**Guided Lab 305.4.1B**

**Hibernate Project Demonstration with IntelliJ Ultimate**

**Lab Overview:**

In this lab, we will walk you through creating our first CRUD application with Hibernate. When developing hibernate applications, we need to provide two sets of configuration in the hibernate.cfg.xml file.

* The first set of configuration contains database specific properties that will be used to create Database connections and Session objects.
* The second set of configurations contains mappings between model classes and database tables.

**Learning Objective:**

At the end of this lab, you will be able to develop **Hibernate** applications.

**Instructions:**

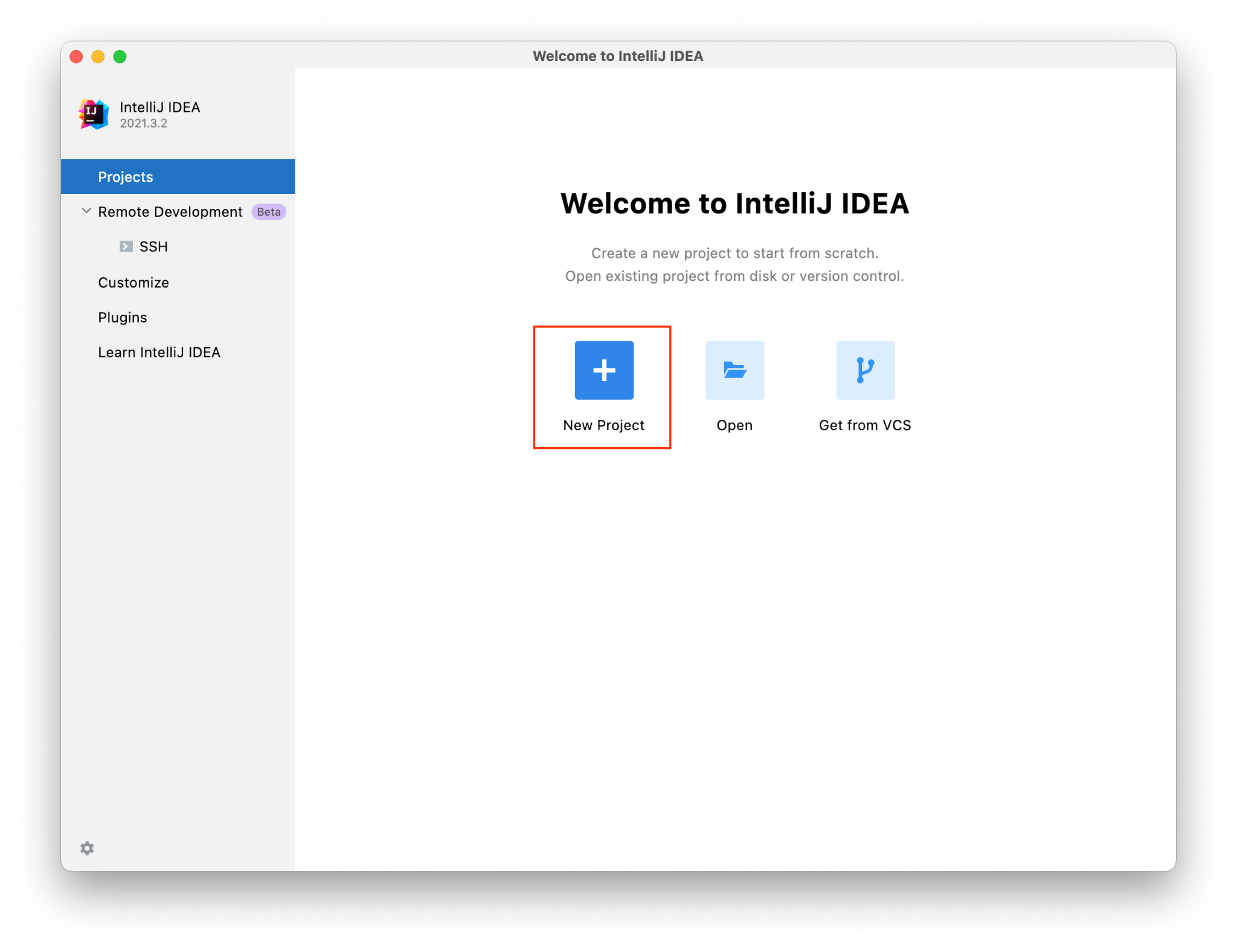
## 

## **Setup a Java Maven Project**

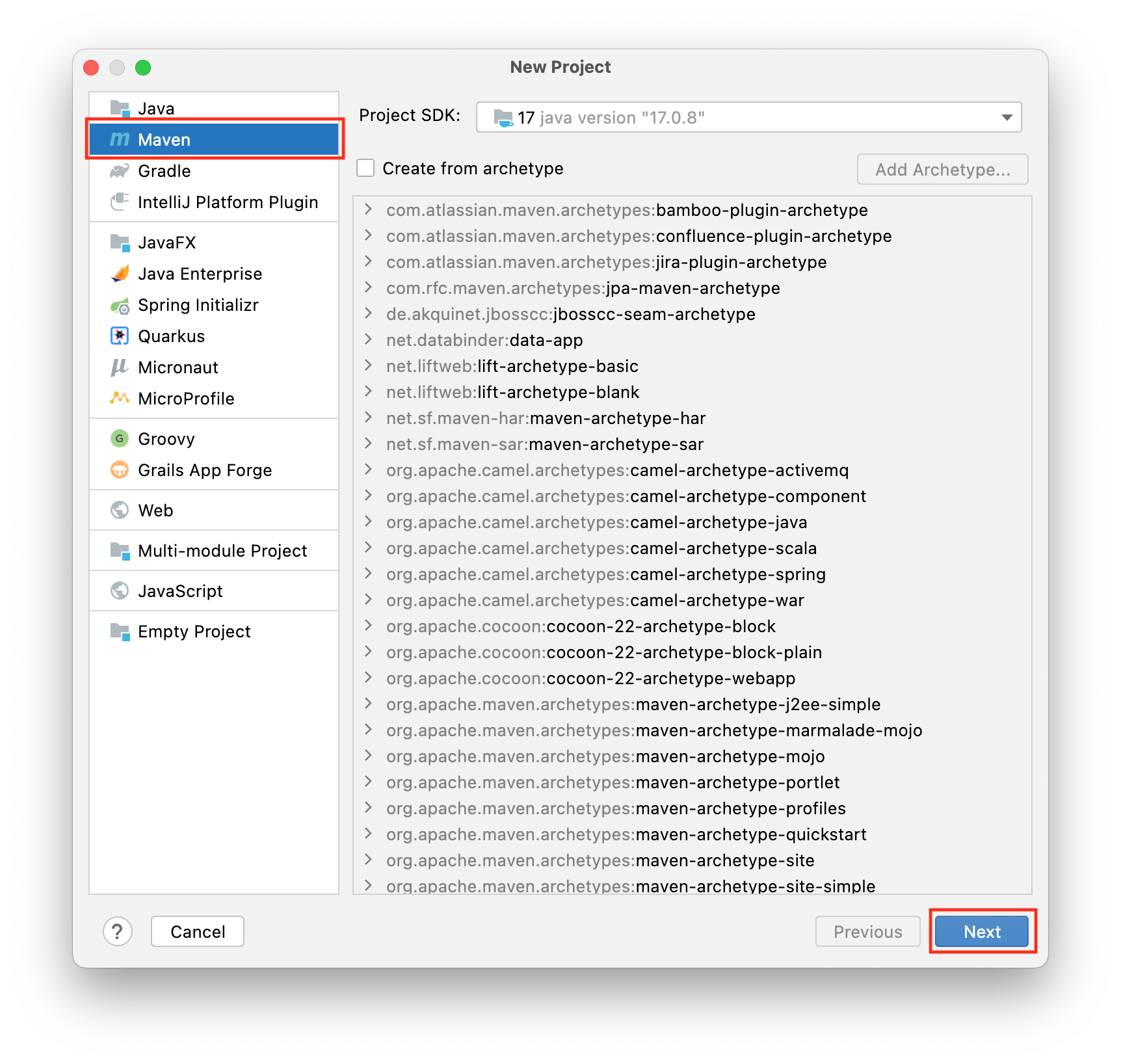
* We are going to create a Maven-based Hibernate application using annotations in the IDE.
* To create the Maven-based Hibernate application in the IDE, we need to follow the below steps:

### 1) Create the Maven Project

* To create a Maven project, click on New Project.

****

* Select Maven and click on Next.



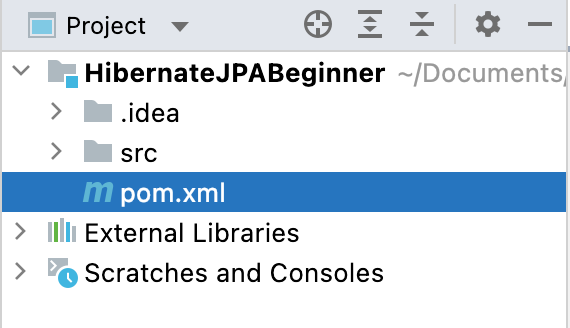
* Name your project **HibernateJPABeginner** and click Next.

### 

### 

### 2) Add Jar Dependencies and Configuration in the pom.xml file.

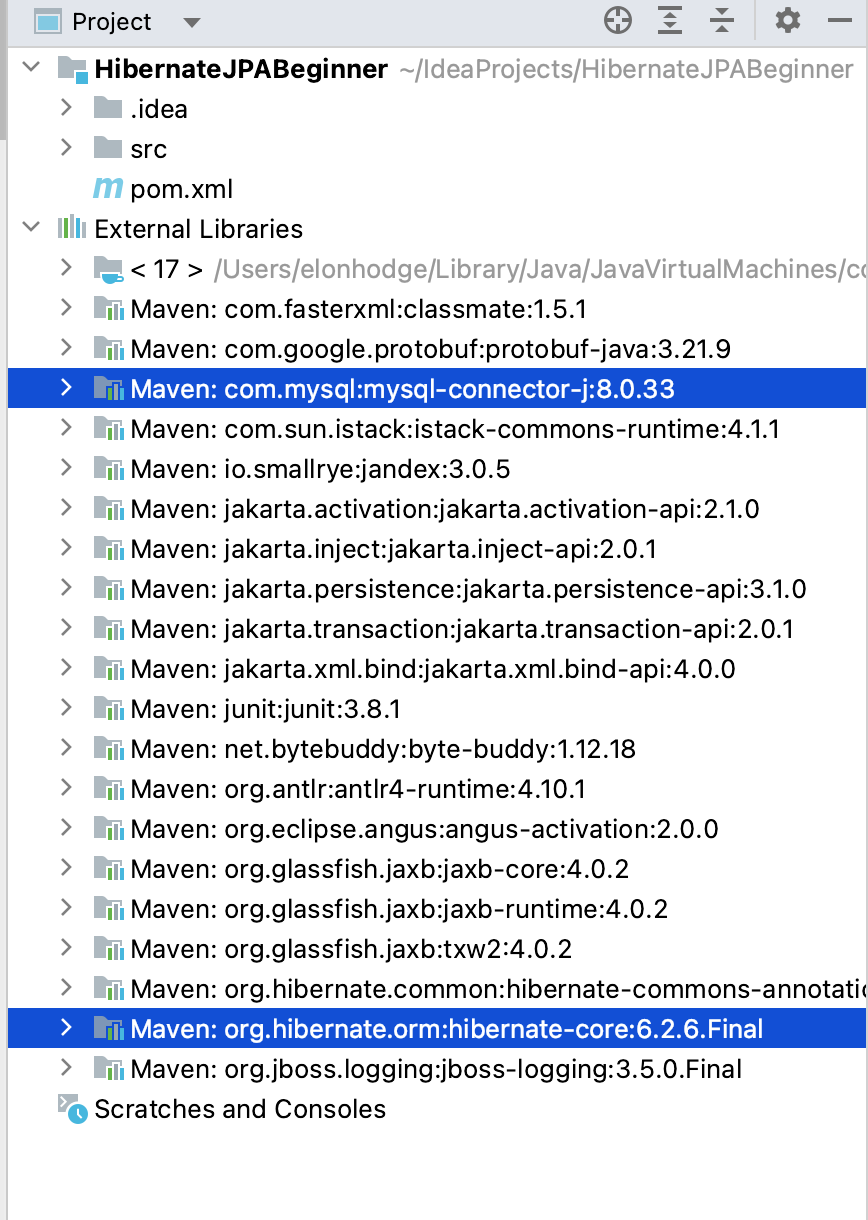
Next, we need to add a couple of jar dependencies for Hibernate, JPA, and MySQL Connector Java in the pom.xml file of our Maven Project.

****

Open the pom.xml file and insert the following XML code under the <dependencies> </dependencies> tag just before the </project> tag:

| <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>org.example</groupId>  <artifactId>HibernateJPABeginner</artifactId>  <version>1.0-SNAPSHOT</version>  <packaging>jar</packaging>  <name>HibernateJPABeginner</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.source>17</maven.compiler.source>  <maven.compiler.target>17</maven.compiler.target>  </properties>  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.13.2</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.hibernate.orm</groupId>  <artifactId>hibernate-core</artifactId>  <version>6.2.6.Final</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.33</version>  </dependency>  </dependencies>  </project> |
| --- |

Here we added two dependencies for the project: hibernate-core and MySQL-connection. Maven automatically downloads the required JAR files, which are shown under the *Maven Dependencies* node in the project:

****

### 3). Create the Persistence Class (Model Class or Pojo).

* Create a package named “model” under the *src/main/Java*
* Create an entity class named “User” under the above package.
* Add the following initial code in the User.java class:

| package model;  public class User {  private Integer id;  private String fullName;  private String email;  private String password;  private int age;  private double salary;  private String city;  public User(String fullName, String email, String password, int age, double salary, String city) {  this.fullName = fullName;  this.email = email;  this.password = password;  this.age = age;  this.salary = salary;  this.city = city;  }  public User() {  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getFullName() {  return fullName;  }  public void setFullName(String fullName) {  this.fullName = fullName;  }  public String getEmail() {  return email;  }  public void setEmail(String email) {  this.email = email;  }  public String getPassword() {  return password;  }  public void setPassword(String password) {  this.password = password;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  public double getSalary() {  return salary;  }  public void setSalary(double salary) {  this.salary = salary;  }  public String getCity() {  return city;  }  public void setCity(String city) {  this.city = city;  }  } |
| --- |

As you can see above, it is just a POJO (Plain Old Java Object) class with some class variables, getter and setter methods, and its constructors.

Now, let’s use some annotations provided by JPA to map this model class to the user table in the database.

| package model;  import jakarta.persistence.\*;  @Entity  @Table(name = "USER")  public class User {  @Column(name = "USER\_ID")  @Id  @GeneratedValue(strategy = GenerationType.*IDENTITY*)  private Integer id;  private String fullName;  private String email;  private String password;  private int age;  private double salary;  private String city;  public User(String fullName, String email, String password, int age, double salary, String city) {  this.fullName = fullName;  this.email = email;  this.password = password;  this.age = age;  this.salary = salary;  this.city = city;  }  public User() {  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getFullName() {  return fullName;  }  public void setFullName(String fullName) {  this.fullName = fullName;  }  public String getEmail() {  return email;  }  public void setEmail(String email) {  this.email = email;  }  public String getPassword() {  return password;  }  public void setPassword(String password) {  this.password = password;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  public double getSalary() {  return salary;  }  public void setSalary(double salary) {  this.salary = salary;  }  public String getCity() {  return city;  }  public void setCity(String city) {  this.city = city;  }  } |
| --- |

* @Entity annotation specifies that the class is an entity.
* @Table maps the entity with the table. If no @Table is defined, the default value is used: the class name of the entity.
* @Id declares the identifier property of the entity.
* @Column maps the entity's field to the table's column. If @Column is omitted, the default value is used: the field name of the entity.

### 

### 4) Configuration File

Right-click on the resources folder **under the *src/main/*** and create a new file. Name the file “hibernate.cfg.xml” inside the resources folder. Add the following code below. You will have to enter your username and password to connect to your database.

hibernate.cfg.xml

| *<?*xml version="1.0" encoding="UTF-8"*?>*  <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd"*>*  <hibernate-configuration>  <session-factory>  *<!-- Drop and re-create the database on startup -->*  <property name="hibernate.hbm2ddl.auto"> update </property>  *<!-- Database connection settings -->*  <property name="connection.driver\_class">com.mysql.cj.jdbc.Driver</property>  <property name="connection.url">jdbc:mysql://localhost:3306/usersDb?createDatabaseIfNotExist=true</property>  <property name="connection.username">*<!--TODO Add username -->*/property>  <property name="connection.password">*<!--TODO Add password -->*</property>  *<!-- MySQL DB dialect -->*  <property name="dialect">org.hibernate.dialect.MySQLDialect</property>  *<!-- print all executed SQL on console -->*  <property name="hibernate.show\_sql" >true </property>  <property name="hibernate.format\_sql" >true </property>  *<!-- Mapping entity file -->*  <mapping class="model.User"/>  </session-factory>  </hibernate-configuration> |
| --- |

*Hibernate has many* [*configuration properties*](https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/session-configuration.html)*. Apart from the standard connection properties, it is worth mentioning the dialect property, which allows us to specify the name of the SQL dialect for the database.*

[Click here for reference document for Hibernate XML Config → *https://www.onlinetutorialspoint.com/hibernate/hbm2ddl-auto-example-hibernate-xml-config.html*](https://www.onlinetutorialspoint.com/hibernate/hbm2ddl-auto-example-hibernate-xml-config.html)

### 5) Developing CRUD Operations

We will use this application to save a few user records, and then we will apply CRUD operations to those records.

Create a **Package** called **controller** inside ***src/main/Java*.** Then create a class called **UserController.java**. It will have four methods.

| *addUser(factory,session);*  *findUser(factory,session,3);*  *updateUser(session,3);*  *deleteUser(session,4);* |
| --- |

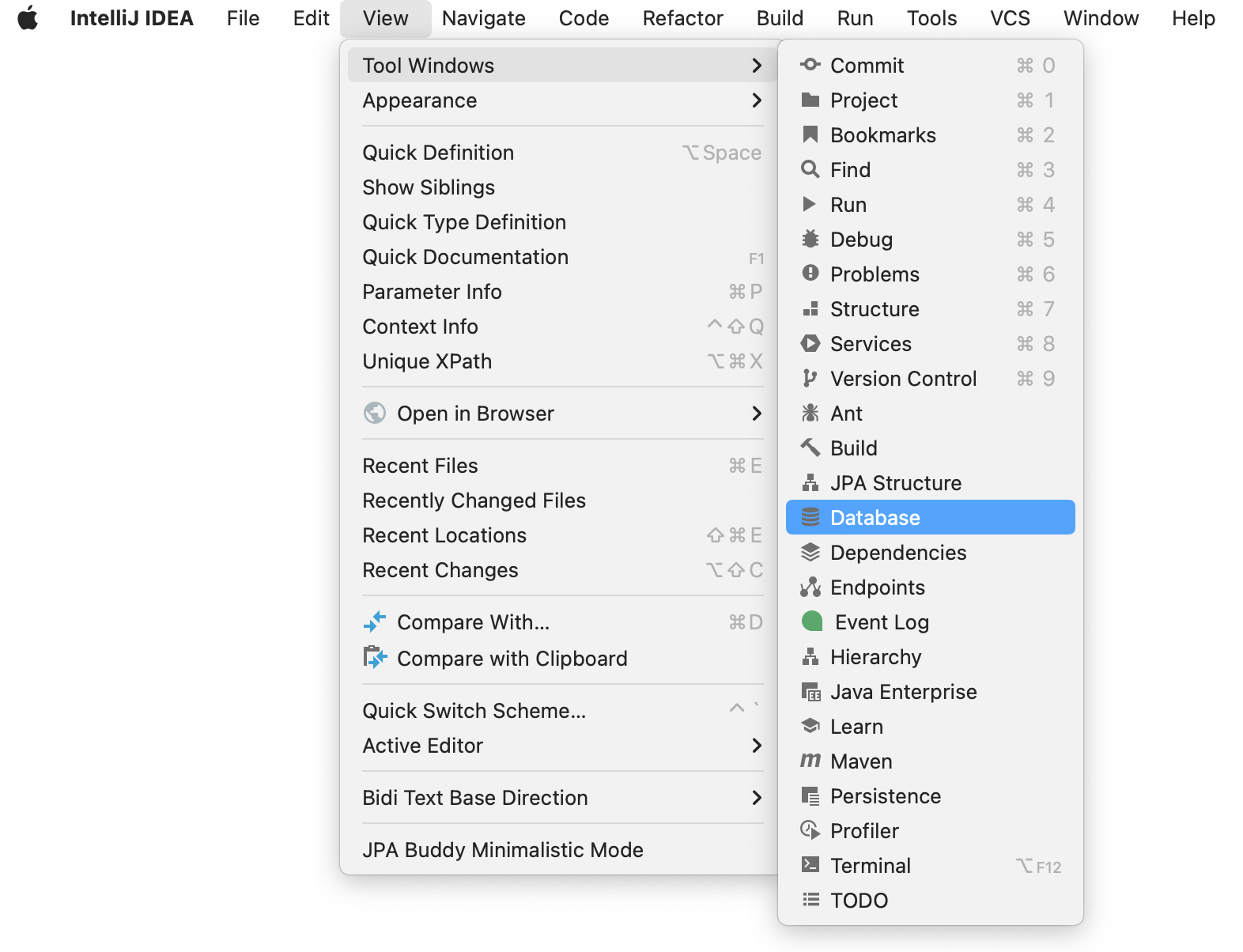
## 

## **5.1) Create a User and add the User Records to the Database**

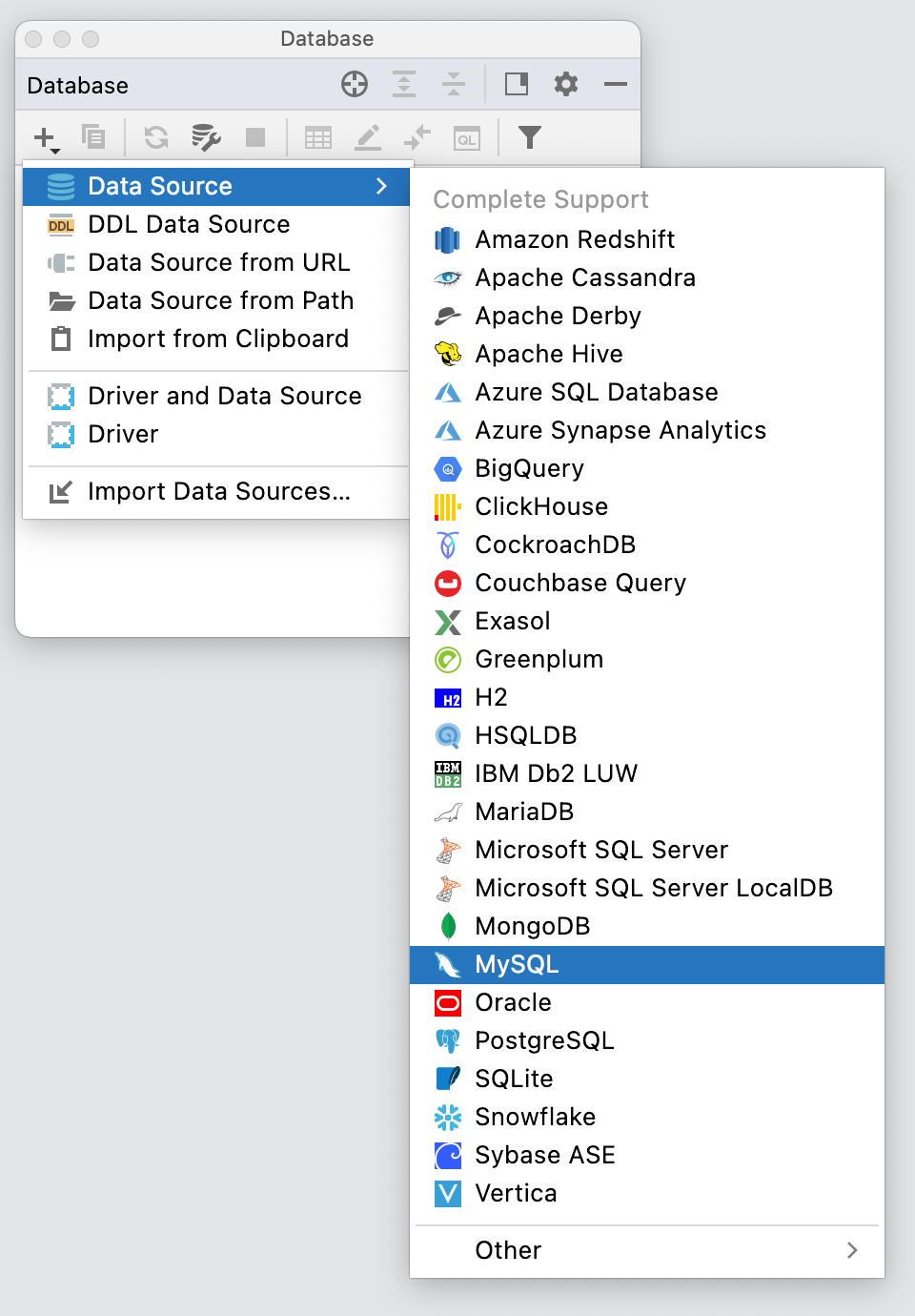
The **user** table will be generated/created in the **userDb** Database. Run the following code.

| package controller;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  public class UserController {  public static void main(String[] args) {  SessionFactory factory = new Configuration().configure().buildSessionFactory();  Session session = factory.openSession();  *// addUser(factory,session);*  *//*  *// findUser(factory,session,3);*  *//*  *// updateUser(session,3);*  *//*  *// deleteUser(session,4);*  factory.close();  session.close();  }  } |
| --- |

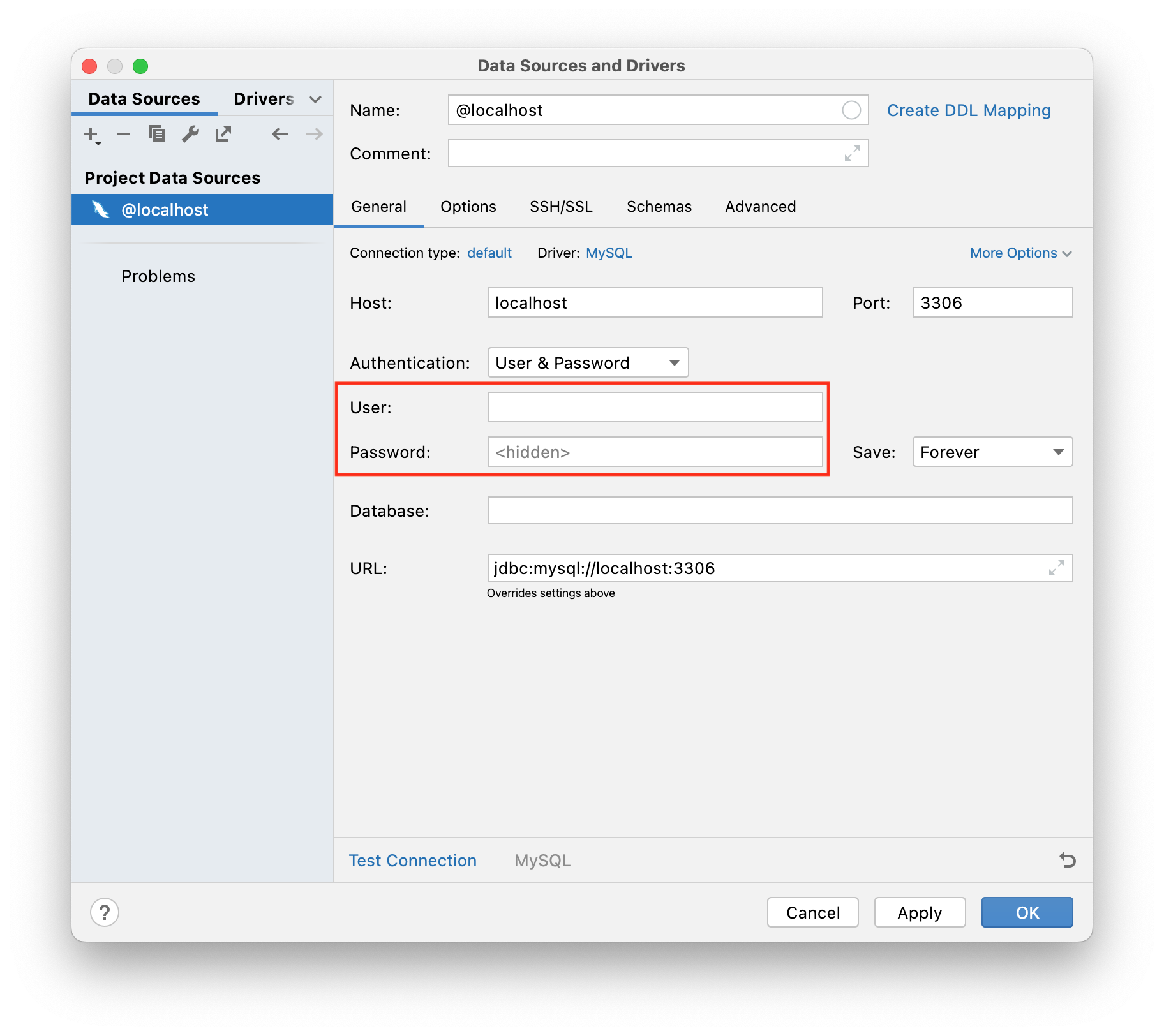
You can access your Database menu from **View** → **Tool** → **Database** or by clicking on the **Database** tab in the far right-hand corner.



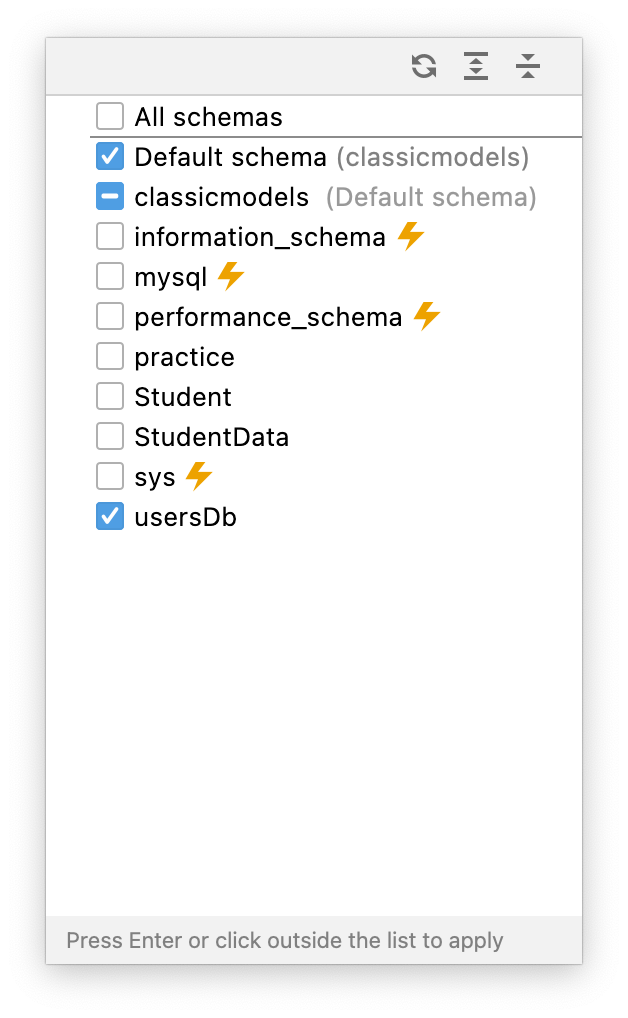
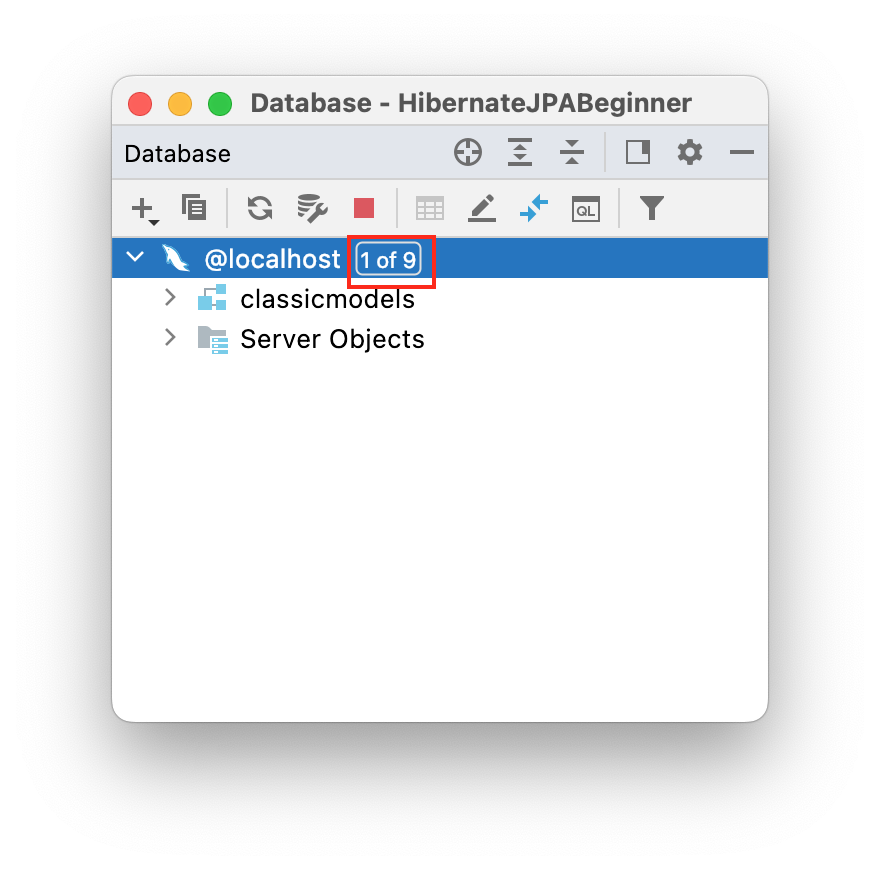
Next, click on the **+** sign to create a new data source and go to MYSQL.



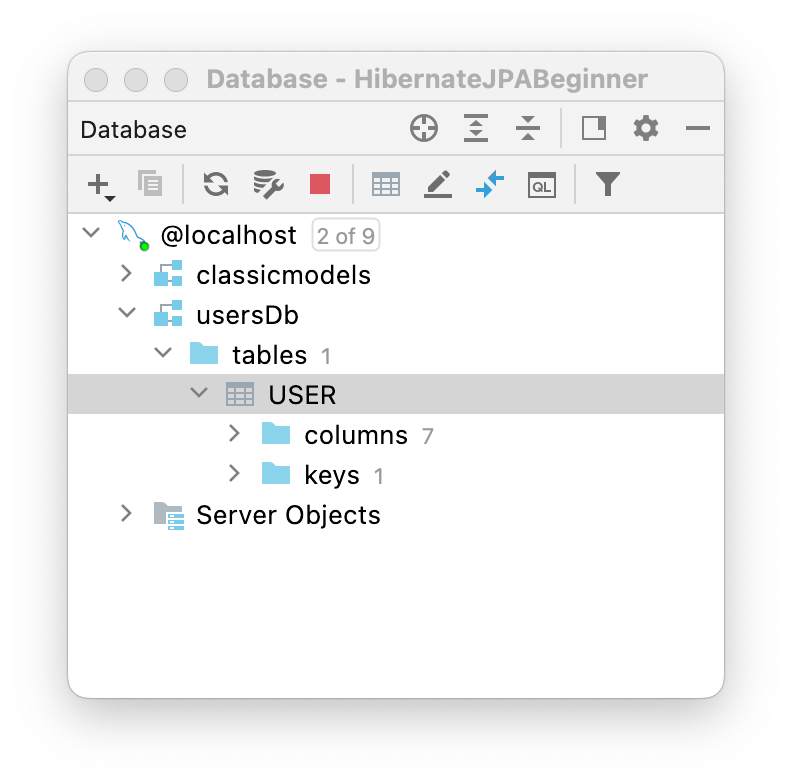
Add your User name and Password, and then click okay.



From the database tab, click on the small box next to @localhost or the name you gave your data source if it differs. Next click on usersDb, and press enter or click outside the list to apply changes.



You will be able to see your database table from the **Database** tab.

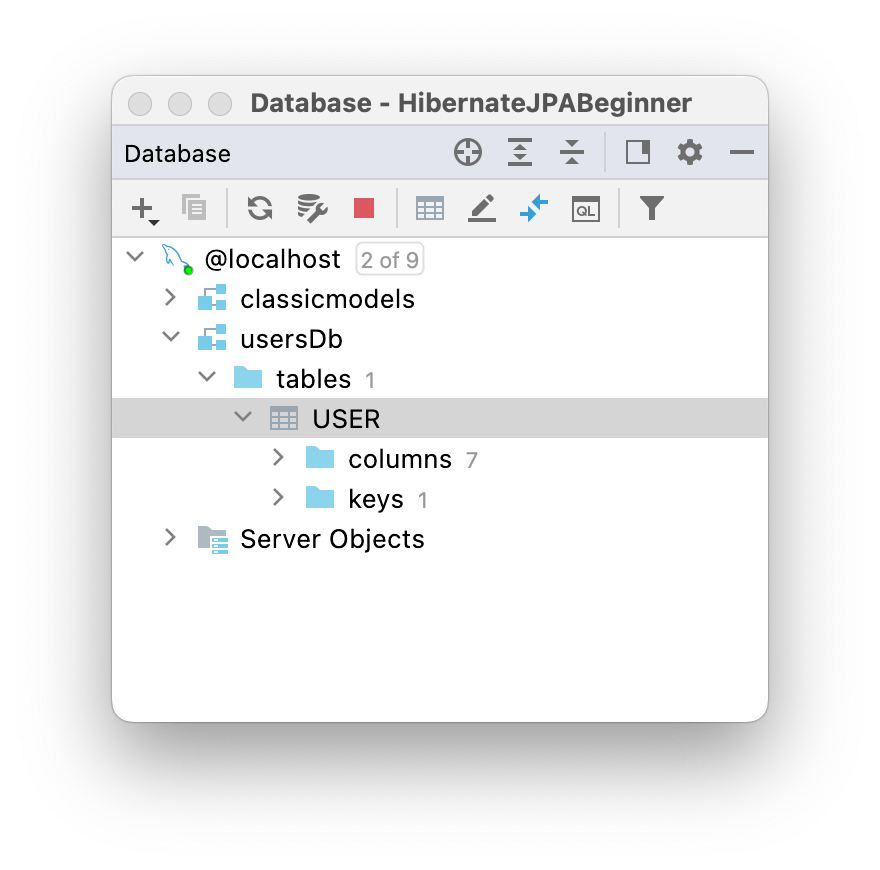


To create a User entity instance using JPA, add the following Java code and run the addUser method.

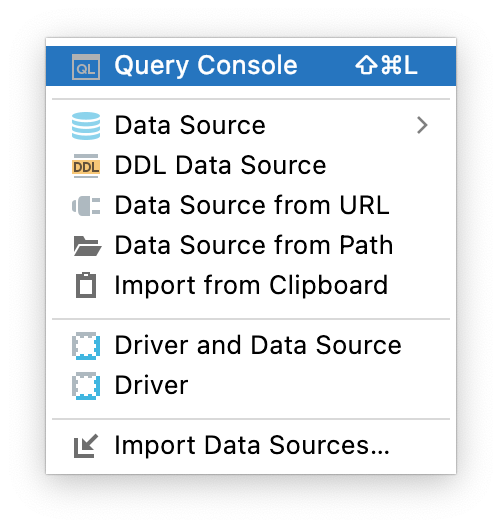
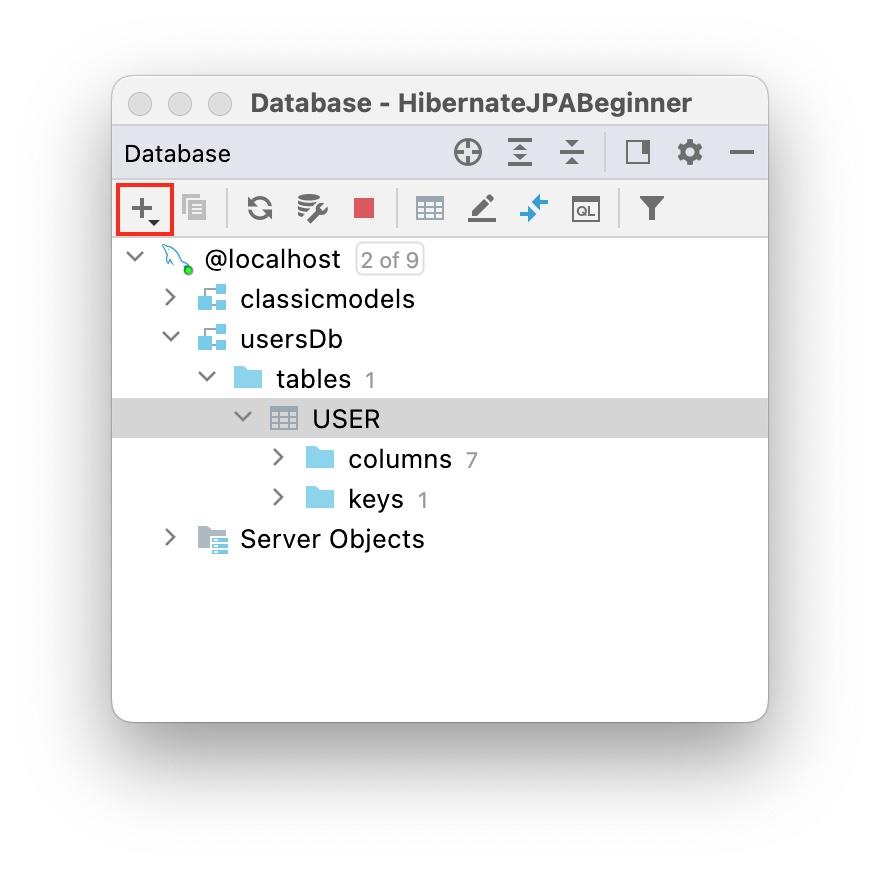
| package controller;  import model.User;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.Transaction;  import org.hibernate.cfg.Configuration;  public class UserController {  public static void main(String[] args) {  SessionFactory factory = new Configuration().configure().buildSessionFactory();  Session session = factory.openSession();  *//Todo uncomment addUser(factory,session);*  *//findUser(factory,session,2);*  *// updateUser(session,3);*  *//deleteUser(session,4);*  }  public static void addUser(SessionFactory factory, Session session) {  Transaction transaction = session.beginTransaction();  User uOne = new User();  uOne.setEmail("haseeb@gmail.com");  uOne.setFullName("Moh Haseeb");  uOne.setPassword("has123");  uOne.setSalary(2000.69);  uOne.setAge(20);  uOne.setCity("NYC");  User uTwo = new User();  uTwo.setEmail("James@gmail.com");  uTwo.setFullName("James Santana");  uTwo.setPassword("James123");  uTwo.setSalary(2060.69);  uTwo.setAge(25);  uTwo.setCity("Dallas");  User uThree = new User();  uThree.setEmail("Shahparan@gmail.com");  uThree.setFullName("AH Shahparan");  uThree.setPassword("Shahparan123");  uThree.setSalary(3060.69);  uThree.setAge(30);  uThree.setCity("Chicago");  */\*========= We can pass value/data by using constructor =========\*/*  User uFour = new User("Christ", "christ@gmail.com", "147852", 35, 35000.3, "NJ");  User uFive = new User("Sid", "Sid", "s258", 29, 4000.36, "LA");  *//Integer userid = null;*  session.persist(uOne);  session.persist(uTwo);  session.persist(uThree);  session.persist(uFour);  session.persist(uFive);  transaction.commit();  System.*out*.println("successfully saved");  factory.close();  session.close();  }  } |
| --- |

For the output, you can use either of these options:

1. Go to your database and double click on User.

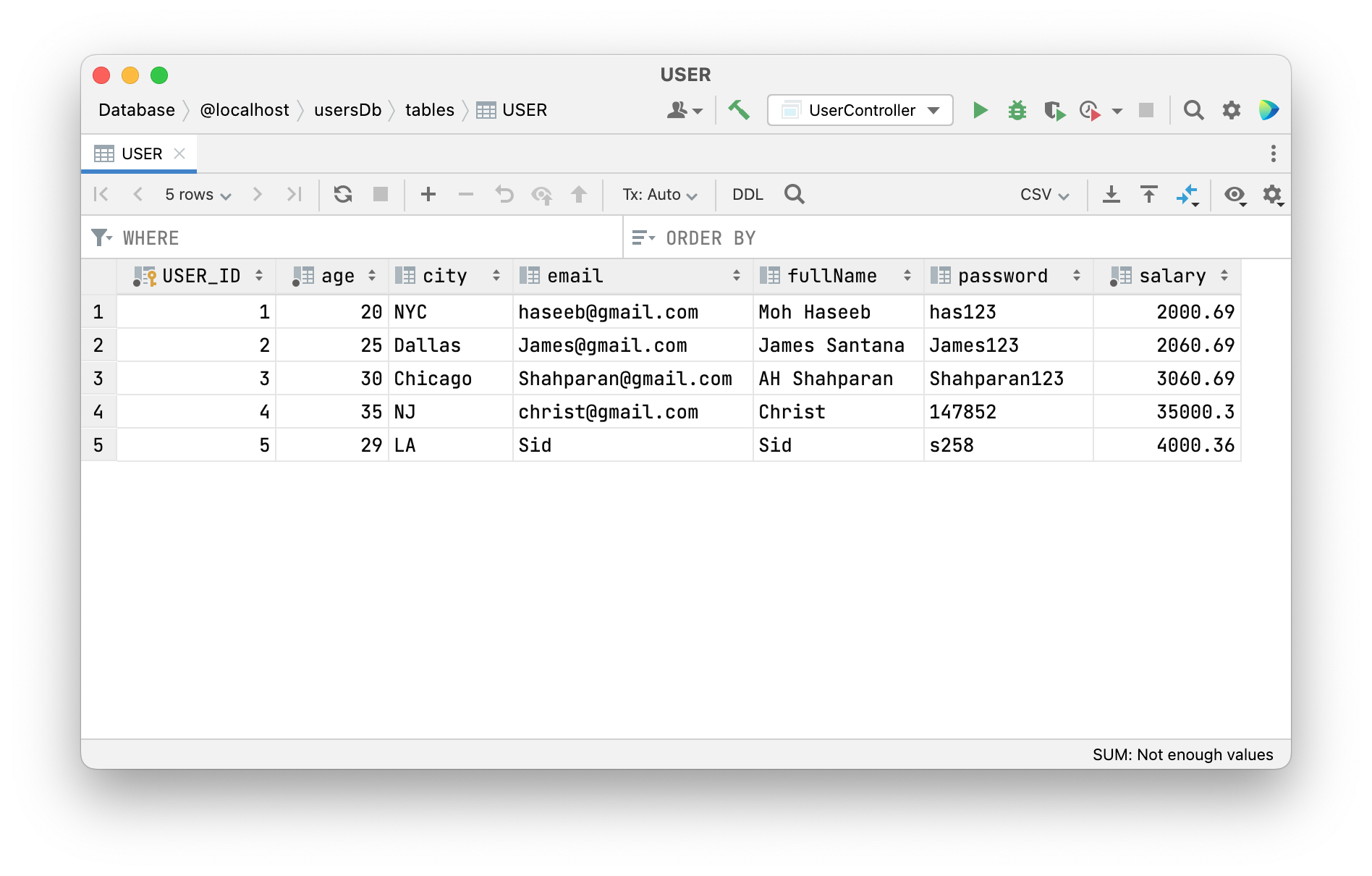


1. Click on the + sign and create a new Query Console.



Type the following queries.

| USE usersDb;  SELECT *\** FROM USER; |
| --- |

**Result:**

### 

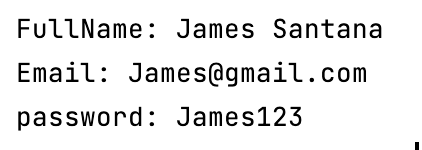
### 6) Find a User

The **session.get(Class, id)** returns an object of the specified class that maps a row in the database table. If no row is found, it returns null. Add the method below:

| public static void findUser(SessionFactory factory,Session session,int userId){  *// Todo comment out addUser method and uncomment findUser method*  Transaction tx = session.beginTransaction();  User u = session.get(User.class, userId);  System.*out*.println("FullName: " + u.getFullName());  System.*out*.println("Email: " + u.getEmail());  System.*out*.println("password: " + u.getPassword());  *//Close resources*  tx.commit();  factory.close();  session.close();  } |
| --- |

Before you Run the method, ***findUser*()** and ***addUser()****.* Comment out the *method* addUser() from the main class.

**Output**:



### 7) Updating a User record

**Session.update() and session.merge()** are both used for updating the rows of data in the database.

**Session.update()**: If you are sure that the session does not contain an already persistent instance with the same identifier, use **update** to save the data in hibernate.

**Session.merge():** If you want to save your modifications at any time without knowing about the state of a session, use merge() in hibernate.

Add the method below:

| public static void updateUser(Session session, int userId) {  *// Todo comment out findUser method and uncomment updateUser method*  Transaction tx = session.beginTransaction();  User u = new User();  u.setId(userId);  u.setEmail("mhaseeb@perscholas");  u.setFullName("M Haseeb");  u.setPassword("123456");  session.merge(u);  session.getTransaction().commit();  session.close();  } |
| --- |

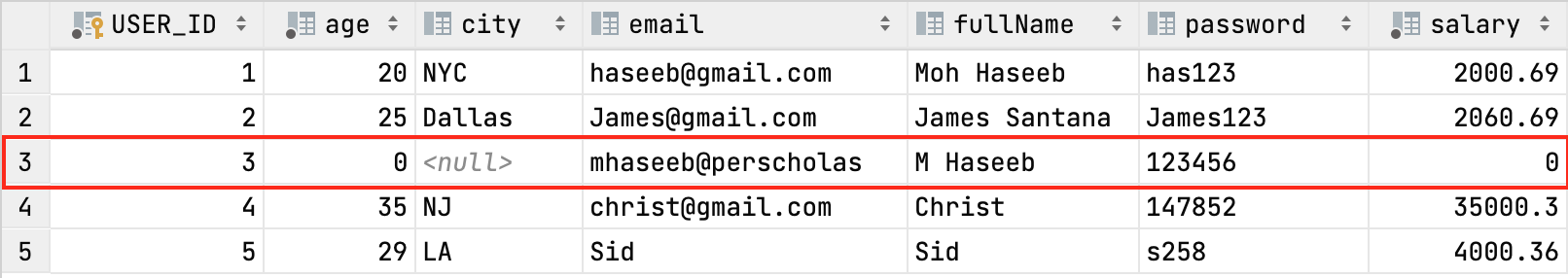
Before you run the method updateUser(), comment out the method find user.

For the output, you have two options.

1. Go to your database and double click on User.
2. Run the following query in your query console.

| USE usersDb;  SELECT *\** FROM USER; |
| --- |

Result:

****

### 8) Performing DELETE operation:

We will call the **session.remove(Object)** method to remove a mapped object from the database. Add the method below.

| public static void deleteUser(Session session,int userId){  *// Todo comment out updateUser method and uncomment deleteUser method*  SessionFactory factory = new Configuration().configure().buildSessionFactory();  Transaction tx = session.beginTransaction();  User u = new User();  u.setId(userId);  session.remove(u);  tx.commit();  session.close();  factory.close();  } |
| --- |

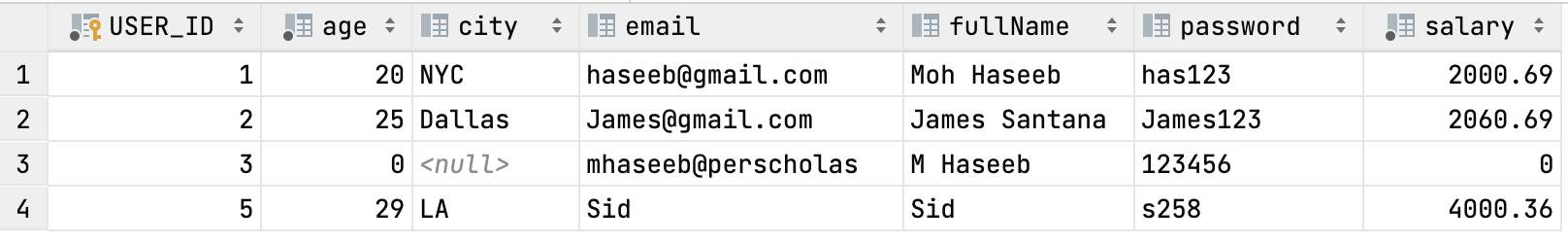
Before you run the method deleteUser(), comment out the method updateUser.

For the output,you have two options.

1. Go to your database and double click on User.
2. Run the following query in your query console.

| USE usersDb;  SELECT *\** FROM USER; |
| --- |

Result:



**Reference:**

<https://www.objectdb.com/api/java/jpa/Query>

<https://stackoverflow.com/questions/37066024/what-is-the-mariadb-dialect-class-name-for-hibernate>

<https://docs.jboss.org/hibernate/core/3.3/reference/en-US/html/objectstate.html>

**Submission Instructions:**

Include the following deliverables in your submission -

* + Submit your source code or screenshot using the Start Assignment button in the top-right corner of the assignment page in Canvas.

**CANVAS STAFF USE ONLY: Canvas Submission Guideline:**

| **Instructions for Canvas Assignment Creation** |
| --- |
| **Assignment Name: Guided Lab 305.4.1 Hibernate Project Demonstration with IntelliJ Ultimate**  **Points:** **100**  **Assignment Group: Module 303: Java SE Review (Not Graded)**  **Display Grade As: Complete/Incomplete**  **Do not count this assignment towards the final grade: Checked**  **Submission Types: File Uploads**  **Everything else is the default.** |

## 