# Guided LAB 305.4.3A - Demonstration - Named Queries in Hibernate

**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 the **second part**, 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

## Step 1: Set up the Java Maven Project and add Jar Dependencies.

* Create a new Maven Project - **Left-click on File Menu → New → Maven Project.**
* For **Hibernate** and **MySQL databases:** Add the following **jar dependencies** in the **pom.xml** file under **</dependencies> tag** of your Maven project.

| | <dependencies>  <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core --> <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-core</artifactId>  <version>5.5.7.Final</version> </dependency>  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-annotations</artifactId>  <version>3.5.5-Final</version> </dependency>  *<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->*  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.25</version>  </dependency>  </dependencies> | | --- | |
| --- | --- |

## Step 2: Create the Persistence Class (Model class or Pojo).

* Create a package called **“com.perscholas.model.”**
* Create an entity class named **“Employee.java”** under the above package.
* Then we will use annotations to map this table to the corresponding table in the database.
* Here is the initial code for the **Employee** class.

| import java.util.Date; import javax.persistence.Entity; import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Id; import javax.persistence.Table; @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 + "]";  } } |
| --- |

## 

## 

## **Step 3: Create the Hibernate Configuration file (hibernate.cfg.xml)**

* **For Eclipse IDE**: To create the configuration file, right-click on **src/main/java** → **New → Other - search files from search panel → Click on File → Specify the file name** **“hibernate.cfg.xml”** → **Finish**.
* **For the IntelliJ IDE:** Create a configuration file named **hibernate.cfg.xml** under the **resources** folder and write the following code in it.

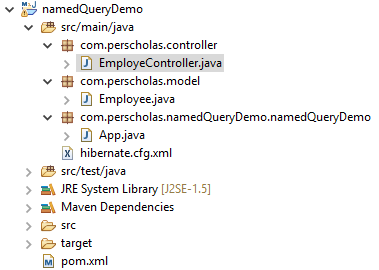
***Note****: in this lab, we will use the* ***“userdb”*** *database, but if you want to use another database for that, you have to change the database name in the code below.*

* Open newly created file and paste the following XML code:

| <?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 schema 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">root</property>  <property name="connection.password">password</property>  <!--MySQL DB dialect -->  <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>  <!-- Echo all executed SQL to stdout -->  <property name="hibernate.show\_sql" >true </property>  <property name="hibernate.format\_sql" >true </property>   <!-- Validate the database schema on startup  <property name="hbm2ddl.auto">validate</property>-->   <!-- Mapping entity file -->  <mapping class="com.perscholas.model.Employee"/> </session-factory> </hibernate-configuration> |
| --- |

## **Step 4: Create a table in the database using Hibernate.**

Create a package called **com.perscholas.controller.** Create the **“EmployeController'' class** in this package as shown in the below screenshot.



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

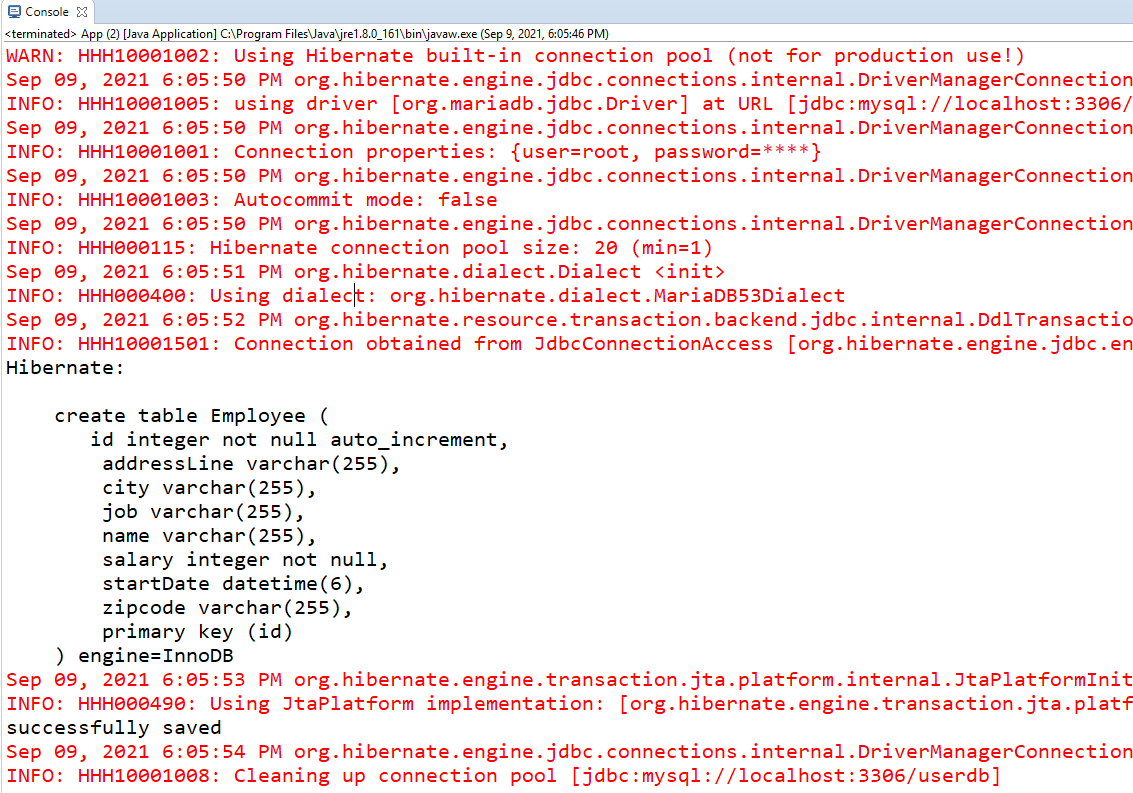
| import org.hibernate.Session; import org.hibernate.SessionFactory; import org.hibernate.Transaction; import org.hibernate.cfg.Configuration; public class EmployeController {  public void createEmployeeTable()  {  SessionFactory factory = new Configuration().configure().buildSessionFactory();  Session session = factory.openSession();  Transaction t = session.beginTransaction();   Employee uone = new Employee();  t.commit();  System.out.println("successfully saved");   factory.close();   session.close();   }  } |
| --- |

## **Step 5: Run an Application.**

Open the **App.java** class, which contains a main method() that you can find under ***src/main/java.***

Create an instance of the EmployeController class and call the createEmployeeTable() method as shown below.

| import com.perscholas.controller.EmployeController; public class App  { public static void main( String[] args )  {  System.out.println( "Hello World!" );  EmployeController e = new EmployeController();  e.createEmployeeTable();  } } |
| --- |

Run your code. You might see similar output on your console.

Let's enter a few employees' records for the demonstration. To do so, execute the following SQL script in the MySQL workbench.

| 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'); |
| --- |

# 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.

| import java.util.Date; import javax.persistence.Entity; import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Id; import javax.persistence.Table; import org.hibernate.annotations.NamedQueries; import org.hibernate.annotations.NamedQuery; @Entity @Table *//Using @NamedQuery for single HQL* @NamedQuery(name="getallEmployee", query="select e from Employee e") *//Using @NamedQueries for multiple HQL* @NamedQueries({  @NamedQuery(name="get\_Emp\_name\_by\_id", query="select e.name,e.salary,e.job from Employee e where id=:id"),  @NamedQuery(name="get\_all\_dept", query="from Employee"),  @NamedQuery( name = "findEmployeeByName", query = "from Employee e where e.name = :name" ),  @NamedQuery(name ="empDepAlias", 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 'Finanace'"  + "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 + "]";  } } |
| --- |

Now, go to the “EmployeController” class and add the below code to this class. In the code, we created three methods named *findEmployeeById(), findEmployeeByname()* and *ShowOfficeCodes\_AsDepartment()* and each method uses ***NamedQuery.***

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

| EmployeController class   | import java.util.Iterator; import java.util.List; import javax.persistence.TypedQuery; import org.hibernate.Session; import org.hibernate.SessionFactory; import org.hibernate.Transaction; import org.hibernate.cfg.Configuration; import com.perscholas.model.Employee; public class EmployeController {    public void findEmployeeByname() {  SessionFactory factory = new Configuration().configure().buildSessionFactory();  Session session = factory.openSession(); //------------ Hibernate Named Query -------------  **TypedQuery query = session.getNamedQuery("findEmployeeByName");**   query.setParameter("name","Tom Thele");   List<Employee> employees=query.getResultList();   Iterator<Employee> itr=employees.iterator();   while(itr.hasNext()){   Employee e=itr.next();   System.out.println(e);   }   factory.close();   session.close();   } public void findEmployeeById()  { SessionFactory factory = new Configuration().configure().buildSessionFactory();  Session session = factory.openSession(); //------------ Hibernate Named Query -------------    **TypedQuery query = session.getNamedQuery("get\_Emp\_name\_by\_id");   query.setParameter("id",3);**   List<Object[]> emName= query.getResultList();  for(Object[] o: emName)  {  System.out.println("Employee name: " +o[0] +" | Employee Salary: "+ o[1] +" | Emp Job Title: "+ o[2]);  }  factory.close();   session.close();  }  public void ShowOfficeCodes\_AsDepartment()  {  SessionFactory factory = new Configuration().configure().buildSessionFactory();  Session session = factory.openSession(); //------------ Hibernate Named Query -------------  **TypedQuery query = session.getNamedQuery("empDepAlias");**  List<Object[]> list = query.getResultList();  for(Object[] e: list )  {  System.out.println("OfficeCode: " + e[1] + " | Dep Name: " +e[3]+ " | Employee Name: " + e[2]);  }  factory.close();   session.close();   }  } | | --- |     You can call the above methods from the main() method, as shown below. The **main()** method is defined in the ***App.java*** class.  You can execute each method() one by one as shown below.   | public class App  {  public static void main( String[] args )  {  System.out.println( "Hello World!" );  EmployeController e = new EmployeController();  //e.createEmployeeTable();  //e.findEmployeeByname();  //e.findEmployeeById();  e.ShowOfficeCodes\_AsDepartment();  } } | | --- | |
| --- | --- | --- |

## 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.

## 