

Introduction to DBMS

1. Introduction to SQL

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

Answer:

Query 1: Create Database

```
CREATE DATABASE school_db;
```

Query 2: Use Database

```
USE school_db;
```

Query 3: Create Table students

```
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. On the left, the database structure is visible, showing a database named 'school_db' containing a table named 'students'. The main panel displays the results of a query: 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0003 seconds.)'. Below this, the SQL query 'SELECT * FROM `students`' is shown. The results section is empty, indicating no data has been inserted into the table yet.

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

Answer:

Query 1: Insert 5 Records

```
INSERT INTO students (student_id, student_name, age, class, address) VALUES  
(1, 'Rahul Sharma', 12, '6A', 'Ahmedabad'),  
(2, 'Priya Patel', 13, '7B', 'Surat'),  
(3, 'Amit Shah', 11, '5C', 'Vadodara'),  
(4, 'Neha Mehta', 14, '8A', 'Rajkot'),  
(5, 'Karan Joshi', 12, '6B', 'Bhavnagar');
```

Query 2: Retrieve All Records

```
SELECT * FROM students;
```

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1 > Database: school_db > Table: students
- Toolbar:** Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, Tracking, Triggers
- Query Result:** Shows the output of the query "SELECT * FROM `students`". It displays 5 rows of data with columns: student_id, student_name, age, class, and address.
- Data:**

student_id	student_name	age	class	address
1	Rahul Sharma	12	6A	Ahmedabad
2	Priya Patel	13	7B	Surat
3	Amit Shah	11	5C	Vadodara
4	Neha Mehta	14	8A	Rajkot
5	Karan Joshi	12	6B	Bhavnagar

2. SQL Syntax

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

Answer:

Query: Retrieve Specific Columns (student_name and age)

`SELECT student_name, age FROM students;`

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. The left sidebar lists databases and tables, including 'students'. The main area displays the results of the query `SELECT student_name, age FROM students;`. The results table has columns 'student_name' and 'age', showing five rows of data: Rahul Sharma (12), Priya Patel (13), Amit Shah (11), Neha Mehta (14), and Karan Joshi (12). Each row includes edit, copy, delete, and export options.

student_name	age
Rahul Sharma	12
Priya Patel	13
Amit Shah	11
Neha Mehta	14
Karan Joshi	12

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

Answer:

Query: Retrieve All Students Whose Age > 10

`SELECT * FROM students WHERE age > 10;`

The screenshot shows the phpMyAdmin interface for the same 'school_db'. The results of the query `SELECT * FROM students WHERE age > 10;` are displayed. The results table includes columns 'student_id', 'student_name', 'age', 'class', and 'address'. The rows show students with ages 12, 13, 11, 14, and 12, corresponding to the previous results. The 'age' column is explicitly labeled in the query results.

student_id	student_name	age	class	address
1	Rahul Sharma	12	6A	Ahmedabad
2	Priya Patel	13	7B	Surat
3	Amit Shah	11	5C	Vadodara
4	Neha Mehta	14	8A	Rajkot
5	Karan Joshi	12	6B	Bhavnagar

3. SQL Constraints

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

Answer:

Query: Create teachers Table

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

The screenshot shows the phpMyAdmin interface for MySQL version 5.7.24. The left sidebar lists databases: employee_information_table, information_schema, mysql, performance_schema, phpmyadmin, school_db, test, and world_data. The 'school_db' database is selected. The main area shows the 'Table structure' for the 'teachers' table. The table has four columns: 'teacher_id' (int(11), primary key, not null), 'teacher_name' (varchar(50), not null), 'subject' (varchar(50), not null), and 'email' (varchar(100), unique, not null). Below the table structure, the 'Indexes' section shows two indexes: 'teacher_id' (PRIMARY, BTREE, column 0, cardinality 1, collation A, not null) and 'email' (BTREE, column 0, cardinality 1, collation A, yes).

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit	teacher_id	PRIMARY	BTREE	Yes	teacher_id	0	A	No	
Edit	email	BTREE	Yes	No	email	0	A	Yes	

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

Answer:

Query 1: Add teacher_id column to students table

`ALTER TABLE students ADD teacher_id INT;`

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	student_id	int(11)			No	None			Change Drop More
2	student_name	varchar(50)	utf8mb4_general_ci		Yes	NULL			Change Drop More
3	age	int(11)			Yes	NULL			Change Drop More
4	class	varchar(20)	utf8mb4_general_ci		Yes	NULL			Change Drop More
5	address	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More
6	teacher_id	int(11)			Yes	NULL			Change Drop More

Add 1 column(s) after teacher_id [Go](#)

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Rename Drop	PRIMARY	BTREE	Yes	No	student_id	5	A	No	

Create an index on 1 columns [Go](#)

Query 2: Add FOREIGN KEY Constraint

`ALTER TABLE students`

`ADD CONSTRAINT fk_teacher`

`FOREIGN KEY (teacher_id)`

`REFERENCES teachers(teacher_id);`

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	student_id	int(11)			No	None			Change Drop More
2	student_name	varchar(50)	utf8mb4_general_ci		Yes	NULL			Change Drop More
3	age	int(11)			Yes	NULL			Change Drop More
4	class	varchar(20)	utf8mb4_general_ci		Yes	NULL			Change Drop More
5	address	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More
6	teacher_id	int(11)			Yes	NULL			Change Drop More

Add 1 column(s) after teacher_id [Go](#)

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Rename Drop	PRIMARY	BTREE	Yes	No	student_id	5	A	No	
Edit Rename Drop	fk_teacher	BTREE	No	No	teacher_id	2	A	Yes	

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

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

Answer:

Query: Create a table courses

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

The screenshot shows the phpMyAdmin interface for a MySQL database named 'school_db'. On the left, the database structure is shown with tables like 'employee_information_table', 'information_schema', 'mysql', 'performance_schema', 'phpmyadmin', 'school_db', 'students', 'teachers', 'test', and 'world_data'. In the main panel, the 'Table structure' tab is selected for the 'courses' table. The table has three columns: 'course_id' (int(11), primary key, null), 'course_name' (varchar(100), collation utf8mb4_general_ci, null), and 'course_credits' (int(11), null). An index is defined on 'course_id' with type BTREE, unique Yes, and cardinality 0. Below the table structure, there is a section for creating an index on 1 column(s).

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

Answer:

Query: Create a Database university_db

```
CREATE DATABASE university_db;
```

The screenshot shows the phpMyAdmin interface for a MySQL database named 'university_db'. On the left, the database structure is shown with tables like 'employee_information_table', 'information_schema', 'mysql', 'performance_schema', 'phpmyadmin', 'school_db', 'students', 'teachers', 'test', and 'world_data'. In the main panel, the 'Structure' tab is selected for the 'university_db' database. A message at the top states 'No tables found in database'. Below it, there is a 'Create new table' form with 'Table name' and 'Number of columns' fields, both currently empty. There is also a 'Create' button.

5. ALTER Command

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

Answer:

Query: Add a New Column (course_duration) to courses Table

ALTER TABLE courses ADD course_duration VARCHAR(50);

The screenshot shows the phpMyAdmin interface for the 'courses' table. In the 'Structure' tab, the columns are listed as follows:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	course_id	int(11)			No	None			Change Drop More
2	course_name	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More
3	course_credits	int(11)			Yes	NULL			Change Drop More
4	course_duration	varchar(50)	utf8mb4_general_ci		Yes	NULL			Change Drop More

Below the table structure, there is a section for 'Indexes' with one index defined:

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

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

Answer:

Query: Drop the Column (course_credits) from courses Table

ALTER TABLE courses DROP COLUMN course_credits;

The screenshot shows the phpMyAdmin interface for the 'courses' table. In the 'Structure' tab, the columns are listed as follows:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	course_id	int(11)			No	None			Change Drop More
2	course_name	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More
3	course_duration	varchar(50)	utf8mb4_general_ci		Yes	NULL			Change Drop More

Below the table structure, there is a section for 'Indexes' with one index defined:

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

6. DROP Command

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

Answer:

Query: Drop the teachers table

DROP TABLE teachers;

The screenshot shows the phpMyAdmin interface. On the left, the database tree is visible with 'school_db' selected. In the center, the 'SQL' tab is active. A query is entered in the text area: 'DROP TABLE teachers;'. Below the text area, a message box displays: 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0011 seconds)'. At the bottom of the screen, there are three buttons: 'Edit inline', 'Edit', and 'Create PHP code'.

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

Answer:

Query 1: Drop the students table and verify removal

DROP TABLE students;

The screenshot shows the phpMyAdmin interface. On the left, the database tree is visible with 'school_db' selected. In the center, the 'SQL' tab is active. A query is entered in the text area: 'DROP TABLE students;'. Below the text area, a message box displays: 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0014 seconds)'. At the bottom of the screen, there are three buttons: 'Edit inline', 'Edit', and 'Create PHP code'.

Query 2: Verify that the table is removed

SHOW TABLES;

The screenshot shows the phpMyAdmin interface for a MySQL database named 'school_db'. The left sidebar lists databases like 'information_schema', 'mysql', 'performance_schema', 'phpmyadmin', and 'school_db'. The 'school_db' database is selected, showing tables such as 'employee_information_table' and 'courses'. The main query results area displays the output of the 'SHOW TABLES' command:

```
SHOW TABLES;
```

Your SQL query has been executed successfully.

Tables_in_school_db

Tables_in_school_db
courses

Query results operations

Print Copy to clipboard Create view

Bookmark this SQL query

Label: Let every user access this bookmark

Bookmark this SQL query

7. Data Manipulation Language (DML)

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

Answer:

Query: Insert 3 Records into courses Table

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

VALUES

```
(101, 'Computer Science Basics', '3 Months'),
```

```
(102, 'Database Management', '2 Months'),
```

```
(103, 'Web Development', '4 Months');
```

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The 'courses' table is selected. The data grid displays three rows of course information:

	course_id	course_name	course_duration
<input type="checkbox"/>	101	Computer Science Basics	3 Months
<input type="checkbox"/>	102	Database Management	2 Months
<input type="checkbox"/>	103	Web Development	4 Months

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

Answer:

Query: Update Course Duration of a Specific Course

Example: Update duration of course_id = 102

```
UPDATE courses SET course_duration = '3.5 Months' WHERE course_id = 102;
```

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The 'courses' table is selected. The data grid displays three rows of course information, with the second row updated:

	course_id	course_name	course_duration
<input type="checkbox"/>	101	Computer Science Basics	3 Months
<input type="checkbox"/>	102	Database Management	3.5 Months
<input type="checkbox"/>	103	Web Development	4 Months

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

Answer:

Query: Delete a course using DELETE Command

Example: Delete course with course_id = 103

DELETE FROM courses **WHERE** course_id = 103;

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The 'courses' table is selected. The table structure is shown with columns: course_id, course_name, and course_duration. There are three rows of data:

course_id	course_name	course_duration
101	Computer Science Basics	3 Months
102	Database Management	3.5 Months
103	Software Engineering	4 Months

The row for course_id 103 is highlighted. The 'Edit' and 'Delete' buttons are visible for each row. The 'Delete' button for course_id 103 is highlighted.

8. Data Query Language (DQL)

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

Answer:

Query: Retrieve all courses

```
SELECT * FROM courses;
```

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. The left sidebar lists databases and tables, including 'courses'. The main area displays the results of the query 'SELECT * FROM courses;'. The results table has columns: course_id, course_name, and course_duration. Two rows are shown: one for 'Computer Science Basics' (3 Months) and another for 'Database Management' (3.5 Months). The 'course_id' column is sorted in descending order.

course_id	course_name	course_duration
101	Computer Science Basics	3 Months
102	Database Management	3.5 Months

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

Answer:

Query: Sort Courses by course_duration (Descending Order)

```
SELECT * FROM courses ORDER BY course_duration DESC;
```

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. The left sidebar lists databases and tables, including 'courses'. The main area displays the results of the query 'SELECT * FROM courses ORDER BY course_duration DESC;'. The results table has columns: course_id, course_name, and course_duration. The rows are sorted by course_duration in descending order, with 'Database Management' (3.5 Months) at the top and 'Computer Science Basics' (3 Months) at the bottom. The 'course_id' column is also sorted in descending order.

course_id	course_name	course_duration
102	Database Management	3.5 Months
101	Computer Science Basics	3 Months

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

Answer:

Query: Show only top 2 courses using LIMIT

`SELECT * FROM courses ORDER BY course_duration DESC LIMIT 2;`

The screenshot shows the phpMyAdmin interface for a MySQL database named 'school_db'. The left sidebar lists databases like 'employee_information_table', 'information_schema', 'mysql', 'performance_schema', 'phpmyadmin', 'school_db', 'test', 'university_db', and 'world_data'. The 'courses' table under 'school_db' is selected. The main area displays the results of the query: 'SELECT * FROM courses ORDER BY course_duration DESC LIMIT 2;'. The results table shows two rows:

	course_id	course_name	course_duration
102	Database Management	3.5 Months	
101	Computer Science Basics	3 Months	

Below the table are 'Query results operations' buttons: Print, Copy to clipboard, Export, Display chart, and Create view.

9. Data Control Language (DCL)

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

Answer:

Query 1: Create two new users user1 and user2

Step 1: Create user1

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

Step 2: Create user2

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

User	Host
root	127.0.0.1
root	%
pma	localhost
root	localhost
user1	localhost
user2	localhost

Query 2: Grant SELECT permission on courses table to user1

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

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

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

Answer:

Query 1: Revoke INSERT permission from user1

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

Query 2: Grant INSERT permission to user2

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

The screenshot shows the phpMyAdmin interface for a database named 'school db'. On the left, the database structure is visible, including tables like 'employee_information_table', 'information_schema', 'mysql', 'performance_schema', and 'phpmyadmin'. The main area displays two sets of grants:

Grants for user1@localhost:

```
GRANT USAGE ON *.* TO user1@localhost IDENTIFIED BY '';
GRANT SELECT ON `school_db`.`courses` TO 'user1'@'localhost';
GRANT INSERT ON `school_db`.`courses` TO 'user1'@'localhost';
```

Grants for user2@localhost:

```
SHOW GRANTS FOR 'user2'@'localhost';
GRANT USAGE ON *.* TO user2@localhost IDENTIFIED BY '';
GRANT INSERT ON `school_db`.`courses` TO 'user2'@'localhost';
```

A message at the bottom indicates: "Your SQL query has been executed successfully."

10. Transaction Control Language (TCL)

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

Answer:

Query: Insert a few rows + COMMIT

START TRANSACTION;

INSERT INTO courses (course_id, course_name, course_duration)

VALUES

(104, 'Cyber Security', '2 Months'),

(105, 'Data Analytics', '3 Months');

COMMIT;

	course_id	course_name	course_duration
<input type="checkbox"/>	101	Computer Science Basics	3 Months
<input type="checkbox"/>	102	Database Management	3.5 Months
<input type="checkbox"/>	104	Cyber Security	2 Months
<input type="checkbox"/>	105	Data Analytics	3 Months

- Rows 104 and 105 will appear permanently.

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

Answer:

Query: Insert additional rows + ROLLBACK

START TRANSACTION;

INSERT INTO courses (course_id, course_name, course_duration)

VALUES (106, 'Cloud Computing', '5 Months');

ROLLBACK;

course_id	course_name	course_duration
101	Computer Science Basics	3 Months
102	Database Management	3.5 Months
104	Cyber Security	2 Months
105	Data Analytics	3 Months

- **Course_id 106 will NOT appear because rollback cancelled the insert.**

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

Answer:

Query: SAVEPOINT + ROLLBACK TO SAVEPOINT

START TRANSACTION;

-- Create savepoint

SAVEPOINT before_update;

-- Update a course

UPDATE courses SET course_duration = '6 Months' WHERE course_id = 101;

-- Rollback only this update

ROLLBACK TO before_update;

-- Commit remaining changes

COMMIT;

course_id	course_name	course_duration
101	Computer Science Basics	3 Months
102	Database Management	3.5 Months
104	Cyber Security	2 Months
105	Data Analytics	3 Months

- Course_id 101 duration will remain '3 Months' (rollback undo the update)

11. SQL Joins

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

Answer:

Query 1: Create two tables

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

The screenshot shows the 'Table structure' tab for the 'departments' table in the 'school_db' database. The table has two columns: 'dept_id' (INT, Primary Key) and 'dept_name' (VARCHAR(50)). An index named 'dept_id' is present on the 'dept_id' column. The 'Indexes' section shows the primary key and the index.

```
2. CREATE TABLE employees (
```

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

The screenshot shows the 'Table structure' tab for the 'employees' table in the 'school_db' database. The table has three columns: 'emp_id' (INT, Primary Key), 'emp_name' (VARCHAR(50)), and 'dept_id' (INT). An index named 'dept_id' is present on the 'dept_id' column. The 'Indexes' section shows the primary key and the index.

Query 2: Insert sample records

1. INSERT INTO departments VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT');

The screenshot shows the phpMyAdmin interface for the 'departments' table in the 'school_db' database. The table structure is displayed with columns 'dept_id' and 'dept_name'. Three rows have been inserted:

dept_id	dept_name
1	HR
2	Finance
3	IT

2. INSERT INTO employees VALUES

(101, 'Anita', 1),

(102, 'Rahul', 2),

(103, 'Meera', 3);

The screenshot shows the phpMyAdmin interface for the 'employees' table in the 'school_db' database. The table structure is displayed with columns 'emp_id', 'emp_name', and 'dept_id'. Three rows have been inserted:

emp_id	emp_name	dept_id
101	Anita	1
102	Rahul	2
103	Meera	3

Query 3: Perform INNER JOIN

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

The screenshot shows the phpMyAdmin interface. On the left, the database structure is visible with various schemas and tables listed. The main area displays the results of the executed SQL query:

```
Showing rows 0 - 2 (3 total, Query took 0.0010 seconds)  
SELECT employees.emp_id, employees.emp_name, departments.dept_name FROM employees INNER JOIN departments ON employees.dept_id = departments.dept_id;
```

The results table shows the following data:

emp_id	emp_name	dept_name
101	Anita	HR
102	Rahul	Finance
103	Meera	IT

At the bottom, there are options for printing or exporting the results.

- INNER JOIN shows **only matching** employee – department records.

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

Answer:

Query: LEFT JOIN (Show all departments even without employees)

`SELECT departments.dept_id, departments.dept_name, employees.emp_name`

`FROM departments`

`LEFT JOIN employees ON departments.dept_id = employees.dept_id;`

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. The left sidebar lists various databases and tables. The main area displays the results of a SQL query:

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

The results table shows the following data:

dept_id	dept_name	emp_name
1	HR	Anita
2	Finance	Rahul
3	IT	Meera

12. SQL Group By

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

Answer:

Query: GROUP BY – Count Employees in Each Department

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

```
GROUP BY dept_id;
```

The screenshot shows the phpMyAdmin interface. On the left, the database structure is visible with the 'Tables' section expanded, showing 'dept_id' and 'total_employees'. In the center, the query results are displayed in a table:

dept_id	total_employees
1	1
2	1
3	1

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

Answer:

Important note:

- To use Lab 2, your employees table must have a salary column, e.g.:

```
ALTER TABLE employees ADD salary INT;
```

- And insert/update salaries:

```
UPDATE employees SET salary = 30000 WHERE emp_id = 101;
```

```
UPDATE employees SET salary = 45000 WHERE emp_id = 102;
```

```
UPDATE employees SET salary = 50000 WHERE emp_id = 103;
```

The screenshot shows the phpMyAdmin interface for the 'employees' table in the 'school_db' database. The table has four columns: emp_id, emp_name, dept_id, and salary. The data is as follows:

emp_id	emp_name	dept_id	salary
101	Anita	1	30000
102	Rahul	2	45000
103	Meera	3	50000

Query: Find Average Salary per Department (Using AVG)

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

The screenshot shows the phpMyAdmin interface running the provided SQL query. The results are displayed in a table with two columns: dept_id and average_salary.

dept_id	average_salary
1	30000.0000
2	45000.0000
3	50000.0000

13. SQL Stored Procedure

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

Answer:

Query: Stored Procedure to Retrieve Employees by Department

- **Stored Procedure**

DELIMITER \$\$

CREATE PROCEDURE GetEmployeesByDepartment(**IN** dept **INT**)

BEGIN

 SELECT emp_id, emp_name, dept_id

 FROM employees

 WHERE dept_id = dept;

END \$\$

DELIMITER ;

- **How to Execute the Procedure**

CALL GetEmployeesByDepartment(2);

The screenshot shows the phpMyAdmin interface. In the left sidebar, the database structure is visible, including the 'employees' table under the 'school_db' schema. The main area shows the results of a query: 'Showing rows 0 - 0 (1 total, Query took 0.0104 seconds.)'. Below this, the SQL command 'CALL GetEmployeesByDepartment(2);' is shown. The results table displays one row: emp_id 102, emp_name Rahul, and dept_id 2. At the bottom, there are 'Query results operations' buttons for Print, Copy to clipboard, and Create view.

- This procedure returns employees who work in a specific department.

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

Answer:

Query: Stored Procedure to Retrieve Course Details by course_id

• **Stored Procedure**

```
DELIMITER $$
```

```
CREATE PROCEDURE GetCourseDetails(IN c_id INT)
```

```
BEGIN
```

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

```
    FROM courses
```

```
    WHERE course_id = c_id;
```

```
END $$
```

```
DELIMITER ;
```

• **How to Execute the Procedure**

```
CALL GetCourseDetails(101);
```

The screenshot shows the phpMyAdmin interface. On the left, the database structure is visible with the 'employees' table selected. In the main area, a query has been run: `CALL GetCourseDetails(101);`. The results show a single row from the 'courses' table:

course_id	course_name	course_duration
101	Computer Science Basics	3 Months

- This procedure returns only the details of the course whose ID you pass.

14. SQL View

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

Answer:

Query: Create a View to Show Employees with Department Names

- **Step 1: Create a VIEW using JOIN**

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

- **Step 2: To see the view**

```
SELECT * FROM employee_department_view;
```

The screenshot shows the phpMyAdmin interface with the database 'school_db' selected. In the left sidebar, the 'Views' section is expanded, showing the 'employee_department_view'. The main area displays the results of the query 'SELECT * FROM employee_department_view;'. The results are presented in a table with columns: emp_id, emp_name, salary, and dept_name. The data shows three rows: (101, Anita, 30000, HR), (102, Rahul, 45000, Finance), and (103, Meera, 50000, IT). Below the table, there are buttons for Edit, Copy, Delete, and Export.

emp_id	emp_name	salary	dept_name
101	Anita	30000	HR
102	Rahul	45000	Finance
103	Meera	50000	IT

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

Answer:

Query: Modify the View to Exclude Employees with Salary < 50,000

- **Use OR REPLACE to update the view**

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

- To check updated view

```
SELECT * FROM employee_department_view;
```

The screenshot shows the phpMyAdmin interface for a MySQL database named 'school_db'. The left sidebar lists databases, schemas, tables, and views. The main area displays the results of a SQL query:

```
SELECT * FROM `employee_department_view`
```

The results show one row:

	emp_id	emp_name	salary	dept_name
	103	Meera	50000	IT

Below the table, there are buttons for Edit, Copy, Delete, and Export. At the bottom, there are links for Print, Copy to clipboard, Export, Display chart, Create view, and Bookmark this SQL query.

15. SQL Triggers

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

Answer:

Query: Trigger to Log New Employee Insertions

- **Step 1: Create a log table**

You need a table to store log records.

```
CREATE TABLE employee_log (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    emp_id INT,
    emp_name VARCHAR(50),
    action_performed VARCHAR(20),
    log_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

The screenshot shows the phpMyAdmin interface for the school_db database. On the left, the database structure is visible with tables like employee_information_table, courses, departments, employees, and employee_log. The main window displays the 'Table structure' for the employee_log table. The table has five columns: log_id (int(11), primary key, auto_increment), emp_id (int(11)), emp_name (varchar(50)), action_performed (varchar(20)), and log_time (timestamp, default current_timestamp). Below the table structure, there are tabs for 'Indexes' and 'Triggers'. The 'Indexes' tab shows a primary key on log_id. The 'Triggers' tab is currently empty.

- **Step 2: Create the trigger**

This trigger will run AFTER a new employee is inserted.

DELIMITER \$\$

```
CREATE TRIGGER log_new_employee
```

```
AFTER INSERT ON employees
```

```
FOR EACH ROW
```

```
BEGIN
```

```

INSERT INTO employee_log (emp_id, emp_name, action_performed)
VALUES (NEW.emp_id, NEW.emp_name, 'INSERT');

END $$
```

DELIMITER ;

- **Sample Output**

(After inserting a new employee)

```
INSERT INTO employees (emp_id, emp_name, dept_id, salary)
```

```
VALUES (107, 'Suresh', 2, 48000);
```

emp_id	emp_name	dept_id	salary
101	Anita	1	30000
102	Rahul	2	45000
103	Meera	3	50000
107	Suresh	2	48000

- **Running:**

```
SELECT * FROM employee_log;
```

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

Answer:

Query: Trigger to Automatically Update last_modified Timestamp

- **Step 1: Add a column to employees table**

```
ALTER TABLE employees
```

```
ADD last_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP;
```

- **Step 2: Create the UPDATE trigger**

This trigger updates the timestamp whenever an employee record changes.

DELIMITER \$\$

CREATE TRIGGER update_last_modified

BEFORE UPDATE ON employees

FOR EACH ROW

BEGIN

SET NEW.last_modified = CURRENT_TIMESTAMP;

END \$\$

DELIMITER ;

- **Example Update**

UPDATE employees

SET salary = 55000

WHERE emp_id = 101;

The screenshot shows the phpMyAdmin interface for the 'employees' table. The table structure is as follows:

	emp_id	emp_name	dept_id	salary
1	101	Anita	1	55000
2	102	Rahul	2	45000
3	103	Meera	3	50000
4	107	Suresh	2	48000

- **Output (View the updated timestamp)**

SELECT emp_id, emp_name, salary, last_modified **FROM** employees;

The screenshot shows the phpMyAdmin interface for the 'employees' table after the update. The table structure is as follows:

	emp_id	emp_name	dept_id	salary	last_modified
1	101	Anita	1	55000	2025-11-19 19:51:02
2	102	Rahul	2	45000	2025-11-19 19:51:02
3	103	Meera	3	50000	2025-11-19 19:51:02
4	107	Suresh	2	48000	2025-11-19 19:51:02

- Automatically updated — no need to manually set last_modified.

16. Introduction to PL/SQL

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

Answer:

Query: Print total number of employees

```
SELECT COUNT(*) AS total_employees FROM employees;
```

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. In the left sidebar, under the 'Tables' section, there is a table named 'employees'. The main query editor window contains the following PL/SQL code:

```
CALL total_employees();
```

The results pane shows a single row with the value '4' under the column 'total_employees'. Below the results, there are options to 'Print', 'Copy to clipboard', 'Create view', and 'Bookmark this SQL query'.

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

Answer:

Query: Calculate Total Salary

```
SELECT SUM(salary) AS total_salary FROM employees;
```

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. In the left sidebar, under the 'Tables' section, there is a table named 'employees'. The main query editor window contains the following PL/SQL code:

```
SELECT SUM(salary) AS total_salary FROM employees;
```

The results pane shows a single row with the value '198000' under the column 'total_salary'. Below the results, there are options to 'Print', 'Copy to clipboard', 'Export', 'Display chart', 'Create view', and 'Bookmark this SQL query'.

17. PL/SQL Control Structures

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

Answer:

Query 1: Stored Procedure

```
DELIMITER $$
```

```
CREATE PROCEDURE check_department(IN p_emp_id INT)
```

```
BEGIN
```

```
    DECLARE v_dept INT;
```

```
    SELECT dept_id INTO v_dept
```

```
    FROM employees
```

```
    WHERE emp_id = p_emp_id;
```

```
    IF v_dept = 2 THEN
```

```
        SELECT 'Employee belongs to Department 2' AS message;
```

```
    ELSE
```

```
        SELECT 'Employee does NOT belong to Department 2' AS message;
```

```
    END IF;
```

```
    END $$
```

```
DELIMITER ;
```

Query 2: Call the Procedure

```
CALL check_department(102);
```

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. In the left sidebar, there are sections for 'Recent' and 'Favorites', and a tree view of the database schema under 'Tables', 'Procedures', and 'Views'. The main area displays the results of a SQL query:

```
SET @p0='102'; CALL `check_department`(@p0);
Execution results of routine 'check_department'
message
Employee belongs to Department 2.
```

Below this, the 'Routines' section lists four stored procedures:

Name	Type	Returns
GetCourseDetails	PROCEDURE	Edit Execute Export Drop
GetEmployeesByDepartment	PROCEDURE	Edit Execute Export Drop
check_department	PROCEDURE	Edit Execute Export Drop
total_employees	PROCEDURE	Edit Execute Export Drop

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

Answer:

Query 1: Stored Procedure with FOR-LIKE LOOP (MySQL uses WHILE)

DELIMITER \$\$

```
CREATE PROCEDURE show_employee_names()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE v_name VARCHAR(50);

    DECLARE emp_cursor CURSOR FOR
        SELECT emp_name FROM employees;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
```

OPEN emp_cursor;

read_loop: LOOP

FETCH emp_cursor INTO v_name;

IF done = 1 THEN

```

    LEAVE read_loop;

END IF;

SELECT v_name AS employee_name;

END LOOP;

CLOSE emp_cursor;

END $$

DELIMITER ;

```

Query 2: Call the procedure

```
CALL show_employee_names();
```

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. In the 'Procedures' section, the 'show_employee_names()' procedure is selected. The 'Execution results of routine 'show_employee_names'' tab is open, displaying the output of the query:

```

CALL `show_employee_names`();
+-----+
| employee_name |
+-----+
| Anita        |
| Rahul        |
| Meera        |
| Suresh       |
+-----+

```

Below this, the 'Routines' section lists the stored procedures available in the database:

Name	Type	Returns
GetCourseDetails	PROCEDURE	Edit Execute Export Drop
GetEmployeesByDepartment	PROCEDURE	Edit Execute Export Drop
check_department	PROCEDURE	Edit Execute Export Drop
show_employee_names	PROCEDURE	Edit Execute Export Drop

18. SQL Cursors

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

Answer:

Query 1: Stored Procedure

```
DELIMITER $$
```

```
CREATE PROCEDURE show_employee_details()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE v_id INT;
    DECLARE v_name VARCHAR(50);
    DECLARE v_dept INT;
    DECLARE v_salary INT;

    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 v_id, v_name, v_dept, v_salary;
        IF done = 1 THEN
            LEAVE read_loop;
        END IF;

        SELECT CONCAT(
            'ID: ', v_id,
```

```

      ', Name: ', v_name,
      ', Dept: ', v_dept,
      ', Salary: ', v_salary
    ) AS Employee_Details;
  END LOOP;

CLOSE emp_cursor;
END$$

```

DELIMITER ;

Query 2: Run the procedure

```
CALL show_employee_details();
```

Name	Type	Returns
GetCourseDetails	PROCEDURE	
GetEmployeesByDepartment	PROCEDURE	
check_department	PROCEDURE	
Console_v_employee_details	PROCEDURE	

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

Answer:

Query 1: Stored Procedure

DELIMITER \$\$

CREATE PROCEDURE show_courses()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE v_id INT;

DECLARE v_name VARCHAR(100);

DECLARE v_duration VARCHAR(50);

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;

course_loop: LOOP

FETCH course_cursor INTO v_id, v_name, v_duration;

IF done = 1 THEN

LEAVE course_loop;

END IF;

SELECT CONCAT(

'Course ID: ', v_id,

', Name: ', v_name,

', Duration: ', v_duration

```

) AS Course_Details;

END LOOP;

CLOSE course_cursor;

END$$

DELIMITER ;

```

Query 2: Run the procedure

```
CALL show_courses();
```

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

```
CALL `show_courses`();
```

The results are shown in a table format:

Execution results of routine 'show_courses'		
Course_Details		
Course ID: 101	Name: Computer Science Basics	Duration: 3 Months
Course ID: 102	Name: Database Management	Duration: 3.5 Months
Course ID: 104	Name: Cyber Security	Duration: 2 Months
Course ID: 105	Name: Data Analytics	Duration: 3 Months

Below this, there is a section titled 'Routines' which lists several stored procedures:

Name	Type	Returns
GetCourseDetails	PROCEDURE	
GetEmployeesByDepartment	PROCEDURE	
check_department	PROCEDURE	
Console_N_courses	PROCEDURE	

19. Rollback and Commit Savepoint

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

Answer:

Query: Create Savepoint → Insert Records → Rollback to Savepoint

-- Start the transaction

START TRANSACTION;

-- Insert 1st record

```
INSERT INTO employees (emp_id, emp_name, dept_id, salary)  
VALUES (110, 'TestUser1', 1, 30000);
```

-- Create savepoint

SAVEPOINT sp1;

-- Insert 2nd record

```
INSERT INTO employees (emp_id, emp_name, dept_id, salary)  
VALUES (111, 'TestUser2', 2, 40000);
```

-- Rollback to savepoint (means the 2nd insert will be removed)

ROLLBACK TO sp1;

-- Commit remaining changes

COMMIT;

The screenshot shows the phpMyAdmin interface. On the left, the database structure is visible, including the school_db schema which contains tables like employee_information_table, courses, departments, employees, and employee_log. The main area shows the employees table with the following data:

emp_id	emp_name	dept_id	salary	last_modified
101	Anita	1	55000	2025-11-19 19:51:02
102	Rahul	2	45000	2025-11-19 19:51:02
103	Meera	3	50000	2025-11-19 19:51:02
107	Suresh	2	48000	2025-11-19 19:51:02
110	TestUser1	1	30000	2025-11-21 14:28:43

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

Answer:

Query: Commit part of a transaction → Rollback remaining changes

-- Start transaction

```
START TRANSACTION;
```

-- Insert two new employees

```
INSERT INTO employees (emp_id, emp_name, dept_id, salary)
VALUES (120, 'PartialUser1', 1, 35000);
```

```
INSERT INTO employees (emp_id, emp_name, dept_id, salary)
VALUES (121, 'PartialUser2', 2, 36000);
```

-- Create savepoint

```
SAVEPOINT sp2;
```

-- Insert another record

```
INSERT INTO employees (emp_id, emp_name, dept_id, salary)
VALUES (122, 'RollbackUser', 3, 37000);
```

-- Commit changes BEFORE savepoint (120, 121)

```
RELEASE SAVEPOINT sp2;
```

```
COMMIT;
```

-- Now rollback remaining uncommitted changes

```
ROLLBACK;
```

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The left sidebar lists various databases and their structures. The 'Tables' section under 'school_db' contains a single entry for the 'employees' table. The main panel displays the contents of the 'employees' table:

	emp_id	emp_name	dept_id	salary	last_modified
<input type="checkbox"/>	101	Anita	1	55000	2025-11-19 19:51:02
<input type="checkbox"/>	102	Rahul	2	45000	2025-11-19 19:51:02
<input type="checkbox"/>	103	Meera	3	50000	2025-11-19 19:51:02
<input type="checkbox"/>	107	Suresh	2	48000	2025-11-19 19:51:02
<input type="checkbox"/>	110	TestUser1	1	30000	2025-11-21 14:28:43
<input type="checkbox"/>	120	PartialUser1	1	35000	2025-11-21 14:33:24
<input type="checkbox"/>	121	PartialUser2	2	36000	2025-11-21 14:33:24
<input type="checkbox"/>	122	RollbackUser	3	37000	2025-11-21 14:33:24