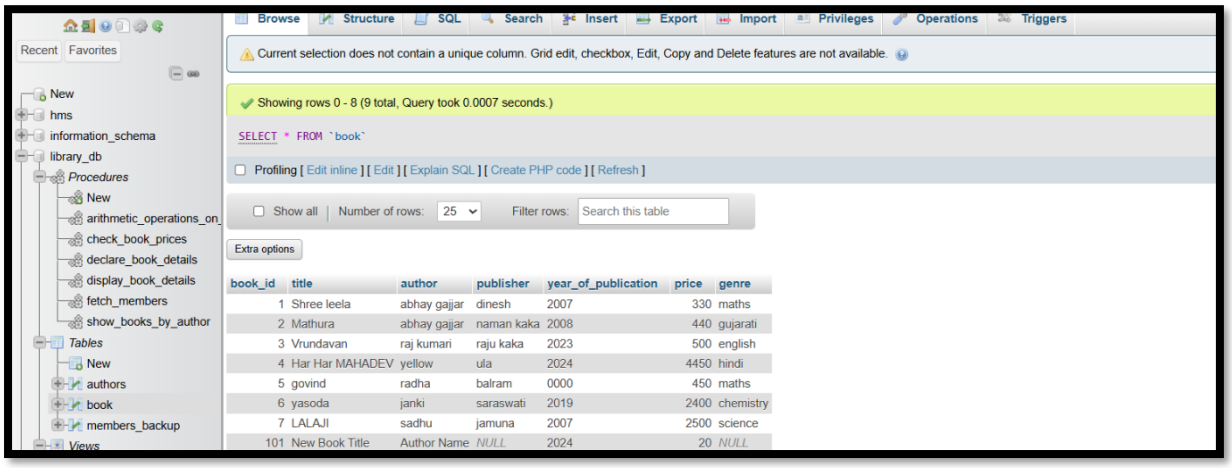
Answer :

```
BEGIN
    DECLARE v_book_id INT DEFAULT 101;
    DECLARE v_title VARCHAR(100) DEFAULT 'New Book Title';
```

```
    DECLARE v_author VARCHAR(100) DEFAULT 'Author Name';
    DECLARE v_price DECIMAL(10, 2) DEFAULT 19.99;
    DECLARE v_year_of_publication year DEFAULT 2024;

    INSERT INTO book (book_id, title, author, price, year_of_publication)
    VALUES (v_book_id, v_title, v_author, v_price, v_year_of_publication);

    SELECT CONCAT('New book "', v_title, '" by ', v_author, ' has been successfully added.') AS message;
END
```

Output :





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

Answer :

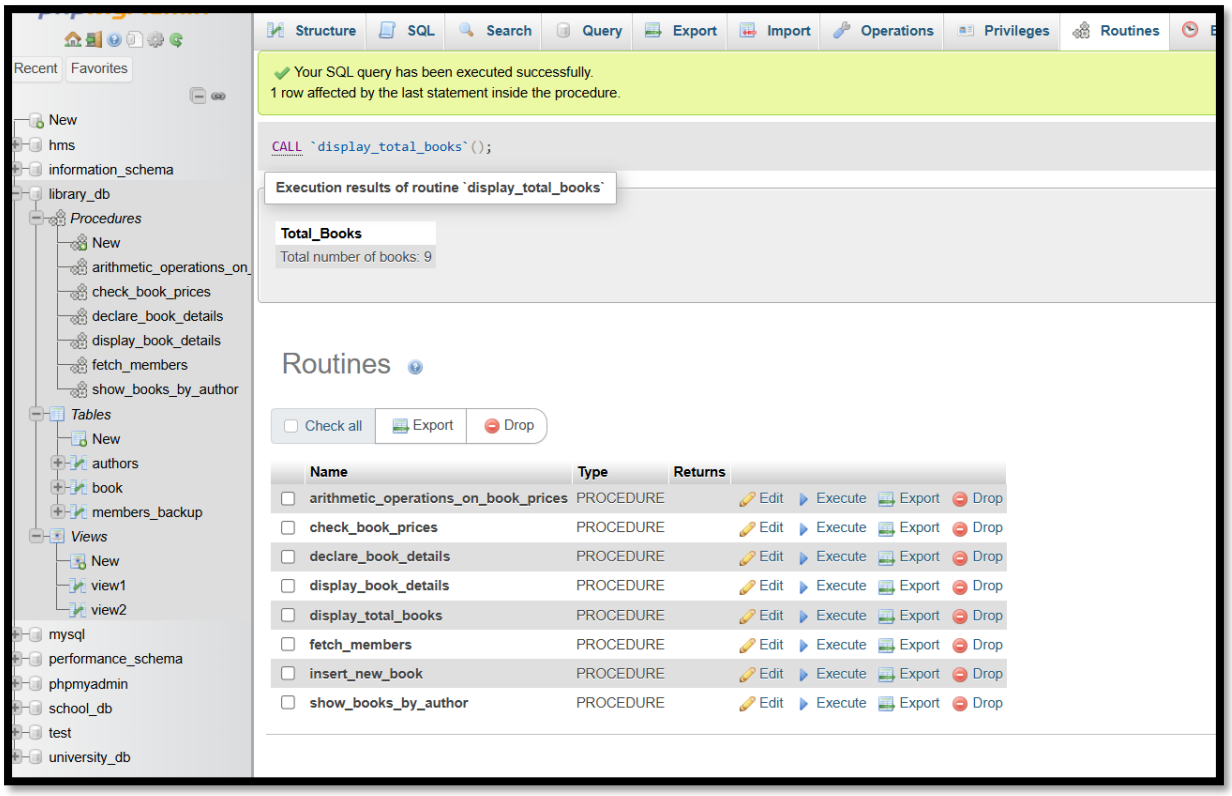
```
BEGIN
    DECLARE v_total_books INT;
```

SELECT COUNT(book_id) INTO v_total_books FROM book;

SELECT CONCAT('Total number of books: ', v_total_books) AS Total_Books;

END

Output :



20.) PL/SQL Syntax

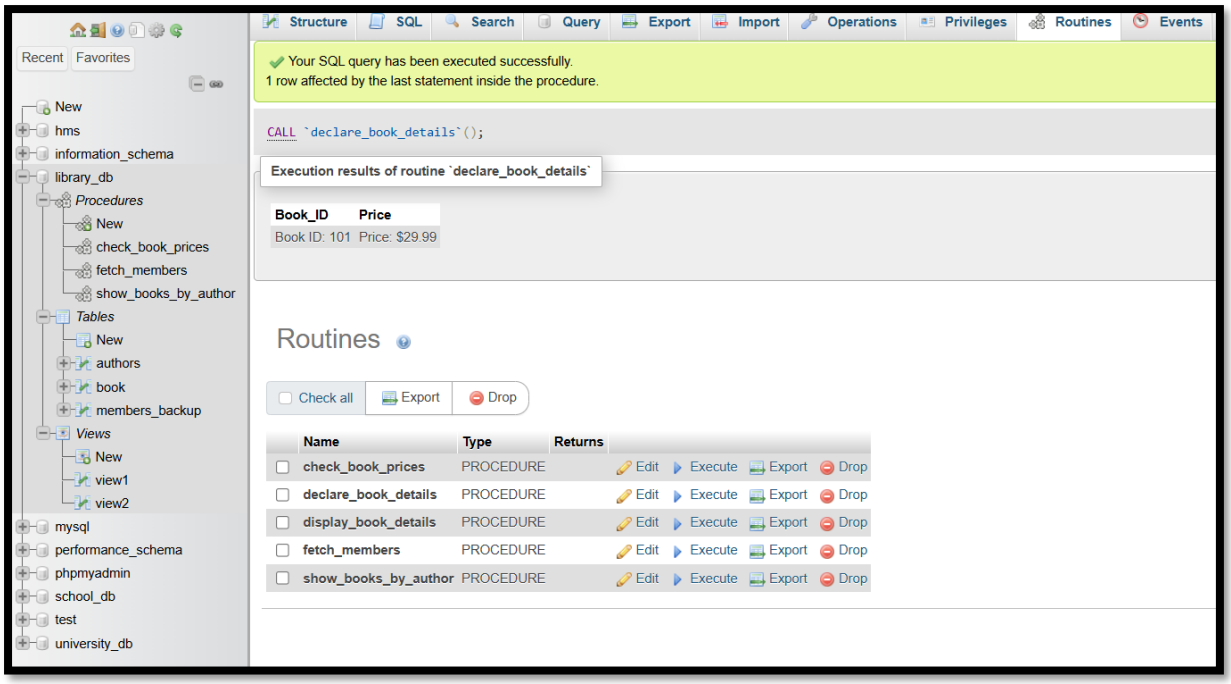
LAB EXERCISES:

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

Answer :

```
DELIMITER //

CREATE PROCEDURE declare_book_details()
BEGIN
    DECLARE v_book_id INT DEFAULT 101;
    DECLARE v_price DECIMAL(10, 2) DEFAULT 29.99;

    -- Display the values
    SELECT CONCAT('Book ID: ', v_book_id) AS Book_ID,
CONCAT('Price: $', v_price) AS Price;
END //
DELIMITER ;
```

Output :



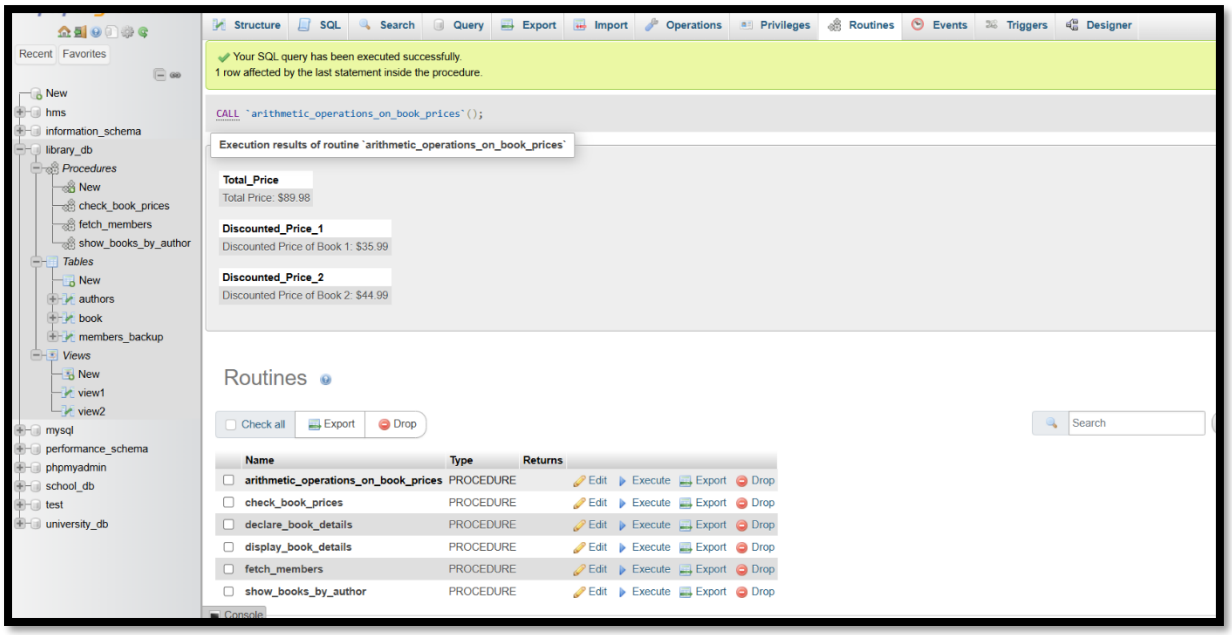
Lab 4: Write a PL/SQL block using constants and perform arithmetic operations on book prices.

Answer :

```
DELIMITER //

CREATE PROCEDURE arithmetic_operations_on_book_prices()
BEGIN
    DECLARE c_price1 DECIMAL(10, 2) DEFAULT 39.99;
    DECLARE c_price2 DECIMAL(10, 2) DEFAULT 49.99;
    DECLARE c_discount DECIMAL(5, 2) DEFAULT 0.10;
```

```
    DECLARE v_total_price DECIMAL(10, 2);
    DECLARE v_discounted_price1 DECIMAL(10, 2);
    DECLARE v_discounted_price2 DECIMAL(10, 2);

    SET v_total_price = c_price1 + c_price2;
    SET v_discounted_price1 = c_price1 - (c_price1 * c_discount);
    SET v_discounted_price2 = c_price2 - (c_price2 * c_discount);

    SELECT CONCAT('Total Price: $', v_total_price) AS Total_Price;
    SELECT CONCAT('Discounted Price of Book 1: $', v_discounted_price1) AS Discounted_Price_1;
    SELECT CONCAT('Discounted Price of Book 2: $', v_discounted_price2) AS Discounted_Price_2;
END //
DELIMITER ;
```

Output :



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.

Answer :

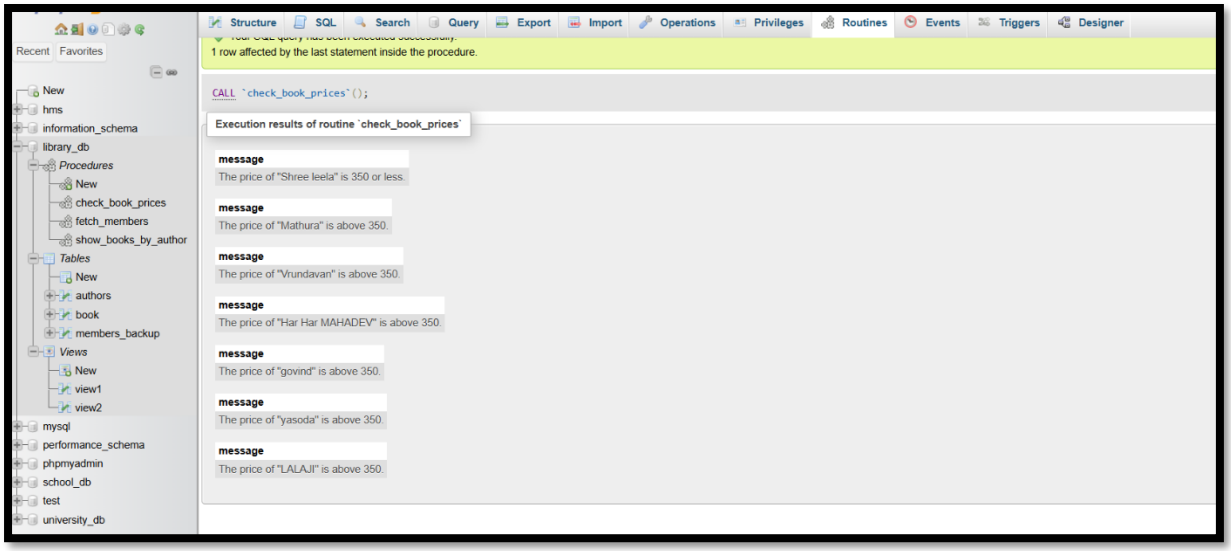
DELIMITER //

CREATE PROCEDURE check_book_prices()

```
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE v_title VARCHAR(100);
    DECLARE v_price DECIMAL(10, 2);

    DECLARE book_cursor CURSOR FOR
        SELECT title, price FROM book;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET
done = 1;

    OPEN book_cursor;

    read_loop: LOOP
        FETCH book_cursor INTO v_title, v_price;
        IF done THEN
            LEAVE read_loop;
        END IF;
        IF v_price > 350 THEN
            SELECT CONCAT('The price of "', v_title, '" is
above 350.') AS message;
```

```
        ELSE
            SELECT CONCAT('The price of "', v_title, '" is 350 or less.') AS message;
        END IF;
    END LOOP;

    CLOSE book_cursor;
END //
DELIMITER ;
```

Output :



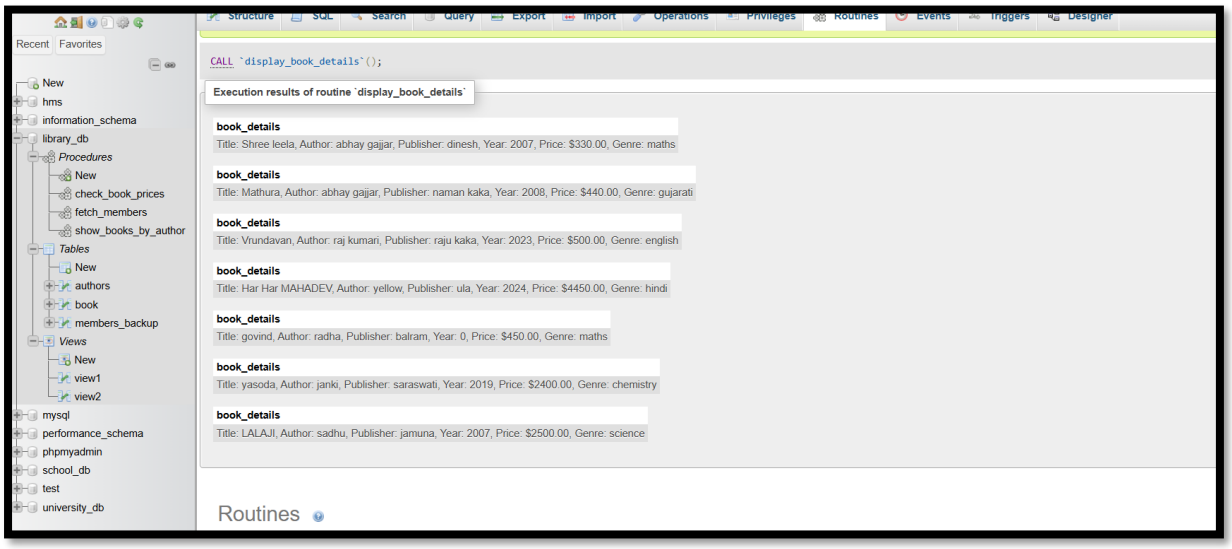
Lab 4: Use a FOR LOOP in PL/SQL to display the details of all books one by one.

Answer :

```
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE v_title VARCHAR(100);
    DECLARE v_author VARCHAR(100);
    DECLARE v_publisher VARCHAR(100);
    DECLARE v_year_of_publication INT;
    DECLARE v_price DECIMAL(10, 2);
    DECLARE v_genre VARCHAR(100);

    DECLARE book_cursor CURSOR FOR
        SELECT title, author, publisher, year_of_publication, price, genre FROM book;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN book_cursor;

    read_loop: LOOP
        FETCH book_cursor INTO v_title, v_author, v_publisher, v_year_of_publication, v_price, v_genre;
```

```
        IF done THEN
            LEAVE read_loop;
        END IF;
        SELECT CONCAT('Title: ', v_title, ', Author: ', v_author, ', Publisher: ', v_publisher, ', Year: ', v_year_of_publication, ', Price: $', v_price, ', Genre: ', v_genre) AS book_details;
    END LOOP;

    CLOSE book_cursor;
END
```

Output :



22.) SQL Cursors

LAB EXERCISES:

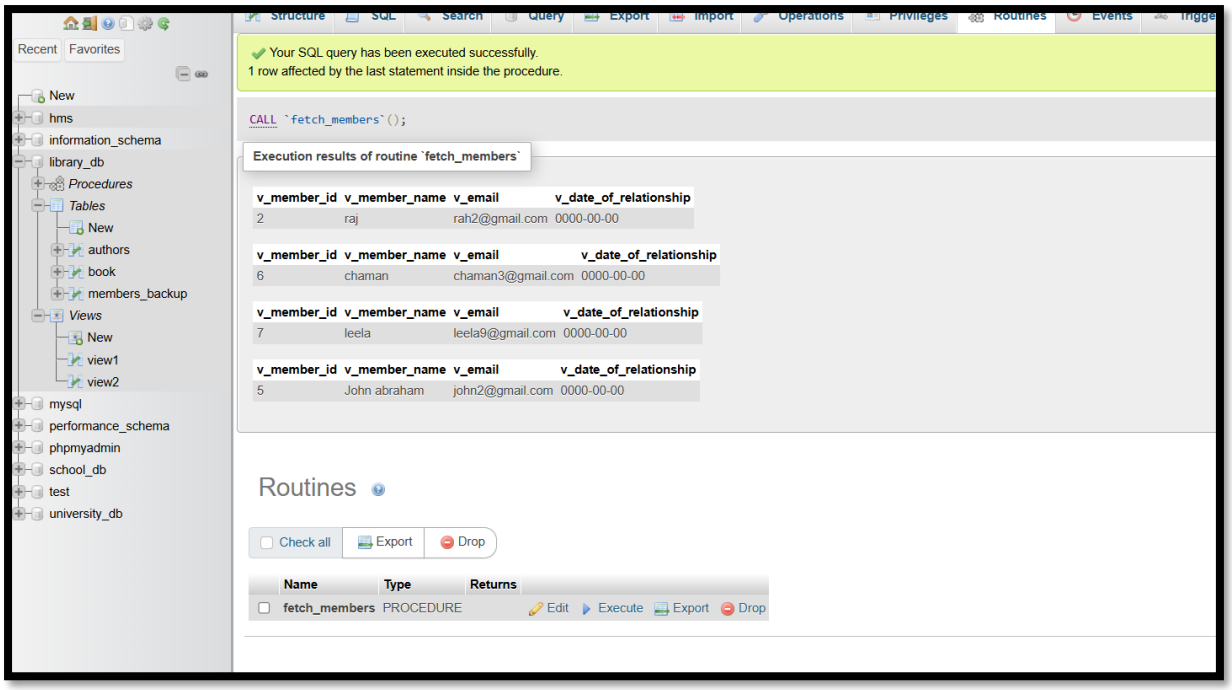
Lab 3: **Write a PL/SQL block using an explicit cursor to fetch and display all records from the members table.**

Answer :

```
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE member_id INT;
    DECLARE member_name VARCHAR(100);
    DECLARE email VARCHAR(100);
    DECLARE date_of_relationship DATE;

    DECLARE member_cursor CURSOR FOR
        SELECT member_id, member_name, email, date_of_relationship FROM members_backup;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN member_cursor;
```

```
    read_loop: LOOP
        FETCH member_cursor INTO member_id, member_name, email, date_of_relationship;
        IF done THEN
            LEAVE read_loop;
        END IF;
        SELECT member_id, member_name, email, date_of_relationship;
    END LOOP;

    CLOSE member_cursor;
END
```

Output :



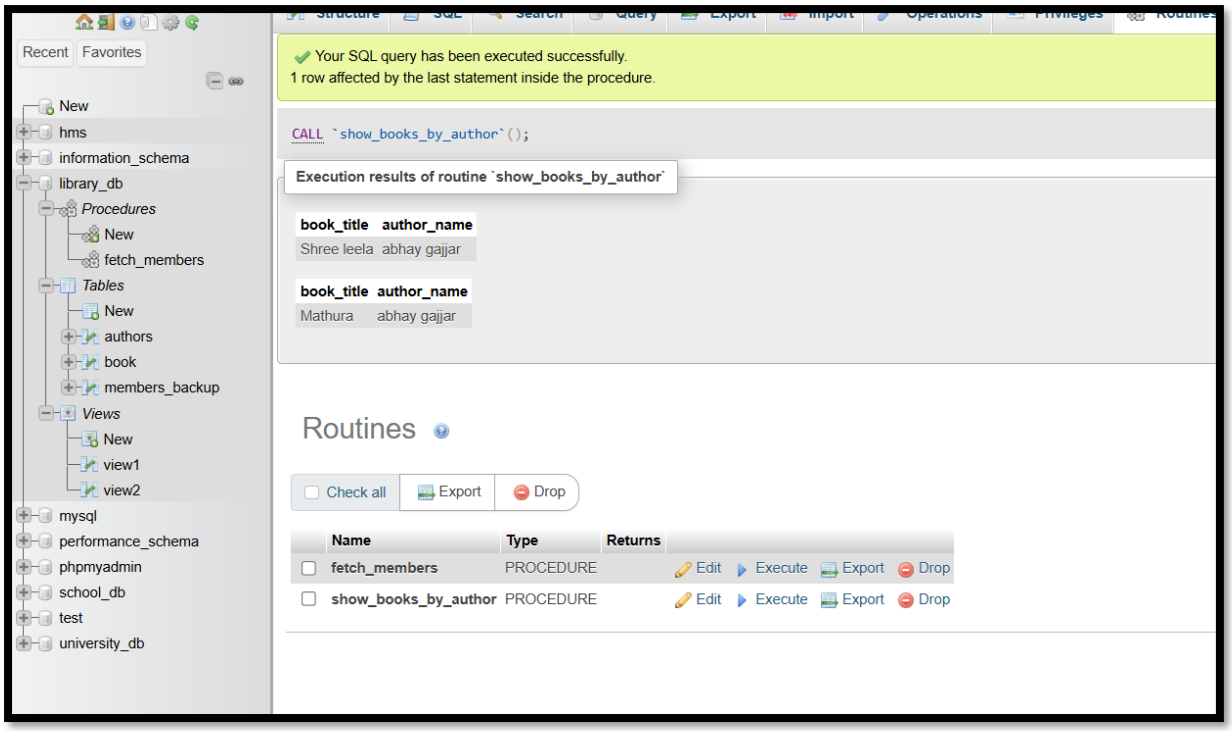
Lab 4: Create a cursor to retrieve books by a particular author and display their titles.

Answer :

```
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE book_title VARCHAR(100);
    DECLARE author_name VARCHAR(100);

    DECLARE book_cursor CURSOR FOR
        SELECT title, author FROM book WHERE author =
'abhay gajjar';
```

```
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET
done = 1;
    OPEN book_cursor;
    read_loop: LOOP
        FETCH book_cursor INTO book_title,
author_name;
        IF done THEN
            LEAVE read_loop;
        END IF;
        SELECT book_title, author_name;
    END LOOP;
    CLOSE book_cursor;
END
```

Output :



23.) Rollback and Commit Savepoint

LAB EXERCISES:

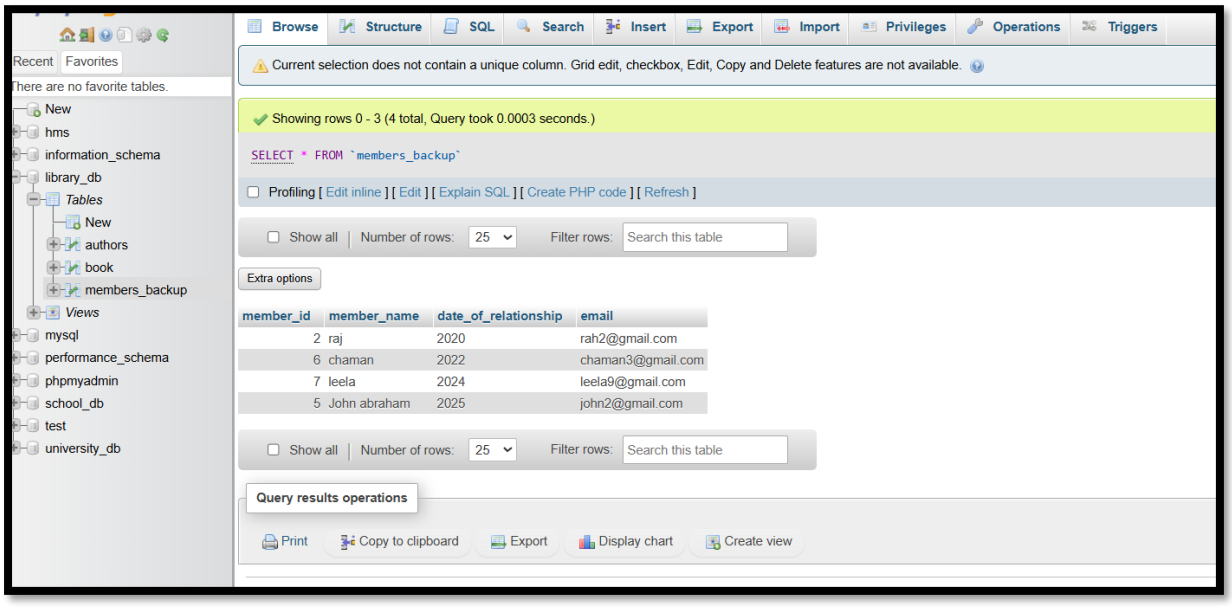
Lab 3: Perform a transaction that includes inserting a new member, setting a SAVEPOINT, and rolling back to the savepoint after making updates.

Answer :

START TRANSACTION;

```
INSERT INTO members_backup (member_id, member_name, email, date_of_relationship) VALUES (5, 'John abraham', 'john2@gmail.com', 2025);

SAVEPOINT before_update;

UPDATE members_backup

SET email ='grant8@gmail.com'

WHERE member_id = 1;

UPDATE members_backup

SET email = 'ola9@gmail.com'

WHERE member_id = 2;

ROLLBACK TO SAVEPOINT before_update;

COMMIT;
```

Output :

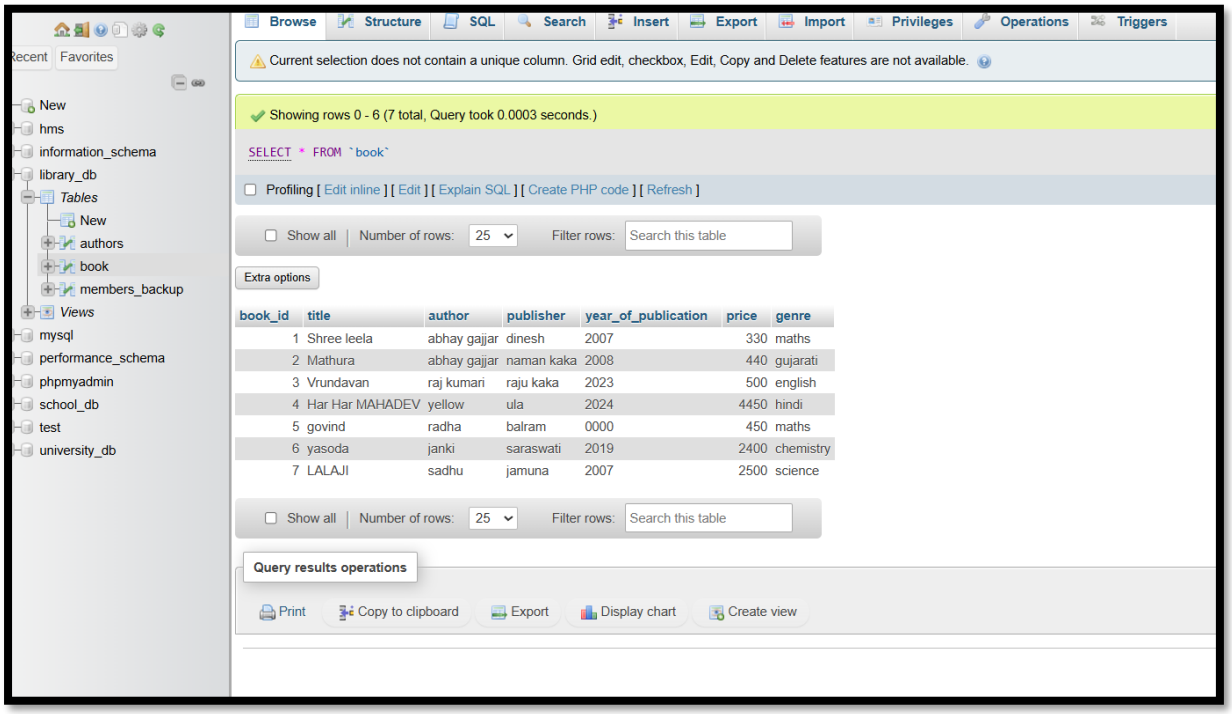


Lab 4: Use COMMIT after successfully inserting multiple books into the books table, then use ROLLBACK to undo a set of changes made after a savepoint.

Answer :

```
START TRANSACTION;

INSERT INTO book
VALUES(6,"yasoda","janki","saraswati",2019,2400,60),(
7,"LALAJI","sadhu","jamuna",2007,2500,70);

COMMIT;

SAVEPOINT BEFORE_update;

UPDATE book

SET price=300

WHERE book_id=1;

UPDATE book

SET price=500

WHERE book_id=2;

ROLLBACK TO SAVEPOINT BEFORE_update;

COMMIT;
```

Output :


Recent
Favorites
New
hms
information_schema
library_db
Tables
New
authors
book
members_backup
Views
mysql
performance_schema
phpmyadmin
school_db
test
university_db
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Triggers
Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.
Showing rows 0 - 6 (7 total, Query took 0.0003 seconds.)
SELECT * FROM `book`
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all | Number of rows: 25 Filter rows: Search this table
Extra options
book_id title author publisher year_of_publication price genre
1 Shree leela abhay gajjar dinesh 2007 330 maths
2 Mathura abhay gajjar naman kaka 2008 440 gujarati
3 Vrundavan raj kumari raju kaka 2023 500 english
4 Har Har MAHADEV yellow ula 2024 4450 hindi
5 govind radha balram 0000 450 maths
6 yasoda janki saraswati 2019 2400 chemistry
7 LALAJI sadhu jamuna 2007 2500 science
Show all | Number of rows: 25 Filter rows: Search this table
Query results operations
Print
Copy to clipboard
Export
Display chart
Create view