

Module 4 – Introduction to DBMS Introduction to SQL

Theory Questions:

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

SQL (Structured Query Language) is a standard language used to create, manage, and retrieve data from relational databases. It is essential because it allows users to efficiently store, retrieve, update, and manage large volumes of structured data.

2. Explain the difference between DBMS and RDBMS.

DBMS	RDBMS
Stores data as files	Stores data in tables (relations)
No relationship between data	Uses relationships via keys
Less secure	More secure
Examples: File system	Examples: MySQL, Oracle

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

SQL helps define database structure, insert/update/delete data, retrieve information, control access, and manage transactions in relational databases.

4. What are the key features of SQL?

Key features of SQL

- Easy to learn
- Standardized language
- Supports CRUD operations
- Secure and scalable
- Works with large databases

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

```
CREATE DATABASE school_db;
```

```
USE school_db;
```

```
CREATE TABLE students (
```

```
    student_id INT PRIMARY KEY,
```

```
    student_name VARCHAR(50),
```

```
    age INT,
```

```
    class VARCHAR(20),
```

```
    address VARCHAR(100)
```

```
);
```

The screenshot shows the phpMyAdmin interface with the following details:

- Left sidebar:** Shows the database tree with databases: New, information_schema, mysql, performance_schema, phpmyadmin, school_db, and test.
- Top menu:** Server: 127.0.0.1, Databases, SQL, Status, User accounts, Export, Import, Settings, Replication, Variables, More.
- SQL tab:**
 - Query 1: `CREATE DATABASE school_db;` Result: MySQL returned an empty result set (i.e. zero rows). (Query took 0.0021 seconds.) [Edit inline] [Edit] [Create PHP code]
 - Query 2: `USE school_db;` Result: Error: #1046 No database selected [Edit inline] [Edit] [Create PHP code]
 - Query 3: `CREATE TABLE students (student_id INT PRIMARY KEY, student_name VARCHAR(50), age INT, class VARCHAR(20), address VARCHAR(100));` Result: MySQL returned an empty result set (i.e. zero rows). (Query took 0.0167 seconds.) [Edit inline] [Edit] [Create PHP code]
- Status tab:** Shows various MySQL status metrics.
- User accounts tab:** Shows user management options.
- Export tab:** Options for exporting data.
- Import tab:** Options for importing data.
- Settings tab:** Database configuration options.
- Replication tab:** Replication settings.
- Variables tab:** MySQL system variables.
- More tab:** Additional administrative tools.

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

INSERT INTO students VALUES

```
(1, 'Sweta', 12, '6A', 'Delhi'),  
(2, 'Ankita', 11, '5B', 'Mumbai'),  
(3, 'Abhay', 13, '7A', 'Pune'),  
(4, 'Sayan', 10, '5A', 'Chennai'),  
(5, 'Tanjua', 14, '8B', 'Bangalore');
```

SELECT * FROM students;

The screenshot shows the phpMyAdmin interface for a MySQL database named 'school_db'. The 'students' table is selected. The SQL tab contains the following code:

```
INSERT INTO students VALUES (1, 'Sweta', 12, '6A', 'Delhi'), (2, 'Ankita', 11, '5B', 'Mumbai'), (3, 'Abhay', 13, '7A', 'Pune'), (4, 'Sayan', 10, '5A', 'Chennai'), (5, 'Tanjua', 14, '8B', 'Bangalore');
```

The results section shows a green success message: "Showing rows 0 - 4 (5 total, Query took 0.0003 seconds.)". Below it is the query:

```
SELECT * FROM students;
```

The data grid displays the following 5 rows:

	student_id	student_name	age	class	address
<input type="checkbox"/>	1	Sweta	12	6A	Delhi
<input type="checkbox"/>	2	Ankita	11	5B	Mumbai
<input type="checkbox"/>	3	Abhay	13	7A	Pune
<input type="checkbox"/>	4	Sayan	10	5A	Chennai
<input type="checkbox"/>	5	Tanjua	14	8B	Bangalore

2. SQL Syntax

The basic components of SQL syntax include:

- **Keywords** – Reserved words such as SELECT, FROM, WHERE, INSERT, UPDATE, DELETE
- **Identifiers** – Names of database objects like tables and columns
- **Clauses** – Parts of SQL statements that define specific actions (e.g., WHERE, ORDER BY)
- **Operators** – Used to compare or manipulate data (=, >, <, AND, OR)
- **Expressions** – Combinations of columns, operators, and values
- **Literals** – Fixed values such as numbers or strings (10, 'John')

2. Write the general structure of an SQL SELECT statement

SELECT column1, column2, ...

FROM table_name

WHERE condition

ORDER BY column_name;

(Note: WHERE and ORDER BY clauses are optional.)

3. Explain the role of clauses in SQL statements

Clauses define different parts of an SQL statement and control how data is retrieved or manipulated:

- **SELECT** – Specifies which columns to display
- **FROM** – Specifies the table containing the data
- **WHERE** – Filters records based on conditions
- **ORDER BY** – Sorts the result set
- **GROUP BY** – Groups rows with similar values
- **HAVING** – Filters grouped data

Each clause performs a specific function and together they form a complete SQL query.

LAB EXERCISES

Lab 1: Retrieve specific columns (student_name and age) from the students table

```
SELECT student_name, age
```

```
FROM students;
```

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
- Database:** school_db
- Table:** students
- Query Result:** The results of the query `SELECT student_name, age FROM students;` are displayed. The output shows 5 rows:

	student_name	age
1	Sweta	12
2	Ankita	11
3	Abhay	13
4	sayan	10
5	Tanuja	14

- Operations:** Buttons for Edit, Copy, Delete, and Export are available for each row.
- Navigation:** Buttons for Show all, Number of rows (set to 25), Filter rows, Search this table, and Sort by key (set to None) are present.
- Bottom Options:** Buttons for Check all, With selected:, Edit, Copy, Delete, and Export.

Lab 2: Retrieve all students whose age is greater than 10

```
SELECT *
```

```
FROM students
```

```
WHERE age > 10;
```

localhost/phpmyadmin/index.php?route=/table/sql&db=school_db&table=students

phpMyAdmin

Recent Favorites

Server: 127.0.0.1 Database: school_db Table: students

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

Show query box

Showing rows 0 - 3 (4 total, Query took 0.0004 seconds.)

SELECT * FROM students WHERE age > 10;

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	student_id	student_name	age	class	address
<input type="checkbox"/>	1	Sweta	12	6A	Delhi
<input type="checkbox"/>	2	Ankita	11	5B	Mumbai
<input type="checkbox"/>	3	Abhay	13	7A	Pune
<input type="checkbox"/>	5	Tanuja	14	8B	Bangalore

Check all With selected: Edit Copy Delete Export

Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Query results operations

3. SQL Constraints

Theory Questions

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

Constraints are rules applied to table columns to enforce data integrity and maintain accuracy in the database. Common types of constraints:

1. **PRIMARY KEY** – Uniquely identifies each row in a table. Cannot contain NULL values.
2. **FOREIGN KEY** – Ensures referential integrity by linking a column in one table to a primary key in another table.
3. **NOT NULL** – Ensures a column cannot have NULL values.
4. **UNIQUE** – Ensures all values in a column are unique.
5. **CHECK** – Ensures values in a column satisfy a specific condition (e.g., age > 0).
6. **DEFAULT** – Assigns a default value to a column if no value is specified during insertion.

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

Constraint	Purpose	Key Points
PRIMARY KEY	Uniquely identifies each row in its own table	Cannot be NULL; only one primary key per table
FOREIGN KEY	Maintains referential integrity by linking to a primary key in another table	Can have duplicate values; ensures consistency across tables

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

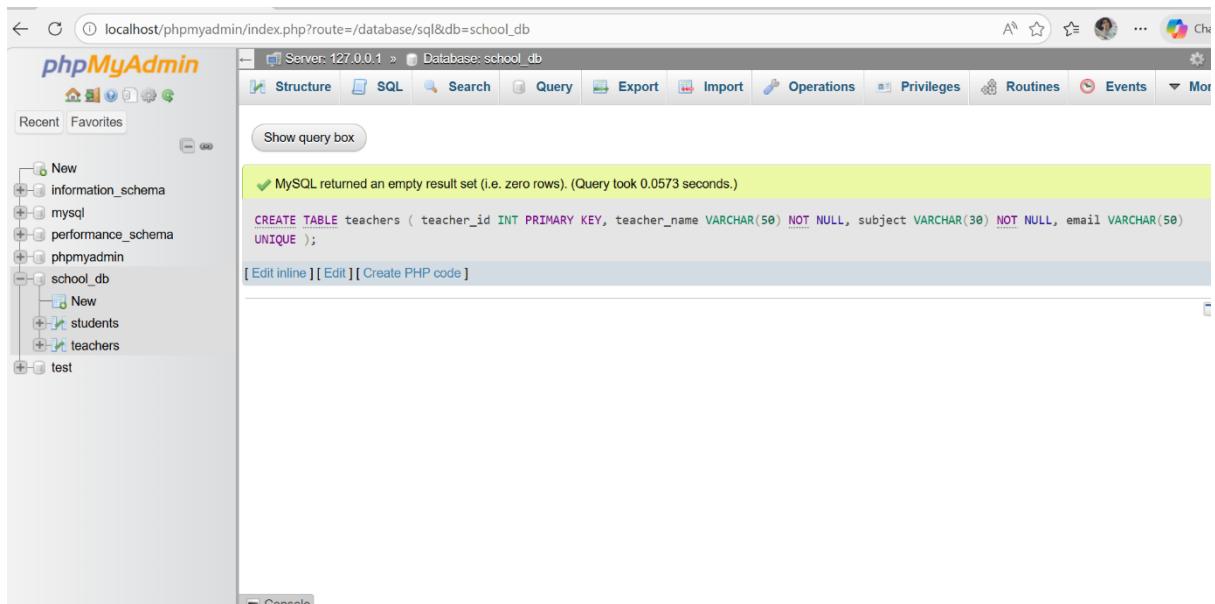
- **NOT NULL**: Ensures a column must always have a value; it cannot be left empty.

- **UNIQUE:** Ensures that all values in the column are distinct; no duplicates are allowed.

LAB EXERCISES

Lab 1: Create a teachers table with constraints

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



Lab 2: Implement a FOREIGN KEY constraint to relate teacher_id with students table

Assuming the students table has a teacher_id column to reference the teacher:

```
ALTER TABLE students
```

```
ADD teacher_id INT,
```

```
ADD CONSTRAINT fk_teacher
```

```
FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id);
```

localhost/phpmyadmin/index.php?route=/table/sql&db=school_db&table=teachers

phpMyAdmin

Recent Favorites

New information_schema mysql performance_schema phpmyadmin school_db New students teachers test

Show query box

MySQL returned an empty result set (i.e. zero rows). (Query took 0.1147 seconds.)

```
ALTER TABLE students ADD teacher_id INT, ADD FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id);
```

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

Console

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
- Database:** school_db
- Table:** teachers

The left sidebar shows the database structure with the following schema:

- information_schema
- mysql
- performance_schema
- phpmyadmin
- school_db** (selected):
 - New
 - students
 - teachers
- test

The main area displays the results of a MySQL query:

```
ALTER TABLE students ADD teacher_id INT, ADD FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id);
```

A message at the top indicates: "MySQL returned an empty result set (i.e. zero rows). (Query took 0.1147 seconds.)".

Below the message are three buttons: [Edit inline], [Edit], and [Create PHP code].

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

Theory Questions

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

Data Definition Language (DDL) is a subset of SQL used to define, modify, and manage the structure of database objects.

DDL commands deal with the **schema** of the database rather than the data itself.

Common DDL commands include:

- CREATE
- ALTER
- DROP
- TRUNCATE
- RENAME

2. Explain the CREATE command and its syntax.

The **CREATE** command is used to create new database objects such as databases, tables, views, and indexes.

General syntax for creating a table:

```
CREATE TABLE table_name (
    column1 datatype constraints,
    column2 datatype constraints,
    ...
);
```

General syntax for creating a database:

```
CREATE DATABASE database_name;
```

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

- **Data types** define the kind of data a column can store (e.g., INT, VARCHAR, DATE) and help maintain data accuracy.
- **Constraints** enforce rules on the data to ensure integrity and consistency (e.g., PRIMARY KEY, NOT NULL, UNIQUE).

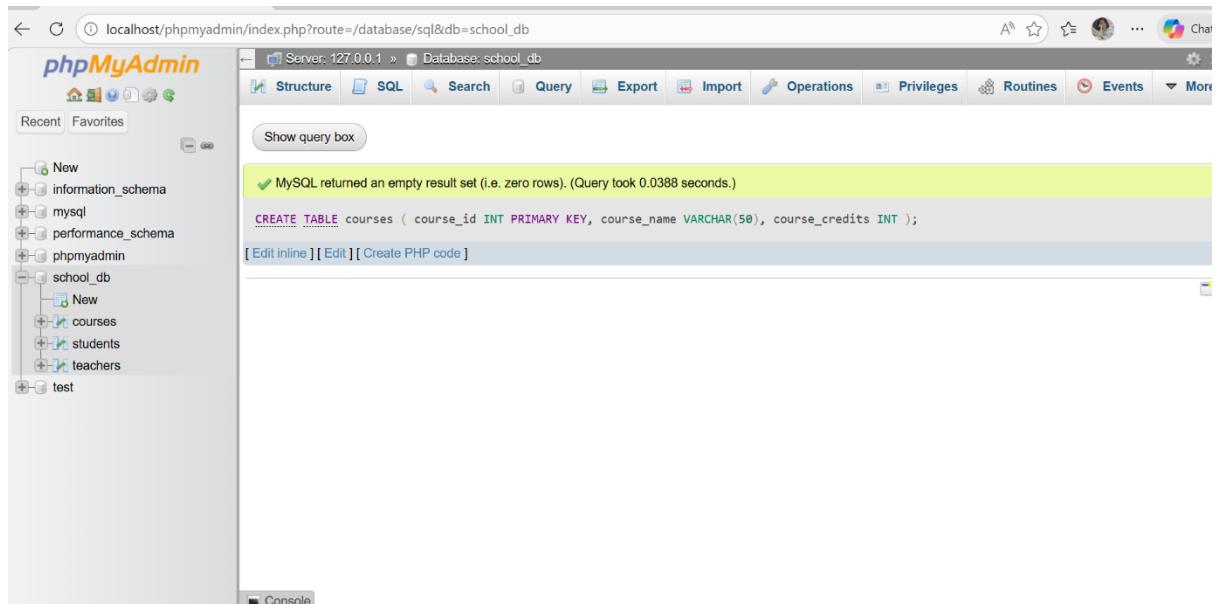
Together, they:

- Prevent invalid data entry
- Improve data reliability
- Enforce business rules at the database level

LAB EXERCISES

Lab 1: Create a courses table

```
CREATE TABLE courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100),
    course_credits INT
);
```



Lab 2: Create a database named university_db

```
CREATE DATABASE university_db;
```

localhost/phpmyadmin/index.php?route=server/sql

phpMyAdmin

Recent Favorites

Server: 127.0.0.1

Databases SQL Status User accounts Export Import Settings Replication Variables More

Show query box

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0012 seconds.)

CREATE DATABASE university_db;

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

Error: #1046 No database selected

New

information_schema

mysql

performance_schema

phpmyadmin

school_db

New

courses

students

teachers

test

university_db

The screenshot shows the phpMyAdmin interface for MySQL. On the left, there's a tree view of databases: New, information_schema, mysql, performance_schema, phpmyadmin, school_db (which contains New, courses, students, teachers), test, and university_db. The university_db database is currently selected. The main panel has a toolbar at the top with tabs for Databases, SQL, Status, User accounts, Export, Import, Settings, Replication, Variables, and More. Below the toolbar, a message says "MySQL returned an empty result set (i.e. zero rows). (Query took 0.0012 seconds.)". A query box contains "CREATE DATABASE university_db;". Below the query box, a warning message says "Error: #1046 No database selected". There are buttons for Edit inline, Edit, and Create PHP code.

ALTER COMMAND

Theory Questions

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

The **ALTER** command is used to **change the structure of an existing table** in a database.

It allows you to:

- Add new columns
- Modify existing columns
- Drop columns
- Rename columns or tables (in some DBMS) It does **not** change the data inside the table directly, only the table structure.

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

Add a column

```
ALTER TABLE table_name
```

```
ADD column_name datatype;
```

Modify a column

```
ALTER TABLE table_name
```

```
MODIFY column_name new_datatype;
```

Drop a column

```
ALTER TABLE table_name
```

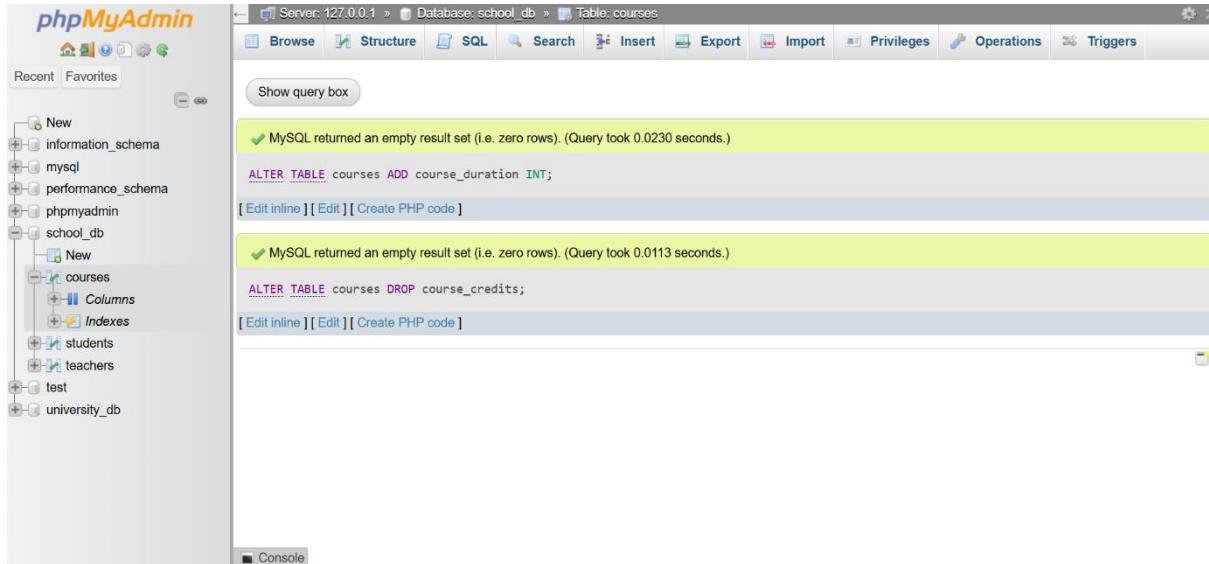
```
DROP column_name;
```

LAB EXERCISES

Lab 1: Add a column course_duration to the courses table

ALTER TABLE courses

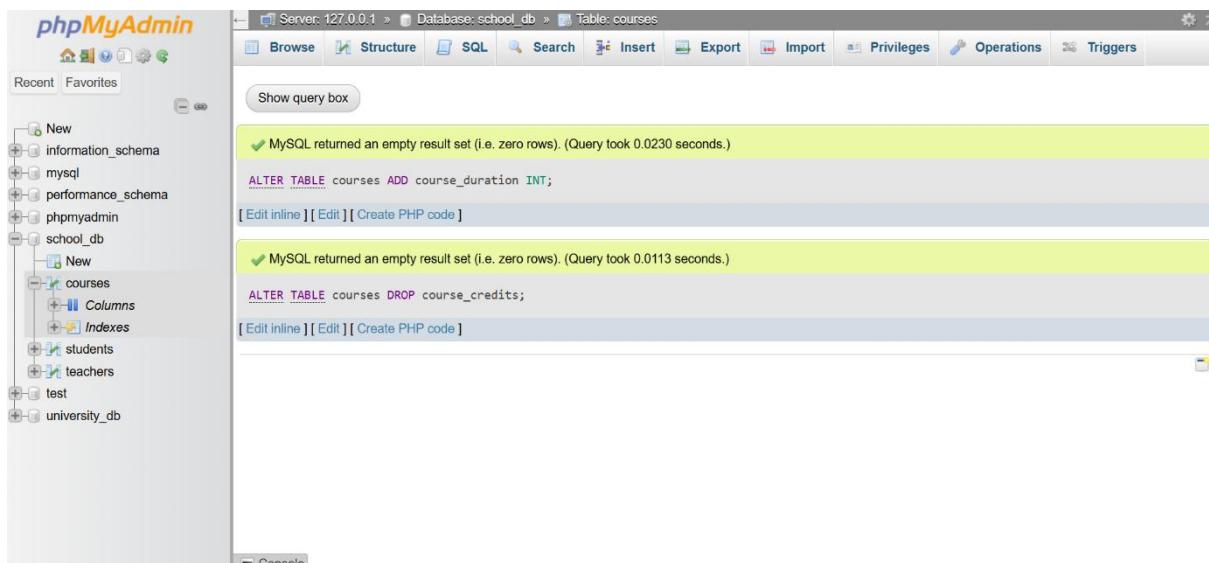
ADD course_duration INT;



Lab 2: Drop the course_credits column from the courses table

ALTER TABLE courses

DROP course_credits;



6. DROP Command

Theory Questions

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

The **DROP** command is used to **permanently delete database objects** such as:

- Tables
- Databases
- Views
- Indexes

When a table is dropped, **both the table structure and all its data are removed permanently** from the database.

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

- All **records stored in the table are permanently deleted**
- The **table structure is removed**
- Any **indexes, triggers, and constraints** associated with the table are also deleted
- The action **cannot be rolled back** (unless backups exist)
- Any queries or applications depending on the table may stop working

LAB EXERCISES

Lab 1: Drop the teachers table from the school_db database

USE school_db;

DROP TABLE teachers;

Lab 2: Drop the students table from the school_db database and verify its removal

Step 1: Drop the table

USE school_db;

DROP TABLE students;

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The left sidebar lists databases: New, information_schema, mysql, performance_schema, phpmyadmin, school_db, test, and university_db. The 'school_db' database is selected. The main panel shows the 'Table: students' page. The SQL query history at the bottom shows two queries:

```
USE school_db;  
DROP TABLE students;
```

Both queries have a green checkmark indicating they were successful. The first query took 0.0001 seconds and the second took 0.0125 seconds.

Step 2: Verify that the table has been removed

SHOW TABLES;

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The left sidebar lists databases: New, information_schema, mysql, performance_schema, phpmyadmin, school_db, test, and university_db. The 'school_db' database is selected. The main panel shows the 'Tables_in_school_db' view. The table names listed are 'courses' and 'teachers'. There is a warning message above the table list: "Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available." Below the table list, there are options to "Print", "Copy to clipboard", and "Create view".

7. Data Manipulation Language (DML)

Theory Questions

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

INSERT

The **INSERT** command is used to **add new records (rows)** into a table.

```
INSERT INTO table_name VALUES (...);
```

UPDATE

The **UPDATE** command is used to **modify existing records** in a table.

```
UPDATE table_name SET column = value WHERE condition;
```

DELETE

The **DELETE** command is used to **remove records** from a table.

```
DELETE FROM table_name WHERE condition;
```

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

The **WHERE clause** is crucial because:

- It specifies **which rows should be updated or deleted**
- Prevents **accidental modification or deletion of all records**
- Ensures **data accuracy and safety**

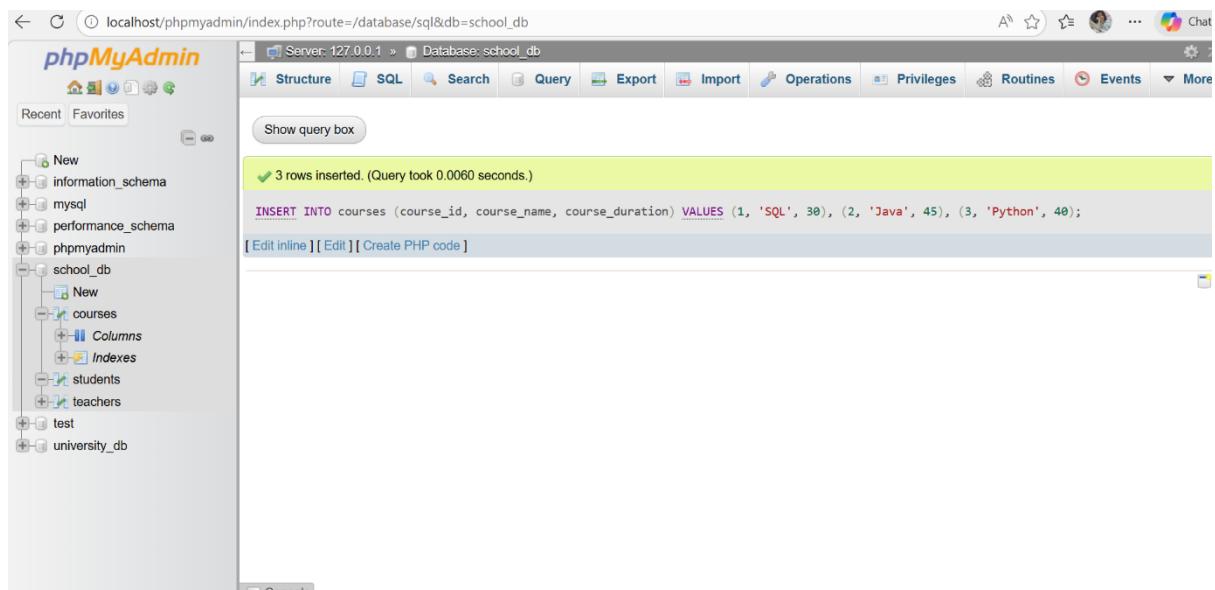
LAB EXERCISES

Lab 1: Insert three records into the courses table

INSERT INTO courses (course_id, course_name, course_duration)

VALUES

```
(1, 'SQL', 30),  
(2, 'Java', 45),  
(3, 'Python', 40);
```



Lab 2: Update the course duration of a specific course

UPDATE courses

```
SET course_duration = 50
```

```
WHERE course_id = 2;
```

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The left sidebar lists databases: New, information_schema, mysql, performance_schema, phpmyadmin, school_db, test, and university_db. The 'courses' table under 'school_db' is selected. The main area displays the results of an UPDATE query:

```
UPDATE courses SET course_duration = 50 WHERE course_id = 2;
```

The status bar at the bottom indicates "1 row affected. (Query took 0.0070 seconds.)".

Lab 3: Delete a course with a specific course_id

DELETE FROM courses

WHERE course_id = 3;

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The left sidebar lists databases: New, information_schema, mysql, performance_schema, phpmyadmin, school_db, test, and university_db. The 'courses' table under 'school_db' is selected. The main area displays the results of a DELETE query:

```
DELETE FROM courses WHERE course_id = 3;
```

The status bar at the bottom indicates "1 row deleted. (Query took 0.0040 seconds.)".

8. Data Query Language (DQL)

Theory Questions

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

The **SELECT** statement is used to **retrieve data from one or more tables** in a database.

It allows users to:

- Fetch all or specific columns
- Apply conditions
- Sort results
- Limit the number of records displayed

Basic syntax:

```
SELECT column_name(s)
```

```
FROM table_name;
```

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

WHERE Clause

- Used to **filter records** based on conditions
- Returns only rows that satisfy the given condition

```
SELECT * FROM courses
```

```
WHERE course_duration > 5;
```

ORDER BY Clause

- Used to **sort the result set**

- Can be sorted in **ascending (ASC)** or **descending (DESC)** order

`SELECT * FROM courses`

`ORDER BY course_duration DESC;`

LAB EXERCISES

Lab 1: Retrieve all courses from the courses table

`SELECT * FROM courses;`

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. The left sidebar shows the database structure with a 'courses' table under the 'New' schema. The main area displays the results of the query `SELECT * FROM courses;`. The results table has columns: course_id, course_name, and course_duration. There are two rows: one for SQL with a duration of 30, and another for Java with a duration of 50.

course_id	course_name	course_duration
1	SQL	30
2	Java	50

Lab 2: Sort the courses based on course_duration in descending order

`SELECT * FROM courses`

`ORDER BY course_duration DESC;`

The screenshot shows the phpMyAdmin interface for the 'courses' table in the 'school_db' database. The table has three columns: course_id, course_name, and course_duration. The data is ordered by course_duration in descending order (DESC). The first row is Java with a duration of 50, and the second row is SQL with a duration of 30.

course_id	course_name	course_duration
2	Java	50
1	SQL	30

Lab 3: Show only the top two courses using LIMIT

SELECT * FROM courses

LIMIT 2;

The screenshot shows the phpMyAdmin interface for the 'courses' table in the 'school_db' database. The table has three columns: course_id, course_name, and course_duration. The data is ordered by course_duration in descending order (DESC). The query 'SELECT * FROM courses ORDER BY course_duration DESC LIMIT 2;' has been run, resulting in the same two rows: Java (course_id 2, duration 50) and SQL (course_id 1, duration 30).

course_id	course_name	course_duration
2	Java	50
1	SQL	30

9. Data Control Language (DCL)

Theory Questions

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

GRANT

- Used to **give permissions (privileges)** to users or roles
- Controls access to database objects like tables, views, and procedures

REVOKE

- Used to **remove previously granted permissions**
- Helps maintain database security

2. How do you manage privileges using these commands?

Privileges are managed by:

- Assigning specific permissions (SELECT, INSERT, UPDATE, DELETE)
- Granting access only to required users
- Revoking permissions when they are no longer needed

Example:

GRANT SELECT ON courses TO user1;

REVOKE SELECT ON courses FROM user1;

LAB EXERCISES

Lab 1: Create two users and grant SELECT permission to user1

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

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

```
GRANT SELECT ON courses TO 'user1'@'localhost';
```

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
- Database:** school_db
- Table:** courses
- Toolbar:** Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, Triggers.
- Left sidebar:** Shows databases like information_schema, mysql, performance_schema, phpmyadmin, school_db, and tables courses, students, teachers, test, university_db.
- SQL History:** Three queries are listed:
 - CREATE USER 'user1' IDENTIFIED BY 'password1';
 - CREATE USER 'user2' IDENTIFIED BY 'password2';
 - GRANT SELECT ON courses TO 'user1';
- Messages:** Each query has a green success message indicating zero rows affected.

Lab 2: Revoke INSERT permission from user1 and grant it to user2

Revoke INSERT from user1

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

Grant INSERT to user2

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

phpMyAdmin

Server: 127.0.0.1 » Database: school_db » Table: courses

Recent Favorites

New

information_schema

mysql

performance_schema

phpmyadmin

school_db

New

courses

Columns

Indexes

students

teachers

test

university_db

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

Show query box

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0027 seconds.)

REVOKE INSERT ON courses FROM user1;

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

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0026 seconds.)

GRANT INSERT ON courses TO user2;

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

Console

The screenshot shows the phpMyAdmin interface. On the left, the database schema is displayed under the 'school_db' database, including tables like 'courses', 'students', and 'teachers'. The 'courses' table is selected. The main area shows two SQL queries. The first query, 'REVOKE INSERT ON courses FROM user1;', was successful, as indicated by the green message 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0027 seconds.)'. The second query, 'GRANT INSERT ON courses TO user2;', was also successful, indicated by another green message 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0026 seconds.)'. There are buttons for 'Edit inline', 'Edit', and 'Create PHP code' for each query.

Transaction Control Language (TCL)

Theory Questions

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

COMMIT

- Permanently **saves all changes** made during the current transaction
- After COMMIT, changes cannot be undone

ROLLBACK

- **Undoes changes** made during the current transaction
- Restores the database to its last committed state

2. Explain how transactions are managed in SQL databases

A **transaction** is a sequence of one or more SQL statements treated as a **single unit of work**.

Transactions follow the **ACID properties**:

- **Atomicity** – All operations succeed or none do
- **Consistency** – Database remains valid before and after transaction
- **Isolation** – Transactions do not interfere with each other
- **Durability** – Committed data is permanently saved

Transactions are managed using:

- BEGIN / START TRANSACTION
- COMMIT
- ROLLBACK
- SAVEPOINT

LAB EXERCISES

Lab 1: Insert rows and use COMMIT to save changes

START TRANSACTION;

INSERT INTO courses (course_id, course_name, course_duration)

VALUES

(201, 'Artificial Intelligence', 10),
(202, 'Machine Learning', 9);

COMMIT;

Lab 2: Insert rows and use ROLLBACK to undo changes

START TRANSACTION;

INSERT INTO courses (course_id, course_name, course_duration)

VALUES

(203, 'Blockchain', 6);

ROLLBACK;

Lab 3: Use SAVEPOINT to roll back specific changes

START TRANSACTION;

UPDATE courses

SET course_duration = 12

WHERE course_id = 201;

```
SAVEPOINT before_second_update;
```

```
UPDATE courses
```

```
SET course_duration = 11
```

```
WHERE course_id = 202;
```

```
ROLLBACK TO before_second_update;
```

```
COMMIT;
```

11. SQL Joins

Theory Questions

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

A **JOIN** in SQL is used to **combine rows from two or more tables** based on a **related column** (usually a primary key–foreign key relationship).

Types of JOINS

JOIN Type	Description
INNER JOIN	Returns only the rows that have matching values in both tables
LEFT JOIN (LEFT OUTER JOIN)	Returns all rows from the left table and matching rows from the right table; unmatched rows show NULL
RIGHT JOIN (RIGHT OUTER JOIN)	Returns all rows from the right table and matching rows from the left table
FULL OUTER JOIN	Returns all rows from both tables ; unmatched rows contain NULL values

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

Joins combine data by:

- Matching a **common column** between tables
- Allowing queries to retrieve related information stored across different tables
- Reducing data redundancy by following database normalization principles

Example:

```
SELECT e.name, d.department_name
```

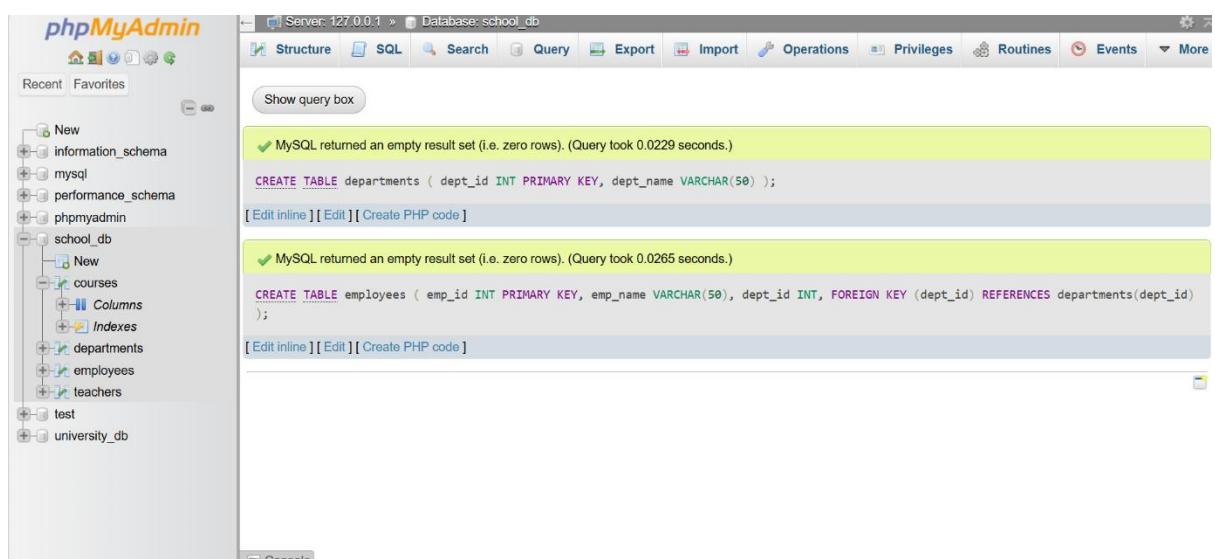
```
FROM employees e
JOIN departments d
ON e.department_id = d.department_id;
```

LAB EXERCISES

Lab 1: Create tables and perform an INNER JOIN

Create departments table

```
CREATE TABLE departments (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR(50)
);
```



Create employees table

```
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50),
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
);
```

Insert sample data

INSERT INTO departments VALUES

```
(1, 'CSE'),  
(2, 'IT'),
```

INSERT INTO employees VALUES

```
(1, 'Sweta', 1),  
(2, 'Brijesh', 2),
```

INNER JOIN query

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

The screenshot shows the phpMyAdmin interface with the following details:

- Left Panel:** Shows the database structure with databases like information_schema, mysql, performance_schema, phpmyadmin, school_db, test, and university_db.
- Top Bar:** Shows the URL as "localhost/phpmyadmin/index.php?route=/table/sql&db=school_db&table=employees".
- Header:** Shows the current selection is "school_db" and the table is "employees".
- Toolbar:** Includes buttons for Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, and Triggers.
- Message Bar:** A warning message: "Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available." with a link to "Edit".
- Query Result Area:** Shows the result of the executed query:

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

emp_name	dept_name
brijesh	IT
sweta	CSE
- Bottom Bar:** Includes buttons for Print, Copy to clipboard, Export, Display chart, and Create view.

Lab 2: Use LEFT JOIN to show all departments

```
SELECT d.dept_name, e.emp_name
```

```
FROM departments d
```

```
LEFT JOIN employees e
```

```
ON d.dept_id = e.dept_id;
```

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. The left sidebar lists various databases and their tables. In the main area, a query has been run:

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

The results show two rows of data:

dept_name	emp_name
IT	brijesh
CSE	sweta

Below the results, there are buttons for 'Print', 'Copy to clipboard', 'Export', 'Display chart', and 'Create view'.

12. SQL GROUP BY

Theory Questions

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

The **GROUP BY** clause is used to **group rows that have the same values** in one or more columns.

It is commonly used with **aggregate functions** to perform calculations on each group.

Common aggregate functions:

- COUNT() – counts rows
- SUM() – calculates total
- AVG() – calculates average
- MIN() – finds minimum value
- MAX() – finds maximum value

Example:

```
SELECT dept_id, COUNT(*)
```

```
FROM employees
```

```
GROUP BY dept_id;
```

2. Explain the difference between GROUP BY and ORDER BY

GROUP BY

ORDER BY

Groups rows into summary rows Sorts rows in the result set

Used with aggregate functions Used for display ordering

Reduces number of rows Does not change row count

Works before ORDER BY Executes after GROUP BY

LAB EXERCISES

Lab 1: Count employees in each department

```
SELECT dept_id, COUNT(emp_id) AS total_employees  
FROM employees  
GROUP BY dept_id;
```

The screenshot shows a MySQL query results interface. At the top, a green bar indicates "Showing rows 0 - 1 (2 total, Query took 0.0003 seconds.)". Below this is the SQL query: "SELECT dept_id, COUNT(emp_id) AS total_employees FROM employees GROUP BY dept_id;". A toolbar below the query includes options for Profiling, Edit inline, Edit, Explain SQL, Create PHP code, and Refresh. The main area displays a table with two rows:

dept_id	total_employees
1	1
2	1

Below the table are additional filtering and search controls. A "Query results operations" toolbar at the bottom right includes Print, Copy to clipboard, Export, Display chart, and Create view. A "Console" button is located at the bottom left.

Lab 2: Find the average salary in each department

```
SELECT dept_id, AVG(salary) AS average_salary  
FROM employees  
GROUP BY dept_id;
```

localhost/phpmyadmin/index.php?route=/table/sql&db=school_db&table=employees

phpMyAdmin

Recent Favorites

New information_schema mysql performance_schema phpmyadmin school_db New courses departments employees teachers test university_db

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

Showing rows 0 - 1 (total, Query took 0.0004 seconds.)

SELECT dept_id, AVG(salary) AS average_salary FROM employees GROUP BY dept_id;

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all Number of rows: 25 Filter rows: Search this table

Extra options

dept_id	average_salary
1	25000.0000
2	35000.0000

Check all With selected: Edit Copy Delete Export

Show all Number of rows: 25 Filter rows: Search this table

Query results operations

Print Copy to clipboard Export Display chart Create view

Console

The screenshot shows the phpMyAdmin interface for a MySQL database named 'school_db'. The current table is 'employees'. The interface includes a sidebar with database and schema navigation, a top menu with various administration functions, and a bottom section for query operations like printing or exporting the results.

The table data is as follows:

dept_id	average_salary
1	25000.0000
2	35000.0000

13. SQL Stored Procedure

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

A stored procedure is a precompiled set of one or more SQL statements stored in the database and executed as a single unit.

Difference between Stored Procedure and SQL Query:

Stored Procedure	SQL Query
Stored in the database	Written and executed on the fly
Can accept parameters	Parameters are limited
Can contain multiple SQL statements	Usually a single statement
Improves performance	Executed every time
Can include logic (IF, LOOP)	No control statements

2. Explain the advantages of using stored procedures.

Advantages of Stored Procedures:

1. **Improved performance** (precompiled)
2. **Reusability** – write once, use many times
3. **Security** – direct table access can be restricted
4. **Reduced network traffic**
5. **Better maintainability**
6. **Supports programming logic** (conditions, loops)

LAB EXERCISES

Assumed Table Structures

employees (

 emp_id INT,

```
    emp_name VARCHAR(100),  
    department VARCHAR(50),  
    salary DECIMAL(10,2)  
);
```

```
courses (  
    course_id INT,  
    course_name VARCHAR(100),  
    duration INT,  
    fees DECIMAL(10,2)  
);
```

Lab 1: Stored Procedure to Retrieve Employees by Department

Stored Procedure

```
DELIMITER //
```

```
CREATE PROCEDURE GetEmployeesByDepartment (
```

```
    IN dept_name VARCHAR(50)
```

```
)
```

```
BEGIN
```

```
    SELECT *
```

```
    FROM employees
```

```
    WHERE department = dept_name;
```

```
END //
```

```
DELIMITER ;
```

Execution

```
CALL GetEmployeesByDepartment('HR');
```

Lab 2: Stored Procedure to Return Course Details Using course_id

Stored Procedure

```
DELIMITER //
```

```
CREATE PROCEDURE GetCourseDetails (
```

```
    IN c_id INT
```

```
)
```

```
BEGIN
```

```
SELECT *
FROM courses
WHERE course_id = c_id;
END //
```

DELIMITER ;

Execution

```
CALL GetCourseDetails(201);
```

SQL view

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

A **view** in SQL is a **virtual table** that is created using a **SELECT query**. It does not store data itself; it displays data from one or more tables.

Difference between View and Table

View	Table
Virtual table	Physical table
Does not store data	Stores data permanently
Created using SELECT query	Created using CREATE TABLE
Uses less storage	Requires storage
Data is always up-to-date	Data stored independently

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

Advantages of Views:

- 1. Data security** – restrict access to sensitive columns
- 2. Simplifies complex queries**
- 3. Provides data abstraction**
- 4. Improves query consistency**
- 5. Easy maintenance**
- 6. Reusable queries**

LAB EXERCISES

Assumed Table Structures

```
employees (
    emp_id INT,
    emp_name VARCHAR(100),
    salary DECIMAL(10,2),
    dept_id INT
);

departments (
    dept_id INT,
    dept_name VARCHAR(100)
);
```

Lab 1: Create a View Showing Employees with Department Names

Create View

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

Use the View

```
SELECT * FROM emp_department_view;
```

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The left sidebar lists databases (information_schema, mysql, performance_schema, phpmyadmin, school_db, test, university_db) and their structures (Tables and Views). The 'school_db' database is selected, and its 'Tables' section is expanded, showing 'departments', 'employees', 'teachers', and 'emp_department_view'. The 'Views' section is also expanded, showing 'New' and 'emp_department_view'. The 'emp_department_view' view is selected, and its data is displayed in a grid. The grid has columns: emp_id, emp_name, salary, dept_name. The data is as follows:

emp_id	emp_name	salary	dept_name
1	brijesh	25000	IT
2	sweta	35000	CSE
3	Ankita	25000	CSE
4	Tanuja	50000	IT
5	Shankar	67000	IT

Lab 2: Modify the View to Exclude Employees with Salary Below \$50,000

Modify View (Using CREATE OR REPLACE)

CREATE OR REPLACE VIEW emp_department_view AS

SELECT

```
e.emp_id,  
e.emp_name,  
e.salary,  
d.dept_name
```

FROM employees e

JOIN departments d

ON e.dept_id = d.dept_id

WHERE e.salary >= 50000;

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. On the left, the database structure is visible, including the 'Tables' and 'Views' sections. In the main SQL tab, a message indicates an empty result set. Below it, the SQL query for creating the view is displayed:

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

Below the SQL area are buttons for 'Edit inline', 'Edit', and 'Create PHP code'.

phpMyAdmin

Server: 127.0.0.1 > Database: school_db > View: emp_department_view

Browse Structure SQL Search Insert Export Privileges Operations

Showing rows 0 - 1 (2 total, Query took 0.0005 seconds.)

SELECT * FROM `emp_department_view`

Profile [Edit inline] [Explain SQL] [Create PHP code] [Refresh]

Show all Number of rows: 25 Filter rows: Search this table

Extra options

	emp_id	emp_name	salary	dept_name
<input type="checkbox"/>	4	Tanuja	50000	IT
<input type="checkbox"/>	5	Shankar	67000	IT

Check all With selected: Edit Copy Delete Export

Show all Number of rows: 25 Filter rows: Search this table

Query results operations

Console Copy to clipboard Export Display chart Create view

New information_schema mysql performance_schema phpmyadmin school_db test university_db

Tables

courses departments employees teachers

Views

emp_department_view

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. The current view is 'emp_department_view'. The table has four columns: emp_id, emp_name, salary, and dept_name. There are two rows of data: one for employee ID 4, name Tanuja, salary 50000, and department IT; and another for employee ID 5, name Shankar, salary 67000, and department IT. The interface includes a sidebar with database and schema navigation, and various management tools like Browse, Structure, SQL, and Export.

	emp_id	emp_name	salary	dept_name
<input type="checkbox"/>	4	Tanuja	50000	IT
<input type="checkbox"/>	5	Shankar	67000	IT

TRIGGER IN SQL

Theory Questions

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

A **trigger** in SQL is a **database object** that automatically executes (fires) a set of SQL statements **in response to specific events** on a table or view.

Types of Triggers

Based on Timing

1. BEFORE Trigger

- Executes **before** INSERT, UPDATE, or DELETE
- Used for validation or modifying data before saving

2. AFTER Trigger

- Executes **after** INSERT, UPDATE, or DELETE
- Used for logging or auditing

Based on Event

1. **INSERT Trigger** – fires when a new record is added
2. **UPDATE Trigger** – fires when a record is modified
3. **DELETE Trigger** – fires when a record is removed

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

Trigger Type	Fires When	Common Usage
INSERT	New row is added	Logging new records
UPDATE	Existing row is modified	Tracking changes
DELETE	Row is removed	Backup or audit

LAB EXERCISES

Assumed Table Structures

```
employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(100),
    department VARCHAR(50),
    salary DECIMAL(10,2),
    last_modified TIMESTAMP
);

employee_log (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    emp_id INT,
    action VARCHAR(50),
    action_time TIMESTAMP
);
```

Lab 1: Trigger to Log When a New Employee Is Added

```
CREATE TABLE employee_log
(
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    emp_id INT,
    emp_name VARCHAR(200),
    dept_id INT,
    action_time DATETIME,
    action_description VARCHAR(200)
);

DELIMITER $$

CREATE TRIGGER employee_add
AFTER INSERT
ON employee
FOR EACH ROW
BEGIN
    INSERT INTO employee_log (emp_id, emp_name, dept_id, action_time,
    action_description)
    VALUES (NEW.emp_id, NEW.emp_name, NEW.dept_id, NOW(), 'new employee
added');

END $$

DELIMITER ;
```

phpMyAdmin

Server: 127.0.0.1 » Database: school_db » Table: employees

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

Triggers

Check all Export Drop Create new trigger

Name	Time	Event
employee_add	AFTER INSERT	Edit Export Drop

New information_schema mysql performance_schema phpmyadmin school_db Tables Views test university_db

Console

The screenshot shows the 'Triggers' section of the phpMyAdmin interface for the 'employees' table in the 'school_db' database. A single trigger, 'employee_add', is listed under the 'Time' column as 'AFTER INSERT'. There are buttons for 'Edit', 'Export', and 'Drop'.

phpMyAdmin

Server: 127.0.0.1 » Database: school_db » Table: employee_log

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

Showing rows 0 - 1 (2 total, Query took 0.0006 seconds.)

```
SELECT * FROM `employee_log`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

log_id	emp_id	emp_name	dept_id	action_time	action_description
1	6	poonam	2	2025-12-16 16:00:49	new employee added
2	7	NULL	NULL	2025-12-16 16:00:49	new employee added

Check all With selected: Edit Copy Delete Export

Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Query results operations

Print Copy to clipboard Export Display chart Create view

New information_schema mysql performance_schema phpmyadmin school_db Tables Views test university_db

The screenshot shows the 'employee_log' table in the 'school_db' database. It displays two rows of data: one for a new employee with log_id 1, emp_id 6, and emp_name 'poonam', and another for a new employee with log_id 2, emp_id 7, and emp_name NULL. The table has columns for log_id, emp_id, emp_name, dept_id, action_time, and action_description.

Lab 2: Trigger to Update last_modified on Employee Update

```
DELIMITER $$
```

```
CREATE TRIGGER emp_modification
```

```
BEFORE UPDATE ON employee
```

```
FOR EACH ROW
```

```
BEGIN
```

```
INSERT INTO employee_log (emp_id, emp_name, dept_id, salary, action_time,  
action_description) VALUES (NEW.emp_id, NEW.emp_name, NEW.dept_id,  
NEW.salary, NOW(),'UPDATED');
```

```
END$$
```

```
DELIMITER ;
```

PL/SQL

Theory Questions

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

PL/SQL (Procedural Language / SQL) is Oracle's procedural extension of SQL that allows developers to write **blocks of code** containing **SQL statements along with programming constructs**.

How PL/SQL extends SQL:

- SQL works only with data manipulation
- PL/SQL adds:
 - Variables
 - Conditional statements (IF, CASE)
 - Loops (LOOP, WHILE, FOR)
 - Exception handling
- Allows writing complete programs inside the database

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

Benefits of PL/SQL:

1. **Improved performance** – multiple SQL statements executed in one block
2. **Reduced network traffic**
3. **Better error handling** using exceptions
4. **Modular programming** (procedures, functions)
5. **Secure code** stored inside the database
6. **Reusability and maintainability**

LAB EXERCISES

Lab 1: PL/SQL Block to Print Total Number of Employees

DECLARE

 total_employees NUMBER;

BEGIN

 SELECT COUNT(*)

 INTO total_employees

 FROM employees;

 DBMS_OUTPUT.PUT_LINE('Total Employees: ' || total_employees);

END;

/

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

DELIMITER //

CREATE PROCEDURE total_sale()

BEGIN

 SELECT SUM(total_bill) as total_sale from customer;

END //

DELIMITER ;

phpMyAdmin

Server: 127.0.0.1 » Database: school_db » Table: customer

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

Show query box

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0108 seconds.)

```
CREATE PROCEDURE toatl_sale() BEGIN SELECT SUM(total_bill) AS total_sale FROM customer; END;
```

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

New information_schema mysql performance_schema phpmyadmin school_db Procedures Tables Views test university_db

Console

localhost/phpmyadmin/index.php?route=/database/routines&type=PROCEDURE&db=school_db

Summarize A ⚡ ☆ ⚡ ... Ch

phpMyAdmin

Structure SQL Search Query Export Import Operations Privileges Routines Events More

Your SQL query has been executed successfully.
1 row affected by the last statement inside the procedure.

```
CALL `toatl_sale`();
```

Execution results of routine `toatl_sale`:

total_sale
25000

Routines

Check all Export Drop Search Create new routine

Name	Type	Returns
toatl_sale	PROCEDURE	<input type="button"/> Edit <input type="button"/> Execute <input type="button"/> Export <input type="button"/> Drop
total_empl	PROCEDURE	<input type="button"/> Edit <input type="button"/> Execute <input type="button"/> Export <input type="button"/> Drop

Console

17. PL/SQL Control Structures

Theory Questions:

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

What are control structures?

Control structures in PL/SQL are programming constructs that **control the flow of execution** of a program. They allow PL/SQL programs to make decisions, repeat statements, and execute code conditionally.

PL/SQL control structures are mainly classified into:

1. **Conditional control structures**
2. **Iterative (looping) control structures**
3. **Sequential control structures**

A. IF-THEN Control Structure

The **IF-THEN** structure is used to execute a block of statements **only when a condition is true**.

Syntax

IF condition THEN

 statements;

END IF;

IF-THEN-ELSE

IF condition THEN

 statements;

ELSE

 statements;

END IF;

IF-THEN-ELSIF

```
IF condition1 THEN  
    statements;  
ELSIF condition2 THEN  
    statements;  
ELSE  
    statements;  
END IF;
```

Example

```
IF salary > 50000 THEN  
    bonus := salary * 0.10;  
ELSE  
    bonus := salary * 0.05;  
END IF;
```

Use:

- Decision making
- Validation of data
- Conditional execution of SQL statements

B. LOOP Control Structure

Loops are used to **execute a set of statements repeatedly** until a condition is met.

Types of loops in PL/SQL

1. Basic LOOP

Executes until an explicit EXIT condition is met.

```
LOOP  
    statements;
```

EXIT WHEN condition;

END LOOP;

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

Control structures in PL/SQL allow programmers to combine **procedural logic with SQL**, making it possible to handle complex database operations that cannot be achieved using SQL alone.

Key ways they help:

3. Conditional execution of queries

Using IF-THEN-ELSE, different SQL statements can be executed based on conditions such as input values or query results.

4. Row-by-row processing

Loop structures (FOR, WHILE, LOOP) allow repeated execution of SQL statements, enabling processing of multiple rows returned by a query.

5. Data validation and business logic

Conditions can be checked before inserting, updating, or deleting data, ensuring data integrity and enforcement of business rules.

6. Dynamic decision making

Queries can behave differently at runtime based on variable values, making programs flexible and adaptive.

7. Error handling and control flow

Control structures help manage exceptions and guide program flow during errors, improving reliability.

LAB EXERCISES:

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

DELIMITER //

```
CREATE PROCEDURE CheckDeptByEmpId(IN emp INT)
BEGIN
DECLARE temp_name VARCHAR(100);
SELECT d.dept_name INTO temp_name
FROM employee e
JOIN departments d ON e.dept_id = d.dept_id
WHERE e.emp_id = emp;
IF temp_name IS NOT NULL THEN
SELECT CONCAT('Employee is in ', temp_name) AS
Message;
ELSE
SELECT 'Employee id is invalid ' AS Message;
END IF;
END //
```

DELIMITER ;

phpMyAdmin

Server: 127.0.0.1 » Database: school_db

Recent Favorites

New information_schema mysql performance_schema phpmyadmin school_db Procedures Tables Views test university_db

Structure SQL Search Query Export Import Operations Privileges Routines Events More

Your SQL query has been executed successfully.
1 row affected by the last statement inside the procedure.

```
SET @p0='3'; CALL `CheckDeptByEmpId`(@p0);
```

Execution results of routine 'CheckDeptByEmpId'

Message
Employee is in CSE

Routines

Check all

Search

Name	Type	Returns
<input type="checkbox"/> CheckDeptByEmpId	PROCEDURE	<input type="button" value="Edit"/> <input type="button" value="Execute"/> <input type="button" value="Export"/> <input type="button" value="Drop"/>
<input type="checkbox"/> total_sale	PROCEDURE	<input type="button" value="Edit"/> <input type="button" value="Execute"/> <input type="button" value="Export"/> <input type="button" value="Drop"/>
<input type="checkbox"/> total_empl	PROCEDURE	<input type="button" value="Edit"/> <input type="button" value="Execute"/> <input type="button" value="Export"/> <input type="button" value="Drop"/>

Console

The screenshot shows the phpMyAdmin interface for a MySQL database named 'school_db'. In the left sidebar, there are several databases listed: 'information_schema', 'mysql', 'performance_schema', 'phpmyadmin', 'school_db', and 'university_db'. Under 'school_db', there are 'Tables' (including 'New', 'courses', 'customer', 'departments', 'employees', 'employee_log', and 'teachers') and 'Views'. A 'Procedures' section is also present. The main content area shows the results of executing a SQL statement: 'SET @p0='3'; CALL `CheckDeptByEmpId`(@p0);'. The message box displays the result: 'Employee is in CSE'. Below this, the 'Routines' section lists three procedures: 'CheckDeptByEmpId', 'total_sale', and 'total_empl', each with edit, execute, export, and drop options. At the bottom, there is a 'Console' button.

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

```
DELIMITER //  
  
CREATE PROCEDURE DisplayEmployeeNames()  
BEGIN  
DECLARE total INT DEFAULT 0;  
DECLARE counter INT DEFAULT 1;  
DECLARE empName VARCHAR(100);  
SELECT COUNT(*) INTO total FROM employees;  
WHILE counter <= total DO  
    SELECT emp_name INTO empName  
    FROM employees  
    WHERE emp_id = counter;  
    SELECT CONCAT(counter, '->', empName) AS  
    'Employee name';  
    SET counter = counter + 1;  
END WHILE;  
END //  
DELIMITER ;
```

phpMyAdmin

Server: 127.0.0.1 » Database: school_db

Structure SQL Search Query Export Import Operations Privileges Routines Events M

CALL `DisplayEmployeeNames`();

Execution results of routine 'DisplayEmployeeNames'

Employee name
1-> brijesh

Employee name
2-> sweta

Employee name
3-> Ankita

Employee name
4-> Tanuja

Employee name
5-> Shankar

Employee name
6-> poonam

New information_schema mysql performance_schema phpmyadmin school_db Procedures Tables Views test university_db

Employee name	Value
1->	brijesh
2->	sweta
3->	Ankita
4->	Tanuja
5->	Shankar
6->	poonam

18. SQL Cursors Theory Questions:

1. What is a cursor in PL/SQL?

A **cursor** in PL/SQL is a pointer to a **private SQL work area** that stores information about the execution of a SQL statement. It allows PL/SQL to **process the rows returned by a SQL query one at a time**.

Whenever a SQL statement is executed, Oracle creates a cursor to:

- Parse the SQL statement
- Execute it
- Store and retrieve the result set

Implicit vs. Explicit Cursors

Feature	Implicit Cursor	Explicit Cursor
Created by	Oracle automatically	Programmer explicitly
Used for	INSERT, UPDATE, DELETE, and SELECT INTO	SELECT statements returning multiple rows
Declaration	Not required	Required
Cursor name	SQL	User-defined
Control over processing	Limited	Full control
Cursor attributes	SQL%FOUND, SQL%NOTFOUND, SQL%ROWCOUNT	cursor_name%FOUND, etc.

Implicit Cursor Example

```
BEGIN  
    UPDATE employees  
    SET salary = salary * 1.1
```

```
WHERE department_id = 10;

IF SQL%ROWCOUNT > 0 THEN
    DBMS_OUTPUT.PUT_LINE('Rows updated');
END IF;
END;
```

Explicit Cursor Example

```
DECLARE
    CURSOR emp_cur IS
        SELECT employee_id, salary FROM employees;
        v_emp_id employees.employee_id%TYPE;
        v_salary employees.salary%TYPE;
BEGIN
    OPEN emp_cur;
    LOOP
```

LAB EXERCISES:

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

DELIMITER //

```
CREATE PROCEDURE ShowEmployeeDetails()
BEGIN
DECLARE h_id INT;
DECLARE h_name VARCHAR(100);
DECLARE h_dept INT;
DECLARE h_salary INT;
DECLARE done INT DEFAULT 0;
DECLARE emp_cursor CURSOR FOR
SELECT emp_id, emp_name, dept_id, salary FROM
employees;
DECLARE CONTINUE HANDLER FOR NOT FOUND
SET done = 1;
OPEN emp_cursor;
read_loop: LOOP
FETCH emp_cursor INTO h_id, h_name, h_dept,
h_salary;
IF done = 1 THEN
LEAVE read_loop;
END IF;
SELECT CONCAT('Emp ID : ', h_id, ', name: ', h_name,
', dept id: ', h_dept, ', salary: ', h_salary) AS
Employee_details;
```

```
END LOOP;
```

```
CLOSE emp_cursor;
```

```
END //
```

```
DELIMITER ;
```

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. In the left sidebar, under 'Procedures', there is a single entry for 'ShowEmployeeDetails'. The main area displays the execution results of this procedure, which outputs six rows of employee details:

Employee_details
Emp ID : 1, name: brijesh, dept id: 1, salary: 50000
Employee_details
Emp ID : 2, name: sweta, dept id: 2, salary: 40000
Employee_details
Emp ID : 3, name: Ankita, dept id: 2, salary: 9000
Employee_details
Emp ID : 4, name: Tanuja, dept id: 1, salary: 50000
Employee_details
Emp ID : 5, name: Shankar, dept id: 1, salary: 12000
Employee_details
Emp ID : 6, name: poonam, dept id: 2, salary: 3900

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

```
DELIMITER //
```

```
CREATE PROCEDURE ShowCourses()
```

```
BEGIN
```

```
DECLARE c_id INT;
```

```
DECLARE c_name VARCHAR(100);
```

```
DECLARE c_duration VARCHAR(50);
```

```
DECLARE done INT DEFAULT 0;
```

```
DECLARE course_cursor CURSOR FOR
```

```
SELECT course_id, course_name, course_duration
```

```
FROM courses;
```

```
DECLARE CONTINUE HANDLER FOR NOT FOUND
```

```

SET done = 1;

OPEN course_cursor;

read_loop: LOOP

FETCH course_cursor INTO c_id, c_name, c_duration;

IF done = 1 THEN

LEAVE read_loop;

END IF;

SELECT CONCAT('course id : ', c_id, ', name: ',  

c_name, ', duration: ', c_duration) AS course_details;  

END LOOP;  

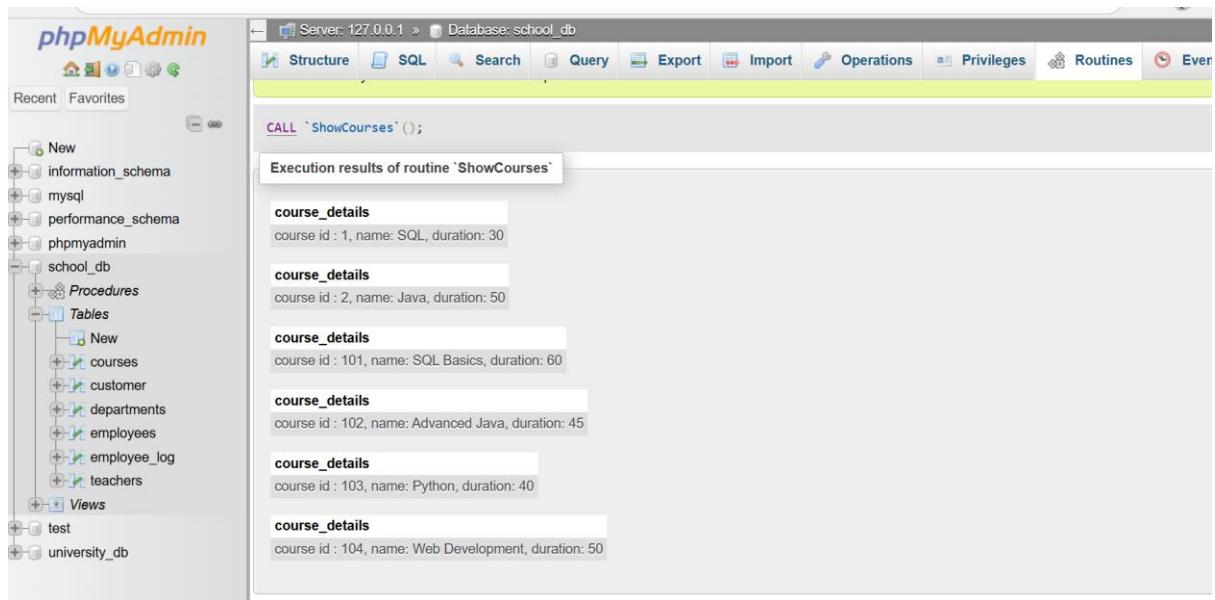
  

CLOSE course_cursor;  

END //
```

DELIMITER ;



19. Rollback and Commit Savepoint

Theory Questions:

1. Concept of SAVEPOINT in Transaction Management

A SAVEPOINT is a marker set within a transaction that allows you to roll back part of the transaction without undoing the entire transaction.

It enables partial rollback, giving finer control over transaction management.

Syntax

```
SAVEPOINT savepoint_name;
```

Interaction of SAVEPOINT with ROLLBACK and COMMIT

ROLLBACK

- ROLLBACK TO savepoint_name;
 - Reverses all changes made after the specified savepoint
 - Changes made before the savepoint remain intact
- ROLLBACK;
 - Cancels the entire transaction
 - Removes all savepoints

COMMIT

- COMMIT;
 - Makes all changes permanent
 - Eliminates all savepoints
 - After a commit, rollback to any savepoint is not possible

Example

```
INSERT INTO orders VALUES (101, 'Laptop');
```

```
SAVEPOINT sp1;
```

```
INSERT INTO orders VALUES (102, 'Phone');
```

```
SAVEPOINT sp2;
```

```
DELETE FROM orders WHERE order_id = 101;
```

```
ROLLBACK TO sp2; -- Undoes DELETE only
```

```
COMMIT;      -- Saves remaining changes
```

2. When is it useful to use savepoints?

Savepoints are useful when:

- Performing complex transactions with multiple steps
- You want to undo only part of a transaction when an error occurs
- Handling conditional logic within a transaction
- Reducing the cost of rolling back a long transaction
- Implementing error recovery in stored procedures
- Testing intermediate transaction states

Real-World Example

In an online order system:

1. Create customer record
2. Create order
3. Process payment

If payment fails, you can rollback to a savepoint before payment, instead of canceling the entire order creation.

LAB EXERCISES:

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

START TRANSACTION;

INSERT INTO employee (emp_name, dept_id, salary)

VALUES ('abhay', 2, 70000);

SAVEPOINT after_first_insert;

INSERT INTO employee (emp_name, dept_id, salary)

VALUES ('sayan', 1, 52000);

ROLLBACK TO after_first_insert;

COMMIT;

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

```
START TRANSACTION;
```

```
INSERT INTO courses (course_id, course_name, course_duration)
```

```
VALUES (1009, 'cloud basics', '6 months');
```

```
SAVEPOINT after_first_course;
```

```
INSERT INTO courses (course_id, course_name, course_duration)
```

```
VALUES (1010, 'data analytics', '7 months');
```

```
RELEASE SAVEPOINT after_first_course;
```

```
COMMIT;
```

```
START TRANSACTION;
```

```
INSERT INTO courses (course_id, course_name, course_duration)
```

```
VALUES (1011, 'ai ml', '8 months');
```

```
SAVEPOINT after_new_course;
```

```
INSERT INTO courses (course_id, course_name, course_duration)
```

```
VALUES (1012, 'seo', '5 months');
```

```
ROLLBACK TO after_new_course;
```

```
COMMIT;
```

```
CREATE DATABASE library_db;
```

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.

```
USE library_db;
```

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

```
INSERT INTO books VALUES
(1, 'The Secrets', 'Rhonda Byrne', 'Brown's ', 192, 100),
(2, 'Mahabharat', 'Vyasa', 'Gita Press', 1960, 120),
(3, 'Ramayana', 'Valmiki', 'Secker & Warbur', 2020, 1200);
```

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
- Database:** library_db
- Table:** books
- Table Structure:** Shows columns: book_id, title, author, publisher, year_of_publication, price.
- Data:**

book_id	title	author	publisher	year_of_publication	price
1	The Secrets	Rhonda Byrne	Brown's	192	100.00
2	Mahabharat	Vyasa	Gita Press	1960	120.00
3	Ramayana	Valmiki	Gita Press	1960	1200.00
- Operations:** Includes options like Edit, Copy, Delete, and Export.
- Query results operations:** Includes Print, Copy to clipboard, Export, Display chart, and Create view.

- **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.**

```
USE library_db;
```

```
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
('Sweta', '2023-01-10', 'swe@email.com'),
```

```
('Ankita', '2023-02-15', 'anki@email.com'),  
('Tina', '2023-03-20', 'tina@email.com'),  
('Pooja', '2023-04-25', 'pooj@email.com'),  
('ahuja', '2023-05-30', 'huja@email.com');
```

The screenshot shows the phpMyAdmin interface for the 'library_db' database. The left sidebar lists databases: information_schema, library_db, mysql, performance_schema, phpmyadmin, school_db, test, and university_db. The 'members' table under 'library_db' is selected. The main area displays the results of the query: 'SELECT * FROM `members`'. The results table has columns: member_id, member_name, date_of_membership, and email. The data shows five rows:

member_id	member_name	date_of_membership	email
1	Sweta	2023-01-10	swe@email.com
2	Ankita	2023-02-15	anki@email.com
3	Tina	2023-03-20	tina@email.com
4	Pooja	2023-04-25	pooj@email.com
5	ahuja	2023-05-30	huja@email.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;
```

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
- Database:** library_db
- Table:** members
- Query Result:** SELECT * FROM members WHERE date_of_membership < '2022-01-01' ORDER BY date_of_membership;
- Table Data:** A table showing two rows of data from the members table.

member_id	member_name	date_of_membership	email
3	Tina	2015-03-20	
4	Pooja	2018-04-25	

A tooltip is visible over the "date_of_membership" column header, providing instructions for sorting results.

- 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 = 'vyasa'  
ORDER BY year_of_publication DESC;
```

The screenshot shows the phpMyAdmin interface for a MySQL database named 'library_db'. The 'books' table is selected. The SQL query entered is:

```
SELECT title FROM books WHERE author = 'vyasa' ORDER BY year_of_publication DESC;
```

The results show one row: 'Mahabharat'. The interface includes a sidebar with databases like 'information_schema', 'library_db', 'mysql', 'performance_schema', 'phpmyadmin', 'school_db', 'test', and 'university_db'. It also features tabs for 'Browse', 'Structure', 'SQL', 'Search', 'Insert', 'Export', 'Import', 'Privileges', 'Operations', and 'Triggers'.

4. SQL Constraints LAB EXERCISES:

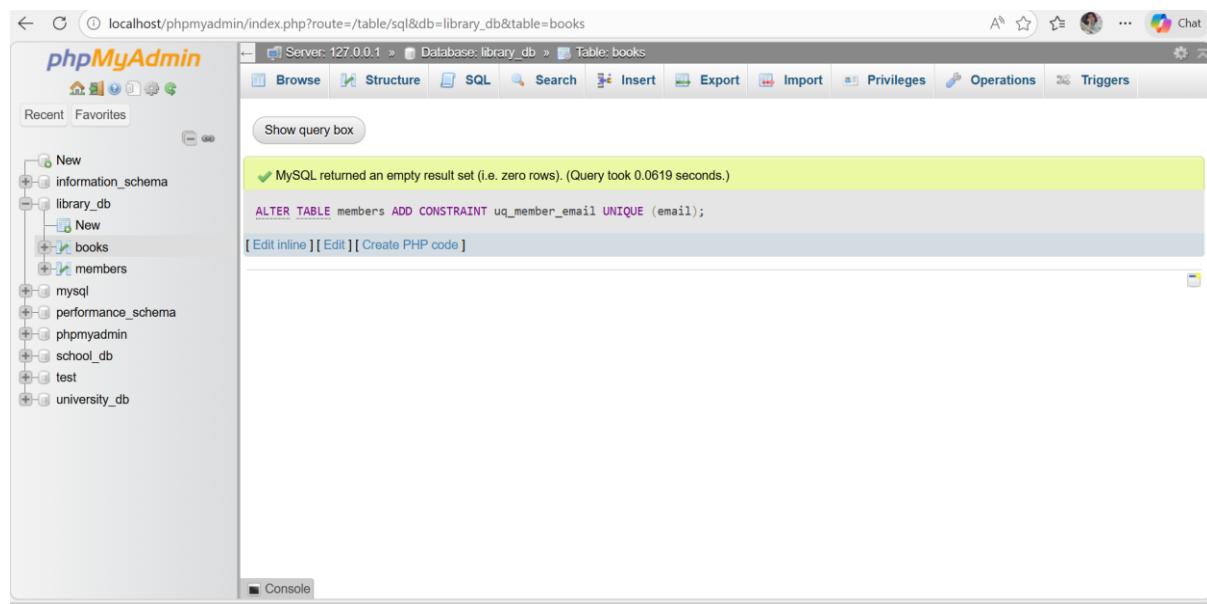
- Lab 3: Add a CHECK constraint to ensure that the price of books in the books table is greater than 0.

- 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 uq_member_email
```

```
UNIQUE (email);
```



5. 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.

USE library_db;

```
CREATE TABLE authors (
```

```
    author_id INT PRIMARY KEY,
```

```
    first_name VARCHAR(50),
```

```
    last_name VARCHAR(50),
```

```
    country VARCHAR(50)
```

```
);
```

The screenshot shows the phpMyAdmin interface for the 'library_db' database. The left sidebar lists databases and tables, with 'authors' selected. The main panel displays the 'Table structure' for the 'authors' table. The table has four columns: 'author_id' (int(11), primary key, not null), 'first_name' (varchar(50), general_ci), 'last_name' (varchar(50), general_ci), and 'country' (varchar(50), general_ci). Below the table structure, there is an 'Indexes' section showing a single index named 'PRIMARY' (BTREE, unique, packed, column 'author_id', cardinality 0, collation A, null no).

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	author_id	int(11)			No	None			Change Drop More
2	first_name	varchar(50)	utf8mb4_general_ci		Yes	NULL			Change Drop More
3	last_name	varchar(50)	utf8mb4_general_ci		Yes	NULL			Change Drop More
4	country	varchar(50)	utf8mb4_general_ci		Yes	NULL			Change Drop More

Indexes

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Rename Drop	PRIMARY	BTREE	Yes	No	author_id	0	A	No	

Create an index on columns [Go](#)

- 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 PRIMARY KEY,
    publisher_name VARCHAR(100),
    contact_number VARCHAR(15),
    address VARCHAR(200),
    UNIQUE (contact_number)
);
```

The screenshot shows the phpMyAdmin interface for the 'library_db' database. The 'publishers' table structure is displayed. The table has four columns: publisher_id (INT, primary key), publisher_name (VARCHAR(100)), contact_number (VARCHAR(15)), and address (VARCHAR(200)). A unique index is defined on the contact_number column. The 'Indexes' section shows two indexes: one for the primary key (publisher_id) and another for the contact_number column.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	publisher_id	int(11)	utf8mb4_general_ci		No	None			Change Drop More
2	publisher_name	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More
3	contact_number	varchar(15)	utf8mb4_general_ci		Yes	NULL			Change Drop More
4	address	varchar(200)	utf8mb4_general_ci		Yes	NULL			Change Drop More

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Rename Drop	PRIMARY	BTREE	Yes	No	publisher_id	0	A	No	
Edit Rename Drop	contact_number	BTREE	Yes	No	contact_number	0	A	Yes	

6. 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 genre VARCHAR(50);

The screenshot shows the phpMyAdmin interface. On the left, the database structure is visible with the 'library_db' database selected. Inside 'library_db', the 'books' table is selected. The main area shows a query result: 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0131 seconds.)' Below this, the executed SQL command is shown: 'ALTER TABLE books ADD genre VARCHAR(50);'. There are three buttons at the bottom of the query box: 'Edit inline', 'Edit', and 'Create PHP code'.

UPDATE books SET genre = 'fiction' WHERE book_id = 1;

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

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

The screenshot shows the phpMyAdmin interface for the library_db database. The left sidebar lists databases and tables. The main area shows the books table with the following data:

book_id	title	author	publisher	year_of_publication	price	genre
1	The Secrets	Rhonda Byrne	Brown's	192	100.00	fiction
2	Mahabharat	Vyasa	Gita Press	1960	120.00	Mythological
3	Ramayana	Valmiki	Gita Press	1960	1200.00	Mythological

Below the table, there are buttons for Edit, Copy, Delete, Check all, and Export.

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

ALTER TABLE members

MODIFY email VARCHAR(100);

The screenshot shows the phpMyAdmin interface after running an ALTER TABLE query. The message bar says: "MySQL returned an empty result set (i.e. zero rows). (Query took 0.0136 seconds.)". The query entered was:

```
ALTER TABLE members MODIFY email VARCHAR(100);
```

7. DROP Command LAB EXERCISES:

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

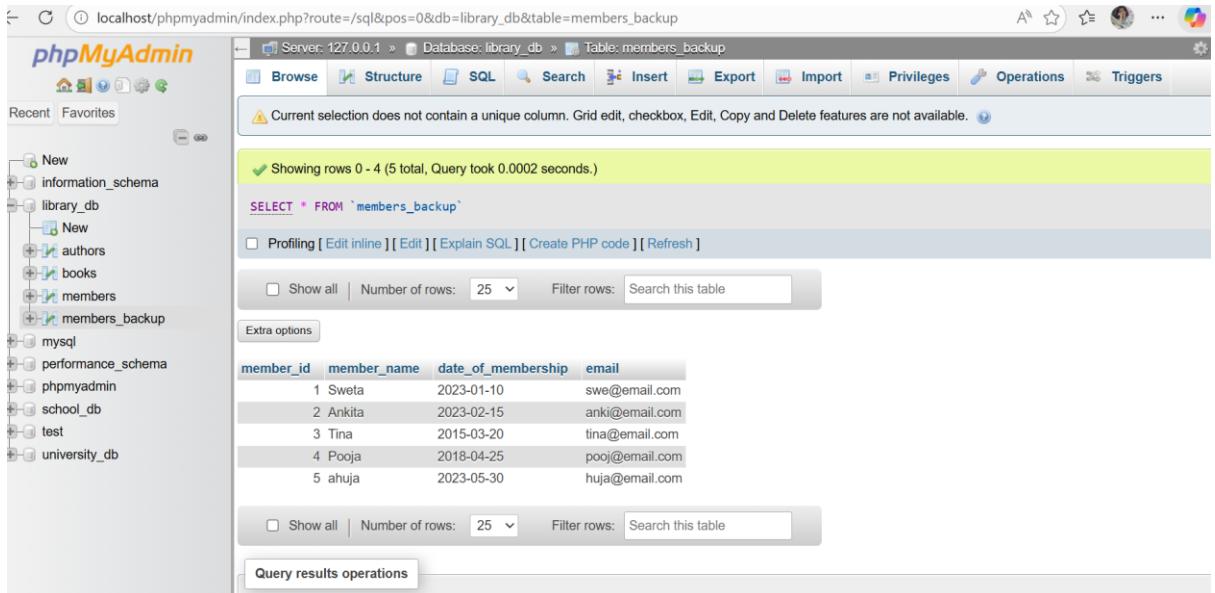
DROP TABLE publishers;

The screenshot shows the phpMyAdmin interface. On the left, there's a tree view of databases: New, information_schema, library_db (selected), New, authors, books, members, mysql, performance_schema, phpmyadmin, school_db, test, university_db. The main area has a toolbar with Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, and Triggers. Below the toolbar, a message says "MySQL returned an empty result set (i.e. zero rows). (Query took 0.0167 seconds.)". Underneath that, the SQL query "DROP TABLE publishers;" is shown, along with "[Edit inline] [Edit] [Create PHP code]".

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

`CREATE TABLE members_backup AS`

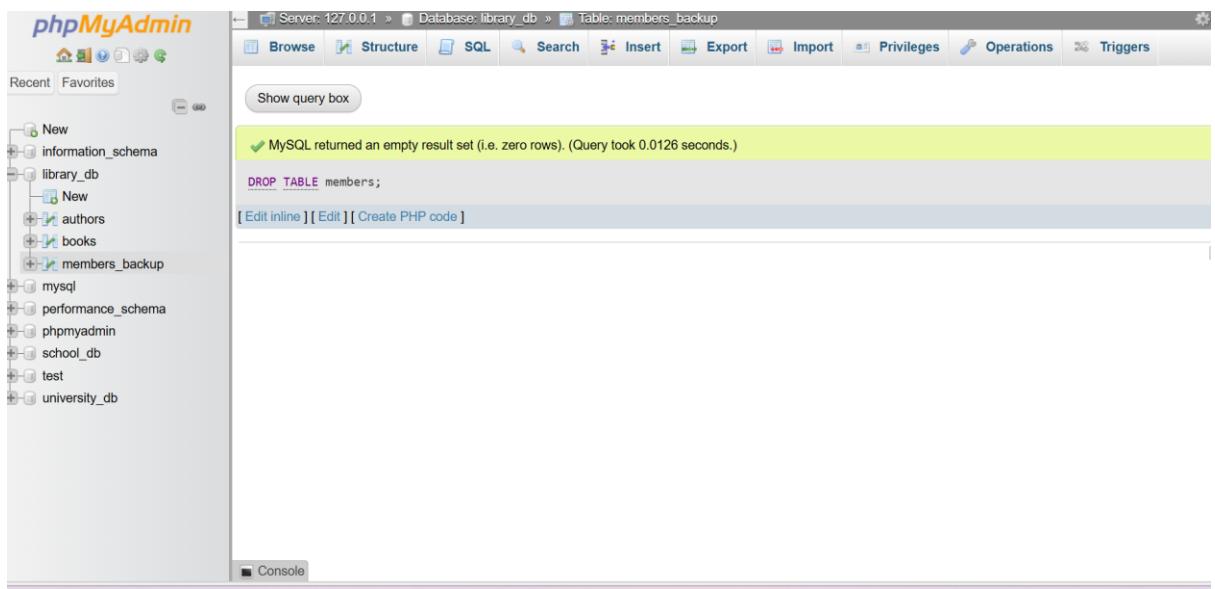
`SELECT * FROM members;`



The screenshot shows the phpMyAdmin interface for the library_db database. The left sidebar shows various databases like information_schema, library_db, mysql, performance_schema, etc. The main area is titled 'Table: members_backup'. It displays a grid of data with columns: member_id, member_name, date_of_membership, and email. The data is as follows:

member_id	member_name	date_of_membership	email
1	Sweta	2023-01-10	swe@email.com
2	Ankita	2023-02-15	anki@email.com
3	Tina	2015-03-20	tina@email.com
4	Pooja	2018-04-25	pooj@email.com
5	ahuja	2023-05-30	huja@email.com

`DROP TABLE members;`



The screenshot shows the phpMyAdmin interface for the library_db database. The left sidebar shows various databases. The main area is titled 'Table: members_backup'. It displays a message: 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0126 seconds.)' Below it, the SQL query shown is `DROP TABLE members;`. There is also a 'Console' tab at the bottom.

8. 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

(author_id, first_name, last_name, country)

VALUES (1, 'vyasa', 'Ved', 'India'),

(2, 'Valmiki', 'Maharishi', 'India'),

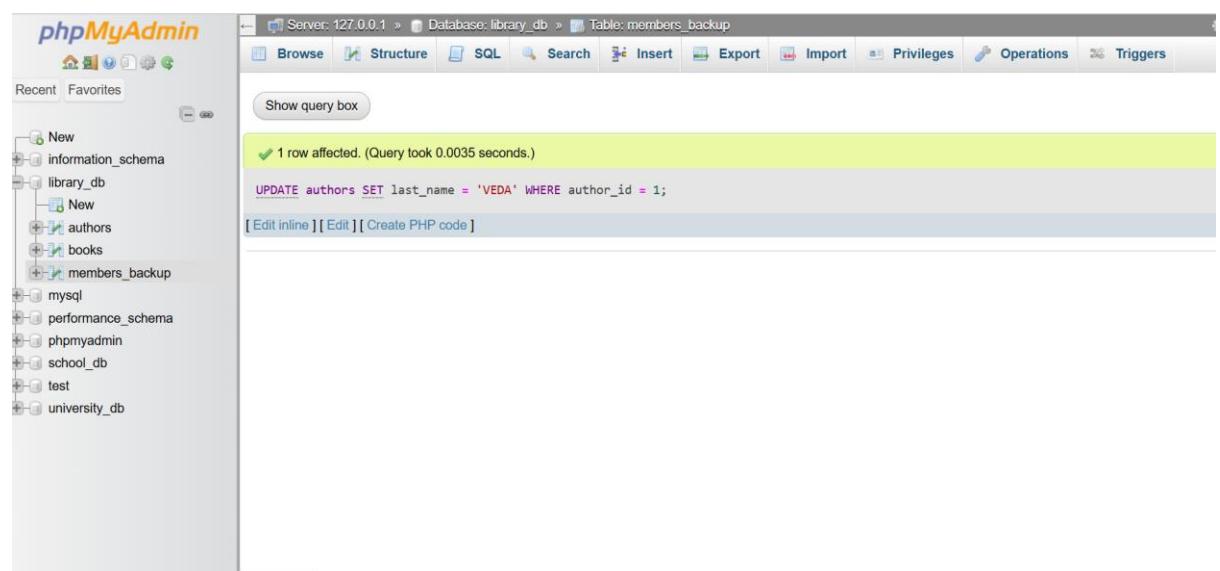
(3, 'Rhoda', 'Byres', 'USA');

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

UPDATE authors

SET last_name = 'VEDA'

WHERE author_id = 1;



The screenshot shows the phpMyAdmin interface. On the left, the database structure is visible with databases like 'library_db' and tables like 'authors'. The main area shows a SQL query: 'UPDATE authors SET last_name = 'VEDA' WHERE author_id = 1;'. Below the query, a message says '1 row affected. (Query took 0.0035 seconds.)'. The top navigation bar includes links for Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, and Triggers.

9. UPDATE Command LAB EXERCISE:

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

UPDATE books

SET year_of_publication = 1940

WHERE book_id = 2;

The screenshot shows the phpMyAdmin interface for the library_db database. The left sidebar lists various databases and their tables. The main area displays the 'books' table with the following data:

book_id	title	author	publisher	year_of_publication	price	genre
1	The Secrets	Rhonda Byrne	Brown's	192	100.00	fiction
2	Mahabharat	Vyasa	Gita Press	1940	120.00	Mythological
3	Ramayana	Valmiki	Gita Press	1960	1200.00	Mythological

The row for book_id 2 is highlighted. At the bottom of the table area, there are buttons for 'Check all', 'With selected:', and 'Edit'. Below the table, there are additional buttons for 'Print', 'Copy to clipboard', 'Export', 'Display chart', and 'Create view'.

• 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;

The screenshot shows the phpMyAdmin interface for a MySQL database named 'library_db'. The left sidebar lists various databases and their tables, including 'information_schema', 'library_db' (which contains 'authors', 'books', and 'members_backup'), 'mysql', 'performance_schema', 'phpmyadmin', 'school_db', 'test', and 'university_db'. The main panel is titled 'Table: books' and displays the results of the SQL query: 'SELECT * FROM `books`'. The results show three rows of book data:

book_id	title	author	publisher	year_of_publication	price	genre
1	The Secrets	Rhonda Byrne	Brown's	192	110.00	fiction
2	Mahabharat	Vyasa	Gita Press	1940	132.00	Mythological
3	Ramayana	Valmiki	Gita Press	1960	1320.00	Mythological

Below the table, there are buttons for 'Edit', 'Copy', 'Delete', 'Check all', and 'With selected:'. At the bottom of the main panel, there are buttons for 'Show all', 'Number of rows: 25', 'Filter rows: Search this table', and 'Sort by key: None'. The bottom section is labeled 'Query results operations' with buttons for 'Print', 'Copy to clipboard', 'Export', 'Display chart', and 'Create view'.

10.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';

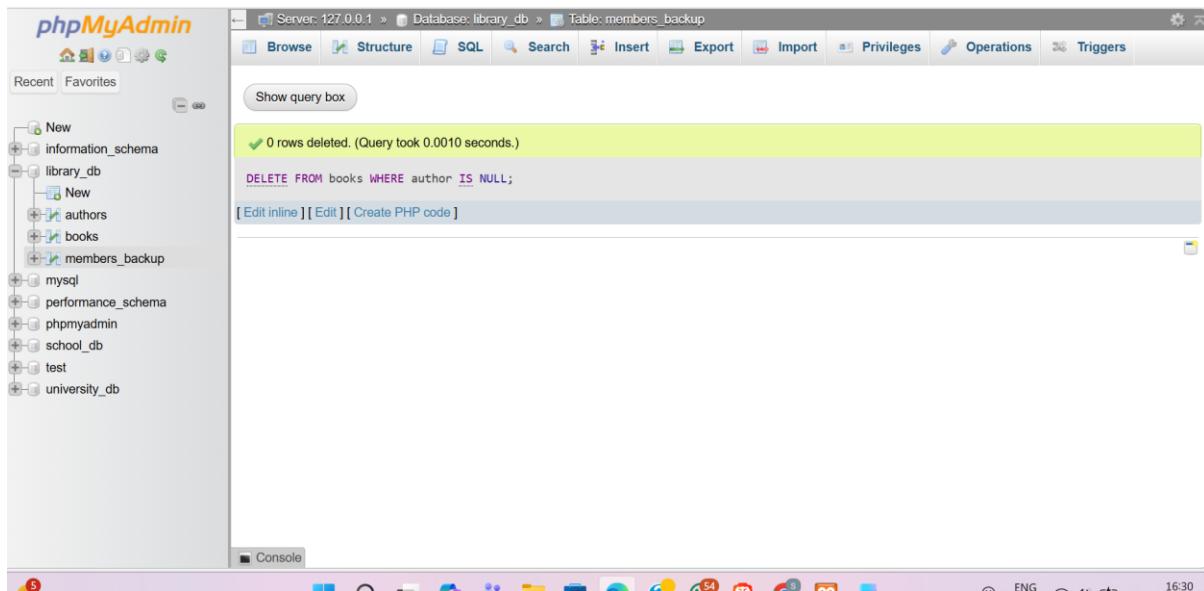
The screenshot shows the phpMyAdmin interface for the 'library_db' database. The 'members_backup' table is selected. The table has four columns: member_id, member_name, date_of_membership, and email. The data shows three rows: Sweta (joined 2023-01-10), Ankita (joined 2023-02-15), and ahuja (joined 2023-05-30). A SQL query is visible in the query editor: 'SELECT * FROM `members_backup`'. The results pane shows the same three rows.

member_id	member_name	date_of_membership	email
1	Sweta	2023-01-10	swe@email.com
2	Ankita	2023-02-15	anki@email.com
5	ahuja	2023-05-30	huja@email.com

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

DELETE 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;
```

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
- Database:** library_db
- Table:** books
- Query Result:** MySQL returned an empty result set (i.e. zero rows). (Query took 0.0008 seconds.)
- SQL Query:** SELECT * FROM books WHERE price BETWEEN 50 AND 100;
- Operations:** Profiling, Edit inline, Edit, Explain SQL, Create PHP code, Refresh.
- Table Headers:** book_id, title, author, publisher, year_of_publication, price, genre
- Query Results Operations:** Create view

- **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;
```

phpMyAdmin

Server: 127.0.0.1 > Database: library_db > Table: books

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

Show query box

✓ Showing rows 0 - 2 (3 total, Query took 0.0002 seconds.) [author: RHONDA BYRNE... - VYASA...]

SELECT * FROM books ORDER BY author ASC LIMIT 3;

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Extra options

	book_id	title	author	publisher	year_of_publication	price	genre
<input type="checkbox"/>	1	The Secrets	Rhonda Byrne	Brown's	192	110.00	fiction
<input type="checkbox"/>	3	Ramayana	Valmiki	Gita Press	1960	1320.00	Mythological
<input type="checkbox"/>	2	Mahabharat	Vyasa	Gita Press	1940	132.00	Mythological

Check all With selected: Edit Copy Delete Copy Export

Query results operations

Print Copy to clipboard Export Display chart Create view

Console

Data Control Language (DCL)

LAB EXERCISES:

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

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

The screenshot shows the phpMyAdmin interface. On the left, the database tree is visible with 'library_db' selected. On the right, the 'SQL' tab is active. A green message bar at the top indicates 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0112 seconds.)'. Below it, the SQL query is displayed: 'GRANT SELECT ON library_db.books TO 'librarian'@'localhost''.

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

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

The screenshot shows the phpMyAdmin interface. On the left, the database tree is visible with 'library_db' selected. On the right, the 'SQL' tab is active. A green message bar at the top indicates 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0029 seconds.)'. Below it, the SQL query is displayed: 'GRANT INSERT, UPDATE ON library_db.members TO 'admin'@'localhost''.

11.REVOKE Command LAB EXERCISES:

- Lab 3: Revoke the INSERT privilege from the user librarian on the books table.**
- Lab 4: Revoke all permissions from user admin on the members table.**

12. 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 TRANSACTION;

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

(4, 'Moby ', 'Herman Melville', 'Harper & Brothers', 1851, 11.50, 'Adventure'),

(5, 'War and Peace', 'Leo Tolstoy', 'The Russian Messenger', 1869, 13.00, 'Historical');

The screenshot shows the phpMyAdmin interface for the 'library_db' database. The 'books' table is selected. The SQL tab shows the following queries:

```
UPDATE `books` SET `title` = 'Moby ' WHERE `books`.`book_id` = 4;
SELECT * FROM `books`
```

The results pane shows 5 rows of data:

book_id	title	author	publisher	year_of_publication	price	genre
1	The Secrets	Rhonda Byrne	Brown's	192	110.00	fiction
2	Mahabharat	Vyasa	Gita Press	1940	132.00	Mythological
3	Ramayana	Valmiki	Gita Press	1960	1320.00	Mythological
4	Moby	Herman Melville	Harper & Brothers	1851	11.50	Adventure
5	War and Peace	Leo Tolstoy	The Russian Messenger	1869	13.00	Historical

COMMIT;

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

START TRANSACTION;

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

```
(8, 'Ulysses', 'James Joyce', 'Shakespeare and Company', 1922, 12.00, 'Modernist');
```

```
ROLLBACK;
```

The screenshot shows the phpMyAdmin interface for a MySQL database named 'library_db'. The left sidebar lists various databases. The main area is titled 'Table: books' and shows the results of an SQL query:

```
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0001 seconds.)  
START TRANSACTION;  
[ Edit inline ] [ Edit ] [ Create PHP code ]  
1 row inserted. (Query took 0.0017 seconds.)  
INSERT INTO books (book_id, title, author, publisher, year_of_publication, price, genre) VALUES (8, 'Ulysses', 'James Joyce', 'Shakespeare and Company', 1922, 12.00, 'Modernist');  
[ Edit inline ] [ Edit ] [ Create PHP code ]  
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0035 seconds.)  
ROLLBACK;  
[ Edit inline ] [ Edit ] [ Create PHP code ]
```

The interface includes tabs for Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, and Triggers.

13.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

b.title,

a.first_name

FROM books b

INNER JOIN authors a

ON b.author_id = a.author_id;

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
- Database:** library_db
- Table:** authors
- Query Result:**

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

Showing rows 0 - 2 (3 total, Query took 0.0004 seconds.)

title	first_name
The Secrets	Rhoda
Mahabharat	vyasa
Ramayana	Valmiki

Extra options: Show all, Number of rows: 25, Filter rows: Search this table, Sort by key: None

Query results operations: Print, Copy to clipboard, Export, Display chart, Create view, Console

- **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

b.title,

a.first_name

FROM books b

FULL OUTER JOIN authors a

ON b.author_id = a.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;
```

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
- Database:** library_db
- Table:** books
- Query Result:**

```
Showing rows 0 - 3 (4 total, Query took 0.0005 seconds.)  
SELECT genre, COUNT(*) AS total_books FROM books GROUP BY genre;  
Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]
```

genre	total_books
Adventure	1
fiction	1
Historical	1
Mythological	2

Show all | Number of rows: 25 | Filter rows: Search this table

Extra options

Query results operations

Print | Copy to clipboard | Export | Display chart | Create view

- **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 total_members  
FROM members_backup  
GROUP BY YEAR(date_of_membership)  
ORDER BY join_year;
```

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
- Database:** library_db
- Table:** members_backup
- Query:** SELECT YEAR(date_of_membership) AS join_year, COUNT(*) AS total_members FROM members_backup GROUP BY YEAR(date_of_membership) ORDER BY join_year;
- Result:** A single row is displayed:

join_year	total_members
2023	3
- Operations:** Buttons for Print, Copy to clipboard, Export, Display chart, and Create view are visible.

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 authorName VARCHAR(100)
```

```
)
```

```
BEGIN
```

```
    SELECT
```

```
        b.book_id,
```

```
        b.title,
```

```
        b.price
```

```
    FROM books b
```

```
    INNER JOIN authors a
```

```
        ON b.author_id = a.author_id
```

```
    WHERE a.first_name = authorName;
```

```
END //
```

DELIMITER ;

phpMyAdmin

Recent | Favorites

Server: 127.0.0.1 » Database: library_db

Structure | SQL | Search | Query | Export | Import | Operations | Privileges | Routines | Events | More

New information_schema library_db

library_db Procedures Tables

Authors books members_backup

mysql performance_schema phpmyadmin school_db test university_db

Your SQL query has been executed successfully.
1 row affected by the last statement inside the procedure.

SET @p0='vyasa'; CALL `GetBooksByAuthor` (@p0);

Execution results of routine 'GetBooksByAuthor'

book_id	title	price
2	Mahabharat	132.00

Routines

Check all

Search

Name	Type	Returns
GetBooksByAuthor	PROCEDURE	

Edit

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

DELIMITER //

```
CREATE PROCEDURE GetBookPrice (
    IN bookId INT,
    OUT bookPrice DECIMAL(10,2)
)
BEGIN
    SELECT price
    INTO bookPrice
    FROM books
    WHERE book_id = bookId;
END //
```

DELIMITER ;

The screenshot shows the phpMyAdmin interface for a MySQL database named 'library_db'. The left sidebar shows the database structure with 'Procedures' expanded, showing 'GetBookPrice' and 'GetBooksByAuthor'. The main area has a green status bar at the top stating 'Your SQL query has been executed successfully.' and '1 row affected by the last statement inside the procedure.' Below this, a query window contains the following SQL code:

```
SET @p0='2'; SET @p1=''; CALL `GetBookPrice`(@p0, @p1); SELECT @p1 AS `bookPrice`;
```

An 'Execution results of routine 'GetBookPrice'' section shows the output:

bookPrice
132.00

Below this is a 'Routines' section with a table:

Name	Type	Returns
GetBookPrice	PROCEDURE	Edit Execute Export Drop
GetBooksByAuthor	PROCEDURE	Edit Execute Export Drop

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_view AS
```

```
SELECT
```

```
    b.title,  
    a.first_name AS author,  
    b.price
```

```
FROM books b
```

```
INNER JOIN authors a
```

```
    ON b.author_id = a.author_id;
```

The screenshot shows the phpMyAdmin interface for a database named 'library_db'. On the left, the database structure is visible, including the 'Tables' section which contains 'authors', 'books', and 'book_details_view'. The 'book_details_view' table is selected. The main panel displays the SQL query used to create the view:

```
SELECT * FROM `book_details_view`
```

Below the query, the results are shown in a grid:

	title	author	price	
<input type="checkbox"/>	Edit	Copy	Delete	The Secrets Rhoda 110.00
<input type="checkbox"/>	Edit	Copy	Delete	Mahabharat vyasa 132.00
<input type="checkbox"/>	Edit	Copy	Delete	Ramayana Valmiki 1320.00

At the bottom of the interface, there are buttons for 'Edit', 'Copy', 'Delete', and 'Export'.

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

CREATE VIEW members_before_2020 AS

SELECT

```
member_id,  
member_name,  
date_of_membership
```

FROM members_backup

WHERE date_of_membership < '2020-01-01';

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
- Database:** library_db
- View:** members_before_2020
- SQL Tab:** Contains the SQL query:

```
SELECT * FROM `members_before_2020`
```
- Operations Tab:** Shows the results of the query, which is empty (zero rows). A message indicates: "MySQL returned an empty result set (i.e. zero rows). (Query took 0.0004 seconds.)"
- Structure Tab:** Available but not selected.
- Browse Tab:** Available but not selected.
- Search Tab:** Available but not selected.
- Insert Tab:** Available but not selected.
- Export Tab:** Available but not selected.
- Privileges Tab:** Available but not selected.
- Operations Tab:** Available but not selected.

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.

```
books(  
    book_id INT,  
    title VARCHAR(100),  
    price DECIMAL(10,2),  
    last_modified TIMESTAMP  
)  
DELIMITER //
```

```
CREATE TRIGGER trg_update_last_modified  
BEFORE UPDATE ON books  
FOR EACH ROW  
BEGIN  
    SET NEW.last_modified = CURRENT_TIMESTAMP;  
END //
```

```
DELIMITER ;
```

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

```
DELIMITER //
```

```
CREATE TRIGGER trg_log_book_delete
AFTER DELETE ON books
FOR EACH ROW
BEGIN
    INSERT INTO log_changes (book_id, action, action_date)
    VALUES (OLD.book_id, 'DELETE', CURRENT_TIMESTAMP);
END //
```

```
DELIMITER ;
```

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.**

```
BEGIN
```

```
    INSERT INTO books (book_id, title, author, price)  
    VALUES (101, 'Database Management Systems', 'C.J. Date', 550);
```

```
    DBMS_OUTPUT.PUT_LINE('New book inserted successfully.');
```

```
END;
```

```
/
```

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

```
DECLARE
```

```
    total_books NUMBER;
```

```
BEGIN
```

```
    SELECT COUNT(*)  
    INTO total_books  
    FROM books;
```

```
    DBMS_OUTPUT.PUT_LINE('Total number of books: ' || total_books);
```

```
END;
```

```
/
```

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.**

```
DECLARE
    v_book_id  NUMBER := 101;
    v_price    NUMBER := 450;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Book ID: ' || v_book_id);
    DBMS_OUTPUT.PUT_LINE('Book Price: ' || v_price);
END;
/
```

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

```
DECLARE
    c_discount_rate CONSTANT NUMBER := 0.10; -- 10% discount
    v_original_price NUMBER := 500;
    v_discount_amount NUMBER;
    v_final_price NUMBER;

BEGIN
    v_discount_amount := v_original_price * c_discount_rate;
    v_final_price := v_original_price - v_discount_amount;

    DBMS_OUTPUT.PUT_LINE('Original Price: ' || v_original_price);
    DBMS_OUTPUT.PUT_LINE('Discount Amount: ' || v_discount_amount);
    DBMS_OUTPUT.PUT_LINE('Final Price after Discount: ' || v_final_price);

END;
/
```

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.**

```
DECLARE
```

```
    v_price NUMBER := 120;
```

```
BEGIN
```

```
    IF v_price > 100 THEN
```

```
        DBMS_OUTPUT.PUT_LINE('The book price is above $100.');
```

```
    ELSE
```

```
        DBMS_OUTPUT.PUT_LINE('The book price is $100 or below.');
```

```
    END IF;
```

```
END;
```

```
/
```

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

```
BEGIN  
FOR rec IN (SELECT book_id, title, price FROM books) LOOP  
    DBMS_OUTPUT.PUT_LINE(  
        'Book ID: ' || rec.book_id ||  
        ', Title: ' || rec.title ||  
        ', Price: ' || rec.price  
    );  
END LOOP;  
END;  
/
```

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.

```
DECLARE  
    CURSOR c_members IS  
        SELECT member_id, name, join_date FROM members;  
  
    v_member_id members.member_id%TYPE;  
    v_name     members.name%TYPE;  
    v_join_date members.join_date%TYPE;  
  
BEGIN  
    OPEN c_members;  
  
    LOOP  
        FETCH c_members INTO v_member_id, v_name, v_join_date;  
        EXIT WHEN c_members%NOTFOUND;  
  
        DBMS_OUTPUT.PUT_LINE(  
            'Member ID: ' || v_member_id ||  
            ', Name: ' || v_name ||  
            ', Join Date: ' || v_join_date  
        );  
    END LOOP;
```

```
CLOSE c_members;  
END;
```

```
/
```

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

```
DECLARE
```

```
    CURSOR c_books IS  
        SELECT title  
        FROM books  
        WHERE author = 'C.J. Date';
```

```
    v_title books.title%TYPE;
```

```
BEGIN
```

```
    OPEN c_books;
```

```
LOOP
```

```
    FETCH c_books INTO v_title;  
    EXIT WHEN c_books%NOTFOUND;
```

```
    DBMS_OUTPUT.PUT_LINE('Book Title: ' || v_title);
```

```
END LOOP;
```

```
CLOSE c_books;
```

```
END;
```

```
/
```

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.

-- Insert a new member

```
INSERT INTO members_backup (member_id, member_name,  
date_of_membership)
```

```
VALUES (201, 'Amit Patel', SYSDATE);
```

```
SAVEPOINT after_insert;
```

```
UPDATE members
```

```
SET name = 'Amit P.'
```

```
WHERE member_id = 201;
```

```
ROLLBACK TO after_insert;
```

```
COMMIT;
```

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

```
INSERT INTO books (book_id, title, price)  
VALUES (301, 'SQL Basics', 250);
```

```
INSERT INTO books (book_id, title, price)  
VALUES (302, 'Advanced PL/SQL', 400);
```

```
COMMIT;
```

```
SAVEPOINT book_changes;  
  
UPDATE books  
SET price = 300  
WHERE book_id = 301;
```

```
DELETE FROM books  
WHERE book_id = 302;
```

```
ROLLBACK TO book_changes;  
COMMIT;
```