

SQL Query Code (Lab Exercise)

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.

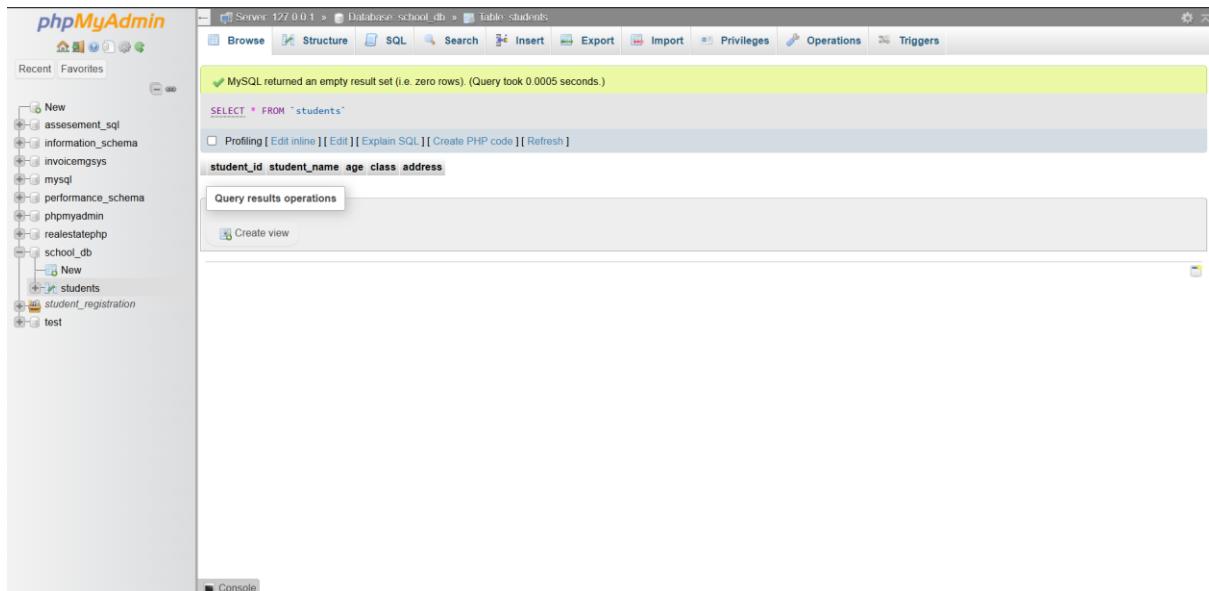
CODE:

```
CREATE DATABASE school_db;
```

```
USE school_db;
```

```
CREATE TABLE students ( student_id INT PRIMARY KEY, student_name VARCHAR(100), age INT, class VARCHAR(20), address VARCHAR(255));
```

SCREENSHOT:



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

CODE:

```
INSERT INTO students (student_id, student_name, age, class, address) VALUES  
(1, 'Rohan Patel', 15, '10A', 'Rajkot'),  
(2, 'Meera Shah', 14, '9B', 'Ahmedabad'),  
(3, 'Kunal Mehta', 16, '11C', 'Surat'),
```

(4, 'Priya Desai', 15, '10B', 'Vadodara'),

(5, 'Aarav Joshi', 13, '8A', 'Jamnagar');

SELECT * FROM students;

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. The 'students' table is selected. The results of the query 'SELECT * FROM students;' are displayed in a grid. The columns are student_id, student_name, age, class, and address. The data shows five rows of student information.

	student_id	student_name	age	class	address
<input type="checkbox"/>	1	Rohan Patel	15	10A	Rajkot
<input type="checkbox"/>	2	Meera Shah	14	9B	Ahmedabad
<input type="checkbox"/>	3	Kunal Mehta	16	11C	Surat
<input type="checkbox"/>	4	Priya Desai	15	10B	Vadodara
<input type="checkbox"/>	5	Aarav Joshi	13	8A	Jamnagar

SQL Syntax

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

Code:

SELECT student_name, age FROM students;

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. The 'students' table is selected. The results of the query 'SELECT student_name, age FROM students;' are displayed in a grid. The columns are student_name and age. The data shows five rows of student information.

	student_name	age
<input type="checkbox"/>	Rohan Patel	15
<input type="checkbox"/>	Meera Shah	14
<input type="checkbox"/>	Kunal Mehta	16
<input type="checkbox"/>	Priya Desai	15
<input type="checkbox"/>	Aarav Joshi	13

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

CODE:

```
SELECT * FROM students
```

```
WHERE age > 10;
```

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for a MySQL database named 'school_db'. The 'students' table is selected. In the SQL tab, the query `SELECT * FROM students WHERE age > 10;` is run, resulting in 5 rows being displayed. The table structure includes columns: student_id, student_name, age, class, and address. The data is as follows:

	student_id	student_name	age	class	address
1	1	Rohan Patel	15	10A	Rajkot
2	2	Meera Shah	14	9B	Ahmedabad
3	3	Kunal Mehta	16	11C	Surat
4	4	Prya Desai	15	10B	Vadodara
5	5	Aarav Joshi	13	8A	Jamnagar

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).

CODE:

```
CREATE TABLE teachers (
    teacher_id INT PRIMARY KEY,
    teacher_name VARCHAR(100) NOT NULL,
    subject VARCHAR(100) NOT NULL,
    email VARCHAR(150) UNIQUE
);
```

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. The left sidebar lists various databases and tables, including 'teachers' under 'school_db'. The main panel displays the structure of the 'teachers' table, which has columns: teacher_id, teacher_name, subject, and email. A SQL query window at the top shows a SELECT statement for the 'teachers' table, resulting in an empty set. Below the query window is a 'Query results operations' section with a 'Create view' button.

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

CODE:

ALTER TABLE students

ADD teacher_id INT;

ALTER TABLE students

ADD CONSTRAINT fk_teacher

FOREIGN KEY (teacher_id)

REFERENCES teachers(teacher_id);

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The left sidebar lists tables, including 'students' under 'school_db'. The main panel displays the structure of the 'students' table, which includes columns: student_id, student_name, age, class, address, and teacher_id. A SQL query window at the top shows a SELECT statement for the 'students' table, returning 5 rows. The results table shows student data with teacher_id values set to NULL. Below the results is an 'Extra options' section with buttons for Edit, Copy, Delete, and other actions. Navigation buttons like 'First', 'Last', and 'Previous' are also present.

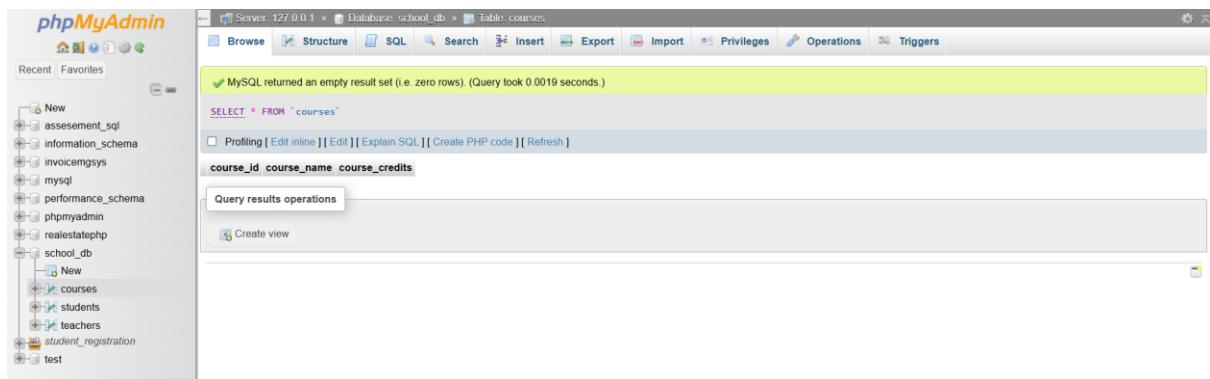
Main SQL Commands and Sub-commands (DDL)

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

CODE:

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

SCREENSHOT:

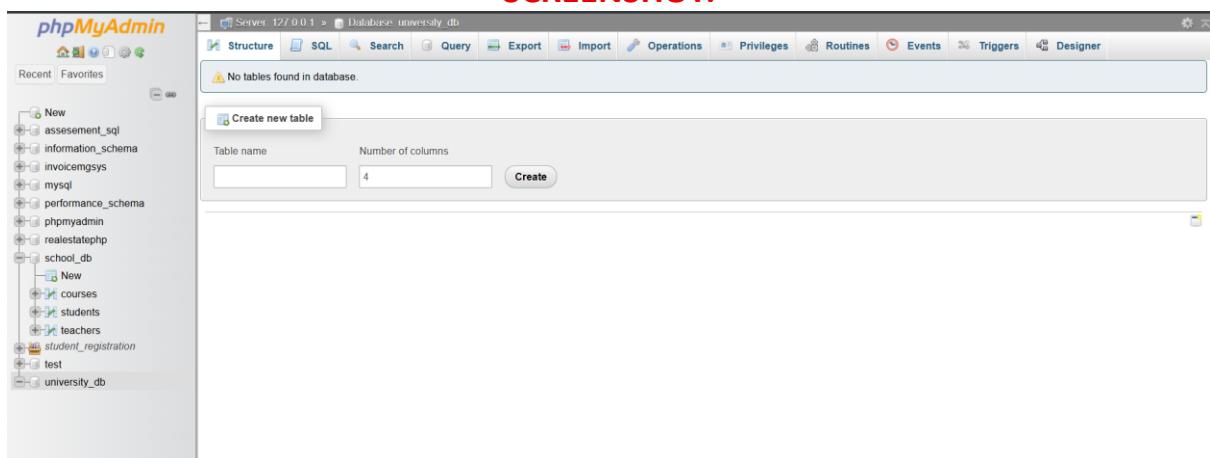


Lab 2: Use the CREATE command to create a database university_db.

CODE:

```
CREATE DATABASE university_db;
```

SCREENSHOT:



ALTER Command

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

CODE:

```
ALTER TABLE courses  
ADD course_duration VARCHAR(50);
```

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The left sidebar lists databases and tables, including 'courses'. The main window displays the 'courses' table structure with columns: course_id, course_name, course_credits, and course_duration. A message at the top indicates an empty result set from the query 'SELECT * FROM `courses`'.

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

CODE:

```
ALTER TABLE courses  
DROP COLUMN course_credits;
```

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The left sidebar lists databases and tables, including 'courses'. The main window displays the 'courses' table structure with columns: course_id, course_name, and course_duration. A message at the top indicates an empty result set from the query 'SELECT * FROM `courses`'.

DROP Command

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

CODE:

```
ALTER TABLE students  
DROP FOREIGN KEY fk_teacher;
```

```
USE school_db;  
DROP TABLE teachers
```

SCREENSHOT:

Table Action Rows Type Collation Size Overhead

courses Browse Structure Search Insert Empty Drop 0 InnoDB utf8mb4_general_ci 16.0 KiB -

students Browse Structure Search Insert Empty Drop 5 InnoDB utf8mb4_general_ci 32.0 KiB -

2 tables Sum 5 InnoDB utf8mb4_general_ci 48.0 KiB 0 B

Create new table

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

CODE:

DROP TABLE students;

SCREENSHOT:

Table Action Rows Type Collation Size Overhead

courses Browse Structure Search Insert Empty Drop 0 InnoDB utf8mb4_general_ci 16.0 KiB -

1 table Sum 0 InnoDB utf8mb4_general_ci 16.0 KiB 0 B

Create new table

Data Manipulation Language (DML)

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

CODE:

**INSERT INTO courses (course_id, course_name, course_duration)
VALUES (1, 'Mathematics', '6 months');**

**INSERT INTO courses (course_id, course_name, course_duration)
VALUES (2, 'Science', '6 months');**

**INSERT INTO courses (course_id, course_name, course_duration)
VALUES (3, 'English', '6 months');**

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for the 'courses' table. The table has columns: course_id, course_name, and course_duration. There are three rows: 1 Mathematics (6 months), 2 Science (6 months), and 3 English (6 months). The 'course_id' column is the primary key.

course_id	course_name	course_duration
1	Mathematics	6 months
2	Science	6 months
3	English	6 months

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

CODE:

UPDATE courses

SET course_duration = '12 months'

WHERE course_name = 'Mathematics';

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for the 'courses' table. The table has columns: course_id, course_name, and course_duration. There are three rows: 1 Mathematics (12 months), 2 Science (6 months), and 3 English (6 months). The 'course_id' column is the primary key.

course_id	course_name	course_duration
1	Mathematics	12 months
2	Science	6 months
3	English	6 months

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

CODE:

DELETE FROM courses

WHERE course_id = 2;

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for the 'courses' table. The table has three columns: course_id, course_name, and course_duration. There are two rows: one for Mathematics (course_id 1, duration 12 months) and one for English (course_id 3, duration 6 months).

course_id	course_name	course_duration
1	Mathematics	12 months
3	English	6 months

Data Query Language (DQL)

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

CODE:

SELECT * FROM courses;

SCREENSHOT:

The screenshot shows the phpMyAdmin interface after running the query 'SELECT * FROM courses'. The results are displayed in a table with the same structure as the original 'courses' table, showing two rows: Mathematics and English.

course_id	course_name	course_duration
1	Mathematics	12 months
3	English	6 months

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

CODE:

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

SCREENSHOT:

The screenshot shows the phpMyAdmin interface after running the query 'SELECT * FROM courses ORDER BY course_duration DESC'. The results are displayed in a table where the rows are sorted by course_duration in descending order, with Mathematics (12 months) appearing before English (6 months).

course_id	course_name	course_duration
1	Mathematics	12 months
3	English	6 months

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

CODE:

SELECT * FROM courses LIMIT 2;

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The 'courses' table is selected. In the SQL tab, the query `SELECT * FROM courses LIMIT 2;` is run, resulting in 2 rows being displayed. The table has columns: course_id, course_name, and course_duration. The data shows two rows: Mathematics (12 months) and English (6 months).

Data Control Language (DCL)

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

CODE:

CREATE USER 'user1'@'localhost' IDENTIFIED BY 'pass1';

CREATE USER 'user2'@'localhost' IDENTIFIED BY 'pass2';

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

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The 'Privileges' tab is selected. Three queries are run in the SQL tab:
1. `CREATE USER 'user1'@'localhost' IDENTIFIED BY 'pass1';`
2. `CREATE USER 'user2'@'localhost' IDENTIFIED BY 'pass2';`
3. `GRANT SELECT ON school_db.courses TO 'user1'@'localhost';`

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

CODE:

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

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

SCREENSHOT:

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

Transaction Control Language (TCL)

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

CODE:

START TRANSACTION;

```
INSERT INTO courses (course_id, course_name, course_duration)
VALUES (4, 'History', '5 months');
```

```
INSERT INTO courses (course_id, course_name, course_duration)
VALUES (5, 'Geography', '4 months');
```

COMMIT;

SCREENSHOT:

	course_id	course_name	course_duration
<input type="checkbox"/>	1	Mathematics	12 months
<input type="checkbox"/>	3	English	6 months
<input type="checkbox"/>	4	History	5 months
<input type="checkbox"/>	5	Geography	4 months

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

CODE:

START TRANSACTION;

```
INSERT INTO courses (course_id, course_name, course_duration)
VALUES (6, 'Physics', '6 months');
```

```
INSERT INTO courses (course_id, course_name, course_duration)
VALUES (7, 'Chemistry', '6 months');
```

ROLLBACK;

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The transaction log pane displays the following SQL queries:

```

START TRANSACTION;
INSERT INTO courses (course_id, course_name, course_duration) VALUES (6, 'Physics', '6 months');
INSERT INTO courses (course_id, course_name, course_duration) VALUES (7, 'Chemistry', '6 months');
ROLLBACK;

```

Each query is followed by a success message indicating the number of rows inserted or rolled back.

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

CODE:

START TRANSACTION;

UPDATE courses

```

SET course_duration = '10 months'
WHERE course_name = 'Mathematics';

```

SAVEPOINT before_science_update;

UPDATE courses

```

SET course_duration = '8 months'
WHERE course_name = 'Science';

```

ROLLBACK TO SAVEPOINT before_science_update;

COMMIT;

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The 'courses' table is displayed with the following data:

	course_id	course_name	course_duration
<input type="checkbox"/>	1	Mathematics	10 months
<input type="checkbox"/>	3	English	6 months
<input type="checkbox"/>	4	History	5 months
<input type="checkbox"/>	5	Geography	4 months

SQL Joins

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

CODE:

Step 1 — Create the tables

```
CREATE TABLE departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(100)
);

CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    employee_name VARCHAR(100),
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);
```

Step 2 — Insert sample data

```
INSERT INTO departments (department_id, department_name)
VALUES (1, 'HR'), (2, 'Finance'), (3, 'IT');
```

```
INSERT INTO employees (employee_id, employee_name, department_id)
VALUES (101, 'Alice', 1),
(102, 'Bob', 2),
(103, 'Charlie', 3),
(104, 'David', 2);
```

Step 3 — Perform INNER JOIN

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

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for a MySQL database named 'school_db'. On the left, there's a tree view of databases and tables. The 'Structure' tab is selected for the 'departments' table, which has three columns: 'department_id' (INT, Primary Key), 'department_name' (VARCHAR(100)), and 'overhead' (tinyint). The table contains 3 rows. Below the table structure, there's a list of other tables: 'courses', 'departments', and 'employees'. The 'employees' table also has three columns: 'employee_id' (INT, Primary Key), 'employee_name' (VARCHAR(100)), and 'department_id' (INT). It contains 4 rows. At the bottom, there are buttons for 'Check all' and 'With selected:'.

phpMyAdmin

Server: 127.0.0.1 > Database: school_db > table: employees

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

Showing rows 0 - 3 (4 total, Query took 0.0006 seconds)

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

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

	employee_id	employee_name	department_id
<input type="checkbox"/>	101	Alice	1
<input type="checkbox"/>	102	Bob	2
<input type="checkbox"/>	103	Charlie	3
<input type="checkbox"/>	104	David	2

phpMyAdmin

Server: 127.0.0.1 > Database: school_db > table: departments

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

Showing rows 0 - 2 (3 total, Query took 0.0006 seconds)

```
SELECT * FROM `departments`
```

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

	department_id	department_name
<input type="checkbox"/>	1	HR
<input type="checkbox"/>	2	Finance
<input type="checkbox"/>	3	IT

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

CODE:

```
SELECT departments.department_name, employees.employee_name
FROM departments
LEFT JOIN employees
ON departments.department_id = employees.department_id;
```

SCREENSHOT:

phpMyAdmin

Server: 127.0.0.1 > Database: school_db > table: departments

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

Show query box

Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.

Showing rows 0 - 3 (4 total, Query took 0.0008 seconds)

```
SELECT departments.department_name, employees.employee_name FROM departments LEFT JOIN employees ON departments.department_id = employees.department_id;
```

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

department_name	employee_name
HR	Alice
Finance	Bob
IT	Charlie
Finance	David

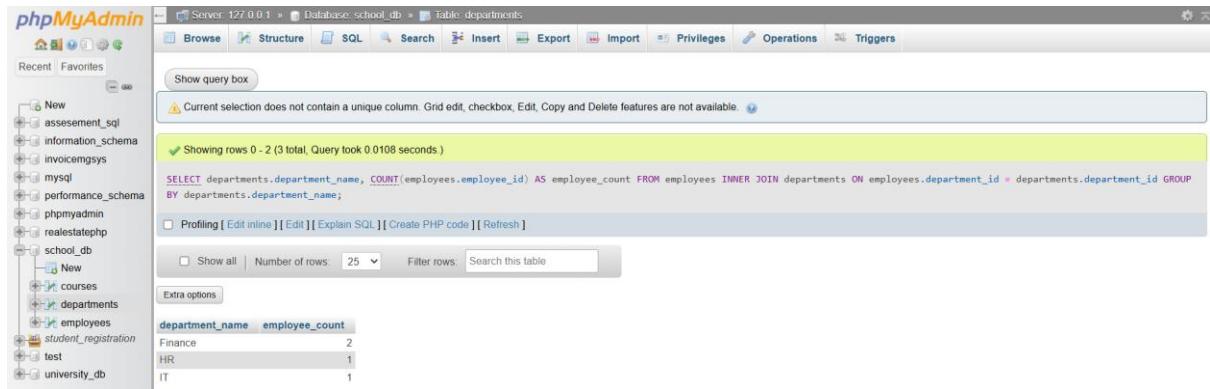
SQL Group By

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

CODE:

```
SELECT departments.department_name, COUNT(employees.employee_id) AS  
employee_count  
FROM employees  
INNER JOIN departments  
ON employees.department_id = departments.department_id  
GROUP BY departments.department_name;
```

SCREENSHOT:



department_name	employee_count
Finance	2
HR	1
IT	1

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

CODE:

```
UPDATE employees  
SET salary = 50000 WHERE employee_id = 101;  
UPDATE employees  
SET salary = 60000 WHERE employee_id = 102;  
UPDATE employees  
SET salary = 55000 WHERE employee_id = 103;  
UPDATE employees  
SET salary = 62000 WHERE employee_id = 104;
```

```
SELECT departments.department_name, AVG(employees.salary) AS average_salary  
FROM employees  
INNER JOIN departments  
ON employees.department_id = departments.department_id  
GROUP BY departments.department_name;
```

SCREENSHOT:

The image contains two separate screenshots of the phpMyAdmin interface, both showing the 'employees' table.

Screenshot 1 (Top): Shows the 'employees' table with 4 rows of data. The columns are employee_id, employee_name, department_id, and salary. The data is:

employee_id	employee_name	department_id	salary
101	Alice	1	50000.00
102	Bob	2	60000.00
103	Charlie	3	55000.00
104	David	2	62000.00

Screenshot 2 (Bottom): Shows the 'departments' table with 3 rows of data. The columns are department_name and average_salary. The data is:

department_name	average_salary
Finance	61000.000000
HR	50000.000000
IT	55000.000000

SQL Stored Procedure

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

CODE:

```

CREATE PROCEDURE GetEmployeesByDepartment(IN dept_id INT)
BEGIN
    SELECT employee_id, employee_name, salary
    FROM employees
    WHERE department_id = dept_id;
END $$
```

SCREENSHOT:

The screenshot shows the phpMyAdmin interface with the 'employees' table selected. A query has been run:

```
CALL GetEmployeesByDepartment(2);
```

The results show 2 rows of data:

employee_id	employee_name	salary
102	Bob	60000.00
104	David	62000.00

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

CODE:

```
CREATE PROCEDURE GetCourseDetails(IN c_id INT)
BEGIN
SELECT course_id, course_name, course_duration
FROM courses
WHERE course_id = c_id;

CALL GetCourseDetails(3);
```

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. In the 'Tables' section, the 'courses' table is selected. The SQL tab contains the executed query: 'CALL GetCourseDetails(3);'. The results pane shows one row from the 'courses' table: course_id 3, course_name 'English', and course_duration '6 months'.

SQL View

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

CODE:

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

SCREENSHOT:

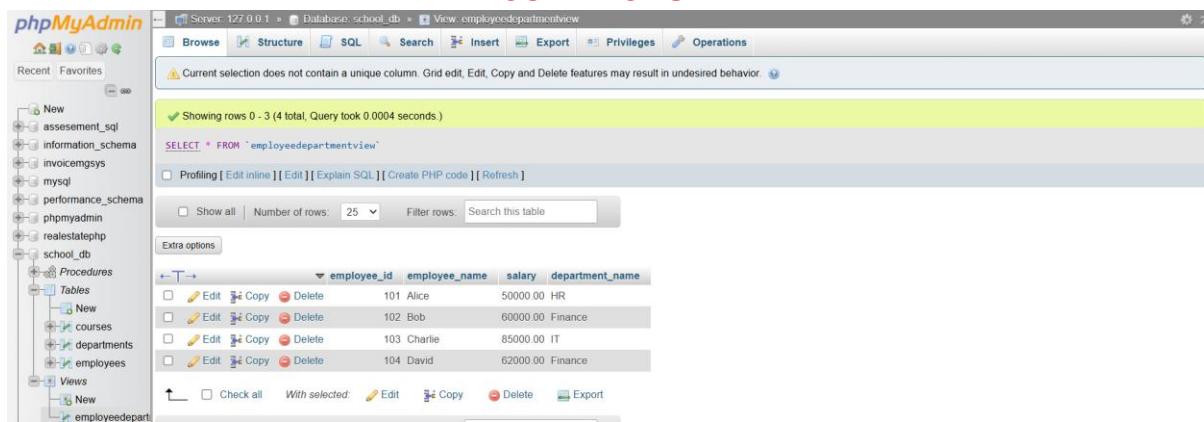
The screenshot shows the phpMyAdmin interface for a database named 'school_db'. In the 'Views' section, a new view named 'EmployeeDepartmentView' is created. The SQL tab shows the definition: 'SELECT * FROM `employeedepartmentview`'. The results pane displays four rows from the view: employee_id 101 (Alice), employee_id 102 (Bob), employee_id 103 (Charlie), and employee_id 104 (David), each associated with their respective department names (HR, Finance, IT, Finance).

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

CODE:

```
CREATE OR REPLACE VIEW EmployeeDepartmentView AS
SELECT e.employee_id, e.employee_name, e.salary, d.department_name
FROM employees e
INNER JOIN departments d
ON e.department_id = d.department_id
WHERE e.salary >= 50000;
```

SCREENSHOT:



The screenshot shows the phpMyAdmin interface for the 'school_db' database. The left sidebar shows various schemas and tables. The main area displays the 'employeedepartmentview' table with the following data:

	employee_id	employee_name	salary	department_name
1	101	Alice	50000.00	HR
2	102	Bob	60000.00	Finance
3	103	Charlie	85000.00	IT
4	104	David	62000.00	Finance

SQL Triggers

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

CODE:

```
CREATE TABLE employee_logs (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    employee_id INT,
    action_type VARCHAR(50),
    action_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The left sidebar lists databases and tables, including 'employee_logs'. The main area shows the structure of the 'employee_logs' table with columns: log_id, employee_id, action_type, and action_time. A query result window displays the message: "MySQL returned an empty result set (i.e. zero rows). (Query took 0.0007 seconds.)".

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

CODE:

```
CREATE TRIGGER update_employee_timestamp
BEFORE UPDATE ON employees
FOR EACH ROW
SET NEW.last_modified = CURRENT_TIMESTAMP;
```

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for the 'school_db' database. The left sidebar lists databases and tables, including 'employees'. The main area shows the structure of the 'employees' table with columns: employee_id, employee_name, department_id, salary, and last_modified. A query result window displays the message: "Showing rows 0 - 3 (4 total, Query took 0.0031 seconds.)". Below the table, a list of employees is shown with their details and edit/delete options.

	employee_id	employee_name	department_id	salary	last_modified
<input type="checkbox"/>	101	Alice	1	50000.00	2025-11-15 18:20:38
<input type="checkbox"/>	102	Bob	2	60000.00	2025-11-15 18:20:38
<input type="checkbox"/>	103	Charlie	3	85000.00	2025-11-15 18:20:38
<input type="checkbox"/>	104	David	2	62000.00	2025-11-15 18:20:38

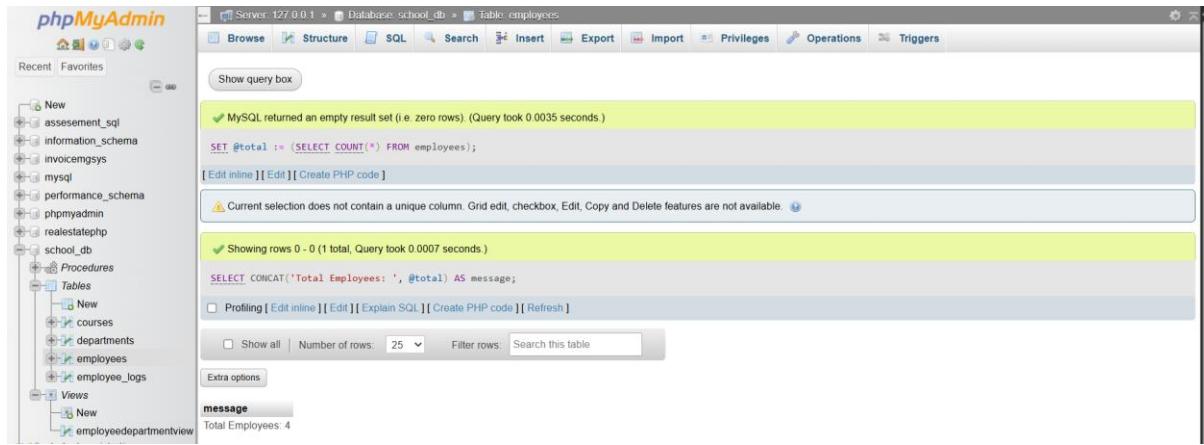
Introduction to PL/SQL

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

CODE:

```
SET @total := (SELECT COUNT(*) FROM employees);
SELECT CONCAT('Total Employees: ', @total) AS message;
```

SCREENSHOT:

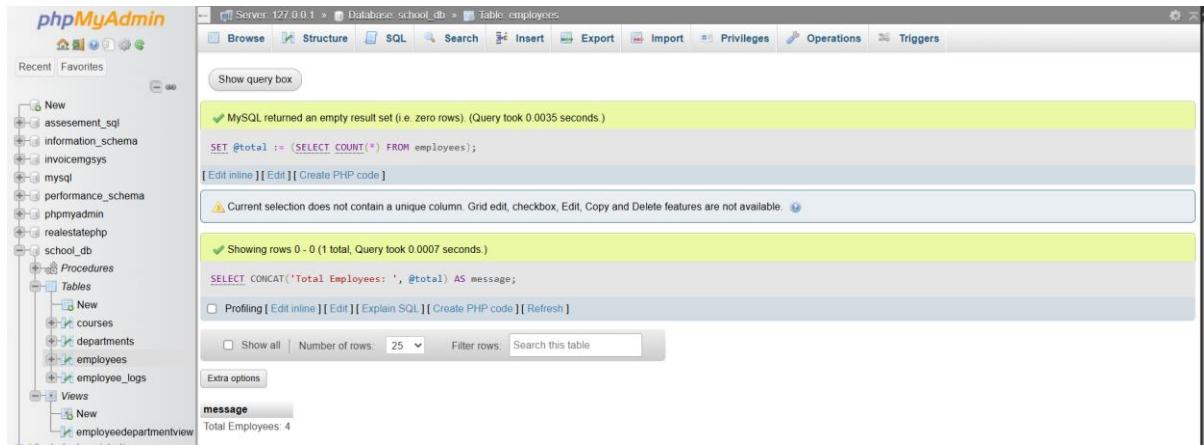


Lab 2: Create a PL/SQL block that calculates the total sales from an orders table.

CODE:

```
SET @sales := (SELECT SUM(total_amount) FROM orders);
SELECT CONCAT('Total Sales: ', @sales) AS message;
```

SCREENSHOT:



PL/SQL Control Structures

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

CODE:

DELIMITER \$\$

```
CREATE PROCEDURE CheckEmployeeDept(IN emp_id INT)
BEGIN
    DECLARE v_dept VARCHAR(100);
    SELECT d.department_name INTO v_dept
    FROM employees e
    JOIN departments d ON e.department_id = d.department_id
    WHERE e.employee_id = emp_id;

    IF v_dept = 'HR' THEN
        SELECT 'Employee works in Human Resources' AS message;
    ELSE
        SELECT 'Employee works in another department' AS message;
    END IF;
END $$
```

DELIMITER ;

CALL CheckEmployeeDept(101);

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for a MySQL database named 'school_db'. The left sidebar lists databases, schemas, and tables. The main area shows the creation of a stored procedure 'CheckEmployeeDept'. The SQL tab contains the following code:

```
CREATE PROCEDURE CheckEmployeeDept(IN emp_id INT) BEGIN DECLARE v_dept VARCHAR(100); SELECT d.department_name INTO v_dept FROM employees e JOIN departments d ON e.department_id = d.department_id WHERE e.employee_id = emp_id; IF v_dept = 'HR' THEN SELECT 'Employee works in Human Resources' AS message; ELSE SELECT 'Employee works in another department' AS message; END IF; END;
```

Below the code, the message 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0059 seconds.)' is displayed. A note indicates that the current selection does not contain a unique column, so grid edit, checkbox, Edit, Copy and Delete features are not available. The results section shows the output of the 'CALL' command:

```
-- Call it: CALL CheckEmployeeDept(101);
```

The results table shows one row with the message 'Employee works in Human Resources'.

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

CODE:

```
CALL display_books();
```

```
SELECT * FROM book_output;
```

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for a database named 'school_db'. The left sidebar lists various databases and their tables. The main area shows the results of a query: 'SELECT * FROM book_output;'. The results table displays three rows of data, each containing an ID, Title, Author, and Price. The columns are labeled 'ID', 'Title', 'Author', and 'Price'.

ID	Title	Author	Price
1	Book A	Author A	...
2	Book B	Author B	...
3	Book C	Author C	...

SQL Cursors

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

CODE:

```
DROP TABLE IF EXISTS member_output;
```

```
CREATE TABLE member_output (
    details VARCHAR(300)
);
```

```
DELIMITER $$
```

```
CREATE PROCEDURE display_members()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE m_id INT;
    DECLARE m_name VARCHAR(100);
    DECLARE m_email VARCHAR(100);

    DECLARE member_cursor CURSOR FOR
        SELECT member_id, member_name, email FROM members;
```

```

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN member_cursor;

read_loop: LOOP
    FETCH member_cursor INTO m_id, m_name, m_email;

    IF done THEN
        LEAVE read_loop;
    END IF;

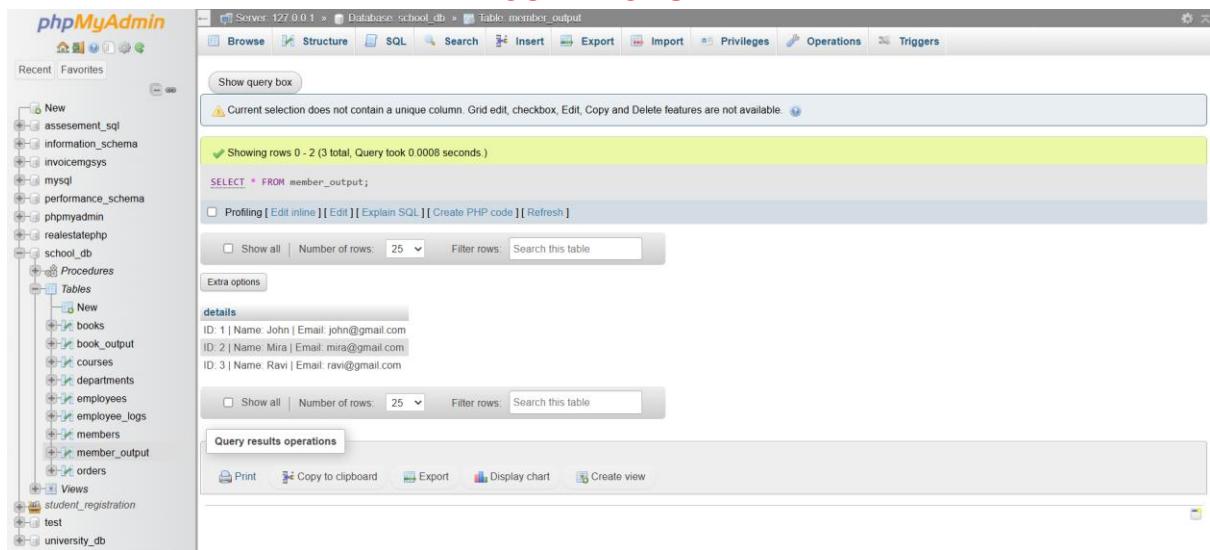
    INSERT INTO member_output VALUES (
        CONCAT(
            'ID: ', m_id,
            ' | Name: ', m_name,
            ' | Email: ', m_email
        )
    );
END LOOP;

CLOSE member_cursor;
END $$

DELIMITER ;

```

SCREENSHOT:



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

CODE:

DELIMITER \$\$

```
CREATE PROCEDURE get_books_by_author(IN p_author VARCHAR(100))
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE b_title VARCHAR(200);

    DECLARE book_cursor CURSOR FOR
        SELECT title FROM books WHERE author = p_author;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN book_cursor;

    read_loop: LOOP
        FETCH book_cursor INTO b_title;

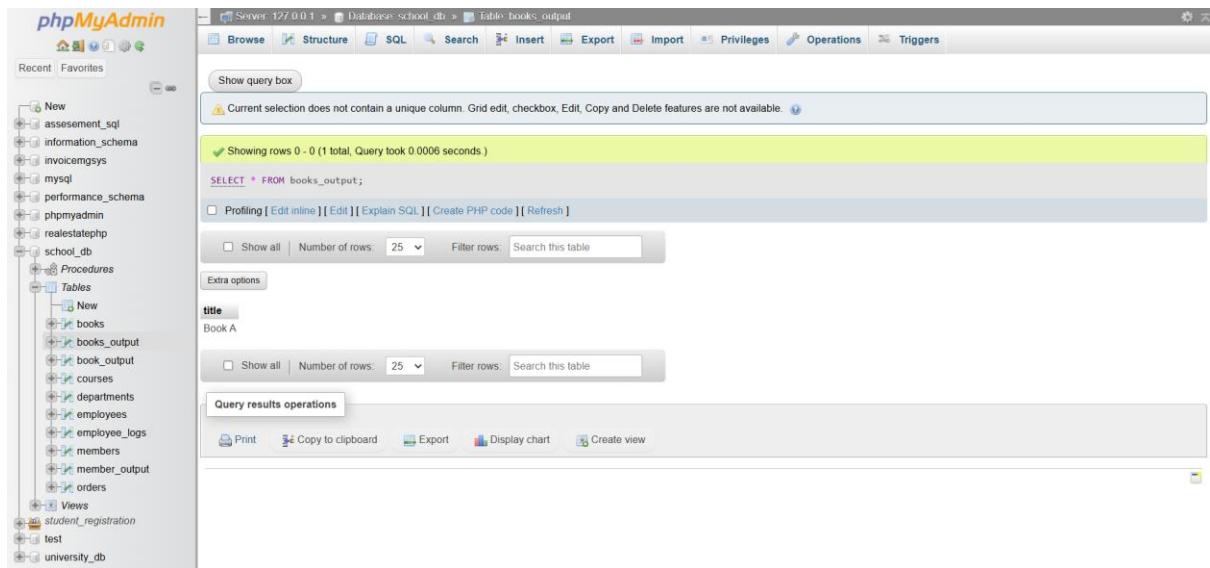
        IF done THEN
            LEAVE read_loop;
        END IF;

        INSERT INTO books_output VALUES (b_title);
    END LOOP;

    CLOSE book_cursor;
END $$
```

DELIMITER ;

SCREENSHOT:



Rollback and Commit Savepoint

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

CODE:

START TRANSACTION;

```
INSERT INTO members (member_id, member_name, email)
VALUES (10, 'Rahul', 'rahul@example.com');
```

```
SAVEPOINT sp1;
```

```
UPDATE members
```

```
SET member_name = 'Rahul Kumar'
WHERE member_id = 10;
```

```
UPDATE members
```

```
SET email = 'rahul.k@example.com'
WHERE member_id = 10;
```

```
ROLLBACK TO sp1;
```

```
COMMIT;
```

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for the 'members' table in the 'school_db' database. The table has three columns: member_id, member_name, and email. There are four rows of data:

member_id	member_name	email
1	John	john@gmail.com
2	Mira	mira@gmail.com
3	Ravi	ravi@gmail.com
10	Rahul	rahul@example.com

The row for 'Rahul' is highlighted with a gray background. The 'Edit' button for this row is also highlighted, indicating it is selected. The top navigation bar shows the database name as 'Database school_db' and the table name as 'table members'. The left sidebar shows the database structure with various tables like 'books', 'departments', 'employees', 'student_registration', etc.

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

CODE:

START TRANSACTION;

INSERT INTO books (book_id, title, author, price)

VALUES

(101, 'Book One', 'Author A', 250),

(102, 'Book Two', 'Author B', 300),

(103, 'Book Three', 'Author A', 400);

COMMIT;

START TRANSACTION;

SAVEPOINT sp1;

UPDATE books SET price = 500 WHERE book_id = 101;

UPDATE books SET title = 'Updated Book' WHERE book_id = 102;

ROLLBACK TO sp1;

COMMIT;

SCREENSHOT:

The screenshot shows the phpMyAdmin interface for a MySQL database named 'school_db'. The left sidebar shows various databases and their structures. The main area is focused on the 'books' table within the 'school_db' database. The table has columns: book_id, title, author, and price. There are six rows of data: (1, Book A, Author A, 250.00), (2, Book B, Author B, 300.00), (3, Book C, Author C, 450.00), (101, Book One, Author A, 250.00), (102, Book Two, Author B, 300.00), and (103, Book Three, Author A, 400.00). Each row has edit, copy, and delete options. Below the table, there are buttons for operations like check all, edit, copy, delete, and export. At the bottom, there are links for print, copy to clipboard, display chart, and create view.

book_id	title	author	price
1	Book A	Author A	250.00
2	Book B	Author B	300.00
3	Book C	Author C	450.00
101	Book One	Author A	250.00
102	Book Two	Author B	300.00
103	Book Three	Author A	400.00