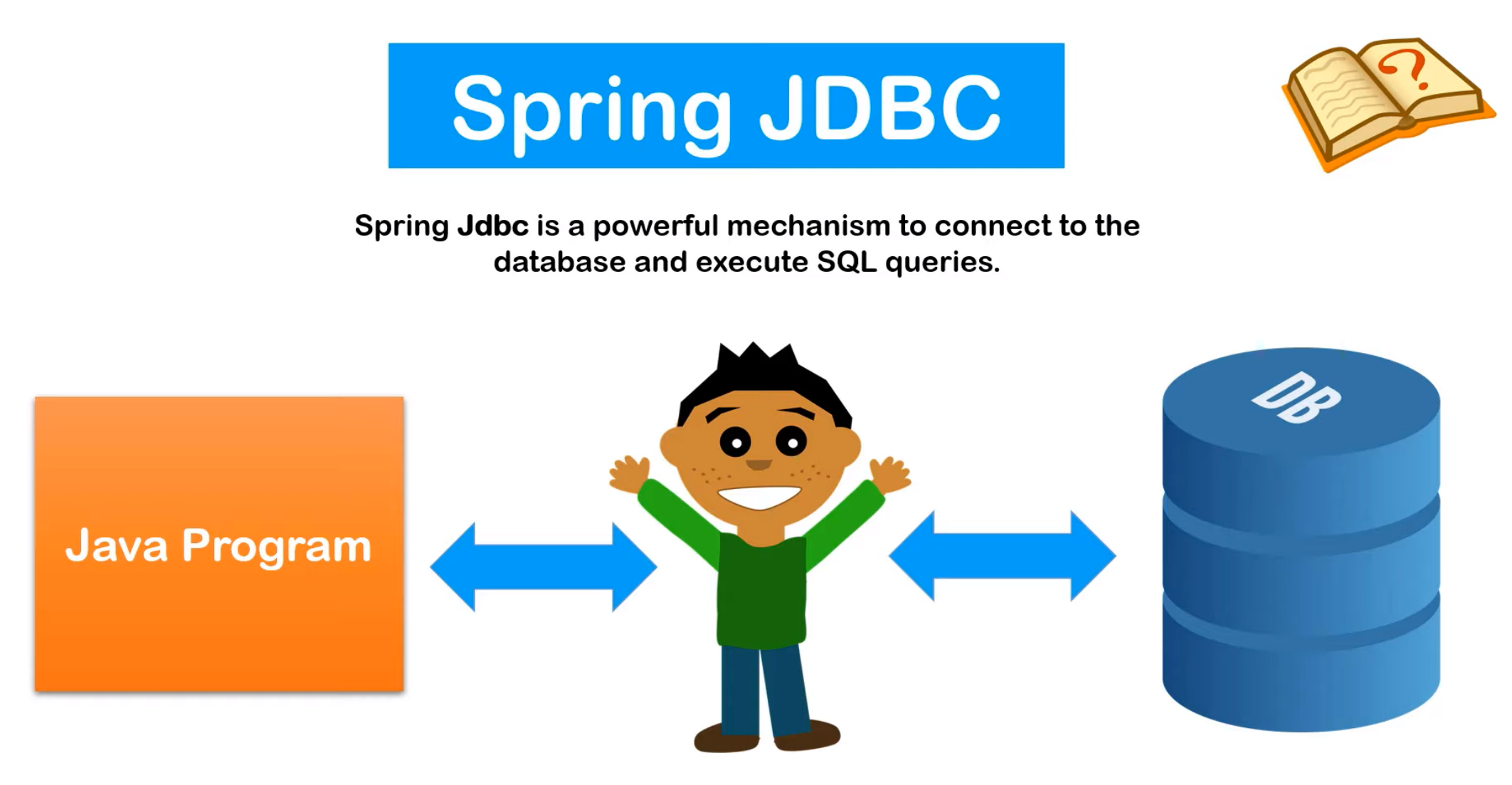
SPRING JDBC

# Intro



We can connect to our DB using Spring JDBC and execute DB queries.

Spring JDBC module (JdbcTemplate.class)

Internally this also uses classic JDBC.

# Installation

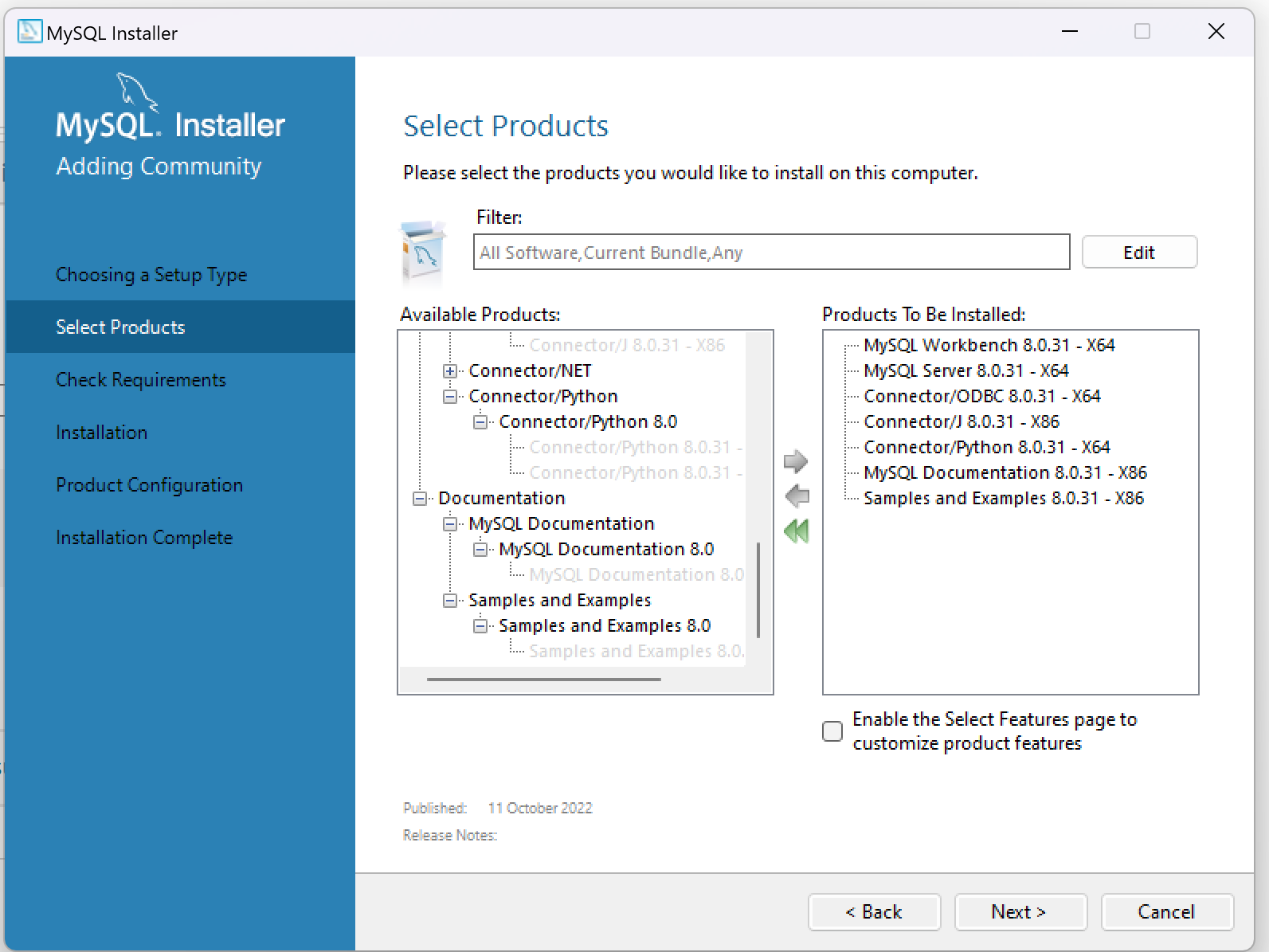
## MySQL

<https://dev.mysql.com/downloads/workbench/>

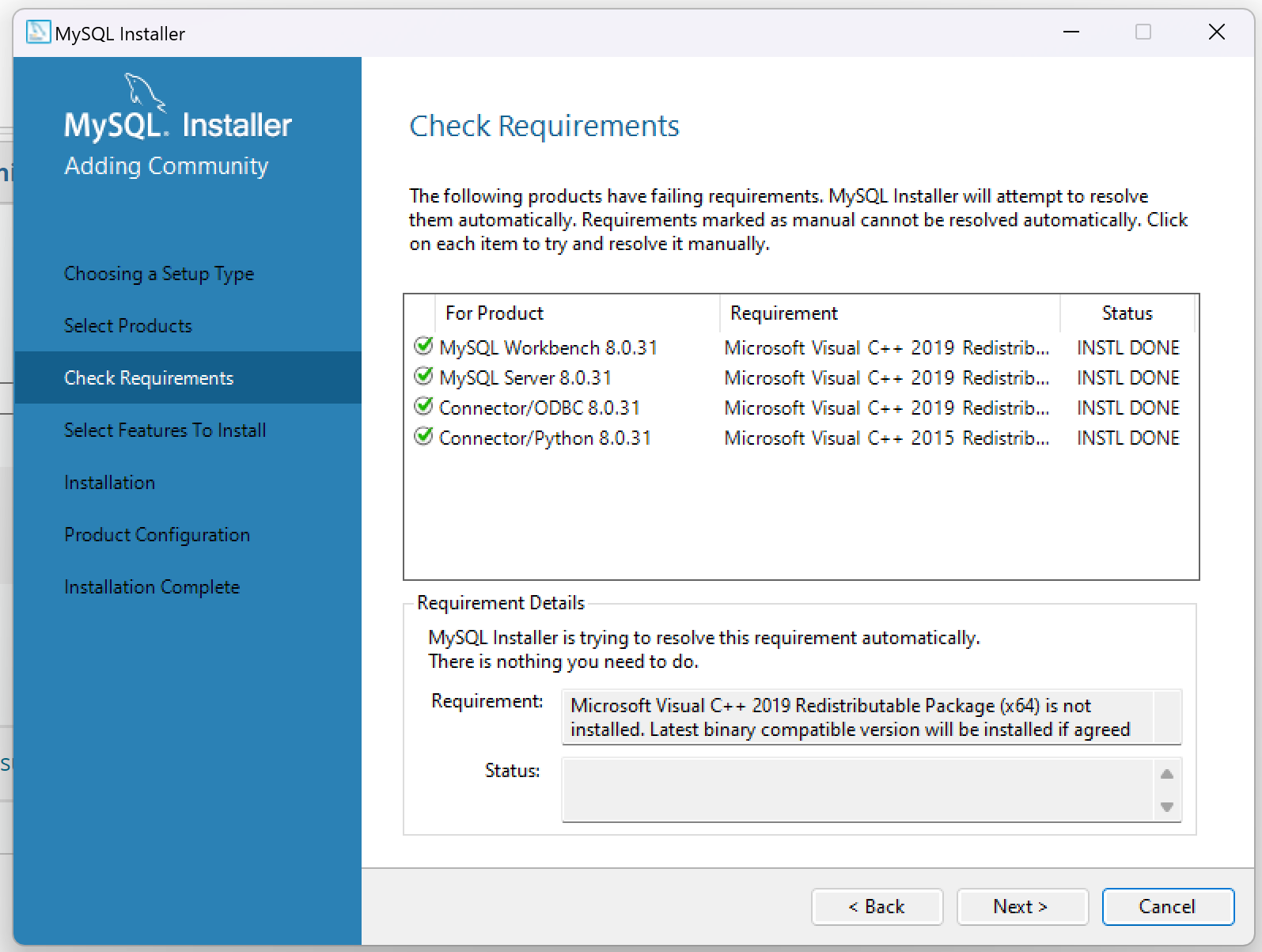
<https://dev.mysql.com/downloads/windows/installer/8.0.html>

431 MB approx.

1. Install Complete OR Custom



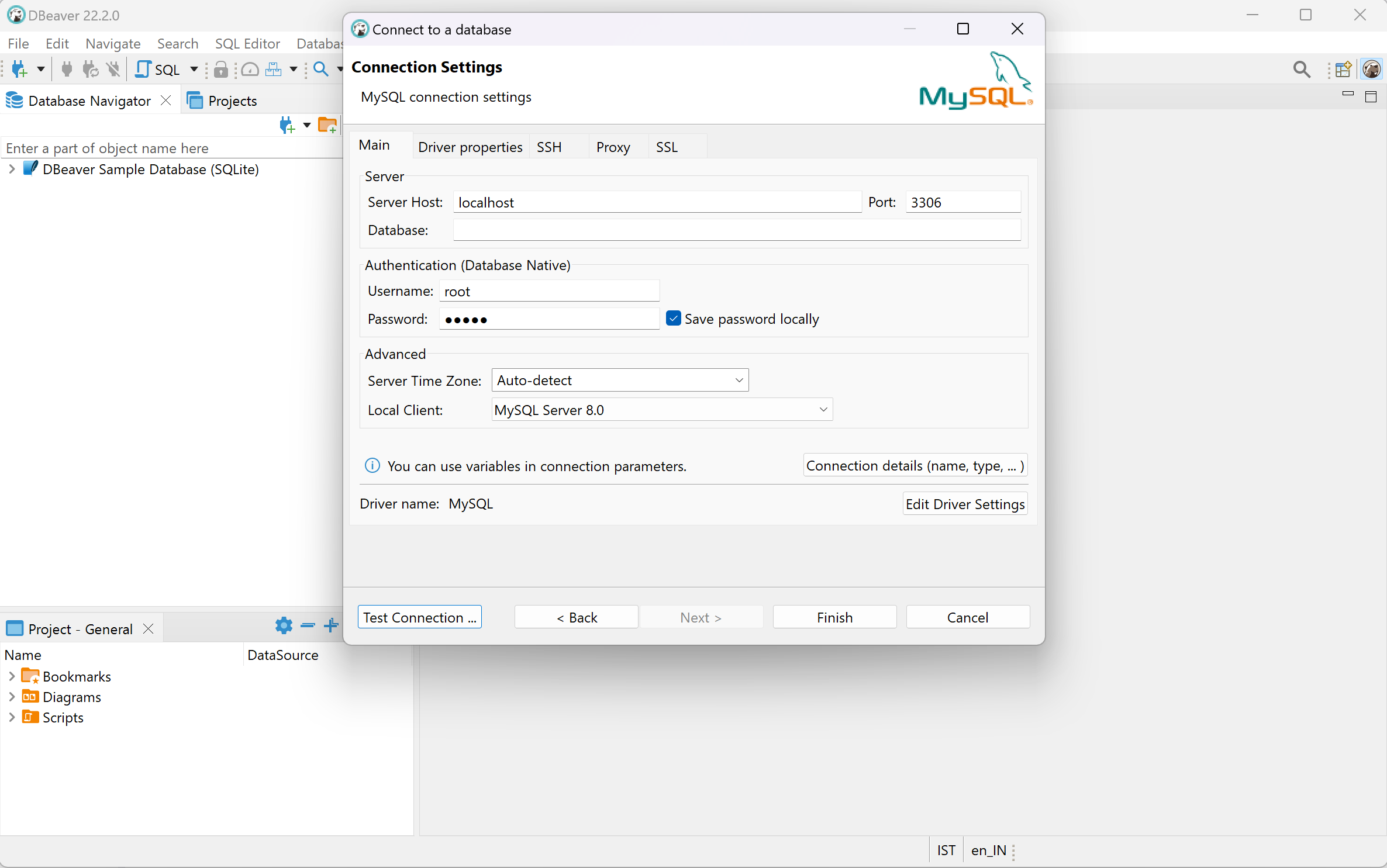
1. Execute the Dependencies



Windows Service Name: MySQL80

root

\*\*\*\*\*suman



<https://dev.mysql.com/doc/refman/8.0/en/connecting.html>

## DB Client

<https://dbeaver.io/download/>

Go for the free community edition.

# About Spring JDBC

## JAVA JDBC

JDBC is an API to perform operation with DB and usually uses lots of code.

We too had to handle the checked Exceptions and coding via JDBC was a tedious task usually involving a lot of redundancy.

* Connection OPEN
* Create Statement
* Execute Statemen
* Connection CLOSE

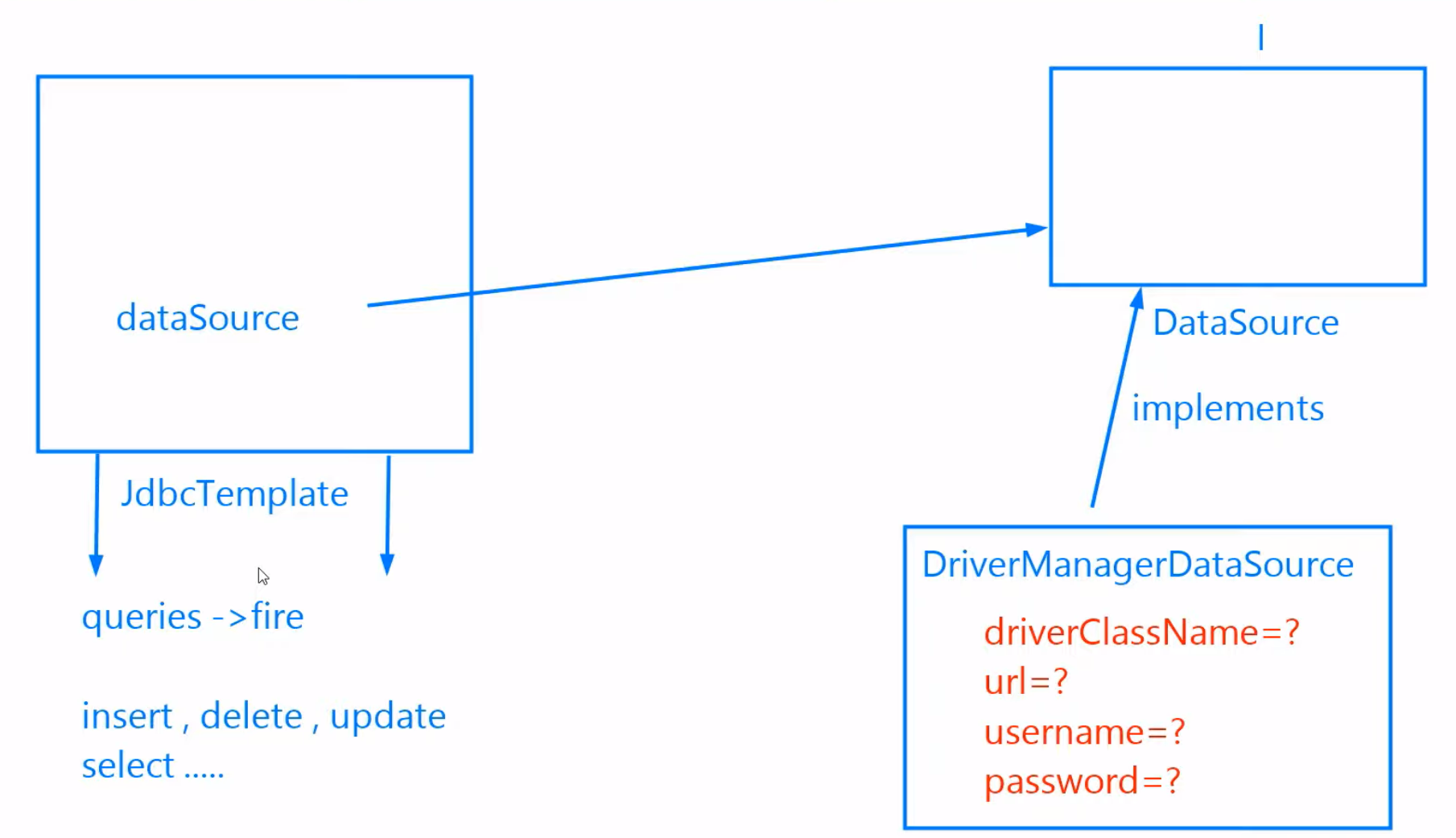
## Spring JDBC

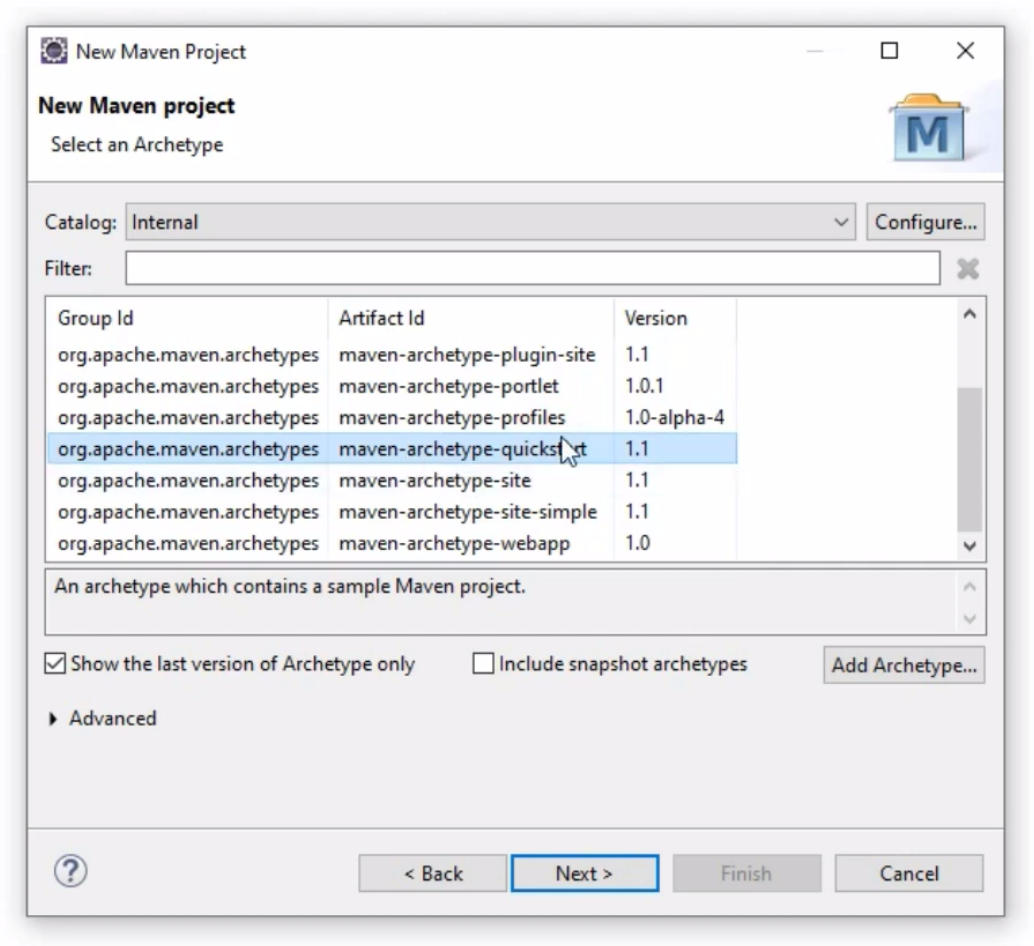
Spring JDBC class provides, **JdbcTemplate** class which contains all the essential method to perform DB operations. The important methods of this class are.

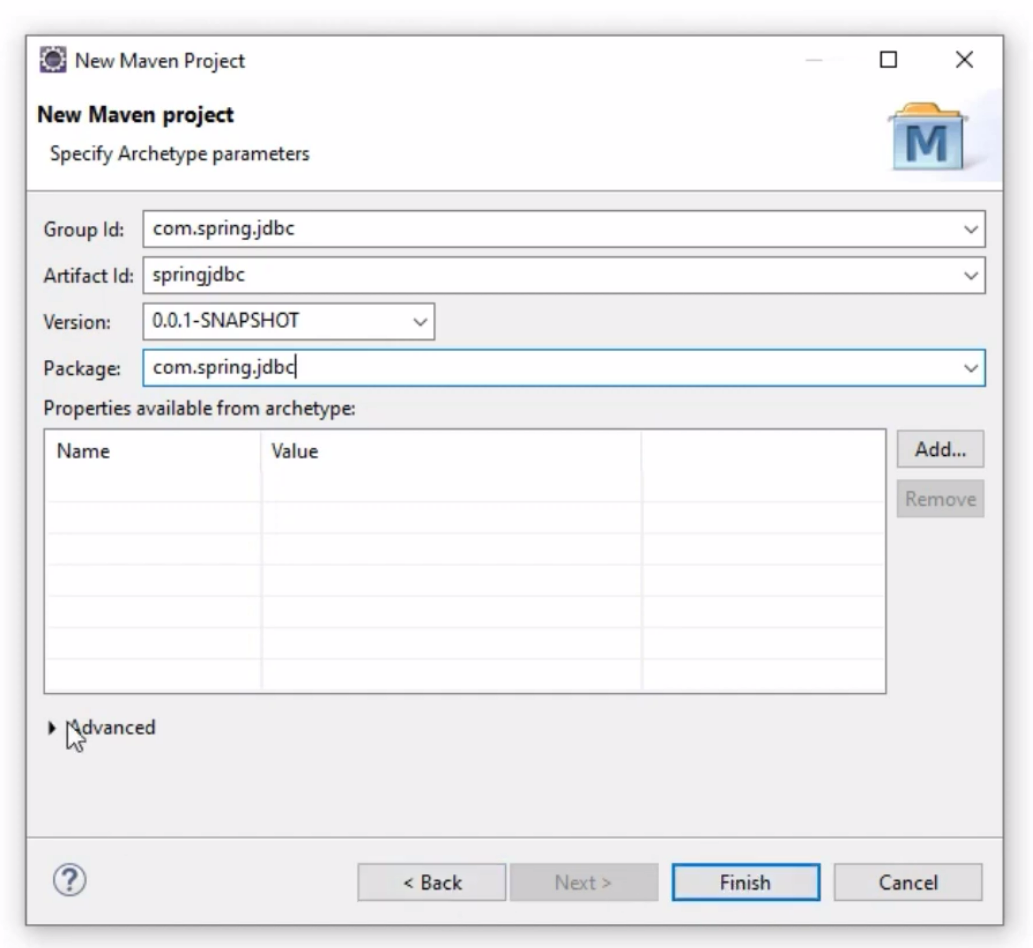
* **update()** ----> Mutate
* **execute()** ----> Selection for Query

**JdbcTemplate** needs an object of implementation of **DataSource** interface. **DataSource** contains all the info about the DB.

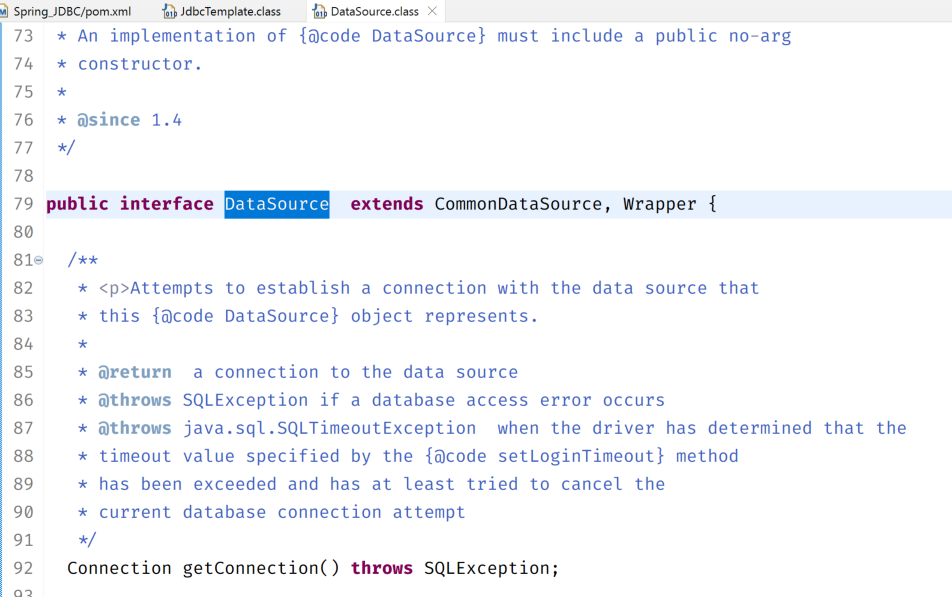
**DriverManagerDataSource** is an implementation of **DataSource** which can be used.











# JDBC-MySQL Configuration

## Initial DB creation

Enter password: \*\*\*\*\*

Welcome to the MySQL monitor. Commands end with; or \g.

Your MySQL connection id is 28

Server version: 8.0.31 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> **create database springjdbc;**

Query OK, 1 row affected (0.08 sec)

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| mysql |

| performance\_schema |

| sakila |

| springjdbc |

| sys |

| world |

+--------------------+

7 rows in set (0.05 sec)

mysql> use springjdbc;

Database changed

**mysql> create table student(id int primary key, name varchar(100) not null,city varchar(200));**

Query OK, 0 rows affected (0.09 sec)

mysql> show tables

-> ;

+----------------------+

| Tables\_in\_springjdbc |

+----------------------+

| student |

+----------------------+

1 row in set (0.03 sec)

mysql> use student;

ERROR 1049 (42000): Unknown database 'student'

mysql> insert into student(id,name,city) values(01,'Suman Shekhar','Delhi');

Query OK, 1 row affected (0.04 sec)

mysql> select \* from student;

+----+---------------+-------+

| id | name | city |

+----+---------------+-------+

| 1 | Suman Shekhar | Delhi |

+----+---------------+-------+

1 row in set (0.00 sec)

mysql> desc student;

+-------+--------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------+--------------+------+-----+---------+-------+

| id | int | NO | PRI | NULL | |

| name | varchar(100) | NO | | NULL | |

| city | varchar(200) | YES | | NULL | |

+-------+--------------+------+-----+---------+-------+

3 rows in set (0.01 sec)

**mysql> CREATE TABLE EXAMPLE\_TIMESTAMP(**

**-> ID INT PRIMARY KEY AUTO\_INCREMENT,**

**-> DATA varchar(100) NOT NULL,**

**-> CUR\_TIMESTAMP TIMESTAMP DEFAULT CURRENT\_TIMESTAMP);**

Query OK, 0 rows affected (0.03 sec)

mysql>

## MySQL JDBC Spring configuration

<https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-reference-driver-name.html>

<https://www.baeldung.com/java-jdbc-url-format>

To connect to a MySQL database from our Java application, let's first add the JDBC driver [mysql-connector-java dependency](https://search.maven.org/search?q=a:mysql-connector-java%20g:mysql) in our pom.xml:

<**dependency**>

<**groupId**>mysql</**groupId**>

<**artifactId**>mysql-connector-java</**artifactId**>

<**version**>8.0.22</**version**>

</**dependency**>

Copy

Next, let's take a look at the generic format of the connection URL supported by the MySQL JDBC driver:

protocol//[hosts][/database][?properties]Copy

Let's see an example of connecting to the MySQL database “my\_database” on the host “mysql.db.server“:

@Test

**public** **void** **givenMysqlDb\_thenCreateConnectionObject**() {

**String** jdbcUrl = "jdbc:mysql://mysql.db.server:3306/my\_database?useSSL=false&serverTimezone=UTC";

**String** username = "dbUser";

**String** password = "1234567";

**try** (**Connection** conn = DriverManager.getConnection(jdbcUrl, username, password)) {

assertNotNull(conn);

} **catch** (SQLException e) {

System.err.format("SQL State: %s\n%s", e.getSQLState(), e.getMessage());

}

}

Copy

The JDBC URL in the example above looks straightforward. It has four building blocks:

* **protocol** – jdbc:mysql:
* **host** – mysql.db.server:3306
* **database** – my\_database
* **properties** – useSSL=false&serverTimezone=UTC

However, sometimes, we may face more complex situations, such as different types of connections or multiple MySQL hosts, and so on.

# Simple Insertion

## Configuration via XML

**config.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans **xmlns**="http://www.springframework.org/schema/beans"

**xmlns:xsi**="http://www.w3.org/2001/XMLSchema-instance"

**xmlns:context**="http://www.springframework.org/schema/context"

**xmlns:p**="http://www.springframework.org/schema/p"

**xmlns:util**="http://www.springframework.org/schema/util"

**xsi:schemaLocation**="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/util

http://www.springframework.org/schema/util/spring-util.xsd">

<!-- <context:component-scan base-package="com.spring.core.javaconfig"/> -->

<bean

**class**="org.springframework.jdbc.datasource.DriverManagerDataSource" **name**="dso">

<!-- Deprecated -->

<!-- <property name="driverClassName" value="com.mysql.jdbc.Driver"/> -->

<!-- NEW driver -> com.mysql.cj.jdbc.Driver -->

<property **name**="driverClassName" **value**=" com.mysql.cj.jdbc.Driver" />

<property **name**="url" **value**="jdbc:mysql://localhost:3306/springjdbc" />

<property **name**="username" **value**="root"></property>

<property **name**="password" **value**="suman"></property>

</bean>

<bean **class**="org.springframework.jdbc.core.JdbcTemplate"

**name**="jdbcTemplate" **p:dataSource-ref**="dso">

<!-- <property name="dataSource"> -->

<!-- <ref bean="dso"/> -->

<!-- </property> -->

</bean>

<bean **class**="com.spring.jdbc.dao.StudentDAOImp1" **name**="studentDAO" **p:jdbcTemplate-ref**="jdbcTemplate" />

</beans>

## Entities

**Student.java**

package com.spring.jdbc.entities;

public class Student {

private int id;

private String name;

private String city;

public Student(int id, String name, String city) {

super();

this.id = id;

this.name = name;

this.city = city;

}

public Student() {

super();

}

@Override

public String toString() {

return "Student [id=" + id + ", name=" + name + ", city=" + city + "]";

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getCity() {

return city;

}

public void setCity(String city) {

this.city = city;

}

}

**TimestampEntity.java**

package com.spring.jdbc.entities;

import java.sql.Timestamp;

public class TimestampEntity {

private int id;// id is autogen

private String data;

private Timestamp cur\_timestamp;

public TimestampEntity(int id, String data, Timestamp cur\_timestamp) {

super();

this.id = id;

this.data = data;

this.cur\_timestamp = cur\_timestamp;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getData() {

return data;

}

public void setData(String data) {

this.data = data;

}

public Timestamp getCur\_timestamp() {

return cur\_timestamp;

}

public void setCur\_timestamp(Timestamp cur\_timestamp) {

this.cur\_timestamp = cur\_timestamp;

}

@Override

public String toString() {

return "ExampleTimestamp [id=" + id + ", data=" + data + ", cur\_timestamp=" + cur\_timestamp + "]";

}

public TimestampEntity(String data, Timestamp cur\_timestamp) {

super();

this.data = data;

this.cur\_timestamp = cur\_timestamp;

}

public TimestampEntity() {

super();

}

}

## Implementations

**StudentDAO.java**

package com.spring.jdbc.dao;

import java.util.List;

import com.spring.jdbc.entities.Student;

public interface StudentDAO {

public int insert(Student student);

public int change(Student student);

public int delete(int sId);

public Student getSingleStudent(int sId);

public List<Student> getMultipleStudent();

}

**StudentDAOImp1.java**

package com.spring.jdbc.dao;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.stereotype.Component;

import com.spring.jdbc.entities.Student;

@Component("studentDAO")

public class StudentDAOImp1 implements StudentDAO {

@Autowired

private JdbcTemplate jdbcTemplate;

public JdbcTemplate getJdbcTemplate() {

return jdbcTemplate;

}

// @Autowired

public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

System.out.println("setter used");

}

public int insert(Student student) {

// insert query

// TODO Auto-generated method stub

String query = "INSERT INTO STUDENT(ID,NAME,CITY) VALUES(?,?,?)";

int rowsInserted = this.jdbcTemplate.update(query, student.getId(), student.getName(), student.getCity());

return rowsInserted;

}

public int change(Student student) {

// update data

// TODO Auto-generated method stub

String query = "UPDATE STUDENT SET NAME=? , CITY=? WHERE ID=?";

int rowsUpdated = this.jdbcTemplate.update(query, student.getName(), student.getCity(), student.getId());

return rowsUpdated;

}

public int delete(int sId) {

// delete data

// TODO Auto-generated method stub

String query = "DELETE FROM STUDENT WHERE ID=?";

int rowsDeleted = this.jdbcTemplate.update(query, sId);

return rowsDeleted;

}

public Student getSingleStudent(int sId) {

// select single Student data

// using RowMapperImp1 class

// RowMapper<Student> rowMapper = new RowMapperImp1();

String query = "SELECT \* FROM STUDENT WHERE ID=?";

// using anonymous inner class

Student student = this.jdbcTemplate.queryForObject(query, new RowMapper<Student>() {

// <Object> by default if we don't specify <Student>

// this will be used by JAVA internally

public Student mapRow(ResultSet rs, int rowNum) throws SQLException {

// TODO Auto-generated method stub

Student student = new Student();

student.setId(rs.getInt(1));

student.setName(rs.getString(2));

student.setCity(rs.getString(3));

return student;

}

}, sId);// sID and other parameter depends on the number of placeholder in the Query

return student;

}

public List<Student> getMultipleStudent() {

// select multiple Student data

String query = "SELECT \* FROM STUDENT WHERE 1=1";

RowMapper<Student> rowMapper = new RowMapperImp1();

// using anonymous inner class

List<Student> students = this.jdbcTemplate.query(query, rowMapper);

return students;

}

}

## Main class

package com.spring.jdbc;

import java.sql.PreparedStatement;

import java.sql.Timestamp;

import java.util.HashMap;

import java.util.Map;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.jdbc.core.JdbcTemplate;

import com.mysql.cj.util.StringUtils;

import com.spring.jdbc.dao.StudentDAOImp1;

import com.spring.jdbc.entities.Student;

import com.spring.jdbc.entities.TimestampEntity;

public class AppXML {

public static void main(String[] args) {

System.out.println("START of main()");

// via XML

ApplicationContext context = new ClassPathXmlApplicationContext("com/spring/jdbc/config.xml");

JdbcTemplate template = context.getBean("jdbcTemplate", JdbcTemplate.class);

StudentDAOImp1 si = context.getBean("studentDAO", StudentDAOImp1.class);

int updated = 0;

// INSERT Query 1

String query1 = "INSERT INTO STUDENT(ID,NAME,CITY) VALUES(?,?,?)";

updated = template.update(query1, 06, "Vijay", "Kolkata");// fire query

System.out.println("ROWS UPDATED Q1: " + updated);

updated = si.insert(new Student(10, "Balwinder", "Punjab"));

System.out.println("ROWS UPDATED Q1.1: " + updated);

// INSERT Query 2

String query2 = "INSERT INTO example\_timestamp (DATA,CUR\_TIMESTAMP) VALUES(?,?)";

TimestampEntity et = new TimestampEntity();

et.setData("TimeStamp Testing");

System.out.println("CHECK => " + StringUtils.isEmptyOrWhitespaceOnly(et.getData()));

et.setCur\_timestamp(new Timestamp(System.currentTimeMillis()));

updated = template.update(query2, et.getData(), et.getCur\_timestamp());

System.out.println("ROWS UPDATED Q2: " + updated);

// INSERT Query 3

String query3 = "INSERT INTO example\_timestamp (DATA,CUR\_TIMESTAMP) VALUES(?,CURRENT\_TIMESTAMP(5))";

updated = template.update(query3, et.getData());

System.out.println("ROWS UPDATED Q3: " + updated);

// updated = template.update(query3, et.getData(), et.getCur\_timestamp());// only 1 place holder in query3

// System.out.println("ROWS UPDATED Q3.1: " + updated);

// Using HashMap to store the place-holders parameter

final Map<Integer, Object> params = new HashMap<Integer, Object>();

params.put(1, "the current time is");

// params.put(3, et.getCur\_timestamp());// only 1 place holder in query3

// for (Integer key : params.keySet()) {

// System.out.println(key + " " + params.get(key));

// ps.setObject(index, params.get(index));

// }

updated = template.update(query3, (PreparedStatement ps) -> {

for (Integer index : params.keySet()) {

// System.out.println(index + " " + params.get(index));

ps.setObject(index, params.get(index));

}

});

System.out.println("ROWS UPDATED Q4: " + updated);

System.out.println("END of main()");

}

}