

## Module 4 – Introduction to DBMS

### 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.

#### Query -1

##### Database Query

```
CREATE DATABASE school_db;
```

#### Query – 2

##### students Table Query

```
CREATE TABLE students (
    student_id INT PRIMARY KEY AUTO_INCREMENT,
    student_name VARCHAR(50),
    age INT,
    class VARCHAR(20),
    address VARCHAR(100)
);
```

The screenshot shows the 'Table structure' tab in MySQL Workbench. The 'student\_id' column is defined as an int(11) type with AUTO\_INCREMENT, set as the primary key (Primary). The 'address' column is defined as a varchar(100) type. There is one index defined on the 'student\_id' column.

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

**Indexes:**

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
<span>Edit</span> <span>Rename</span> <span>Drop</span>	PRIMARY	BTREE	Yes	No	student_id	0	A	No	

**Partitions:**

No partitioning defined!

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

### Query -1

#### 5 Records Insert Query

```
INSERT INTO students (student_name, age, class, address) VALUES
('Rahul Patel', 12, '7th', 'Ahmedabad'),
('Priya Shah', 11, '6th', 'Surat'),
('Amit Mehta', 13, '8th', 'Rajkot'),
('Neha Joshi', 10, '5th', 'Vadodara'),
('Karan Desai', 12, '7th', 'Bhavnagar);
```

Server: 127.0.0.1 » Database: school\_db » Table: students

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

Showing rows 0 - 4 (5 total, Query took 0.0002 seconds.)

```
SELECT * FROM `students`
```

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	Rahul Patel	12	7th	Ahmedabad
<input type="checkbox"/>	2	Priya Shah	11	6th	Surat
<input type="checkbox"/>	3	Amit Mehta	13	8th	Rajkot
<input type="checkbox"/>	4	Neha Joshi	10	5th	Vadodara
<input type="checkbox"/>	5	Karan Desai	12	7th	Bhavnagar

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

Console

## Query – 2

### All Records SELECT Query

```
SELECT * FROM students;
```

The screenshot shows the MySQL Workbench interface with the following details:

- Server:** 127.0.0.1
- Database:** school\_db
- Table:** students

The main area displays the results of the query:

```
SELECT * FROM students;
```

The results show the following data:

student_id	student_name	age	class	address
1	Rahul Patel	12	7th	Ahmedabad
2	Priya Shah	11	6th	Surat
3	Amit Mehta	13	8th	Rajkot
4	Neha Joshi	10	5th	Vadodara
5	Karan Desai	12	7th	Bhavnagar

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

At the bottom, there are additional buttons for **Print**, **Copy to clipboard**, **Export**, **Display chart**, and **Create view**.

## 2. SQL Syntax

**Lab 1:** Write SQL queries to retrieve specific columns (student\_name and age) from the students table.

### Query – 1

#### student\_name and age

```
SELECT student_name, age FROM students;
```

The screenshot shows the phpMyAdmin interface for a database named 'students'. The 'Table: students' page is displayed. The table has columns: student\_name and age. The data is:

	student_name	age
<input type="checkbox"/>	Rahul	9
<input type="checkbox"/>	Amit	11
<input type="checkbox"/>	Neha	10
<input type="checkbox"/>	Priya	12
<input type="checkbox"/>	Karan	8

Below the table, there are buttons for 'Edit', 'Copy', 'Delete', 'Check all', and 'With selected:'. The status bar at the bottom shows 'localhost/phpmyadmin/index.php?route=/table/sql&db=students&table=students#'

## Lab2: Retrieve all students whose age is greater than 10;

```
SELECT *  
FROM students  
WHERE age > 10;
```

The screenshot shows the phpMyAdmin interface for a database named 'students'. The 'Table: students' page is displayed. The query 'SELECT \* FROM students WHERE age > 10;' has been run, resulting in 2 rows being shown.

	student_id	student_name	age	class	address
<input type="checkbox"/>	2	Amit	11	6th	Ahmedabad
<input type="checkbox"/>	4	Priya	12	7th	Rajkot

Below the table, there are buttons for 'Edit', 'Copy', 'Delete', 'Check all', and 'With selected:'. The status bar at the bottom shows 'localhost/phpmyadmin/index.php?route=/table/sql&db=students&table=students#'

### 3.SQLConstraints

**Lab 1:** Create a table teachers with the following columns: teacher\_id (Primary Key), teacher\_name (NOT NULL), subject (NOT NULL), and email (UNIQUE).

#### Query – 1

##### 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 the 'students' database. The 'teachers' table is selected. The 'Table structure' tab is active, displaying the following columns:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	teacher_id	int(11)			No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
2	teacher_name	varchar(50)	utf8mb4_general_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
3	subject	varchar(50)	utf8mb4_general_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
4	email	varchar(100)	utf8mb4_general_ci		Yes	NULL			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>

Below the table structure, there are buttons for 'Check all', 'With selected:', and various actions like 'Browse', 'Change', 'Drop', 'Primary', 'Unique', 'Index', 'Spatial', and 'Fulltext'. The 'Indexes' section shows two indexes:

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
<a href="#">Edit</a> <a href="#">Rename</a> <a href="#">Drop</a>	PRIMARY	BTREE	Yes	No	teacher_id	0	A	No	
<a href="#">Edit</a> <a href="#">Rename</a> <a href="#">Drop</a>	email	BTREE	Yes	No	email	0	A	Yes	

Below the indexes, there is a button 'Create an index on 1 columns Go'.

## Lab2: Implement a FOREIGN KEY constraint to relate teacher\_id with students table

Query 1: Add teacher\_id column to students table

ALTER TABLE students

ADD teacher\_id INT;

The screenshot shows the 'Table structure' tab for the 'students' table in phpMyAdmin. The table structure is as follows:

#	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

In the 'Indexes' section, there is one primary key named 'PRIMARY' defined on the 'student\_id' column.

Query 2: Add FOREIGN KEY Constraint

ALTER TABLE students

ADD CONSTRAINT fk\_teacher

FOREIGN KEY (teacher\_id)

REFERENCES teachers(teacher\_id);

The screenshot shows the phpMyAdmin interface for a MySQL database named 'students'. The current table is 'students'. The 'Structure' tab is selected, displaying the table's columns and their properties. The 'Indexes' tab is also visible, showing two indexes: one primary key on 'student\_id' and one foreign key on 'teacher\_id'.

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

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
<a href="#">Edit</a> <a href="#">Rename</a> <a href="#">Drop</a>	PRIMARY	BTREE	Yes	No	student_id	5	A	No	
<a href="#">Edit</a> <a href="#">Rename</a> <a href="#">Drop</a>	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.

### Query: Create a table courses

```
CREATE TABLE courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(50),
    course_credits INT
);
```

Table structure for courses

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	course_id	int(11)			No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
2	course_name	varchar(50)	utf8mb4_general_ci		Yes	NULL			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
3	course_credits	int(11)			Yes	NULL			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>

Add 1 column(s) after course\_credits

Indexes

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
<a href="#">Edit</a> <a href="#">Rename</a> <a href="#">Drop</a>	PRIMARY	BTREE	Yes	No	course_id	0	A	No	

Create an index on 1 columns

Partitions

No partitioning defined!

Partition table

Information

## Lab2: Create a database named university\_db

```
CREATE DATABASE university_db;
```

Show query box

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0008 seconds.)

```
CREATE DATABASE university_db;
```

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

## 5. ALTER Command

**Lab 1:** Modify the courses table by adding a column course\_duration using the ALTER command.

**Query: Add a New Column (course\_duration) to courses Table**

The screenshot shows the phpMyAdmin interface for the 'courses' table in the 'students' database. The 'Structure' tab is selected. A new column 'course\_duration' is being added, indicated by the 'Add' button and the 'after course\_name' dropdown. The table structure is 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(50)	utf8mb4_general_ci	Yes	NULL				Change  Drop  More
3	course_credits	int(11)		Yes	NULL				Change  Drop  More
4	course_duration	varchar(30)	utf8mb4_general_ci	Yes	NULL				Change  Drop  More

The 'Indexes' section shows a primary key named 'PRIMARY' on the 'course\_id' column. The 'Partitions' section indicates 'No partitioning defined!'. The 'Information' section shows the table is empty.

**Lab 2:** Drop the course\_credits column from the courses table.

**Query: Drop the Column (course\_credits) from courses Table**

```
ALTER TABLE courses
```

```
DROP course_credits;
```

The screenshot shows the phpMyAdmin interface for the 'students' database. The 'courses' table is selected. The table structure is as follows:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	course_id	int(11)			No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
2	course_name	varchar(50)	utf8mb4_general_ci		Yes	NULL			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
3	course_duration	varchar(30)	utf8mb4_general_ci		Yes	NULL			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>

Indexes:

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
<a href="#">Edit</a> <a href="#">Rename</a> <a href="#">Drop</a>	PRIMARY	BTREE	Yes	No	course_id	0	A	No	

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

Partitions: [No partitioning defined!](#)

Partition table

Information

Console Space usage Row statistics

## 6. DROP Command

**Lab 1: Drop the teachers table from the school\_db database.**

DROP TABLE teachers;

The screenshot shows the phpMyAdmin interface for the 'students' database. The SQL tab contains the following query:

```
DROP TABLE teachers;
```

The results pane shows:

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0058 seconds.)

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

Console

**Lab 2: Drop the students table from the school\_db database and verify that the table has been removed.**

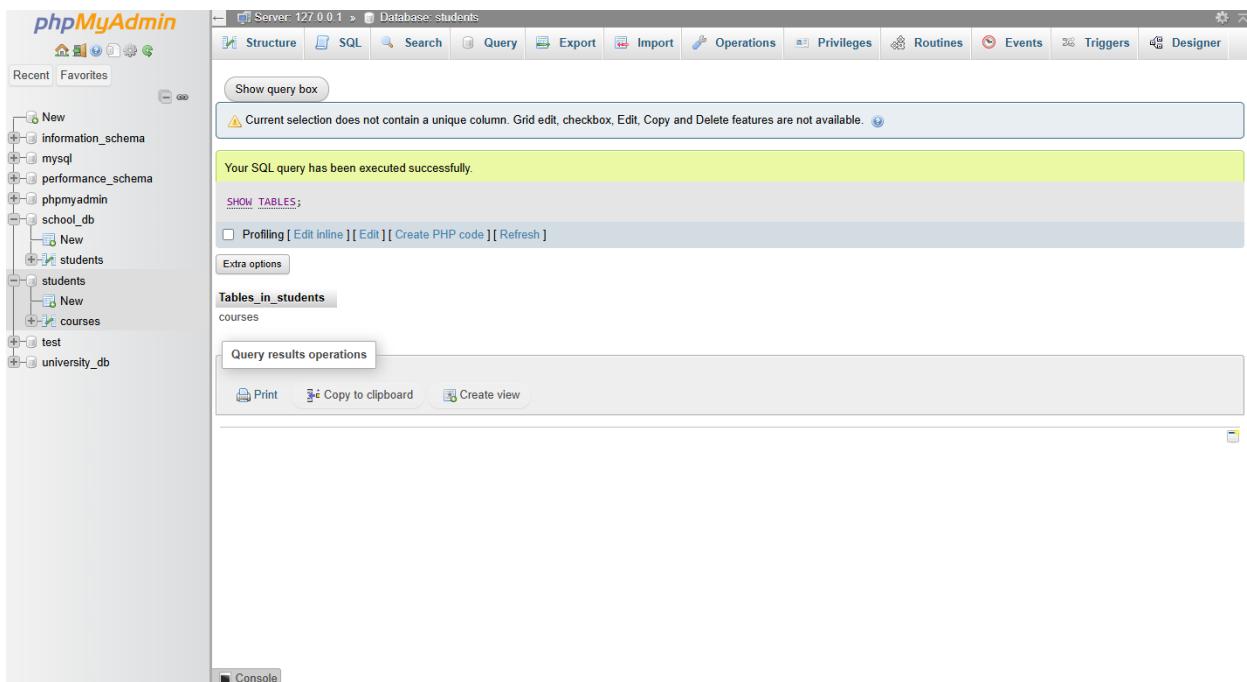
**Query 1: Drop the students table and verify removal**

DROP TABLE students;

The screenshot shows the phpMyAdmin interface. On the left, the database tree is visible with the 'school\_db' database selected. The 'SQL' tab is active. In the main area, a green message bar indicates: 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0078 seconds.)'. Below this, the SQL query 'DROP TABLE students;' is shown. At the bottom of the interface, there is a 'Console' tab.

**Query 2: Verify that the table is removed**

SHOW TABLES;



## 7. Data Manipulation Language (DML)

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

### Query: Insert 3 Records into courses Table

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

```
VALUES
```

```
(101, 'Computer Science', '3 Years'),
```

```
(102, 'B.Com', '3 Years'),
```

```
(103, 'BCA', '3 Years');
```

The screenshot shows the phpMyAdmin interface for a MySQL database named 'students'. The left sidebar lists databases: information\_schema, mysql, performance\_schema, phpmyadmin, school\_db, students, test, and university\_db. The 'courses' table is selected under the 'students' database. The main area displays the table structure with columns: course\_id, course\_name, and course\_duration. Data rows show course IDs 101, 102, and 103 with names Computer Science, B.Com, and BCA respectively, all having a duration of 3 Years. There are buttons for Edit, Copy, Delete, and Export.

course_id	course_name	course_duration
101	Computer Science	3 Years
102	B.Com	3 Years
103	BCA	3 Years

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

### Query: Update Course Duration of a Specific Course

UPDATE courses

SET course\_duration = '4 Years'

WHERE course\_id = 103;

The screenshot shows the phpMyAdmin interface for the 'courses' table in the 'students' database. The table has columns: course\_id, course\_name, and course\_duration. There are three records:

course_id	course_name	course_duration
101	Computer Science	3 Years
102	B.Com	3 Years
103	BCA	4 Years

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

#### Query: Delete a course using DELETE Command

DELETE FROM courses

WHERE course\_id = 102;

The screenshot shows the phpMyAdmin interface for the 'courses' table in the 'students' database after a delete operation. The table now has only one record:

course_id	course_name	course_duration
103	BCA	4 Years

## 8. Data Query Language (DQL)

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

### Query: Retrieve all courses

```
SELECT * FROM courses;
```

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
- Database:** students
- Table:** courses
- Query Result:**

```
Showing rows 0 - 1 (2 total. Query took 0.0002 seconds.)  
SELECT * FROM courses;
```
- Table Structure:** course\_id, course\_name, course\_duration
- Data Rows:**

course_id	course_name	course_duration
101	Computer Science	3 Years
103	BCA	4 Years
- Operations:** Print, Copy to clipboard, Export, Display chart, Create view

**Lab 2: Sort the courses based on course\_duration in descending order using ORDER BY.**

### Query: Sort Courses by course\_duration (Descending Order)

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

The screenshot shows the phpMyAdmin interface for a MySQL database named 'students'. On the left, a tree view lists databases like 'information\_schema', 'mysql', 'performance\_schema', 'phpmyadmin', 'school\_db', 'test', and 'university\_db'. Under 'school\_db', there are 'New' and 'students' schemas, with 'courses' being the selected table. The top navigation bar includes 'Browse', 'Structure', 'SQL', 'Search', 'Insert', 'Export', 'Import', 'Privileges', 'Operations', and 'Triggers'. A message box at the top says 'Showing rows 0 - 1 (2 total, Query took 0.0002 seconds) [course\_duration: 4 YEARS... - 3 YEARS...]' with a green checkmark. Below it is a SQL query: 'SELECT \* FROM courses ORDER BY course\_duration DESC;'. The main area displays the 'courses' table with two rows:

	course_id	course_name	course_duration
	103	BCA	4 Years
	101	Computer Science	3 Years

With the last row highlighted. Below the table are buttons for 'Edit', 'Copy', 'Delete', 'Check all', and 'With selected:'. At the bottom of the table area are buttons for 'Print', 'Copy to clipboard', 'Export', 'Display chart', and 'Create view'. A 'Console' button is at the bottom left.

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

**Query: Show only top 2 courses using LIMIT**

```
SELECT *
```

```
FROM courses
```

```
LIMIT 2;
```

The screenshot shows the phpMyAdmin interface. On the left, the database structure is visible with databases like information\_schema, mysql, performance\_schema, phpmyadmin, school\_db, students, test, and university\_db. The 'courses' table is selected under the 'students' database. The main area displays the results of the query: 'SELECT \* FROM courses LIMIT 2;'. The results show two rows of data:

course_id	course_name	course_duration
101	Computer Science	3 Years
103	BCA	4 Years

Below the table, there are options for 'Edit', 'Copy', and 'Delete' for each row. At the bottom, there are buttons for '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.**

### Query 1: Create two new users user1 and user2

```
CREATE USER 'user1'@'localhost' IDENTIFIED BY 'user1pass';
```

```
CREATE USER 'user2'@'localhost' IDENTIFIED BY 'user2pass';
```

The screenshot shows the phpMyAdmin interface with the following details:

- Left sidebar:** Shows databases: information\_schema, mysql, performance\_schema, phpmyadmin, school\_db, students, test, university\_db.
- Top menu:** Databases, SQL, Status, User accounts, Export, Import, Settings, Replication, Variables,Charsets, Engines, More.
- SQL tab content:**
  - MySQL returned an empty result set (i.e. zero rows). (Query took 0.0015 seconds.)
  - CREATE USER 'user1'@'localhost' IDENTIFIED BY 'user1pass';
  - [Edit inline] [Edit] [Create PHP code]
  - Error: #1046 No database selected
  - MySQL returned an empty result set (i.e. zero rows). (Query took 0.0012 seconds.)
  - CREATE USER 'user2'@'localhost' IDENTIFIED BY 'user2pass';
  - [Edit inline] [Edit] [Create PHP code]
  - Error: #1046 No database selected
- Bottom:** Console tab.

## Query 2: Grant SELECT permission on courses table to user1

GRANT SELECT

ON students.courses

TO 'user1'@'localhost';

The screenshot shows the phpMyAdmin interface with the following details:

- Left sidebar:** Shows databases: information\_schema, mysql, performance\_schema, phpmyadmin, school\_db, students (with New and courses), test, university\_db.
- Top menu:** Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, Triggers.
- SQL tab content:**
  - MySQL returned an empty result set (i.e. zero rows). (Query took 0.0017 seconds.)
  - GRANT SELECT ON students.courses TO 'user1'@'localhost';
  - [Edit inline] [Edit] [Create PHP code]
- Bottom:** Console tab.

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

**Query 1: Revoke INSERT permission from user1**

```
REVOKE INSERT students.courses FROM 'user1'@'localhost';
```

**Query 2: Grant INSERT permission to user2**

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

The screenshot shows the phpMyAdmin interface. The left sidebar lists databases: New, information\_schema, mysql, performance\_schema, phpmyadmin, school\_db, students, New, courses, test, and university\_db. The main area shows two queries in the SQL tab:

```
REVOKE INSERT ON students.courses FROM 'user1'@'localhost';
GRANT INSERT ON students.courses TO 'user2'@'localhost';
```

Both queries have green checkmarks indicating success. The status bar at the bottom shows "Console".

## 10. Transaction Control Language (TCL)

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

**Query: Insert a few rows + COMMIT**

```
START TRANSACTION;
```

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

```
VALUES
```

```
(104, 'MBA', '2 Years'),
```

```
(105, 'MCA', '2 Years');
```

COMMIT;

The screenshot shows the phpMyAdmin interface for the 'courses' table in the 'students' database. The table structure is displayed with columns: course\_id, course\_name, and course\_duration. There are four rows: 101 Computer Science (3 Years), 103 BCA (4 Years), 104 MBA (2 Years), and 105 MCA (2 Years). The bottom section of the interface shows the 'Query results operations' panel with options like Print, Copy to clipboard, Export, Display chart, and Create view.

- Rows **104** and **105** will appear permanently.

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

**Query: Insert additional rows + ROLLBACK**

START TRANSACTION;

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (106, 'BBA', '3 Years');

ROLLBACK;

The screenshot shows the phpMyAdmin interface for a MySQL database named 'students'. The left sidebar lists databases like 'information\_schema', 'mysql', 'performance\_schema', 'phpmyadmin', 'school\_db', 'students', 'test', and 'university\_db'. The main area displays the 'courses' table with the following data:

course_id	course_name	course_duration
101	Computer Science	3 Years
103	BCA	4 Years
104	MBA	2 Years
105	MCA	2 Years

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

- 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.**

#### Query: SAVEPOINT + ROLLBACK TO SAVEPOINT

START TRANSACTION;

```
-- Create savepoint
SAVEPOINT before_update;
```

```
-- Update a course
UPDATE courses SET course_duration='4 Years' WHERE course_id=104;
```

```
-- Rollback only this update
ROLLBACK TO before_update;
```

```
-- Commit remaining changes
COMMIT;
```

1 row affected.

```
UPDATE `courses` SET `course_duration` = '4 Years' WHERE `courses`.`course_id` = 104;
```

Showing rows 0 - 3 (4 total, Query took 0.0002 seconds)

	course_id	course_name	course_duration
<input type="checkbox"/>	101	Computer Science	3 Years
<input type="checkbox"/>	103	BCA	4 Years
<input type="checkbox"/>	104	MBA	4 Years
<input type="checkbox"/>	105	MCA	2 Years

With selected:  Edit  Copy  Delete

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

- Course\_id 104 duration will remain '4 years' (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.**

### Query 1: Create two tables

1. CREATE TABLE departments (
 dept\_id INT PRIMARY KEY,
 dept\_name VARCHAR(50)
 );

phpMyAdmin

Server: 127.0.0.1 > Database: school\_db > Table: departments

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

**Table structure**

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	dept_id	int(11)	utf8mb4_general_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
2	dept_name	varchar(50)	utf8mb4_general_ci		Yes	NULL			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>

Check all With selected: Browse Change Drop Primary Unique Index Spatial Fulltext

Print Propose table structure Move columns Normalize

Add 1 column(s) after dept\_name Go

**Indexes**

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
<a href="#">Edit</a> <a href="#">Rename</a> <a href="#">Drop</a>	PRIMARY	BTREE	Yes	No	dept_id	0	A	No	

Create an index on 1 columns Go

**Partitions**

No partitioning defined!

**Partition table**

**Information**

Space usage Row statistics

localhost/phpmyadmin/url.php?url=https%3A%2F%2Fdev.mysql.com%2F... Format dynamic

2. CREATE TABLE employees (

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

phpMyAdmin

Server: 127.0.0.1 > Database: school\_db > Table: employees

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

**Table structure**

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	emp_id	int(11)	utf8mb4_general_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
2	emp_name	varchar(50)	utf8mb4_general_ci		Yes	NULL			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
3	dept_id	int(11)	utf8mb4_general_ci		Yes	NULL			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>

Check all With selected: Browse Change Drop Primary Unique Index Spatial Fulltext

Print Propose table structure Move columns Normalize

Add 1 column(s) after dept\_id Go

**Indexes**

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
<a href="#">Edit</a> <a href="#">Rename</a> <a href="#">Drop</a>	PRIMARY	BTREE	Yes	No	emp_id	0	A	No	

Create an index on 1 columns Go

**Partitions**

No partitioning defined!

**Partition table**

**Information**

Console Space usage Row statistics

## Query 2: Insert sample records

### 1. INSERT INTO departments VALUES

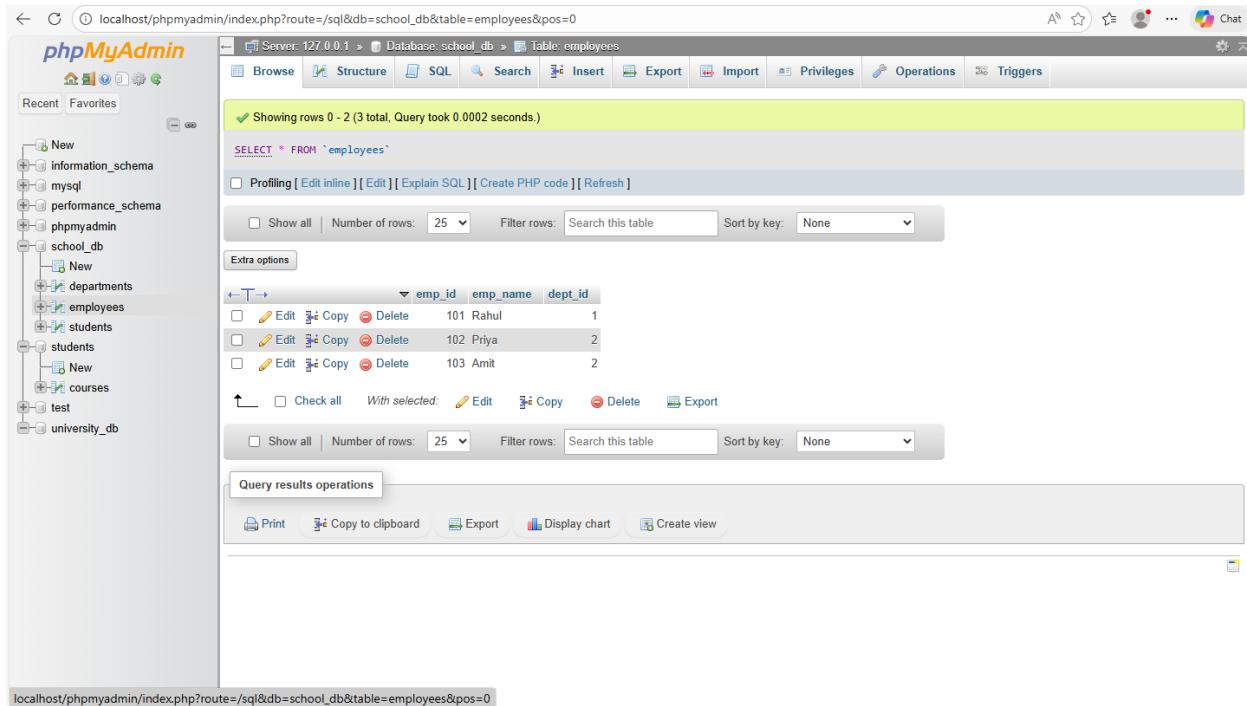
```
(1, 'HR'),  
(2, 'IT'),  
(3, 'Finance');
```

The screenshot shows the phpMyAdmin interface for the 'school\_db' database. The left sidebar lists databases like 'information\_schema', 'mysql', 'performance\_schema', 'phpmyadmin', 'school\_db', 'test', and 'university\_db'. The 'departments' table under 'school\_db' is selected. The table has columns 'dept\_id' and 'dept\_name'. Three rows have been inserted: (1, 'HR'), (2, 'IT'), and (3, 'Finance'). The row for 'IT' is currently selected. The bottom section shows a query results operations toolbar with options like Print, Copy to clipboard, Export, Display chart, and Create view.

dept_id	dept_name
1	HR
2	IT
3	Finance

### 2. INSERT INTO employees VALUES

```
(101, 'Rahul', 1),  
(102, 'Priya', 2),  
(103, 'Amit', 2);
```

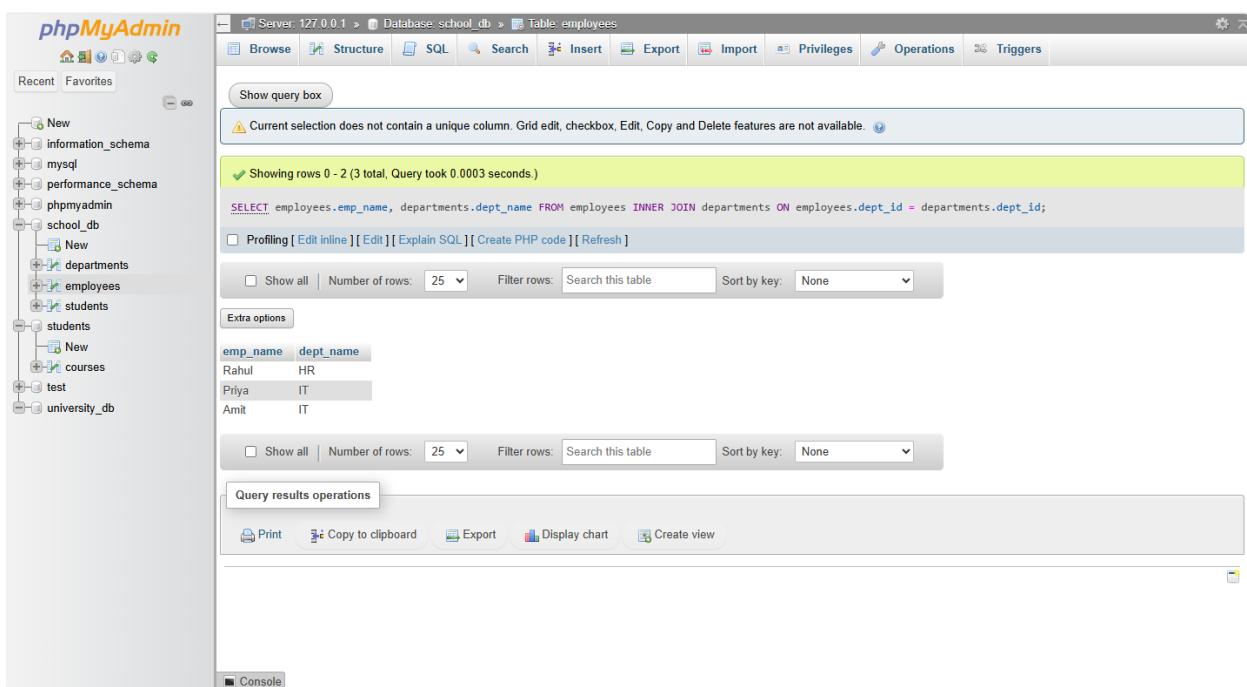


The screenshot shows the phpMyAdmin interface for the 'employees' table in the 'school\_db' database. The table has columns: emp\_id, emp\_name, and dept\_id. The data is:

emp_id	emp_name	dept_id
101	Rahul	1
102	Priya	2
103	Amit	2

### Query 3: Perform INNER JOIN

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



The screenshot shows the phpMyAdmin interface after executing the INNER JOIN query. The result set contains two rows:

emp_name	dept_name
Rahul	HR
Priya	IT
Amit	IT

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

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

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

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

dept_name	emp_name
HR	Rahul
IT	Priya
IT	Amit
Finance	NULL

## 12. SQL Group By

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

#### 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 school\_db schema selected. The employees table is currently being viewed. A query has been run:

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

The results show the following data:

dept_id	total_employees
1	1
2	2

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

### 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 = 40000 WHERE emp_id = 102;
UPDATE employees SET salary = 45000 WHERE emp_id = 103;
```

Showing rows 0 - 2 (3 total, Query took 0.0002 seconds.)

```
SELECT * FROM `employees`
```

	emp_id	emp_name	dept_id	salary
<input type="checkbox"/>	101	Rahul	1	30000
<input type="checkbox"/>	102	Priya	2	40000
<input type="checkbox"/>	103	Amit	2	45000

Query results operations

[Print](#) [Copy to clipboard](#) [Export](#) [Display chart](#) [Create view](#)

### Query: Find Average Salary per Department (Using AVG)

```
SELECT dept_id, AVG(salary) AS average_salary
```

```
FROM employees
```

```
GROUP BY dept_id;
```

Show query box

Showing rows 0 - 1 (2 total, Query took 0.0003 seconds.)

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

	dept_id	average_salary
<input type="checkbox"/>	1	30000.0000
<input type="checkbox"/>	2	42500.0000

Query results operations

[Print](#) [Copy to clipboard](#) [Export](#) [Display chart](#) [Create view](#)

## 13. SQL Stored Procedure

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

### Query: Stored Procedure to Retrieve Employees by Department

```
DELIMITER $$
```

```
CREATE PROCEDURE getEmployeesByDepartment(IN d_id INT)
```

```
BEGIN
```

```
    SELECT *
```

```
    FROM employees
```

```
    WHERE dept_id = d_id;
```

```
END $$
```

```
DELIMITER ;
```

#### • How to Execute the Procedure

```
CALL getEmployeesByDepartment(2);
```

The screenshot shows the phpMyAdmin interface with the following details:

- Left Panel:** Database tree showing databases like information\_schema, mysql, performance\_schema, phpmyadmin, school\_db, and tables such as departments, employees, students, courses, test, and university\_db.
- Top Bar:** Server: 127.0.0.1 > Database: school\_db > Table: employees
- Toolbar:** Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, Triggers.
- SQL Tab:** Shows the query: `CALL getEmployeesByDepartment(2);`. Below it, there are buttons for [Edit inline], [Edit], and [Create PHP code].
- Results Tab:** Displays the results of the query in a table:

emp_id	emp_name	dept_id	salary
102	Priya	2	40000
103	Amit	2	45000
- Bottom Bar:** Buttons for Print, Copy to clipboard, Create view, and a Console tab.

- 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.

### Query: Stored Procedure to Retrieve Course Details by course\_id

DELIMITER \$\$

```
CREATE PROCEDURE getCourseDetails(IN c_id INT)
BEGIN
    SELECT *
    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 databases like information\_schema, mysql, performance\_schema, phpmyadmin, school\_db, students, and university\_db. The current view is on the 'courses' table under the 'students' database. The main area shows the query:

```
CALL getCourseDetails(101);
```

Below the query, the result is displayed in a table:

course_id	course_name	course_duration
101	Computer Science	3 Years

- 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.**

### Query: Create a View to Show Employees with Department Names

Step 1: Create a VIEW using JOIN

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

Step 2: To see the view

```
SELECT * FROM emp_dept_view;
```

The screenshot shows the phpMyAdmin interface for the school\_db database. The left sidebar shows the database structure with tables like employees, departments, and emp\_dept\_view. The main area displays the results of the query `SELECT * FROM emp_dept_view;`. The results table has columns: emp\_id, emp\_name, salary, and dept\_name. Three rows are shown:

	emp_id	emp_name	salary	dept_name
<input type="checkbox"/>	101	Rahul	30000	HR
<input type="checkbox"/>	102	Priya	40000	IT
<input type="checkbox"/>	103	Amit	45000	IT

Below the table are standard MySQL edit, copy, and delete buttons. At the bottom of the results panel are buttons for Print, Copy to clipboard, Export, Display chart, and Create view.

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

```
CREATE OR REPLACE VIEW emp_dept_view AS
```

```
SELECT
```

```
    employees.emp_id,  
    employees.emp_name,  
    employees.salary,  
    departments.dept_name
```

```
FROM employees
```

```
INNER JOIN departments
```

```
ON employees.dept_id = departments.dept_id
```

```
WHERE employees.salary >= 50000;
```

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

Use OR REPLACE to update the view

```
CREATE OR REPLACE VIEW emp_dept_view AS
```

```
SELECT
```

```
    employees.emp_id,  
    employees.emp_name,  
    employees.salary,  
    departments.dept_name
```

```
FROM employees
```

```
INNER JOIN departments
```

```
ON employees.dept_id = departments.dept_id
```

```
WHERE employees.salary >= 50000;
```

#### **To check updated view**

```
SELECT * FROM `emp_dept_view`
```

The screenshot shows the phpMyAdmin interface for a MySQL database named 'school\_db'. The left sidebar shows the database structure with tables like 'departments', 'employees', 'students', and 'emp\_dept\_view'. The main area displays the 'emp\_dept\_view' table with one row selected:

	emp_id	emp_name	salary	dept_name
	103	Amit	50000	IT

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

## 15. SQL Triggers

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

### Query: Trigger to Log New Employee Insertions

Step 1: Create a log table

```
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 'employee\_log' table in the 'school\_db' database. The table structure is as follows:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	log_id	int(11)			No	None		AUTO_INCREMENT	<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
2	emp_id	int(11)			Yes	NULL			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
3	emp_name	varchar(50)	utf8mb4_general_ci		Yes	NULL			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
4	action_performed	varchar(20)	utf8mb4_general_ci		Yes	NULL			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
5	log_time	timestamp			No	current_timestamp()			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>

Indexes:

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
<a href="#">Edit</a> <a href="#">Rename</a> <a href="#">Drop</a>	PRIMARY	BTREE	Yes	No	log_id	0	A	No	

Create an index on  columns [Go](#)

Partitions:

No partitioning defined!

- Step 2: Create the trigger

```
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);
```

The screenshot shows the phpMyAdmin interface for the 'school\_db' database. The left sidebar lists databases like information\_schema, mysql, performance\_schema, phpmyadmin, school\_db, students, test, and university\_db. The 'Tables' section under school\_db contains 'New', 'departments', 'employees', 'employee\_log', and 'students'. The 'employees' table is selected, displaying 4 rows of data:

	emp_id	emp_name	dept_id	salary
<input type="checkbox"/>	101	Rahul	1	30000
<input type="checkbox"/>	102	Priya	2	40000
<input type="checkbox"/>	103	Amit	2	50000
<input type="checkbox"/>	107	Suresh	2	48000

Below the table, there are buttons for 'Check all', 'With selected:', 'Edit', 'Copy', 'Delete', and 'Export'. The bottom of the interface includes 'Query results operations' with options like Print, Copy to clipboard, Export, Display chart, and Create view.

- Running:

```
SELECT * FROM employee_log;
```

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

#### Query: Trigger to Automatically Update last\_modified Timestamp

- Step 1: Add a column to employees table

```
ALTER TABLE employees
ADD last_modified TIMESTAMP;
```

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, students, test, university\_db) and tables (departments, employees, employee\_log, students) under the school\_db database. The main area displays the 'employees' table with the following data:

	emp_id	emp_name	dept_id	salary	last_modified
<input type="checkbox"/>	101	Rahul	1	30000	2025-12-29 18:28:46
<input type="checkbox"/>	102	Priya	2	40000	2025-12-29 18:28:46
<input type="checkbox"/>	103	Amit	2	50000	2025-12-29 18:28:46
<input type="checkbox"/>	107	Suresh	2	48000	2025-12-29 18:28:46

Below the table, there are buttons for Edit, Copy, Delete, and other operations. The bottom of the interface includes a 'Query results operations' section with Print, Copy to clipboard, Export, Display chart, and Create view options.

- Step 2: Create the UPDATE trigger

DELIMITER \$\$

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

DELIMITER ;

The screenshot shows the phpMyAdmin interface for the 'employees' table in the 'school\_db' database. The table has columns: emp\_id, emp\_name, dept\_id, salary, and last\_modified. The data is as follows:

	emp_id	emp_name	dept_id	salary	last_modified
<input type="checkbox"/>	101	Rahul	1	30000	2025-12-29 18:28:46
<input type="checkbox"/>	102	Priya	2	40000	2025-12-29 18:28:46
<input type="checkbox"/>	103	Amit	2	50000	2025-12-29 18:28:46
<input type="checkbox"/>	107	Suresh	2	48000	2025-12-29 18:28:46

- Example Update

UPDATE employees

SET salary = 60000

WHERE emp\_id = 101;

The screenshot shows the phpMyAdmin interface for the 'employees' table in the 'school\_db' database after an update. The salary for employee ID 101 has been updated to 60000. The data is as follows:

	emp_id	emp_name	dept_id	salary	last_modified
<input type="checkbox"/>	101	Rahul	1	60000	2025-12-29 18:33:08
<input type="checkbox"/>	102	Priya	2	40000	2025-12-29 18:28:46
<input type="checkbox"/>	103	Amit	2	50000	2025-12-29 18:28:46
<input type="checkbox"/>	107	Suresh	2	48000	2025-12-29 18:28:46

- Output (View the updated timestamp)

`SELECT emp_id, salary, last_modified FROM employees;`

emp_id	salary	last_modified
101	60000	2025-12-29 18:33:08
102	40000	2025-12-29 18:28:46
103	50000	2025-12-29 18:28:46
107	48000	2025-12-29 18:28:46

- 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.

**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'. The left sidebar lists databases like 'information\_schema', 'mysql', 'performance\_schema', 'phpmyadmin', 'school\_db', 'students', 'test', and 'university\_db'. Under 'school\_db', there are 'Procedures', 'Tables' (including 'departments', 'employees', 'employee\_log', and 'students'), and 'Views'. The current table selected is 'employees'. The top navigation bar includes 'Browse', 'Structure', 'SQL', 'Search', 'Insert', 'Export', 'Import', 'Privileges', 'Operations', and 'Triggers'. A message in the main area says 'Your SQL query has been executed successfully.' Below it is the query: 'SELECT COUNT(\*) AS total\_employees FROM employees;'. The result is 'total\_employees' with a value of '4'. At the bottom, there are 'Query results operations' buttons: 'Print', 'Copy to clipboard', 'Export', 'Display chart', and 'Create view'.

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

### Query: Calculate Total Salary

SELECT SUM(salary) AS total\_salary FROM employees;

The screenshot shows the phpMyAdmin interface for a database named 'school\_db'. The left sidebar lists databases like 'information\_schema', 'mysql', 'performance\_schema', 'phpmyadmin', 'school\_db', 'students', 'test', and 'university\_db'. Under 'school\_db', there are 'Procedures', 'Tables' (including 'departments', 'employees', 'employee\_log', and 'students'), and 'Views'. The current table selected is 'employees'. The top navigation bar includes 'Browse', 'Structure', 'SQL', 'Search', 'Insert', 'Export', 'Import', 'Privileges', 'Operations', and 'Triggers'. A message in the main area says 'Showing rows 0 - 0 (1 total, Query took 0.0003 seconds.)'. Below it is the query: 'SELECT SUM(salary) AS total\_salary FROM employees;'. The result is 'total\_salary' with a value of '198000'. At the bottom, there are 'Query results operations' buttons: 'Print', 'Copy to clipboard', 'Export', 'Display chart', and 'Create view'.

## 17. PL/SQL Control Structures

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

### 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 MySQL database named 'school\_db'. The left sidebar displays the database structure with 'Tables' like 'departments', 'employees', and 'student\_log' under the 'school\_db' schema. The main query window shows the execution of the stored procedure 'check\_department' with parameter value 102. The output message 'Employee belongs to Department 2' is displayed, indicating that the employee with ID 102 is in Department 2.

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

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

```
DELIMITER $$

CREATE PROCEDURE DisplayEmployeeNames()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE ename VARCHAR(50);

    DECLARE emp_cursor CURSOR FOR
        SELECT emp_name FROM employee;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN emp_cursor;

    read_loop: LOOP
        FETCH emp_cursor INTO ename;

        IF done = 1 THEN
            LEAVE read_loop;
        END IF;

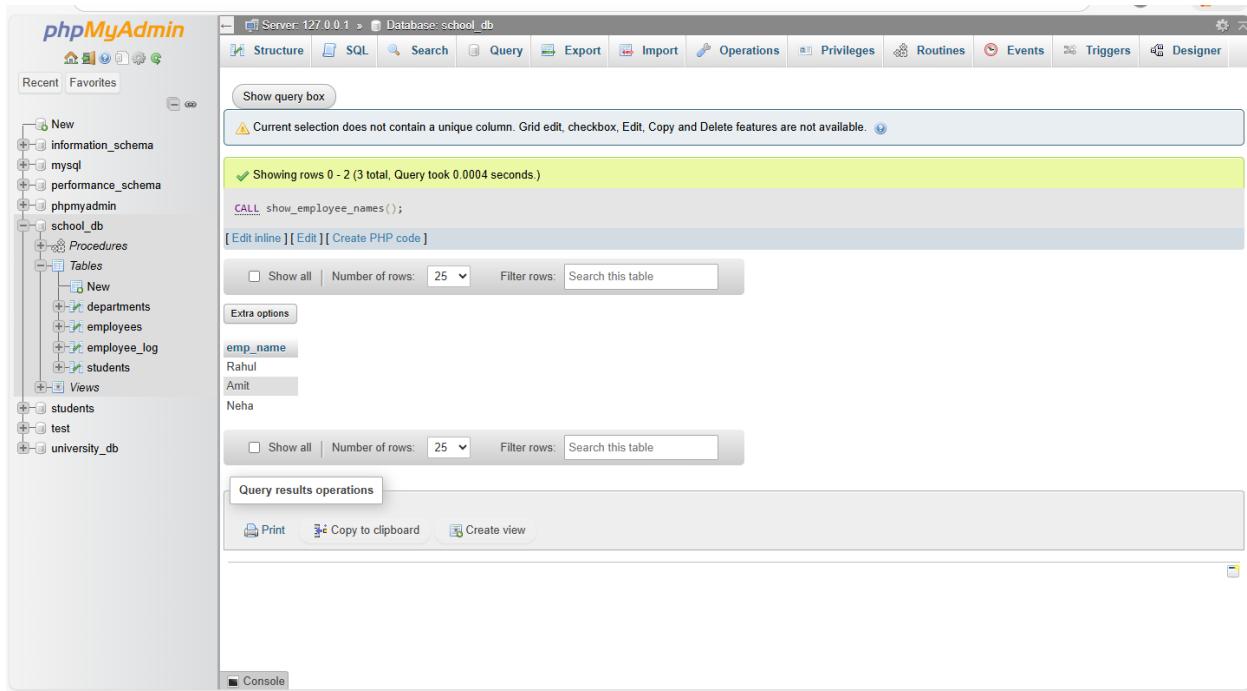
        SELECT ename AS Employee_Name;
    END LOOP;

    CLOSE emp_cursor;
END $$

DELIMITER ;
```

**Query 2: Call the procedure**

```
CALL show_employee_names();
```



## 18. SQL Cursors

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

### 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();
```

The screenshot shows the phpMyAdmin interface for the 'school\_db' database. The left sidebar lists various databases and their structures. The main area displays the results of the query 'CALL show\_employee\_details();', which returns three rows of data:

emp_id	emp_name	Department	salary
1	Rahul	HR	50000
2	Amit	IT	40000
3	Neha	Finance	45000

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

### 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 the 'school\_db' database. The left sidebar displays the database schema with tables like 'courses', 'departments', 'employees', 'employee\_log', and 'students'. The main area shows the execution results of the stored procedure 'show\_courses'. The results are presented in four rows, each labeled 'Course\_Details' with the following data:

Course ID	Name	Duration
101	Computer Science	3 Years
103	BCA	4 Years
104	MBA	4 Years
105	MCA	2 Years

Below the results, there is a section titled 'Routines' listing several stored procedures:

Name	Type	Returns	Actions
CheckDepartment	PROCEDURE		Edit Execute Export Drop
DisplayEmployeeNames	PROCEDURE		Edit Execute Export Drop
check_department	PROCEDURE		Edit Execute Export Drop
getCourseDetails	PROCEDURE		Edit Execute Export Drop
Console	PROCEDURE		Edit Execute Export Drop

## 19. Rollback and Commit Savepoint

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

### 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 (4, 'TestUser1', 1, 30000);
```

-- Create savepoint

```
SAVEPOINT sp1;
```

-- Insert 2nd record

```
INSERT INTO employees (emp_id, emp_name, dept_id, salary)
VALUES (5, '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 for the 'school\_db' database. The left sidebar shows the database structure with 'Tables' expanded, displaying 'employees' and other tables like 'courses', 'departments', 'students', etc. The main area shows the 'employees' table with the following data:

emp_id	emp_name	dep_id	salary
1	Rahul	HR	50000
2	Amit	IT	40000
3	Neha	Finance	45000
4	TestUser1	1	30000

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

**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 (5, 'PartialUser1', 1, 35000);

INSERT INTO employees (emp\_id, emp\_name, dept\_id, salary)

VALUES (6, 'PartialUser2', 2, 36000);

```
-- Create savepoint
```

```
SAVEPOINT sp2;
```

```
-- Insert another record
```

```
INSERT INTO employees (emp_id, emp_name, dept_id, salary)  
VALUES (7, 'RollbackUser', 3, 37000);
```

```
-- Commit changes BEFORE savepoint (5,6)
```

```
RELEASE SAVEPOINT sp2;
```

```
COMMIT;
```

```
-- Now rollback remaining uncommitted changes
```

```
ROLLBACK;
```

The screenshot shows the phpMyAdmin interface for the school\_db database. The left sidebar shows the database structure with tables like information\_schema, mysql, performance\_schema, phpmyadmin, school\_db, and university\_db. The school\_db table is selected, showing its structure with columns emp\_id, emp\_name, dep\_id, and salary. A query result table displays 7 rows of data:

emp_id	emp_name	dep_id	salary
1	Rahul	HR	50000
2	Amit	IT	40000
3	Neha	Finance	45000
4	TestUser1	1	30000
5	PartialUser1	1	35000
6	PartialUser2	2	36000
7	RollbackUser	3	37000