Using JDBC for Simple Queries

Estimated time: **45 minutes**

Overview

In this lab, you will learn:

- How to use **JDBC Statements** to execute SQL queries.
- How to retrieve data using **basic SQL queries**.
- How to refine and manipulate query results using **filtering, sorting, grouping, and pagination**.
- How to **combine data from multiple tables** using JOINs.
- Best practices for **error handling** in JDBC.

By the end of this lab, you will be able to execute SQL queries using JDBC efficiently.

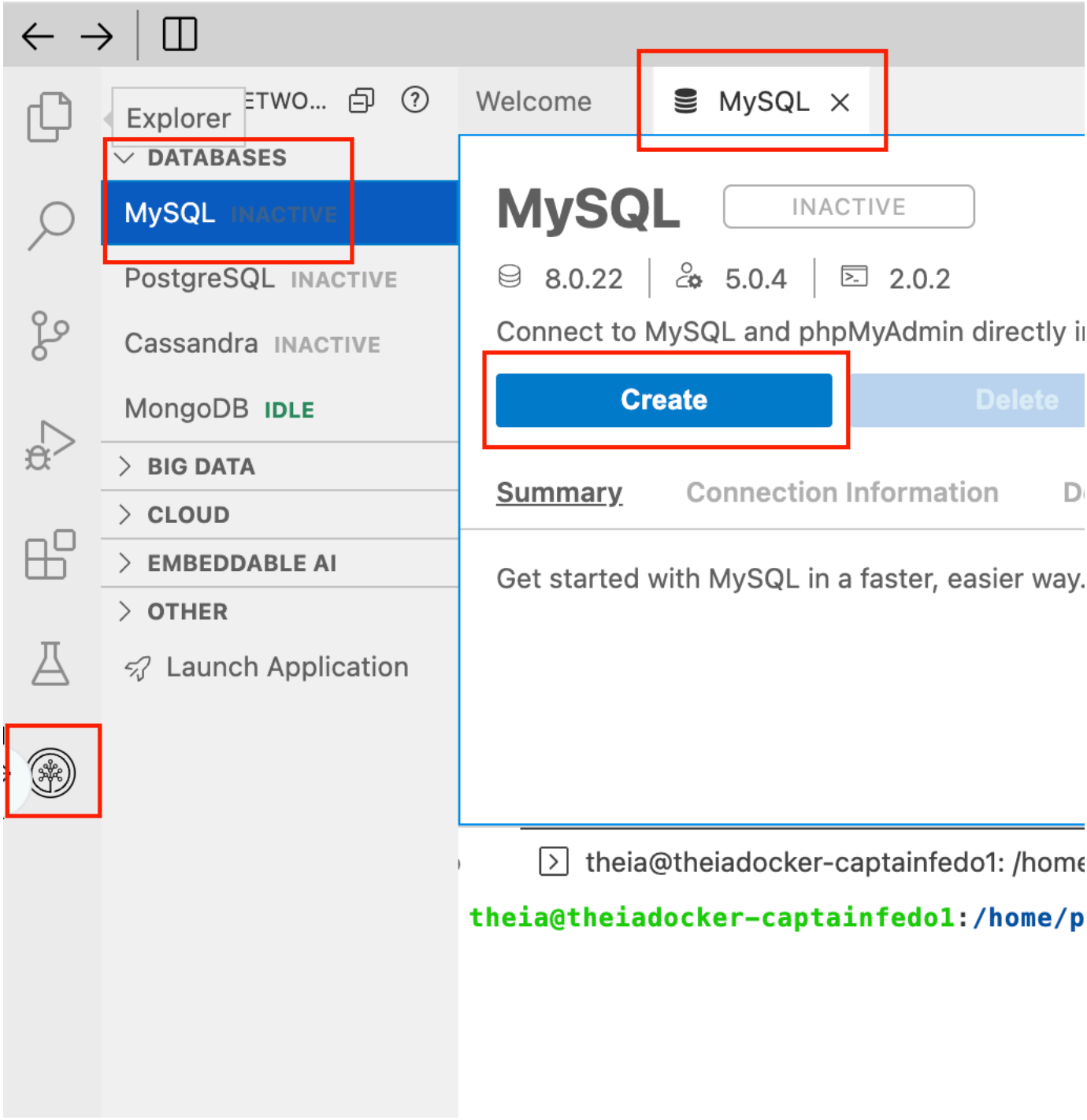
Learning Objectives

After completing this lab, you will be able to:

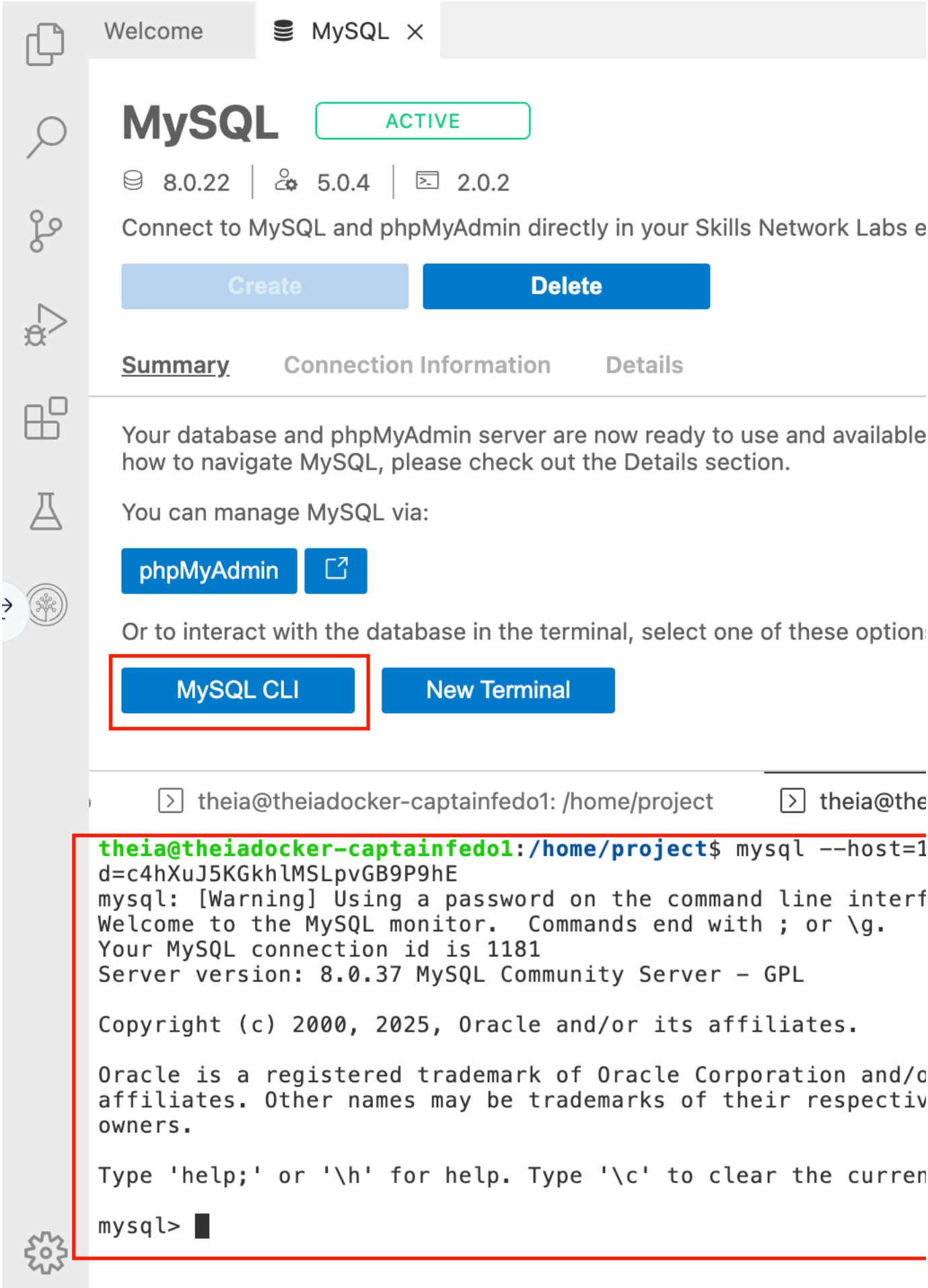
- Use **JDBC Statement** to execute SQL queries.
- Apply filtering, sorting, and grouping to refine results.
- Use **JOINs** to retrieve data from multiple tables.
- Handle NULL values and errors properly in JDBC.

Starting MySQL Database

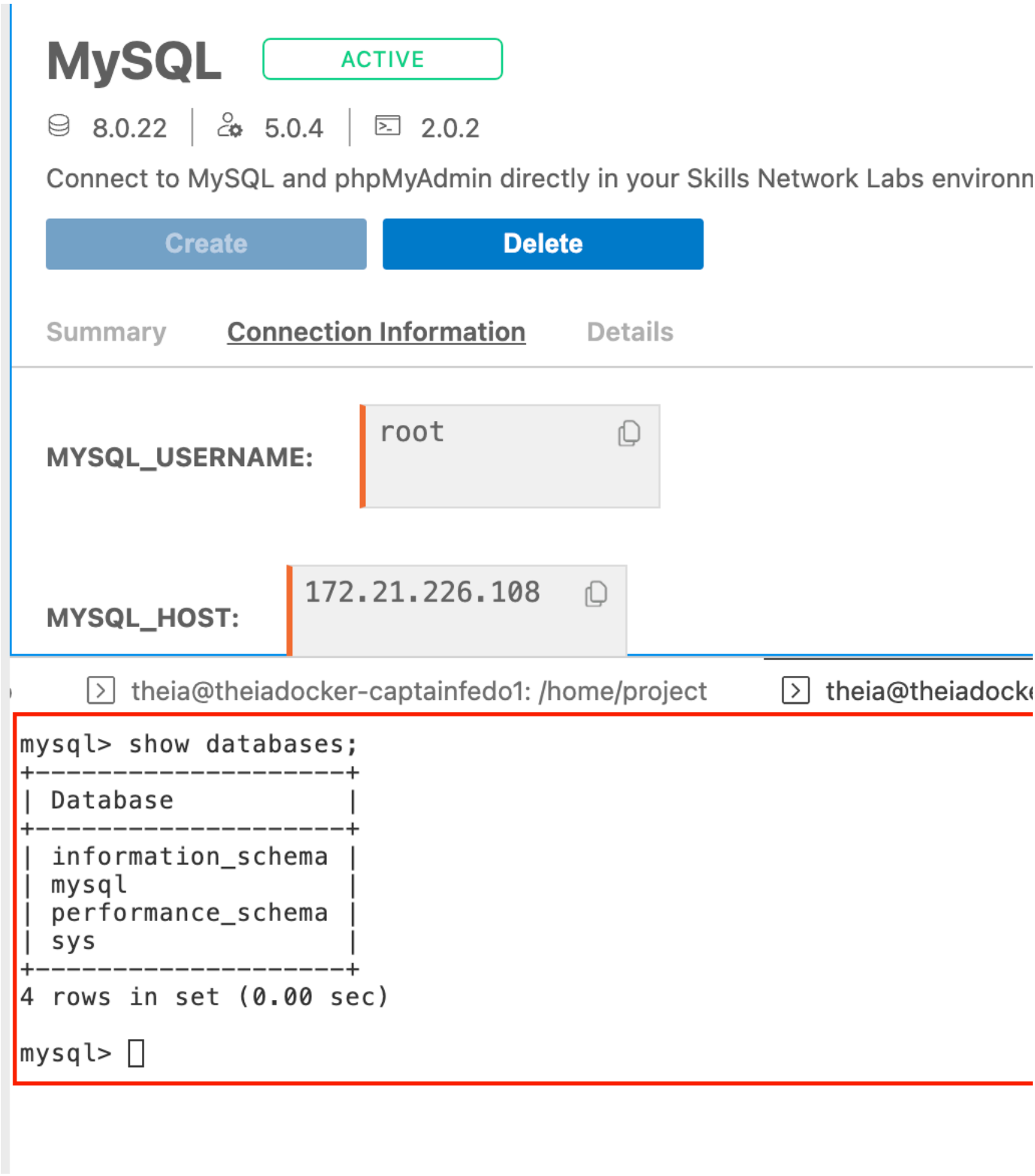
1. The first step is to create the MySQL instance in the IDE environment. Click the toolbox icon at the bottom of the left-hand panel. This will bring up another panel on the left. Select MySQL in the databases category. This will open a new tab. Click the `Create` button.



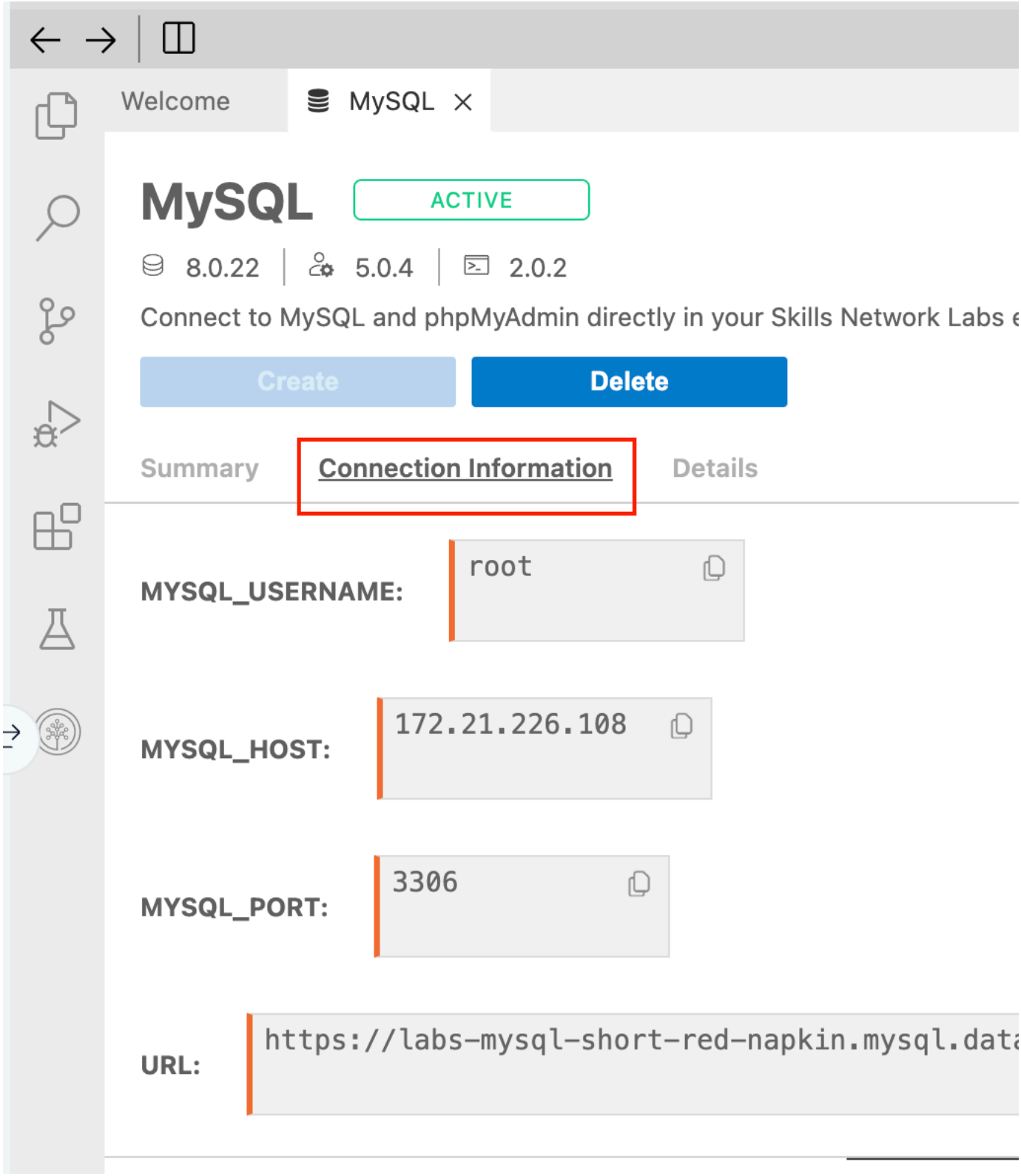
2. The MySQL Database will start in a few minutes. Once it has started, click the `MySQL CLI`button in the `Summary` tab. This will open the MySQL terminal at the bottom of the screen.



3. You can now enter all the **MySQL-based** commands in this lab in this terminal. This example shows the command to list all the databases.



4. You can find the username, password, and connection URL information in the `Connection Information` tab if you need to connect to this database outside the IDE environment.



Setting up the Tables

You will need to create three tables for this lab: `Courses`, `Students`, and `Enrollments` in this order for the dependencies to work.

Create a database

The first step is to create a database. Use the following command to create a database called MY_DATABASE.

```
CREATE DATABASE MY_DATABASE;
```

Use the `show databases;` command to test whether the command worked. You should see your database listed in the output:

```
+--------------------+
| Database           |
+--------------------+
| MY_DATABASE        |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)
```

Next, use the database for future commands.

```
USE MY_DATABASE;
```

Create the Courses table

Run the following SQL command in your MySQL CLI to create the Courses table.

```
CREATE TABLE Courses (
    CourseID INT PRIMARY KEY AUTO_INCREMENT,
    CourseName VARCHAR(100) NOT NULL,
    Department VARCHAR(50),
    Credits INT CHECK (Credits > 0) -- Ensuring credit hours are positive
);
```

Now, populate the Courses table with sample courses.

```
INSERT INTO Courses (CourseName, Department, Credits) VALUES
('Introduction to Programming', 'Computer Science', 4),
('Calculus I', 'Mathematics', 3),
('Linear Algebra', 'Mathematics', 3),
('Probability and Statistics', 'Mathematics', 3),
('Differential Equations', 'Mathematics', 4),
('English Literature', 'Humanities', 3),
('Modern History', 'History', 3),
('Physics I', 'Physics', 4),
('Database Systems', 'Computer Science', 4),
('Artificial Intelligence', 'Computer Science', 3);
```

Create the Students table

Let's create a table called Students with four columns. StudentID, Name, Age, and Major. The StudentID is the primary key. Use the `CREATE TABLE` command to create the table.

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100) NOT NULL,
    Age INT,
    Major VARCHAR(50),
    GPA DECIMAL(3,2)
);
```

Let's populate this table with the following data.

```
INSERT INTO Students (Name, Age, Major, GPA) VALUES
('Alice Johnson', 20, 'Computer Science', 3.8),
('Bob Smith', 22, 'Mathematics', 3.5),
('Charlie Brown', 19, 'History', 3.2),
('David Lee', 21, 'Computer Science', 3.9),
('Eve Wilson', 23, 'English', 3.4),
('Frank Miller', 20, 'Mathematics', 3.6),
('Grace Davis', 22, 'History', 3.1),
('Henry Garcia', 19, 'Computer Science', 3.7),
('Ivy Rodriguez', 21, 'English', 3.3),
('Jack Martinez', 23, 'Mathematics', 3.6),
('Karen White', 20, 'Computer Science', 3.8),
('Liam Green', 22, 'Mathematics', 3.5),
('Mia Taylor', 19, 'History', 3.3),
('Noah Anderson', 21, 'English', 3.4),
('Olivia Thomas', 23, 'Computer Science', 3.9),
('Peter Jackson', 20, 'Mathematics', 3.6),
('Quinn Moore', 22, 'History', 3.2),
('Ryan Martin', 19, 'English', 3.1),
('Sophia Thompson', 21, 'Computer Science', 3.7),
('Tyler Garcia', 23, 'Mathematics', 3.4),
('Ursula Perez', 20, 'Computer Science', 3.5);
```

Create the Enrollments table

This table links students to the courses they are enrolled in.

```
CREATE TABLE Enrollments (
    EnrollmentID INT PRIMARY KEY AUTO_INCREMENT,
    StudentID INT,
    CourseID INT,
    EnrollmentDate DATE NOT NULL,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
```

Next, let's add some data to this table.

```
INSERT INTO Enrollments (StudentID, CourseID, EnrollmentDate) VALUES
(1, 1, '2024-09-05'),  -- Alice Johnson in Intro to Programming
(1, 2, '2024-09-05'),  -- Alice Johnson in Calculus I
(3, 3, '2024-09-05'),  -- Charlie Brown in Linear Algebra
(5, 1, '2024-09-05'),  -- Eve Wilson in Intro to Programming
(7, 2, '2024-09-05'),  -- Grace Davis in Calculus I
(9, 3, '2024-09-05'),  -- Henry Garcia in Linear Algebra
(11, 1, '2024-09-05'), -- Karen White in Intro to Programming
(13, 2, '2024-09-05'), -- Mia Taylor in Calculus I
(15, 3, '2024-09-05'), -- Noah Anderson in Linear Algebra
(17, 1, '2024-09-05'), -- Ryan Martin in Intro to Programming
(19, 2, '2024-09-05'), -- Sophia Thompson in Calculus I
(21, 3, '2024-09-05'); -- Tyler Garcia in Linear Algebra
```

Connecting MySQL with Java Using VS Code

To connect **MySQL** with **Java** using **VS Code**, follow the steps below:

Step 1: Open Terminal in VS Code

1. Switch to the standard `theia@theiadocker` terminal (not the MySQL CLI one).
2. Use the command below to ensure you are in the `project` directory.

```
cd /home/project
```

Step 2: Create the Maven project

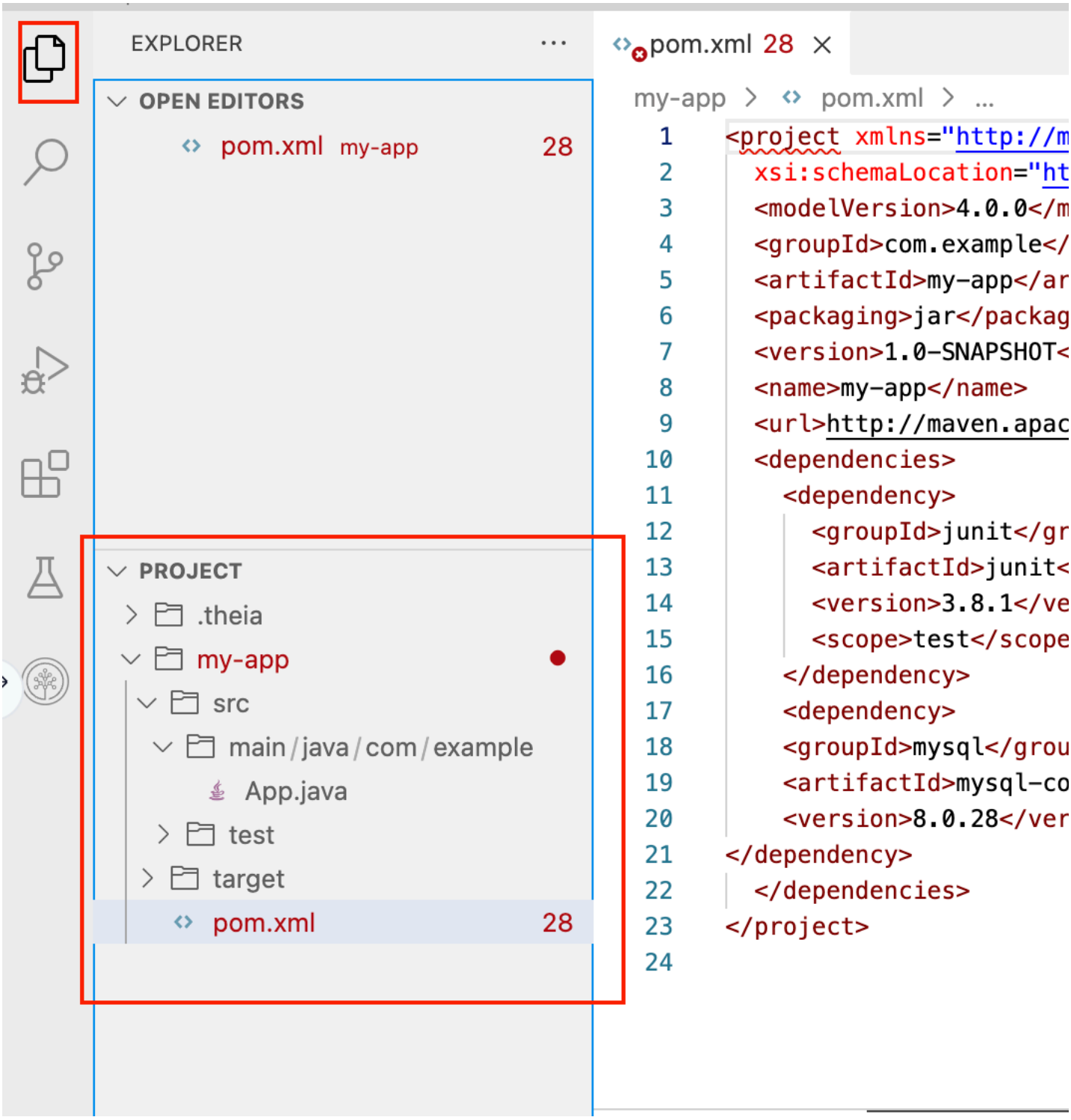
```
mvn archetype:generate \
    -DgroupId=com.example \
    -DartifactId=my-app \
    -Dversion=1.0-SNAPSHOT \
    -Dpackage=com.example \
    -DinteractiveMode=false
```

This Maven command generates a new project structure.

- It uses the `archetype:generate` goal to create a project based on a predefined template.
- The -D flags specify key project details:
  - `-DgroupId` (typically a reverse domain name)
  - `-DartifactId` (the project's name)
  - `-Dversion`
  - `-Dpackage`
- The `-DinteractiveMode=false` flag ensures the command runs non-interactively, using the provided parameters without prompting for further input.

Essentially, it's a quick way to bootstrap a new Java project with a standard directory layout and pom.xml configuration.

This should create a new directory called `my-app` in the project directory. You can see the directory structure in the `Explore` panel of the IDE.



Step 3: Add MySQL Connector Dependency

Open the `pom.xml` file in your project folder.

You will see some errors in `pom.xml` file. This can be fixed by changing `http` to `https` for the `project` tag. Change from:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
...
```

To the following:

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
...
```

Add the following MySQL connector dependency within the `<dependencies>` tag:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.28</version>
</dependency>
```

So it might look something like:

```
    <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.28</version>
    </dependency>
  </dependencies>
```

- After modifying your `pom.xml`, it should look like this:

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>my-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>my-app</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.28</version>
    </dependency>
  </dependencies>
</project>
```

Step 4: Install the dependencies

1. Run the command below to make sure you are in the `my-app` directory and run the `mvn install` command to install the dependencies.

```
cd /home/project/my-app && mvn install
```

The output should show a success message:

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.060 s
[INFO] Finished at: 2025-03-21T02:07:45-04:00
[INFO] ------------------------------------------------------------------------
```

Step 5: Navigate to the `App.java` File

Here's the path: **home/project/my-app/src/main/java/com/example/App.java**

Alternatively, click the link below to open the file.

Open **App.java** in IDE

Step 6: Establish Database Connectivity

To establish a connection to MySQL, replace the code with the following code inside the `App.java`file.

```
package com.example;
import java.sql.*;
public class App {
    public static void main(String[] args) {
        String url = "jdbc:mysql://{MYSQL_HOST}:3306/{DATABASE_NAME}"; // Change to your database
        String user = "{MYSQL_USERNAME}"; // Your database username
        String password = "{MYSQL_PASSWORD}"; // Your database password
        try {
            // Establishing connection
            Connection conn = DriverManager.getConnection(url, user, password);
            Statement statement = conn.createStatement();
            System.out.println("connection successful!");
            // Your JDBC code here
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

To establish the connection, you need to replace the `{MYSQL_HOST}`, `{DATABASE_NAME}`, `{MYSQL_USERNAME}`, and `{MYSQL_PASSWORD}` with the database details. You'll find all the details, except the Database Name, in the MySQL connection information as explained at the start of the lab.

- Replace `{MYSQL_HOST}` with the MYSQL_HOST value in the MySQL connection information
- Replace the `{DATABASE_NAME}`with your Database name. In this case it is `MY_DATABASE`.
- Replace `{MYSQL_USERNAME}`, and `{MYSQL_PASSWORD}` as per your MySQL database credentials.

For example, for my environment, it looks like this:

```
String url = "jdbc:mysql://172.21.128.49:3306/MY_DATABASE"; // Change to your database
String user = "root"; // Your database username
String password = "WUrb0ISQ3hroWlAqHWSUsfiM"; // Your database password
```

After setting up the project, you can proceed with running your JDBC queries inside the `try` block.

Run the Application

Make sure you are in the `my-app` directory in the terminal. If you are not in the `my-app` directory, you can use the command below.

```
cd /home/project/my-app
```

Use the following command to run the project.

```
mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

You should see the logs say "connection successful!".

```
theia@theiadocker-captainfedo1:/home/project/my-app$ mvn exec:java -Dexec.mainClass="com.example.App"
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< com.example:my-app >------------------------
[INFO] Building my-app 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.5.0:java (default-cli) @ my-app ---
connection successful!
```

Simple Querying

Now, as the database connection has been established you can run SQL queries using Java Database Connectivity (JDBC).

Query 1: Retrieve information using the SELECT command

To retrieve information about all students, you can add the following code to the `/home/project/my-app/src/main/java/com/example/App.java` where it says `// Your JDBC code here`.

```
ResultSet resultSet = statement.executeQuery("SELECT * FROM Students");
while (resultSet.next()) {
    System.out.println("Student ID: " + resultSet.getInt("StudentID"));
    System.out.println("Name: " + resultSet.getString("Name"));
    System.out.println("Major: " + resultSet.getString("Major"));
}
```

Explanation

1. **executeQuery("SELECT * FROM Students")**: Runs the SQL `SELECT` query.
2. **ResultSet**: Stores the retrieved data.
3. **next() method**: Moves to the next row in the result set.

Run the code using the terminal again. Ensure you are in the `my-app` directory. We ask that first you `clean install` before running.

```
mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

You should see all the students printed in the terminal.

```
connection successful!
Student ID: 1
Name: Alice Johnson
Major: Computer Science
Student ID: 2
Name: Bob Smith
Major: Mathematics
Student ID: 3
Name: Charlie Brown
Major: History
Student ID: 4
Name: David Lee
Major: Computer Science
...
```

Query 2: Filter Data with WHERE Clause

Next, you want to retrieve students from a specific major. Add the following code under the previous code and save the file.

```
resultSet = statement.executeQuery(
    "SELECT * FROM Students WHERE Major = 'Computer Science'"
);
System.out.println("Students with major of Computer Science");
while (resultSet.next()) {
    System.out.println("Name: " + resultSet.getString("Name"));
}
```

Explanation

- `WHERE Major = 'Computer Science'` filters students by major.
- **ResultSet Navigation**: The `next()` method is used to iterate through the filtered results.

Run the code using the terminal again, ensuring you are in the `my-app` directory.

```
mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

You will see the students with the major of `Computer Science` printed at the end of the previous output.

```
Students with major of Computer Science
Name: Alice Johnson
Name: David Lee
Name: Henry Garcia
Name: Karen White
Name: Olivia Thomas
Name: Sophia Thompson
Name: Ursula Perez
```

Query 3: Sort results with ORDER BY

You also want to print students sorted by GPA (descending order). Add the following code under the previous code. Save the file before you run it.

```
resultSet = statement.executeQuery(
    "SELECT * FROM Students ORDER BY GPA DESC"
);
System.out.println("Printing students with descending GPAs");
while (resultSet.next()) {
    System.out.println("Name: " + resultSet.getString("Name"));
    System.out.println("GPA: " + resultSet.getDouble("GPA"));
}
```

Explanation

- **ORDER BY GPA DESC**: Sorts results by GPA in descending order.
- Sorting can be applied to multiple columns using **ORDER BY GPA DESC, Name ASC**.

Run the code using the terminal again, ensuring you are in the `my-app` directory.

```
mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

You should see GPAs (most to least) printed with student names. You'll find them at the end of the previous output.

```
Printing students with descending GPAs
Name: David Lee
GPA: 3.9
Name: Olivia Thomas
GPA: 3.9
Name: Alice Johnson
GPA: 3.8
Name: Karen White
GPA: 3.8
Name: Sophia Thompson
GPA: 3.7
```

Query 4: Limit Results

Then, you want to retrieve only the top five students by GPA.

Add the following code right below the previous code.

```
resultSet = statement.executeQuery(
    "SELECT * FROM Students ORDER BY GPA DESC LIMIT 5"
);
System.out.println("Limiting resultset to 5");
while (resultSet.next()) {
    System.out.println("Name: " + resultSet.getString("Name"));
}
```

Explanation

- **LIMIT 5**: Restricts the result set to only 5 rows.
- In **SQL Server**, use `TOP 5` instead of `LIMIT`.

Now, run the code using the terminal again, ensuring you are in the `my-app` directory.

```
mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

This time, you should only see top 5 students by GPA. This will be printed below the previous output.

```
Limiting resultset to 5
Name: David Lee
Name: Olivia Thomas
Name: Alice Johnson
Name: Karen White
Name: Henry Garcia
```

Intermediate Querying

Query 1: Aggregate data with GROUP BY

Let's say you want to count the students by major. Add the following code right below the previous code.

```
resultSet = statement.executeQuery(
    "SELECT Major, COUNT(*) AS StudentCount FROM Students GROUP BY Major"
);
System.out.println("Count students by major");
while (resultSet.next()) {
    System.out.println("Major: " + resultSet.getString("Major"));
    System.out.println("Number of Students: " + resultSet.getInt("StudentCount"));
}
```

Explanation

- **COUNT(*)**: Counts the number of students per major.
- **GROUP BY Major**: Groups students by major.

Now, run the code using the terminal again, ensuring you are in the `my-app` directory.

```
mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

You should see an output like this at the end of the previous result sets.

```
Count students by major
Major: Computer Science
Number of Students: 7
Major: Mathematics
Number of Students: 6
Major: History
```

```
Number of Students: 4
Major: English
Number of Students: 4
```

Query 2: Join tables

Next, you want to retrieve the student names and their courses. Add the following code after the previous code.

```
resultSet = statement.executeQuery(
    "SELECT Students.Name, Courses.CourseName " +
    "FROM Students " +
    "INNER JOIN Enrollments ON Students.StudentID = Enrollments.StudentID " +
    "INNER JOIN Courses ON Enrollments.CourseID = Courses.CourseID"
);
System.out.println("Retrieve Student Names and Their Courses");
while (resultSet.next()) {
    System.out.println("Student: " + resultSet.getString("Name"));
    System.out.println("Course: " + resultSet.getString("CourseName"));
}
```

Explanation

- **INNER JOIN**: Combines data from multiple tables based on a common key.
- This query retrieves students along with the courses they are enrolled in.

Again, run the code using the terminal again, ensuring you are in the `my-app` directory.

```
mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

You should see the following output at the end of the logs.

```
Retrieve Student Names and Their Courses
Student: Alice Johnson
Course: Introduction to Programming
Student: Alice Johnson
Course: Calculus I
Student: Charlie Brown
Course: Linear Algebra
Student: Eve Wilson
Course: Introduction to Programming
Student: Grace Davis
Course: Calculus I
Student: Ivy Rodriguez
Course: Linear Algebra
Student: Karen White
Course: Introduction to Programming
Student: Mia Taylor
Course: Calculus I
Student: Olivia Thomas
Course: Linear Algebra
Student: Quinn Moore
Course: Introduction to Programming
Student: Sophia Thompson
Course: Calculus I
Student: Ursula Perez
Course: Linear Algebra
```

Query 3: Handle NULL values

Suppose you want to retrieve students without a declared major. Add the following code after the previous code.

```
resultSet = statement.executeQuery(
    "SELECT * FROM Students WHERE Major IS NULL"
);
System.out.println("Retrieve Students Without a Declared Major");
while (resultSet.next()) {
    System.out.println("Name: " + resultSet.getString("Name"));
}
```

Run the code using the terminal again, ensuring you are in the `my-app` directory.

```
mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

Since all students have a declared major, you will see nothing in the logs.

```
Retrieve Students Without a Declared Major
```

Run Maven Project

Follow the steps below to run your Maven project.

Step 1: Open the Terminal in VS code

- Navigate to your project directory using the terminal.

Step 2: Run the command to build the project

```
mvn clean install
```

Step 3: Run the application

- Use the following command to run the project.

```
mvn exec:java -Dexec.mainClass="com.example.App"
```

- Replace `com.example.App` with the actual main class of your project if you have used a custom name for your main class.

Completed Solution

Here is the complete `App.java` file if you got stuck.

```
package com.example;
import java.sql.*;
public class App {
    public static void main(String[] args) {
        String url = "jdbc:mysql://172.21.128.49:3306/MY_DATABASE"; // Change to your database
        String user = "root"; // Your database username
        String password = "WUrb0ISQ3hroWlAqHWSUsfiM"; // Your database password
        try {
            // Establishing connection
            Connection conn = DriverManager.getConnection(url, user, password);
            Statement statement = conn.createStatement();
            System.out.println("connection successful!");
            // Your JDBC code here
            ResultSet resultSet = statement.executeQuery("SELECT * FROM Students");
            while (resultSet.next()) {
                System.out.println("Student ID: " + resultSet.getInt("StudentID"));
                System.out.println("Name: " + resultSet.getString("Name"));
                System.out.println("Major: " + resultSet.getString("Major"));
            }
            resultSet = statement.executeQuery(
                    "SELECT * FROM Students WHERE Major = 'Computer Science'");
            System.out.println("Students with major of Computer Science");
            while (resultSet.next()) {
                System.out.println("Name: " + resultSet.getString("Name"));
            }
            resultSet = statement.executeQuery(
                    "SELECT * FROM Students ORDER BY GPA DESC");
            System.out.println("Printing students with descending GPAs");
            while (resultSet.next()) {
                System.out.println("Name: " + resultSet.getString("Name"));
                System.out.println("GPA: " + resultSet.getDouble("GPA"));
            }
            resultSet = statement.executeQuery(
                    "SELECT * FROM Students ORDER BY GPA DESC LIMIT 5");
            System.out.println("Limiting resultset to 5");
            while (resultSet.next()) {
                System.out.println("Name: " + resultSet.getString("Name"));
            }
            resultSet = statement.executeQuery(
                    "SELECT Major, COUNT(*) AS StudentCount FROM Students GROUP BY Major");
            System.out.println("Count students by major");
            while (resultSet.next()) {
                System.out.println("Major: " + resultSet.getString("Major"));
                System.out.println("Number of Students: " + resultSet.getInt("StudentCount"));
            }
            resultSet = statement.executeQuery(
                    "SELECT Students.Name, Courses.CourseName " +
                            "FROM Students " +
                            "INNER JOIN Enrollments ON Students.StudentID = Enrollments.StudentID " +
                            "INNER JOIN Courses ON Enrollments.CourseID = Courses.CourseID");
            System.out.println("Retrieve Student Names and Their Courses");
            while (resultSet.next()) {
                System.out.println("Student: " + resultSet.getString("Name"));
                System.out.println("Course: " + resultSet.getString("CourseName"));
            }
            resultSet = statement.executeQuery(
                    "SELECT * FROM Students WHERE Major IS NULL");
            System.out.println("Retrieve Students Without a Declared Major");
            while (resultSet.next()) {
                System.out.println("Name: " + resultSet.getString("Name"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Conclusion and Next Steps

Congratulations! You have successfully executed SQL queries using JDBC, demonstrating your ability to connect Java applications to a MySQL database and perform a variety of data retrieval and manipulation tasks. You've learned how to use JDBC Statement objects to execute SQL queries, filter and sort results, aggregate data, join tables, and handle NULL values.

This lab provided a practical introduction to using JDBC for simple queries, laying a strong foundation for more complex database interactions. By mastering these fundamental techniques, you can build robust and efficient Java applications that interact with relational databases.

Next Steps:

1. Parameterized queries and prepared statements: Explore how to use PreparedStatement to prevent SQL injection and improve query performance. Practice creating parameterized queries that accept user input safely.
2. Transactions and batch processing: Learn how to manage database transactions to ensure data integrity and consistency. Experiment with batch processing to execute multiple SQL statements efficiently.
3. Advanced error handling: Dive deeper into JDBC error handling by catching specific SQLExceptions and providing more informative error messages. Practice implementing robust error handling strategies in your JDBC code.

Author(s)

Upkar Lidder