Integrating MySQL with Java

CLO: 2

Objectives

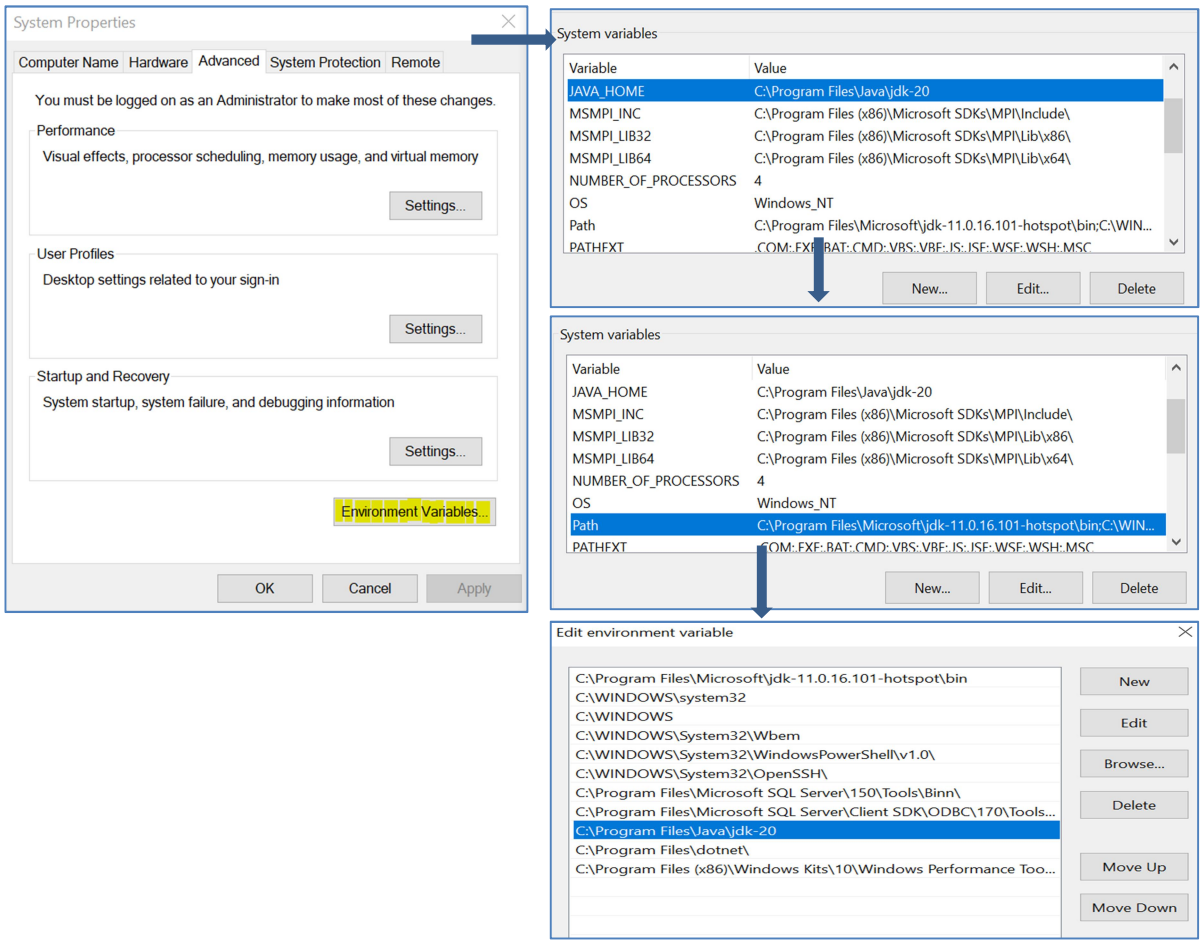
- Integrate **MySQL** with **Java**
- Perform **SQL queries** using code

This manual provides step-by-step instructions for integrating **MySQL** with **Java** to perform SQL operations programmatically. It covers environment setup, establishing a database connection, and executing SQL queries like SELECT, INSERT, UPDATE, and DELETE using **JDBC** (Java Database Connectivity).

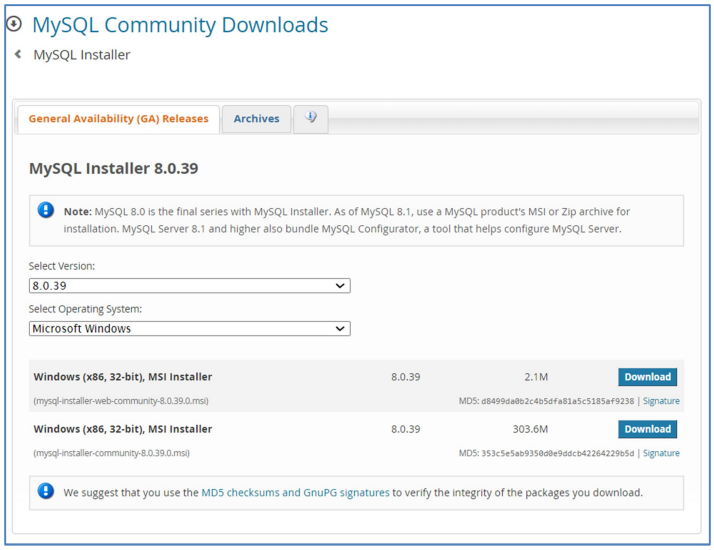
Installation and Configuration

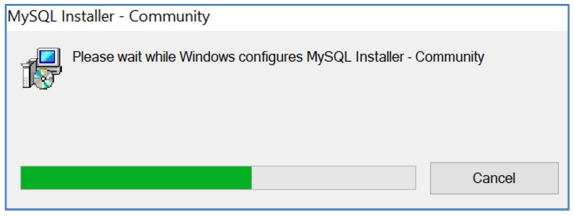
Ensure the following software is installed on your system before proceeding:

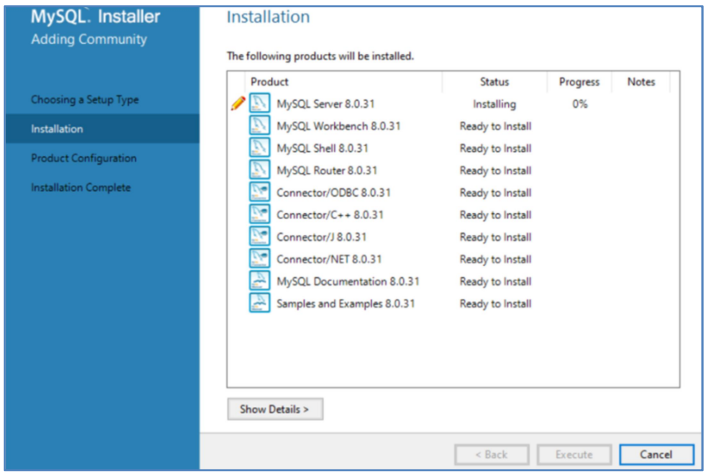
1. **Java Development Kit (JDK)**:
   - Download from Oracle JDK.
   - Make sure to set your JAVA_HOME environment variable.



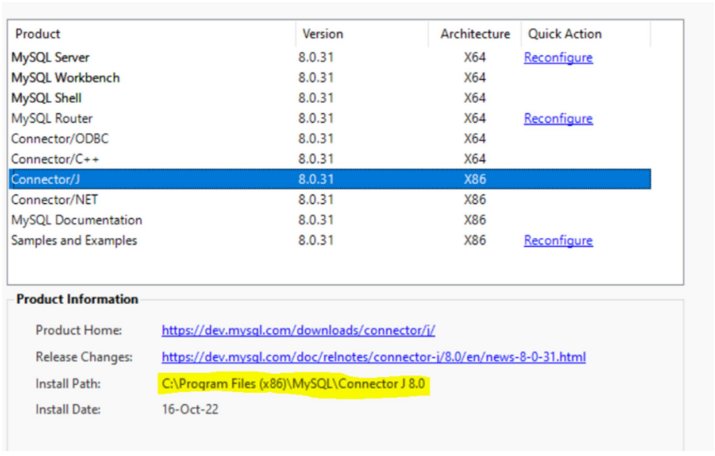
2. **MySQL**:
    - Download and install from MySQL Downloads.



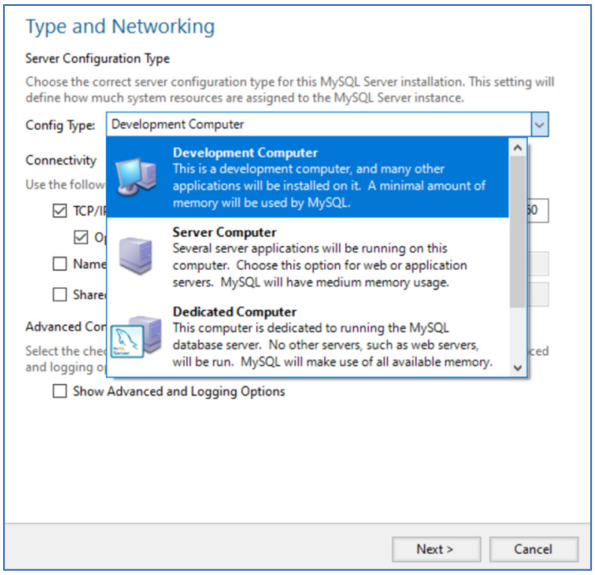


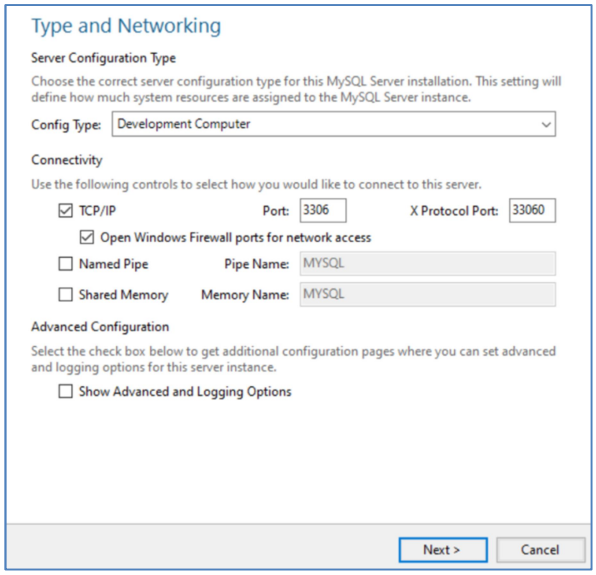


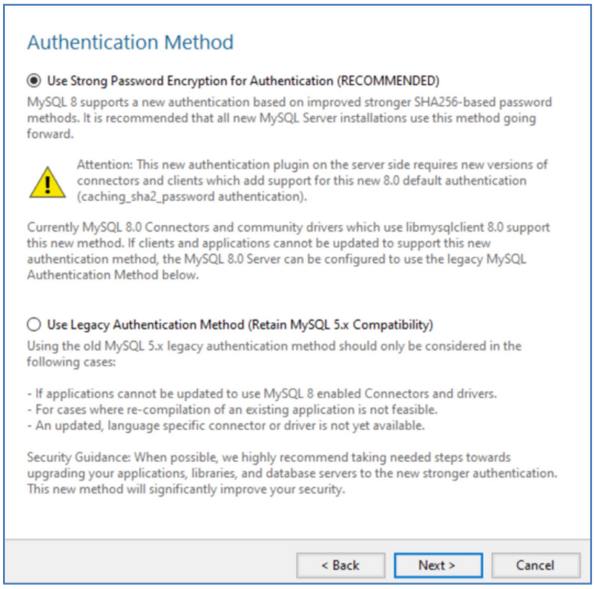
3. **MySQL Connector/J**:
    - The JDBC driver required for Java MySQL connectivity.

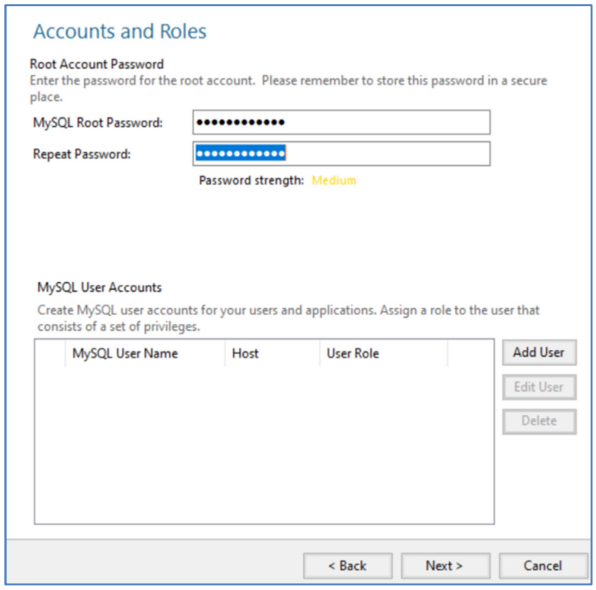


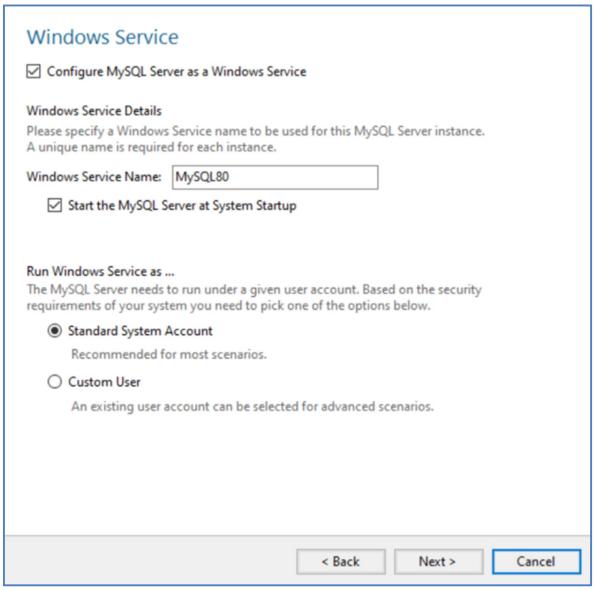
4. DB Configuration

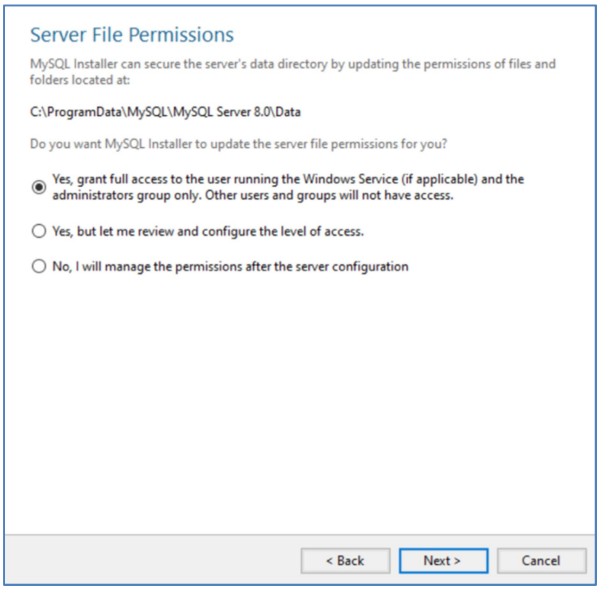












Logging Options

Please select the logs you want to activate for this server in addition to the error log.
On production computers, it can be beneficial to separate the log files from the data.
Specify a file name to save the logs in the data directory (default) or browse to a different location.
You must provide an absolute path when specifying the path together with the file name.

Error Log: DESKTOP-8TQ5QNO.err

☑ General Log

The general query log is a general record of what the MySQL Server is doing.
It should only be used to track down issues.

File Path: DESKTOP-8TQ5QNO.log

☑ Slow Query Log

The slow query log consists of SQL statements that took more than the given value of seconds to execute. It is recommended to turn this log on.

File Path: DESKTOP-8TQ5QNO-slow.log Seconds: 10

☑ Binary Log

The binary log contains all database events and is used for replication and data recovery operations. Enabling the log has a performance impact on the server.
Enter the log name without a file extension.

File Path: DESKTOP-8TQ5QNO-bin

< Back | Next > | Cancel

Apply Configuration

Click [Execute] to apply the changes

Configuration Steps | Log

- ○ Writing configuration file
- ○ Updating Windows Firewall rules
- ○ Adjusting Windows service
- ○ Initializing database (may take a long time)
- ○ Updating permissions for the data folder and related server files
- ○ Starting the server
- ○ Applying security settings
- ○ Updating the Start menu link

< Back | Execute | Cancel

Apply Configuration

The configuration operation has completed.

Configuration Steps | Log

- ✓ Writing configuration file
- ✓ Updating Windows Firewall rules
- ✓ Adjusting Windows service
- ✓ Initializing database (may take a long time)
- ✓ Updating permissions for the data folder and related server files
- ✓ Starting the server
- ✓ Applying security settings
- ✓ Updating the Start menu link

The configuration for MySQL Server 8.0.31 was successful.
Click Finish to continue.

Finish

MySQL Router Configuration

☐ Bootstrap MySQL Router for use with InnoDB Cluster

This wizard can bootstrap MySQL Router to direct traffic between MySQL applications and InnoDB Cluster. Applications that connect to the router will be automatically directed to an available read/write or read-only member of the cluster.

The boostrapping process requires a connection to InnoDB Cluster. In order to register the MySQL Router for monitoring, use the current Read/Write instance of the cluster.

Hostname:
Port: 3306
Management User: root
Password: Test Connection

MySQL Router requires specification of a base port (between 80 and 65532). The first port is used for classic read/write connections. The other ports are computed sequentially after the first port. If any port is indicated to be in use, please change the base port.

Classic MySQL protocol connections to InnoDB Cluster:
Read/Write: 6446
Read Only: 6447

X Protocol connections to InnoDB Cluster:
Read/Write: 6448
Read Only: 6449

Finish | Cancel

Advanced Options

Server ID: 1

A unique numeric identifier used in a replication topology.
If binary logging is enabled, a Server ID must be specified.

Table Names Case:

- ◉ Lower Case (default):
  This option sets the configuration variable lower_case_table_names = 1.
- ○ Preserve Given Case:
  This option sets the configuration variable lower_case_table_names = 2.

< Back | Next > | Cancel

Installation Complete

The installation procedure has been completed.

Copy Log to Clipboard

☑ Start MySQL Workbench after setup
☑ Start MySQL Shell after setup

The MySQL Shell is an advanced MySQL client application that can be used to work with single MySQL Server instances. Further, it can be used to create and manage InnoDB Cluster, an integrated solution for high availability and scalability of MySQL databases, without requiring advanced MySQL expertise.

Refer to the following links for documentation, tutorials and examples on MySQL Shell:

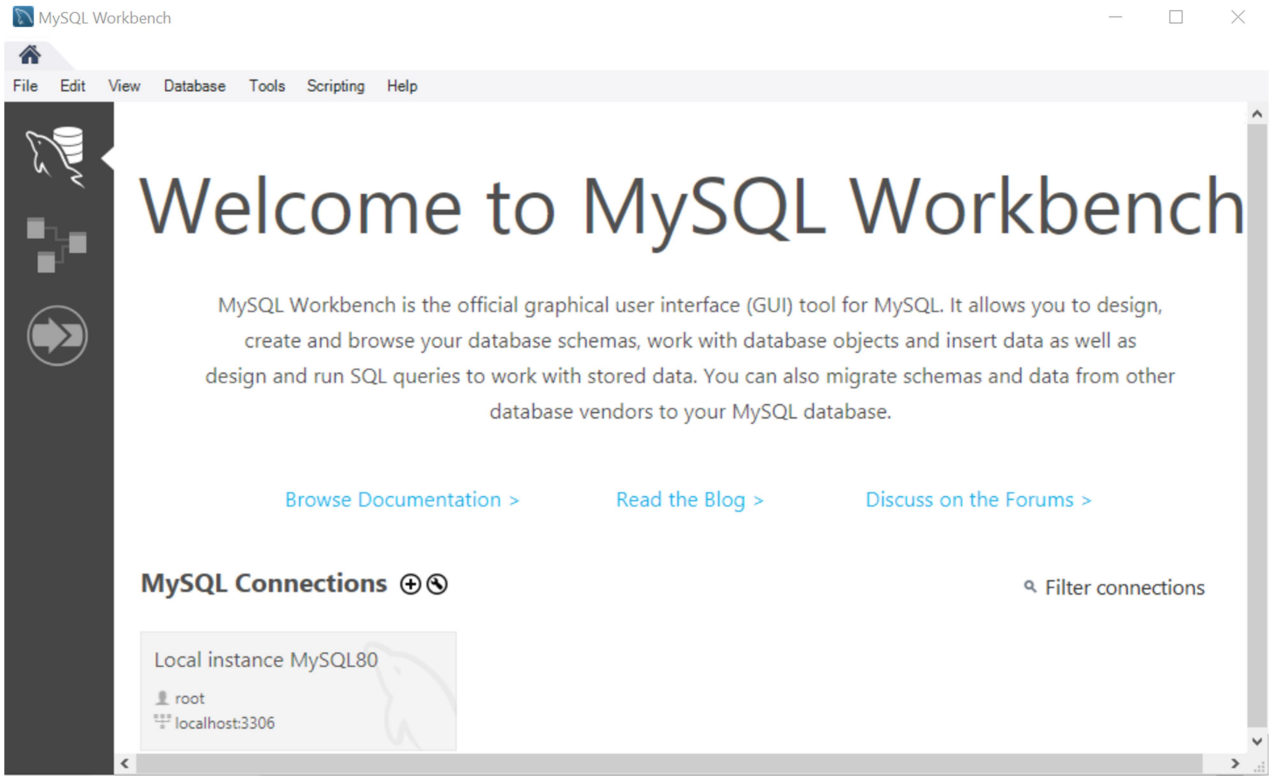
MySQL Shell Documentation
Setting up a Real World Cluster Blog
The All New MySQL InnoDB ReplicaSet Blog
Changing Cluster Options Live Blog

Finish

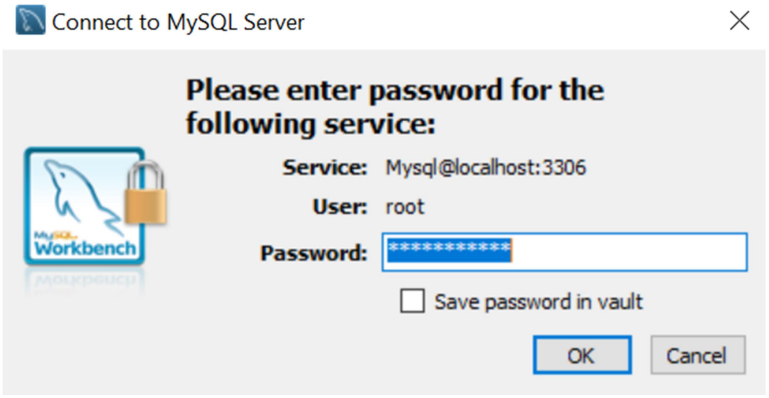
5. **IDE** (Integrated Development Environment):
    - Use an IDE like **Eclipse** or **IntelliJ IDEA** for Java development.

Setting up the MySQL Database

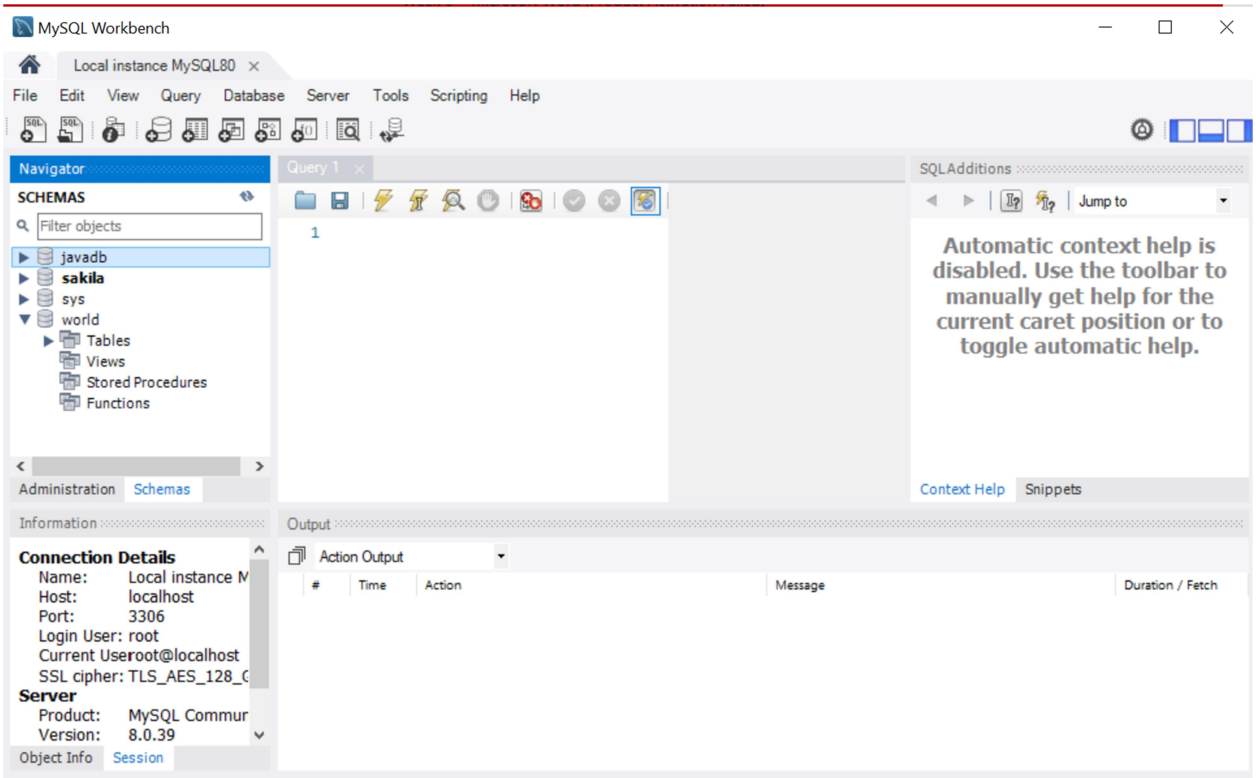
1. Open MySql Workbench



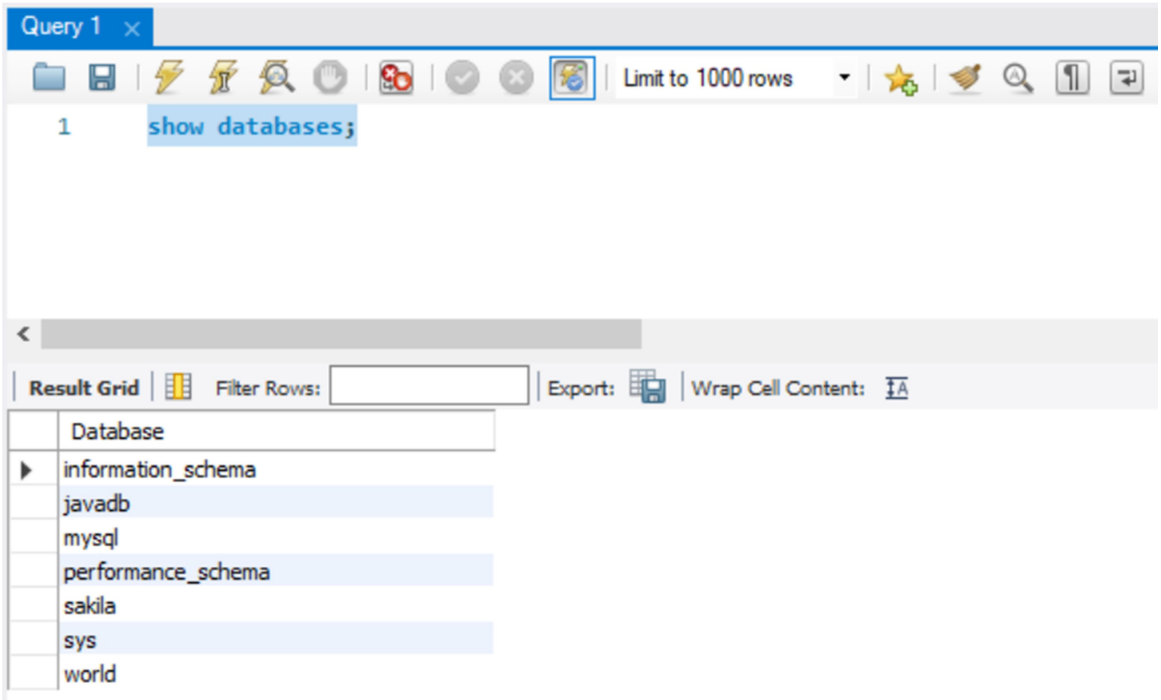
2. Create or connect to already existing MySQL Connection



3. Go-through MySQL workbench platform



4. Create SQL tab for executing queries or use the one created by default. Write SQL Command to see all the available Databases

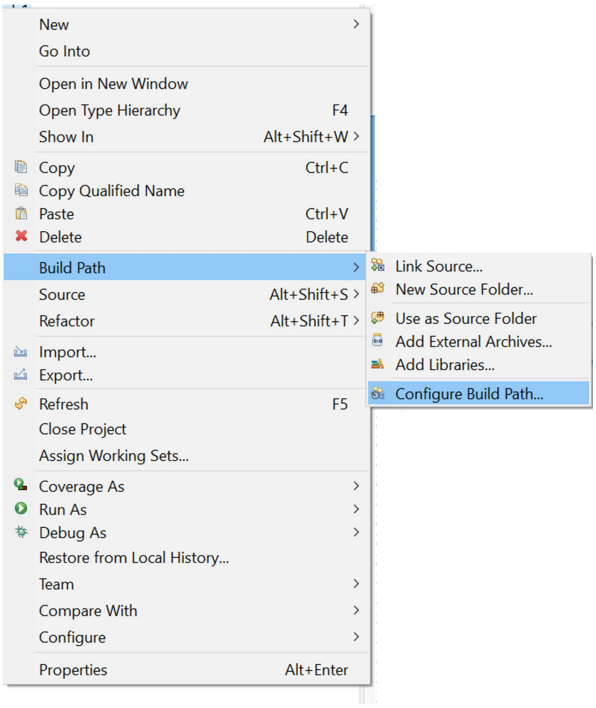


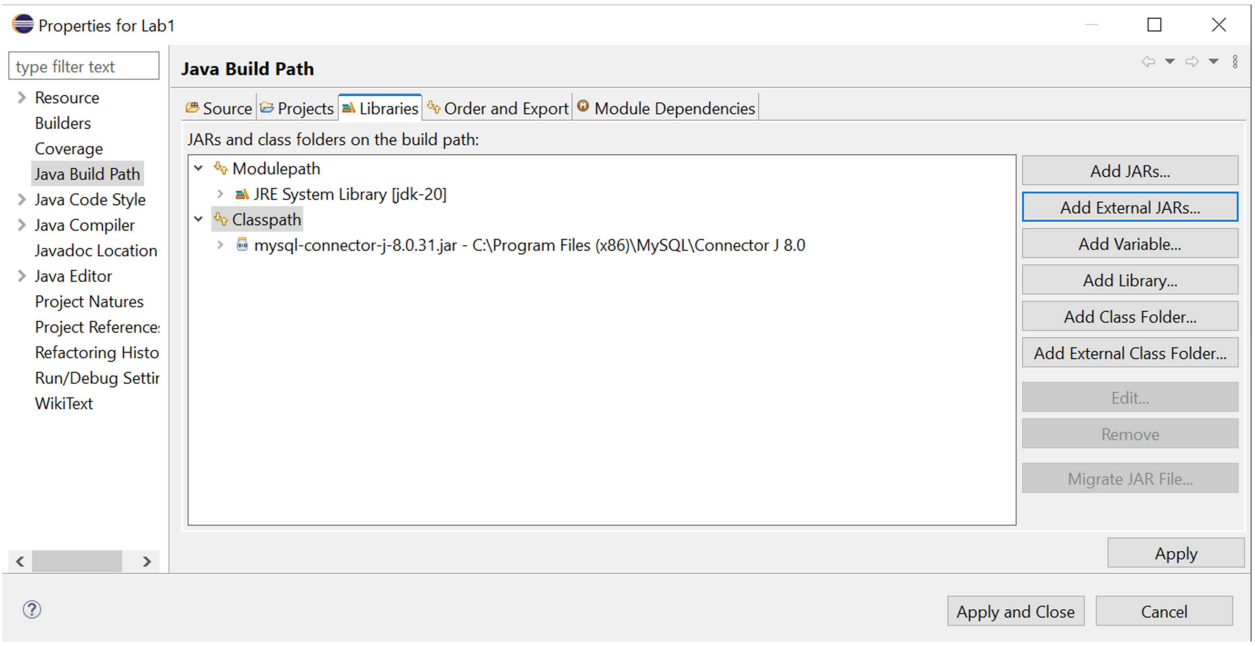
5. Practice executing all the SQL commands (especially mentioned on cheat sheet) using MySQL Workbench platform. (Weighted evaluation)

Setting up Java Project

1. **Add MySQL Connector to Project:**

   - In **Eclipse**: Right-click on your project → Build Path → Configure Build Path → Add External JARs → Select the MySQL Connector `.jar` file.
   - In **IntelliJ IDEA**: File → Project Structure → Libraries → Add Library → Select the MySQL Connector `.jar` file.





Database integration

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

1. Import JDBC Libraries
2. Create connection
3. Create statement
4. Execute queries
5. Close connection

```java
package com.java.practice.mysql;

import java.sql.Timestamp;
//Step 1: Import JDBC Libraries
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;

public class mysql_queries {
    // Database URL, User name, and Password
    private static final String DB_URL = "jdbc:mysql://localhost:3306/sakila";
    private static final String USER = "root";
    private static final String PASSWORD = "blackParrot3";

    public static void main(String[] args) {
        Connection connection = null;
        Statement statement = null;

        try {
            // Step 2: Create Connection

                // a. Register JDBC driver (automatically done in newer versions)
            Class.forName("com.mysql.cj.jdbc.Driver");

                // b. Open a connection
            System.out.println("Connecting to database...");
            connection = DriverManager.getConnection(DB_URL, USER, PASSWORD);
            System.out.println("Successfully connected to MySQL!");

            // Step 3:  Create statement
            statement = connection.createStatement();

            // Step 4:  Execute SQL SELECT query
            String sql = "SELECT * FROM actor";
            ResultSet resultSet = statement.executeQuery(sql);

            // Process the result set
            while (resultSet.next()) {
                int id = resultSet.getInt("actor_id");
                String first_name = resultSet.getString("first_name");
                String last_name = resultSet.getString("last_name");
                Timestamp last_update = resultSet.getTimestamp("last_update");

                System.out.println("ID: " + id + ", First Name: " + first_name +
                                   ", Last Name: " + last_name + ", Last Update: " + last_update);
            }
            resultSet.close();

        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            // Step 5:  Close connection
            try {
                if (connection != null) {
                    connection.close();
                }
            } catch (SQLException se) {
                se.printStackTrace();
            }
        }
    }

}
```

Output

```
Connecting to database...
Successfully connected to MySQL!
ID: 1, First Name: PENELOPE, Last Name: GUINESS, Last Update: 2006-02-15 04:34:33.0
ID: 2, First Name: NICK, Last Name: WAHLBERG, Last Update: 2006-02-15 04:34:33.0
ID: 3, First Name: ED, Last Name: CHASE, Last Update: 2006-02-15 04:34:33.0
ID: 4, First Name: JENNIFER, Last Name: DAVIS, Last Update: 2006-02-15 04:34:33.0
ID: 5, First Name: JOHNNY, Last Name: LOLLOBRIGIDA, Last Update: 2006-02-15 04:34:33.0
ID: 6, First Name: BETTE, Last Name: NICHOLSON, Last Update: 2006-02-15 04:34:33.0
ID: 7, First Name: GRACE, Last Name: MOSTEL, Last Update: 2006-02-15 04:34:33.0
ID: 8, First Name: MATTHEW, Last Name: JOHANSSON, Last Update: 2006-02-15 04:34:33.0
ID: 9, First Name: JOE, Last Name: SWANK, Last Update: 2006-02-15 04:34:33.0
ID: 10, First Name: CHRISTIAN, Last Name: GABLE, Last Update: 2006-02-15 04:34:33.0
ID: 11, First Name: ZERO, Last Name: CAGE, Last Update: 2006-02-15 04:34:33.0
ID: 12, First Name: KARL, Last Name: BERRY, Last Update: 2006-02-15 04:34:33.0
ID: 13, First Name: UMA, Last Name: WOOD, Last Update: 2006-02-15 04:34:33.0
ID: 14, First Name: VIVIEN, Last Name: BERGEN, Last Update: 2006-02-15 04:34:33.0
ID: 15, First Name: CUBA, Last Name: OLIVIER, Last Update: 2006-02-15 04:34:33.0
ID: 16, First Name: FRED, Last Name: COSTNER, Last Update: 2006-02-15 04:34:33.0
ID: 17, First Name: HELEN, Last Name: VOIGHT, Last Update: 2006-02-15 04:34:33.0
ID: 18, First Name: DAN, Last Name: TORN, Last Update: 2006-02-15 04:34:33.0
ID: 19, First Name: BOB, Last Name: FAWCETT, Last Update: 2006-02-15 04:34:33.0
ID: 20, First Name: LUCILLE, Last Name: TRACY, Last Update: 2006-02-15 04:34:33.0
ID: 21, First Name: KIRSTEN, Last Name: PALTROW, Last Update: 2006-02-15 04:34:33.0
ID: 22, First Name: ELVIS, Last Name: MARX, Last Update: 2006-02-15 04:34:33.0
ID: 23, First Name: SANDRA, Last Name: KILMER, Last Update: 2006-02-15 04:34:33.0
ID: 24, First Name: CAMERON, Last Name: STREEP, Last Update: 2006-02-15 04:34:33.0
ID: 25, First Name: KEVIN, Last Name: BLOOM, Last Update: 2006-02-15 04:34:33.0
ID: 26, First Name: RIP, Last Name: CRAWFORD, Last Update: 2006-02-15 04:34:33.0
ID: 27, First Name: JULIA, Last Name: MCQUEEN, Last Update: 2006-02-15 04:34:33.0
```

Best Practices

1. **Always Close Resources**: Use `finally` blocks (or Java 7+ try-with-resources) to close connections, statements, and result sets.
2. **Prepared Statements**: Use `PreparedStatement` instead of `Statement` to avoid SQL injection attacks.
3. **Error Handling**: Implement robust error handling for database connectivity issues.
4. **Transactions**: Use transactions for atomic operations (group multiple queries in one logical unit).