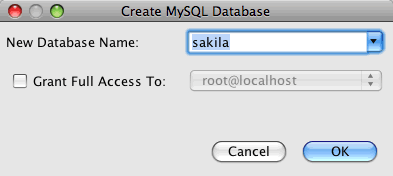
Creating the Database

This tutorial uses a MySQL database called sakila, a free sample MySQL database that is available from the MySQL site. The sakila database is not included when you install the IDE so you need to first create the database to follow this tutorial.

To create the sakila database you can download and install the Sakila Sample Database plugin using the Plugins manager. After you install the plugin the sakila database is added to the list of databases in the Create MySQL database dialog box.

For more information on configuring the IDE to work with MySQL, see the [Connecting to a MySQL Database](https://netbeans.org/kb/docs/ide/mysql.html) tutorial.

1. Open the Plugins manager and install the Sakila Sample Database plugin.
2. After installing the plugin, start the MySQL database by expanding the Databases node in the Services window, right-clicking the MySQL Server node and choosing Start.
3. Right-click the MySQL Server node and choose Create Database.
4. Select the Sakila database from the New Database Name drop down list in the Create MySQL Database dialog box. Click OK.  
   

When you click OK a Sakila node appears under the MySQL Server node.

1. Right-click the Sakila node and choose Connect.

When you click Connect a database connection node for the Sakila database (jdbc:mysql://localhost:3306/sakila [*username* on Default]) is listed under the Databases node. When a connection is open you can view the data in the database by expanding the connection node.

Creating the Web Application Project

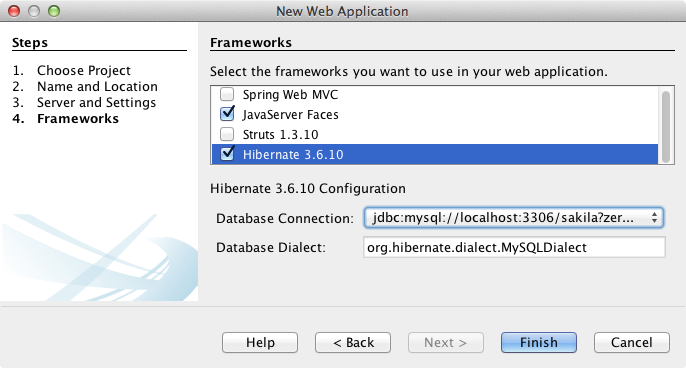
In this exercise you will create a web application project and add the Hibernate libraries to the project. When you create the project, you will select Hibernate in the Frameworks panel of the New Project wizard and specify the database.

1. Choose File > New Project (Ctrl-Shift-N; ⌘-Shift-N on Mac) from the main menu. Select Web Application from the Java Web category and click Next.
2. Type **DVDStore** for the project name and set the project location.
3. Deselect the Use Dedicated Folder option, if selected. Click Next.

For this tutorial there is little reason to copy project libraries to a dedicated folder because you will not need to share libraries with other users.

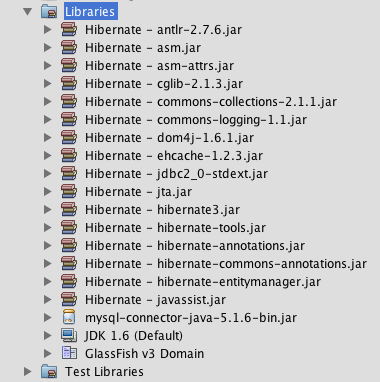
1. Set the server to the GlassFish Server and set the Java EE Version to Java EE 6 Web or Java EE 7 Web. Click Next.
2. Select the JavaServer Faces checkbox and use the default JSF 2.x libraries.
3. Select the Hibernate checkbox in the list of frameworks.
4. Select the sakila database from the Database Connection drop down list. Click Finish.

**Note:** If the sakila database is not available as an option in the Frameworks panel in the wizard, check to see if the connection is listed under the Databases node in the Services window. If the connection is not there, you need to create the database connection.



When you click Finish, the IDE creates the web application project and opens the hibernate.cfg.xml file and index.xhtml in the editor.

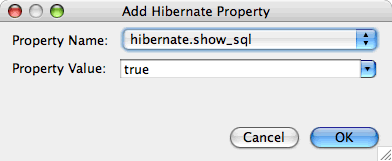
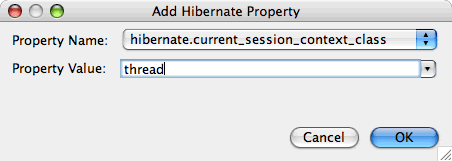
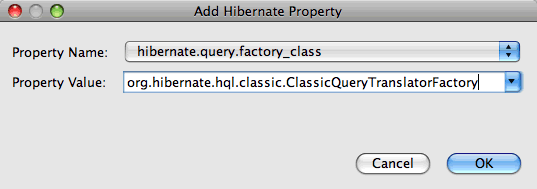
If you expand the Libraries node in the Projects window, you can see that the IDE added the Hibernate libraries to the project.



Modifying the Hibernate Configuration File

When you create a new project that uses the Hibernate framework, the IDE automatically creates the hibernate.cfg.xml configuration file at the root of the context classpath of the application (in the Files window, src/java). The file is located in the <default package> under the Source Packages node in the Projects window. The configuration file contains information about the database connection, resource mappings, and other connection properties. You can edit the file using the multi-view editor, or edit the XML directly in the XML editor.

In this exercise you will edit the default properties specified in hibernate.cfg.xml to enable debug logging for SQL statements and to enable Hibernate's session context management.

1. Open hibernate.cfg.xml in the Design tab. You can open the file by expanding the <default package> node under Source Packages in the Projects window and double-clicking hibernate.cfg.xml.
2. In the multi-view XML editor, expand the Configuration Properties node under Optional Properties.
3. Click Add to open the Add Hibernate Property dialog box.
4. In the dialog box, select the hibernate.show\_sql property and set the value to true. This enables the debug logging of the SQL statements.  
   
5. Expand the Miscellaneous Properties node and click Add.
6. In the dialog box, select the properties hibernate.current\_session\_context\_class and set the value to thread to enable Hibernate's automatic session context management.  
   
7. Click Add again under the Miscellaneous Properties node and select hibernate.query.factory\_class in the Property Name dropdown list.
8. Select **org.hibernate.hql.classic.ClassicQueryTranslatorFactory** as the Property Value. Click OK.  
   

If you click the XML tab in the editor you can see the file in XML view. Your file should look similar to the following (the three new properties are bold):

<hibernate-configuration>

<session-factory name="session1">

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/sakila</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">######</property>

**<property name="hibernate.show\_sql">true</property>**

**<property name="hibernate.current\_session\_context\_class">thread</property>**

**<property name="hibernate.query.factory\_class">org.hibernate.hql.classic.ClassicQueryTranslatorFactory</property>**

</session-factory>

</hibernate-configuration>

1. Save your changes to the file.

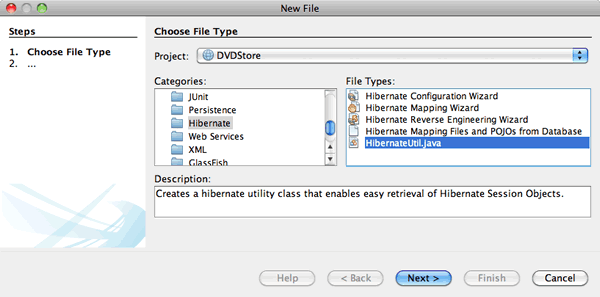
You can close the file because you do not need to edit the file again.

Creating the HibernateUtil.java Helper File

To use Hibernate you need to create a helper class that handles startup and that accesses Hibernate's SessionFactory to obtain a Session object. The class calls configure() and loads the hibernate.cfg.xml configuration file and then builds the SessionFactory to obtain the Session object.

In this section you use the New File wizard to create the helper class HibernateUtil.java.

1. Right-click the Source Packages node and select New > Other to open the New File wizard.
2. Select Hibernate from the Categories list and HibernateUtil.java from the File Types list. Click Next.
3. Type **HibernateUtil** for the class name and **dvdrental** for the package. Click Finish.



When you click Finish, HibernateUtil.java opens in the editor. You can close the file because you do not need to edit the file.

Generating Hibernate Mapping Files and Java Classes

In this tutorial you use a POJO (plain old Java object) to represent the data in each of the tables in the database that you will use. The Java class specifies the fields for the columns in the tables and uses simple setters and getters to retrieve and write the data. To map the POJOs to the tables you can use a Hibernate mapping file or use annotations in the class.

You can use the Hibernate Mapping Files and POJOs from a Database wizard to create multiple POJOs and mapping files based on database tables. When you use the wizard you select all the tables for which you want POJOs and mapping files and the IDE then generates the files for you based on the database tables and adds the mapping entries to hibernate.cfg.xml. When you use the wizard you can choose the files that you want the IDE to generate (only the POJOs, for example) and select code generation options (generate code that uses EJB 3 annotations, for example).

**Note.** The IDE also has wizards to help you create individual POJOs and mapping files from scratch.

Creating the Hibernate Reverse Engineering File

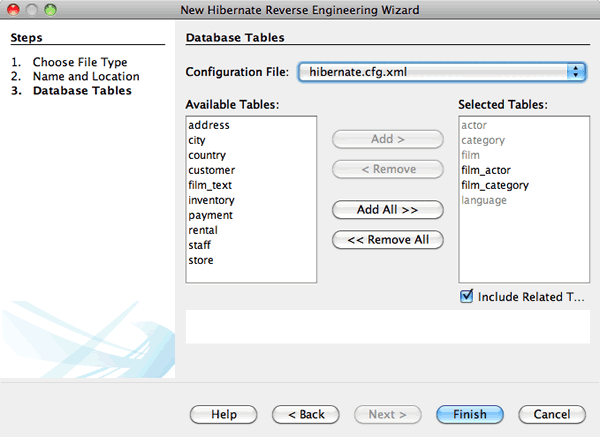
If you want to use the Hibernate Mapping Files and POJOs from a Database wizard, you first need to create a hibernate.reveng.xml reverse engineering file. The Hibernate Mapping Files and POJOs from a Database wizard requires hibernate.reveng.xml and hibernate.cfg.xml.

The reverse engineering file enables you to have greater control over the database mapping strategy. The Hibernate Reverse Engineering Wizard creates a reverse engineering file with a default configuration that you can edit in the XML editor.

To create the Hibernate reverse engineering file, perform the following steps.

1. Right-click the Source Packages node in the Projects window and choose New > Other to open the New File wizard.
2. Select Hibernate Reverse Engineering Wizard in the Hibernate category. Click Next.
3. Specify hibernate.reveng as the File Name and src/java for the Folder. Click Next.
4. Select hibernate.cfg.xml from the Configuration File drop down list, if not selected.
5. Select the following tables from Available Tables and click Add to add the tables to Selected Tables.
   * actor
   * category
   * film
   * film\_actor
   * film\_category
   * language

Click Finish.



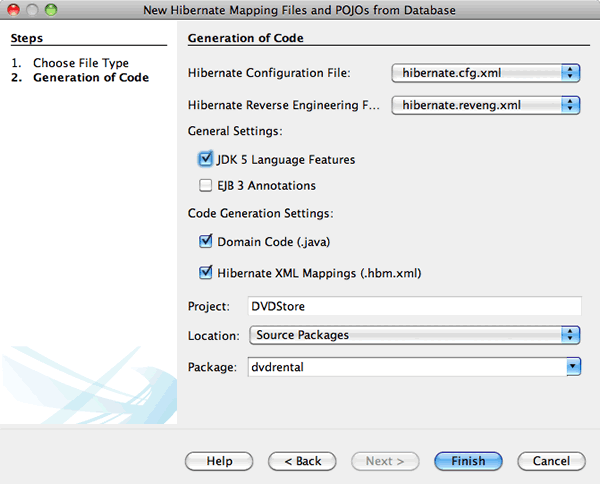
The wizard generates a hibernate.reveng.xml reverse engineering file and opens the file in the editor. You can close the reverse engineering file because you will not need to edit the file.

For more details about working with the hibernate.reveng.xml file, see [Chapter 5. Controlling reverse engineering](http://docs.jboss.org/tools/2.1.0.Beta1/hibernatetools/html/reverseengineering.html) in the [Hibernate Tools Reference Guide](http://docs.jboss.org/tools/2.1.0.Beta1/hibernatetools/html/).

Creating the Hibernate Mapping Files and POJOs

You can use the Hibernate Mapping Files and POJOs from a Database wizard to generate files for you. The wizard can generate a POJO and a corresponding mapping file for each table that you select in the wizard. The mapping files are XML files that contain data about how the columns in the tables are mapped to the fields in the POJOs. You need to have the hibernate.reveng.xml and hibernate.cfg.xml files to use the wizard.

To create the POJOS and mapping files using a wizard, perform the following steps.

1. Right-click the Source Packages node in the Projects window and choose New > Other to open the New File wizard.
2. Select Hibernate Mapping Files and POJOs from a Database in the Hibernate category. Click Next.
3. Ensure that the hibernate.cfg.xml and hibernate.reveng.xml files are selected in the drop down lists.
4. Select **JDK 5 Language Features** under the General Settings options.
5. Ensure that the **Domain Code** and **Hibernate XML Mappings** options are selected.
6. Select **dvdrental** for the Package name. Click Finish.  
   

When you click Finish the IDE generates POJOs and Hibernate mapping files with the fields mapped to the columns specified in hibernate.reveng.xml. The IDE also adds mapping entries to hibernate.cfg.xml.

<hibernate-configuration>

<session-factory>

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/sakila</property>

<property name="hibernate.connection.username">myusername</property>

<property name="hibernate.connection.password">mypassword</property>

<property name="hibernate.show\_sql">true</property>

<property name="hibernate.current\_session\_context\_class">thread</property>

<property name="hibernate.query.factory\_class">org.hibernate.hql.classic.ClassicQueryTranslatorFactory</property>

<mapping resource="dvdrental/FilmActor.hbm.xml"/>

<mapping resource="dvdrental/Language.hbm.xml"/>

<mapping resource="dvdrental/Film.hbm.xml"/>

<mapping resource="dvdrental/Category.hbm.xml"/>

<mapping resource="dvdrental/Actor.hbm.xml"/>

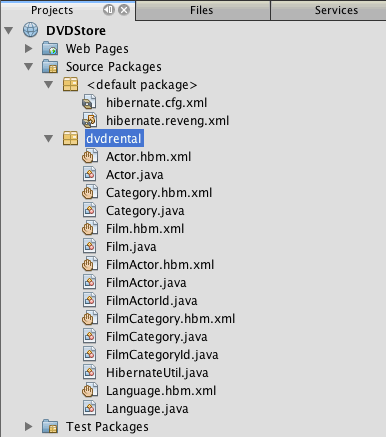
<mapping resource="dvdrental/FilmCategory.hbm.xml"/>

</session-factory>

</hibernate-configuration>

**Note.** Confirm that the mapping elements are listed after the property elements in the hibernate.cfg.xml file.

You can expand the dvdrental package to see the files generated by the wizard.



You can use the Hibernate Mapping wizard if you want to create a Hibernate mapping file that maps a specific table to a specific class.

For more details about working with the hibernate.reveng.xml file, see [Chapter 5. Basic O/R Mapping](http://docs.jboss.org/hibernate/core/3.3/reference/en/html/mapping.html) in the [Hibernate Reference Documentation](http://docs.jboss.org/hibernate/stable/core/reference/en/html/).

Creating the FilmHelper.java Helper Class

You will now create a helper class in the dvdrental package that will be used to perform Hibernate queries on the database. You will use the Hibernate Query Language (HQL) editor to construct and test the queries for retrieving data. After you test the queries you will create methods in the helper class that construct and run the queries. You will then invoke the methods in the helper class from a JSF managed bean.

Creating the Class

In this section you use the New File wizard to create the helper class FilmHelper.java in the dvdrental package. You will create a Hibernate session by calling getSessionFactory in HibernateUtil.java and create some helper methods to create queries to retrieve data from the database. You will invoke the helper methods from the JSP pages.

1. Right-click the dvdrental source package node and select New > Java Class to open the New File wizard.
2. Type **FilmHelper** for the class name.
3. Confirm that **dvdrental** is selected as the Package. Click Finish.
4. Adding the following code (in bold) to create a Hibernate session.
5. public class FilmHelper {
6. **Session session = null;**
7. **public FilmHelper() {**
8. **this.session = HibernateUtil.getSessionFactory().getCurrentSession();**
9. **}**

}

1. Right-click in the editor and choose Fix Imports (Alt-Shift-I; ⌘-Shift-I on Mac) to add any required import statements (org.hibernate.Session) and save your changes.

You will now modify FilmHelper.java to add methods that query the DB.

Enumerating Film Titles and Retrieving Actors Using an HQL Query

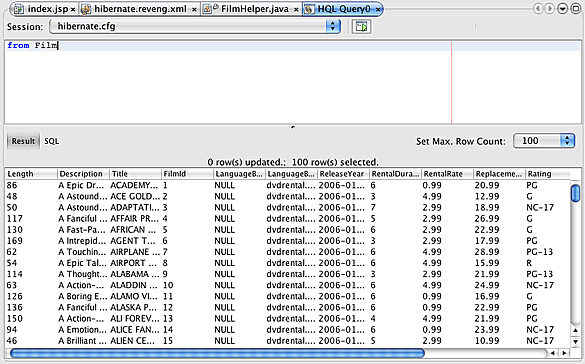
In this exercise you will create a Hibernate Query Language (HQL) query that queries the database to retrieve a list of film titles from the Film table. You will then add a method that queries both the Actor and Film\_actor tables to fetch the actors involved in a particular film.

The Film table has 1000 records so the method to retrieve the list of films should be able to retrieve records based on the filmId primary key. You will use the HQL editor to construct and test the HQL query. After you have created the correct query you will add a method to the class that can generate the proper query.

1. Right-click the project node in the Projects window and choose Clean and Build.
2. Right-click hibernate.cfg.xml in the Projects window and choose Run HQL Query to open the HQL query editor.
3. Select hibernate.cfg from the drop down list in the toolbar.
4. Test the connection by typing the following in the editor and clicking the Run HQL Query button ( Run HQL Query button ) in the toolbar.

from Film

When you click Run HQL Query you can see the results of the query in the bottom window of the HQL query editor.



If you click the SQL button you can see the equivalent SQL query.

select film0\_.film\_id as col\_0\_0\_ from sakila.film film0\_

1. Type the following query to retrieve the records in the Film table where the film id is between 100 and 200.

from Film as film where film.filmId between 100 and 200

The result window displays a list of records. Now that you have tested that the query returns the desired results, you can use the query in the helper class.

1. Add the following method getFilmTitles to FilmHelper.java to retrieve the films where the film id is between a certain range specified by the variables startID and endID.
2. public List getFilmTitles(int startID, int endID) {
3. List<Film> filmList = null;
4. try {
5. org.hibernate.Transaction tx = session.beginTransaction();
6. Query q = session.createQuery ("from Film as film where film.filmId between '"+startID+"' and '"+endID+"'");
7. filmList = (List<Film>) q.list();
8. } catch (Exception e) {
9. e.printStackTrace();
10. }
11. return filmList;

}

1. Add the following method getActorsByID that retrieves the actors in a particular film. The method constructs the query using filmId as the input variable.
2. public List getActorsByID(int filmId){
3. List<Actor> actorList = null;
4. try {
5. org.hibernate.Transaction tx = session.beginTransaction();
6. Query q = session.createQuery ("from Actor as actor where actor.actorId in (select filmActor.actor.actorId from FilmActor as filmActor where filmActor.film.filmId='" + filmId + "')");
7. actorList = (List<Actor>) q.list();
8. } catch (Exception e) {
9. e.printStackTrace();
10. }
11. return actorList;

}

1. Fix your imports and save your changes.

When you fix your imports you want to choose java.util.List and org.hibernate.Query.

Adding Additional Helper Methods

You will now add additional helper methods that create queries based on an input variable. You can check the queries in the HQL query editor.

1. Add the following method to retrieve a list of categories according to filmId.
2. public Category getCategoryByID(int filmId){
3. List<Category> categoryList = null;
4. try {
5. org.hibernate.Transaction tx = session.beginTransaction();
6. Query q = session.createQuery("from Category as category where category.categoryId in (select filmCat.category.categoryId from FilmCategory as filmCat where filmCat.film.filmId='" + filmId + "')");
7. categoryList = (List<Category>) q.list();
8. } catch (Exception e) {
9. e.printStackTrace();
10. }
11. return categoryList.get(0);

}

1. Add the following method to retrieve a single film according to filmId.
2. public Film getFilmByID(int filmId){
3. Film film = null;
4. try {
5. org.hibernate.Transaction tx = session.beginTransaction();
6. Query q = session.createQuery("from Film as film where film.filmId=" + filmId);
7. film = (Film) q.uniqueResult();
8. } catch (Exception e) {
9. e.printStackTrace();
10. }
11. return film;

}

1. Add the following method to retrieve the film language according to langId.
2. public String getLangByID(int langId){
3. Language language = null;
4. try {
5. org.hibernate.Transaction tx = session.beginTransaction();
6. Query q = session.createQuery("from Language as lang where lang.languageId=" + langId);
7. language = (Language) q.uniqueResult();
8. } catch (Exception e) {
9. e.printStackTrace();
10. }
11. return language.getName();

}

1. Save your changes.