**HIBERNATE FRAMWORK**

|  |
| --- |
| Phan Hoài Thu |
| Dương Quang Trà |
| Nguyễn Hải Cơ |
| Thái Thị Thu Đông |
| Cao Khắc Huy |
| Nguyễn Nhật Khánh |
| Võ Thị Mỹ Viên |
| Hồ Duy Tây |
| Huỳnh Tấn Pháp |
| Lê Phúc Lợi |

**MỤC LỤC**

[**1.** **ORM** 3](#_Toc21556625)

[**2.** **GIỚI THIỆU HIBERNATE** 4](#_Toc21556626)

[**3.** **KIẾN TRÚC HIBERNATE** 5](#_Toc21556635)

[**4.** **Môi Trường** 3](#_Toc21556643)

[**5.** **FILE CẤU HÌNH XML** 6](#_Toc21556644)

[**6.** **FILE MAPPING TRONG HIBERNATE** 10](#_Toc21556645)

[**7.** **Thuộc tính type** 15](#_Toc21556647)

[**8.** **Cách ánh xạ từ bảng trong csdl sang entity class** 17](#_Toc21556648)

[**9.** **Lớp persistent** 26](#_Toc21556649)

[**10.** **Sessions** 26](#_Toc21556650)

[**12.** **Annotations** 30](#_Toc21556654)

[**13.** **QUERY LANGUAGE** 34](#_Toc21556659)

[**13. Criteria Query** 38](#_Toc21556672)

[**14.** **CÁC KIỂU TRUY VẤN NATIVE SQL** 42](#_Toc21556679)

[**15. BỘ NHỚ CACHE** 43](#_Toc21556682)

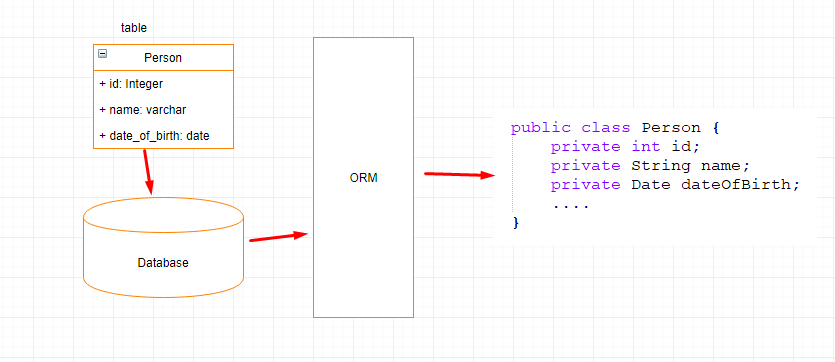
[**16. Batch Processing** 46](#_Toc21556686)

[**17. INTERCEPTORS** 48](#_Toc21556688)

# **ORM**

Một ứng dụng có thể chia làm 3 phần như sau: phần giao diện người dùng (**UI layer**), phần xử lý nghiệp vụ (**business layer**) và phần chứa dữ liệu (**data layer**). Cụ thể ra, business layer sẽ có thể chia nhỏ thành 2 layer con là **business logic layer** (chỉ quan tâm đến ý nghĩa của các nghiệp vụ, các tính toán mang nhằm thoả mãn yêu cầu của người dùng) và **persitence layer**. Persistence layer chịu trách nhiệm **giao tiếp** với data layer (thường là một hệ quản trị cơ sở dữ liệu quan hệ - Relational DBMS). Persistence layer sẽ đảm nhiệm các nhiệm vụ **mở kết nối, truy xuất và lưu trữ** dữ liệu vào các Relational DBMS.

ORM (**O**bject **R**elational **M**apping), là một kỹ thuật/cơ chế lập trình thực hiện **ánh xạ** CSDL sang các đối tượng trong các ngôn ngữ lập trình hướng đối tượng. Việc sử dụng ORM cho phép lập trình viên thao tác với database 1 cách hoàn toàn tự nhiên, dễ hiểu thông qua các đối tượng. Lập trình viên không cần tới loại database, kiểu dữ liệu trong database…



**Java ORM Framework:** Một persistent framework là một dịch vụ ORM lưu và truy xuất các đối tượng vào một cơ sở dữ liệu quan hệ.

* Enterprise JavaBeans Entity Beans
* Java Data Objects
* TopLink
* Spring DAO
* Hibernate

**Lợi ích của ORM:**

* Tự động hóa chuyển đổi từ object sang table và ngược lại từ table sang object rất nhanh chóng giúp giảm thời gian cũng như chi phí phát triển.
* ORMcũng cần ít code hơn store procedures, có khả năng thay thế số lượng lớn store procedudres cần phát triển.
* ORM cũng giúp tăng tốc thực thi của hệ thống.

# **GIỚI THIỆU HIBERNATE**

Hibernate framework là một giải pháp ORM (Object Relational Mapping) mã nguồn mở, gọn nhẹ. Hibernate giúp đơn giản hoá sự phát triển của ứng dụng java để tương tác với cơ sở dữ liệu.Hibernate là 1 Framework cho persistence layer: Thực hiện giao tiếp giữa tầng ứng dụng với tầng dữ liệu (kết nối, truy xuất, lưu trữ…)

## Lợi ích của Hibernate Framework

a. Mã nguồn mở và nhẹ**:**Hibernate Framework là mã nguồn mở có giấy phép LGPL.

b. Hiệu suất nhanh**:** Hiệu suất của Hibernate Framework là nhanh bởi vì bộ nhớ cache được sử dụng trong nội bộ Hibernate Framework.Có hai loại bộ nhớ cache trong Hibernate Framework, gồm bộ nhớ **cache cấp một** và bộ nhớ **cache cấp hai**. Bộ nhớ cache cấp một được bật bằng lệnh mặc định.

c. Truy vấn CSDL độc lập**:** HQL (Hibernate Query Language) là phiên bản hướng đối tượng của SQL. Nó tạo ra các truy vấn cơ sở dữ liệu độc lập. Vì vậy, bạn không cần phải viết các truy vấn cơ sở dữ liệu cụ thể. Trước Hibernate, nếu dự án có cơ sở dữ liệu bị thay đổi, chúng ta cần phải thay đổi truy vấn SQL dẫn đến sự cố bảo trì.

d. Tạo bảng tự động**:** Hibernate framework cung cấp phương tiện để tạo ra các bảng cơ sở dữ liệu tự động.

e. Đơn giản lệnh join phức tạp**:** Có thể lấy dữ liệu từ nhiều bảng một cách dễ dàng với Hibernate framework.

f. Cung cấp thống kê truy vấn và trạng thái cơ sở dữ liệu**:** Hibernate hỗ trợ bộ nhớ cache truy vấn và cung cấp số liệu thống kê về truy vấn và trạng thái cơ sở dữ liệu.

**Database được hỗ trợ:**

* HSQL Database Engine
* DB2/NT
* MySQL
* PostgreSQL
* FrontBase
* Oracle
* Microsoft SQL Server Database
* Sybase SQL Server
* Informix Dynamic Server

# **KIẾN TRÚC HIBERNATE**

## Các thành phần trong kiến trúc Hibernate



* + 1. Đối tượng Configuration**:** Các đối tượng cấu hình chỉ tạo ra trong quá trình khời chạy ứng dụng **lần đầu tiên**, nó đại diện cho một tập tin cấu hình, thuộc tính theo yêu cầu của Hibernate. Các đối tượng cấu hình cung cấp hai thành phần:
* **Kết Nối CSDL:** Được xử lý thông qua tập tin cấu hình được hỗ trợ bởi Hibernate. Tập tin này là **hibernate.properties** hay **hibernate.cfg.xml.** File Hibernate.cfg.xml là tập tin được load đầu tiên khi khởi chạy ứng dụng Hibernate. Nó chứa các thông tin về kết nối (database kết nối là gì, username và password kết nối với database là gì, tên database được kết nối), driver kết nối…
* **Cài Đặt Lớp Mapping:** Thành phần này tạo ra sự kết nối giữa các lớp Java và các bảng cơ sở dữ liệu. Mỗi file mapping (ánh xạ) có dạng \*\*\*.hbm.xml có nhiệm vụ đặc tả mỗi quan hệ giữa các thuộc tính của đối tượng và các trường trong bảng CSDL (cột nào ứng với thuộc tính nào của class, kiểu giá trị là gì, quan hệ giữa các bảng chuyển thành quan hệ giữa các lớp ra sao…).
  + 1. Đối tượng SessionFactory**:** SessionFactory là đối tượng lớn trong ứng dụng nên thường nó được tạo ra trong **suốt quá trình khởi động** và giữ để sử dụng sau. Mỗi cơ sở dữ liệu sử dụng một tập tin cấu hình riêng biệt và ứng với một cơ sở dữ liệu sẽ là một đối tượng SessionFactory, nếu bạn đang sử dụng nhiều cơ sở dữ liệu bạn sẽ phải tạo ra nhiều đối tượng SessionFactory.
    2. Đối Tượng Session:Session được sử dụng để **kết nối** với cơ sở dữ liệu. Đối tượng Session được khởi tạo khi **tương tác** với cơ sở dữ liệu. Đối tượng liên tục được lưu lại và lấy thông qua một đối tượng Session.

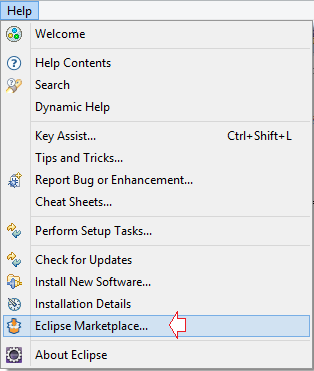
Các đối tượng Session không nên giữ mở trong một thời gian dài bởi vì không an toàn.

* + 1. Đối Tượng Transaction**:** Đây là một đối tượng tùy chọn và ứng dụng Hibernate có thể chọn không sử dụng, thay vì quản lý giao dịch trong mã ứng dụng của riêng mình.
    2. Đối tượng Query: Đối tượng truy vấn sử dụng SQL hoặc Hibernate Query Language (HQL) để lấy dữ liệu từ cơ sở dữ liệu.
    3. Đối tượng Criteria**:** Đối tượng Criteria được sử dụng để tạo và thực hiện truy vấn các tiêu chí định hướng đối tượng để lấy các đối tượng.

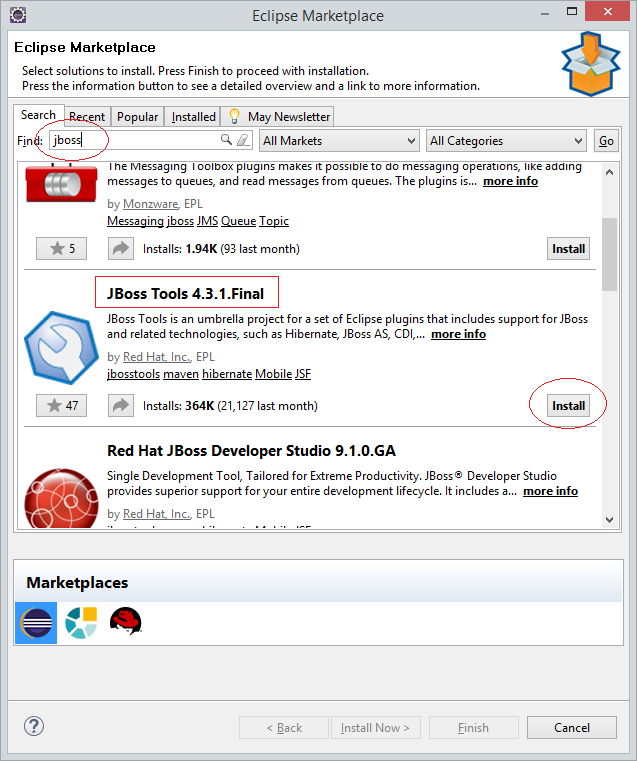
# **Môi Trường**

Cài đặt <https://o7planning.org/vi/10119/cai-dat-jboss-tools-vao-eclipse>

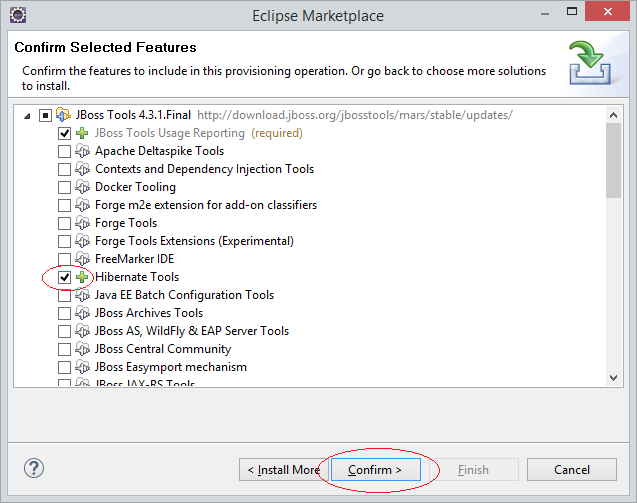
Để cài đặt ***"JBoss Tools"*** từ ***"Eclipse Markplace.."*** trên Eclipse chọn:

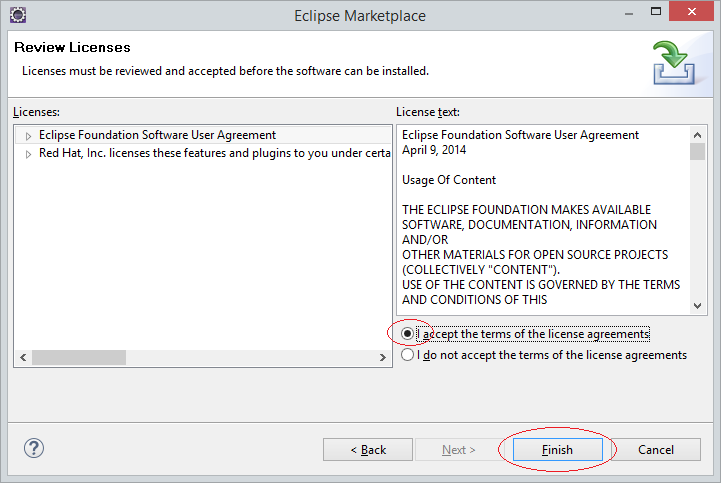


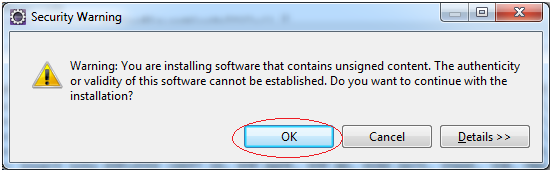
Trên **"Eclipse Marketplace.."** tìm kiếm theo từ khóa **JBoss**, Và cài đặt **"JBoss Tools"** tương ứng với phiên bản Eclipse của bạn, ở đây tôi đang sử dụng Eclipse phiên bản 4.5 (MARS).



Trong **"JBoss Tools"** có rất nhiều công cụ, chọn các công cụ mà bạn cần đến.







Cách add thư viện Hibernate vào project:

Tạo **maven** project: Thêm các thư viện sau vào file pom.xml

<!-- Hibernate -->

<!-- http://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.1.0.Final</version>

</dependency>

<!-- http://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-entitymanager</artifactId>

<version>5.1.0.Final</version>

</dependency>

<!-- http://mvnrepository.com/artifact/org.hibernate/hibernate-c3p0 -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-c3p0</artifactId>

<version>5.1.0.Final</version>

</dependency>

<!-- MySQL JDBC driver -->

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

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.34</version>

</dependency>

# **FILE CẤU HÌNH XML**

Hibernate yêu cầu phải thiết định thông tin các lớp để lớp Java ánh xạ đến các bảng cơ sở dữ liệu. Tất cả các thông tin như vậy thường được cung cấp dưới dạng file thuộc tính Java tiêu chuẩn có tên gọi là hibernate.properties, hoặc dưới dạng file XML có tên **hibernate.cfg.xml.** Hầu hết các thuộc tính lấy giá trị mặc định của chúng và không bắt buộc phải chỉ định chúng trong file properties trừ khi nó thực sự là bắt buộc. Tập tin này được lưu trong thư mục gốc của ứng dụng.

Dưới đây là danh sách các thuộc tính quan trọng mà bạn sẽ cần để cấu hình cho một cơ sở dữ liệu:

* **hibernate.dialect**: Thuộc tính này làm cho Hibernate tạo ra SQL thích hợp cho cơ sở dữ liệu đã chọn.
* **hibernate.connection.driver\_class:** JDBC driver class.
* **hibernate.connection.url:** JDBC URL của cơ sở dữ liệu.
* **hibernate.connection.username:** username của cơ sở dữ liệu.
* **hibernate.connection.password:** password của cơ sở dữ liệu.
* **hibernate.connection.autocommit:** Cho phép chế độ autocommit để được sử dụng cho JDBC connection.
* **hibernate.connection.pool\_size:** Giới hạn số kết nối đang chờ trong pool kết nối cơ sở dữ liệu Hibernate.

Ví dụ về file hibernate.cfg.xml:

**<?xml version="1.0" encoding="utf-8"?>**

**<!DOCTYPE hibernate-configuration SYSTEM**

**"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">**

**<hibernate-configuration>**

**<session-factory>**

**<property name="hibernate.dialect">**

**org.hibernate.dialect.MySQLDialect**

**</property>**

**<property name="hibernate.connection.driver\_class">**

**com.mysql.jdbc.Driver**

**</property>**

**<!-- thông tin kết nối đến database -->**

**<property name="hibernate.connection.url">**

**jdbc:mysql://localhost/testdb**

**</property>**

**<property name="hibernate.connection.username">**

**root**

**</property>**

**<property name="hibernate.connection.password">**

**1234567890**

**</property>**

**<!-- Danh sách các XML mapping file-->**

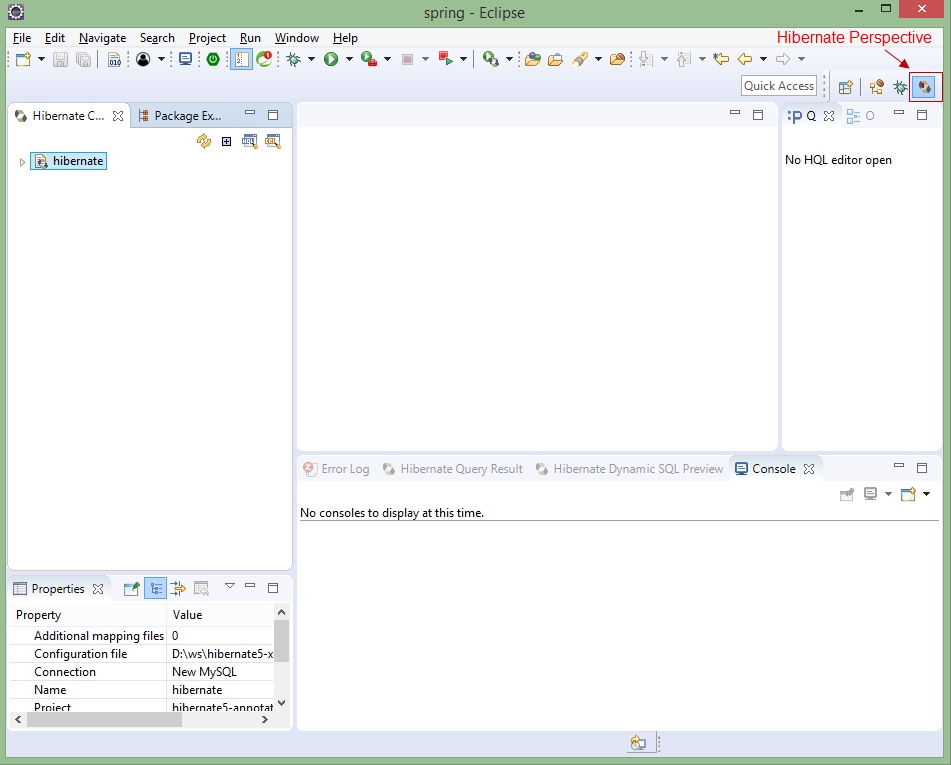
**<mapping resource="Employee.hbm.xml"/>**

**</session-factory>**

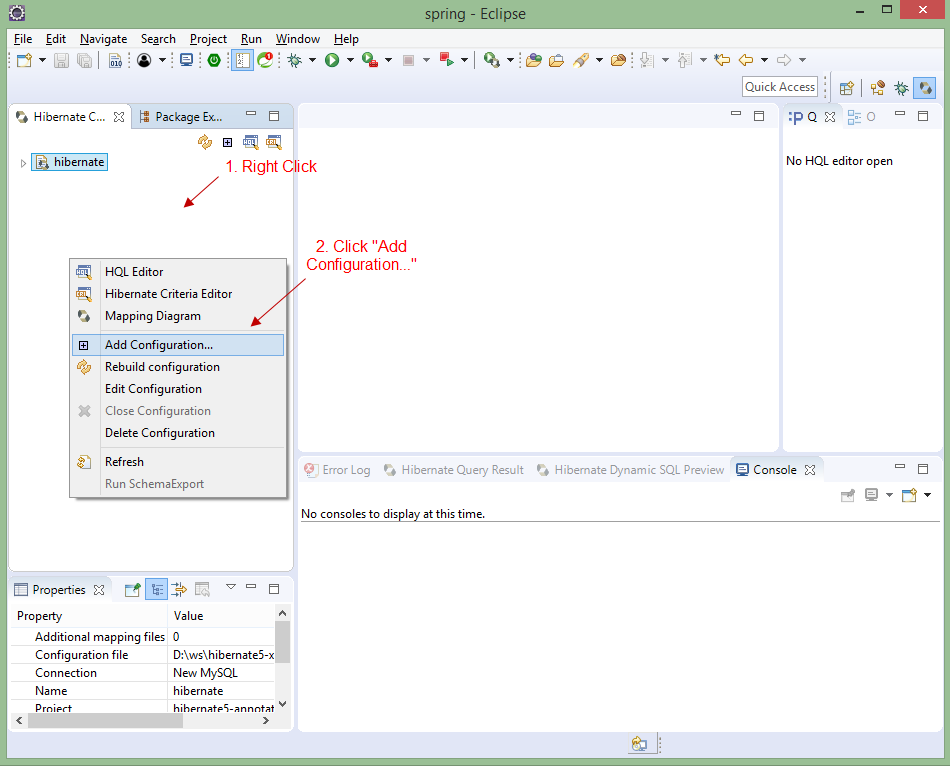
**</hibernate-configuration>**

**Cách tạo Hibernate Configuration với Hibernate Tools (hibernate.cfg.xml) với Hibernate Tools**. Tạo “hibernate.cfg.xml” kết nối với **MySQL database**:

Chọn “**Windows**” –> “**Perspective**” –> “**Open Perspective**” –> “**Others…**” –> chọn “**Hibernate**“.

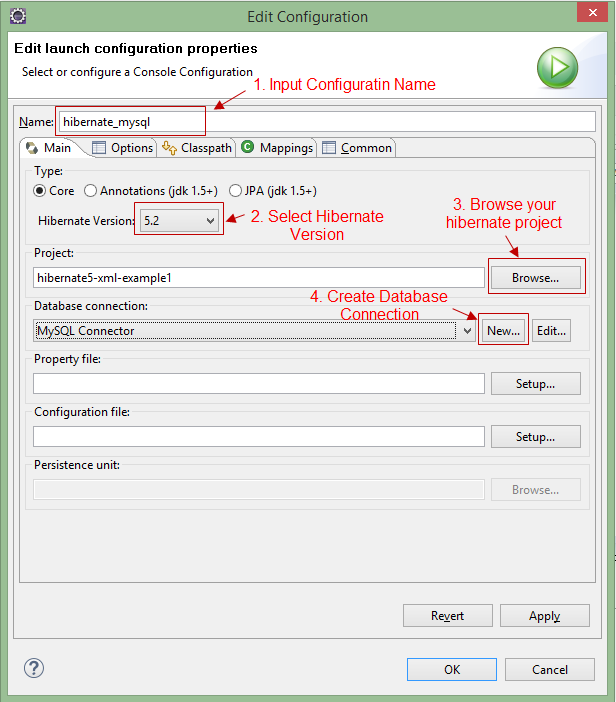


In Hibernate Configuration, right click and select “**Add Configuration…**”



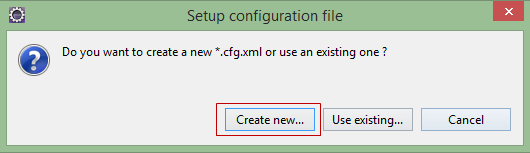
In “**Edit Configuration**” dialog box:

1. In “**Name**” box, input your configuration name.
2. In “**Hibernate Version**” dropdown, select Hibernate Version.
3. In “**Project**” box, click on the “**Browse…**” button to select your hibernate project.
4. In “**Database Connection**” box, click “**New…**” button to create your DB settings.

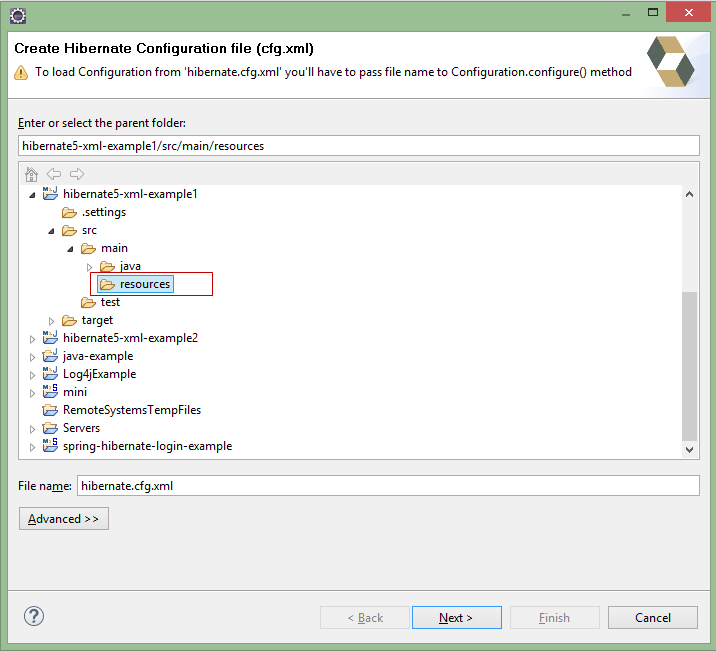


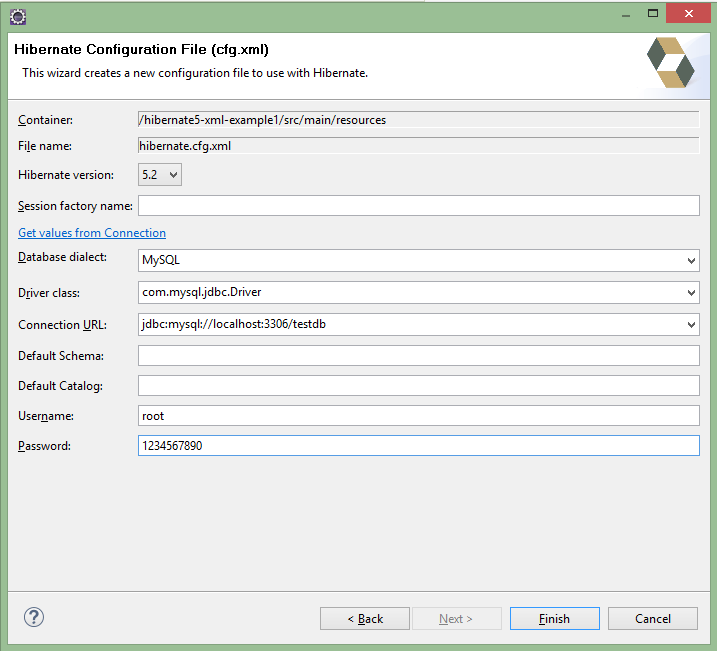
1. In “**Configuation File**” box, click “**Setup…**” button to create new or use existing hibernate configuration file (hibernate.cfg.xml).

Click “**Create new…**” to create new hibernate configuration file (hibernate.cfg.xml):



1. Select your hibernate **project and folder** to create hibernate configuration file (hibernate.cfg.xml).



1. Input your **MySql database information**:  
   
2. Click “**Finish**” button.

# **FILE MAPPING TRONG HIBERNATE**

File mapping được sử dụng để ánh xạ một hoặc nhiều lớp trong java với các bảng trong database.

Chúng ta hãy xem chi tiết hơn về các phần tử của file mapping trong Hibernate như sau:

* Tài liệu mapping là một tài liệu XML có **<hibernate-mapping>** là phần tử gốc chứa tất cả các phần tử **<class>**.
* Các phần tử **<class>** được sử dụng để định nghĩa ánh xạ cụ thể từ các lớp Java sang các bảng cơ sở dữ liệu. Tên lớp Java được chỉ định sử dụng thuộc tính **name** của phần tử lớp và tên bảng cơ sở dữ liệu được chỉ định sử dụng thuộc tính **table**.
* Phần tử **<meta>** là thành phần tùy chọn và có thể được sử dụng để tạo ra mô tả lớp.
* Phần tử **<id>** ánh xạ thuộc tính ID duy nhất trong lớp tới khóa chính của bảng cơ sở dữ liệu. Thuộc tính **name** của id id đề cập đến thuộc tính trong lớp và thuộc tính **column** đề cập đến cột trong bảng cơ sở dữ liệu. Thuộc tính **type** giữ kiểu ánh xạ hibernate, các kiểu mapping này sẽ chuyển đổi từ kiểu dữ liệu Java sang SQL.
* Phần tử **<generator>** bên trong phần tử id được sử dụng để tự động tạo giá trị cho khóa chính. Thiết lập thuộc tính **class** của phần tử generator được đặt là **native** để cho phép hibernate chọn **identity**, **sequence** hoặc **hilo** để tạo khoá chính tùy thuộc vào khả năng của cơ sở dữ liệu.
* Phần tử **<property>** được sử dụng để ánh xạ một thuộc tính của lớp Java vào một cột trong bảng cơ sở dữ liệu. Thuộc tính **name** của phần tử đề cập đến thuộc tính trong lớp và thuộc tính **column** đề cập đến cột trong bảng cơ sở dữ liệu. Thuộc tính **type** giữ kiểu ánh xạ hibernate, các kiểu mapping này sẽ chuyển đổi từ kiểu dữ liệu Java sang SQL.

Ví dụ:

**public class Employee {**

**private int id;**

**private String Name;**

**private int salary;**

**public Employee() {}**

**public Employee(String name, int salary) {**

**this.Name = name;**

**this.salary = salary;**

**}**

**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;**

**}**

**}**

|  |
| --- |
| **create table EMPLOYEE (** |

|  |
| --- |
| **id INT NOT NULL auto\_increment,** |

|  |
| --- |
| **name  VARCHAR(50) default NULL,** |

|  |
| --- |
| **salary     INT  default NULL,** |

|  |
| --- |
| **PRIMARY KEY (id)** |

|  |
| --- |
| **);** |

Dựa trên hai thực thể trên, chúng ta có thể định nghĩa file mapping XML sau, chỉ dẫn Hibernate làm thế nào để ánh xạ lớp hoặc các lớp đã định nghĩa vào các bảng cơ sở dữ liệu.

**<?xml version="1.0" encoding="utf-8"?>**

**<!DOCTYPE hibernate-mapping PUBLIC**

**"-//Hibernate/Hibernate Mapping DTD//EN"**

**"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">**

**<hibernate-mapping>**

**<class name="Employee" table="EMPLOYEE">**

**<meta attribute="class-description">**

**Lớp này chứa chi tiết về employee.**

**</meta>**

**<id name="id" type="int" column="id">**

**<generator class="native"/>**

**</id>**

**<property name="firstName" column="first\_name" type="string"/>**

**<property name="salary" column="salary" type="int"/>**

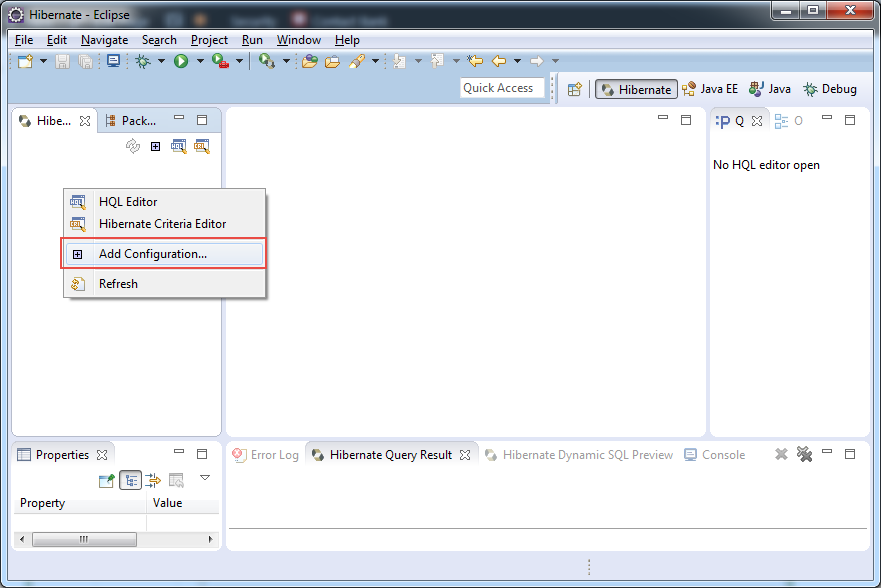
**</class>**

**</hibernate-mapping>**

Nên lưu file mapping với định dạng <classname>.hbm.xml. Trong ví dụ trên file mapping được lưu với tên Employee.hbm.xml.

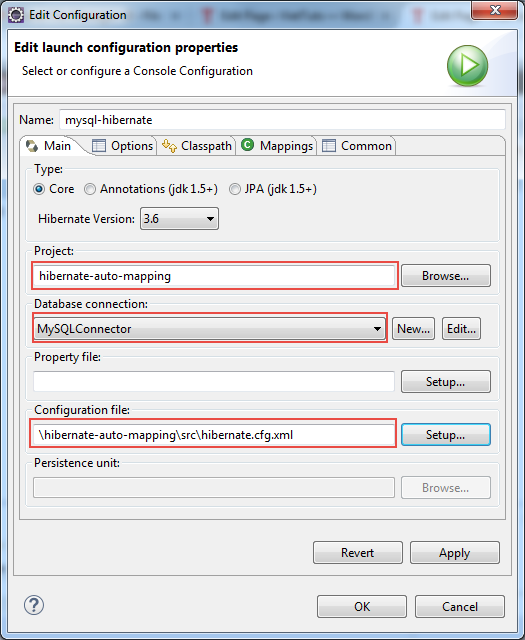
Cách sử dụng Hibernate Tools tạo các file mapping (hbm) và annotation từ các table trong database một cách tự động:

Trong Hibernate Perspective, right click và chọn **“Add Configuration…”**

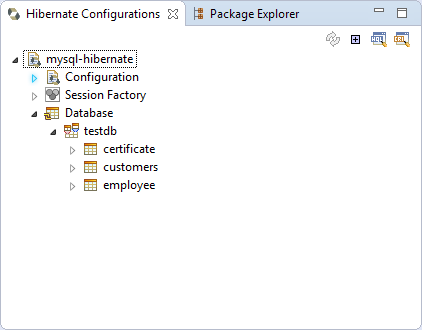


Tại **“Edit Configuration”** dialog

1. Tại hạng mục “Project”, click button “Browser…” để chọn project.
2. Tại hạng mục **“Database Connection”**, click button **“New”** để thiết lập kết nối đến database của bạn. Lưu ý: trong hướng dẫn này chúng tôi đã tạo ra một database connection có tên là “MySQLConnector”, bạn cũng có thể tạo ra các database connection cho riêng mình.
3. Tại hạng mục **“Configuration File”**, click button **“Setup”** để tạo mới hoặc sử dụng file cấu hình hibernate “hibernate.cfg.xml” đã tồn tại trong project.



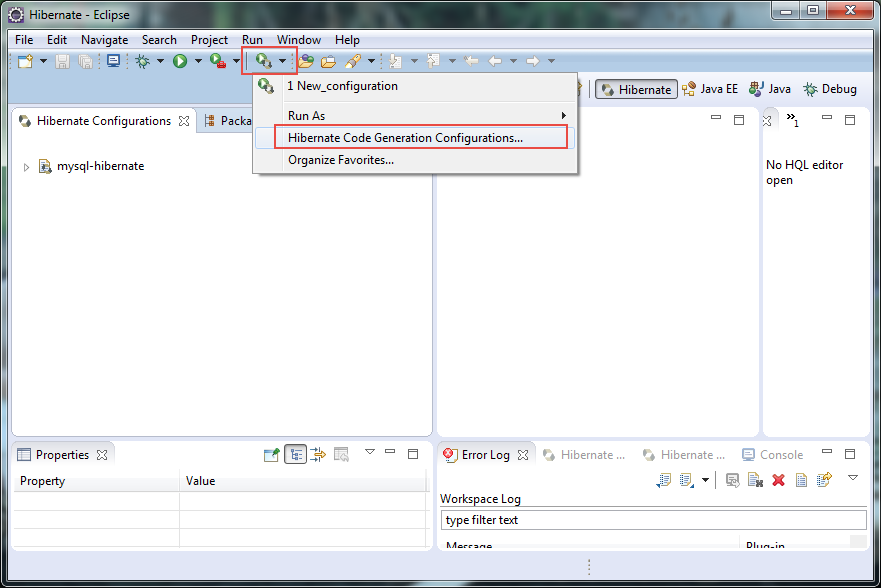
Xem danh sách các bảng trong database của bạn tại **“Hibernate Perspective”**



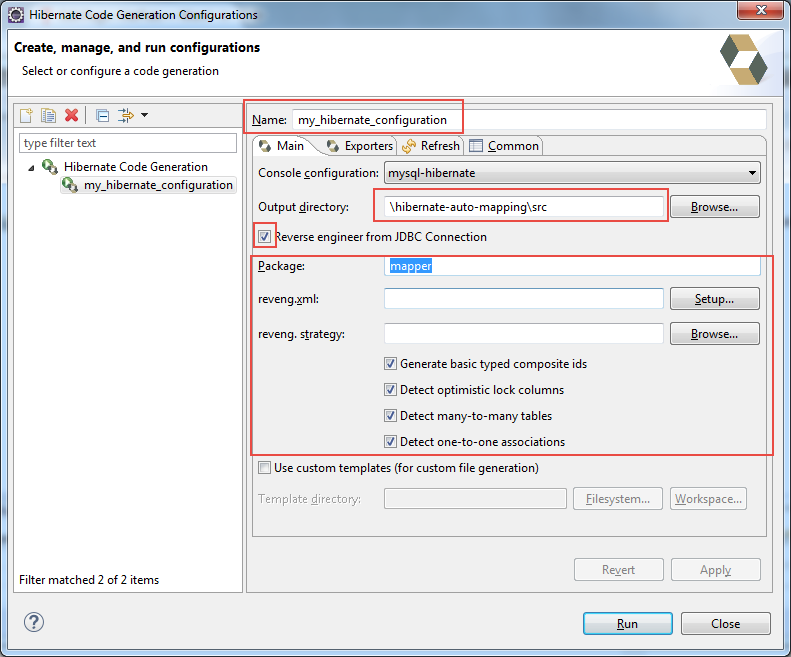
**Tạo Hibernate Code một cách tự động**

Đến bước này là bạn đã có thể tạo ra các file mapping (hbm) và annotation, bạn cũng có thể tạo ra file cấu hình hibernate “hibernate.cfg.xml” nếu như nó chưa tồn tại.

Click vào biểu tượng **“Hibernate code generation”**, sau đó click **“Hibernate Code Generation Configuration”**.

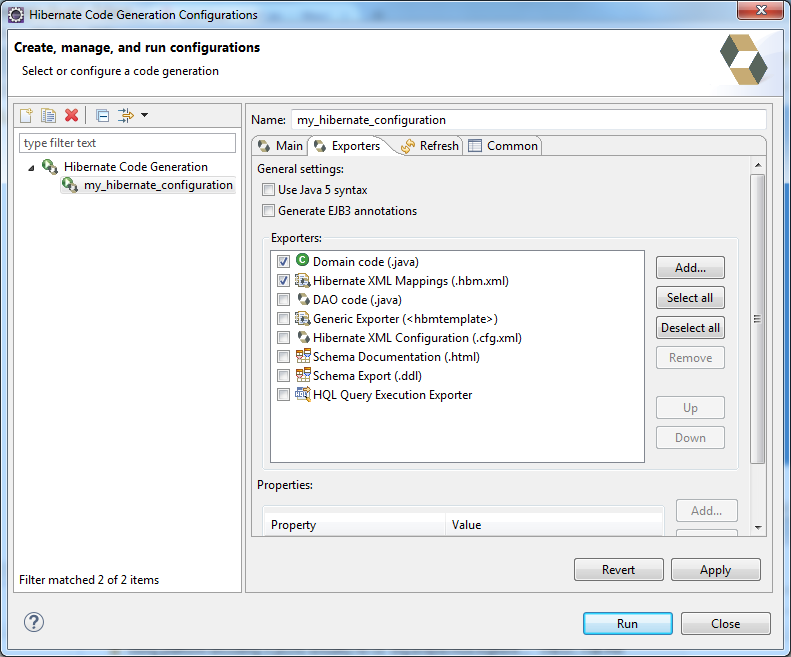


Tạo new configuration: Nhập hạng mục **Name**, sau đó chọn tab **“Main”**, rồi nhập hạng mục **“Output directory”** – nơi lưu các file được tạo ra, và check vào tùy chọn **“Reverse engineer from JDBC Connection”**.



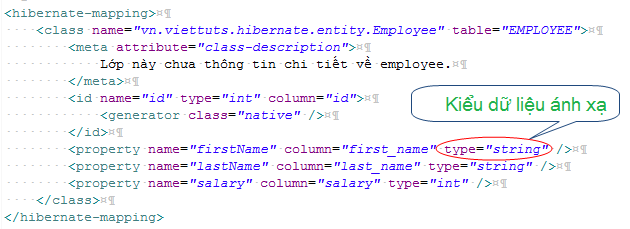
Tại tab **“Exporter”**, chọn những gì bạn muốn Hibernate Tools tạo ra cho bạn chẳng hạn như Entity, mapping file (.hbm.xml) , DAO, annotation code, …

Cuối cùng, click button **“Run”**



# **Thuộc tính type**

Khi bạn chuẩn bị tạo tài liệu mapping trong Hibernate, bạn phải ánh xạ các kiểu dữ liệu Java vào các kiểu dữ liệu RDBMS. Các kiểu khai báo và sử dụng trong các file mapping không phải là các kiểu dữ liệu Java; Chúng cũng không phải là các kiểu cơ sở dữ liệu SQL. Các loại này được gọi là các kiểu dữ liệu ánh xạ Hibernate, có thể dịch từ kiểu dữ liệu Java sang SQL và ngược lại.



Trong bài này liệt kê tất cả các kiểu dữ liệu cơ bản, date and time, đối tượng, và các mapping dựng sẵn khác.

**Các kiểu dữ liệu nguyên thủy**

|  |  |  |
| --- | --- | --- |
| **Mapping type** | **Java type** | **ANSI SQL Type** |
| integer | int hoặc java.lang.Integer | INTEGER |
| long | long hoặc java.lang.Long | BIGINT |
| short | short hoặc java.lang.Short | SMALLINT |
| float | float hoặc java.lang.Float | FLOAT |
| double | double hoặc java.lang.Double | DOUBLE |
| big\_decimal | java.math.BigDecimal | NUMERIC |
| character | java.lang.String | CHAR(1) |
| string | java.lang.String | VARCHAR |
| byte | byte hoặc java.lang.Byte | TINYINT |
| boolean | boolean hoặc java.lang.Boolean | BIT |
| yes/no | boolean hoặc java.lang.Boolean | CHAR(1) (‘Y’ hoặc ‘N’) |
| true/false | boolean hoặc java.lang.Boolean | CHAR(1) (‘T’ hoặc ‘F’) |

**Các kiểu Date and time**

|  |  |  |
| --- | --- | --- |
| **Mapping type** | **Java type** | **ANSI SQL Type** |
| date | java.util.Date hoặc java.sql.Date | DATE |
| time | java.util.Date hoặc java.sql.Time | TIME |
| timestamp | java.util.Date hoặc java.sql.Timestamp | TIMESTAMP |
| calendar | java.util.Calendar | TIMESTAMP |
| calendar\_date | java.util.Calendar | DATE |

**Các kiểu Binary và đối tượng lớn**

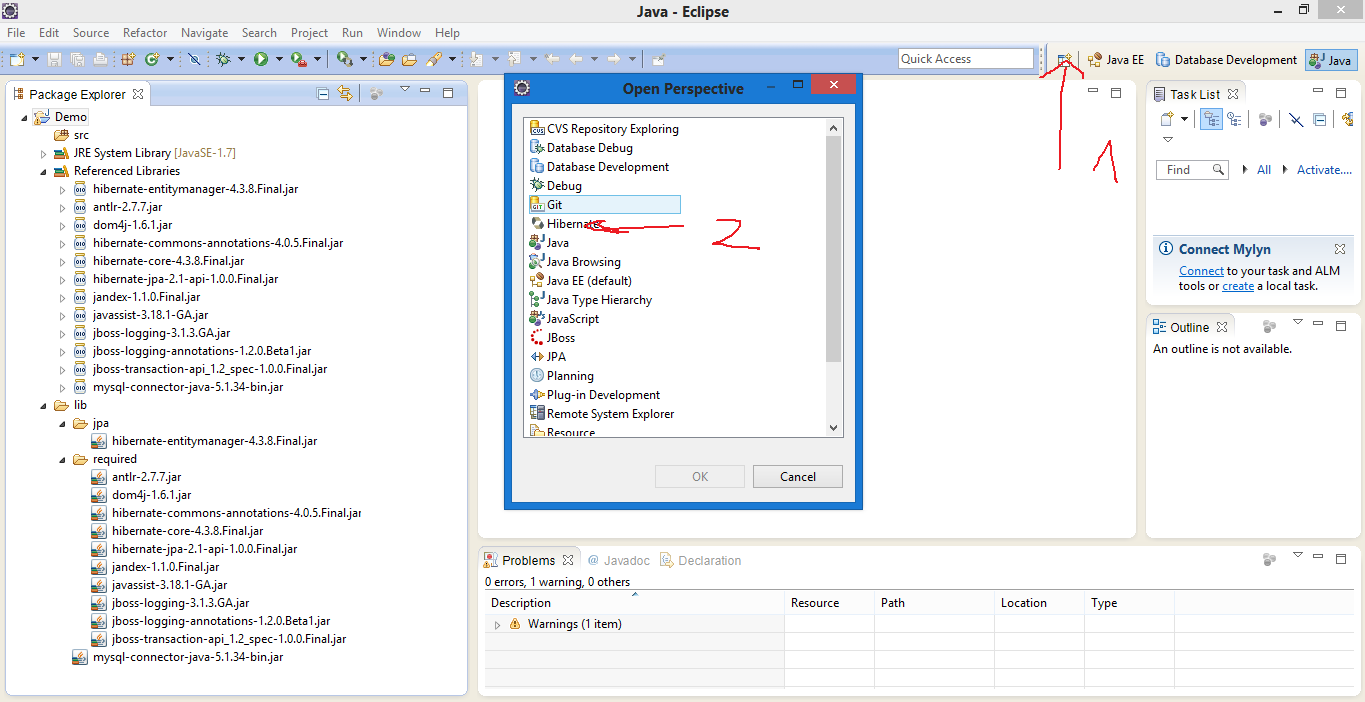
|  |  |  |
| --- | --- | --- |
| **Mapping type** | **Java type** | **ANSI SQL Type** |
| binary | byte[] | VARBINARY (hoặc BLOB) |
| Text | java.lang.String | CLOB |
| serializable | any Java class that implements java.io.Serializable | VARBINARY (hoặc BLOB) |
| Clob | java.sql.Clob | CLOB |
| Blob | java.sql.Blob | BLOB |

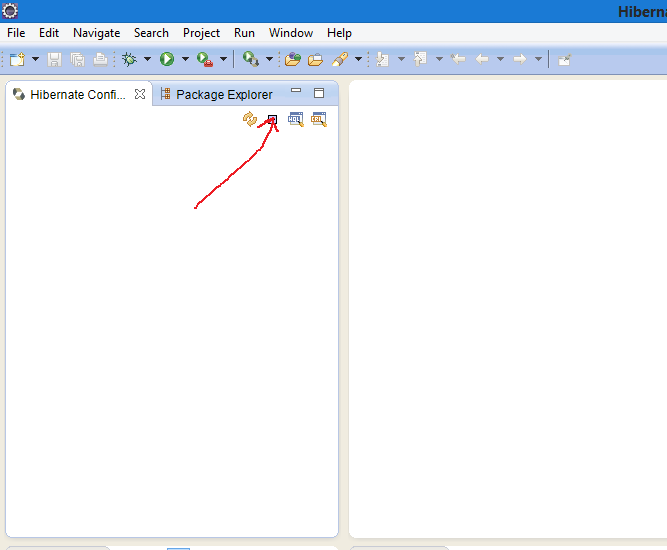
**Các kiểu JDK-related**

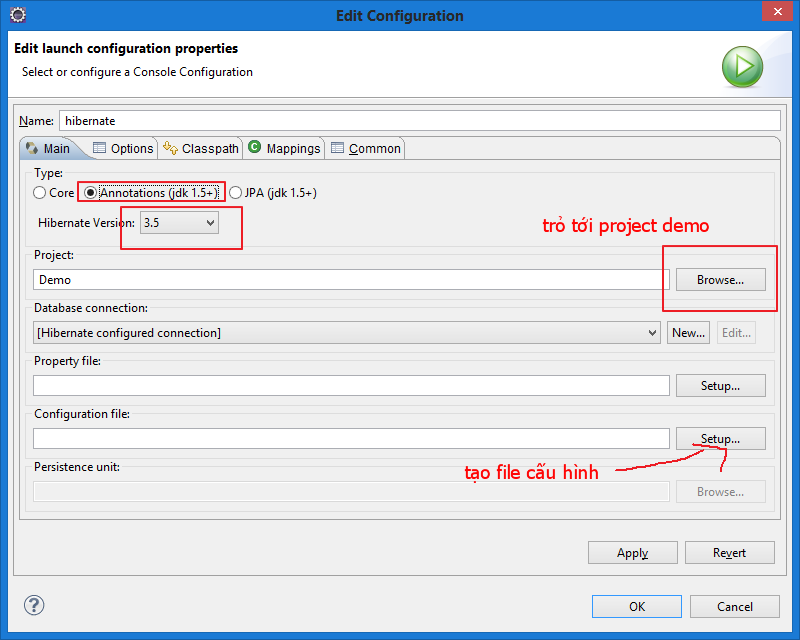
|  |  |  |
| --- | --- | --- |
| **Mapping type** | **Java type** | **ANSI SQL Type** |
| class | java.lang.Class | VARCHAR |
| locale | java.util.Locale | VARCHAR |
| timezone | java.util.TimeZone | VARCHAR |
| currency | java.util.Currency | VARCHAR |

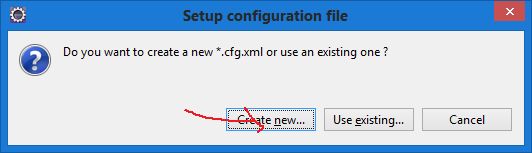
# **Cách ánh xạ từ bảng trong csdl sang entity class**

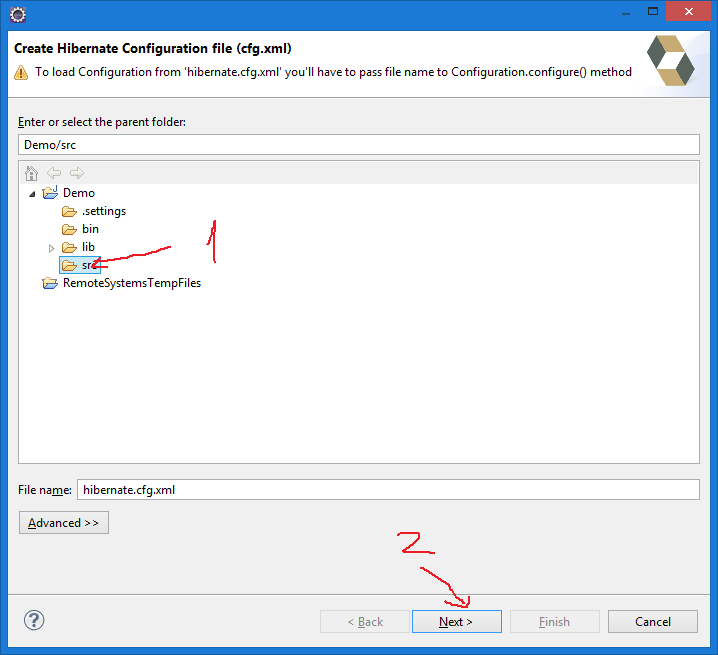
Thực hiện kết nối database và tạo ra các entity trên project vừa tạo.

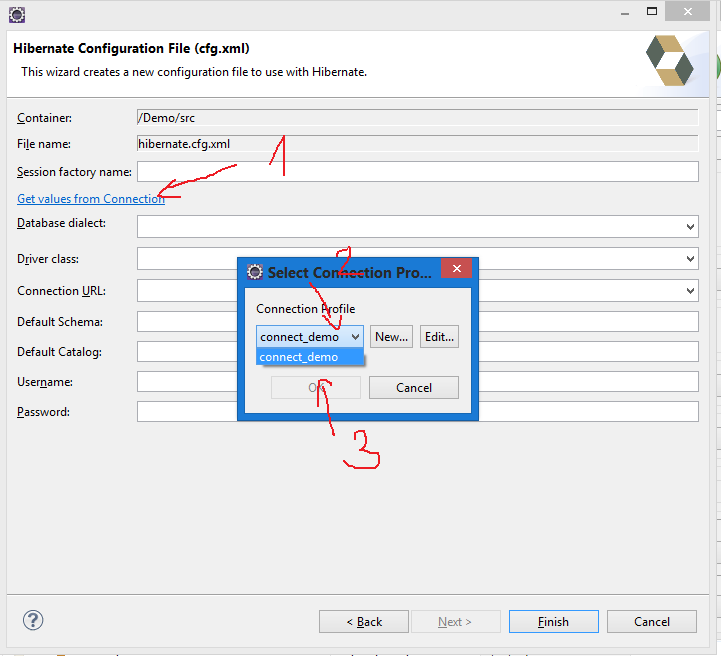


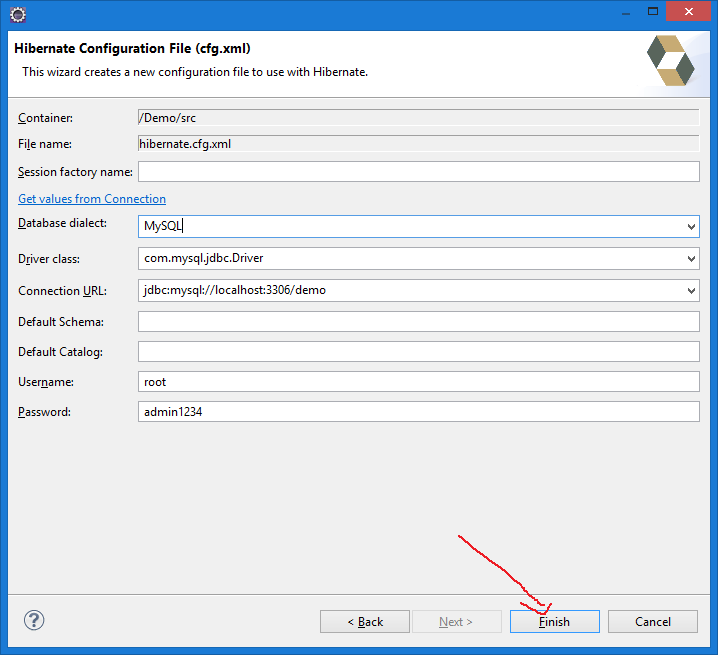


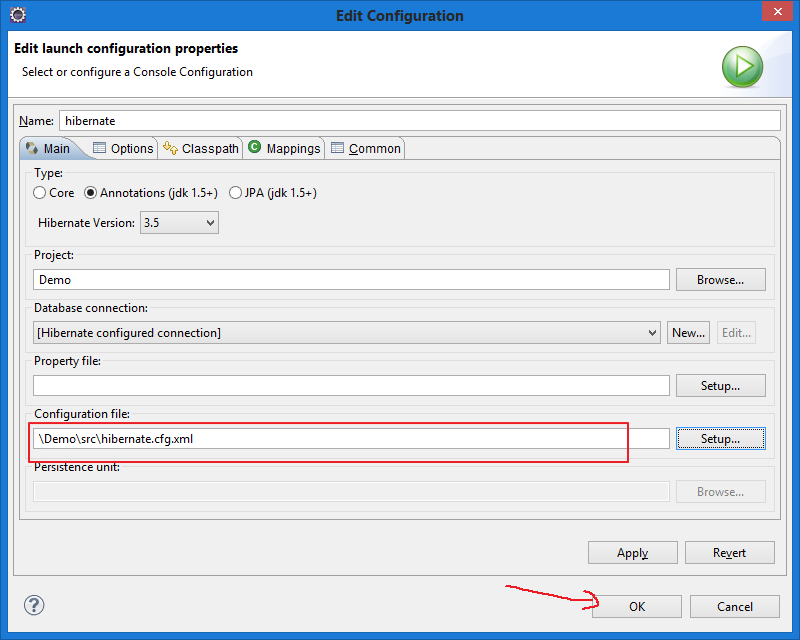


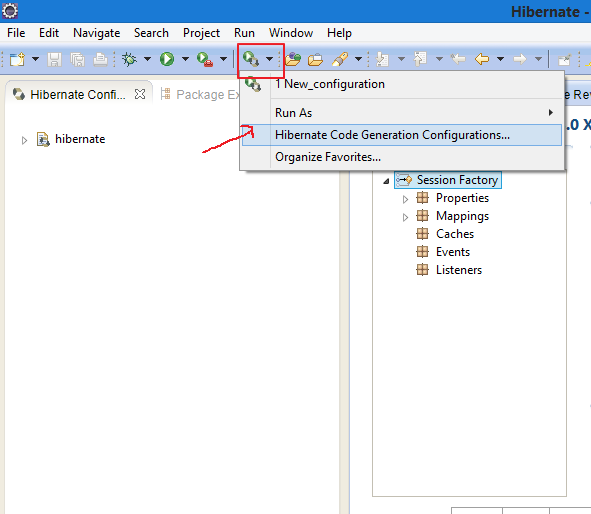


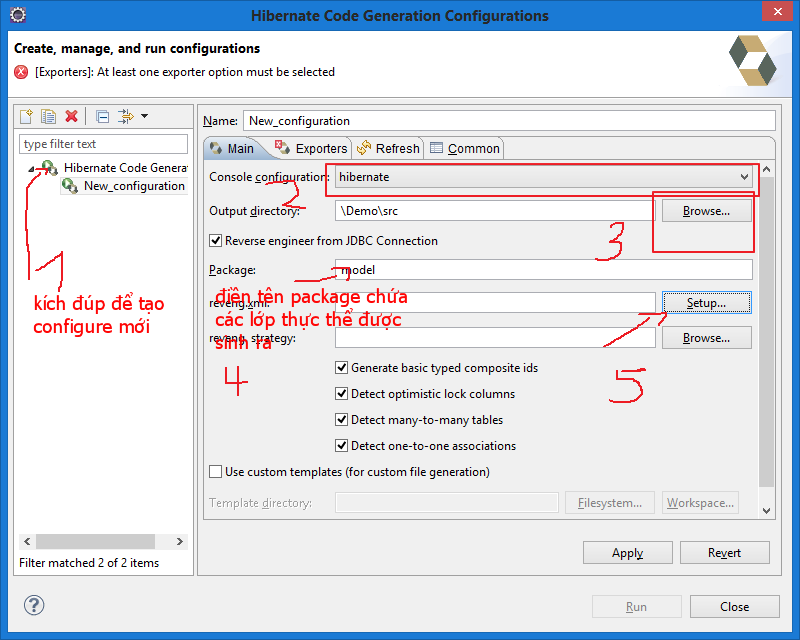


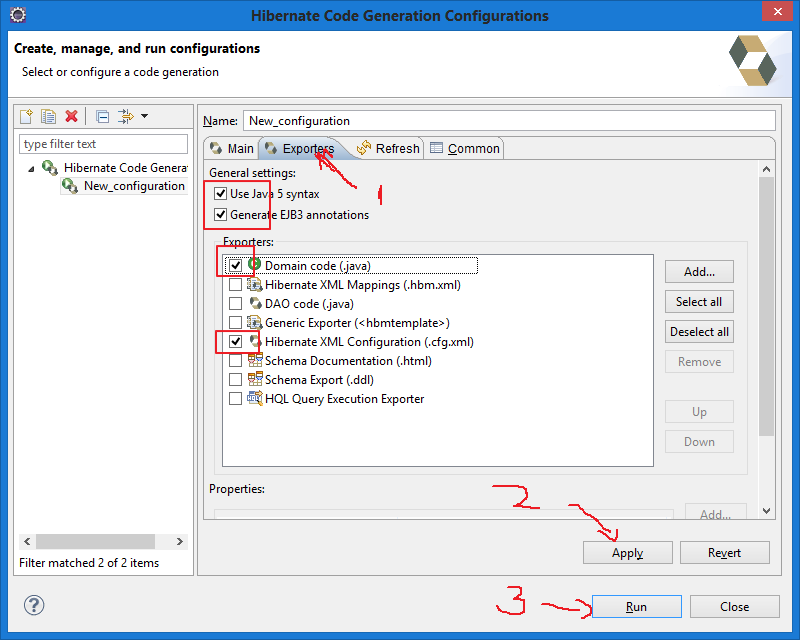


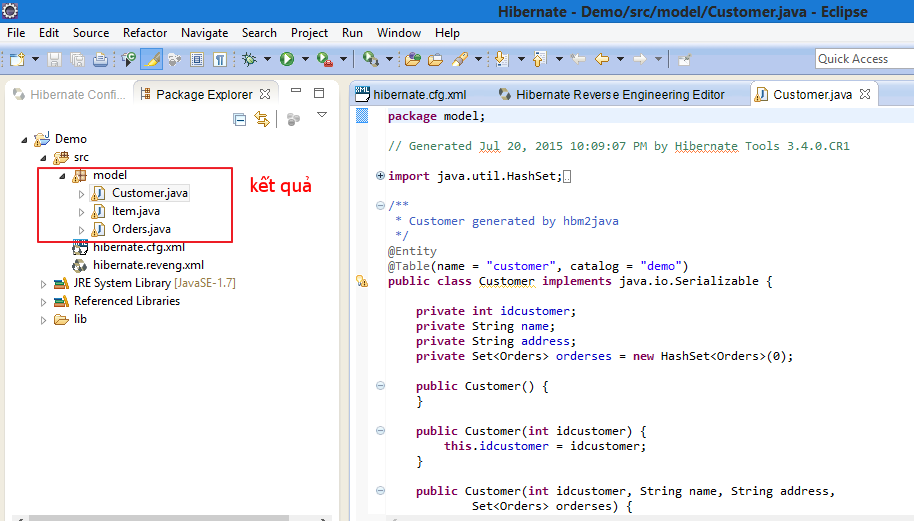












# **Lớp persistent**

Các lớp Java có các đối tượng hoặc các thể hiện sẽ được lưu trữ trong các bảng cơ sở dữ liệu được gọi là các **lớp persistent** trong Hibernate. Hibernate hoạt động tốt nhất nếu các lớp này tuân theo một số quy tắc đơn giản. Có các quy tắc chính sau đây của các lớp persistent:

* Tất cả các lớp Java persistent cần có constructor mặc định.
* Tất cả các lớp phải có một ID để cho phép dễ dàng xác định các đối tượng của bạn trong Hibernate và cơ sở dữ liệu. Thuộc tính này ánh xạ tới cột khoá chính của một bảng cơ sở dữ liệu (optional).
* Tất cả các thuộc tính phải có phương thức setter và getter.
* Lớp persistent là non-final hoặc được impliments từ interface có các phương thức là public (optional).
* Tất cả các lớp không extend hoặc implement một vài lớp và interface đặc biệt được yêu cầu bởi EJB framework.

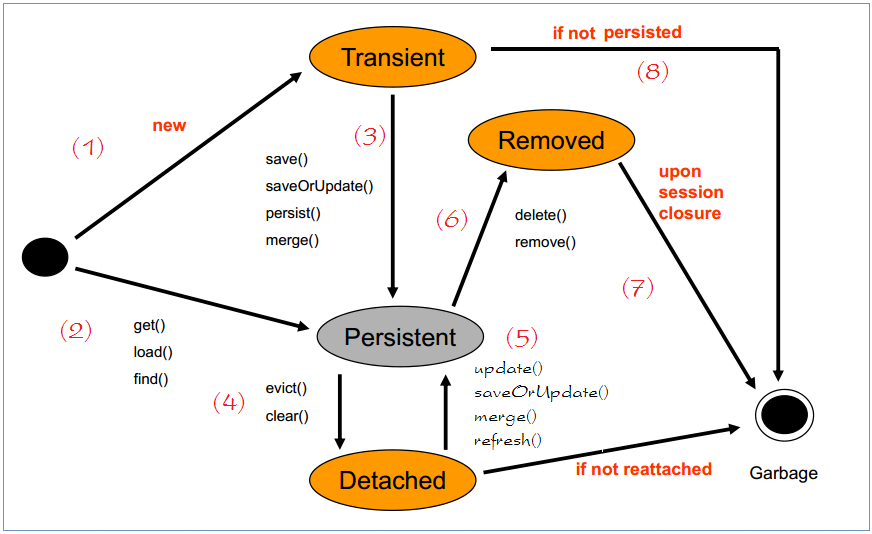
**Đối tượng persistent** có nghĩa là các đối tượng cụ thể có thể sống trong ứng dụng. Chúng có thể được lưu vào một data store và được tái tạo tại một điểm sau này.

# **Sessions**

Một Session trong Hibernate được sử dụng để có được một kết nối vật lý với một cơ sở dữ liệu. Đối tượng Session là nhẹ và được thiết kế để được khởi tạo mỗi khi cần tương tác với cơ sở dữ liệu. Các đối tượng persistent được lưu và truy xuất thông qua một đối tượng Session.

Các đối tượng session không nên được mở trong một thời gian dài bởi vì chúng không phải là thread an toàn và nên được tạo ra và đóng khi sau khi sử dụng. Chức năng chính của session là cung cấp các thao tác create, update, read và delete cho các thể hiện các lớp thực thể được ánh xạ. Các thể hiện có thể tồn tại ở một trong ba trạng thái sau đây tại một thời điểm nhất định:

* **transient:** Một thể hiện mới của một lớp persistent không liên quan đến một Session, không có đại diện trong cơ sở dữ liệu và không có giá trị định danh được coi là transient (tạm thời) bởi Hibernate.
* **persistent:** Bạn có thể tạo một thể hiển transient persistent bằng cách liên kết nó với một session. Một thể hiện persistent có một dại diện trong cơ sở dữ liệu, một giá trị định danh và được liên kết với một Session.
* **detached:** Khi chúng ta đóng session trong Hibernate, thể hiện persistent sẽ trở thành một thể hiện detached.
* **Removed**: Đã bị xóa - dưới DB



### Transient -> Persistent

**Session** gọi một trong các phương thức **save, saveOrUpdate, persist, merge** sẽ đẩy đối tượng **Transient** vào sự quản lý của **Hibernate** và đối tượng này chuyển sang trạng thái **Persistent**. Nó tương đương với hành động **insert** hoặc **update** trong **Database**.

### Persistent -> Detached

Đối tượng **Session** gọi phương thức **evict(..)** hoặc **clear()** để đuổi các đối tượng có trạng thái **Persistent** ra khỏi sự quản lý của **Hibernate**, giờ các đối tượng này sẽ có trạng thái mới là **Detached** (Bị tách ra).  Nếu nó không được đính lại ( **Re-Attached**), nó sẽ bị bộ gom rác của **Java** ( **Java Garbage Collector**) loại bỏ theo cơ chế thông thường.

### Detached -> Persistent

Sử dụng một trong các phương thức: **update(..), saveOrUpdate(..), merge(..)** sẽ đưa một đối tượng có trạng thái **Detached** vào sự quản lý của **Hibernate**. Nó tương đương với hành động **update** hoặc **insert** dưới **Database**. Đối tượng sẽ chuyển sang trạng thái **Persistent**.

### **Persistent -> Removed**

Đối tượng **Session** gọi phương thức **remove(..)** hoặc **delete(..)** để xóa một đối tượng (Bản ghi), đối tượng **Persistent** sẽ chuyển sang trạng thái **Removed** (Đã bị loại bỏ).

Một transaction thông thường nên sử dụng cấu trúc sau đây:

**Session session = factory.openSession();**

**Transaction tx = null;**

**try {**

**tx = session.beginTransaction();**

**// do some work……**

**tx.commit();**

**}**

**catch (Exception e) {**

**if (tx!=null) tx.rollback();**

**e.printStackTrace();**

**}finally { session.close(); }**

Nếu session ném một ngoại lệ, transaction phải được rollback và session phải là detached.

**Các phương thức của interface Session**

Có nhiều phương thức được cung cấp bởi giao tiếp Session nhưng bảng dưới đây chỉ liệt kê vài phương thức quan trọng.

|  |  |
| --- | --- |
|  | **Các phương thức của Session và mô tả** |
| 1 | **Transaction beginTransaction()**  Bắt đầu một đơn vị công việc và trả về đối tượng Transaction liên quan. |
| 2 | **void cancelQuery()** Hủy việc thực hiện truy vấn hiện tại. |
| 3 | **void clear()** clear session. |
| 4 | **Connection close()** Đóng session. |
| 5 | **Criteria createCriteria(Class persistentClass)** Tạo một thể hiện Criteria, đối với lớp entity đã cho hoặc một superclass của một lớp entity. |
| 6 | **Criteria createCriteria(String entityName)**  Tạo một thể hiện Criteria, đối với entity name đã cho. |
| 7 | **void update(String entityName, Object object)** Update persistent instance với bộ nhận dạng của detached instance đã cho. |
| 8 | **Query createFilter(Object collection, String queryString)**  Tạo một thể hiện Query mới cho collection và chuỗi lọc nhất định. |
| 9 | **Query createQuery(String queryString)**  Tạo một thể hiện Query mới cho chuỗi truy vấn HQL đã cho. |
| 10 | **SQLQuery createSQLQuery(String queryString)**  Tạo một thể hiện SqlQuery mới cho chuỗi truy vấn SQL đã cho. |
| 11 | **void delete(Object object)** Xóa persistent instance từ datastore. |
| 12 | **void delete(String entityName, Object object)**  Xóa persistent instance từ datastore. |
| 13 | **Session get(String entityName, Serializable id)**  Trả về persistent instance của thực thể được đặt tên với một số nhận dạng nhất định, hoặc null nếu không có persistent instance. |
| 14 | **SessionFactory getSessionFactory()**  Get session factory cái mà tạo ra session này. |
| 15 | **void refresh(Object object)**  Đọc lại trạng thái của thể hiện đã cho từ cơ sở dữ liệu cơ bản. |
| 16 | **Transaction getTransaction()**  Get thể hiện của Transaction liên quan đến session này. |
| 17 | **boolean isConnected()** Kiểm tra xem session này có đang kết nối không. |
| 18 | **boolean isDirty()** Session này có chứa bất kỳ thay đổi nào phải được đồng bộ với cơ sở dữ liệu không? |
| 19 | **boolean isOpen()** Kiểm tra xem session vẫn đang được open không. |
| 20 | **void saveOrUpdate(Object object)**  Thực hiện save(Object) hoặc update(Object). |
| 21 | **void update(Object object)** Update persistent instance với bộ nhận dạng của detached instance đã cho. |

* + - 1. **ORM trong Hibernate**

Có 3 chủ đề O/R mapping trong Hibernate quan trọng nhất mà chúng ta phải tìm hiểu chi tiết là:

## a .Collections mapping

* Nếu một entity hoặc lớp có một tập các giá trị trong một biến Collection cụ thể, thì chúng ta có thể ánh xạ các giá trị đó bằng bất kỳ một trong các interface collections có sẵn trong java.
* Hibernate có thể persist các thể hiện của **java.util.Map, java.util.Set, java.util.SortedMap, java.util.SortedSet, java.util.List** và bất kỳ mảng nào của các entity hoặc giá trị persistent.

|  |  |
| --- | --- |
| **Kiểu Collection** | **Mapping và môt tả** |
| [java.util.Set](https://viettuts.vn/hibernate/hibernate_set_mapping.htm) | Điều này được ánh xạ với một phần tử <set> và được khởi tạo với java.util.HashSet |
| [java.util.SortedSet](https://viettuts.vn/hibernate/hibernate_sortedset_mapping.htm) | Điều này được ánh xạ với một phần tử <set> và được khởi tạo với java.util.TreeSet. Thuộc tính **sort** để định nghĩa một tập được sắp xếp hoặc không. |
| [java.util.List](https://viettuts.vn/hibernate/hibernate_list_mapping.htm) | Điều này được ánh xạ với một phần tử <list> và được khởi tạo với java.util.ArrayList |
| [java.util.Collection](https://viettuts.vn/hibernate/hibernate_bag_mapping.htm) | Điều này được ánh xạ với một phần tử <bag> or <ibag> và được khởi tạo với java.util.ArrayList |
| [java.util.Map](https://viettuts.vn/hibernate/hibernate_map_mapping.htm) | Điều này được ánh xạ với một phần tử <map> và được khởi tạo với java.util.HashMap |
| [java.util.SortedMap](https://viettuts.vn/hibernate/hibernate_sortedmap_mapping.htm) | Điều này được ánh xạ với một phần tử <map> và được khởi tạo với java.util.TreeMap. Thuộc tính **sort** để định nghĩa một tập được sắp xếp hoặc không. |

* Mảng được Hibernate hỗ trợ với <primitive-array> cho các kiểu giá trị Java nguyên thủy và <array> cho các kiểu dữ liệu khác. Tuy nhiên, chúng hiếm khi được sử dụng.

## b. Association Mapping

Việc ánh xạ các liên kết giữa các lớp entity và các mối quan hệ giữa các bảng chính là linh hồn của ORM. Dưới đây là 4 kiểu liên kết:

|  |  |
| --- | --- |
| [Many-to-One](https://viettuts.vn/hibernate/hibernate-many-to-one-relationship-voi-mysql-tren-eclipse) | Mapping mối quan hệ many-to-one trong Hibernate |
| [One-to-One](https://viettuts.vn/hibernate/hibernate-one-to-one-relationship-voi-mysql-tren-eclipse) | Mapping mối quan hệ one-to-one trong Hibernate |
| [One-to-Many](https://viettuts.vn/hibernate/hibernate-one-to-many-relationship-voi-mysql-tren-eclipse) | Mapping one-to-many relationship using Hibernate |
| [Many-to-Many](https://viettuts.vn/hibernate/hibernate-many-to-many-relationship-voi-mysql-tren-eclipse) | Mapping mối quan hệ many-to-many trong Hibernate |

## c. Component Mapping

Trường hợp một lớp Entity có thể có một tham chiếu đến một lớp khác như là một biến thành viên. Nếu lớp được tham chiếu đó không có vòng đời của chính nó và hoàn toàn phụ thuộc vào vòng đời của lớp thực thể sở hữu, lớp được tham chiếu đó được gọi là lớp Component.

Việc ánh xạ Collection của Component kiểu này cũng tương tự như việc ánh xạ một collection bình thường.

[**Component Mappings**](https://viettuts.vn/hibernate/hibernate_component_mappings.htm): Mapping cho một lớp có một tham chiếu đến một lớp khác như là một biến thành viên. Trường hợp này chúng ta sử dụng phần tử **<component>** để thiết lập sự tồn tại của các thuộc khác của lớp của biến tham chiếu bên trong lớp sở hữu nó.

# **Annotations**

Trong việc phát triển các thư viện, framework, công cụ,… chúng ta cần đánh dấu các hàm, class hoặc phương thức để có thể tìm kiếm đúng và nhanh hơn cho một tiến trình xử lý.

Cho đến nay, bạn đã thấy cách Hibernate sử dụng file mapping XML để chuyển đổi dữ liệu từ Java POJO sang các bảng cơ sở dữ liệu và ngược lại. Hibernate Annotations là cách mới nhất để định nghĩa các mapping mà không cần sử dụng file XML. Bạn có thể sử dụng các annotation bổ sung hoặc để thay thế siêu dữ mapping XML.

Hibernate Annotations là cách mạnh mẽ để cung cấp siêu dữ liệu cho việc mapping đối tượng và bảng quan hệ. Tất cả các siêu dữ liệu được định nghĩa vào file java POJO cùng với mã này giúp người sử dụng đồng thời hiểu được cấu trúc bảng và POJO trong quá trình phát triển.

Nếu bạn muốn ứng dụng của bạn chuyển sang các ứng dụng ORM tương thích khác của EJB 3, bạn phải sử dụng các annotation để biểu diễn các thông tin mapping nhưng vẫn cần sử dụng các mapping dựa trên XML.

**Nội dung chính:**

## Thiết lập môi trường cho Hibernate Annotation:

## Trước hết, bạn phải đảm bảo rằng bạn đang sử dụng JDK 5.0 nếu không bạn cần phải nâng cấp JDK của mình lên JDK 5.0 hoặc bản mới hơn để được hỗ trợ sử dụng annotation.

## Thứ hai, bạn sẽ cần phải cài đặt gói Hibernate 3.x annotations, có sẵn từ sourceforge ([Download Hibernate Annotations](https://sourceforge.net/projects/hibernate/files/hibernate-annotations/)) và sao chép hibernate-annotations.jar, lib/hibernate-comons-annotations.jar và lib/ejb3-persistence.jar từ file tải về đến CLASSPATH của bạn.Hoặc bạn có thể add các dependency vào file pom.xml như sau:

**<dependencies>**

**<dependency>**

**<groupId>**org.hibernate**</groupId>**

**<artifactId>**hibernate-core**</artifactId>**

**<version>**3.6.3.Final**</version>**

**</dependency>**

**<dependency>**

**<groupId>**org.hibernate**</groupId>**

**<artifactId>**hibernate-annotations**</artifactId>**

**<version>**3.5.6-Final**</version>**

**</dependency>**

**<dependency>**

**<groupId>**javassist**</groupId>**

**<artifactId>**javassist**</artifactId>**

**<version>**3.12.1.GA**</version>**

**</dependency>**

**<dependency>**

**<groupId>**mysql**</groupId>**

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

**<version>**5.1.6**</version>**

**</dependency>**

**</dependencies>**

## Tạo lớp Entity với các annotation

Như đã đề cập ở trên trong khi làm việc với Hibernate Annotation tất cả các siêu dữ liệu được định nghĩa vào file java POJO cùng với mã này giúp người sử dụng đồng thời hiểu được cấu trúc bảng và lớp POJO trong quá trình phát triển.

Tạo bảng **EMPLOYEE** để lưu trữ đối tượng

|  |
| --- |
| create table testdb.**EMPLOYEE** ( |
| id **INT NOT NULL** auto\_increment, |

|  |
| --- |
| first\_name **VARCHAR**(20) default **NULL**, |
| last\_name  **VARCHAR**(20) default **NULL**, |

|  |
| --- |
| salary     **INT**  default **NULL**, |
| **PRIMARY KEY** (id) |

|  |
| --- |
| ); |

Sau đây là việc mapping lớp Employee với các annotation để ánh xạ các đối tượng với bảng **EMPLOYEE** được định nghĩa ở trên:

**import javax.persistence.\*;**

**@Entity**

**@Table(name = "EMPLOYEE")**

**public class Employee {**

**@Id @GeneratedValue**

**@Column(name = "id")**

**private int id;**

**@Column(name = "first\_name")**

**private String firstName;**

**@Column(name = "last\_name")**

**private String lastName;**

**@Column(name = "salary")**

**private int salary;**

**public Employee() {}**

**public int getId() {**

**return id;**

**}**

**public void setId( int id ) {**

**this.id = id;**

**}**

**public String getFirstName() {**

**return firstName;**

**}**

**public void setFirstName( String first\_name ) {**

**this.firstName = first\_name;**

**}**

**public String getLastName() {**

**return lastName;**

**}**

**public void setLastName( String last\_name ) {**

**this.lastName = last\_name;**

**}**

**public int getSalary() {**

**return salary;**

**}**

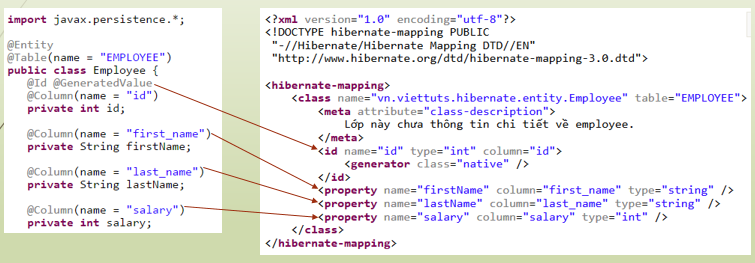
**public void setSalary( int salary ) {**

**this.salary = salary;**

**}**

**}**

Sử dụng chú thích trên tương đương với file mapping xml như sau:



Dưới đây giải thích các chú thích được sử dụng trong lớp trên:

**@Entity Annotation:**

Các chú thích tiêu chuẩn EJB 3 được chứa trong gói javax.persistence, vì vậy trước tiên chúng ta phải import gói này. Thứ hai, chúng ta đã sử dụng chú thích Entity vào lớp Employee đánh dấu lớp này như một Entity Bean, do đó nó phải có một constructor không có đối số mà có kiểu là public.

**@Table Annotation:**

Chú thích @Table cho phép bạn chỉ định các chi tiết của bảng sẽ được sử dụng để lưu trữ thực thể trong cơ sở dữ liệu. Chú thích @Table cung cấp bốn thuộc tính, cho phép bạn ghi đè tên của bảng, danh mục, lược đồ, bắt buộc các ràng buộc duy nhất trên các cột trong bảng. Trong lớp Employee ở trên chúng ta đang sử dụng tên bảng là EMPLOYEE.

**@Id and @GeneratedValue Annotations:**

Mỗi entity bean sẽ có một khóa chính, mà bạn chú thích trên lớp với chú thích @Id. Khóa chính có thể là một trường duy nhất hoặc kết hợp nhiều trường tùy thuộc vào cấu trúc bảng của bạn.

Theo mặc định, chú thích @Id sẽ tự động xác định chiến lược tạo primary key, nhưng bạn có thể ghi đè bằng cách áp dụng chú thích **@GeneratedValue** có hai tham số **strategy** và **generator**. Chúng ta chỉ sử dụng chiến lược tạo key mặc định. Cho phép Hibernate xác định loại kiểu tạo nào để sử dụng làm cho mã di chuyển giữa các cơ sở dữ liệu khác nhau.

**@Column Annotation:**

Chú thích @Column được sử dụng để chỉ định chi tiết của cột mà trường hoặc thuộc tính sẽ được ánh xạ. Bạn có thể sử dụng chú thích cột với các thuộc tính được sử dụng phổ biến nhất sau đây:

* Thuộc tính **name** được sử dụng để chị định tên cột nào trong db map với tên trường được chú thích.
* Thuộc tính **length** cho phép kích thước của cột được sử dụng để ánh xạ một giá trị đặc biệt cho một giá trị String.
* Thuộc tính **nullable** cho phép cột được đánh dấu KHÔNG NULL khi schema được tạo ra.
* Thuộc tính **unique** cho phép cột được đánh dấu chỉ chứa các giá trị duy nhất.

# **QUERY LANGUAGE**

**Hibernate Query Language (HQL)** là một ngôn ngữ truy vấn hướng đối tượng, tương tự như SQL nhưng thay vì làm việc trên các bảng và cột, HQL làm việc với các đối tượng persistent và các thuộc tính của chúng. Các truy vấn HQL được dịch bởi Hibernate thành các truy vấn SQL thông thường, lần lượt thực hiện các công việc trên cơ sở dữ liệu.

Mặc dù bạn có thể sử dụng các câu lệnh SQL trực tiếp với Hibernate bằng cách sử dụng [**Native SQL**](https://viettuts.vn/hibernate/native-sql-trong-hibernate)**.** Nhưng chúng tôi khuyên bạn nên sử dụng HQL bất cứ khi nào có thể để tránh sự phức tạp về tính linh hoạt của cơ sở dữ liệu và tận dụng chiến lược generation và caching của Hibernate.

Các từ khoá như SELECT, FROM và WHERE vv không phân biệt chữ hoa và chữ thường nhưng các thuộc tính như tên bảng và cột có phân biệt chữ hoa chữ thường trong HQL.

## Mệnh đề FROM

Bạn sẽ sử dụng mệnh đề **FROM** nếu bạn muốn tải các các đối tượng persistent vào bộ nhớ. Sau đây là cú pháp đơn giản sử dụng mệnh đề FROM:

|  |
| --- |
| **String hql = "FROM Employee";** |
| **Query query = session.createQuery(hql);** |

|  |
| --- |
| **List results = query.list();** |

Nếu bạn cần sử dụng tên lớp đầy đủ trong HQL, chỉ cần chỉ định gói và tên lớp như sau:

|  |
| --- |
| **String hql = "FROM com.hibernatebook.criteria.Employee";** |

## Mệnh đề AS

Mệnh đề **AS**có thể được sử dụng để gán các bí danh cho các lớp trong các truy vấn HQL, đặc biệt khi bạn có các truy vấn dài. Ví dụ:

|  |
| --- |
| **String hql = "FROM Employee AS E";** |
| **Query query = session.createQuery(hql);** |

|  |
| --- |
| **List results = query.list();** |

Từ khoá **AS** là tùy chọn và bạn cũng có thể chỉ định bí danh trực tiếp sau tên lớp:

|  |
| --- |
| **String hql = "FROM Employee E";** |

## Mệnh đề SELECT

Mệnh đề SELECT cung cấp kiểm soát nhiều hơn các tập kết quả hơn mệnh đề from. Nếu bạn muốn lấy ra những thuộc tính cần thiết của các đối tượng thay vì đối tượng hoàn chỉnh, hãy sử dụng mệnh đề SELECT. Sau đây là cú pháp đơn giản của việc sử dụng mệnh đề SELECT để chỉ lấy ra trường first\_name của đối tượng Employee:

|  |
| --- |
| **String hql = "SELECT E.firstName FROM Employee E";** |
| **Query query = session.createQuery(hql);** |

|  |
| --- |
| **List results = query.list();** |

**Chú ý: Employee.firstName là thuộc tính của đối tượng Employee chứ không phải là một trường của bảng EMPLOYEE…**

## Mệnh đề WHERE

Nếu bạn muốn thu hẹp các đối tượng cụ thể được trả về từ cơ sở dữ liệu, bạn sử dụng mệnh đề WHERE. Sau đây là cú pháp đơn giản sử dụng mệnh đề WHERE:

|  |
| --- |
| **String hql = "FROM Employee E WHERE E.id = 10";** |
| **Query query = session.createQuery(hql);** |

|  |
| --- |
| **List results = query.list();** |

## Mệnh đề ORDER BY

Để sắp xếp kết quả truy vấn HQL của bạn, bạn sẽ cần sử dụng mệnh đề **ORDER BY**. Bạn có thể sắp xếp các kết quả theo bất kỳ thuộc tính nào trên các đối tượng trong tập kết quả hoặc tăng dần (ASC) hoặc giảm dần (DESC). Sau đây là cú pháp đơn giản của việc sử dụng mệnh đề ORDER BY:

|  |
| --- |
| **String hql = "FROM Employee E WHERE E.id > 10 ORDER BY E.salary DESC";** |
| **Query query = session.createQuery(hql);** |

|  |
| --- |
| **List results = query.list();** |

Nếu bạn muốn sắp xếp theo nhiều thuộc tính, bạn chỉ cần thêm các thuộc tính bổ sung vào cuối của mệnh đề ORDER BY, được phân cách bằng dấu phẩy như sau:

|  |
| --- |
| **String hql = "FROM Employee E WHERE E.id > 10 " +** |
| **"ORDER BY E.firstName DESC, E.salary DESC ";** |

|  |
| --- |
| **Query query = session.createQuery(hql);** |
| **List results = query.list();** |

## Mệnh đề GROUP BY

Mệnh đề này cho phép Hibernate lấy thông tin từ cơ sở dữ liệu và phân nhóm chúng dựa trên giá trị của một thuộc tính. Sau đây là cú pháp đơn giản của việc sử dụng mệnh đề GROUP BY:

|  |
| --- |
| **String hql ="SELECT SUM(E.salary),E.firtName FROM Employee E " +** |
| **"GROUP BY E.firstName";** |

|  |
| --- |
| **Query query = session.createQuery(hql);** |
| **List results = query.list();** |

## Sử dụng các tham số đặt tên:

Hibernate hỗ trợ các tham số được đặt tên trong các truy vấn HQL của nó. Điều này làm cho việc viết các truy vấn HQL chấp nhận đầu vào từ người dùng dễ dàng và bạn không phải phòng chống các cuộc tấn công SQL injection. Sau đây là cú pháp đơn giản sử dụng các tham số được đặt tên:

|  |
| --- |
| **String hql = "FROM Employee E WHERE E.id = :employee\_id";** |
| **Query query = session.createQuery(hql);** |

|  |
| --- |
| **query.setParameter("employee\_id",10);** |
| **List results = query.list();** |

## Mệnh đề UPDATE

Cập nhật hàng loạt là kiểu mới đối với HQL trong Hibernate 3. Và xóa khác nhau trong Hibernate 3 so với Hibernate 2. Giao tiếp Query giờ đây chứa một phương thức gọi là executeUpdate() để thực hiện các câu lệnh HQL UPDATE hoặc DELETE.

Mệnh đề **UPDATE** có thể được sử dụng để cập nhật một hoặc nhiều thuộc tính của một hoặc nhiều đối tượng. Sau đây là cú pháp đơn giản sử dụng mệnh đề UPDATE:

|  |
| --- |
| **String hql = "UPDATE Employee set salary = :salary "  +** |
| **"WHERE id = :employee\_id";** |

|  |
| --- |
| **Query query = session.createQuery(hql);** |
| **query.setParameter("salary", 1000);** |

|  |
| --- |
| **query.setParameter("employee\_id", 10);** |
| **int result = query.executeUpdate();** |

## Mệnh đề DELETE

Mệnh đề **DELETE** được sử dụng để xóa một hoặc nhiều đối tượng. Sau đây là cú pháp đơn giản sử dụng mệnh đề DELETE:

|  |
| --- |
| **String hql = "DELETE FROM Employee "  + "WHERE id = :employee\_id";** |

|  |
| --- |
| **Query query = session.createQuery(hql);** |
| **query.setParameter("employee\_id", 10);** |

|  |
| --- |
| **int result = query.executeUpdate();** |

## Mệnh đề INSERT

HQL chỉ hỗ trợ mệnh đề INSERT INTO nơi mà các bản ghi có thể được insert từ một đối tượng này sang một đối tượng khác. Cú pháp đơn giản sử dụng mệnh đề INSERT INTO:

|  |
| --- |
| **String hql = "INSERT INTO Employee(firstName,lastName,salary)" +** |
| **"SELECT firstName, lastName, salary FROM old\_employee";** |

|  |
| --- |
| **Query query = session.createQuery(hql);** |
| **int result = query.executeUpdate();** |

## Các phương thức tổng hợp

HQL hỗ trợ một loạt các phương thức tổng hợp tương tự như SQL. Chúng hoạt động theo cùng một cách trong HQL như trong SQL và sau đây là danh sách các chức năng có sẵn:

|  |  |  |
| --- | --- | --- |
| **No.** | **Các hàm** | **Mô tả** |
| 1 | avg(property name) | Trung bình của các giá trị của một thuộc tính. |
| 2 | count(property name or \*) | Số lần một thuộc tính xảy ra trong kết quả. |
| 3 | max(property name) | Giá tri lớn nhất trong các giá trị của một trường. |
| 4 | min(property name) | The minimum value of the property values |
| 5 | sum(property name) | Tính tổng các giá trị của một trường |

Từ khóa **distinct** chỉ tính các giá trị duy nhất trong tập hàng. Truy vấn sau đây sẽ chỉ trả về số lần duy nhất:

|  |
| --- |
| **String hql = "SELECT count(distinct E.firstName) FROM Employee E";** |
| **Query query = session.createQuery(hql);** |

|  |
| --- |
| **List results = query.list();** |

## Phân trang bằng việc sử dụng Query

Có hai phương thức của giao tiếp Query cho việc phân trang.

|  |  |
| --- | --- |
| **No.** | **Phương thức & Mô tả** |
| 1 | **Query setFirstResult(int startPosition)**  Phương thức này lấy một số nguyên đại diện cho hàng đầu tiên trong tập kết quả, bắt đầu với hàng 0. |
| 2 | **Query setMaxResults(int maxResult)**  Phương thức này cho Hibernate tìm kiếm một số **maxResults**của các đối tượng. |

Sử dụng hai phương thức trên, chúng ta có thể xây dựng một thành phần phân trang trong ứng dụng Web hoặc Swing. Ví dụ có thể mở rộng để lấy 10 hàng cùng một lúc:

**String hql = "FROM Employee";**

**Query query = session.createQuery(hql);**

**query.setFirstResult(1);**

**query.setMaxResults(10);**

**List results = query.list();**

# **13. Criteria Query**

Trong các ứng dụng kết nối cơ sở dữ liệu, việc xây dựng câu truy vấn hay query, là điều cực kỳ quan trọng vì chúng sẽ giúp tăng tốc độ tải dữ liệu của ứng dụng. Framework Hibernate cho phép chúng ta viết câu truy vấn bằng "code" thay vì phải gõ các mệnh đề như Select, From,.. bằng API mang tên Criteria. Điều này sẽ giúp lập trình viên tiết kiệm thời gian khi thao tác với cơ sở dữ liệu và tránh được các lỗi cú pháp.

## a. Câu truy vấn cơ bản

Thông thường, để viết một câu truy vấn "Lấy tất cả đối tượng trong lớp Employee ", chúng ta có thể làm như sau:

1. String hql = "FROM Employee";
2. Query query = session.createQuery(hql);
3. List results = query.list();

Đoạn code trên sẽ lấy tất cả đối tượng (dòng) trong lớp Employee và trả về cho results. Với Criteria, chúng ta có thể "code" câu truy vấn mà không cần đến FROM như sau.

1. Criteria ctr = session.createCriteria(Employee.class);
2. List results = ctr.list();

Đoạn code với Criteria sẽ cho cùng kết quả là tất cả đối tượng có trong lớp Employee, nhưng tiết kiệm thời gian và lập trình viên sẽ không cần ghi đầy đủ câu truy vấn.

## Restrictions – Điều kiện

Để lấy các đối tượng trong lớp Employee có số views lớn hơn 10.000, ta có thể làm như sau.

1. String hql = "FROM Employee WHERE Employee.views > 10000";
2. Query query = session.createQuery(hql);
3. List results = query.list();

Nếu sử dụng Criteria, điều kiện WHERE có thể được viết lại dễ hiểu hơn cho lập trình viên.

1. Criteria ctr = session.createCriteria(Employee.class);
2. ctr.add(Restrictions.gt("views", 10000));
3. List results = ctr.list();

Hàm **add(Criterion ctn)** sẽ thêm một điều kiện vào query của chúng ta, với **Restrictions** là lớp con chứa các phép so sánh cần thiết. Ở đoạn code trên, query sử dụng **Restrictions.gt** là **Greater Than**, sẽ lấy các đối tượng thuộc lớp Employee có thuộc tính được khai báo là views có giá trị lớn hơn 10000.

Với cách làm này, câu truy vấn tuy không thể hiện rõ như SQL hay cách viết SELECT - FROM, nhưng sẽ giúp lập trình viên dễ kiểm soát và bảo trì khi có sự cố xảy ra.

Dưới đây là bảng các Restrictions:

| **Restriction** | **Description** | **Example** |
| --- | --- | --- |
| Restrictions.gt | Greater Than - giá trị so sánh phải lớn hơn số X | ctr.add(Restrictions.gt("views", 10000)); |
| Restrictions.lt | Less Than - giá trị so sánh phải nhỏ hơn số X | ctr.add(Restrictions.lt("views", 10000)); |
| Restrictions.like | Tìm đối tượng có giá trị tương đương, giống nhau, không phân biệt hoa thường | ctr.add(Restrictions.like("title", "HQL", MatchMode.ANYWHERE)); |
| Restrictions.ilike | Tương tự like, nhưng có phân biệt hoa thường | ctr.add(Restrictions.ilike("title", "HqL", MatchMode.ANYWHERE)); |
| Restrictions.between | Giá trị phải trong khoảng X và Y | ctr.add(Restrictions.between("views", 5000, 10000)); |
| Restrictions.isNull | Kiểm tra thuộc tính NULL hay không | ctr.add(Restrictions.isNull("date\_created")); |
| Restrictions.isNotNull | Kiểm tra thuộc tính có khác NULL hay không | ctr.add(Restrictions.isNotNull("date\_created")); |
| Restrictions.isEmpty | Kiểm tra thuộc tính có rỗng hay không | ctr.add(Restrictions.isEmpty("title")); |
| Restrictions.isNotEmpty | Kiểm tra thuộc tính có khác rỗng hay không | ctr.add(Restrictions.isNotEmpty("title")); |
| Restrictions.and | Kết hợp AND giữa các điều kiện | Sẽ giải thích thêm ở mục tiếp theo |
| Restrictions.or | Kết hợp OR giữa các điều kiện | Sẽ giải thích thêm ở mục tiếp theo |

## AND/OR

Để sử dụng And/Or trong câu truy vấn, chúng ta sử dụng **LogicalExpression**.

1. Criteria ctr = session.createCriteria(Employee.class);
2. Criterion views = Restriction.gt("views", 10000);
3. Criterion comments = Restriction.gt("comments", 10);
4. LogicalExpression andExp = Restrictions.and(views, comments);
5. ctr.add(andExp);
6. LogicalExpression orExp = Restrictions.or(views, comments);
7. ctr.add(orExp);
8. List results = ctr.list();

Khác với các ví dụ trước, khi sử dụng**LogicalExpression**, chúng ta không **add** trực tiếp điều kiện về views và comments mà sẽ kết hợp chúng lại bằng **Restrictions.and** hoặc **Restrictions.or**, sau đó mới **add** Restriction này vào ctr.

## Paging – Phân trang

Chúng ta sẽ sử dụng hàm **setFirstResult** và **setMaxResults** được cung cấp bởi lớp **Criteria**.

1. Criteria ctr = session.createCriteria(Articles.class);
2. ctr.setFirstResult(11);
3. ctr.setMaxResults(20);
4. List results = ctr.list();

Câu truy vấn sẽ lấy đối tượng từ vị trí thứ 11 đến 20 (trang 2, mỗi trang 10 kết quả).

## Sorting – Sắp xếp

Để sắp xếp các kết quả của truy vấn, chúng ta sử dụng **addOrder()**, với tham số là **Order.desc()** hoặc **Order.asc()**.

1. Criteria ctr = session.createCriteria(Employee.class);
2. ctr.add(Restrictions.gt("views", 10000));
3. ctr.addOrder(Order.desc("views"));
4. List results = ctr.list();

Để thêm nhiều Order, chúng ta chỉ việc ghi như sau.

1. ctr.addOrder(Order.desc("views"));
2. ctr.addOrder(Order.desc("comments"));

## Hàm tính toán

Tương tự với **Restrictions**, chúng ta có thể thêm các phép tính với giá trị vào câu truy vấn sử dụng **setProjection**.

Criteria ctr = session.createCriteria(Employee.class);

ctr.setProjection(Projections.rowCount());

List results = ctr.list();

if (!results.isEmpty()) {

Integer rowCount = (Integer) results.get(0);

System.out.println("Number of rows: " + rowCount); }

**Projections.rowCount()** sẽ đếm số đối tượng có trong lớp Articles và trả về một object Integer trong list.

Dưới đây là bảng các Projection khác:

| **Projection** | **Description** | **Example** |
| --- | --- | --- |
| Projections.rowCount | Đếm tổng số đối tượng (dòng) trong lớp | ctr.setProjection(Projections.rowCount()); |
| Projections.avg | Tính trung bình giá trị của thuộc tính (cột) | ctr.setProjection(Projections.avg("views")); |
| Projections.countDistinct | Đếm số đối tượng (không trùng lắp) trong lớp | ctr.setProjection(Projections.countDistinct("author")); |
| Projections.max | Lấy giá trị lớn nhất của thuộc tính | ctr.setProjection(Projections.max("views")); |
| Projections.min | Lấy giá trị nhỏ nhất của thuộc tính | ctr.setProjection(Projections.min("views")); |
| Projections.sum | Tính tổng các giá trị trong thuộc tính | ctr.setProjection(Projections.sum("views")); |

# **CÁC KIỂU TRUY VẤN NATIVE SQL**

Bạn có thể sử dụng Native SQL trong Hibernate để thực hiện các truy vấn cơ sở dữ liệu nếu bạn muốn sử dụng các tính năng cơ sở dữ liệu cụ thể như hint hoặc từ khoá CONNECT trong Oracle. Hibernate 3.x cho phép bạn chỉ định SQL viết tay, bao gồm các stored procedures, và tất cả các thao tác create, update, delete và select.

Sau khi bạn truyền một chuỗi chứa truy vấn SQL đến phương thức createSQLQuery(), bạn phải liên kết kết quả trả về SQL cho thực thể Hibernate hiện có, một join, hoặc một kết quả vô hướng (scalar) sử dụng các phương thức addEntity(), addJoin() và addScalar() tương ứng.

## Truy vấn vô hướng (Scalar)

Các truy vấn SQL cơ bản nhất là để có được một danh sách các vô hướng (giá trị) từ một hoặc nhiều bảng. Sau đây là cú pháp để sử dụng SQL gốc cho các giá trị vô hướng. Ví dụ:

|  |
| --- |
| **String sql = "SELECT first\_name, salary FROM EMPLOYEE";** |
| **SQLQuery query = session.createSQLQuery(sql);** |

|  |
| --- |
| **query.setResultTransformer(Criteria.ALIAS\_TO\_ENTITY\_MAP);** |
| **List results = query.list();** |

## Truy vấn Entity

Các truy vấn ở trên là về trả về các giá trị vô hướng từ resultset. Sau đây là cú pháp để có được các đối tượng Entity từ một truy vấn Native SQL thông qua addEntity().

|  |
| --- |
| **String sql = "SELECT \* FROM EMPLOYEE";** |
| **SQLQuery query = session.createSQLQuery(sql);** |

|  |
| --- |
| **query.addEntity(Employee.class);** |

1. **Truy vấn Entity với Named SQL**

Sau đây là cú pháp để nhận các đối tượng thực thể từ truy vấn native sql thông qua addEntity() và sử dụng truy vấn Named SQL.

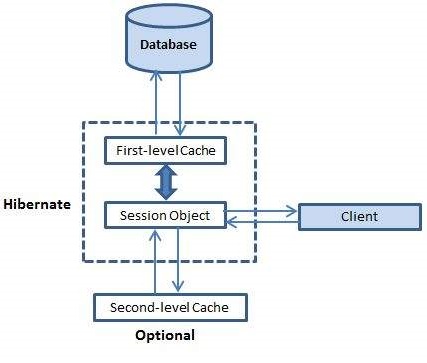
|  |
| --- |
| **String sql = "SELECT \* FROM EMPLOYEE WHERE id = :employee\_id";** |
| **SQLQuery query = session.createSQLQuery(sql);** |

|  |
| --- |
| **query.addEntity(Employee.class);** |
| **query.setParameter("employee\_id", 10);** |

# **15. BỘ NHỚ CACHE**

## a. Bộ nhớ cache trong Hibernate

Dùng trong việc tối ưu hóa hiệu của suất ứng dụng. Nó nằm giữa ứng dụng và cơ sở dữ liệu để tránh số lượng lượt truy cập cơ sở dữ liệu càng nhiều càng tốt để cung cấp cho một hiệu suất tốt hơn cho các ứng dụng.



*Sơ đồ bộ nhớ đệm*

## Các kiểu bộ nhớ cache trong Hibernate

Có 3 kiểu bộ nhớ cache trong Hibernate đó là: Bộ nhớ cache cấp một (First-level), Bộ nhớ cache cấp hai (Second-level) và Bộ nhớ cache cấp truy vấn.

**+ Bộ nhớ cache cấp một (First-level)**

Bộ nhớ cache cấp một là bộ nhớ cache Session và là một bộ nhớ cache bắt buộc thông qua đó tất cả các yêu cầu phải vượt qua. Đối tượng Session giữ một đối tượng thuộc quyền sở hữu của nó trước khi commit nó vào cơ sở dữ liệu.

**+ Bộ nhớ cache cấp hai (Second-level)**

Bộ nhớ cache cấp hai là một bộ nhớ cache tùy chọn. Và bộ nhớ cache cấp một sẽ luôn được thảo luận trước khi thực hiện xác định vị trí một đối tượng trong bộ nhớ cache cấp hai. Bộ nhớ cache cấp hai có thể được cấu hình trên cơ sở mỗi lớp và mỗi collection và chịu trách nhiệm chính trong việc lưu trữ các đối tượng trong các session.

**+ Bộ nhớ cache cấp truy vấn**

Hibernate cũng thực hiện bộ nhớ cache cho các kết quả truy vấn kết hợp chặt chẽ với bộ nhớ cache cấp hai. Đây là tính năng tùy chọn và yêu cầu thêm hai vùng bộ nhớ cache vật lý giữ kết quả truy vấn và các dấu thời gian khi một bảng được cập nhật lần cuối. Điều này chỉ hữu ích cho các truy vấn được chạy thường xuyên với các tham số giống nhau.

## Sử dụng bộ nhớ cache trong Hibernate

Hibernate sử dụng bộ nhớ cache cấp một theo mặc định và bạn không phải làm gì để sử dụng bộ nhớ cache cấp một. Hãy đi thẳng đến bộ nhớ cache cấp hai tùy chọn.

Bộ nhớ cache cấp hai Hibernate được thiết lập trong hai bước. Trước hết, bạn phải quyết định chiến lược truy cập đồng thời nào để sử dụng. Sau đó, bạn cấu hình hết hạn của bộ nhớ cache và thuộc tính bộ nhớ cache vật lý bằng cách sử dụng bộ nhớ cache provider.

***+* Chiến lược truy cập đồng thời**

Chiến lược truy cập đồng thời là bộ điều chỉnh có trách nhiệm lưu trữ các mục dữ liệu trong bộ nhớ cache và lấy chúng từ bộ nhớ cache. Nếu muốn kích hoạt bộ nhớ cache cấp hai, bạn sẽ phải quyết định, đối với mỗi lớp và collection persistent, mà chiến lược truy cập đồng thời vào bộ nhớ cache để sử dụng.

* **Transactional**: Sử dụng chiến lược này để đọc – chủ yếu là dữ liệu, nơi mà nó là quan trọng để ngăn ngừa dữ liệu cũ trong các transaction đồng thời, trong trường hợp hiếm hoi của một cập nhật.
* **Read-write**: Một lần nữa sử dụng chiến lược này để đọc – chủ yếu là dữ liệu, nơi nó là quan trọng để ngăn ngừa dữ liệu cũ trong các transaction đồng thời, trong trường hợp hiếm hoi của một cập nhật.
* **Nonstrict-read-write**: Chiến lược này không đảm bảo tính nhất quán giữa bộ nhớ cache và cơ sở dữ liệu. Sử dụng chiến lược này nếu dữ liệu hầu như không thay đổi và một khả năng nhỏ dữ liệu cũ không phải là mối quan tâm quan trọng.
* **Read-only**: Một chiến lược truy cập đồng thời phù hợp với dữ liệu mà không bao giờ thay đổi. Chỉ sử dụng nó cho dữ liệu tham khảo.

**+ Cache provider**

Bước tiếp theo của bạn sau khi lựa chọn chiến lược đồng thời bạn sẽ sử dụng cho các lớp ứng cử viên bộ nhớ cache là việc chọn lựa một cache provider. Hibernate buộc bạn phải chọn một cache provider duy nhất cho toàn bộ ứng dụng.

|  |  |  |
| --- | --- | --- |
| **STT** | **Cache Provider** | **Mô tả** |
| 1 | EHCache | Nó có thể cache trong bộ nhớ RAM hoặc trên đĩa cứng và clustered caching và nó hỗ trợ bộ nhớ cache kết quả truy vấn Hibernate tuỳ chọn. |
| 2 | OSCache | Hỗ trợ bộ nhớ đệm vào bộ nhớ RAM và đĩa cứng trong một JVM duy nhất, với một tập hợp đầy đủ các chính sách hết hạn và hỗ trợ bộ nhớ truy vấn. |
| 3 | warmCache | Một bộ nhớ cache cluster dựa trên JGroups. Nó sử dụng huỷ bỏ hiệu lực clustered nhưng không hỗ trợ bộ nhớ cache truy vấn Hibernate |
| 4 | JBoss Cache | Một bộ nhớ cache cluster được sao chép hoàn toàn hợp lệ được transaction dựa trên thư viện đa nhóm JGroups. Nó hỗ trợ nhân bản hoặc hủy bỏ hiệu lực, giao tiếp đồng bộ hoặc không đồng bộ, và khóa lạc quan và bi quan. Bộ nhớ truy vấn cache Hibernate được hỗ trợ |

Mỗi cache provider không tương thích với mọi chiến lược truy cập đồng thời. Ma trận tương thích sau sẽ giúp bạn chọn một sự kết hợp thích hợp.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Strategy/Provider** | **Read-only** | **Nonstrictread-write** | **Read-write** | **Transactional** |
| EHCache | X | X | X |  |
| OSCache | X | X | X |  |
| SwarmCache | X | X |  |  |
| JBoss Cache | X |  |  | X |

**+ Bộ nhớ cache truy vấn**

Để sử dụng bộ nhớ cache truy vấn, trước tiên bạn phải kích hoạt nó bằng cách sử dụng thuộc tính **hibernate.cache.use\_query\_cache = “true”** trong tệp tin cấu hình. Bằng cách thiết lập thuộc tính này thành **true**, bạn cho phép Hibernate tạo ra các bộ đệm cần thiết trong bộ nhớ để giữ các tập truy vấn và bộ nhận dạng (identifier). Ví dụ:

|  |
| --- |
| **Session session = SessionFactory.openSession();** |
| **Query query = session.createQuery("FROM EMPLOYEE");** |

|  |
| --- |
| **query.setCacheable(true);** |
| **List users = query.list();** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SessionFactory.closeSession();**  Hibernate cũng hỗ trợ hỗ trợ bộ nhớ cache rất tinh vi thông qua khái niệm vùng nhớ cache. Một vùng nhớ cache là một phần của bộ nhớ cache được đặt tên.   |  | | --- | | **Session session = SessionFactory.openSession();** | | **Query query = session.createQuery("FROM EMPLOYEE");** |  |  | | --- | | **query.setCacheable(true);** | | **query.setCacheRegion("employee");** |  |  | | --- | | **List users = query.list();** | | **SessionFactory.closeSession();** | |

# **16. Batch Processing**

## a. Batch processing là gì?

Ta có tình huống như sau: cần insert 100000 bản ghi vào database. Nếu insert lần lượt và đẩy từng đối tượng một thì thời gian sẽ rất lâu vì phải mở/đóng connection nhiều lần.

**Session session = SessionFactory.openSession();**

**Transaction tx = session.beginTransaction();**

**for ( int i=0; i<100000; i++ ) {**

**Employee employee = new Employee(.....);**

**session.save(employee);**

**}**

**tx.commit();**

**session.close();**

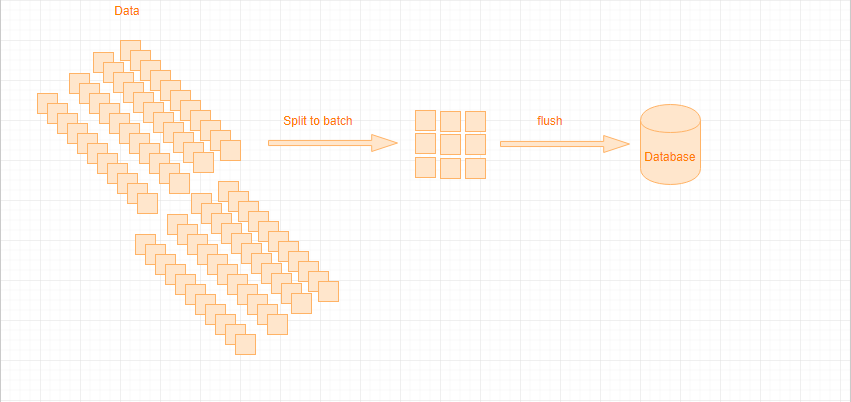
Mặc định, Hibernate sẽ lưu trữ tất cả các đối tượng đã tồn tại trong bộ nhớ cache cấp session và cuối cùng ứng dụng của bạn sẽ rơi vào tình trạng **OutOfMemoryException** khi ở hàng 50.000. Bạn có thể sử dụng **batch processing** trong Hibernate để giải quyết vấn đề này. (Lý do là hibernate sẽ lưu tất cả 50000 bản ghi vào bộ nhớ cache nhưng bộ nhớ cache không đủ)

=> **Giải pháp**: Batch Processing

Batch Processing là xử lý theo lô, tức là ta sẽ insert và đẩy từng lô bản ghi vào database.

Ví dụ ở đây ta sẽ insert và đẩy 50 bản ghi vào database cùng 1 lúc, như thế sẽ giảm số lượng connection tới database, đồng thời tránh được lỗi OutOfMemoryException.

Để sử dụng tính năng batch processing trong Hibernate, đầu tiên hãy đặt **hibernate.jdbc.batch\_size** có kích thước batch là X = 10 đến 50 hoặc một số khác nào đó tùy thuộc vào kích thước đối tượng. Điều này sẽ nói với các Hibernate rằng mỗi lô có X bản ghi được insert theo batch.



Để thực hiện điều này trong đoạn code trên, chúng ta sẽ cần phải sửa đổi như sau:

**Session session = sessionFactory.openSession();**

**Transaction tx = session.beginTransaction();**

**for ( int i=0; i<100000; i++ ) {**

**Customer customer = new Customer(.....);**

**Session.save(customer);**

**if ( i % 50 == 0 ) { //50, same as the JDBC batch size**

**//flush a batch of inserts and release memory:**

**session.flush();**

**session.clear(); } }**

**tx.commit();**

**session.close();**

Đoạn code trên sẽ làm việc tốt cho các hoạt động INSERT, nhưng khi bạn thực hiện hoạt động UPDATE thì bạn có sử dụng code sau đây:

**Session session = sessionFactory.openSession();**

**Transaction tx = session.beginTransaction();**

**ScrollableResults employeeCursor = session.createQuery("FROM EMPLOYEE").scroll();**

**int count = 0;**

**while ( employeeCursor.next() ) {**

**Employee employee = (Employee) employeeCursor.get(0);**

**employee.updateEmployee();**

**seession.update(employee);**

**if ( ++count % 50 == 0 ) {**

**session.flush();**

**session.clear();**

**}**

**}**

**tx.commit();**

**session.close();**

Sửa file cấu hình để thêm tính năng **hibernate.jdbc.batch\_size nếu muốn dùng batch:**

|  |
| --- |
| **<!-- thiết định kích thước batch -->** |

|  |
| --- |
| **<property name="hibernate.jdbc.batch\_size">** |

|  |
| --- |
| **50** |

|  |
| --- |
| **</property>** |

# **17. INTERCEPTORS**

## Khái niệm

Trong Hibernate, một đối tượng sẽ được tạo ra và tồn tại. Một khi đối tượng đã được thay đổi, nó phải được lưu trở lại cơ sở dữ liệu. Quá trình này tiếp tục cho đến khi đối tượng tiếp theo được yêu cầu, và nó sẽ được tải từ các persistent store. Do đó, một đối tượng đi qua các giai đoạn khác nhau trong vòng đời của nó. Giao tiếp Interceptor trong Hibernate cung cấp các phương thức có thể được gọi ở các giai đoạn khác nhau để thực hiện một số nhiệm vụ được yêu cầu. Các phương thức này là gọi lại từ session tới ứng dụng, cho phép ứng dụng kiểm tra và/hoặc thao tác các thuộc tính của một đối tượng persistent trước khi nó được lưu, cập nhật, xóa hoặc nạp.

## b. Ưu điểm và nhược điểm

* **Ưu điểm:**
  + Dễ sử dụng: dễ dàng quản lí các kết nối database và dễ fix bug, cung cấp sẵn nhiều API truy vấn
  + Tính độc lập: không cần quan tâm tới cơ sở dữ liệu sử dụng khi viết câu lệnh SQL.
  + Tính hướng đối tượng: tập trung xử lý theo hướng đối tượng, phù hợp sử dụng trong các case CRUD(Create, Read, Update, Delete)
  + Tính tin cậy: Hibernate đã được kiểm thử và khá an toàn trong các truy vấn
* **Nhược điểm:**
* Không hỗ trợ các câu truy vấn phức tạp
* Một số trường hợp vẫn phải dùng native SQL do Hibernate không thể cover hết tất cả các cú pháp của các hệ quản trị cơ sử dữ liệu.
* Bị hạn chế sự can thiệp vào câu lệnh SQL do nó được tự động sinh ra.

## c. Các phương thức có sẵn trong Interceptor Interface

|  |  |
| --- | --- |
| **No.** | **Phương thức và mô tả** |
| 1 | **findDirty()**  Phương thức này được gọi khi phương thức **flush() được gọi trên một đối tượng Session.** |
| 2 | **instantiate()**  Phương thức này được gọi khi một lớp persistent được khởi tạo. |
| 3 | **isUnsaved()**  Phương thức này được gọi khi một đối tượng được truyền vào phương thức **saveOrUpdate()**/ |
| 4 | **onDelete()**  Phương thức này được gọi trước khi một đối tượng bị xóa. |
| 5 | **onFlushDirty()**  Phương thức này được gọi khi Hibernate phát hiện ra rằng một đối tượng là dirty(tức là đã được thay đổi) trong quá trình flush nghĩa là thao tác update. |
| 6 | **onLoad()**  Phương thức này được gọi trước khi một đối tượng được khởi tạo. |
| 7 | **onSave()**  Phương thức này được gọi trước khi một đối tượng được lưu. |
| 8 | **postFlush()**  Phương thức này được gọi sau khi flush và một đối tượng đã được update trong bộ nhớ. |
| 9 | **preFlush()**  Phương thức này được gọi trước một flush. |

## Ví dụ

Để xây dựng một interceptor bạn có thể impliments lớp Interceptor trực tiếp hoặc extents lớp EmptyInterceptor. Sau đây là các bước đơn giản để sử dụng chức năng Interceptor trong Hibernate:

**Tạo các interceptor:** Tạo lớp **MyInterceptor** kế thừa **EmptyInterceptor** nơi phương thức của Interceptor sẽ được gọi tự động khi đối tượng Employee được tạo ra và cập nhật. Bạn có thể thực hiện nhiều phương thức hơn theo yêu cầu của người dùng.

**import java.io.Serializable;**

**import java.util.Iterator;**

**import org.hibernate.EmptyInterceptor;**

**import org.hibernate.type.Type;**

**import hibernate.interceptor.entity.Employee;**

**public class MyInterceptor extends EmptyInterceptor {**

**public void onDelete(Object entity, Serializable id, Object[] state, String[] propertyNames, Type[] types) {**

**// do nothing**

**}**

**// This method is called when Employee object gets updated.**

**public boolean onFlushDirty(Object entity, Serializable id, Object[] currentState, Object[] previousState,String[] propertyNames, Type[] types) {**

**if (entity instanceof Employee) {**

**System.out.println("Update Operation");**

**return true;**

**}**

**return false;**

**}**

**public boolean onLoad(Object entity, Serializable id, Object[] state, String[] propertyNames, Type[] types) {**

**// do nothing**

**return true;**

**}**

**// This method is called when Employee object gets created.**

**public boolean onSave(Object entity, Serializable id, Object[] state, String[] propertyNames, Type[] types) {**

**if (entity instanceof Employee) {**

**System.out.println("Create Operation");**

**return true;**

**}**

**return false;**

**}**

**// called before commit into database**

**public void preFlush(Iterator iterator) {**

**System.out.println("preFlush");**

**}**

**// called after committed into database**

**public void postFlush(Iterator iterator) {**

**System.out.println("postFlush");**

**}**

**}**

**Tạo lớp POJO**

**package hibernate.interceptor.entity;**

**public class Employee {**

**private int id;**

**private String firstName;**

**private String lastName;**

**private int salary;**

**public Employee() {}**

**public Employee(String fname, String lname, int salary) {**

**this.firstName = fname;**

**this.lastName = lname;**

**this.salary = salary;**

**}**

**public int getId() {**

**return id;**

**}**

**public void setId(int id) {**

**this.id = id;**

**}**

**public String getFirstName() {**

**return firstName;**

**}**

**public void setFirstName(String first\_name) {**

**this.firstName = first\_name;**

**}**

**public String getLastName() {**

**return lastName;**

**}**

**public void setLastName(String last\_name) {**

**this.lastName = last\_name;**

**}**

**public int getSalary() {**

**return salary;**

**}**

**public void setSalary(int salary) {**

**this.salary = salary;**

**}**

**}**

**Tạo bảng cơ sở dữ liệu:** Sẽ có một bảng tương ứng với mỗi đối tượng mà bạn sẵn sàng cung cấp persistent. Các đối tượng trên cần phải được lưu trữ và truy xuất vào bảng RDBMS sau:

|  |
| --- |
| create table **EMPLOYEE** ( |
| id **INT NOT NULL** auto\_increment, |

|  |
| --- |
| first\_name **VARCHAR**(20) default **NULL**, |
| last\_name  **VARCHAR**(20) default **NULL**, |

|  |
| --- |
| salary     **INT**  default NULL, |
| **PRIMARY KEY** (id) |

|  |
| --- |
| ); |

**Tạo file mapping**

**<?xml version="1.0" encoding="utf-8"?>**

**<!DOCTYPE hibernate-mapping PUBLIC**

**"-//Hibernate/Hibernate Mapping DTD//EN"**

**"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">**

**<hibernate-mapping>**

**<class name="hibernate.interceptor.entity.Employee" table="EMPLOYEE">**

**<meta attribute="class-description">**

**Lớp này chứa chi tiết về employee**

**</meta>**

**<id name="id" type="int" column="id">**

**<generator class="native"/>**

**</id>**

**<property name="firstName" column="first\_name" type="string"/>**

**<property name="lastName" column="last\_name" type="string"/>**

**<property name="salary" column="salary" type="int"/>**

**</class>**

**</hibernate-mapping>**

**Tạo file cấu hình Hibernate**

|  |
| --- |
| **<?xml version="1.0" encoding="UTF-8"?>**  **<!DOCTYPE hibernate-configuration SYSTEM**  **"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">**  **<hibernate-configuration>**  **<session-factory>**  **<property name="hibernate.dialect">**  **org.hibernate.dialect.MySQLDialect**  **</property>**  **<property name="hibernate.connection.driver\_class">**  **com.mysql.jdbc.Driver**  **</property>**  **<!-- Thông tin kết nối db -->**  **<property name="hibernate.connection.url">**  **jdbc:mysql://localhost/testdb**  **</property>**  **<property name="hibernate.connection.username">**  **root**  **</property>**  **<property name="hibernate.connection.password">**  **1234567890**  **</property>**  **<!-- List các file XML mapping -->**  **<mapping resource="Employee.hbm.xml" />**  **</session-factory>**  **</hibernate-configuration>** |

**Tạo lớp ứng dụng:** Ở đây cần lưu ý rằng trong khi tạo đối tượng Session, chúng ta đã sử dụng lớp Interceptor như một đối số.

**import java.util.Iterator;**

**import java.util.List;**

**import org.hibernate.HibernateException;**

**import org.hibernate.Session;**

**import org.hibernate.SessionFactory;**

**import org.hibernate.Transaction;**

**import org.hibernate.cfg.Configuration;**

**import hibernate.interceptor.entity.Employee;**

**public class ManageEmployee {**

**private static SessionFactory factory;**

**public static void main(String[] args) {**

**try {**

**factory = new Configuration().configure().buildSessionFactory();**

**} catch (Throwable ex) {**

**System.err.println("Failed to create sessionFactory object." + ex);**

**throw new ExceptionInInitializerError(ex); }**

**ManageEmployee ME = new ManageEmployee();**

**// Add few employee records in database**

**Integer empID1 = ME.addEmployee("Zara", "Ali", 1000);**

**Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);**

**Integer empID3 = ME.addEmployee("John", "Paul", 10000);**

**// List down all the employees**

**ME.listEmployees();**

**// Update employee's records**

**ME.updateEmployee(empID1, 5000);**

**// Delete an employee from the database**

**ME.deleteEmployee(empID2);**

**// List down new list of the employees**

**ME.listEmployees();**

**}**

**// Method to CREATE an employee in the database**

**public Integer addEmployee(String fname, String lname, int salary){**

**Session session = factory.openSession(new MyInterceptor());**

**Transaction tx = null;**

**Integer employeeID = null;**

**try {**

**tx = session.beginTransaction();**

**Employee employee = new Employee(fname, lname, salary);**

**employeeID = (Integer) session.save(employee);**

**tx.commit();**

**} catch (HibernateException e) {**

**if (tx != null) tx.rollback();**

**e.printStackTrace();**

**} finally { session.close(); }**

**return employeeID;**

**}**

**// Method to READ all the employees**

**public void listEmployees() {**

**Session session = factory.openSession(new MyInterceptor());**

**Transaction tx = null;**

**try {**

**tx = session.beginTransaction();**

**List employees = session.createQuery("FROM Employee").list();**

**for (Iterator iterator = employees.iterator();iterator.hasNext();) {**

**Employee employee = (Employee) iterator.next();**

**System.out.print("First Name: " + employee.getFirstName());**

**System.out.print(" Last Name: " + employee.getLastName());**

**System.out.println(" Salary: " + employee.getSalary()); }**

**tx.commit();**

**} catch (HibernateException e) {**

**if (tx != null)**

**tx.rollback();**

**e.printStackTrace();**

**} finally { session.close(); }**

**}**

**// Method to UPDATE salary for an employee**

**public void updateEmployee(Integer EmployeeID, int salary) {**

**Session session = factory.openSession(new MyInterceptor());**

**Transaction tx = null;**

**try {**

**tx = session.beginTransaction();**

**Employee employee =**

**(Employee) session.get(Employee.class, EmployeeID);**

**employee.setSalary(salary);**

**session.update(employee);**

**tx.commit();**

**} catch (HibernateException e) {**

**if (tx != null)**

**tx.rollback();**

**e.printStackTrace();**

**} finally { session.close(); }**

**}**

**// Method to DELETE an employee from the records**

**public void deleteEmployee(Integer EmployeeID) {**

**Session session = factory.openSession(new MyInterceptor());**

**Transaction tx = null;**

**try {**

**tx = session.beginTransaction();**

**Employee employee =**

**(Employee) session.get(Employee.class, EmployeeID);**

**session.delete(employee);**

**tx.commit();**

**} catch (HibernateException e) {**

**if (tx != null)**

**tx.rollback();**

**e.printStackTrace();**

**} finally { session.close(); }**

**}**

**}**

## Kết quả run trên Eclipse

Create Operation

preFlush

postFlush

Create Operation

preFlush

postFlush

Create Operation

preFlush

postFlush

First Name: Zara Last Name: Ali Salary: 1000

First Name: Daisy Last Name: Das Salary: 5000

First Name: John Last Name: Paul Salary: 10000

preFlush

postFlush

preFlush

Update Operation

postFlush

preFlush

postFlush

First Name: Zara Last Name: Ali Salary: 5000

First Name: John Last Name: Paul Salary: 10000

preFlush

postFlush