Lab 3: Introduction to Databases and SQL

Overview

In this lab, you will explore the fundamentals of relational databases, focusing on MySQL, one of the most popular database management systems. Through hands-on exercises, you will learn to create and manage databases, build tables, define relationships, and query data using SQL (Structured Query Language). By the end of this lab, you will have a solid understanding of database structure and the basic SQL commands used in real-world applications.

Prerequisites

- Basic knowledge of relational databases and SQL concepts.
- XAMPP installed on your local machine (for Apache server, MySQL, PHP, and Perl).

Lab Objectives

- Understand the basic concepts of relational databases.
- Learn how to set up a MySQL database using XAMPP.
- Create and manage tables with various data types.
- Perform basic SQL queries (SELECT, INSERT, UPDATE, DELETE) and understand their practical applications.

Exercise Instructions

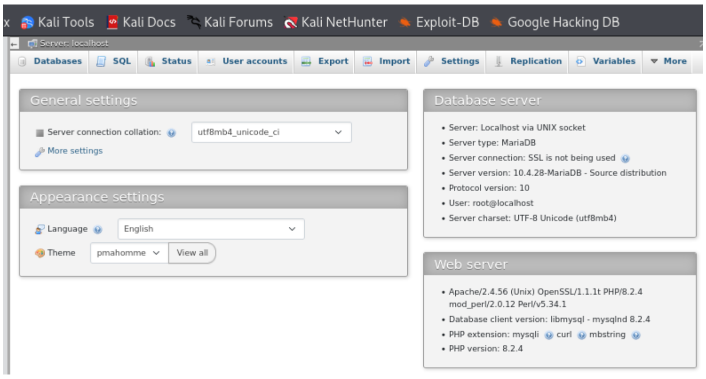
Exercise 1: Setting Up MySQL with XAMPP

1. Install and Run XAMPP:

- If you haven't already, download and install XAMPP from https://www.apachefriends.org/.
- Launch XAMPP and start the Apache and MySQL services from the XAMPP Control Panel.

2. Access phpMyAdmin:

- Open your web browser and navigate to http://localhost/phpmyadmin/.
- phpMyAdmin is a web-based tool provided by XAMPP for managing MySQL databases.



General settings

Server connection collation: utf8mb4_unicode_ci

More settings

Appearance settings

Language: English

Theme: pmahomme | View all

Database server

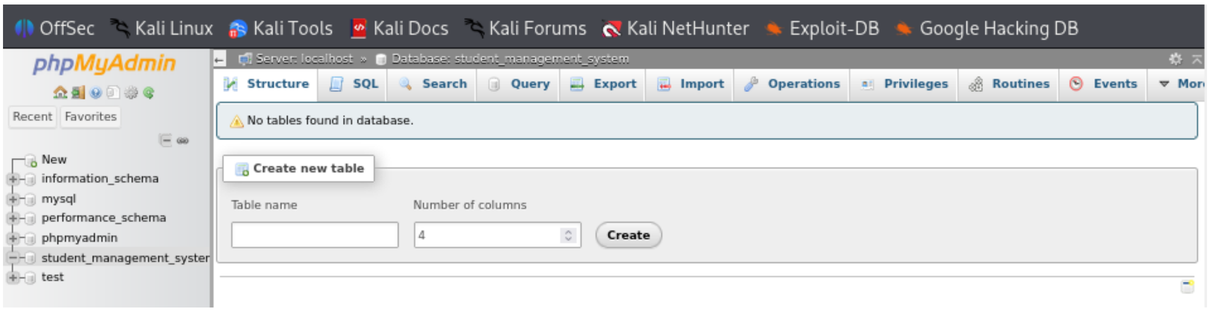
- Server: Localhost via UNIX socket
- Server type: MariaDB
- Server connection: SSL is not being used
- Server version: 10.4.28-MariaDB - Source distribution
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unicode (utf8mb4)

Web server

- Apache/2.4.56 (Unix) OpenSSL/1.1.1t PHP/8.2.4 mod_perl/2.0.12 Perl/v5.34.1
- Database client version: libmysql - mysqlnd 8.2.4
- PHP extension: mysqli curl mbstring
- PHP version: 8.2.4

3. Create a New Database:

- In phpMyAdmin, click on the Databases tab.
- Enter the name student_management_system for your new database and click Create.
- This database will store information about students, courses, and enrollments.



Server: localhost » Database: student_management_system

No tables found in database.

Table name

Number of columns 4

4. Reflection:

- Explain why a web application like a Student Management System would need a relational database. What are the benefits of using MySQL in such applications?

Reflection

A web application like a Student Management System (SMS) requires a relational database because it allows students to register for multiple courses, courses can have many enrolled students, and administrators need to manage enrolment records.

Benefits of using MySQL in such applications

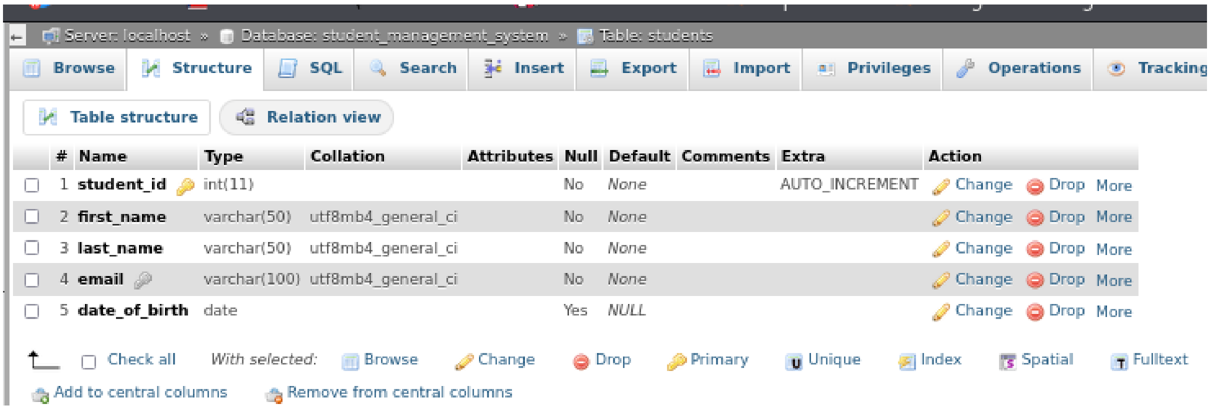
- it enforces data types, constraints, and relationships, which helps ensure the accuracy and consistency of student, course, and enrolment data.
- It can efficiently handle large datasets without performance issue
- It supports authentication and permissions, which allow different users (e.g., students, lecturers, administrators) to have appropriate access levels.
- Administrators can create, manage, and query databases without needing deep command-line knowledge, making it beginner-friendly.
- it allows dynamic web applications like SMS to store, update, and retrieve data in real-time.

Exercise 2: Creating Tables and Defining Columns

1. Create a students Table:

- After creating the database, click on the student_management_system database.
- Create a new table named students with the following columns:
  - student_id (INT, Primary Key, Auto Increment)
  - first_name (VARCHAR(50))

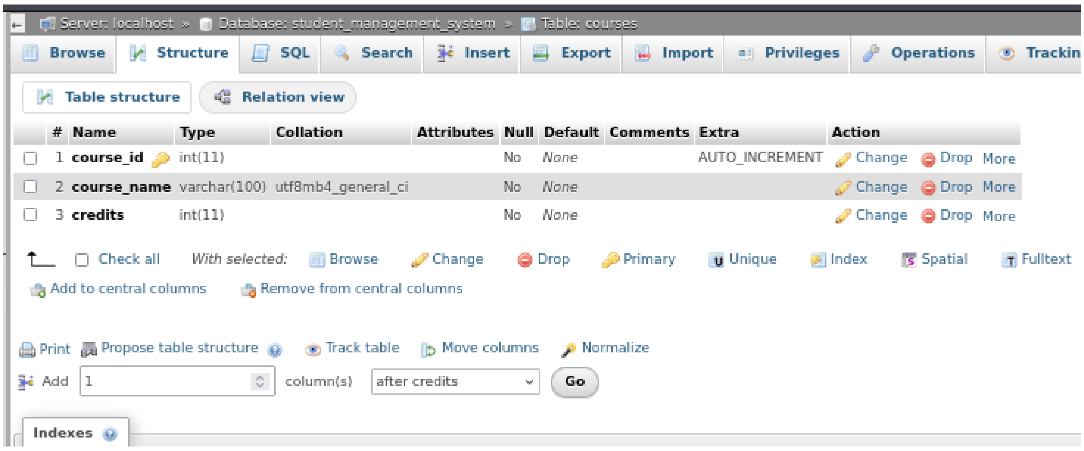
    - last_name (VARCHAR(50))
    - email (VARCHAR(100), Unique)
    - date_of_birth (DATE)
- Click Save to create the table.



	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
□	1	student_id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
□	2	first_name	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
□	3	last_name	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
□	4	email	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
□	5	date_of_birth	date			Yes	NULL			Change Drop More

2. Create a courses Table:

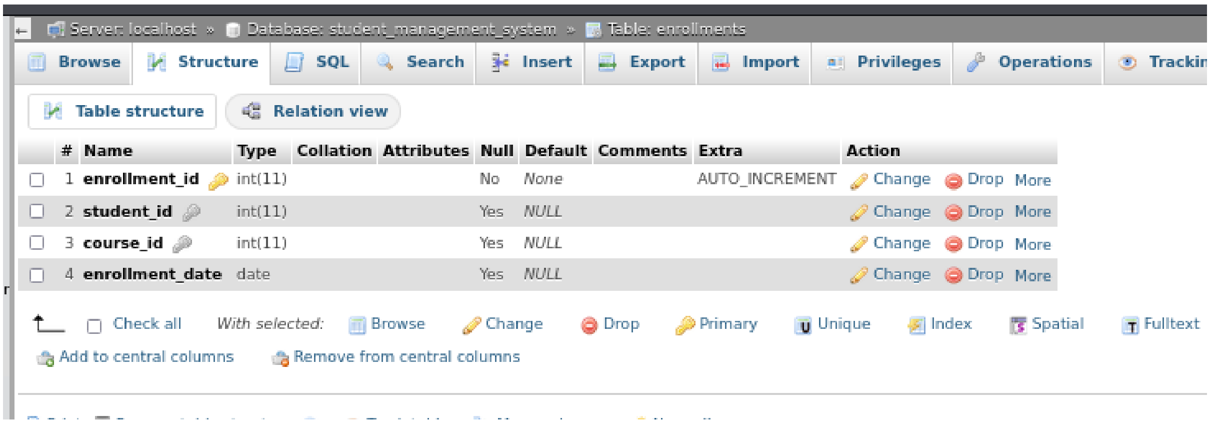
- Create a second table named courses with the following columns:
    - course_id (INT, Primary Key, Auto Increment)
    - course_name (VARCHAR(100))
    - redits (INT)
- Click Save.



	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
□	1	course_id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
□	2	course_name	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
□	3	credits	int(11)			No	None			Change Drop More

3. Create an enrollments Table (Relationship Between Students and Courses):

- Create a third table named enrollments to track which students are enrolled in which courses. This table should have the following columns:
    - enrollment_id (INT, Primary Key, Auto Increment)
    - student_id (INT, Foreign Key referencing students.student_id)
    - course_id (INT, Foreign Key referencing courses.course_id)
    - enrollment_date (DATE)
- Click Save to complete the table creation.



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**enrollment_id**	int(11)			No	*None*		AUTO_INCREMENT	Change Drop More
2	**student_id**	int(11)			Yes	*NULL*			Change Drop More
3	**course_id**	int(11)			Yes	*NULL*			Change Drop More
4	**enrollment_date**	date			Yes	*NULL*			Change Drop More

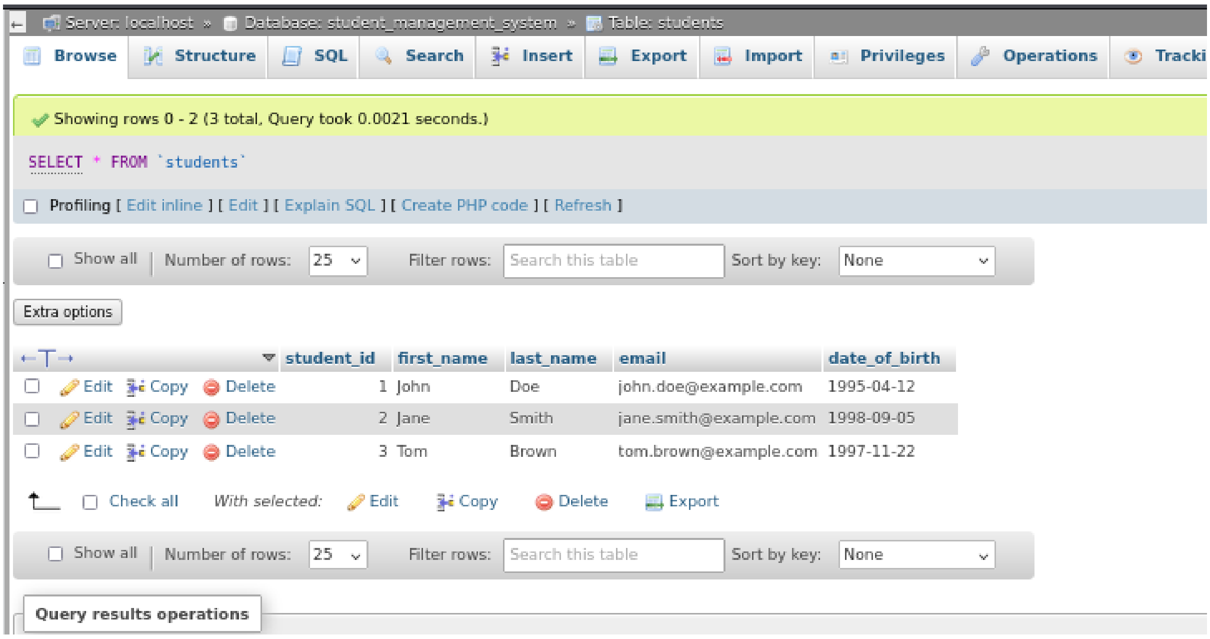
4. Reflection:

- Discuss how relationships are established between tables in a relational database. Why are foreign keys important in maintaining data integrity?
  - Relationships in a relational database are established by linking a **primary key** in one table to a **foreign key** in another. This creates logical connections (e.g., students linked to their enrollments, courses linked to their enrollments).
  - **Foreign keys are important** because they enforce **referential integrity**, ensuring that records in related tables are valid and consistent (e.g., you can't enroll a student or course that doesn't exist).

Exercise 3: Inserting Data into the Tables

1. Insert Sample Data into the students Table:

- Open the SQL tab in phpMyAdmin and execute the following SQL statement to insert some students:
- INSERT INTO students (first_name, last_name, email, date_of_birth)
- VALUES
- ('John', 'Doe', 'john.doe@example.com', '1995-04-12'),
- ('Jane', 'Smith', 'jane.smith@example.com', '1998-09-05'),
- ('Tom', 'Brown', 'tom.brown@example.com', '1997-11-22');


Server: localhost » Database: student_management_system » Table: students
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Tracki
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
Check all
With selected: Edit Copy Delete Export
Query results operations

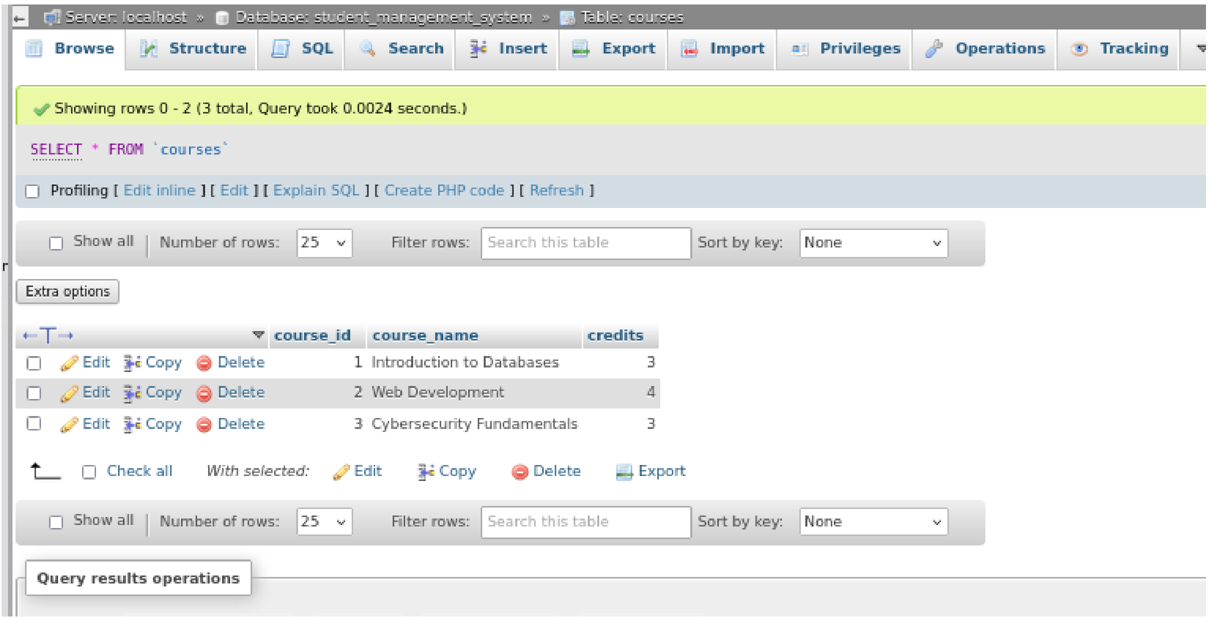
✓ Showing rows 0 - 2 (3 total, Query took 0.0021 seconds.)

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

	student_id	first_name	last_name	email	date_of_birth
□ Edit Copy Delete	1	John	Doe	john.doe@example.com	1995-04-12
□ Edit Copy Delete	2	Jane	Smith	jane.smith@example.com	1998-09-05
□ Edit Copy Delete	3	Tom	Brown	tom.brown@example.com	1997-11-22

2. Insert Sample Data into the courses Table:

- Insert some sample courses using the following SQL statement:
- INSERT INTO courses (course_name, credits)
- VALUES
  - ('Introduction to Databases', 3),
  - ('Web Development', 4),
  - ('Cybersecurity Fundamentals', 3);


Server: localhost » Database: student_management_system » Table: courses
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Tracking
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
Check all
With selected: Edit Copy Delete Export
Query results operations

✓ Showing rows 0 - 2 (3 total, Query took 0.0024 seconds.)

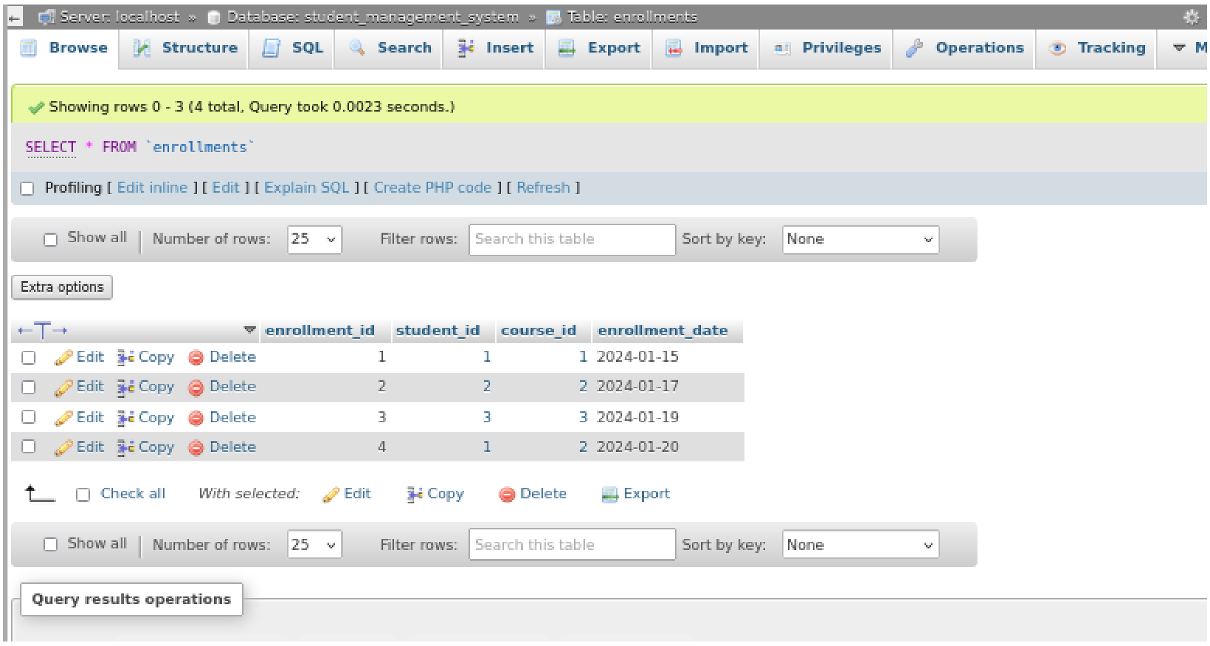
```
SELECT * FROM `courses`
```

	course_id	course_name	credits
□ Edit Copy Delete	1	Introduction to Databases	3
□ Edit Copy Delete	2	Web Development	4
□ Edit Copy Delete	3	Cybersecurity Fundamentals	3

3. Insert Sample Data into the enrollments Table:

- Enroll the students into different courses using this SQL statement:

- INSERT INTO enrollments (student_id, course_id, enrollment_date)
- VALUES
- (1, 1, '2024-01-15'),
- (2, 2, '2024-01-17'),
- (3, 3, '2024-01-19'),
- (1, 2, '2024-01-20');



Showing rows 0 - 3 (4 total, Query took 0.0023 seconds.)

```
SELECT * FROM `enrollments`
```

Number of rows: 25

Sort by key: None

	enrollment_id	student_id	course_id	enrollment_date
□ Edit Copy Delete	1	1	1	2024-01-15
□ Edit Copy Delete	2	2	2	2024-01-17
□ Edit Copy Delete	3	3	3	2024-01-19
□ Edit Copy Delete	4	1	2	2024-01-20

4. Reflection:

- Explain how inserting data into tables allows the database to become useful for managing information. Why is data consistency crucial when inserting related data across tables?
  - Without data, the database is just an empty structure with no practical use.
  - Data consistency is critical: enrollment records must reference valid students and valid courses.
  - If inconsistent data is inserted (e.g., enrolling a student who doesn’t exist), the system produces errors and unreliable results.
  - Consistent data ensures the database remains accurate, reliable, and useful for managing information in applications.

Exercise 4: Querying the Database

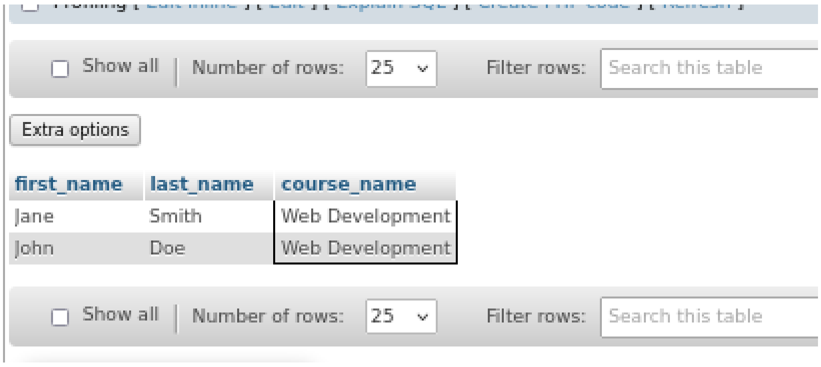
1. Retrieve All Students:

- Use a SQL query to retrieve all records from the students table:
  - SELECT * FROM students;

2. Retrieve Students Enrolled in a Specific Course:

- Write a SQL query to retrieve students enrolled in "Web Development":

- SELECT s.first_name, s.last_name, c.course_name
- FROM students s
- JOIN enrollments e ON s.student_id = e.student_id
- JOIN courses c ON e.course_id = c.course_id
- WHERE c.course_name = 'Web Development';



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

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

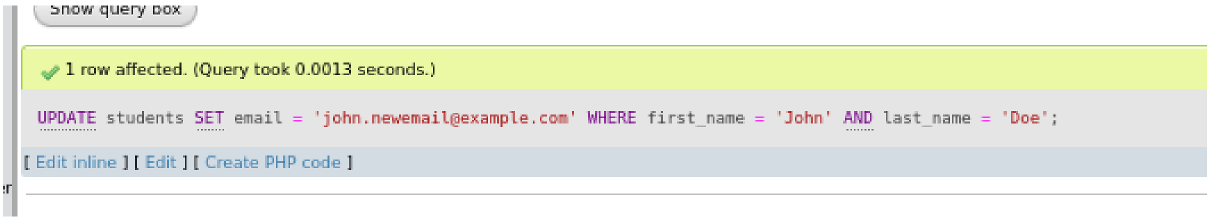
Extra options

first_name	last_name	course_name
Jane	Smith	Web Development
John	Doe	Web Development

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

3. Update a Student's Email Address:

- Update the email address for John Doe using the following SQL statement:
  - UPDATE students
  - SET email = 'john.newemail@example.com'
  - WHERE first_name = 'John' AND last_name = 'Doe';

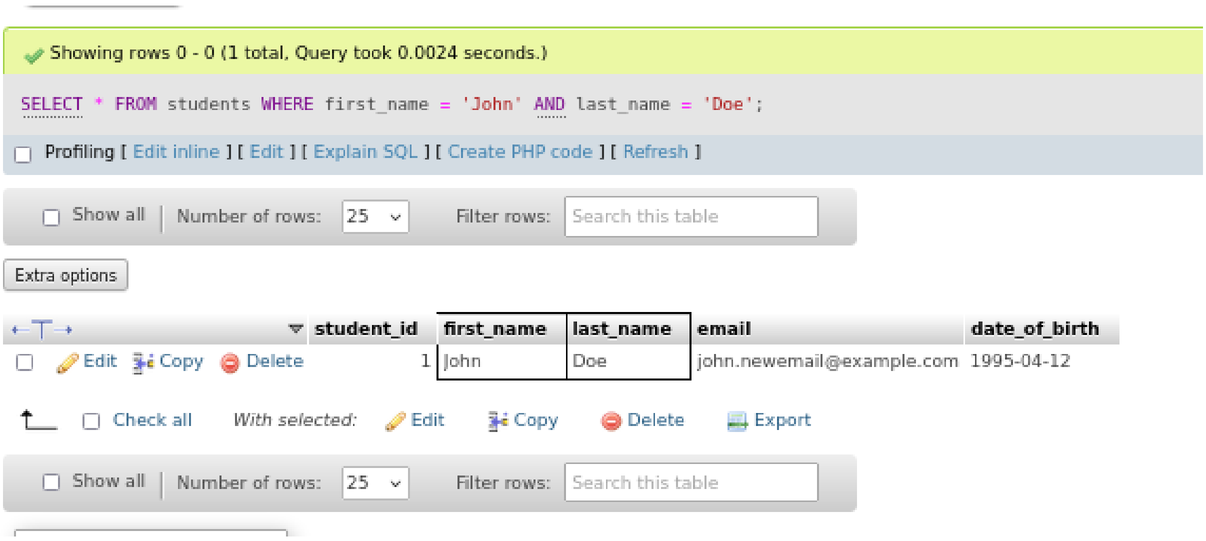


Show query box

✓ 1 row affected. (Query took 0.0013 seconds.)

```
UPDATE students SET email = 'john.newemail@example.com' WHERE first_name = 'John' AND last_name = 'Doe';
```

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



✓ Showing rows 0 - 0 (1 total, Query took 0.0024 seconds.)

```
SELECT * FROM students WHERE first_name = 'John' AND last_name = 'Doe';
```

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

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

Extra options

←T→	student_id	first_name	last_name	email	date_of_birth
□ Edit Copy Delete	1	John	Doe	john.newemail@example.com	1995-04-12

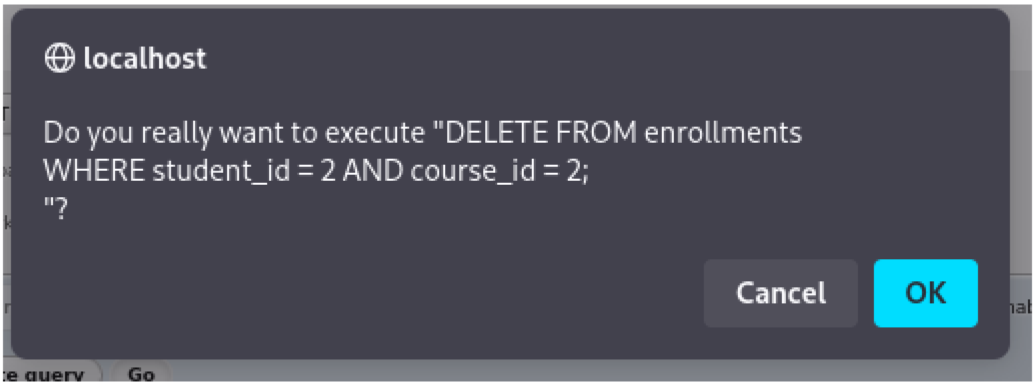
□ Check all *With selected:* Edit Copy Delete Export

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

4. Delete an Enrollment Record:

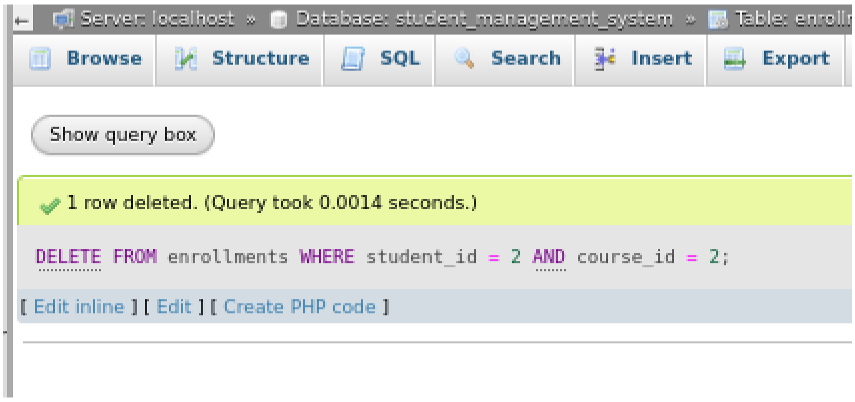
- Delete the enrollment record for Jane Smith from the enrollments table:
  - DELETE FROM enrollments

- WHERE student_id = 2 AND course_id = 2;



localhost

Do you really want to execute "DELETE FROM enrollments
WHERE student_id = 2 AND course_id = 2;
"?

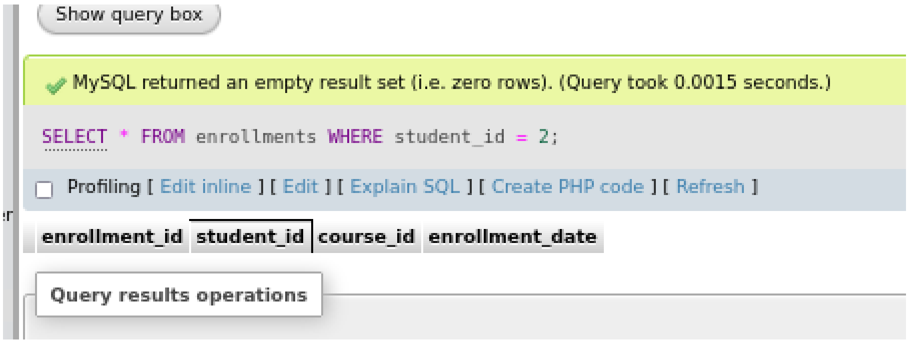


Server: localhost » Database: student_management_system » Table: enroll
Browse
Structure
SQL
Search
Insert
Export
Show query box

✓ 1 row deleted. (Query took 0.0014 seconds.)

```
DELETE FROM enrollments WHERE student_id = 2 AND course_id = 2;
```

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



✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0015 seconds.)

```
SELECT * FROM enrollments WHERE student_id = 2;
```

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

enrollment_id	student_id	course_id	enrollment_date

5. Reflection:

Discuss how SQL allows you to query and manipulate data in a relational database. How do JOINs facilitate retrieving related information from multiple tables?

- **SQL allows you to query and manipulate data in a relational database.**
  - SQL provides powerful commands (`SELECT`, `INSERT`, `UPDATE`, `DELETE`) to query and manipulate data.
  - It allows retrieval of specific information and modification of stored records.

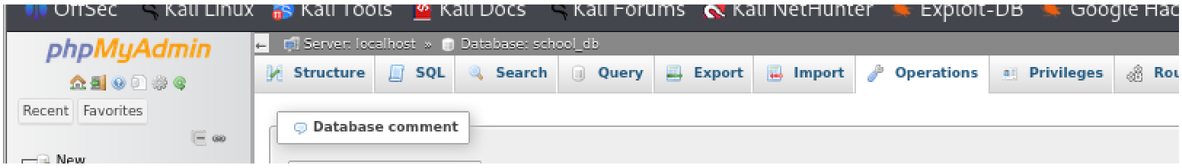
- **Importance of JOINs**:
  - JOINs connect related tables (e.g., students ↔ courses through enrollments).

  - They allow retrieving meaningful combined data (like "Which students are in which courses").
  - Without JOINs, related data would stay isolated in separate tables and be harder to use.

SQL CLASS

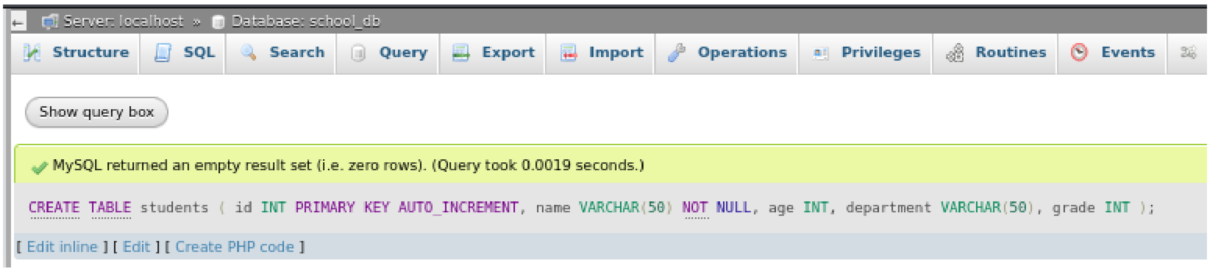
Create a new database

- CREATE DATABASE school_db;
  - USE school_db;



Create students table

- CREATE TABLE students (
  - id INT PRIMARY KEY AUTO_INCREMENT,
  - name VARCHAR(50) NOT NULL,
  - age INT,
  - department VARCHAR(50),
  - grade INT
  -);


Server: localhost » Database: school_db
Structure
SQL
Search
Query
Export
Import
Operations
Privileges
Routines
Events
Show query box

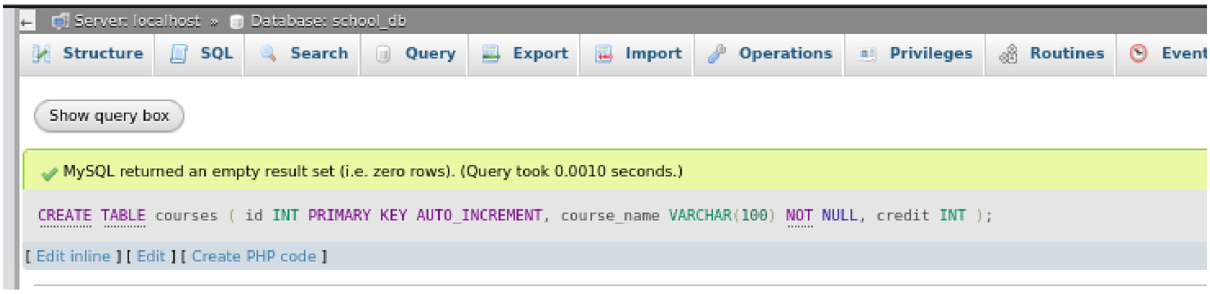
✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0019 seconds.)

```
CREATE TABLE students ( id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(50) NOT NULL, age INT, department VARCHAR(50), grade INT );
```

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

Create courses table

- CREATE TABLE courses (
  - id INT PRIMARY KEY AUTO_INCREMENT,
  - course_name VARCHAR(100) NOT NULL,
  - credit INT
  -);


Server: localhost » Database: school_db
Structure
SQL
Search
Query
Export
Import
Operations
Privileges
Routines
Event
Show query box

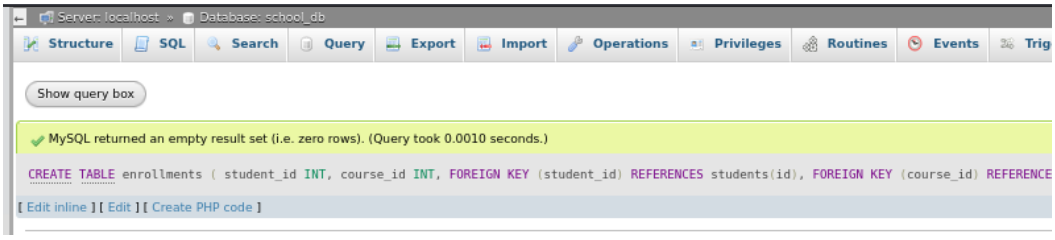
✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0010 seconds.)

```
CREATE TABLE courses ( id INT PRIMARY KEY AUTO_INCREMENT, course_name VARCHAR(100) NOT NULL, credit INT );
```

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

Create enrollments table (many-to-many relationship)

- CREATE TABLE enrollments (
  - student_id INT,
  - course_id INT,
  - FOREIGN KEY (student_id) REFERENCES students(id),
  - FOREIGN KEY (course_id) REFERENCES courses(id)
  -);

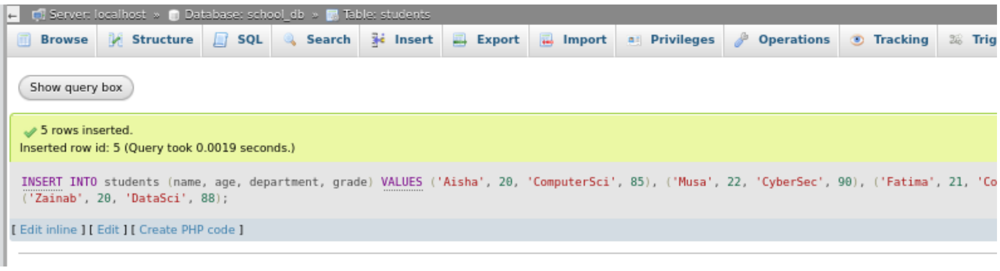


✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0010 seconds.)

```
CREATE TABLE enrollments ( student_id INT, course_id INT, FOREIGN KEY (student_id) REFERENCES students(id), FOREIGN KEY (course_id) REFERENCE
```

Insert into students

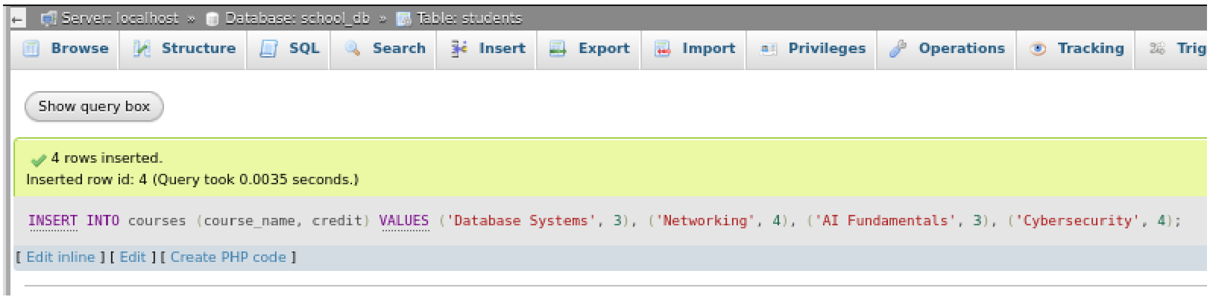
- INSERT INTO students (name, age, department, grade) VALUES
  - ('Aisha', 20, 'ComputerSci', 85),
  - ('Musa', 22, 'CyberSec', 90),
  - ('Fatima', 21, 'ComputerSci', 78),
  - ('John', 23, 'CyberSec', 60),
  - ('Zainab', 20, 'DataSci', 88); -- Insert into courses


Server: localhost » Database: school_db » Table: students
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Tracking
Trig
Show query box
[Edit inline] [Edit] [Create PHP code]

✓ 5 rows inserted.
Inserted row id: 5 (Query took 0.0019 seconds.)

```
INSERT INTO students (name, age, department, grade) VALUES ('Aisha', 20, 'ComputerSci', 85), ('Musa', 22, 'CyberSec', 90), ('Fatima', 21, 'Co
('Zainab', 20, 'DataSci', 88);
```

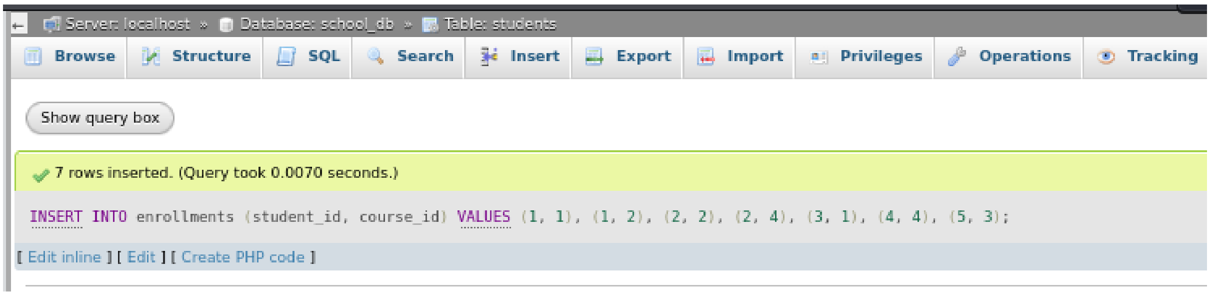
- INSERT INTO courses (course_name, credit) VALUES
  - ('Database Systems', 3),
  - ('Networking', 4),
  - ('AI Fundamentals', 3),
  - ('Cybersecurity', 4); -- Insert into enrollments


Server: localhost » Database: school_db » Table: students
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Tracking
Trig
Show query box
[Edit inline] [Edit] [Create PHP code]

✓ 4 rows inserted.
Inserted row id: 4 (Query took 0.0035 seconds.)

```
INSERT INTO courses (course_name, credit) VALUES ('Database Systems', 3), ('Networking', 4), ('AI Fundamentals', 3), ('Cybersecurity', 4);
```

- INSERT INTO enrollments (student_id, course_id) VALUES
  - (1, 1),
  - (1, 2),
  - (2, 2),
  - (2, 4),
  - (3, 1),
  - (4, 4),
  - (5, 3);

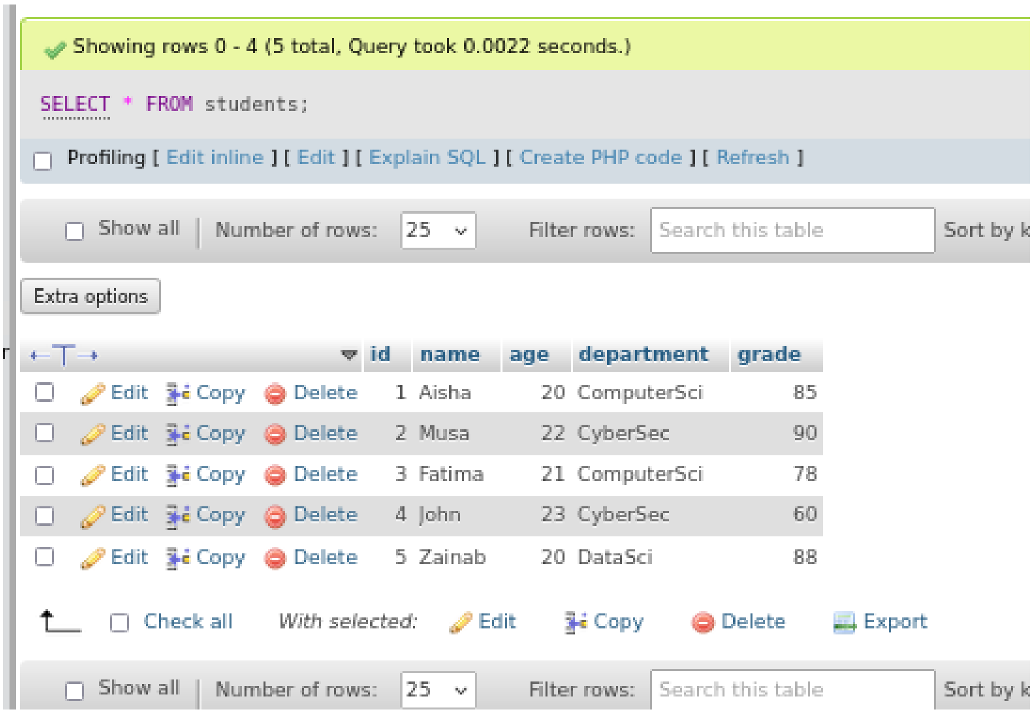

Server: localhost » Database: school_db » Table: students
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Tracking
Show query box
[Edit inline] [Edit] [Create PHP code]

✓ 7 rows inserted. (Query took 0.0070 seconds.)

```
INSERT INTO enrollments (student_id, course_id) VALUES (1, 1), (1, 2), (2, 2), (2, 4), (3, 1), (4, 4), (5, 3);
```

Exercise 1: Select all students

- 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 k
Extra options
□ Check all
With selected: Edit Copy Delete Export

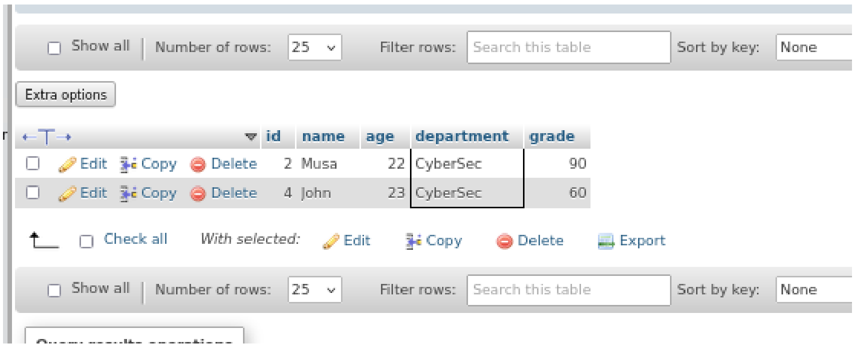
✓ Showing rows 0 - 4 (5 total, Query took 0.0022 seconds.)

```
SELECT * FROM students;
```

	id	name	age	department	grade
□ Edit Copy Delete	1	Aisha	20	ComputerSci	85
□ Edit Copy Delete	2	Musa	22	CyberSec	90
□ Edit Copy Delete	3	Fatima	21	ComputerSci	78
□ Edit Copy Delete	4	John	23	CyberSec	60
□ Edit Copy Delete	5	Zainab	20	DataSci	88

Exercise 2: Filter students by department

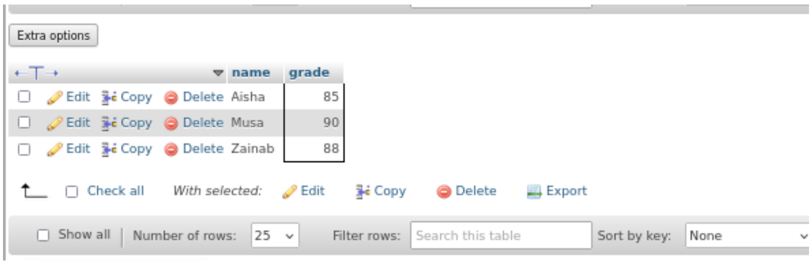
- SELECT * FROM students
    - WHERE department = 'CyberSec';


□ Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Extra options
Check all With selected: Edit Copy Delete Export

	id	name	age	department	grade
□ Edit Copy Delete	2	Musa	22	CyberSec	90
□ Edit Copy Delete	4	John	23	CyberSec	60

Exercise 3: Find top students

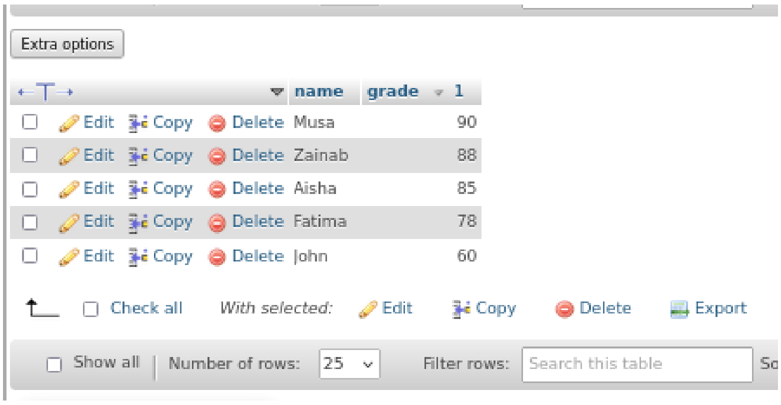
- SELECT name, grade FROM students
    - WHERE grade > 80;


Extra options
Check all With selected: Edit Copy Delete Export
□ Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

	name	grade
□ Edit Copy Delete	Aisha	85
□ Edit Copy Delete	Musa	90
□ Edit Copy Delete	Zainab	88

Exercise 4: Sort students by grade

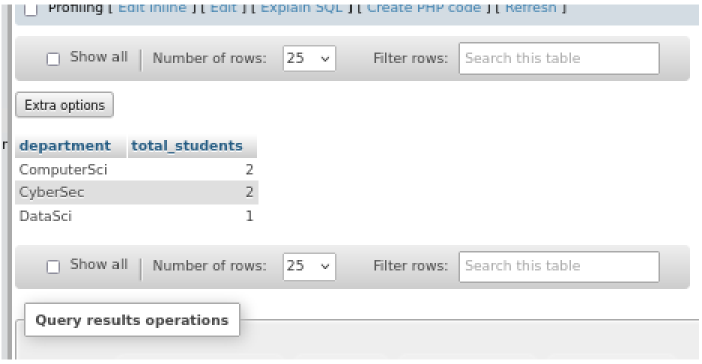
- SELECT name, grade FROM students
    - ORDER BY grade DESC;


Extra options
Check all With selected: Edit Copy Delete Export
□ Show all | Number of rows: 25 Filter rows: Search this table So

	name	grade 1
□ Edit Copy Delete	Musa	90
□ Edit Copy Delete	Zainab	88
□ Edit Copy Delete	Aisha	85
□ Edit Copy Delete	Fatima	78
□ Edit Copy Delete	John	60

Exercise 5: Count students per department

- SELECT department, COUNT(*) AS total_students
    - FROM students
    - GROUP BY department;

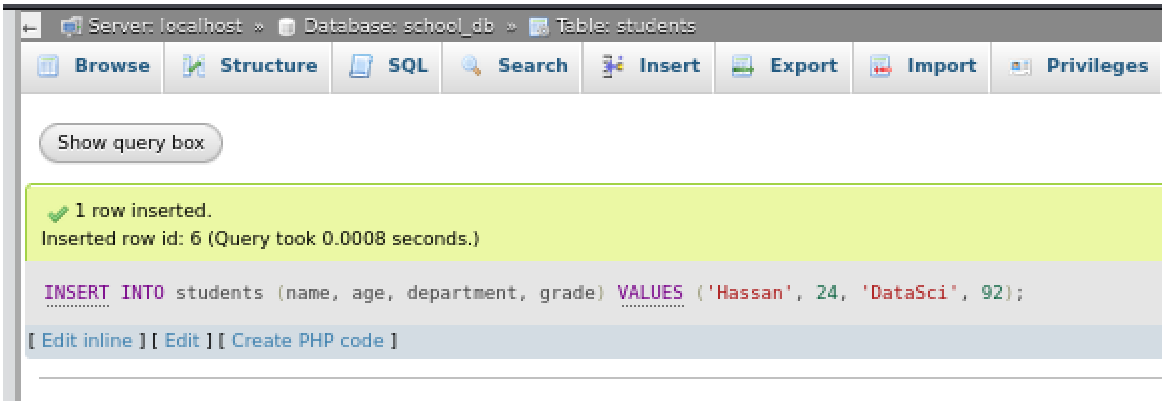

□ Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
□ Show all | Number of rows: 25 Filter rows: Search this table
Extra options

department	total_students
ComputerSci	2
CyberSec	2
DataSci	1

□ Show all | Number of rows: 25 Filter rows: Search this table
Query results operations

Exercise 6: Insert a new student

- INSERT INTO students (name, age, department, grade)
    - VALUES ('Hassan', 24, 'DataSci', 92);


Server: localhost » Database: school_db » Table: students
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Show query box

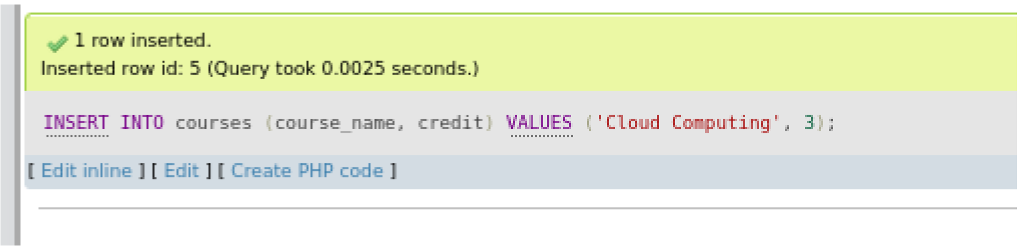
✓ 1 row inserted.
Inserted row id: 6 (Query took 0.0008 seconds.)

```
INSERT INTO students (name, age, department, grade) VALUES ('Hassan', 24, 'DataSci', 92);
```

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

Exercise 7: Insert a new course

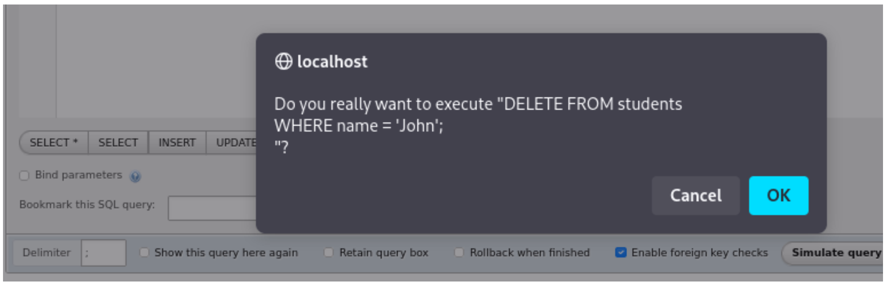
- INSERT INTO courses (course_name, credit)
    - VALUES ('Cloud Computing', 3);


✓ 1 row inserted.
Inserted row id: 5 (Query took 0.0025 seconds.)

```
INSERT INTO courses (course_name, credit) VALUES ('Cloud Computing', 3);
```

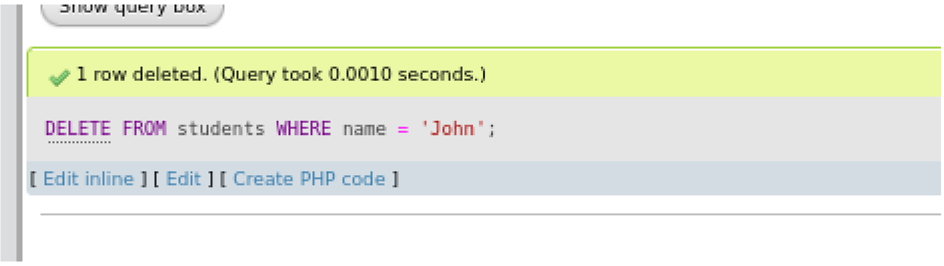
Exercise 8: Delete a student by name

- DELETE FROM students
  - WHERE name = 'John';



Do you really want to execute "DELETE FROM students
WHERE name = 'John';
"?

□ Show this query here again □ Retain query box □ Rollback when finished ☑ Enable foreign key checks

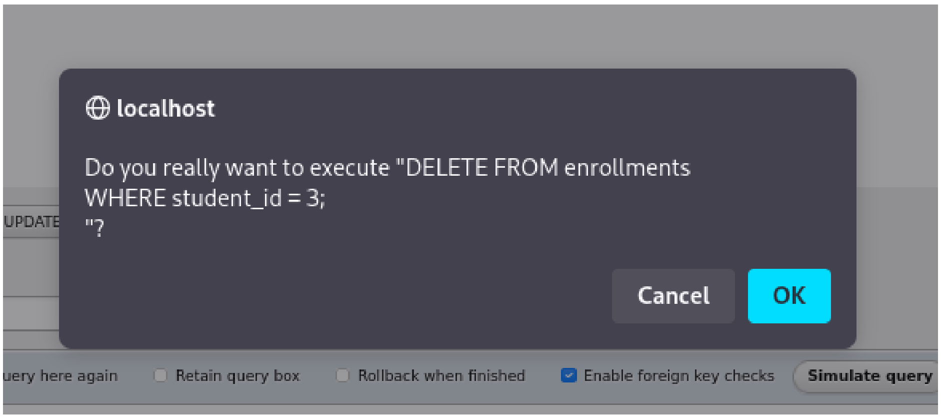


✓ 1 row deleted. (Query took 0.0010 seconds.)

```
DELETE FROM students WHERE name = 'John';
```

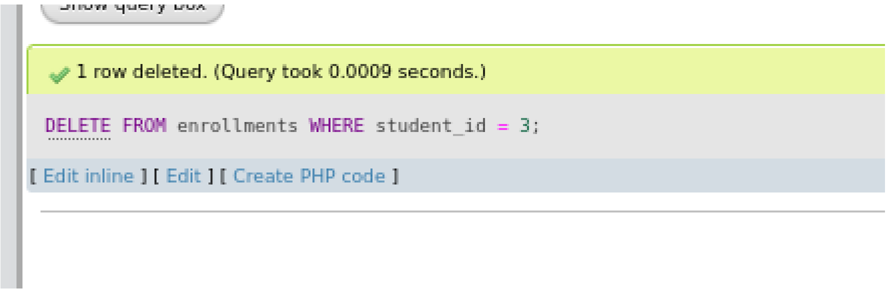
Exercise 9: Delete enrollments for a specific student

- DELETE FROM enrollments
  - WHERE student_id = 3;



Do you really want to execute "DELETE FROM enrollments
WHERE student_id = 3;
"?

uery here again □ Retain query box □ Rollback when finished ☑ Enable foreign key checks



✓ 1 row deleted. (Query took 0.0009 seconds.)

```
DELETE FROM enrollments WHERE student_id = 3;
```

Exercise 10: Drop a table

- DROP TABLE enrollments;

Before Dropping table enrolment

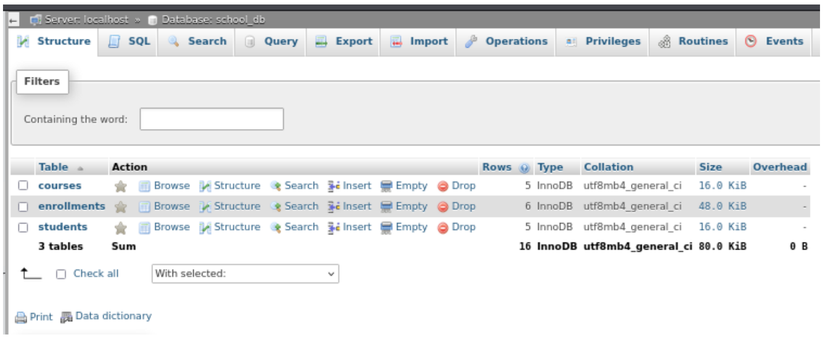
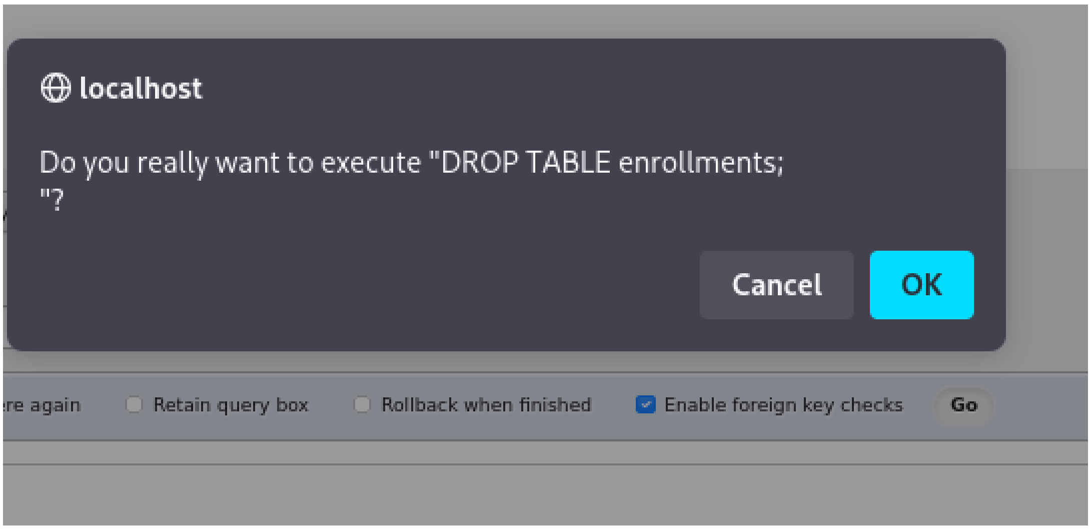

Server: localhost » Database: school_db
Structure SQL Search Query Export Import Operations Privileges Routines Events
Filters
Containing the word:

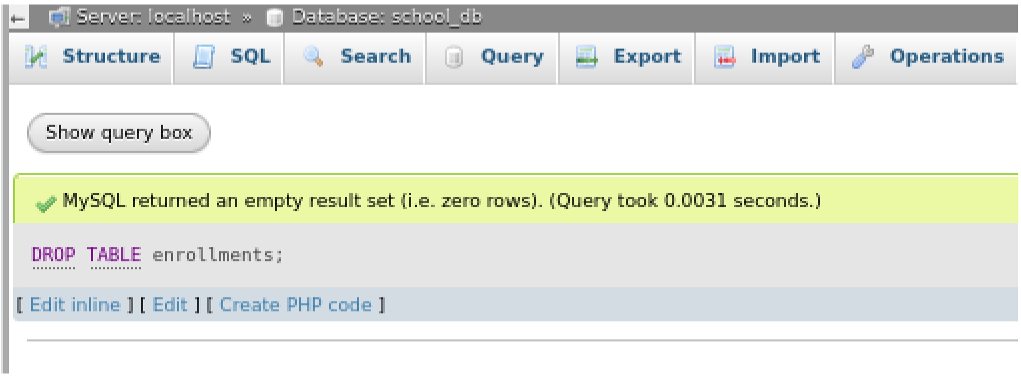
	Table	Action	Rows	Type	Collation	Size	Overhead
□	courses	Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_general_ci	16.0 KiB	-
□	enrollments	Browse Structure Search Insert Empty Drop	6	InnoDB	utf8mb4_general_ci	48.0 KiB	-
□	students	Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_general_ci	16.0 KiB	-
	3 tables	Sum	16	InnoDB	utf8mb4_general_ci	80.0 KiB	0 B

Prompt while dropping Table enrolment



Do you really want to execute "DROP TABLE enrollments;
"?

Table enrolment dropped


Server: localhost » Database: school_db
Structure SQL Search Query Export Import Operations
Show query box

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0031 seconds.)

```
DROP TABLE enrollments;
```

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

Image after Dropping Table enrolment

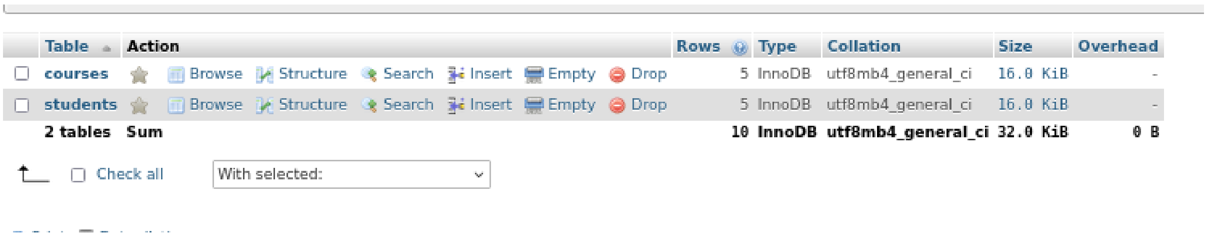


	Table	Action	Rows	Type	Collation	Size	Overhead
□	courses	☆ Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_general_ci	16.0 KiB	-
□	students	☆ Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_general_ci	16.0 KiB	-
	2 tables	Sum	10	InnoDB	utf8mb4_general_ci	32.0 KiB	0 B

Reflection

- **Data Retrieval**:
  - Queries like SELECT * FROM students; show how SQL retrieves complete datasets.
  - Filtering with WHERE and sorting with ORDER BY make the results more meaningful.
- **Data Analysis**:
  - Aggregation with COUNT() and GROUP BY helps summarize data, e.g., number of students per department.
  - Such queries support decision-making and reporting.
- **Data Manipulation**:
  - INSERT statements add new records, making the database dynamic and expandable.
  - UPDATE modifies existing records.
  - DELETE removes specific records while keeping the table intact.
- **Data Integrity & Relationships**:
  - Deleting enrolments for a student (DELETE FROM enrolments WHERE student_id = 3;) shows how relational links are managed without losing the student's main record.
- **Database Management**:
  - DROP TABLE demonstrates structural changes removing entire tables permanently.
  - This emphasizes the need for careful use of destructive commands.
- **Overall Takeaway**:
  - SQL provides a complete toolkit for **CRUD** operations (Create, Read, Update, Delete) and structural management.
  - Proper use of queries ensures that data remains accurate, consistent, and useful for applications like school management systems.