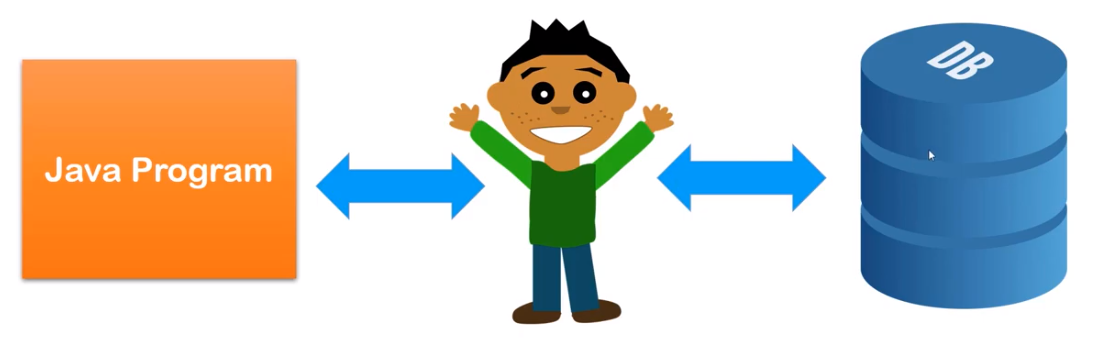
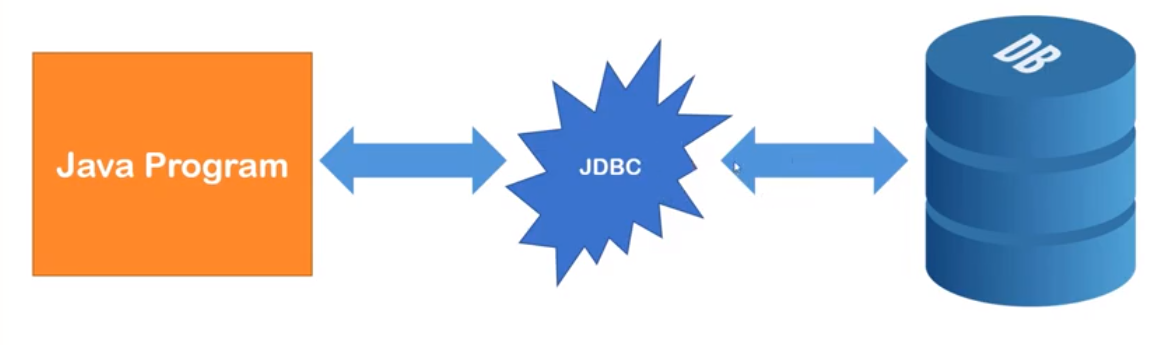
**Spring JDBC**

Spring JDBC is a powerful mechanism to connect with database and execute SQL queries.

A very popular database framework among JAVA devs.

* We will use ***JdbcTemplate*** class for operations.
* Spring JDBC is an abstraction layer on top of JDBC.
* *JDBC is an API to perform operation with database.*

**Spring JDBC guy**



##Lecture 30

*Inside JDBC uses appropriate drivers to interact.*

Problems with the old school JDBC:

1. A lot of redundant and repetitive code
   1. Connection OPEN
   2. Statement
   3. Execute
   4. Connection CLOSE
2. Exception Handling issue
   1. Checked Exception i.e.- SQL Exception
   2. Handle or Throw

Spring provides ***JdbcTemplate*** class which has all the important methods to perform operation with database. And overcome all the drawbacks of JDBC.

JdbcTemplate needs an object of DataSource.

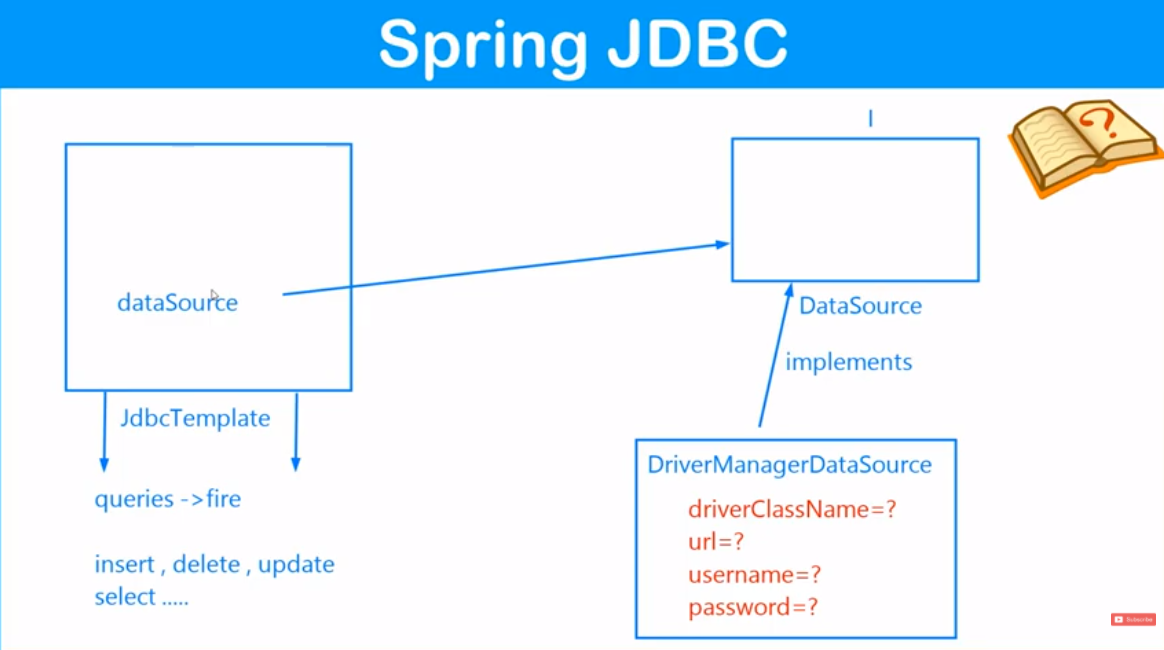
DataSource contains all the credentials for connection.

DriverManagerDataSource is an implementation for DataSource.

So we will inject an instance of DriverManagerDataSource into JdbcTemplate

DriverManagerDataSource needs 4 critical information:

1. driverClassName -> com.mysql.jdbc.driver for MySql
2. url -> protocol:subprotocol:location == jdbc:mysql:localhost:1521/XE//
3. username
4. password



Few common methods which we will explore further.

JDBC Template methods:

* update() -> insert, update, delete {state changing queries}
* execute() -> select Queries

\*\* all the above methods are overloaded.

New -> Maven Project -> Select Workspace ->

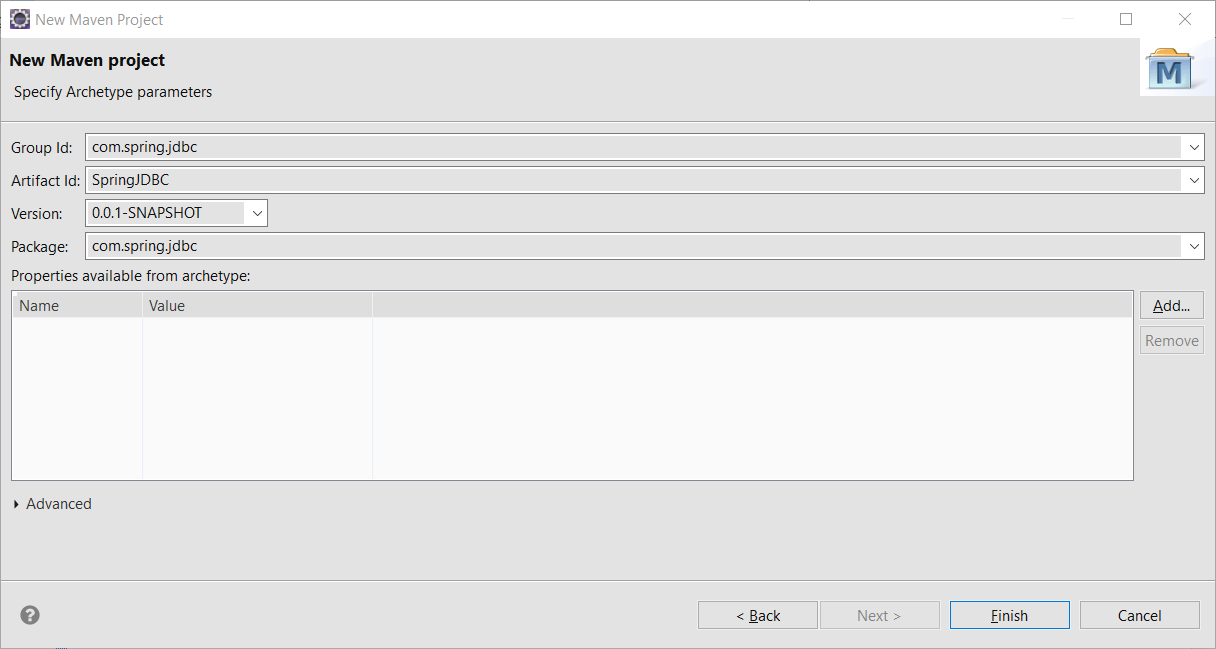
Select

Catalog: Internal

Filter: quick

Go for -> maven-archtype-quickstart 1.1

Enter Artifact ID



Search for

Spring JDBC Maven

Match the Spring JDBC version with SPRING Core & Context

i.e. - 5.3.3

add the dependency to pom.xml along with the SPRING Core & Context dependencies.

Installing MySQL

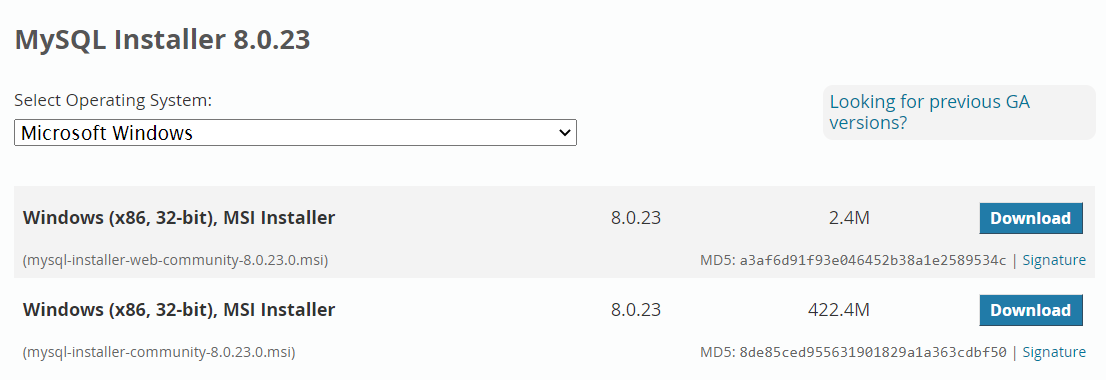
Search -> mysql server download

https://dev.mysql.com/downloads/mysql/

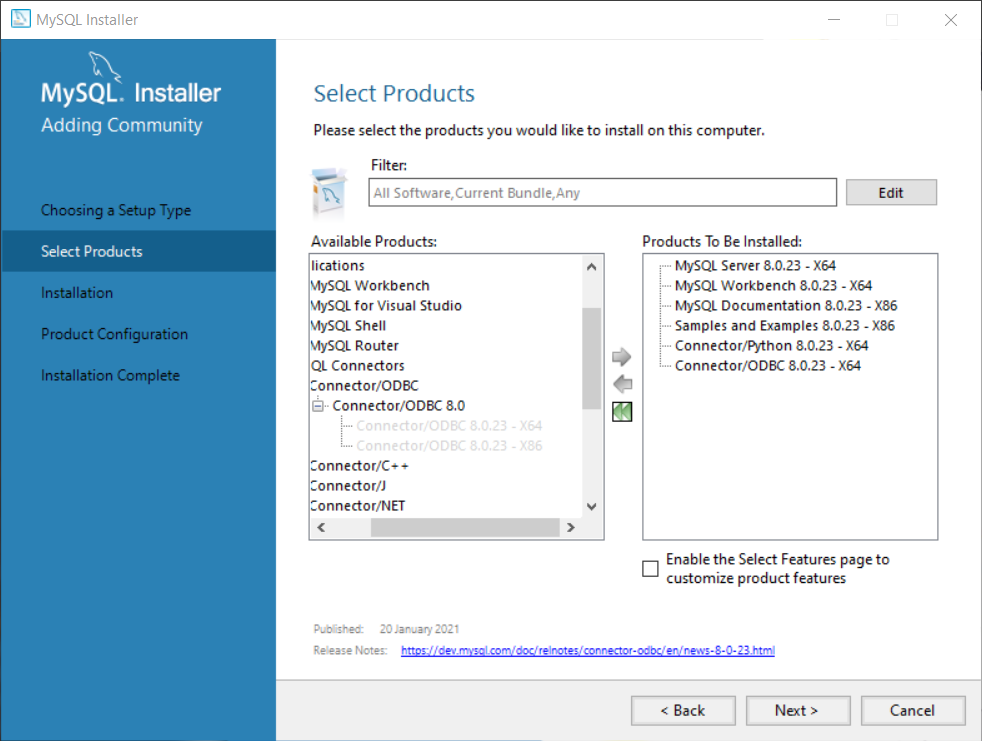


Selecting the required versions (8.0.23)

Click on ***Go to Download Page***



Select the 2nd Option and begin download.

[https://www.youtube.com/watch?v=DxoRUmW44JE](https://www.youtube.com/watch?v=DxoRUmW44JE)

While installing select custom:

[https://answers.microsoft.com/en-us/windows/forum/windows\_other-windows\_programs/how-to-completely-uninstall-mysql/e90e1344-7b90-4319-8b2f-77b271ae66ed#:~:text=Go%20to%20Control%20Panel%20%3EPrograms,which%20includes%20your%20actual%20data](https://answers.microsoft.com/en-us/windows/forum/windows_other-windows_programs/how-to-completely-uninstall-mysql/e90e1344-7b90-4319-8b2f-77b271ae66ed#:~:text=Go to Control Panel >Programs,which includes your actual data).

Thank you for writing to Microsoft Community Forums.

Which version of Windows is installed on your device?

Refer the article ‘[**Which Windows operating system am I running**](https://support.microsoft.com/en-us/help/13443/windows-which-operating-system)’ to provide the details.

I suggest you to follow the steps mentioned below to uninstall MySQL from the device.

Step 1

Uninstall MySQL from Control Panel. To do so,

Go to Control Panel >Programs and Features > Select MySQL > Click on Uninstall.

After you have uninstalled MySQL through Add/Remove programs, you now need to remove the existing database information which includes your actual data.  If you need any of that data, be sure to back it up. Next you can delete the data directory.

Step 2

Run Command Prompt as Administrator and execute the following command to stop and remove MySQL service.

Net stop MySQL

Sc delete MySQL

Step 3

1. Click on Start, type in Show hidden files and folders.
2. Select the View tab and select Show hidden files and folders.
3. Now explore the following locations and delete following folders.

C:\Program Files\MySQL

C:\Program Files (x86)\MySQL

C:\ProgramData\MySQL

And if exists, delete the folder from the location.

C:\Users\[User-Name]\AppData\Roaming\MySQL

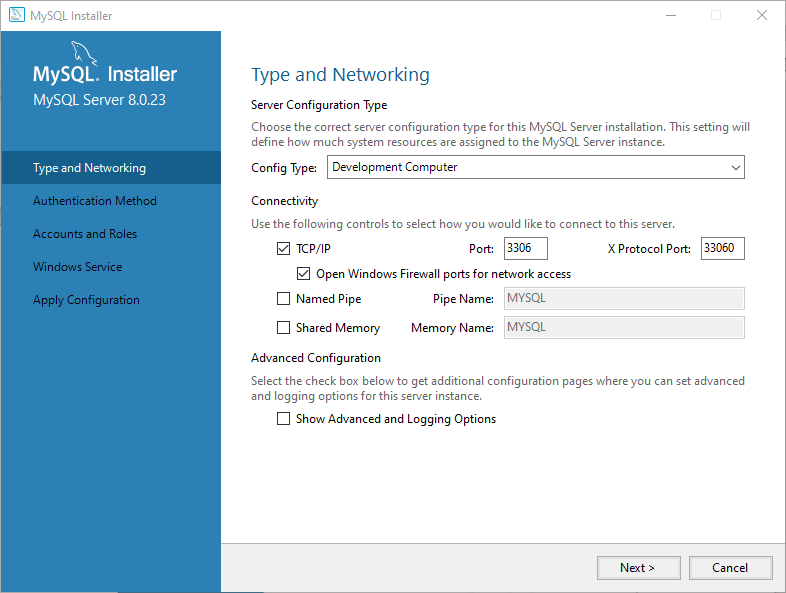
1. Restart your device and check the status.

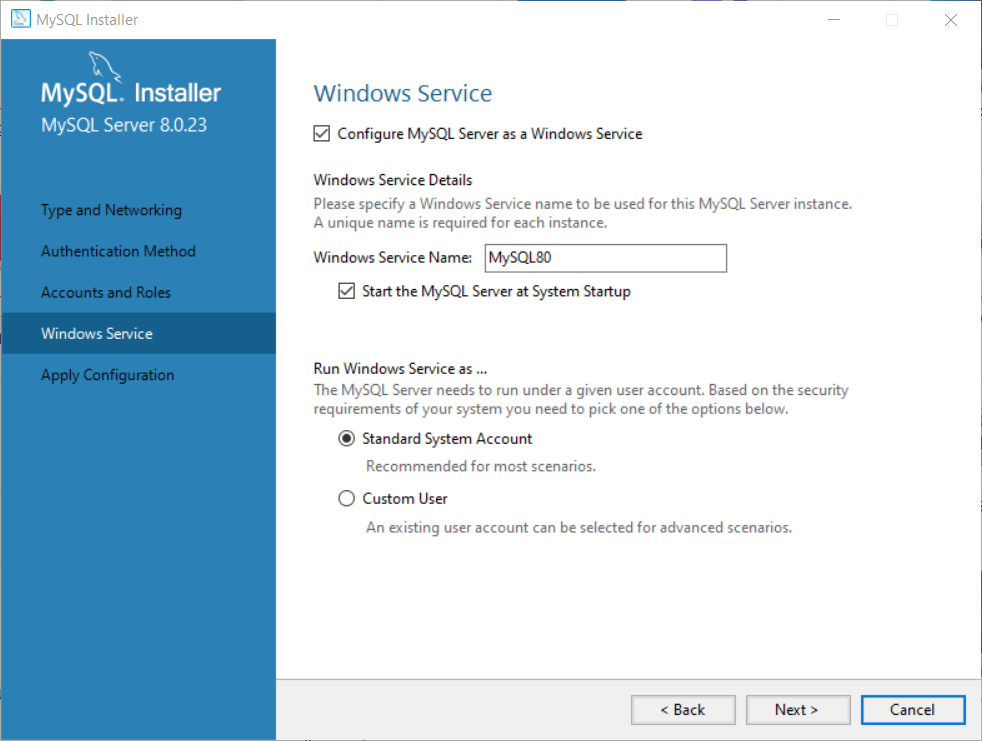
Get back to us for further assistance on this issue. We will be happy to help you.

Regards,

Muralidhar

Microsoft Community Moderator





**MISCELLANEOUS INSTALLED**

* MySQL Server
* MySQL Command Line Client
* Workbench

FEW GUI Clients

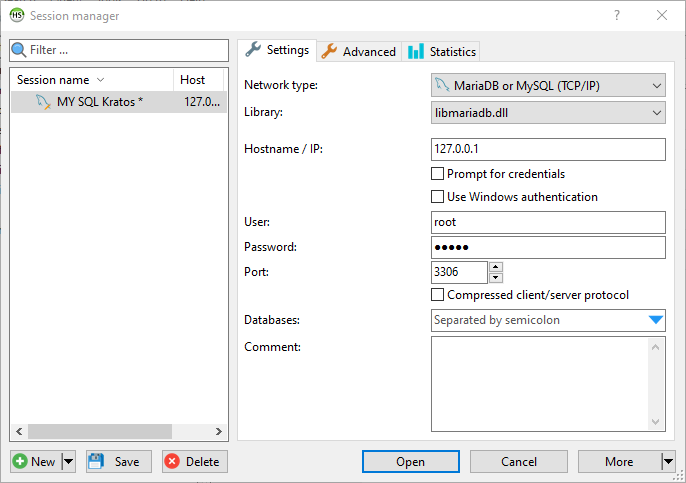
TABLE PLUS

[https://tableplus.com/blog/2018/11/what-is-the-default-username-password-in-mysql.html#:~:text=In%20MySQL%2C%20by%20default%2C%20the,skip%2Dgrant%2Dtables%20option](https://tableplus.com/blog/2018/11/what-is-the-default-username-password-in-mysql.html#:~:text=In MySQL%2C by default%2C the,skip-grant-tables option).

https://codingsight.com/10-best-mysql-gui-tools/

]

https://www.heidisql.com/download.php?download=installer



BASICS;

> \*\*\*\*

> MySQL

DB1

DB2

DB8

DB3

*MySQL Command Line Client*

*MySQL Database Server*

**DB1**

table 2

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

table 1

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

So A database consists of many tables.

And simply we can access them from CMD firing appropriate queries.

open MySQL Command Line Client

pwd \*\*\*\*

> show databases;

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

| Database |

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

| information\_schema |

| mysql |

| performance\_schema |

| sakila |

| sys |

| world |

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

> create database [db\_name];

**mysql> create database demo;**

Query OK, 1 row affected (0.01 sec)

**> drop database [db\_name];**

*deletes the database*

> use [db\_name];

switches the database

**mysql> use demo;**

Database changed

> create table [table\_name] (col1 type constraints, col2 , col3.....)

**mysql> create table user (id int(11) primary key, name varchar(100) not null , city varchar (50));**

Query OK, 0 rows affected, 1 warning (0.04 sec)

**mysql> show tables;**

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

| Tables\_in\_demo |

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

| user |

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

*1 row in set (0.01 sec)*

**mysql> desc user;**

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

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

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

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

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

| city | varchar(50) | YES | | NULL | |

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

3 rows in set (0.03 sec)

**mysql> drop table user;**

Query OK, 0 rows affected (0.02 sec)

**mysql> alter table user rename to student;**

Query OK, 0 rows affected (0.03 sec)

**mysql> truncate table student;**

data flushed

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

Query OK, 1 row affected (0.02 sec)

**mysql> insert into student values (02,'Apurv','chennai');**

Query OK, 1 row affected (0.01 sec)

**mysql> alter table student add country varchar(50);**

Query OK, 0 rows affected (0.05 sec)

Records: 0 Duplicates: 0 Warnings: 0

**mysql> update student set country ='India';**

Query OK, 2 rows affected (0.01 sec)

Rows matched: 2 Changed: 2 Warnings: 0

**mysql> update student set country ='Scotland' where name ='Apurv';**

Query OK, 1 row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

**mysql> update student set country ='China', city='Shanghai' where name='Suman';**

Query OK, 1 row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

**mysql> select \* from student;**

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

| id | name | city | country |

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

| 1 | Suman | Shanghai | China |

| 2 | Apurv | chennai | Scotland |

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

2 rows in set (0.01 sec)

**mysql> delete from student;**

bad alternative to truncate.

**mysql> insert into student values (03,'Malay','chennai','India');**

Query OK, 1 row affected (0.00 sec)

**mysql> delete from student where id=03;**

Query OK, 1 row affected (0.00 sec)

**mysql> create database springjdbc;**

Query OK, 1 row affected (0.01 sec)

**mysql> use springjdbc;**

Database changed

mysql> Query OK, 1 row affected (0.01 sec)

->

->

-> ^C

mysql>

mysql>  **ctrl + c** to go to a fresh line

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

Query OK, 0 rows affected (0.06 sec)

**mysql> show tables;**

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

| Tables\_in\_springjdbc |

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

| student |

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

1 row in set (0.00 sec)

**mysql> insert into student(id,name,city) values (100,'Suman Shekhar','Delhi');**

Query OK, 1 row affected (0.01 sec)

**mysql> desc student;**

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

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

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

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

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

| city | varchar(100) | YES | | NULL | |

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

3 rows in set (0.01 sec)

**mysql> select \* from student;**

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

| id | name | city |

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

| 100 | Suman Shekhar | Delhi |

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

1 row in set (0.00 sec)

We also need MySQL connector jars for interacting with database.

MYSQL Server 5.7

Search for MySQL connector in Maven Repository.

Click the 1st option [MySQL Connector/J](https://mvnrepository.com/artifact/mysql/mysql-connector-java)

Click the matching version with MySQL database (8.0.23)

Add the dependency.

So Dependencies used:

1. Spring CORE
2. Spring CONTEXT
3. Spring Annotations
4. Spring JDBC
5. Spring MySQL Connector

Pom.xml

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

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.spring.jdbc</groupId>

<artifactId>SpringJDBC</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>SpringJDBC</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

*<!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->*

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.3.3</version>

</dependency>

*<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->*

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.3</version>

</dependency>

*<!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->*

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>5.3.3</version>

</dependency>

*<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->*

<dependency>

<groupId>mysql</groupId>

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

<version>8.0.23</version>

</dependency>

*<!-- dependecy to use annotation -->*

*<!-- @PostConstruct,@PreDestroy -->*

<dependency>

<groupId>javax.annotation</groupId>

<artifactId>javax.annotation-api</artifactId>

<version>1.3.2</version>

</dependency>

</dependencies>

</project>

Ctrl+Shift+T (OPEN Type)

Search for JDBCTemplate to get a glimpse of the class.

Google Search -> mysql jdbc driver class

Table 1–5 MySQL Connector/J Driver Database Connection Information

| Parameter | Value |
| --- | --- |
| Driver Jar Files | mysql-connector-java-3.0.11-stable-bin.jar |
| Driver Java Class Name | ~~com.mysql.jdbc.Driver~~  com.mysql.cj.jdbc.Driver |
| URL Connection String | jdbc:mysql://server-name:server-port/database-name  tech:database://Localhost:3306/springjdbc  **Note –** NOTE: Default server port is 3306 |
| User Name | Login name of the account used to access the database. |
| Password | Password associated with the login account name used to connect to the database. |

START of main()

Loading class `**com.mysql.jdbc.Driver**'. This is deprecated. The new driver class is `**com.mysql.cj.jdbc.Driver**'. The driver is automatically registered via the SPI and manual loading of the driver class is generally unnecessary.

RECORD INSERTED -> 1

|  |  |  |
| --- | --- | --- |
| RDBMS | JDBC driver name | URL format |
| MySQL | ~~com.mysql.jdbc.Driver~~  com.mysql.cj.jdbc.Driver | **jdbc:mysql://**hostname/ databaseName |
| ORACLE | oracle.jdbc.driver.OracleDriver | **jdbc:oracle:thin:@**hostname:port Number:databaseName |
| DB2 | COM.ibm.db2.jdbc.net.DB2Driver | **jdbc:db2:**hostname:port Number/databaseName |
| Sybase | com.sybase.jdbc.SybDriver | **jdbc:sybase:Tds:**hostname: port Number/databaseName |

https://www.tutorialspoint.com/jdbc/jdbc-db-connections.htm

##Lecture 31

Steps to change the JAVA library

right click

properties

JAVA build path => libra

remove, add library -> JRE System Library -> ...

##Lecture 32,33,34,35

We need Spring JdbcTemplate.class for CRUD operations.

Ctrl+Shift+T (OPEN Type)

Search for JDBCTemplate to get a glimpse of the class.

**public** **class** JdbcTemplate **extends** JdbcAccessor **implements** JdbcOperations

**public** JdbcTemplate(DataSource dataSource) {

setDataSource(dataSource);

afterPropertiesSet();

}

So an instance of implementation of DataSource need to be injected.

*Ctrl+click on DataSource to open the pointing implementation class*

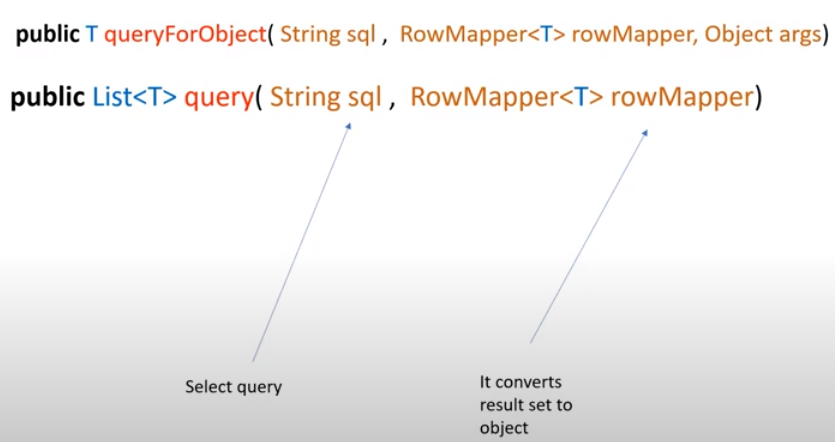
**public** **interface** DataSource **extends** CommonDataSource, Wrapper

DriverManagerDataSource is an implementation of DataSource which we will use.

DriverManagerDataSource **extends** AbstractDriverBasedDataSource **public** **abstract**

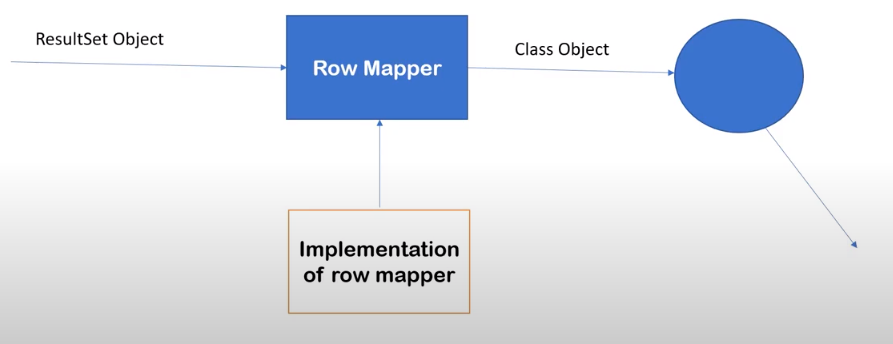
**class** AbstractDriverBasedDataSource **extends** AbstractDataSource {

**public** **abstract** **class** AbstractDataSource **implements** DataSource {



RowMapper is an interface

So we will provide an implementation of RowMapper



****

**J D B C**

**R**

**U**

**D**

We need MySQL connector jars for interacting with database.

MYSQL Server 5.7

Search for MySQL connector in Maven Repository.

Click the 1st option [MySQL Connector/J](https://mvnrepository.com/artifact/mysql/mysql-connector-java)

Click the matching version with MySQL database (8.0.23)

Add the dependency.

So Dependencies used:

1. Spring CORE
2. Spring CONTEXT
3. Spring Annotations
4. Spring JDBC
5. Spring MySQL Connector

Pom.xml

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

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.spring.jdbc</groupId>

<artifactId>SpringJDBC</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>SpringJDBC</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

*<!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->*

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.3.3</version>

</dependency>

*<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->*

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.3</version>

</dependency>

*<!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->*

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>5.3.3</version>

</dependency>

*<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->*

<dependency>

<groupId>mysql</groupId>

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

<version>8.0.23</version>

</dependency>

*<!-- dependecy to use annotation -->*

*<!-- @PostConstruct,@PreDestroy -->*

<dependency>

<groupId>javax.annotation</groupId>

<artifactId>javax.annotation-api</artifactId>

<version>1.3.2</version>

</dependency>

</dependencies>

</project>

**package com.spring.jdbc;**

SpringJDBCJavaConfig.java

package com.spring.jdbc;

import javax.sql.DataSource;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import com.spring.jdbc.dao.StudentDAO;

import com.spring.jdbc.dao.StudentDAOImp1;

@Configuration

public class SpringJDBCJavaConfig {

@Bean("ds")

public DataSource getDataSource() {

/\*

\* DriverManagerDataSource extends AbstractDriverBasedDataSource public abstract

\* class AbstractDriverBasedDataSource extends AbstractDataSource { public

\* abstract class AbstractDataSource implements DataSource {

\*/

DriverManagerDataSource ds = new DriverManagerDataSource();

/\*

\* <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>

\*/

ds.setDriverClassName(" com.mysql.cj.jdbc.Driver");

ds.setUrl("jdbc:mysql://localhost:3306/springjdbc");

ds.setUsername("root");

ds.setPassword("suman");

return ds;

}

@Bean("jdbcTemplate")

public JdbcTemplate getTemplate() {

JdbcTemplate jdbcTemplate = new JdbcTemplate();

jdbcTemplate.setDataSource(getDataSource());

return jdbcTemplate;

}

@Bean("studentDAO")

public StudentDAO getStudentDao() {

StudentDAOImp1 studentDAO = new StudentDAOImp1();

studentDAO.setJdbcTemplate(getTemplate());

return studentDAO;

}

}

AutowiredSpringJDBCJavaConfig.java

package com.spring.jdbc;

import javax.sql.DataSource;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.datasource.DriverManagerDataSource;

@Configuration

/\*

\* Spring Container will scan the below package and

\* return the @Component bean when asked

\*/

@ComponentScan(basePackages = {"com.spring.jdbc.dao"})

public class AutowiredSpringJDBCJavaConfig {

/\*

\* the below beans are required as it will be used to satisfy the dependency for

\* the StudentDAOImp1

\*/

@Bean("ds")

public DataSource getDataSource() {

/\*

\* DriverManagerDataSource extends AbstractDriverBasedDataSource public abstract

\* class AbstractDriverBasedDataSource extends AbstractDataSource {

\* public abstract class AbstractDataSource implements DataSource {

\*/

DriverManagerDataSource ds = new DriverManagerDataSource();

/\*

\* <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>

\*/

ds.setDriverClassName(" com.mysql.cj.jdbc.Driver");

ds.setUrl("jdbc:mysql://localhost:3306/springjdbc");

ds.setUsername("root");

ds.setPassword("suman");

return ds;

}

@Bean("jdbcTemplate")

public JdbcTemplate getTemplate() {

JdbcTemplate jdbcTemplate = new JdbcTemplate();

jdbcTemplate.setDataSource(getDataSource());

return jdbcTemplate;

}

}

App.java

package com.spring.jdbc;

import java.util.List;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.spring.jdbc.dao.StudentDAO;

import com.spring.jdbc.entities.Student;

/\*\*

\* Hello world!

\*

\*/

public class App {

public static void main(String[] args) {

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

// via XML

/\*

\* ApplicationContext context = new

\* ClassPathXmlApplicationContext("com/spring/jdbc/config.xml");

\*/

// via JAVA config

ApplicationContext context =

new AnnotationConfigApplicationContext(AutowiredSpringJDBCJavaConfig.class);

StudentDAO studentDAO = context.getBean("studentDAO", StudentDAO.class);

// Student student = new Student();

// student.setId(52);

// student.setName("Ritika Saha");

// student.setCity("Imphal");

// int rowsInserted = studentDAO.insert(student);

// int rowsUpdated = studentDAO.change(student);

// int rowsDeleted = studentDAO.delete(59);

// System.out.println("RECORD CHANGED -> " + rowsDeleted);

// Student singleStudent = studentDAO.getSingleStudent(54);

// System.out.println(singleStudent);

List<Student> students = studentDAO.getMultipleStudent();

for (Student student : students) {

System.out.println(student);

}

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

}

}

**package com.spring.jdbc.dao;**

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

// TODO Auto-generated method stub

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

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

// using anonymous inner class

Student student = this.jdbcTemplate.queryForObject (query,

//Its mapped for Student

new RowMapper<Student>() {

//<Object> by default

//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);

return student;

}

public List<Student> getMultipleStudent() {

// TODO Auto-generated method stub

// 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;

}

}

RowMapperImp1.java

package com.spring.jdbc.dao;

import java.sql.ResultSet;

import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

import com.spring.jdbc.entities.Student;

public class RowMapperImp1 implements RowMapper<Student>{

//<Object> by default

//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;

}

}

**com.spring.jdbc.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**();

// **TODO** Auto-generated constructor stub

}

@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;

}

}

**SPRING ORM**

Few popular ORM

* + Hibernate
  + Toplink
  + iBatis

Benefits:

1. Less coding wrt to pure Hibernate.
2. Easy to test
3. Better Exception Handling techniques and you also can add your custom Exception
4. Integrated trans
5. action management
6. Can take benefit of all the Spring feature

Class ABC{

String name;

String college;

}

ORM

|  |  |
| --- | --- |
| Name | College |
|  |  |
|  |  |

*Java Code Database Server*In Spring We have HibernateTemplate class to work with.

Few methods of HibernateTemplate:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. Save | 1. update | 1. insert | 1. get | 1. loadAll |

*Interface*

ProductDao

ProductDaoImpl

HibernateTemplate

*depends on*

ProductDaoImple is an

implementation of ProductDao interface

*depends on*

*Interface*

SessionFactory

*implements SessionFactory*

LocalSessionFactoryBean

JPA is a specification (API) given by SUN/Oracle and Hibernate is an implementation of JPA

*Search spring orm maven*

Match the version with Spring CORE and add to the existing dependencies.

Spring JDBC is a dependency of SPRING ORM **5.3.3**

***public*** **class** HibernateTemplate **implements** HibernateOperations, InitializingBean {

*Search hibernate maven*

*Click on the Hibernate Core Relocation* **5.3.3**