# CRUD in MySQL using JDBC

Estimated time: **45 minutes**

## Overview

In this lab, you will learn:

- **How to perform CRUD (Create, Read, Update, Delete) operations using JDBC**
- **Differences between Statement, PreparedStatement, and CallableStatement**
- **How to implement pagination and handle errors effectively**
- **Best practices for secure and efficient CRUD operations**

By the end of this lab, you will be able to perform CRUD operations securely using JDBC.

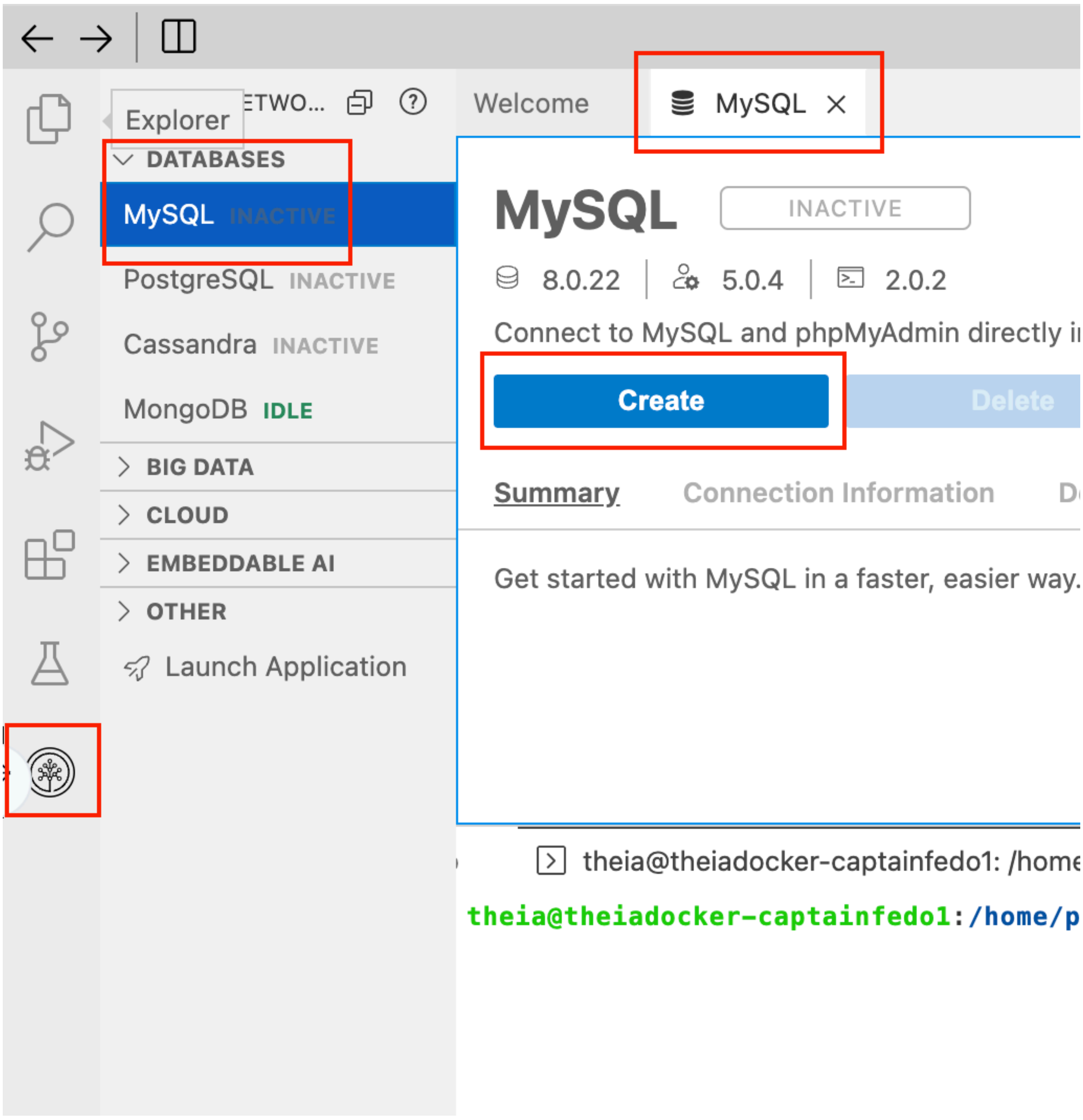
## Learning Objectives

After completing this lab, you will be able to:

- Insert, retrieve, update, and delete data in a relational database using JDBC
- Use **PreparedStatement** for secure queries
- Execute **stored procedures** using CallableStatement
- Implement **pagination** for large datasets

# Start 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.

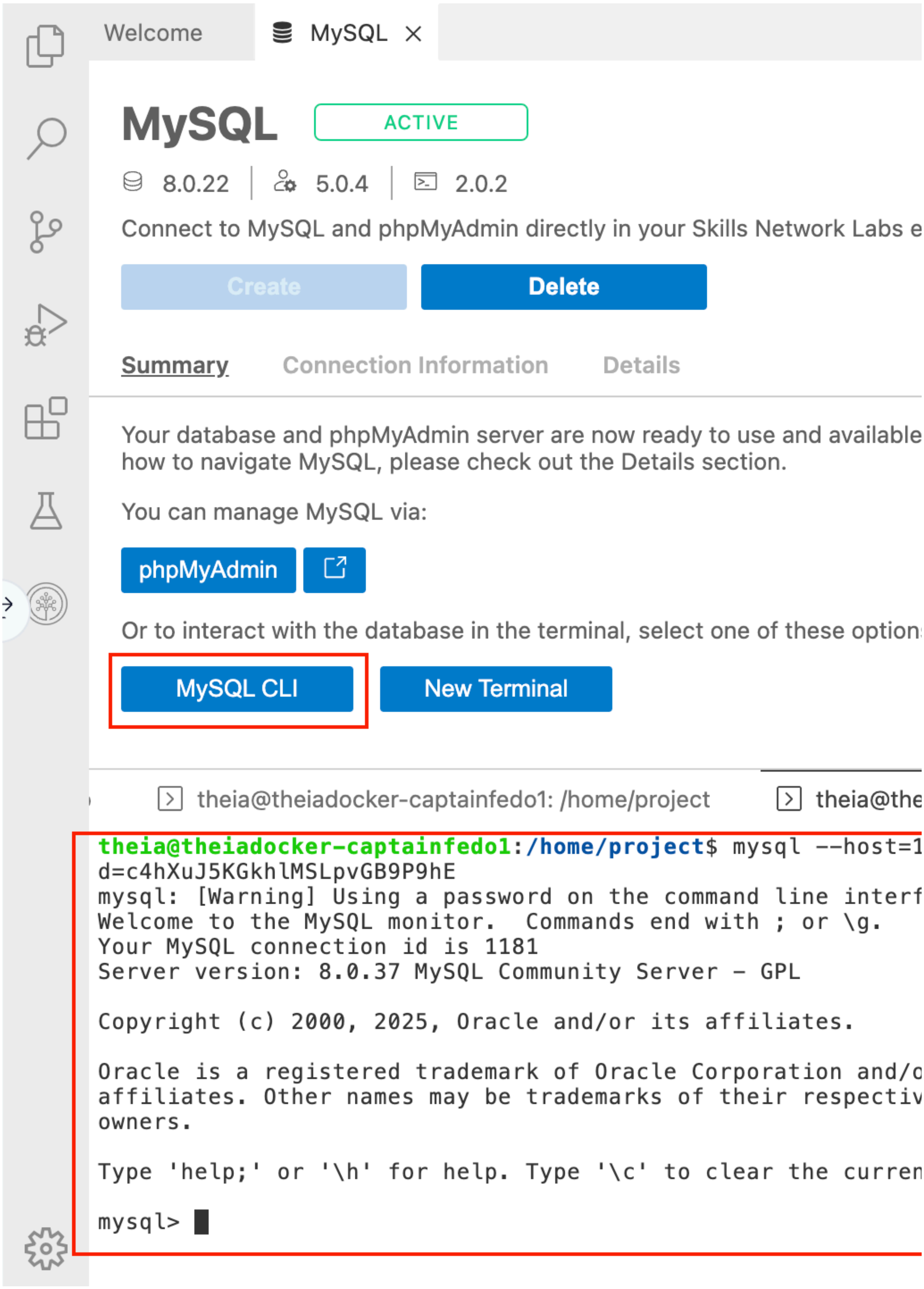


Connect to MySQL and phpMyAdmin directly i

Get started with MySQL in a faster, easier way.

```
theia@theiadocker-captainfedo1: /home
theia@theiadocker-captainfedo1:/home/p
```

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.



Connect to MySQL and phpMyAdmin directly in your Skills Network Labs e

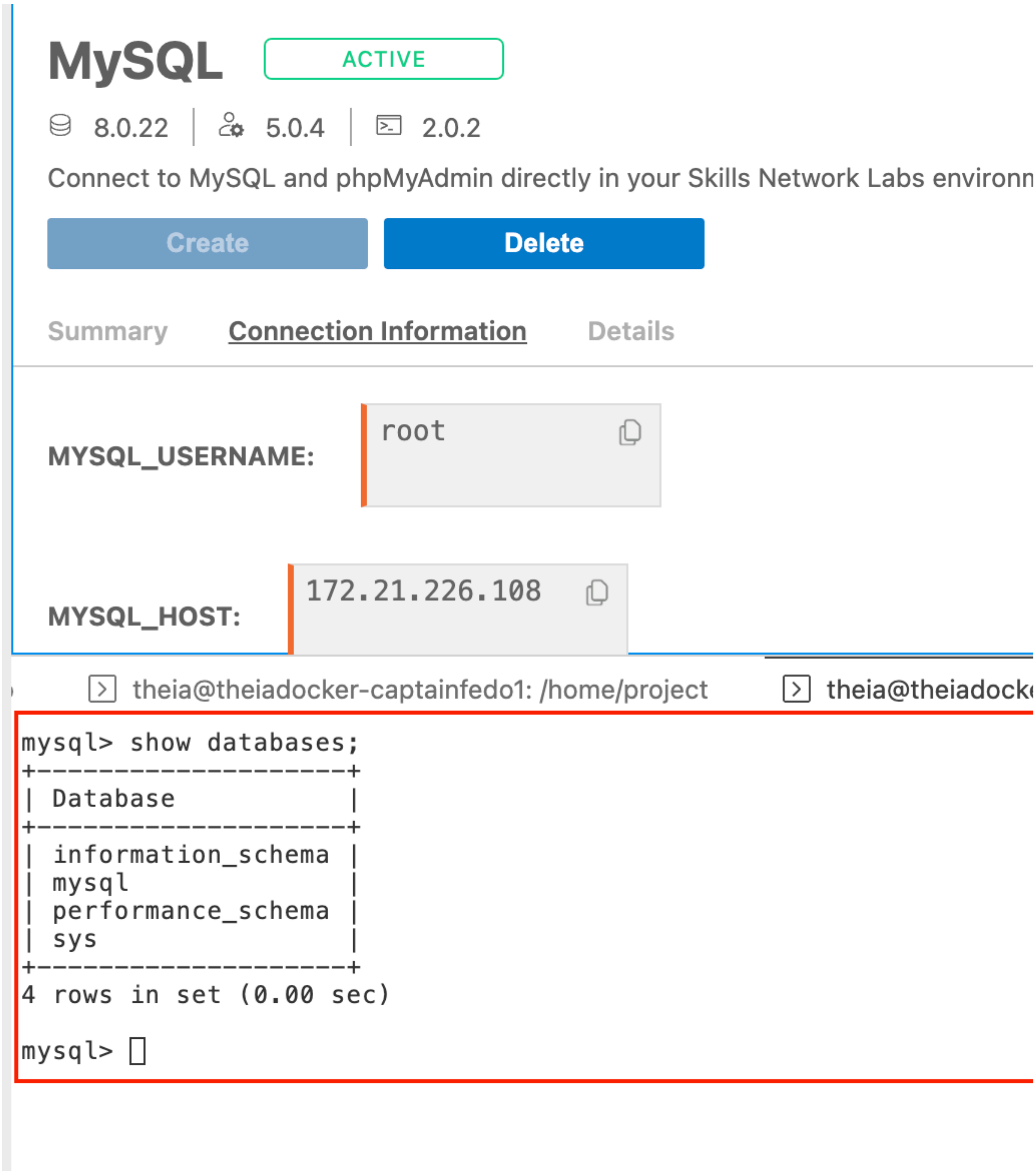
Your database and phpMyAdmin server are now ready to use and available
how to navigate MySQL, please check out the Details section.

You can manage MySQL via:

Or to interact with the database in the terminal, select one of these option

```
theia@theiadocker-captainfedo1:/home/project$ mysql --host=1
d=c4hXuJ5KGkhlMSLpvGB9P9hE
mysql: [Warning] Using a password on the command line interf
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1181
Server version: 8.0.37 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/o
affiliates. Other names may be trademarks of their respectiv
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the curren

mysql> 
```

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



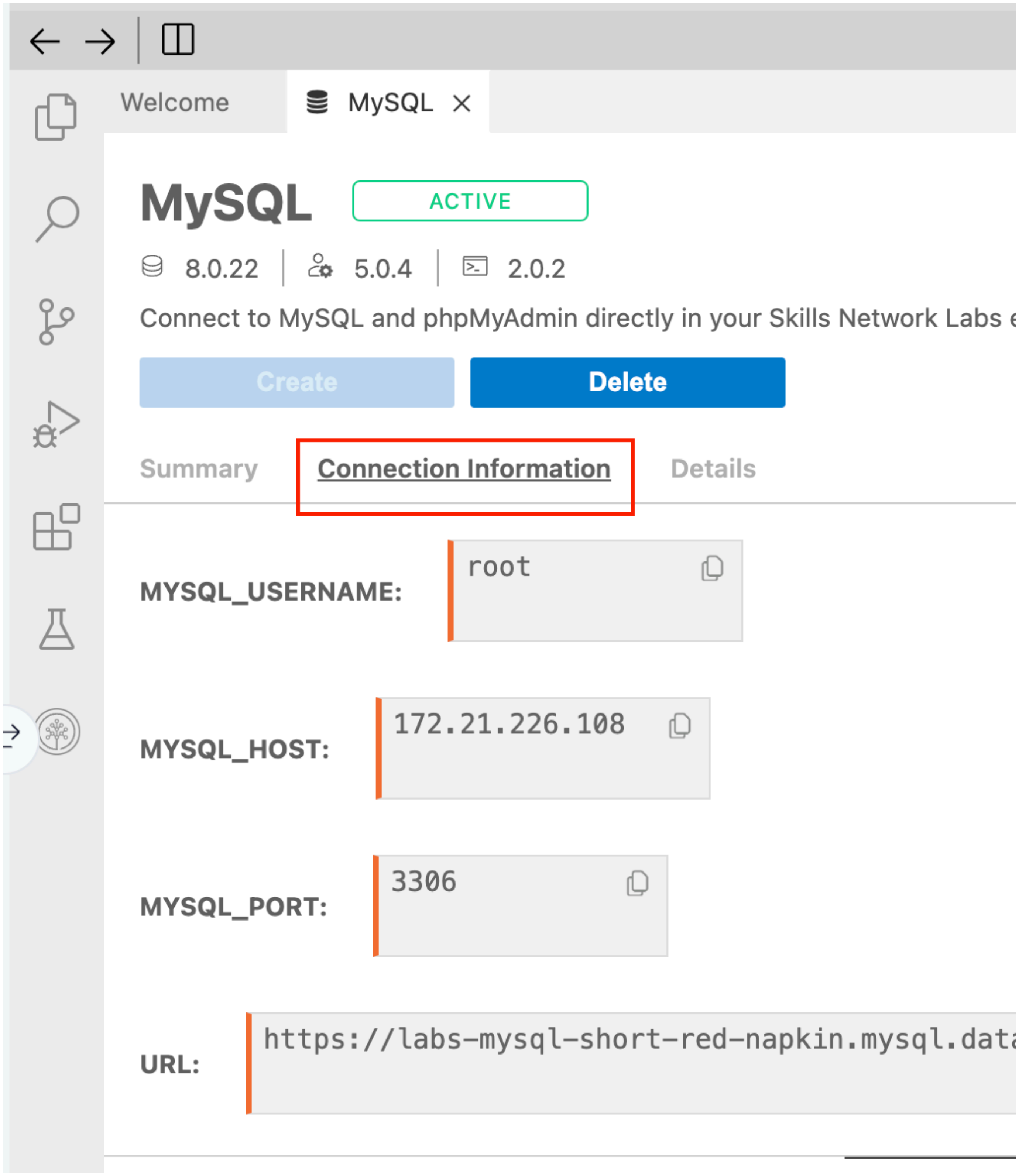
MySQL
ACTIVE
8.0.22
5.0.4
2.0.2
Connect to MySQL and phpMyAdmin directly in your Skills Network Labs environn
Create
Delete
Summary
Connection Information
Details
theia@theiadocker-captainfedo1: /home/project
theia@theiadock

MYSQL_USERNAME: root

MYSQL_HOST: 172.21.226.108

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)

mysql> 
```

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.



Connect to MySQL and phpMyAdmin directly in your Skills Network Labs e

MYSQL_USERNAME: root

MYSQL_HOST: 172.21.226.108

MYSQL_PORT: 3306

URL: https://labs-mysql-short-red-napkin.mysql.data

## Setting up the Tables

Open the terminal using the `MySQL CLI` button. You will need to create three tables for this lab: `Courses`, `Students`, and `Enrollments` in this order for the dependencies to work.

### Creating a database

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

```
CREATE DATABASE bookstore;
```

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

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

Next, use the database for future commands.

```
USE bookstore;
```

### Creating the books table

Now, create the books table using the following SQL command:

```
CREATE TABLE books (
    BookID INT PRIMARY KEY AUTO_INCREMENT,
    Title VARCHAR(255) NOT NULL,
    Author VARCHAR(255) NOT NULL,
    Price DOUBLE NOT NULL,
    Genre VARCHAR(100),
    PublicationDate DATE,
    PublisherID INT
);
```

- BookID: Unique identifier for each book.
- Title: Title of the book (required).
- Author: Author's name (required).
- Price: Book price (required).
- Genre: Book genre (optional).
- PublicationDate: Date of publication (optional).
- PublisherID: Reference to the publisher (optional).

Insert sample data into the books table using the following SQL statements:

```
INSERT INTO books (Title, Author, Price, Genre, PublicationDate, PublisherID) VALUES
('Advanced Java', 'Jane Doe', 45.99, 'Programming', '2023-01-10', 1),
('Database Systems', 'John Smith', 39.99, 'Computer Science', '2022-05-15', 2),
('Artificial Intelligence', 'Emily Johnson', 49.99, 'Computer Science', '2023-03-12', 3),
('Calculus Made Easy', 'Isaac Newton', 29.99, 'Mathematics', '2021-11-23', NULL),
('Modern Literature', 'William Blake', 19.99, 'Literature', '2020-09-05', NULL);
```

# Connecting MySQL with Java Using VS Code

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

### Step 1: Open Terminal in the IDE

Navigate to the project directory using the regular non SQL terminal. If you don't have one open, use the `Terminal` -> `New Terminal` to create a new terminal.

```
cd /home/project
```

## Step 2: Create the Maven project

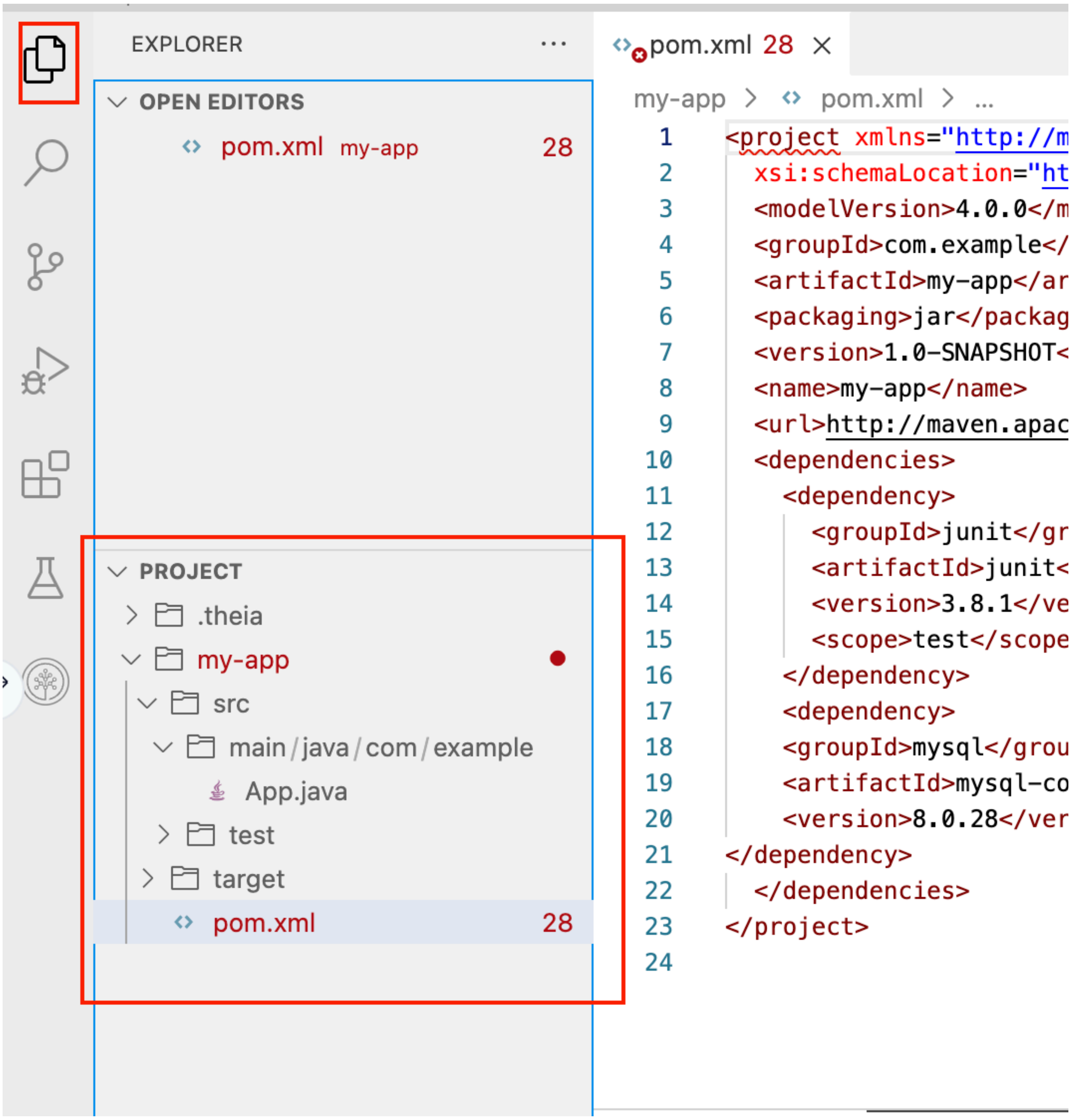
```
mvn archetype:generate \
    -DgroupId=com.example \
    -DartifactId=my-app \
    -Dversion=1.0-SNAPSHOT \
    -Dpackage=com.example \
    -DinteractiveMode=false
```

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. Click the **Refresh Explorer** icon if you cannot see it yet.



```
<project xmlns="http://m
  xsi:schemaLocation="ht
  <modelVersion>4.0.0</m
  <groupId>com.example</
  <artifactId>my-app</ar
  <packaging>jar</packag
  <version>1.0-SNAPSHOT<
  <name>my-app</name>
  <url>http://maven.apac
  <dependencies>
    <dependency>
      <groupId>junit</gr
      <artifactId>junit<
      <version>3.8.1</ve
      <scope>test</scope
    </dependency>
    <dependency>
    <groupId>mysql</grou
    <artifactId>mysql-co
    <version>8.0.28</ver
</dependency>
  </dependencies>
</project>
```

## Step 3: Add MySQL Connector Dependency

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

Open **pom.xml** in IDE

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 under 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

Make sure you are in the `my-app` directory in the terminal 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 existing code with the following code inside the `App.java`file. The code has a `main` method with the connection code.

```
package com.example;
import java.sql.*;
public class App {
    public static void main(String[] args) {
        String url = "jdbc:mysql://{MYSQL_HOST}:3306/bookstore"; // Change to your database
        String user = "{MYSQL_USERNAME}"; // Your database username
        String password = "{MYSQL_PASSWORD}"; // Your database password
        try (Connection conn = DriverManager.getConnection(url, user, password)) {
            System.out.println("Connection successful!");
            insertBook(conn);
            readAllBooks(conn);
            readBooksByGenre(conn);
            paginateBooks(conn);
            updateBookPrice(conn);
            deleteBooksByGenre(conn);
            callAddBookProcedure(conn);
            simulateSQLError(conn);
            System.out.println("------------------------");
            System.out.println("Finish!");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public static void insertBook(Connection conn) throws SQLException {
        System.out.println("------------------------");
        System.out.println("Insert data into the books table");
    }
    public static void readAllBooks(Connection conn) throws SQLException {
        System.out.println("------------------------");
        System.out.println("Read all books");
    }
    public static void readBooksByGenre(Connection conn) throws SQLException {
        System.out.println("------------------------");
        System.out.println("Read books by genre");
    }
    public static void paginateBooks(Connection conn) throws SQLException {
        System.out.println("------------------------");
        System.out.println("Paginate books by price");
    }
    public static void updateBookPrice(Connection conn) throws SQLException {
        System.out.println("------------------------");
        System.out.println("Update book price");
    }
    public static void deleteBooksByGenre(Connection conn) throws SQLException {
        System.out.println("------------------------");
        System.out.println("Delete books by genre");
    }
    public static void callAddBookProcedure(Connection conn) throws SQLException {
        System.out.println("------------------------");
        System.out.println("Calling stored procedure AddBook");
    }
    public static void simulateSQLError(Connection conn) {
        System.out.println("------------------------");
        System.out.println("Simulating SQL Error");
    }
}
```

To establish the connection, you need to replace the `{MYSQL_HOST}`, `{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 this lab.

- Replace `{MYSQL_HOST}` with the MYSQL_HOST value in the MySQL connection information
- Replace `{MYSQL_USERNAME}`, and `{MYSQL_PASSWORD}` as per your MySQL database credentials.

Run the code with the following command in the terminal:

```
cd /home/project/my-app && mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

You should see an output similar to the following:

```
...
...
[INFO] --- exec:3.5.0:java (default-cli) @ my-app ---
Connection successful!
------------------------
Insert data into the books table
------------------------
Read all books
------------------------
Read books by genre
------------------------
Paginate books by price
------------------------
Update book price
------------------------
Delete books by genre
------------------------
Calling stored procedure AddBook
------------------------
Simulating SQL Error
------------------------
...
...
```

# Create Operation (Insert Data)

### Insert a New Book Using PreparedStatement

The `insertBook` method inserts a new book record into the `books` table using a secure, parameterized query.

Copy the code below into the open `App.java` file to replace the existing `insertBook` method in the file.

```
public static void insertBook(Connection conn) throws SQLException {
    System.out.println("------------------------");
    System.out.println("Inserting a book using PreparedStatement");
    try (PreparedStatement ps = conn.prepareStatement(
            "INSERT INTO books (Title, Author, Price, Genre, PublicationDate, PublisherID) VALUES (?, ?, ?, ?, ?, ?)")) {
        ps.setString(1, "Advanced Java");
        ps.setString(2, "Jane Doe");
        ps.setDouble(3, 45.99);
        ps.setString(4, "Programming");
        ps.setDate(5, java.sql.Date.valueOf("2023-01-10"));
        ps.setInt(6, 1);
        int rows = ps.executeUpdate();
        System.out.println("Rows inserted: " + rows);
    }
}
```

### Explanation

- Uses PreparedStatement for safe parameter binding
- Avoids SQL injection by separating data from SQL
- Inserts one book with hardcoded details
- executeUpdate() returns number of affected rows

Run the code with the following command in the terminal:

```
cd /home/project/my-app && mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

You should see an output similar to the following:

```
...
[INFO] --- exec:3.5.0:java (default-cli) @ my-app ---
Connection successful!
------------------------
Inserting a book using PreparedStatement
Rows inserted: 1
------------------------
...
```

# Read Operation (Retrieve Data)

The `readAllBooks` method uses a regular Statement to fetch and display all books from the table.

Copy the code below into the open `App.java` file to replace the existing `readAllBooks` method in the file.

### Retrieve All Books

```
public static void readAllBooks(Connection conn) throws SQLException {
    System.out.println("------------------------");
    System.out.println("Retrieving all books");
    try (Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery("SELECT * FROM books")) {
        while (rs.next()) {
            System.out.println("Title: " + rs.getString("Title"));
            System.out.println("Author: " + rs.getString("Author"));
            System.out.println("Price: " + rs.getDouble("Price"));
        }
    }
}
```

### Explanation

- Uses Statement to execute a simple SQL query
- Loops through results with ResultSet.next()
- Retrieves values using column names

Run the code with the following command in the terminal:

```
cd /home/project/my-app && mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

You should see an output similar to the following:

```
...
------------------------
Inserting a book using PreparedStatement
Rows inserted: 1
------------------------
Retrieving all books
Title: Advanced Java
Author: Jane Doe
Price: 45.99
Title: Database Systems
Author: John Smith
Price: 39.99
Title: Artificial Intelligence
Author: Emily Johnson
Price: 49.99
Title: Calculus Made Easy
Author: Isaac Newton
Price: 29.99
Title: Modern Literature
Author: William Blake
Price: 19.99
Title: Advanced Java
Author: Jane Doe
Price: 45.99
Title: Advanced Java
Author: Jane Doe
Price: 45.99
------------------------
...
```

## Retrieve Books by Genre Using PreparedStatement

The `readBooksByGenre` method uses a PreparedStatement to retrieve only books of a specific genre; in this case, "Programming".

Copy the code below into the open `App.java` file to replace the existing `readBooksByGenre` method in the file.

```java
public static void readBooksByGenre(Connection conn) throws SQLException {
    System.out.println("------------------------");
    System.out.println("Retrieving books by genre: Programming");
    try (PreparedStatement ps = conn.prepareStatement(
            "SELECT * FROM books WHERE Genre = ?")) {
        ps.setString(1, "Programming");
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            System.out.println("Title: " + rs.getString("Title"));
        }
    }
}
```

## Explanation

- Demonstrates parameterized SELECT query
- Filters rows based on Genre column
- Prints book titles that match the specified genre

Run the code with the following command in the terminal:

```
cd /home/project/my-app && mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

You should see an output similar to the following:

```
...
-----------------------
Retrieving books by genre: Programming
Title: Advanced Java
Title: Advanced Java
Title: Advanced Java
Title: Advanced Java
-----------------------
...
```

# Implementing Pagination

The `paginateBooks` method uses SQL pagination to retrieve the top 5 most expensive books.

Copy the code below into the open `App.java` file to replace the existing `paginateBooks` method in the file.

### Retrieve Books with Pagination

```
public static void paginateBooks(Connection conn) throws SQLException {
    System.out.println("-----------------------");
    System.out.println("Paginating books by price (LIMIT 5 OFFSET 0)");
    try (PreparedStatement ps = conn.prepareStatement(
            "SELECT * FROM books ORDER BY Price DESC LIMIT 5 OFFSET 0")) {
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            System.out.println("Title: " + rs.getString("Title"));
            System.out.println("Price: " + rs.getDouble("Price"));
        }
    }
}
```

### Explanation

- Retrieves books sorted by price in descending order
- Uses LIMIT and OFFSET to paginate results
- Shows only the top 5 books

Run the code with the following command in the terminal:

```
cd /home/project/my-app && mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

You should see an output similar to the following:

```
...
-----------------------
Paginating books by price (LIMIT 5 OFFSET 0)
Title: Artificial Intelligence
Price: 49.99
Title: Advanced Java
Price: 45.99
Title: Advanced Java
Price: 45.99
Title: Advanced Java
Price: 45.99
Title: Advanced Java
```

```
Price: 45.99
------------------------
...
```

## Update Operation

The `updateBookPrice` method updates the price of a book based on its title.

Copy the code below into the open `App.java` file to replace the existing `updateBookPrice` method in the file.

### Update Book Price

```
public static void updateBookPrice(Connection conn) throws SQLException {
    System.out.println("------------------------");
    System.out.println("Updating book price");
    try (PreparedStatement ps = conn.prepareStatement(
            "UPDATE books SET Price = ? WHERE Title = ?")) {
        ps.setDouble(1, 49.99);
        ps.setString(2, "Advanced Java");
        int rows = ps.executeUpdate();
        System.out.println("Rows updated: " + rows);
    }
}
```

### Explanation

- Prepares an UPDATE SQL statement with parameters
- Changes the price for books matching the title
- Shows number of rows that were updated

Run the code with the following command in the terminal:

```
cd /home/project/my-app && mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

You should see an output similar to the following:

```
...
------------------------
Updating book price
Rows updated: 6
------------------------
...
```

## Delete Operation

The `deleteBooksByGenre` method deletes books from the database based on the specified genre.

Copy the code below into the open `App.java` file to replace the existing `deleteBooksByGenre` method in the file.

### Delete Books by Genre

```
public static void deleteBooksByGenre(Connection conn) throws SQLException {
    System.out.println("------------------------");
    System.out.println("Deleting books in genre: Programming");
    try (PreparedStatement ps = conn.prepareStatement(
            "DELETE FROM books WHERE Genre = ?")) {
        ps.setString(1, "Programming");
        int rows = ps.executeUpdate();
        System.out.println("Rows deleted: " + rows);
    }
}
```

### Explanation

- Demonstrates DELETE with condition
- Accepts genre as a parameter to limit deletion
- Logs how many records were removed

Run the code with the following command in the terminal:

```
cd /home/project/my-app && mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

You should see an output similar to the following:

```
...
------------------------
Deleting books in genre: Programming
Rows deleted: 7
------------------------
...
```

# Using CallableStatement for Stored Procedures

### Create and Call a Stored Procedure to Insert a Book

Run the following code in the **MySQL CLI** terminal. You may have to open a new one from the MySQL database tab as described in the beginning of the lab.

```
USE bookstore;
```

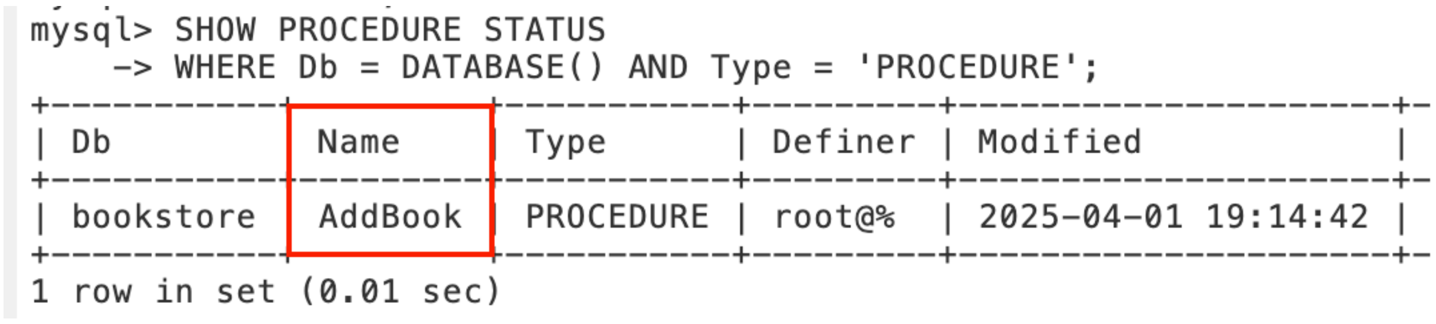
Next, run the following code to create the stored procedure:

```
DELIMITER $$
CREATE PROCEDURE AddBook(
    IN bookTitle VARCHAR(255),
    IN bookAuthor VARCHAR(255),
    IN bookPrice DOUBLE,
    IN bookGenre VARCHAR(100),
    IN publicationDate DATE,
    IN publisherID INT
)
BEGIN
```

```
    INSERT INTO books (Title, Author, Price, Genre, PublicationDate, PublisherID)
    VALUES (bookTitle, bookAuthor, bookPrice, bookGenre, publicationDate, publisherID);
END$$
DELIMITER ;
```

You can use the following code to ensure the procedure was created.

```
SHOW PROCEDURE STATUS
WHERE Db = DATABASE() AND Type = 'PROCEDURE';
```



```
mysql> SHOW PROCEDURE STATUS
    -> WHERE Db = DATABASE() AND Type = 'PROCEDURE';
+-----------+---------+-----------+---------+---------------------+-
| Db        | Name    | Type      | Definer | Modified            |
+-----------+---------+-----------+---------+---------------------+-
| bookstore | AddBook | PROCEDURE | root@%  | 2025-04-01 19:14:42 |
+-----------+---------+-----------+---------+---------------------+-
1 row in set (0.01 sec)
```

The `callAddBookProcedure` method calls a stored procedure named **AddBook** that inserts a book record.

Copy the code below into the open `App.java` file to replace the existing `callAddBookProcedure` method in the file.

**Call AddBook Procedure**

```
public static void callAddBookProcedure(Connection conn) throws SQLException {
    System.out.println("-----------------------");
    System.out.println("Calling stored procedure AddBook");
    try (CallableStatement cs = conn.prepareCall("{CALL AddBook(?, ?, ?, ?, ?, ?)}")) {
        cs.setString(1, "Advanced Java");
        cs.setString(2, "Jane Doe");
        cs.setDouble(3, 45.99);
        cs.setString(4, "Programming");
        cs.setDate(5, java.sql.Date.valueOf("2023-01-10"));
        cs.setInt(6, 1);
        cs.execute();
        System.out.println("Book added successfully using stored procedure.");
    }
}
```

## Explanation

- Uses CallableStatement to invoke stored procedure
- Passes input parameters using .setXXX() methods
- Calls the AddBook procedure and logs success

Switch back to the standard terminal (**not the MySQL CLI one**) and run the following command in the terminal:

```
cd /home/project/my-app && mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

You should see an output similar to the following:

```
...
------------------------
Calling stored procedure AddBook
Book added successfully using stored procedure.
------------------------
...
```

# Error Handling Best Practices

The `simulateSQLError` method intentionally queries a non-existent table to demonstrate SQL error handling.

Copy the code below into the open `App.java` file to replace the existing `simulateSQLError` method in the file.

### Handling SQL Exceptions

```
public static void simulateSQLError(Connection conn) {
    System.out.println("------------------------");
    System.out.println("Simulating SQL Error");
    try (PreparedStatement ps = conn.prepareStatement("SELECT * FROM NonExistentTable")) {
        ps.executeQuery();
    } catch (SQLException e) {
        System.err.println("SQL Error: " + e.getMessage());
        e.printStackTrace();
    }
}
```

### Explanation

- Wraps query in a try-catch block
- Prints custom error message and stack trace
- Demonstrates safe handling of SQL exceptions

Run the code with the following command in the terminal:

```
cd /home/project/my-app && mvn clean install && mvn exec:java -Dexec.mainClass="com.example.App"
```

You should see an output similar to the following:

```
...
------------------------
Simulating SQL Error
SQL Error: Table 'bookstore.NonExistentTable' doesn't exist
java.sql.SQLSyntaxErrorException: Table 'bookstore.NonExistentTable' doesn't exist
        at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:120)
        at com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLExceptionsMapping.java:122)
        at com.mysql.cj.jdbc.ClientPreparedStatement.executeInternal(ClientPreparedStatement.java:953)
        at com.mysql.cj.jdbc.ClientPreparedStatement.executeQuery(ClientPreparedStatement.java:1009)
        at com.example.App.simulateSQLError(App.java:141)
        at com.example.App.main(App.java:21)
        at org.codehaus.mojo.exec.ExecJavaMojo.doMain(ExecJavaMojo.java:375)
        at org.codehaus.mojo.exec.ExecJavaMojo.doExec(ExecJavaMojo.java:364)
        at org.codehaus.mojo.exec.ExecJavaMojo.lambda$execute$0(ExecJavaMojo.java:286)
        at java.base/java.lang.Thread.run(Thread.java:1583)
------------------------
...
```

## Conclusion and Next Steps

Congratulations! You have successfully learned how to perform CRUD operations using JDBC, demonstrating your ability to create, read, update, and delete data within a MySQL database from a Java application. You've also explored pagination, stored procedures, and error handling, all crucial skills for effective database interaction.

This lab provided a practical understanding of how to use PreparedStatement and CallableStatement for secure and efficient database operations, as well as how to manage data retrieval and manipulation.

### Next Steps:

1. Implement More Complex CRUD Operations: Practice creating and executing more complex CRUD operations that involve multiple tables and relationships. For example, try implementing operations that manage both Books and Publishers data, ensuring data integrity across related tables.
2. Enhance Pagination with Dynamic Page Size and Navigation: Extend the pagination implementation to allow users to dynamically change the page size and navigate between pages using previous and next buttons. Implement error handling to prevent out-of-bounds page requests.
3. Create and Use Stored Procedures for Data Validation and Business Logic: Develop stored procedures that enforce data validation rules and implement business logic. For instance, create a stored procedure that checks if a book title already exists before inserting a new book, or a procedure that updates the stock quantity of a book after an order is placed.

## Author(s)

Upkar Lidder