# Guided LAB 305.4.3B - Demonstration - Named Queries in Hibernate with IntelliJ Ultimate

**Lab Overview of the Demonstration**

In Hibernate, we can define a named query either in the Hibernate mapping file or in an entity class. We will use the entity class.

**For database**: It is your choice; you can use the **userDb**database, as we used in previous exercises, or you can use any other database.

This example is split into two parts. In **Part one**, we will set up a new Hibernate project and create a table using Hibernate. In **Part two**, we will utilize ***NamedQueries***.

Annotations @NamedQuery and @NamedQueries are used for HQL expressions, whereas @NamedNativeQuery and @NamedNativeQueries are used for native SQL expressions.

This demonstration shows you how to use the Named Queries annotations in Hibernate applications.

**Learning Objective:**

By the end of this lab, learners will be able to use Named Queries in Java Hibernate.

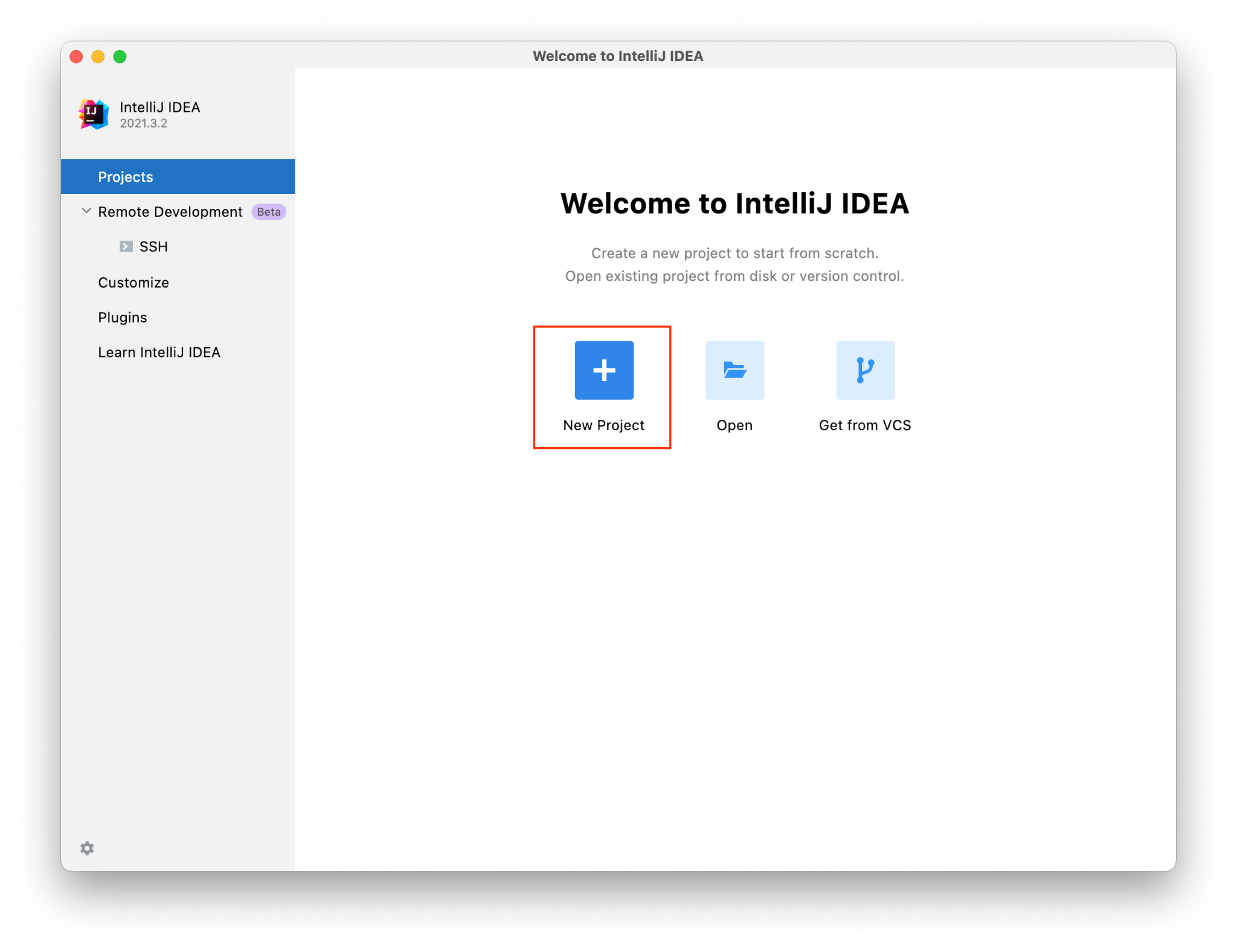
# 

# 

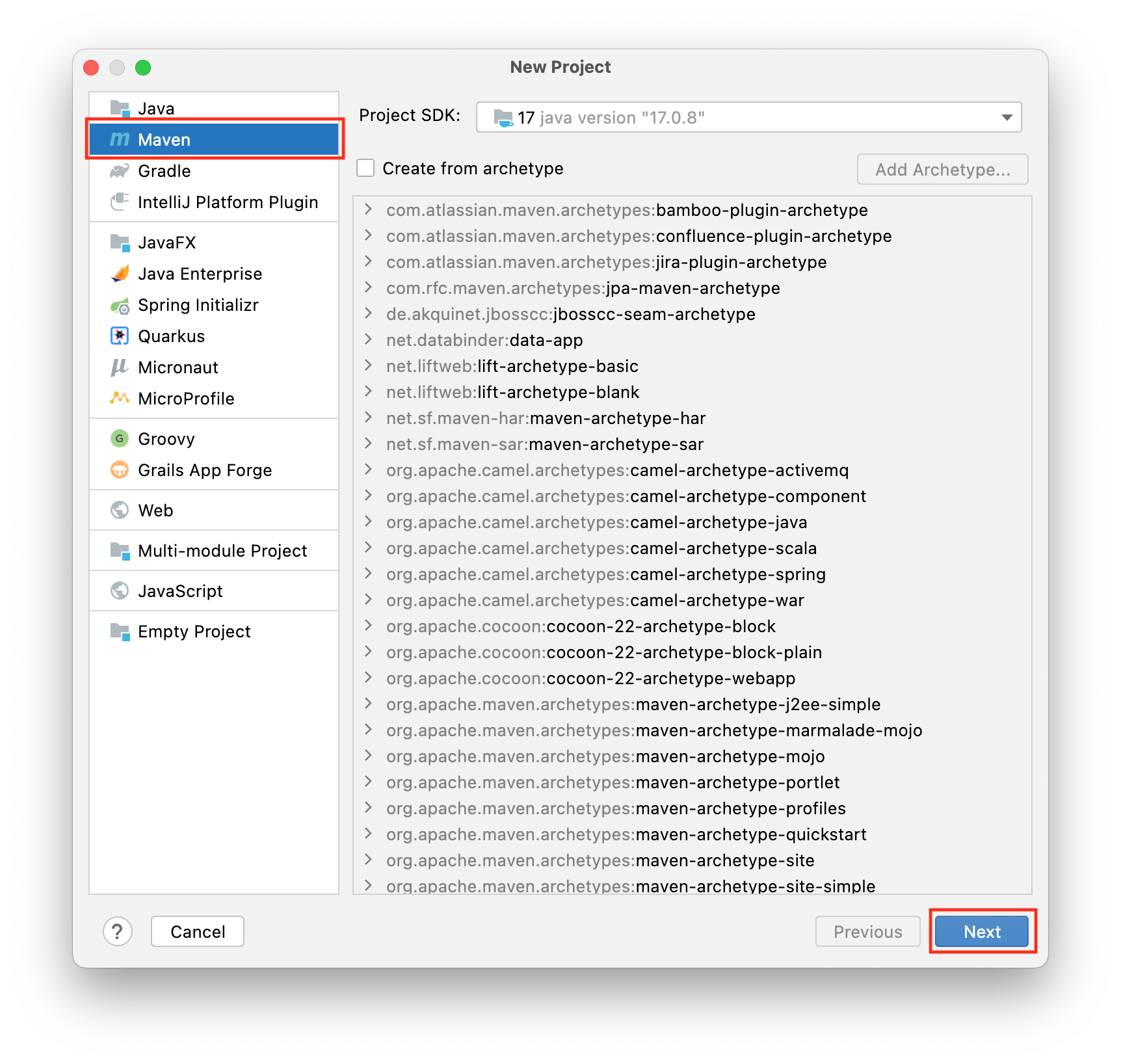
# Part One

### **1) Create the Maven Project**

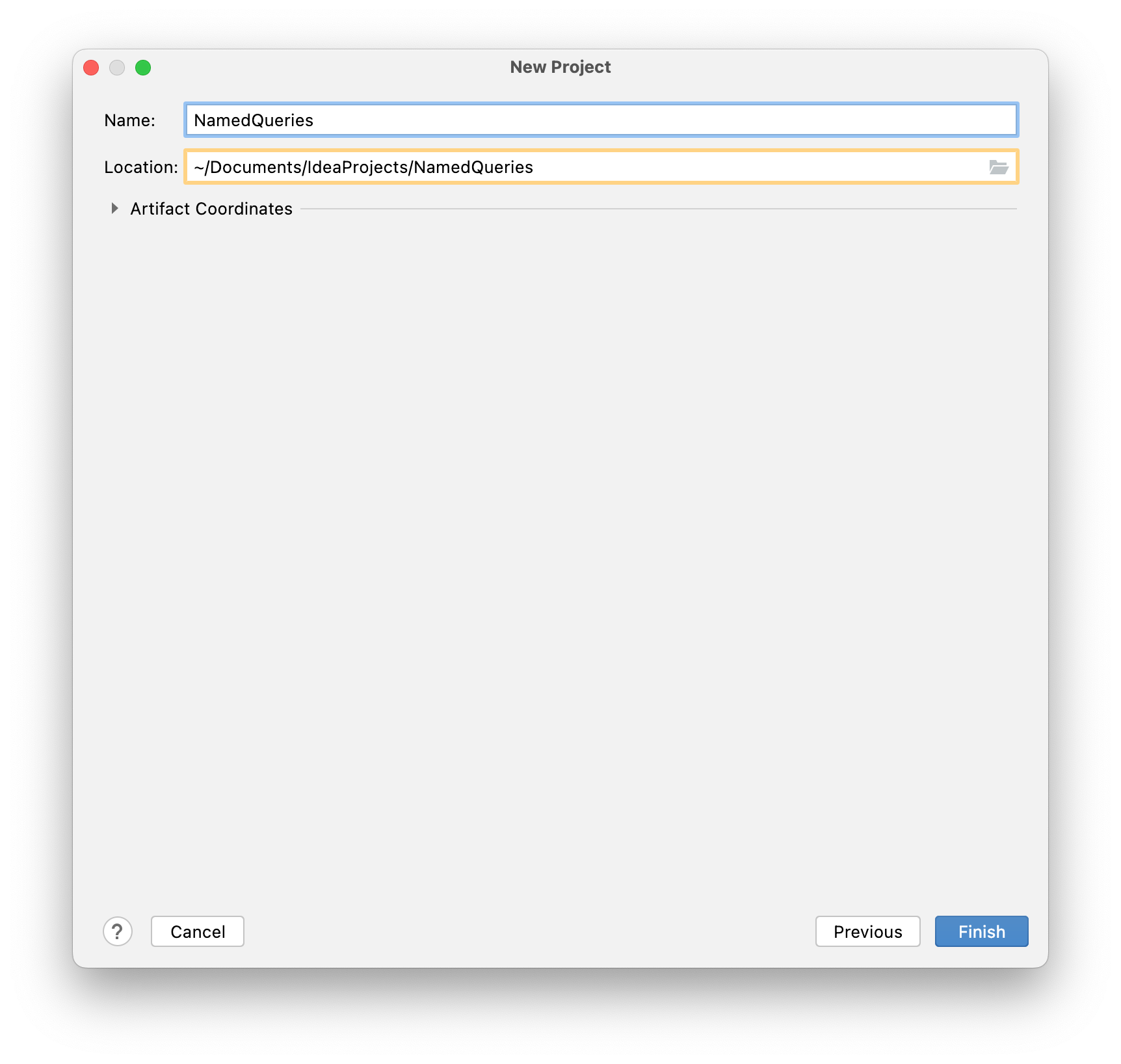
* To create a Maven project click on New Project.

****

* Select Maven and click Next.



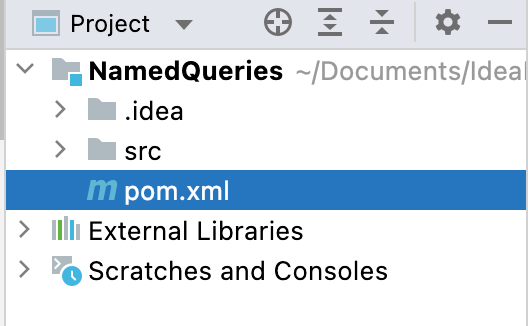
* Name your project **NamedQueries** and click Finish.



### **2) Add Jar dependencies, and configuration in the pom.xml file.**

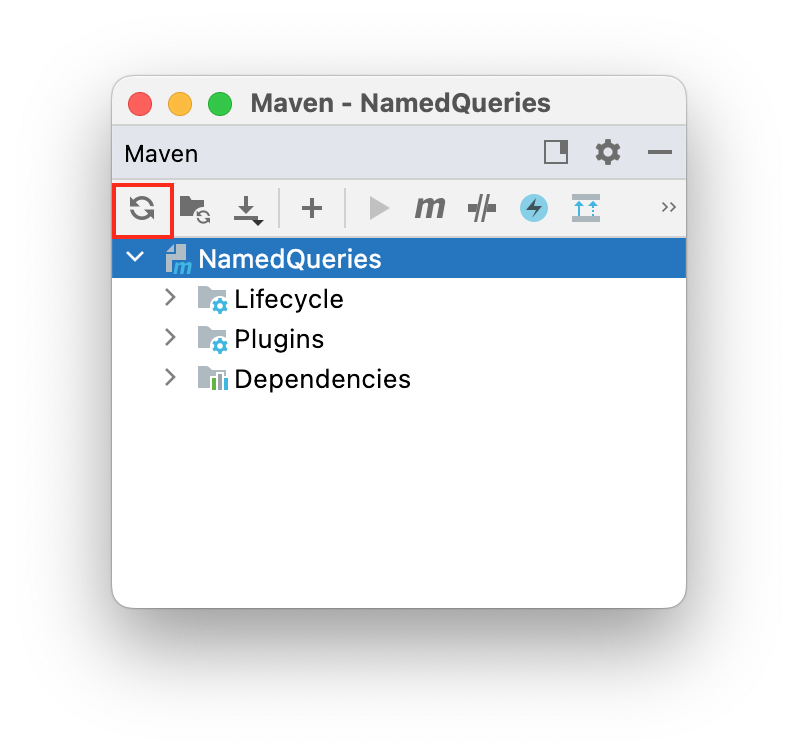
Next, we need to add a couple of jar dependencies for Hibernate, JPA, and MySQL Connector

Java in the pom.xml file of our Maven Project.

****

Open the pom.xml file and insert the following code:

| | *<?*xml version="1.0" encoding="UTF-8"*?>*  <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>NamedQueries</artifactId>  <version>1.0-SNAPSHOT</version>  <properties>  <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> | | --- |   You should see an M icon in the far right-hand corner. Click on it to Load Maven Changes.    Or you can open the Maven tab in the right-hand corner and load the changes by clicking the on the icon in the left-hand corner. |
| --- | --- |



|  |
| --- |

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

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

| package model;  import jakarta.persistence.\*;  import java.util.Date;  @Entity  @Table  public class Employee {  @Id  @GeneratedValue(strategy = GenerationType.*IDENTITY*)  private int id;  private String name;  private int salary;  private String job;  private String addressLine;  private String zipcode;  private String city;  private Date startDate;  private int officeCode;  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 int getSalary() {  return salary;  }  public void setSalary(int salary) {  this.salary = salary;  }  public String getJob() {  return job;  }  public void setJob(String job) {  this.job = job;  }  public String getAddressLine() {  return addressLine;  }  public void setAddressLine(String addressLine) {  this.addressLine = addressLine;  }  public String getZipcode() {  return zipcode;  }  public void setZipcode(String zipcode) {  this.zipcode = zipcode;  }  public String getCity() {  return city;  }  public void setCity(String city) {  this.city = city;  }  public Date getStartDate() {  return startDate;  }  public void setStartDate(Date startDate) {  this.startDate = startDate;  }  public int getOfficeCode(){  return officeCode;  }  public void setOfficeCode(int officeCode) {  this.officeCode = officeCode;  }  @Override  public String toString() {  return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + ", job=" + job + ", addressLine="  + addressLine + ", zipcode=" + zipcode + ", city=" + city + ", startDate=" + startDate + "]";  }  } |
| --- |

## 

### **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.

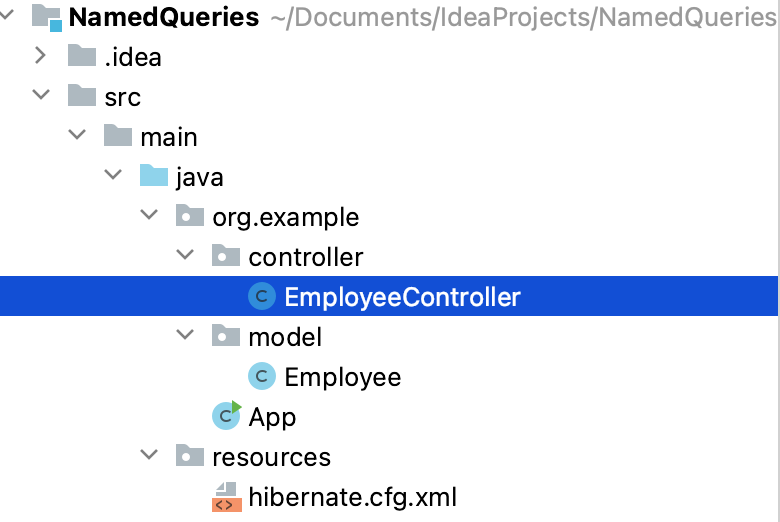
hibernate.cfg.xml

| *<?*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</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.Employee"/>  </session-factory>  </hibernate-configuration> |
| --- |

### 

### **5) Create a table in the database using Hibernate.**

Create a **Package** called **controller** inside ***src/main/java/*** **.** Then create a class called **EmployeeController.java**.

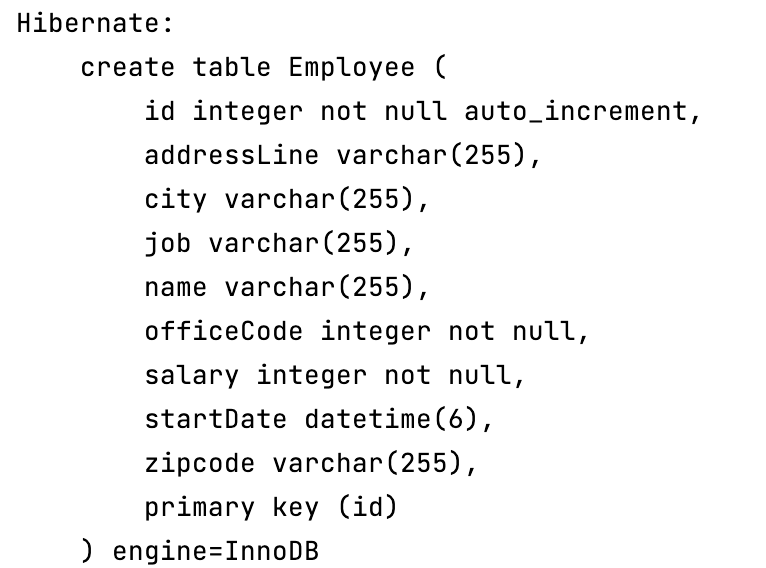


Add the following code to the “EmployeeController” class. The code below will generate **employee** tables in the ***userDb*** database.

| package controller;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  public class EmployeeController {  public static void main(String[] args) {  SessionFactory factory = new Configuration().configure().buildSessionFactory();  Session session = factory.openSession();  factory.close();  session.close();  }  } |
| --- |

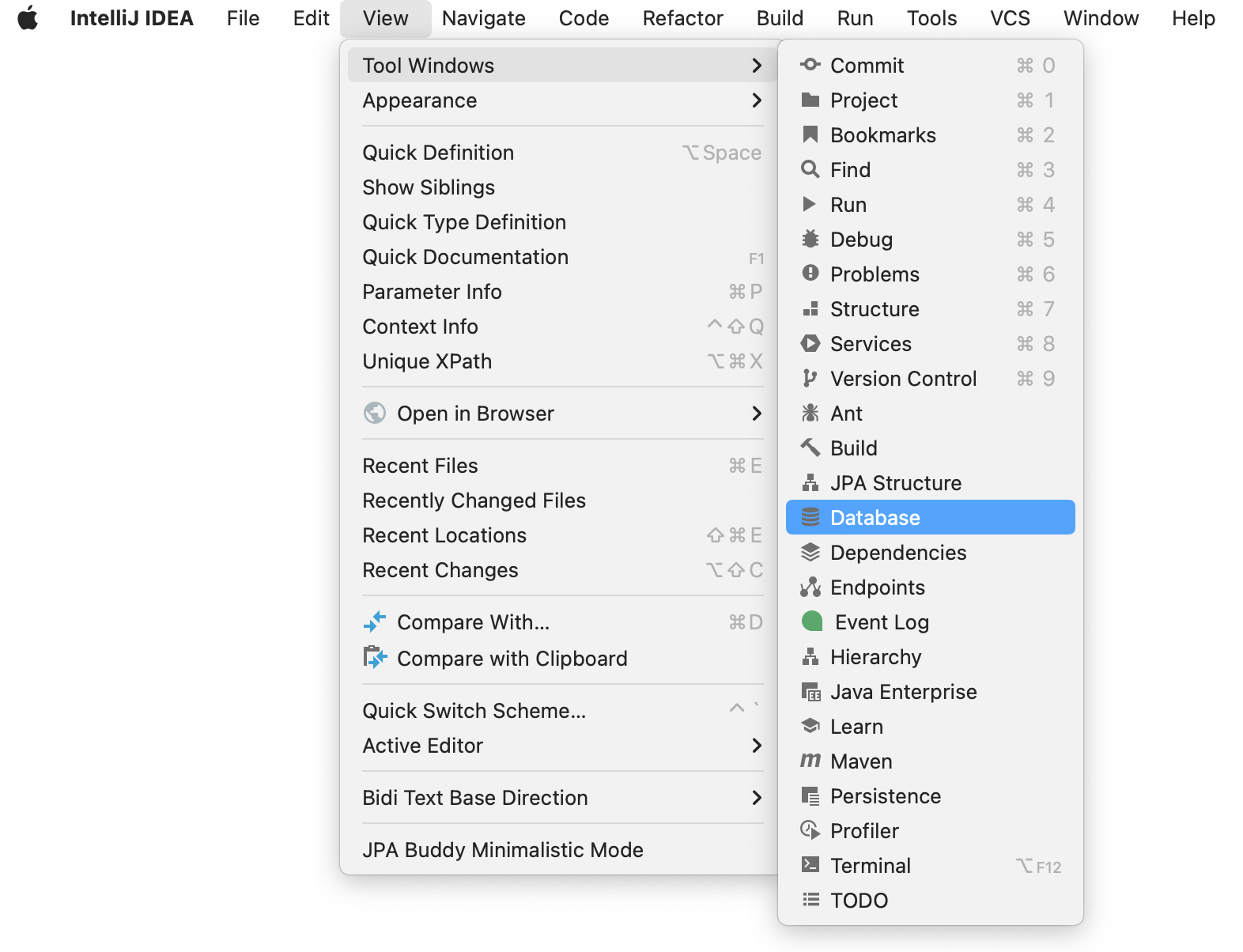
### 6) Run an Application.

Run your code and you will see the following output in your console:

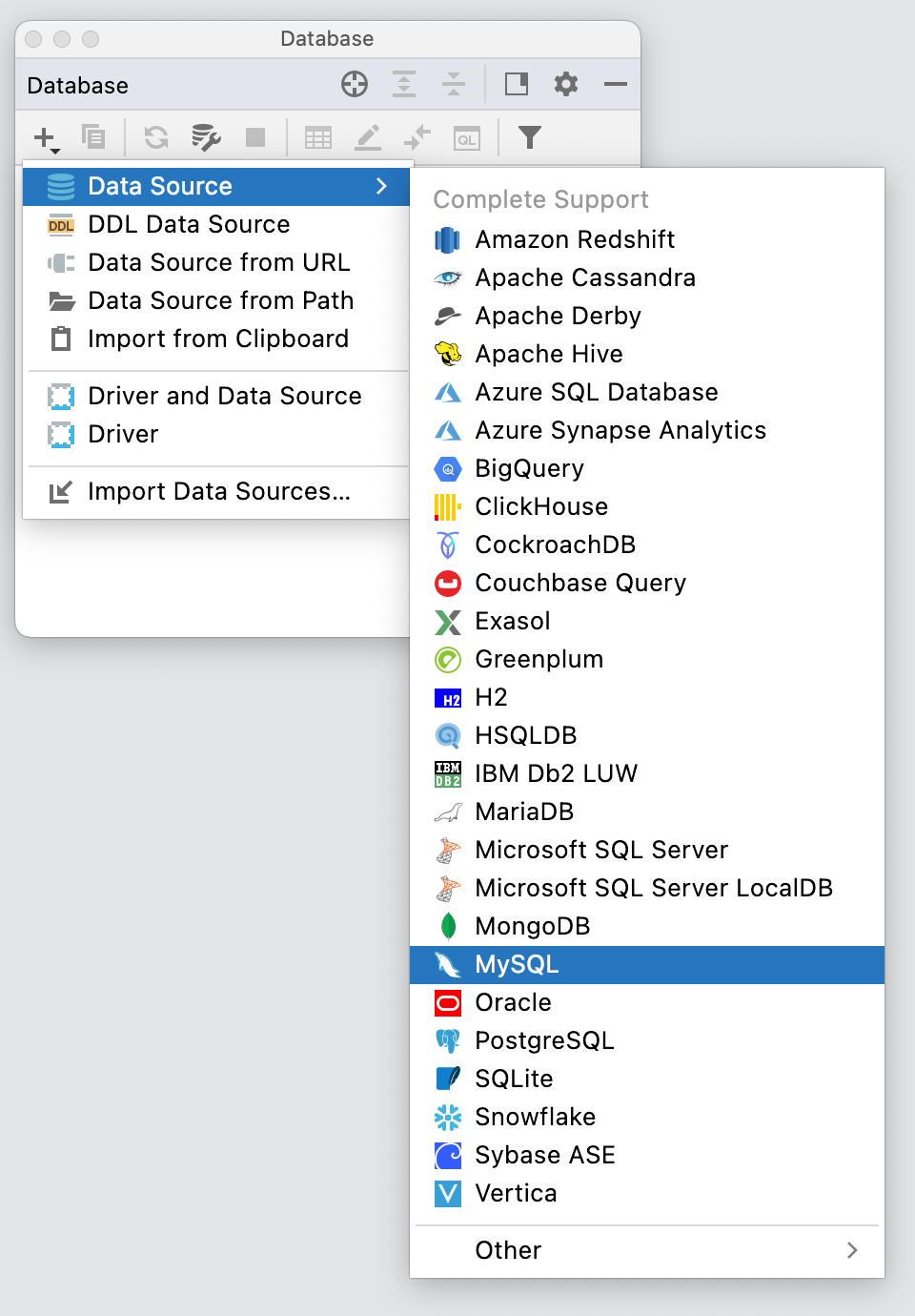


### 7) Access your database

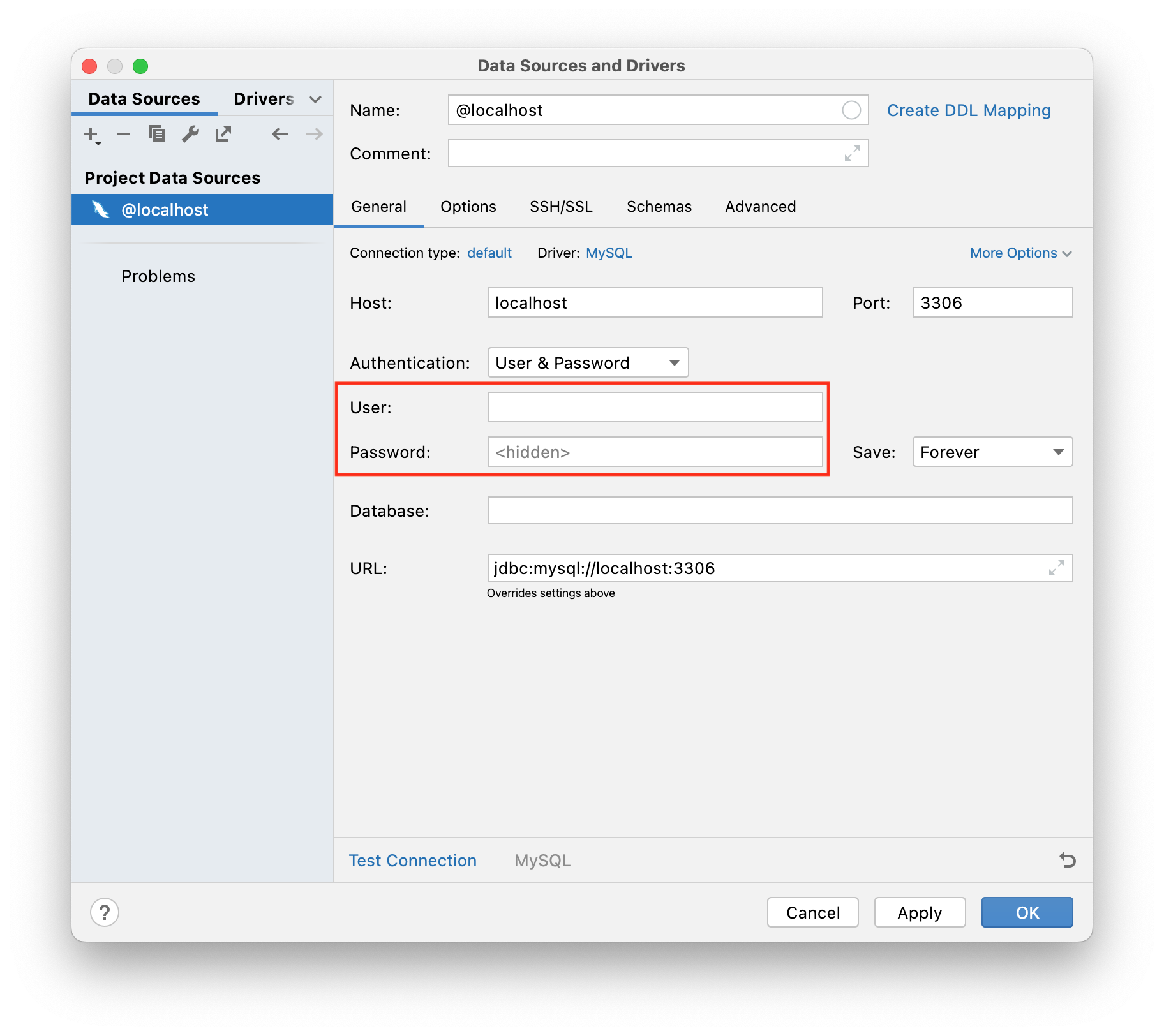
You can access your Database menu from **View** → **Tool** → **Database** or by clicking on the **Database** tab in the far right-hand corner.



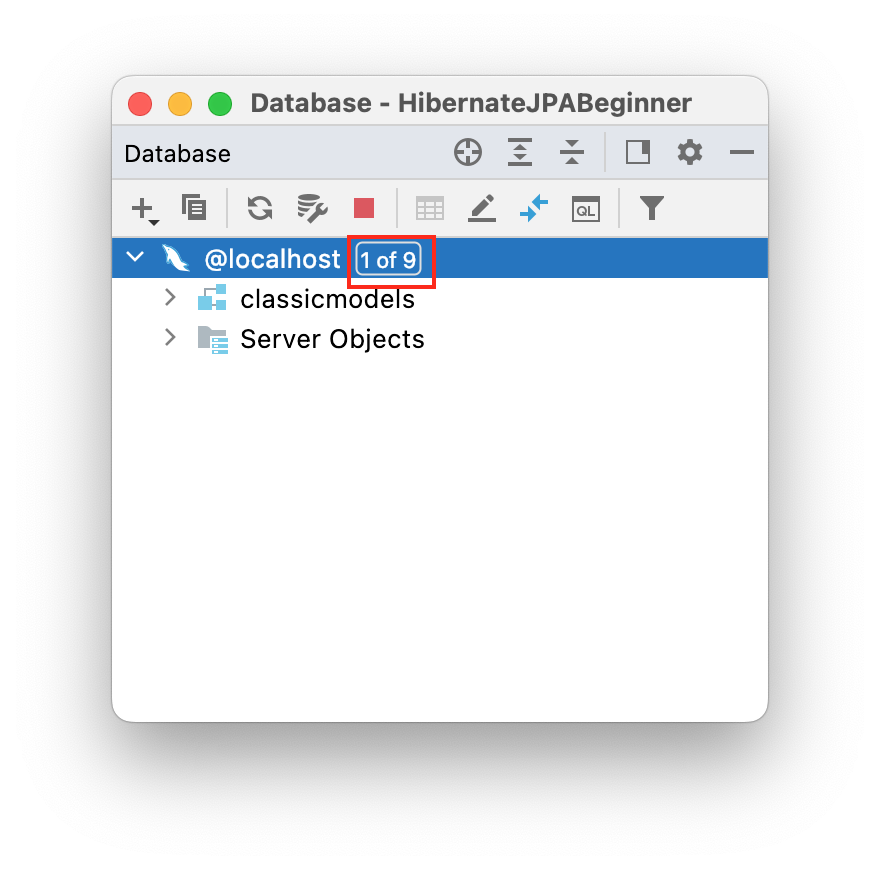
Next click on the + sign to create a new data source and go to MySQL.



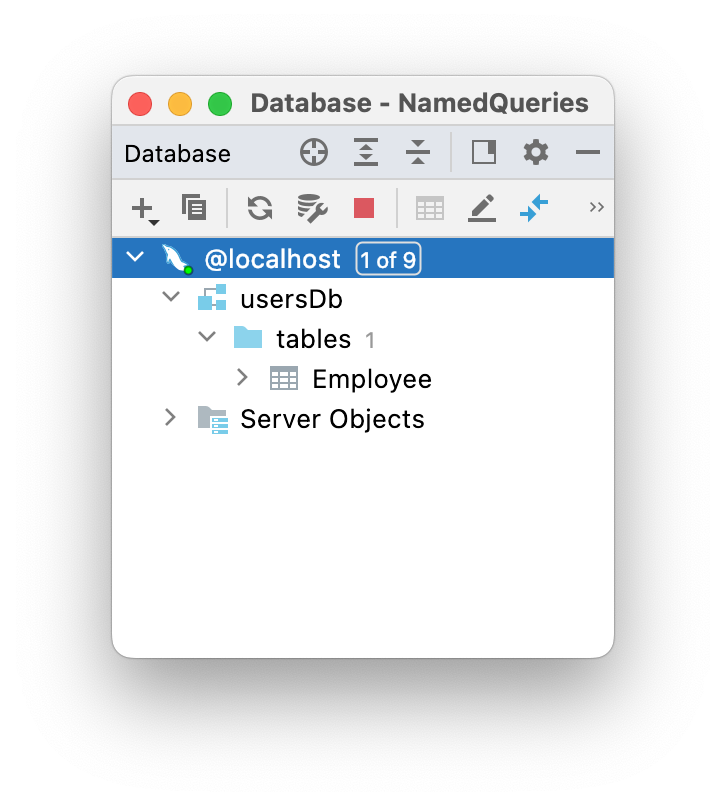
Add your User name and Password, and then click okay.



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.

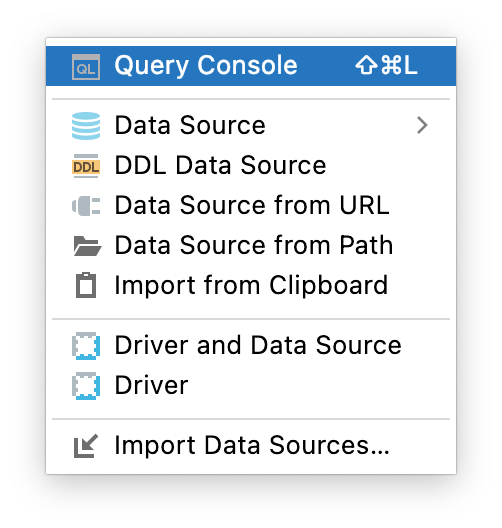
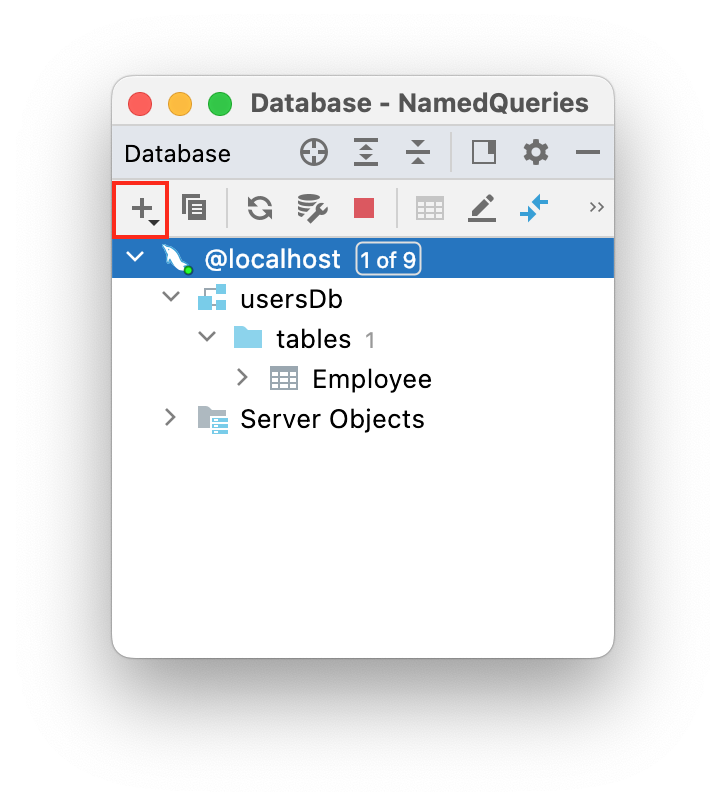


You will be able to see your database table from the **Database** tab.



### 8) Data insertion using the console.

Open your console if it is not already open by clicking on the + sign. Next, click on Query Console.



In your console, add the following statements.

| INSERT INTO `usersDb`.`Employee` (`addressLine`, `city`, `job`, `name`, `officeCode`, `salary`, `startDate`, `zipcode`)  VALUES ('24-10', 'NYC', 'Ceo', 'Tom Thele', 1, '25600', '2021-09-09 18:32:11.000000', '11102'),  ('35-16', 'LA', 'Manager', 'Jenny Ji', 2, '15600', '2021-09-09 18:32:11.000000', '11103'),  ('34-10', 'NJ', 'Cfo', 'M Joseph', 3, '16600', '2021-09-09 18:32:11.000000', '11583'),  ('44-20', 'NYC', 'Manager', 'James Judge', 4, '185600', '2021-09-09 18:32:11.000000', '18983'),  ('44-20', 'Dallas', 'Manager', 'Ramon Rio', 5, '36600', '2021-09-09 18:32:11.000000', '14783'),  ('44-40', 'LA', 'Manager', 'James Santana', 6, '78600', '2021-09-09 18:32:11.000000', '18483'),  ('484-40', 'LA', 'Manager', 'Elded Oreo', 6, '50089', '2021-09-09 18:32:11.000000', '155483'); |
| --- |

# 

Execute the query by clicking on the green play button on the left-hand side.

# Part Two

## **Using @NamedQuery and @NamedQueries annotations**

Open the “**Employee**” entity class. In this class, we will define @NamedQuery and @NamedQueries annotations for using single or multiple named HQL query expressions as shown below.

| package model;  import jakarta.persistence.\*;  import java.util.Date;  @Entity  @Table  *//Using @NamedQuery for single HQL*  @NamedQuery(name = "getAllEmployees", query = "select e from Employee e")  *//Using @NamedQueries for multiple HQL*  @NamedQueries({  @NamedQuery(name = "getEmployeeById", query = "select e.name,e.salary,e.job from Employee e where id=:id"),  @NamedQuery(name = "getAllDepartments", query = "from Employee"),  @NamedQuery(name = "findEmployeeByName", query = "from Employee e where e.name = :name"),  @NamedQuery(name = "employeeDeptAlias", query = "select e, e.officeCode , e.name, " +  "CASE " +  "When (e.officeCode = 1) THEN 'IT'"  + "WHEN (e.officeCode = 6) THEN 'Admin'"  + "WHEN (e.officeCode = 5) THEN 'HR'"  + "WHEN (e.officeCode = 4) THEN 'Developers'"  + "WHEN(e.officeCode = 3) THEN 'Accounts'"  + "WHEN (e.officeCode =2) THEN 'Finance'"  + "ELSE 'General' END FROM Employee e "),  })  public class Employee {  @Id  @GeneratedValue(strategy = GenerationType.*IDENTITY*)  private int id;  private String name;  private int salary;  private String job;  private String addressLine;  private String zipcode;  private String city;  private Date startDate;  private int officeCode;  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 int getSalary() {  return salary;  }  public void setSalary(int salary) {  this.salary = salary;  }  public String getJob() {  return job;  }  public void setJob(String job) {  this.job = job;  }  public String getAddressLine() {  return addressLine;  }  public void setAddressLine(String addressLine) {  this.addressLine = addressLine;  }  public String getZipcode() {  return zipcode;  }  public void setZipcode(String zipcode) {  this.zipcode = zipcode;  }  public String getCity() {  return city;  }  public void setCity(String city) {  this.city = city;  }  public Date getStartDate() {  return startDate;  }  public void setStartDate(Date startDate) {  this.startDate = startDate;  }  public int getOfficeCode() {  return officeCode;  }  public void setOfficeCode(int officeCode) {  this.officeCode = officeCode;  }  @Override  public String toString() {  return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + ", job=" + job + ", addressLine="  + addressLine + ", zipcode=" + zipcode + ", city=" + city + ", startDate=" + startDate + "]";  }  } |
| --- |

Go to the “EmployeController” class and add the following methods. The class consists of three methods named *findEmployeeById(), findEmployeeByName()* and *ShowOfficeCodesAsDepartment(),* and each method uses NamedQuery*.*

The following example demonstrates how to write and execute a query using named parameters.

| **EmployeController class**   | import java.util.List;  import jakarta.persistence.TypedQuery;  import org.example.model.Employee;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import jakarta.persistence.EntityManagerFactory;  import jakarta.persistence.Query;  import jakarta.persistence.TypedQuery;  import org.example.model.Employee;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.Transaction;  import org.hibernate.cfg.Configuration;  public class EmployeeController {  public static void main(String[] args) {  // *findEmployeeByName*();  // *findEmployeeById*();  // *showOfficeCodesAsDepartment*();  }  public static void findEmployeeByName() {  SessionFactory factory = new Configuration().configure().buildSessionFactory();  Session session = factory.openSession();  *//------------ Hibernate Named Query -------------*  TypedQuery<Employee>query = session.createNamedQuery("findEmployeeByName", Employee.class);  query.setParameter("name","Tom Thele");  List<Employee> employees=query.getResultList();  System.*out*.printf("%-10s%-22s%-14s%-14s%-16s%-10s%-10s%s%n", "ID","Name","Salary", "Job","AddressLine","Zipcode","city","startDate");  for (Employee e : employees) {  System.*out*.printf("%-11d%-22s%-14d%-14s%-16s%-10s%-10s%s%n",e.getId(),e.getName(),e.getSalary(),e.getJob(),e.getAddressLine(),e.getZipcode(),e.getCity(),e.getStartDate());  }  factory.close();  session.close();  }  public static void findEmployeeById() { SessionFactory factory = new Configuration().configure().buildSessionFactory();  Session session = factory.openSession();  *//------------ Hibernate Named Query -------------*  TypedQuery<Object[]> query = session.createNamedQuery("getEmployeeById", Object[].class);  query.setParameter("id",3);  List<Object[]> employee= query.getResultList();  System.*out*.printf("%-10s%-12s%s%n", "Name","Salary", "Job");  for (Object[] e : employee) {  System.*out*.printf("%-11s%-12s%s%n",e[0],e[1],e[2]);  }  factory.close();  session.close();  }  public static void showOfficeCodesAsDepartment() {  SessionFactory factory = new Configuration().configure().buildSessionFactory();  Session session = factory.openSession();  *//------------ Hibernate Named Query -------------*  TypedQuery<Object[]> query = session.createNamedQuery("employeeDeptAlias", Object[].class);  List<Object[]> employees = query.getResultList();  System.*out*.printf("%-15s%-17s%s%n", "OfficeCode","Dep Name","Employee Name");  for (Object[] e : employees) {  System.*out*.printf(" %-15s%-17s%s%n",e[1],e[3],e[2]);  }  factory.close();  session.close();  }  } | | --- |     You can call the above methods from the main() method.  Results:  *findEmployeeByName*();    *findEmployeeById*();    *showOfficeCodesAsDepartment*(); |
| --- | --- |

## Conclusion

Named queries are global access, which means that the name of a query has to be unique in XML mapping files or annotations. In a real environment, it is always good practice to isolate all of the named queries into their own files. In addition, named queries stored in the Hibernate mapping files or annotations are easier to maintain than queries scattered throughout the Java code.

**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.

## 