

**Thực hiện :**

1. **Đào Đức Nhã**
2. **Trương Ngọc Tinh Anh**

**Research**

**NỘI DUNG**

[ĐẶT VẤN ĐỀ 5](#_Toc463191868)

[TỔNG QUAN VỀ HIBERNATE 6](#_Toc463191869)

[6.1 O/R Mappings (ORM) và một số khái niệm 6](#_Toc463191870)

[6.1.1 ORM 6](#_Toc463191871)

[6.1.2 JPA (Java Persistence API) là gì ? 7](#_Toc463191872)

[Khái niệm: 7](#_Toc463191873)

[JPA: 7](#_Toc463191874)

[Tại sao nên dùng JPA 7](#_Toc463191875)

[Ưu điểm của JPA 8](#_Toc463191876)

[Phiên bản hiện tại của đặc tả JPA 8](#_Toc463191877)

[Kiến trúc JPA. 8](#_Toc463191878)

[Các tính năng của JPA. 9](#_Toc463191879)

[6.1.3 Peristence layer: 10](#_Toc463191880)

[6. 2 Kiến trúc của hibernate 11](#_Toc463191881)

[6.2.1 Hibernate là gì ? 11](#_Toc463191882)

[Ưu điểm: 11](#_Toc463191883)

[Hiệu xuất: 11](#_Toc463191884)

[6.2.2 Kiến trúc 13](#_Toc463191885)

[Hibernate hỗ trợ trên các RDBMS: 15](#_Toc463191886)

[Hibernate hỗ trợ các kỹ thuật 16](#_Toc463191887)

[Môi trường làm việc 16](#_Toc463191888)

[Phát triển ứng dụng Hibernate 16](#_Toc463191889)

[Cấu trúc của project 16](#_Toc463191890)

[6.2.3 Vòng đời đối tượng trong Hibernate 17](#_Toc463191891)

[6.2.3.1 Trạng thái Transient 17](#_Toc463191892)

[6.2.3.2 Trạng thái Persistent 17](#_Toc463191893)

[6.1.2.3 Trạng thái Removed 18](#_Toc463191894)

[6.2.3.4 Trạng thái Detached 18](#_Toc463191895)

[6.3 Mapping Files 19](#_Toc463191896)

[6.4 Annotation 22](#_Toc463191897)

[6.4.1 Giới thiệu 22](#_Toc463191898)

[6.4.2 Annotation là gì: 22](#_Toc463191899)

[6.4.3 Configuration: 23](#_Toc463191900)

[6.4.4 Một số dạng Annotation 30](#_Toc463191901)

[6.4.4.1 @Entity: 30](#_Toc463191902)

[6.4.4.2 @id và @GeneratedValues(strategy = IDENTITY) 30](#_Toc463191903)

[6.4.4.3 @Table(name = (table\_name)) 31](#_Toc463191904)

[6.4.4.4 @Column 31](#_Toc463191905)

[6.4.4.5 @Temporal(Kiểu dữ liệu) 31](#_Toc463191906)

[6.4.4.4 @OneToOne 31](#_Toc463191907)

[6.4.4.5 @ManyToOne/@OneToMany 33](#_Toc463191908)

[6.5 Hibernate Queries Language (HQL) 37](#_Toc463191909)

[6.5.1 Giới thiệu: 37](#_Toc463191910)

[6.5.2 Mệnh đề From: 37](#_Toc463191911)

[6.5.3 Mệnh đề AS: 38](#_Toc463191912)

[6.5.4 Mệnh đề Where: 38](#_Toc463191913)

[6.5.5 Mệnh đề Order By 39](#_Toc463191914)

[6.5.6 Sử dụng tên tham số 40](#_Toc463191915)

[6.5.7 Mệnh đề Delete: 40](#_Toc463191916)

[6.5.8 Mệnh đề Update 41](#_Toc463191917)

[6.5.9 Mệnh đề Insert 42](#_Toc463191918)

[6.6 Criteria Queries 43](#_Toc463191919)

[6.6.1 Tạo một Criteria: 43](#_Toc463191920)

[6.6.2 Một số hàm Criteria queries thông dụng 43](#_Toc463191921)

[Hàm add() 43](#_Toc463191922)

[Hàm add() + một số Restrictions thông dụng 44](#_Toc463191923)

[AND or OR conditions 44](#_Toc463191924)

[6.6.3. Sắp xếp kết quả tìm kiếm 45](#_Toc463191925)

[6.6.4. Một số hàm đặc biệt 45](#_Toc463191926)

[6.6.5. Một số hàm thống kê: 45](#_Toc463191927)

[6.7 Native SQL 46](#_Toc463191928)

[6.7.1. Truy vấn vô hướng 46](#_Toc463191929)

[6.7.2. Truy vấn thực thể 46](#_Toc463191930)

[6.7.3. Truy vấn SQL có tên 46](#_Toc463191931)

[TÀI LIỆU THAM KHẢO 48](#_Toc463191932)

ĐẶT VẤN ĐỀ

Bất kỳ một ứng dụng doanh nghiệp (enterprise) nào cũng đều cần phải thực hiện các tháo tác (CRUD - Create, read, update, delete) đến database bằng việc lưư trữ, trích rút lượng lớn dữ liệu. Các nhà phát triển(developers) cần phải viết nhiều code để thực hiện các việc này.

Với công nghệ JDBC trước đây, developers sẽ phải chủ động thực hiện các việc: từ mở kết nối vào Database, tạo các Statement, ResultSet cho đến đóng tất cả các thứ đó lại, và code sẽ trở nên cồng kềnh. Developers cần phải làm việc với cả java code và SQL. Thực tế, từng Database khác nhau thì câu SQL có một số phần khác nhau, nên đòi hỏi developers phải nắm được sự khác biệt này để viết code cho phù hợp. Khi ứng dụng muốn chuyển từ database sang database khác(ví dụ từ Oracle sang MS SQL Server) thì chắc chắn sẽ có một số phần của câu SQL cần phải đổi. Công việc sửa code lại đòi hỏi developers phải test lại ứng dụng. Điều này sẽ tốn thời gian phát triển. Để khắc phục nhược điểm này, đã có rất nhiều frameworks ra đời với mục đích giúp xóa đi vấn để về tương thích giữa các Database, giúp developers tập trung vào phần xử lý nghiệp vụ.

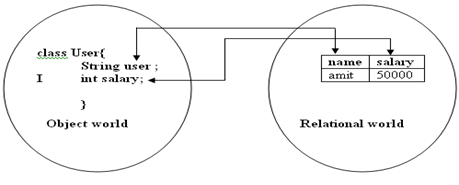
TỔNG QUAN VỀ HIBERNATE

6.1 O/R Mappings (ORM) và một số khái niệm

6.1.1 ORM

**ORM** là viết tắt của Object-Relational Mapping (ORM) là một kỹ thuật lập trình để chuyển đổi dữ liệu giữa các hệ thống khác (không phải là mô hình hướng đối tượng) sang các đối tượng trong ngôn ngữ lập trình hướng đối tượng. Trong trường hợp này chúng ta chuyển dữ liệu từ các dòng dữ liệu trong cơ sở dữ liệu quan hệ sang đối tượng

**ORM** là một phương pháp lập trình chuyển đổi từ mô hình database sang mô hình đối tượng



Hình ORM

**ORM** có nhiều thuận lợi hơn so với những phương pháp truy cập dữ liệu (data access) khác:

* ORM tự động hóa việc chuyển đổi từ object sang table và từ table sang object, giúp giảm thời gian và chi phí phát triển
* ORM cần ít code hơn store procedures, thay thế số lượng lớn store procedures cần phát triển
* Tăng tốc độ thực thi của hệ thống
* Một giải pháp ORM tốt sẽ giúp phát triển ứng dụng nhanh hơn và dễ hỗ trợ hơn

**Đặc điểm của ORM**

* Giúp cho người lập trình làm việc với đối tượng thay cho các bảng trên CSDL
* Che giấu đi chi tiết của các truy vấn SQL bằng OO logic
* Dựa trên JDBC nên
* Tự động hiện thực các đối tượng trong database
* Quản lý giao dịch và tạo khóa tự động

**Một số ORM Frameworks:**

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

**Khái niệm POJO (plain old Java object)**

Là một đối tượng Java thông thường không bị ràng buộc bởi bất kỳ một hạn chế đặc biệt nào

6.1.2 JPA (Java Persistence API) là gì ?

Khái niệm:

Java Persistence API hay JPA là một đăc tả Java cho việc ánh xạ giữa các đối tượng Java tới cơ sở dữ liệu quan hệ sử dụng cộng nghệ phổ biến là ORM( Object Relational Mapping). JPA API cung cấp đầy đủ các công cụ cho phép người lập trình có thể tạo cơ sở dữ liệu một cách đơn giản và nhanh chóng. JPA API có thể dùng để persist một đối tượng business (POJO) vào trong cơ sở dữ liệu hoặc lấy dữ liệu từ cơ sở dữ liệu và ánh xạ ra các đối tượng business một cách đơn giản.

JPA là một đăc tả của Sun, ra đời cùng với bản đặc tả J2EE 5 và JPA như là thành phần của EJB 3.x (JSR 220). JPA chỉ là đặc tả, không phải sản phẩm cụ thể của tần dữ liệu (persistence). Do đó, để cài đặt JPA phải dùng ORM Frameworks cụ thể. Các framework ORM có thể sử dụng cho JPA như **Hibernate**, Toplink, Open JPA,…

Ngày nay đa số các nhà cung cấp đều hỗ trợ cài đặt JPA cho framework persistence của họ. Vì vây, người lập trình có thể lựa chọn nhà cung cấp tốt nhất tùy thuộc yêu cầu ứng dụng của mình.

JPA:

* Là một đặc tả java cho việc ánh xạ (mapping) cho các đối tượng java tới quan hệ cơ sở dữ liệu
* Là công cụ cho phép người dùng tạo CSDL một cách nhanh chóng và đơn giản
* Bao đóng các tác vụ truy xuất tới CSDL

Tại sao nên dùng JPA

* JPA là một đặc tả chuẩn cho các tác vụ ở tầng dữ liệu
* JPA sử dụng các lớp POJO (các class truyền thống)
* Có nhiều framework ORM miễn phí hỗ trợ có thể dùng để phát triển nhiều loại ứng dụng khác nhau.
* Ứng dụng xây dựng trên JPA mang tính di động cao
* Có thể sử dụng cho cả ứng dụng J2EE và J2SE
* Hỗ trợ cầu hình triển khai bằng annotation và xml.

Ưu điểm của JPA

* Đơn giản hóa công nghệ cho tầng persistence (tầng dữ liệu)
* Không phụ thuộc vào các framework ORM (các ORM thường hỗ trợ các đặc tả của JPA và những khái niệm riêng của công nghệ đó)
* Có nhiều nhà cung cấp hỗ trợ cài đặt JPA (miễn phí và thương mại)
* Dữ liệu có thể được lưu trữ thông qua việc ORM. Do đó, vấn đề security và transaction tốt hơn

Phiên bản hiện tại của đặc tả JPA

Hiện tại JPA đã có phiên bản 3.1 (xuất bản ngày 22/04/2013), cung cấp nhiều tính năng hữu ích cho người lập trình.

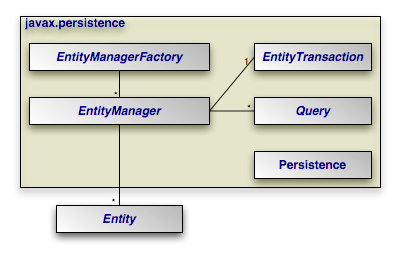
Kiến trúc JPA.

Gồm:

- EntityManagerFactory

- EntityManager

- Entity



Hình Kiến trúc của JPA

*Entity*

Entity là các đối tượng persistence (thực thể của lớp Java) tương ứng với các mẫu tin trong các bảng cơ sở dữ liệu. Entity chỉ là các lớp POJO (là những lớp java ánh xạ từ các bảng trong cơ sở dữ liệu) đơn giản, dễ dàng tạo.

Đặc điểm của một Entity:

* Entity có thể tương tác với cơ sở dữ liệu quan hệ.
* Entity được xác định thông qua một định danh( tương đương với khóa chính trong table của cơ sở dữ liệu quan hệ)
* Entity hỗ trợ giao tác (transaction)
* Entity hỗ trợ kế thừa giống như những lớp Java thông thường khác.

*EntityManager*

EntityManager là một giao diện (interface) cung cấp các API cho việc tương tác với các Entity (quản lý tập các đối tượng).

Một số chức năng cơ bản của EntityManager như:

* Persist: phương thức này dùng để lưu một thực thể mới tạo vào cơ sở dữ liệu
* Merge: dùng để cập nhật trạng thái của entity vào cơ sở dữ liệu.
* Remove: xóa một thể hiện của entity.

*EntityManagerFactory:*

EntityManagerFactory được dùng để tạo ra một thể hiện của EntityManager

*EntityTransaction:*

Có quan hệ 1 - 1 với EntityManage hỗ trợ thao tác transaction cho một tập các tác vụ tới CSDL

*Query*

JPA hỗ trợ thực hiện truy vấn dùng cả hai ngôn ngữ truy vấn theo Java Persistense (Java Persistence Query Language - JPQL) và ngôn ngữ truy vấn có cấu trúc (SQL)

Các tính năng của JPA.

Việc giới thiệu JPA vào trong đặc tả J2EE 5 là một bước tiến lớn trong việc đơn giản hóa các quy trình phát triển ứng dụng. JPA đơn giản hóa mô hình thực thể dữ liệu và cộng thểm một số tính năng mới mà phiên bản EJB trước (EJB 2.0) không có. Giờ đây người lập trình có thể ánh xạ trực tiếp các đối tượng persistence với cơ sở dữ liệu quan hệ. JPA có thể sử dụng bên ngoài container, điều này không dễ thực hiện trong EJB 2.1. Bạn cũng có thể sử dụng JPA trong các ứng dụng swing.

Một số tính năng của JPA:

* JPA hỗ trợ pluggable, tức là có thể sử dụng nhiều nhà hãng cung cấp thứ ba như Hibernate hay Toplink.
* Hỗ trợ annotation
* Giảm bớt số lớp yêu cầu cho việc phát triển persistence.
* Không cần phải viết các mô tả triển khai trong xml. Annotation dựa trên metadata đã hỗ trợ trong các ứng dụng JPA.
* Đã chuẩn hóa ORM và dễ dàng phát triển hơn
* JPA hỗ trợ truy vấn động và tĩnh.
* Nhiều IDE hỗ trợ phát triển ứng dụng JPA và có thể tự động sinh code ánh xạ từ cơ sở dữ liệu thành các entity và ngược lại.

6.1.3 Peristence layer:

Là một ứng dụng có thể được chia làm 3 phần như sau: giao diện người dùng (presentation layer), phần xử lý nghiệp vụ (business layer) và phần chứa dữ liệu (data layer). Cụ thể ra, business layer có thể được chia nhỏ thành 2 layer con là business logic layer (các tính toán nhằm thỏa mãn yêu cầu người dùng) và persistence 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 sẽ đảm nhiệm các nhiệm vụ mở kết nối, truy xuất và lưu trữ dữ liệu vào các Relational DBMS.

6. 2 Kiến trúc của hibernate

6.2.1 Hibernate là gì ?

**Hibernate** vốn là một **thư viện** sinh ra để làm việc với **mọi loại Database**, nó không phụ thuộc vào bạn chọn loại DB nào. Nếu **Java** là "**Viết 1 lần chạy mọi nơ**i" thì **Hibernate** là "**Viết 1 lần chạy trên mọi loại Database**"

- Ra đời năm 2001 bởi Christian Bauer & Gavin King

- Là một ORM (Object-Relational mapping) framework mã nguồn mở được sử dụng khá phổ biến

- Hiện thực cho chuẩn JPA

- Hibernate giúp trong suốt dữ liệu đối với người dùng và lưu trữ tự động

- Xây dựng các đối tượng lưu trữ (persistent objects) dựa theo các khái niệm của OOP: Kế thừa (Inheritance), Đa hình (Polymorphism), các mối quan hệ

Ưu điểm:

- Giúp người phát triển tập trung mô hình đối tượng (domain objects model) mà không quan tâm sâu xuống hệ thống lưu trữ

- Hibernate tập trung vào việc ánh xạ các lớp Java vào các bảng trong CSDL bằng XML hoặc annotation mà không cần phải viết mã Java

- Hibernate cung cấp API đơn giản để lưu trữ và truy xuất các đối tượng Java trực tiếp từ cơ sở dữ liệu

- Nếu có sự thay đổi trong cơ sở dữ liệu hoặc trong bất kỳ bảng nào, chúng ta chỉ cần thay đổi các thuộc tính trong tập tin XML hoặc annotation

- Hibernate có thể được sử dụng cho một ứng dụng máy tính để bàn độc lập và cũng cho các ứng dụng dựa trên web, Tóm lại nó là thích hợp cho tất cả các loại ứng dụng (đứng một mình hoặc máy tính để bàn hoặc bất kỳ ứng dụng web

Hiệu xuất:

- Hiệu suất với việc sử dụng cache lưu trữ đối tượng

- Chiến lược cấu hình cụ thể

Những tiện ích truy vấn phức tạp

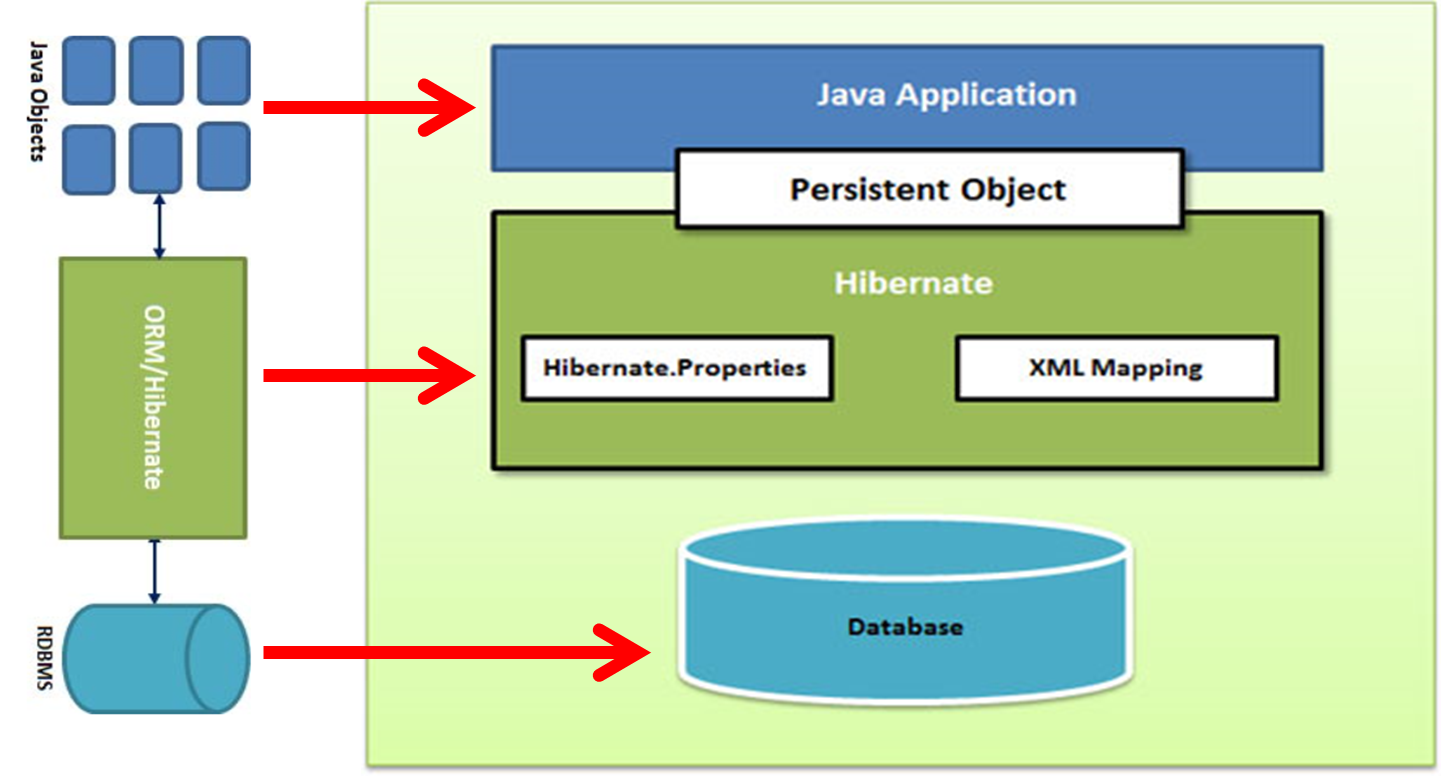
- Cung cấp nhiều API

- Nhiều ví dụ về truy vấn

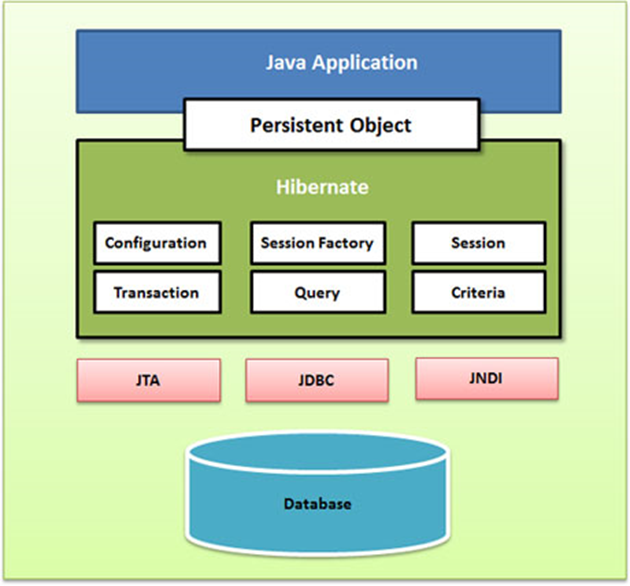
- Ngôn ngữ truy vấn (HQL)

- Câu truy vấn chuẩn (Native SQL)

6.2.2 Kiến trúc



Hình Kiến trúc của Hibernate

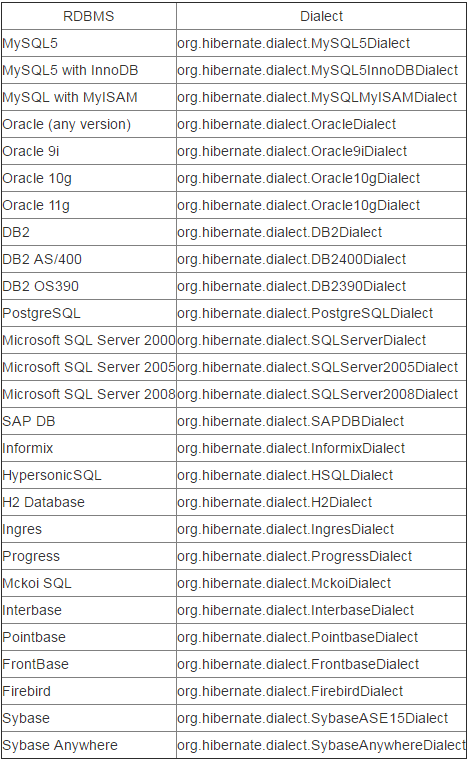


Hình Kiến trúc mở rộng của Hibernate

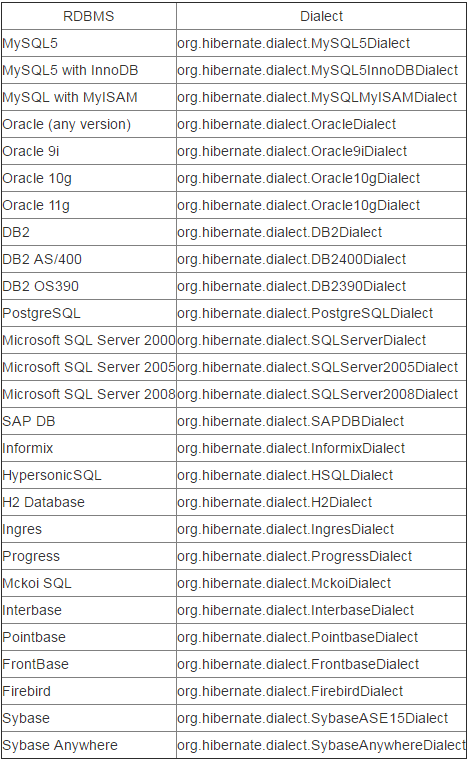
Trong đó:

* **Configuration:** là đối tượng đầu tiên được tạo ra khi tạo ứng dụng Hibernate. Gồm 2 thành phần
  + **Database Connection:** tập tin được hỗ trở hibernate. Là **hibernate.properties** và **hibernate.cfg.xml**. Chứa các thông số cấu hình cho ứng dụng
  + **Class Mapping Setup**: lớp hỗ trợ việc mapping
* **Configuration object** được sử dụng để tạo **SessionFactory** object nhằm cấu hình Hibernate cho ứng dụng, cho phép một đối tượng session được khởi tạo. **SessionFactory** là một thread để bảo vệ đối tượng và được sử dụng bởi tất cả các thread trong ứng dụng
* **Session:**
  + Được sử dụng để nhận một kết nối vật lý với CSDL. Các Session được khởi tạo mỗi khi cần tương tác đến CSDL. Các đối tượng Persistent được lưu trữ và lấy thông tin thông qua một đối tượng Session
  + Các đối tượng session không nên mở rộng một khoảng thời gian dài vì chúng thường không được thread safe và chúng cần được tạo ra và hủy khi cần thiết
* **Transaction object:**
  + Đại diện cho một unit làm việc với CSDL và hầu hết các RDBMS có hỗ trợ chức năng transaction
  + Transaction trong hibernate được quản lý bởi một bộ quản lý transactio và transaction (từ JDBC hoặc JTA)
  + Đây là một đối tượng tùy chọn và ứng dụng Hibernate có thể chọn không sử dụng giao diện này, thay vào đó nó sẽ quản lý transaction trực tiếp trong code của ứng dụng
* **Query:** 
  + Query sử dụng chuỗi SQL hoặc Hibernate Query Language (HQL) để nhận dữ liệu từ CSDL và tạo ra các đối tượng
  + Một Query instance được sử dụng để ràng buộc các tham số truy vấn, giới hạn số lượng kết quả trả về của câu truy vấn, và cuối cùng để thực hiện truy vấn
* **Criteria Object:**
  + Đối tượng này được sử dụng để tạo và thực hiện truy vấn điều kiện theo hướng đối tượng để lấy các đối tượng
* **Thành phần JDBC, JTA, JNDI**
  + JDBC, JTA(Java Transaction API), JNDI( Java Naming and Directory Interface) là các API có sẵn của Java.´
  + JDBC cung cấp một mức trừu tượng cơ sở của các chức năng thường dùng trên CSDL quan hệ, cho phép hibernate kết nối đến bất kì CSDL nào được hổ trợ bởi JDBC driver.´
  + JTA và JNDI cho phép Hibernate được tích hợp với J2EE application servers.

Hibernate hỗ trợ trên các RDBMS:



Hình Các rDBMS hỗ trợ Hibernate

Hibernate hỗ trợ các kỹ thuật

Hình Các rDBMS hỗ trợ Hibernate

* J2EE
* XDoclet Spring
* Eclipse plug-ins
* Maven

Môi trường làm việc

* NetBeans IDE 8.0.2
* JDK 8
* Apache Tomcat 7 / 8
* Wamp Server (My SQL 5.x)

Phát triển ứng dụng Hibernate

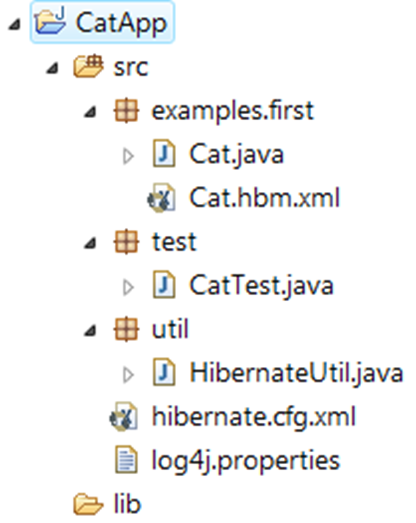
1.Từ mô hình lớp phân tích, thiết kế các lớp POJO (Plain Old Java Objects), biểu diễn các đối tượng persistence.

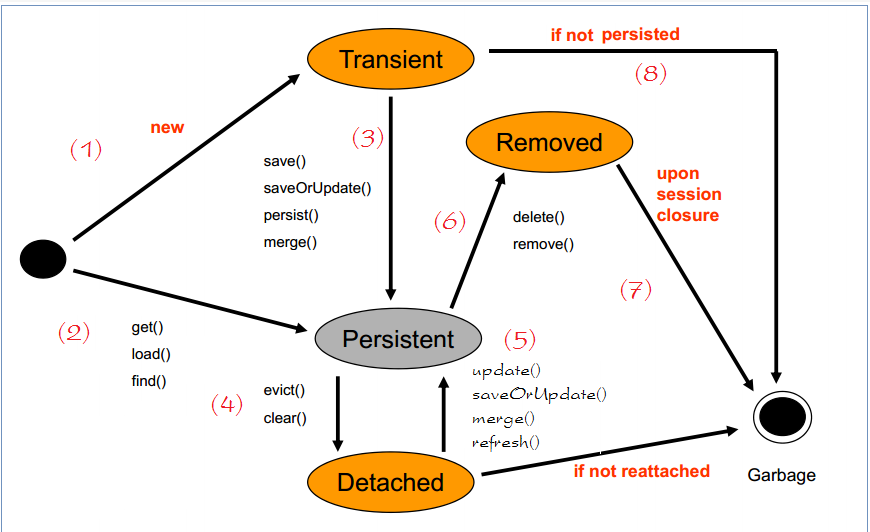
2.Tạo mapping để ánh xạ các đối tượng thành bảng quan hệ (dùng XML hoặc annotation mapping)

3.Tạo file cấu hình hibernate.cfg.xml xác định các thiết lập về kết nối JDBC tới cơ sở dữ liệu

4.Thực hiện Hibernate API để lưu trữ, cập nhật, truy vấn,… các đối tượng persistence

Cấu trúc của project



Hình Cấu trúc của projects

Hình Vòng đời đối tượng trong Hibernate

6.2.3 Vòng đời đối tượng trong Hibernate

6.2.3.1 Trạng thái Transient

* Tất cả các đối tượng khi mới được khởi tạo có trạng thái Transient:  
   Account account = new Account();  
   account là một đối tượng Transient
* Hibernate không biết các đối tượng này
* Không liên quan với dòng trong CSDL
* Đối tượng này sẽ bị mất khi không được tham chiếu.

6.2.3.2 Trạng thái Persistent

* Hibernate biết và quản lý đối tượng này
* Đối tượng có thể là :
  + Lấy một đối tượng đã tồn tại trong CSDL
  + Một đối tượng có trạng thái Transitent đã được lưu
* Thay đổi các đối tượng có trạng thái Persistent được tự động lưu xuống CSDL mà không cần gọi các phương thức thực hiên
* Các đối tượng được đặt làm Persistent khi gọi các phương thức :  
  session.save(account); session.lock(account);  
  session.update(account); session.merge(account

6.1.2.3 Trạng thái Removed

* Một đối tượng có trạng thái Persistent bị xóa khỏi CSDL

session.delete(account);

* Đối tượng Java có thể vẫn tồn tại nhưng nó được bỏ qua bởi Hibernate
* Bất kì thay đổi nào sẽ không thay đổi trong CSDL

6.2.3.4 Trạng thái Detached

* session.close() thay đổi trạng thái đối tượng từ Persistent thành Detached
* Vẫn đại diện cho một dòng trong CSDL
* Không còn được quản lý bởi Hibernate   
   Sự thay đổi của đối tượng không được lưu trong CSDL
* Có thể reattached, đưa về trạng thái Persistent và đối tượng này được lưu vào CSDL

update();

merge();

lock(); // reattaches, nhưng không lưu thay đổi

* Đầu tiên tạo mới một đối tượng từ một entity thì nó sẽ có trạng thái là Transient, ko được hibernate quản lý.
* Nếu lấy một đối tượng Entity bằng các phương thức get, load, find thì đối tượng này ở trạng thái bền vững(Persistent) và được hibernate quản lý.
* Session gọi một trong các phương thức save, saveOrUpdate, persits, merge thì nó sẻ đẩy Entity từ trạng thái Transient sang trang thái bền vững(Persistent)
* Session dùng các hàm evict, clear để đẩy các Entity trang thái bền vững ra khỏi sự quản lý của hibernate vafsex chuyển sang trạng thái Detached
* Sử dụng update(..), saveOrUpdate(..), merge(..) sẽ đính trở lại các đối tượng Detached vào lại. Tùy tình huống nó sẽ tạo ra dưới DB câu lệnh update hoặc insert. Các đối tượng sẽ trở về trạng thái Persistent (bền vững)
* Session gọi method remove(..), delete(..) để xóa một bản ghi, đối tượng persistent giờ chuyển sang trạng thái Removed (Đã bị xóa).

6.3 Mapping Files

Mapping file là một file XML chứa ánh xạ chi tiết giữa persistence class và bảng nó thay thế. Các thành phần bắt buộc của file XML này là:

* **hibernate-mapping:** Đây là thành phần gốc bao tất cả các thành phần khác
* **class:** được sử dụng để ánh xạ tên bản thành persistence class. Tên thuộc tính được sử dụng để xác định tên lớp, và thuộc tính bản được sử dụng để xác định bảng mà lớp thay thế. Ví dụ để ánh xạ một bản mang tên ORDERS thành một persistence class với tên đầy đủ là com.someorg.persist.Order sẽ như dưới đây:

<class name=”com.someorg.persist.Order” table=”ORDERS”>…</class>

* **id:** Thành phần này sử dụng để ánh xạ khóa chính của table thành một instance variable của class. Thành phần cột con của id có thể sử dụng để ánh xạ thành biến tương ứng. Giá trị của khóa chính có thể tự động tạo ra được công bố ở đây. Thành phần generator có thể được sử dụng để nói cho Hibernate một lớp sẽ được sử dụng ở đâu để tự động tạo id, hoặc id được gán bởi ứng dụng.

<id name="id" type="string" unsaved-value="null">

<column name="id" sql-type="char(32)" not-null="true"/>

<generator class="assigned"/>

</id>

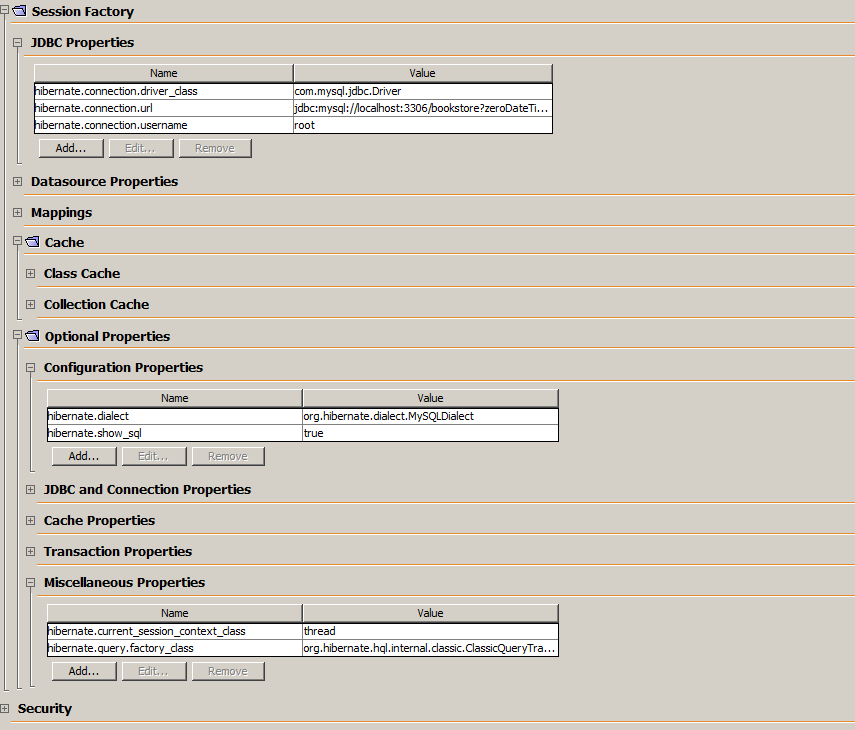
Ví dụ trên nói cho Hibernate rằng, tên của khóa chính là id mà nó đã được ánh xạ thành instance variable "id". Kiểu dữ liệu của nó là "string" được ánh xạ cho cột "id" có kiểu "char(32)" thay thế bởi thuộc tính sql-type. Thuộc ính lớp của generator chứa giá trị "assigned" nghĩa là ứng dụng bản thân nó sẽ cung cấp giá trị cho khóa chính.

* **property:** thành phần này, thuộc về các cột thành phần con của nó, ánh xạ những thuộc tính khác (hoặc các côt thành những instance variable của persistence class. Thuộc tính name của property chứa tên của biến như một giá trị. Thuộc tính name của thành phần column chứa tên của cột mà instance variable được ánh xạ. Các thuộc tính length và sql-type trỏ đến độ dài và kiểu dữ liệu của cột

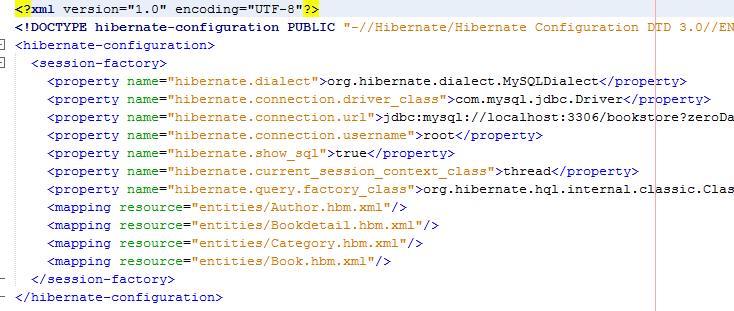
<property name="name">

<column name="name" sql-type="char(255)" not-null="true"/>

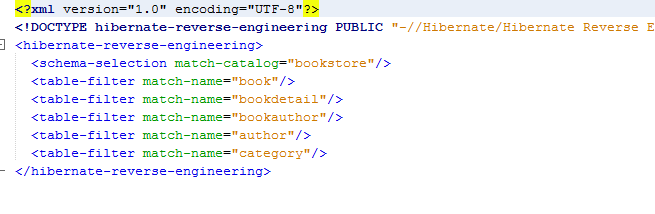
</property>



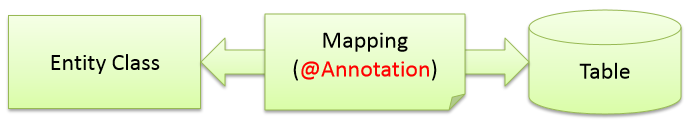
Hình Cấu hình Hibernate 1



Hình Cấu hình Hibernate 2



Hình Cấu hình Hibernate 3

6.4 Annotation

Hình Annotation

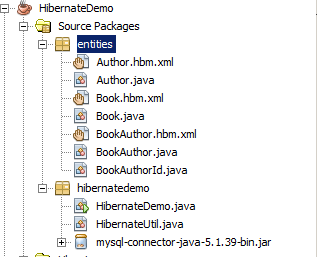
6.4.1 Giới thiệu

* Giống như các tool O/R mapping khác, yêu cầu metadata điều chỉnh sự thay đổi từ dữ liệu từ 1 đại diện khác. Hibernate annotation cung cấp các annotation đẻ mapping metadata.
* JPA chỉ rỏ cách nhận biết sự thành công của 1 O/R mapping minh bạch

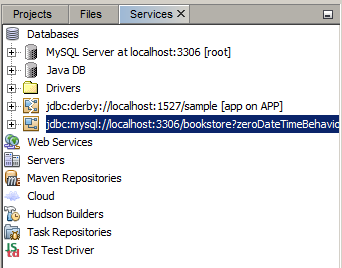
6.4.2 Annotation là gì:

* Annotation (Chú thích) được sử dụng để cung cấp thông tin dữ liệu cho mã Java của bạn. Là thông tin dữ liệu, các Annotation không trực tiếp ảnh hưởng đến việc thực hiện các mã của bạn, mặc dù một số loại chú thích thực sự có thể được sử dụng cho mục đích đó. Annotationđã được thêm vào Java từ Java 5
* Annotation được sử dụng cho các mục đích:
* Chỉ dẫn cho trình biên dịch (Compiler): Java có sẵn 3 Annotation mà bạn có thể sử dụng để cung cấp cho các hướng dẫn để trình biên dịch Java.
* @Deprecated: Đây là một Annotation dùng để chú thích một cái gì đó bị lỗi thời, tốt nhất không nên sử dụng nữa, chẳng hạn như class, hoặc method.
* Chú thích @Deprecated được bộ biên dịch quan tâm để thông báo cho bạn nên dùng một cách nào đó thay thế. Hoặc với các IDE lập trình chẳng hạn như Eclipse nó cũng sẽ có các thông báo cho bạn một cách trực quan.
  + - @Override
    - @SuppressWarnings: Chú thích @SuppressWarnings làm cho các trình biên dịch thôi không cảnh báo một vấn đề của method nào đó. Ví dụ, nếu trong một method có gọi tới một method khác đã lỗi thời, hoặc bên trong method có một ép kiểu không an toàn, trình biên dịch có thể tạo ra một cảnh báo. Bạn có thể tắt các cảnh báo này bằng cách chú thích method này bằng @SuppressWarnings.
* Chỉ dẫn trong thời điểm xây dựng (Build-time): Annotation có thể được được sử dụng tại thời xây dựng (Build-time), khi bạn xây dựng dự án phần mềm của bạn. Quá trình xây dựng bao gồm tạo ra các mã nguồn, biên dịch mã nguồn, tạo ra các file XML (ví dụ như mô tả triển khai), đóng gói mã biên dịch và các tập tin vào một tập tin JAR, v..v. Xây dựng phần mềm thường được thực hiện bởi một công cụ xây dựng tự động như Apache Ant hoặc Apache Maven . Xây dựng các công cụ có thể quét mã Java của bạn và dựa vào các chú thích (Annotation) của bạn để tạo ra mã nguồn hoặc các tập tin khác dựa trên những chú thích đó.
* Chỉ dẫn trong thời gian chạy (Runtime): Thông thường, các Annotation không có mặt trong mã Java của bạn sau khi biên dịch. Tuy nhiên có thể xác định các Annotation của bạn trong thời gian chạy. Các chú thích này sau đó có thể được truy cập thông qua Java Reflection, và được sử dụng để cung cấp cho các hướng dẫn chương trình của bạn, hoặc API của một số bên thứ ba (Third party API).
* Việc chuyển đổi giá trị từ một trường từ Java Object sang một trường của bảng trong database thì phải đánh dấu trường đó tên gì, thuộc bảng nào
* Trong hibernate chúng ta có thể chú thích trực tiếp trên class mà không cần tạo ra các file hbm.xml

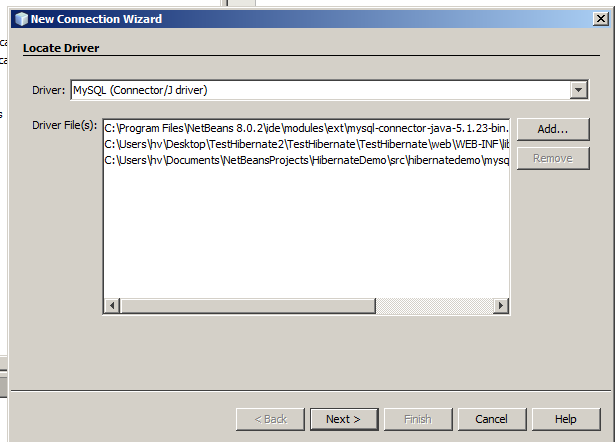
6.4.3 Configuration:

* Đầu tiên phải tải thư viện JDBC và copy bỏ vào thư mục Source(để tiện tạo kết nối,và di chuyển đi nơi khác).
* 

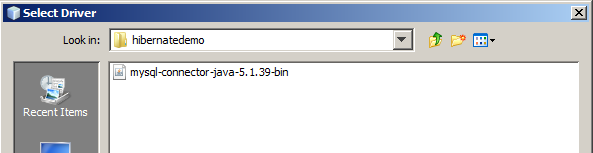
Hình File configuration

* Vào server
* 

Hình Cấu hình server

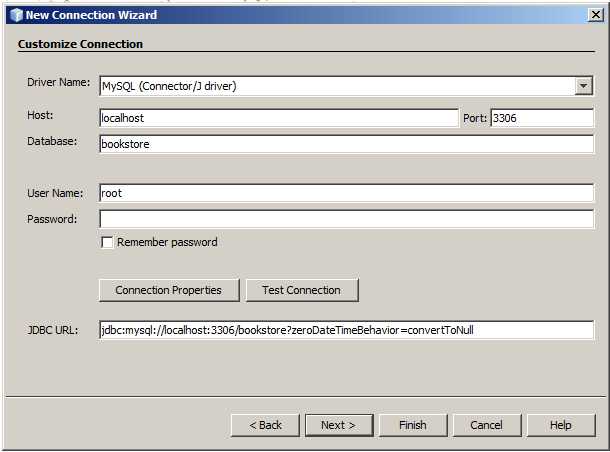
* Tạo 1 kết nối jdbc tới database : Nhấp chuột phải vào Database -> chọn New Connection-> Phần driver chọn MySQL(Connector/J driver) -> Add-> chỉ đường dẫn tới thư mục chứa file jdbc vừa tải
* 

Hình Cấu hình database



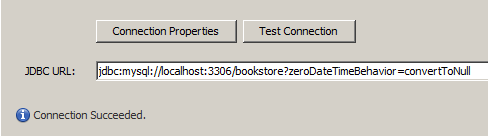
Hình Chọn thư visện connector

-> chọn open-> chọn next trong option New connection wizard ta sẽ có cửa sổ như sau



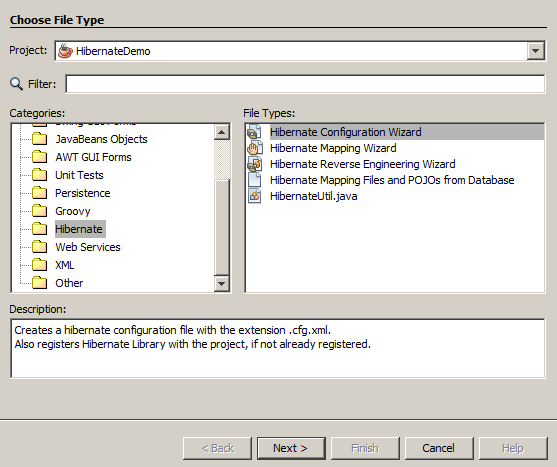
Hình Nhập thông tin cấu hình

* Phần database la tên database sẽ kêt nối tới -> test connection



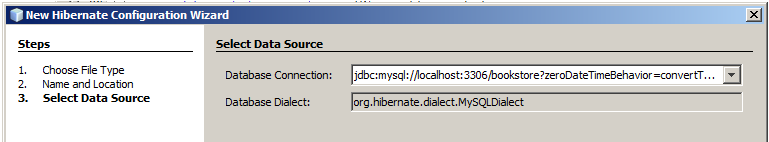
Hình Chọn JDBC URL

* Nếu test thành công thì chọn Finish.
* Sau khi tạo kết nối thành công thì về lại thư mục project tạo 1 file hibernate configuration wizard



Hình Tạo file Hibernate Configuration Wizard

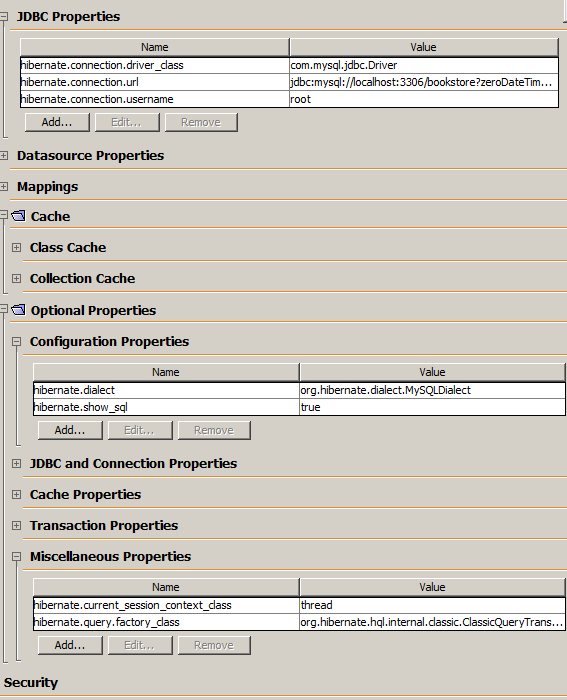
* Trong mục database connection chọn connection vừa mới tạo-> Finish



Hình Chọn database connections

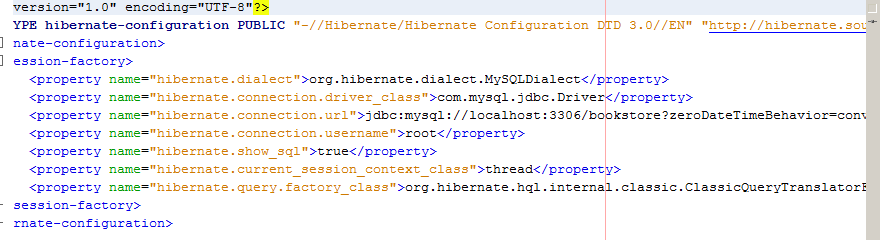
**=>** tạo ra 1 file hibernate.cfg.xml

* Cấu hình file đó như hình



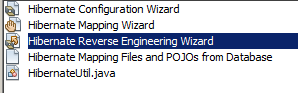
Hình Cấu hình file config

* Có thể vào mục source để xem nội dung cua file đó



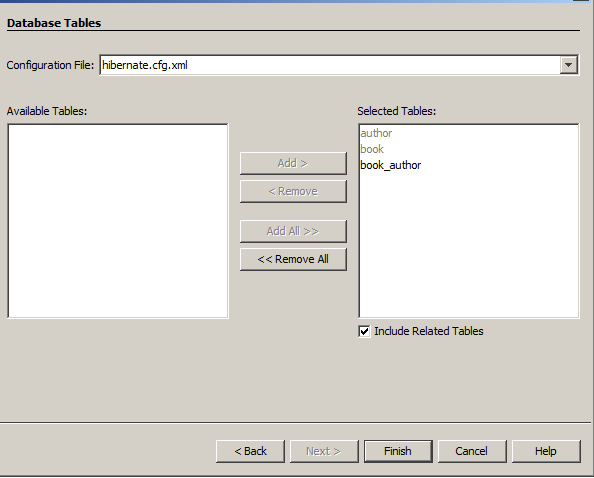
Hình Nội dung file config

* Tiếp theo tao file hibernate.reveng.xml



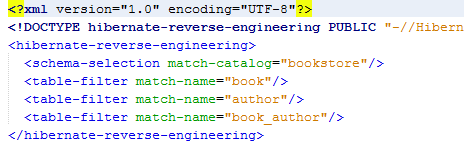
Hình Chọn file Reverse

* Chọn những bảng muốn mapping đến=> Chọn finish



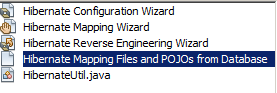
Hình Cấu hình hibernate.cfg.xml

* Nội dung file :



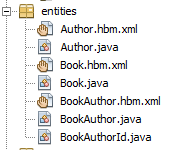
Hình Nội dung file hibernate.cfg.xml

* Tiếp theo tạo những class pojo bang cách



Hình Tạo file mappings

**=**> Các class sẽ được tự động tạo trong project

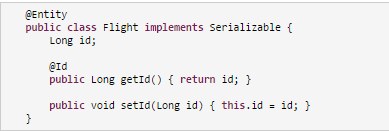


Hình POJO class và mapping files

6.4.4 Một số dạng Annotation

6.4.4.1 @Entity:

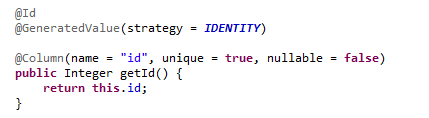
Xác định lớp này như một thực thể



Hình Entity

6.4.4.2 @id và @GeneratedValues(strategy = IDENTITY)

Xác định thuộc tính làm khóa chính

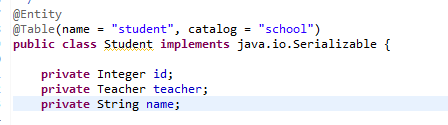


Hình Id và Denerated values

Nếu bảng đó là kiểu số nguyên tự tăng thì thêm chú thích **@GeneratedValue(strategy = IDENTITY)**

6.4.4.3 @Table(name = (table\_name))

**Chỉ định Entity này ánh xạ tới table nào trong database**



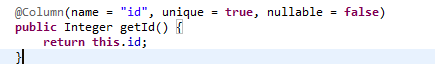
Hình Table

Có thể thêm ràng buộc unique vào table bằng **@UniqueConstraint**

Hình Unique Constraint

6.4.4.4 @Column

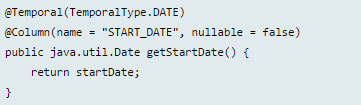
* Xác định thuộc tính trong java class sẽ ánh xạ tới column nào trong table



Hình Column

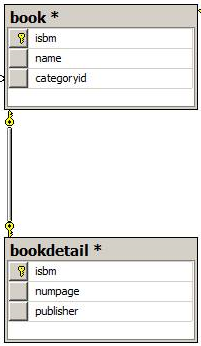
6.4.4.5 @Temporal(Kiểu dữ liệu)

**Dùng để chú thích cho kiểu ngày tháng**



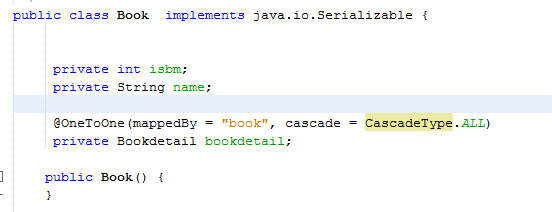
Hình Temporal

6.4.4.4 @OneToOne

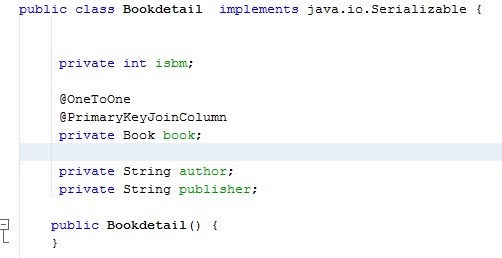
Quan hệ OneToOne dùng để chia sẻ các khóa chính. Thêm các annotation trong các class như Hình sau:

Hình One to one

Ở đây thì 1 book có 1 bookdetail duy nhất và 1 bookdetail thuộc về 1 book duy nhất



Hình Annotation trong class Book



Hình Annotation trong class Bookdetail

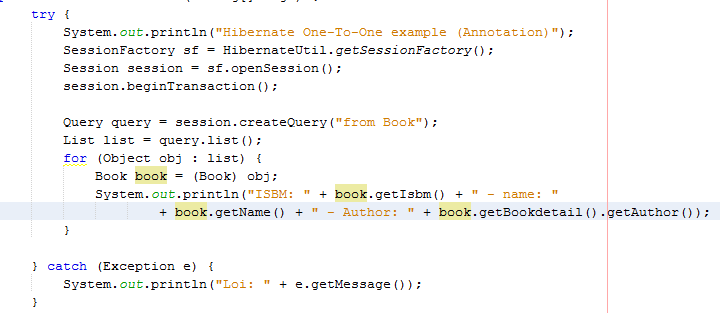
Hoặc nó đã được định nghĩa trong file xml



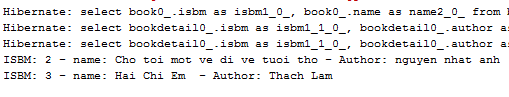
Hình Định nghĩa one to one trong class bookdetail

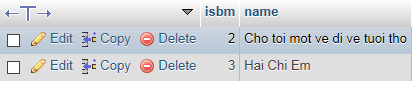


Hình Định nghĩa one to one trong class book



Hình Ví dụ về annotation one to one





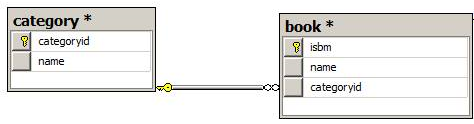


6.4.4.5 @ManyToOne/@OneToMany

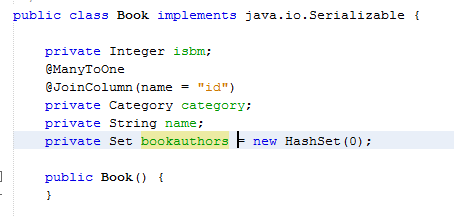
A tham chiếu tới 1 thể hiện của B

B tham chiếu tới 1 tập các thể hiện của A

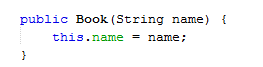
A phải là chủ của quan hệ

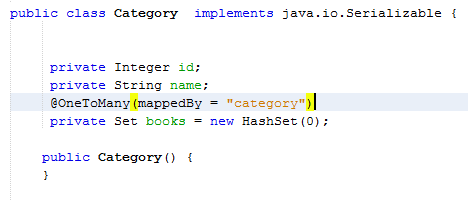


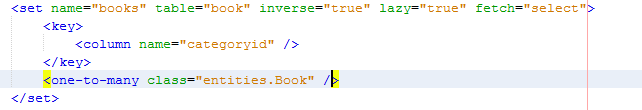
Ở đây một category thì có nhiều cuốn sách và 1 cuốn sách thì thuộc vào 1 category nào đó mà thôi

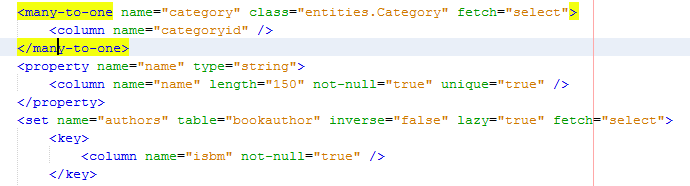


Ở class book có thể tạo thêm 1 constructor

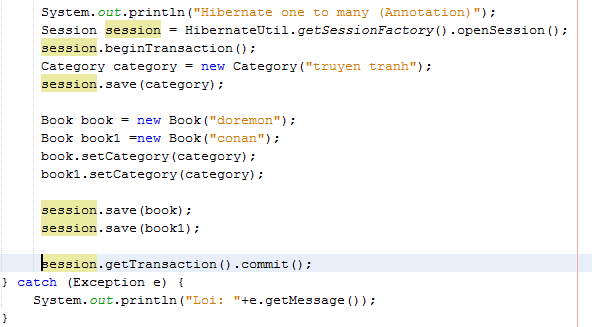




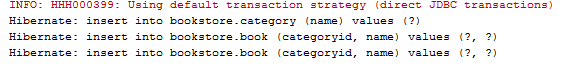




Có các phương thức fetch đũ liệu, mặc định là lazy

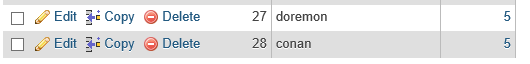


Kết quả của chương trình

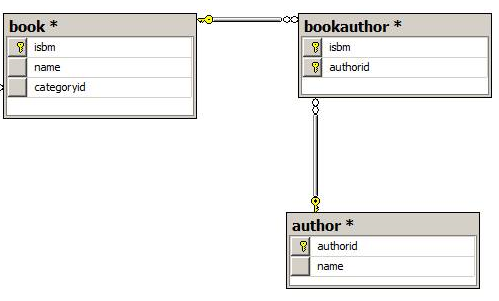


Kết quả trên database





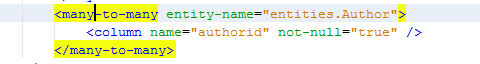
**6.4.4.6** @MayToMany

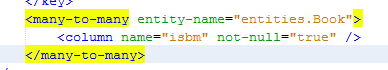
A tham chiếu tới 1 tập thể hiện của B

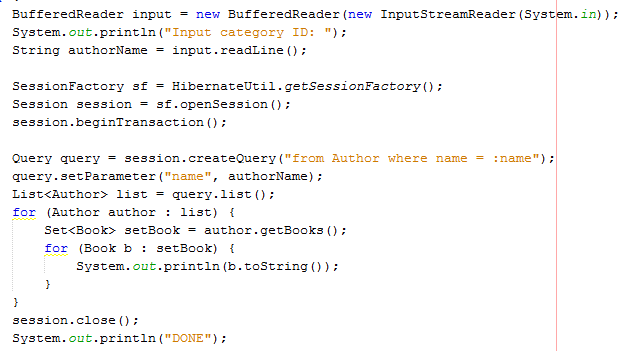
B tham chiếu tới 1 tập thể hiện của A

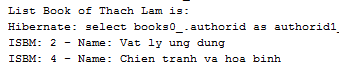
Phải tạo 1 bảng trung gian

Ở đây 1 Book thì có nhiều tác giả và 1 tác giả có nhiều cuốn sách, và chúng sẽ quan hệ với nhau qua 1 bang bookauthor có 2 khóa chính là isbm của bang book và authorid của bảng author









Có 2 kiểu FetchType là Eager và Lazy.

http://stackoverflow.com/questions/2990799/difference-between-fetchtype-lazy-and-eager-in-java-persistence-api

EAGER loading of collections means that they are fetched fully at the time their parent is fetched. So if you have Course and it has List<Student>, all the students are fetched *from the database* at the time the Course is fetched.

LAZY on the other hand means that the contents of the List are fetched only when you try to access them. For example, by calling course.getStudents().iterator(). Calling any access method on the List will initiate a call to the database to retrieve the elements. This is implemented by creating a Proxy around the List (or Set). So for your lazy collections, the concrete types are not ArrayList and HashSet, but PersistentSet and PersistentList (or PersistentBag)

Kiểu Eager nghĩa là load đối tượng lên thì load luôn cả khóa ngoại của nó

Kiểu Lazy thì nó chỉ load đối tượng lên, khi nào cần dùng đến khóa ngoại thì mới khai báo. Hiệu suất của kiểu Lazy tốt hơn kiểu Eager

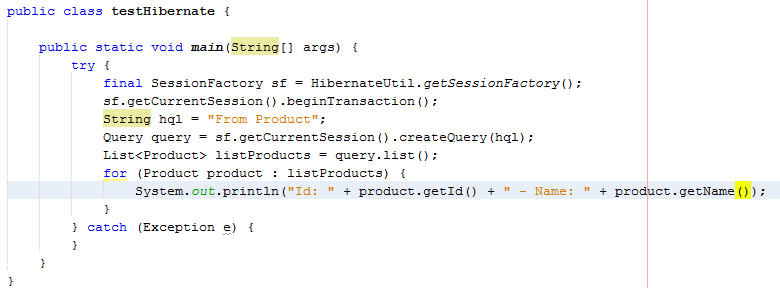
6.5 Hibernate Queries Language (HQL)

6.5.1 Giới thiệu:

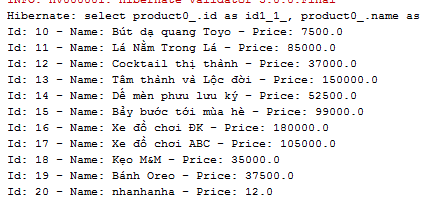
* Hibernate Queries Language (HQL) là một ngôn ngữ truy vấn hướng đối tượng, tương tự như SQL, nhưng nó thực hiện trên các đối tượng persistent của các POJO class. Có thể sử dụng trực tiếp câu lệnh SQL với Hibernate vì HQL sẽ dịch các câu lệnh HQL sang câu lệnh SQL
* Các từ khóa như select, from, where,... không phân biệt chữ hoa chữ thường, còn tên bảng, tên cột trong HQL phân biệt chữ hoa, chữ thường
* HQL chỉ dùng để show dữ liệu lên, cập nhật, xóa dữ liệu tương tự

6.5.2 Mệnh đề From:

* Khi bạn muốn lấy tất cả dữ liệu trong bảng

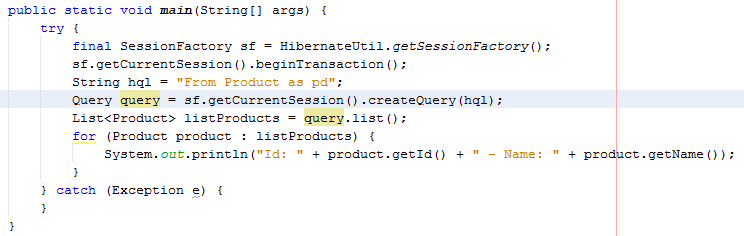


=> Kết quả:

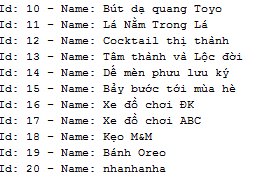


6.5.3 Mệnh đề AS:

* Dùng để gán bí danh cho bảng, được dùng phổ biến trong các câu truy vấn dài

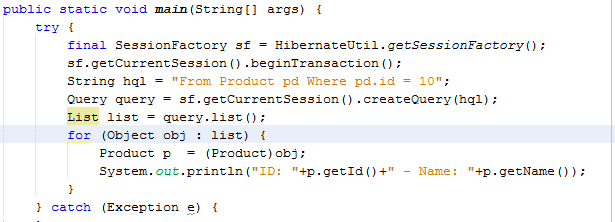


=> Kết quả

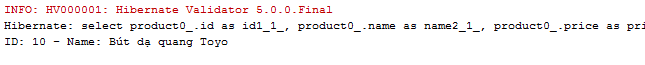


6.5.4 Mệnh đề Where:

* Là điều kiện trong câu truy vấn, dựa vào nó mà trả về kết quả cho câu truy vấn

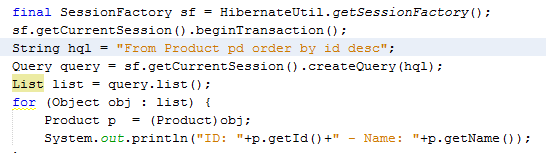


=> Đặt bí danh cho bảng product la pv => rút gọn câu truy vấn => dùng bi danh để truy xuất thuộc tính id và xuất ra sản phẩm có Id bằng 10

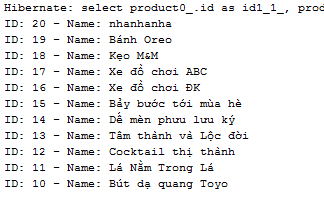


6.5.5 Mệnh đề Order By

Dùng để sắp xếp kết quả truy vấn, DESC là sắp xếp kết quả giảm dần, ASC là sắp xếp kết quả tăng dần

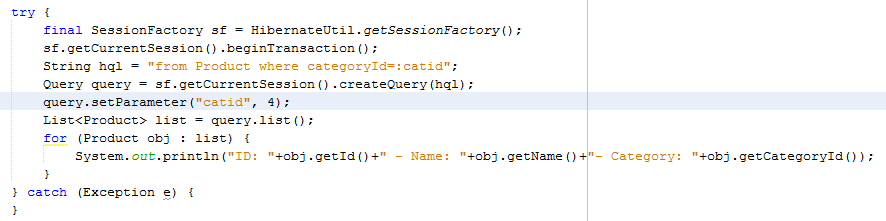


=> sắp xếp các sinh viên theo thứ tự giảm dần và đưa vào trong list.



6.5.6 Sử dụng tên tham số

Hibernate hỗ trợ sữ dụng tên tham số(“:catid”) dùng dấu “:”đứng trước để đặt tên tham số trong câu truy vấn của nó thông qua phương thức setParameter()

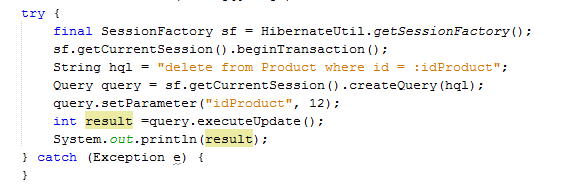


**=> Kết quả:**



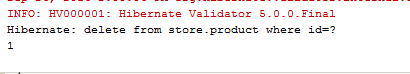
6.5.7 Mệnh đề Delete:

Dùng để xóa 1 hoặc nhiều đối tượng

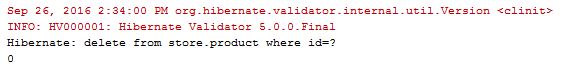


Kết quả trả về là 1 nếu thành công và là 0 nếu thất bại

* Xóa thành công

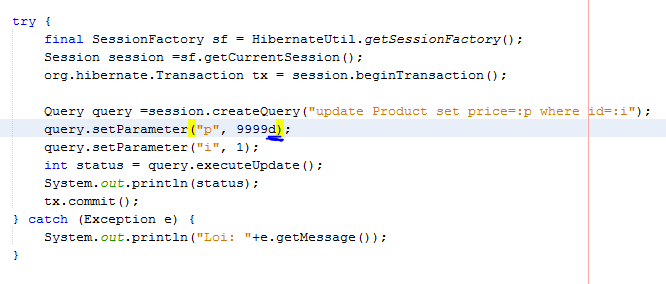


* Xóa thất bại



6.5.8 Mệnh đề Update

Dùng để update một hoặc nhiều thuộc tính của một hoặc nhiều đối tượng



Phải thêm subfix vào giá trị của các parameter nếu đó là kiểu float, double, ..

* Nếu không thì sẽ có thông báo lỗi hiện ra



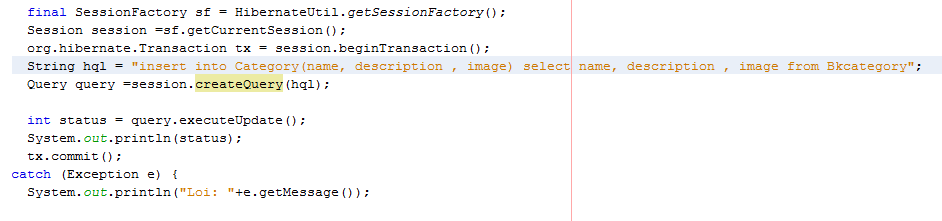
=> Kêt quả là



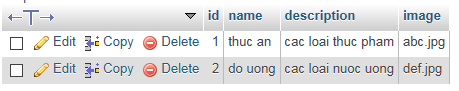


6.5.9 Mệnh đề Insert

HQL không hỗ trợ mệnh đề Insert into table\_name values(...). Nó chỉ hỗ trợ Insert into … Select, tức là insert từ một đối tượng khác, hoặc 1 bảng khác

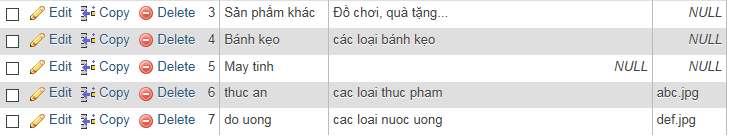


Bảng Bkcategory



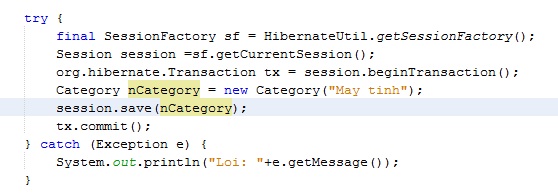
Câu lệnh sẽ chuyển các category từ bảng Bkcategory sang bảng category

=> Kết quả là:



Có thể tạo một object rồi dùng hàm save của Session để thêm vào bảng

Ví dụ tạo 1 object của class POJO Category la nCategory và dùng session.save(nCategory) để lưu category mới vào bảng category



=> Kết quả





6.6 Criteria Queries

Criteria là một interface cho phép thực thi các câu lệnh truy vấn theo hướng đối tượng. Nó hiệu quả hơn HQL trong trường hợp màn hình web tìm kiếm dựa trên các điều kiện khác nhau.

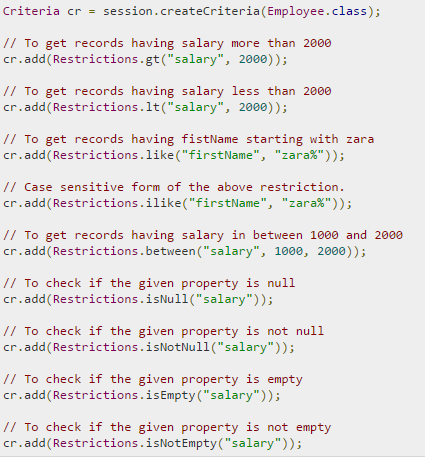
6.6.1 Tạo một Criteria:

s

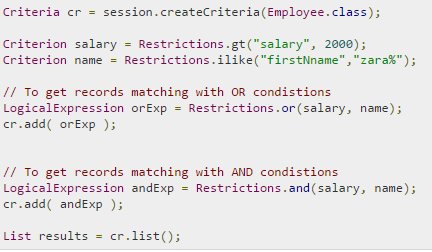
6.6.2 Một số hàm Criteria queries thông dụng

Hàm add()

Hàm add() + một số Restrictions thông dụng

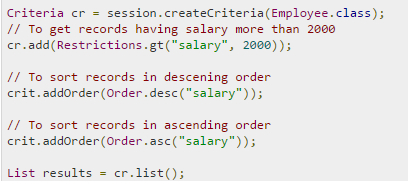


AND or OR conditions



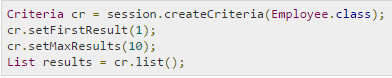
6.6.3. Sắp xếp kết quả tìm kiếm

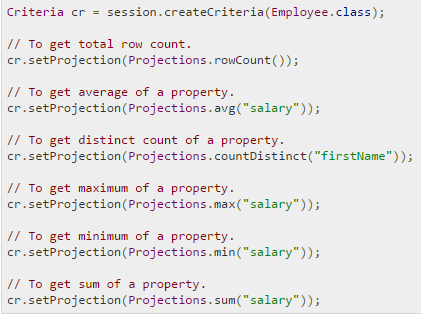
Có thể sắp xếp kêt quả tìm kiếm bằng phương thức Order



6.6.4. Một số hàm đặc biệt

1. public Criteria setFirstResult(int firstResult) lấy ra 1 số dòng đầu tiên (tham số firstResult) từ kết quả truy vấn (bắt đầu từ dòng số 0)
2. public Criteria setMaxResult(int totalResult) lấy ra tổng số dòng được quy định

6.6.5. Một số hàm thống kê:



**HQL vs Criteria queries**

* Criteria queries: là loại truy vấn hướng đối tượng, phù hợp cho các loại truy vấn động. Phù hợp với các truy vấn **Select**, sử dụng số lượng code dài hơn
* HQL: phù hợp cho truy vấn tĩnh và phức tạp. Nó truy vấn mạnh mẽ và dễ sử dụng hơn

**DEMO**

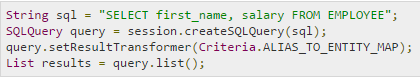
https://drive.google.com/file/d/0B6ogK5HFJ2zqMFU0d2ZUX0VYT2M/view?usp=sharing

6.7 Native SQL

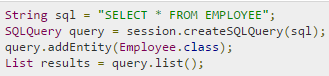
Khởi tạo native sql bằng phương thức createSQLQuery ()



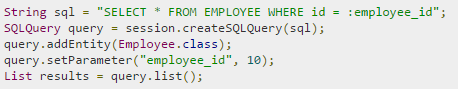
6.7.1. Truy vấn vô hướng



6.7.2. Truy vấn thực thể



6.7.3. Truy vấn SQL có tên



**DEMO**

https://drive.google.com/file/d/0B6ogK5HFJ2zqMFU0d2ZUX0VYT2M/view?usp=sharing

TÀI LIỆU THAM KHẢO

[1] **Hibernate Tutorial,** http://www.tutorialspoint.com/hibernate/

[2] **HIBERNATE - Relational Persistence for Idiomatic Java,**

https://docs.jboss.org/hibernate/orm/3.5/reference/en/html/