**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

***Hibernate Tutorial***

*Hibernate là một dịch vụ truy vấn và lưu trữ Đối tượng – Quan hệ có hiệu suất cao, được cấp phép theo nguồn mở GNU Lesser General Public License (LGPL)**và miễn phí để tải xuống. Hibernate không chỉ quản lí việc ánh xạ từ các Class của Java đến các Table của Database( và từ kiểu dữ liệu Java sang kiểu dữ liệu SQL) mà còn cung cấp các truy vấn và phương tiện truy suất dữ liệu. Hướng dẫn này sẽ chỉ cho bạn các sử dụng Hibernate để phát triển các ứng dụng web dựa trên cơ sở dữ liệu của bạn một cách đơn giản và dễ dàng.*

*Bài hướng dẫn này được thiết kế cho tất cả các lập trình viên Java mà muốn tìm hiểu Hibernate framework và API của nó.*

*Yêu cầu: có sự hiểu biết cơ bản về cơ sở dữ liệu quan hệ, JDBC, SQL sẽ rất hữu ích trong bài hướng dẫn này.*

**MỤC LỤC**

*ORM – Overview \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3*

*Hibernate – Overview \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 8*

*Architecture \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 10*

*Configuration \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 13*

*Sessions \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 18*

*Persistent Class \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 22*

*Mapping Files \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 24*

*Mapping Types \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 27*

*Example \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 30*

*O/R Mappings \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 36*

*Annotations \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 39*

*Query Language \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 45*

*Critetia Queries \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 51*

*Native SQL \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 58*

*Caching \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 64*

*Batch Processing \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 71*

***ORM Overview***

***JDBC là gì ?***

JDBC là viết tắt của Java Database Connectivity. Nó cung cấp một bộ Java API để kế nối đến cơ sở dữ liệ quan hệ từ chương trình Java. Các Java APIs này cho phép chương trình Java thực thi câu lệnh SQL và tương tác với bất kỳ cơ sở dữ liệu tuân thủ SQL.

JDBC cung cấp một kiến trúc linh hoạt để viết một ứng dụng độc lập cơ sở dữ liệu có thể chạy trên các nền tảng khác nhau và tương tác với các DBMS khác nhau mà không cần sửa đổi.

***Ưu và nhược điểm:***

|  |  |
| --- | --- |
| ***Ưu điểm*** | ***Nhược điểm*** |
| - Xử lí CSDL đơn giản và gọn gàng  - Hiệu suất tốt với dữ liệu lớn  - Tốt cho các chương trình nhỏ  - Cú pháp đơn giản, dễ hiểu | ***-*** Phức tạp khi sử dụng trong dự án lớn.  - Chi phí lập trình cao.  - Không đóng gói.  - Khó để thực hiện MVC concept.  - Lệnh truy vấn mỗi DBMS là riêng biệt. |

***Tại sao sử dụng Object Relation Mapping (ORM) ?***

Khi chúng ta làm việc với một hệ thống hướng đối tượng, có một sự không phù hợp giữa mô hình đối tượng và cơ sở dữ liệu quan hệ. Các RDBMS biểu thị dữ liệu theo định dạng bảng trong khi các ngôn ngữ hướng đối tượng, chẳng hạn như Java hoặc C # biểu thị nó dưới dạng biểu đồ liên kết của các đối tượng.

Hãy xem xét các hàm Java sau:

public class Employee {

private int id;

private String first\_name;

private String last\_name;

private int salary;

public Employee() {}

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

this.first\_name = fname;

this.last\_name = lname;

this.salary = salary;

}

public int getId() {

return id;

}

public String getFirstName() {

return first\_name;

}

public String getLastName() {

return last\_name;

}

public int getSalary() {

return salary;

}

}

Hãy xem xét các đối tượng trên sẽ 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)

);

Vấn đề đầu tiên, nếu chúng ta cần sửa đổi thiết kế cơ sở dữ liệu của mình sau khi đã phát triển một vài trang hoặc ứng dụng của chúng ta thì sao? Thứ hai, tải và lưu trữ các đối tượng trong cơ sở dữ liệu quan hệ cho chúng ta thấy năm vấn đề không khớp sau đây.

|  |  |
| --- | --- |
| Sr.No. | Điều không phù hợp và mô tả |
| 1 | ***Độ chi tiết***  Đôi khi bạn sẽ có mô hình đối tượng, có nhiều Class hơn số lượng Table  tương ứng trong CSDL. |
| 2 | ***Kế thừa***  Các RDBMS không định nghĩa bất cứ điều gì tương tự như Kế thừa,  đó là một mô hình tự nhiên trong các ngôn ngữ lập trình hướng đối tượng***.*** |
| 3 | ***Định danh***  Một RDBMS định nghĩa chính xác một khái niệm về 'sự giống nhau':  khóa chính. Tuy nhiên, Java định nghĩa cả danh tính đối tượng (a == b)  và đẳng thức đối tượng (a.equals (b)). |
| 4 | ***Liên kết***  Các ngôn ngữ hướng đối tượng biểu thị các liên kết bằng cách sử dụng các  tham chiếu đối tượng trong khi RDBMS biểu thị một liên kết dưới dạng  khóa ngoài. |
| 5 | ***Cách truy cập***  Các cách bạn truy cập các đối tượng trong Java và trong RDBMS về cơ bản  là khác nhau. |

Object Relation Mapping (ORM) là giải pháp để xử lý tất cả các sự không phù hợp trở ở trên.

***ORM là gì ?***

ORM là viết tắt của **O**bject-**R**elative **M**apping (ORM) là một kỹ thuật lập trình để chuyển đổi dữ liệu giữa cơ sở dữ liệu quan hệ và ngôn ngữ lập trình hướng đối tượng như Java, C #, v.v.

Một hệ thống ORM có những ưu điểm sau so với JDBC:

|  |  |
| --- | --- |
| ***Sr.No*** | ***Ưu điểm*** |
| 1 | Sử dụng code để truy cập đối tượng hơn là các bảng của DB |
| 2 | Ẩn chi tiết các truy vấn SQL |
| 3 | Dựa trên JDBC |
| 4 | Không phải đối phó với sự thay đổi của cơ sở dữ liệu |
| 5 | Các entities dựa trên các business concept hơn là cấu trúc cơ sở  dữ liệu. |
| 6 | Quản lí transaction và tạo key tự động |
| 7 | Phát triển ứng dụng nhanh |

Một giải pháp ORM bao gồm bốn thực thể sau:

|  |  |
| --- | --- |
| ***Sr.No*** | ***Giải pháp*** |
| 1 | Một API để thực hiện các hoạt động CRUD cơ bản trên các đối tượng của các lớp. |
| 2 | Một ngôn ngữ hoặc API để chỉ định các truy vấn tham chiếu đến các lớp và thuộc tính của các lớp. |
| 3 | Một cơ sở cấu hình cho siêu dữ liệu ánh xạ. |
| 4 | Một kỹ thuật để tương tác với các đối tượng transaction để thực  hiện dirty checking, lazy association fetching và các chức năng  tối ưu hóa khác. |

***Java ORM Frameworks***

Có một số framework và các tùy chọn ORM trong Java.A persistent framework là một dịch vụ ORM lưu trữ và truy xuất các đối tượng vào cơ sở dữ liệu quan hệ.

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

***Hibernate – Overview***

Hibernate là một giải pháp Object-Relation Mapping (ORM) cho JAVA. Nó là một framework tác bền vững mã nguồn mở được tạo ra bởi Gavin King vào năm 2001. Đây là một dịch vụ truy vấn và duy trì quan hệ đối tượng mạnh mẽ, hiệu suất cao cho bất kỳ ứng dụng Java nào.

Hibernate ánh xạ các lớp Java vào các bảng cơ sở dữ liệu và từ các kiểu dữ liệu Java thành các kiểu dữ liệu SQL và giải phóng nhà phát triển khỏi 95% các tác vụ lập trình liên quan đến dữ liệu phổ biến.

Hibernate nằm giữa các đối tượng Java truyền thống và máy chủ cơ sở dữ liệu để xử lý tất cả các công việc trong việc duy trì các đối tượng đó dựa trên các cơ chế và mẫu O / R thích hợp.



***Ưu điểm của Hibernate***

* Hibernate ánh xạ các lớp Java vào các bảng cơ sở dữ liệu bằng các tệp XML và không cần viết bất kỳ dòng mã nào.
* Cung cấp các API đơn giản để lưu trữ và truy xuất các đối tượng Java trực tiếp đến và từ cơ sở dữ liệu.
* Nếu có sự thay đổi trong cơ sở dữ liệu hoặc trong bất kỳ bảng nào, thì bạn chỉ cần thay đổi các thuộc tính tệp XML.
* Tóm tắt các loại SQL không quen thuộc và cung cấp một cách để làm việc xung quanh các Đối tượng Java quen thuộc.
* Hibernate không yêu cầu máy chủ ứng dụng hoạt động.
* Điều khiển sự lien kết phức tạp của các đối tượng của cơ sở dữ liệu.
* Giảm thiểu việc truy cập cơ sở dữ liệu với các cách tìm nạp thông minh.
* Cung cấp truy vấn dữ liệu đơn giản.

***Cơ sở dữ liệu hỗ trợ:***

Hibernate hỗ trợ hầu hết tất cả các RDBMS chính. Sau đây là danh sách:

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

***Công nghệ hỗ trợ:***

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

# *Hibernate - Architecture*

Hibernate có kiến ​​trúc phân lớp giúp người dùng hoạt động mà không cần phải biết các API cơ bản. Hibernate sử dụng cơ sở dữ liệu và dữ liệu cấu hình để cung cấp các dịch vụ bền vững (và các đối tượng liên tục) cho ứng dụng.



Sau đây là một cái nhìn chi tiết về Kiến trúc ứng dụng Hibernate với các lớp cốt lõi quan trọng của nó.



Hibernate sử dụng nhiều Java API, như JDBC, Java Transaction API (JTA) và Java Naming and Directory Interface (JNDI). JDBC cung cấp một mức độ trừu tượng thô sơ của chức năng phổ biến đối với cơ sở dữ liệu quan hệ, cho phép hầu hết mọi cơ sở dữ liệu có trình điều khiển JDBC được Hibernate hỗ trợ. JNDI và JTA cho phép Hibernate được tích hợp với các máy chủ ứng dụng J2EE.

Phần sau đây mô tả ngắn gọn về từng đối tượng lớp có liên quan đến Kiến trúc ứng dụng Hibernate.

## *Configuration Object*

Configuration Object là đối tượng Hibernate đầu tiên bạn tạo trong bất kỳ ứng dụng Hibernate nào. Nó thường được tạo chỉ một lần trong quá trình khởi tạo ứng dụng. Nó đại diện cho một tệp cấu hình hoặc thuộc tính được yêu cầu bởi Hibernate.

Đối tượng Cấu hình cung cấp hai thành phần chính -

* **Kết nối cơ sở dữ liệu** - Điều này được xử lý thông qua một hoặc nhiều tệp cấu hình được Hibernate hỗ trợ. Các tệp này là **hibernate.properies** và **hibernate.cfg.xml** .
* **Thiết lập ánh xạ lớp** - Thành phần này tạo kết nối giữa các lớp Java và các bảng cơ sở dữ liệu.

## *Đối tượng SessionFactory*

Configuration Obj được sử dụng để tạo đối tượng SessionFactory, lần lượt cấu hình Hibernate cho ứng dụng bằng tệp cấu hình được cung cấp và cho phép đối tượng Session được khởi tạo. SessionFactory là một đối tượng an toàn của luồng và được sử dụng bởi tất cả các luồng của ứng dụng.

SessionFactory là một đối tượng nặng, nó thường được tạo trong quá trình khởi động ứng dụng và giữ lại để sử dụng. Bạn sẽ cần một đối tượng SessionFactory cho mỗi cơ sở dữ liệu bằng cách sử dụng tệp cấu hình riêng biệt. Vì vậy, nếu bạn đang sử dụng nhiều cơ sở dữ liệu, thì bạn sẽ phải tạo nhiều đối tượng SessionFactory.

## *Đối tượng Session*

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

Các đối tượng Session không nên được giữ mở trong một thời gian dài vì chúng thường không an toàn cho luồng và chúng nên được tạo và hủy chúng khi cần.

## *Đối tượng Transaction*

Transaction đại diện cho một đơn vị công việc với cơ sở dữ liệu và hầu hết RDBMS hỗ trợ chức năng giao dịch. Các giao dịch trong Hibernate được xử lý bởi người quản lý giao dịch và giao dịch cơ bản (từ JDBC hoặc JTA).

Đây là một đối tượng tùy chọn và các ứng dụng Hibernate có thể chọn không sử dụng giao diện này, thay vào đó quản lý các giao dịch trong mã ứng dụng của riêng họ.

## *Đối tượng Query*

Các đối tượng truy vấn sử dụng chuỗi SQL hoặc Hibernate Query Language (HQL) để lấy dữ liệu từ cơ sở dữ liệu và tạo các đối tượng. Một phiên bản truy vấn được sử dụng để liên kết các tham số truy vấn, giới hạn kết quả mà truy vấn trả về và cuối cùng để thực hiện truy vấn.

## *Đối tượng Criteria*

Các đối tượng Criteria được sử dụng để tạo và thực hiện các truy vấn tiêu chí hướng đối tượng để truy xuất đối tượng.

***Hibernate – Configuration***

Hibernate yêu cầu phải biết nơi để ánh xạ các thông tin mà định nghĩa Class có lien quan đến các bảng trong CSDL. Hibernate cũng yêu cầu bộ cài đặt cấu hình lien quan đến database và các tham số khác. Tất cả thông tin này thường được cung cấp ở file thuộc tính Java tên **hibernate.properties** hoặc file XML **hibernate.cfg.xml**

Tôi sử dụng XML **hibernate.cfg.xml** để chỉ định các thuộc tính Hibernate được yêu cầu trong các ví dụ. Hầu hết các thuộc tính lấy các giá trị mặc định của chúng và không bắt buộc phải chỉ định chúng trong tệp thuộc tính trừ khi thực sự cần thiết.

## *Hibernate Properties*

|  |  |
| --- | --- |
| *Sr.No.* | *Properties & Desceptions* |
| 1 | **hibernate.dialect** Thuộc tính này khiến Hibernate tạo các SQL thích hợp cho CSDL đã chọn |
| 2 | **hibernate.connection.driver\_class** JDBC driver class |
| 3 | **hibernate.connection.url** URL để JDBC kết nối với CSDL |
| 4 | **hibernate.connection.username** Username của DB |
| 5 | hibernate.connection.passwordPassword của DB |
| 6 | **hibernate.connection.pool\_size**  Giới hạn số lượng kết nối đang chờ trong nhóm kết nối cơ sở dữ liệu Hibernate. |
| 7 | **hibernate.connection.autocommit**  Cho phép chế độ autocommit được sử dụng cho kết nối JDBC. |

Nếu bạn đang sử dụng cơ sở dữ liệu cùng với máy chủ ứng dụng và JNDI, thì bạn sẽ phải định cấu hình các thuộc tính sau

|  |  |
| --- | --- |
| ***Sr.No*** | *Properties & Desceptions* |
| 1 | **hibernate.connection.datasource**  Tên JNDI định nghĩa trong application server context mà bạn sử dụng cho ứng dụng |
| 2 | **hibernate.jndi. class**  Lớp InitialContext cho JNDI. |
| 3 | **hibernate.jndi. <JNDIpropertyname>**  Chuyển bất kỳ thuộc tính JNDI nào bạn muốn cho JNDI *InitialContext* . |
| 4 | **hibernate.jndi.url**  Cung cấp URL cho JNDI. |
| 5 | **hibernate.connection.username**  Tên người dùng cơ sở dữ liệu. |
| 6 | **ngủ đông.connection.password**  Mật khẩu cơ sở dữ liệu. |

## *Hibernate với CSDL MYSQL*

Tệp cấu hình XML phải tuân theo DTD Cấu hình Hibernate 3, có sẵn tại <http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd>

<?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>

<!-- Assume test is the database name -->

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

jdbc:mysql://localhost/test

</property>

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

root

</property>

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

root123

</property>

<!-- List of XML mapping files -->

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

</session-factory>

</hibernate-configuration>

## *Tệp cấu hình trên có bao gồm thẻ <mapping>, thẻ này lien quan đến việc ánh xạ file hibernate và ta sẽ xem cách sử dụng ở các chapter sau.*

## *Dưới đây là 1 list các DB dialect như sau:*

|  |  |
| --- | --- |
| **Sr.No.** | **Database & Dialect Property** |
| 1 | **DB2**  org.hibernate.dialect.DB2Dialect |
| 2 | **HSQLDB**  org.hibernate.dialect.HSQLDialect |
| 3 | **HypersonicSQL**  org.hibernate.dialect.HSQLDialect |
| 4 | **Informix**  org.hibernate.dialect.InformixDialect |
| 5 | **Ingres**  org.hibernate.dialect.IngresDialect |
| 6 | **Interbase**  org.hibernate.dialect.InterbaseDialect |
| 7 | **Microsoft SQL Server 2000**  org.hibernate.dialect.SQLServerDialect |
| 8 | **Microsoft SQL Server 2005**  org.hibernate.dialect.SQLServer2005Dialect |
| 9 | **Microsoft SQL Server 2008**  org.hibernate.dialect.SQLServer2008Dialect |
| 10 | **MySQL**  org.hibernate.dialect.MySQLDialect |
| 11 | **Oracle (any version)**  org.hibernate.dialect.OracleDialect |
| 12 | **Oracle 11g**  org.hibernate.dialect.Oracle10gDialect |
| 13 | **Oracle 10g**  org.hibernate.dialect.Oracle10gDialect |
| 14 | **Oracle 9i**  org.hibernate.dialect.Oracle9iDialect |
| 15 | **PostgreSQL**  org.hibernate.dialect.PostgreSQLDialect |
| 16 | **Progress**  org.hibernate.dialect.ProgressDialect |
| 17 | **SAP DB**  org.hibernate.dialect.SAPDBDialect |
| 18 | **Sybase**  org.hibernate.dialect.SybaseDialect |
| 19 | **Sybase Anywhere**  org.hibernate.dialect.SybaseAnywhereDialect |

## *Hibernate - Session*

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

Các đối tượng Session không nên được giữ mở trong một thời gian dài vì chúng thường không an toàn cho luồng và chúng nên được tạo và hủy chúng khi cần. Chức năng chính của Session là cung cấp tạo, đọc và xóa các hoạt động cho các phiên bản của các Class Entities được ánh xạ.

Trường hợp có thể tồn tại ở một trong ba trạng thái sau tại một thời điểm nhất định

* **transient** – Một phiên bản mới của lớp mà không lien quan tới Session và không có thể hiện ở DB và không có giá trị định danh được coi là **tạm thời(transient)**.
* **persistent** – Bạn có thể tạo một phiên bản **tạm thời(transient)** bằng cách lien kết nói với Session. Một phiên bản lien tục(persistent instance) có thể hiện trong DB, có giá trị, định danh được kết nối với Session.
* **detached** – Một khi ta đóng Session, **persistent** sẽ trở thành **detached** .

Một phiên bản Session là tuần tự hóa nếu lớp lien tục của nó là tuần tự hóa. Một Transaction điển hình được dùng như thành ngữ sau:

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 ngoại lệ, transaction được rollback và session phải bị loại bỏ

## *Session Interface Methods*

## Có một số phương thức được cung cấp bởi Session Interface, nhưng tôi sẽ chỉ liệt kê một vài phương thức quan trọng mà ta sẽ sử dụng trong hướng dẫn này. Bạn có thể xem tài liệu Hibernate cho một danh sách đầy đủ các phương pháp liên quan đến Session và SessionFactory .

|  |  |
| --- | --- |
| **Sr.Không.** | **Phương pháp và mô tả phiên** |
| 1 | **Transaction beginTransaction()**  Bắt đầu một đơn vị công việc và trả về đối tượng Transaction liên quan. |
| 2 | **void cancelQuery ()**  Hủy bỏ việc thực hiện truy vấn hiện tại. |
| 3 | **void clear()**  Hoàn toàn xóa session. |
| 4 | **Connection close()**  Kết thúc session bằng cách giải phóng kết nối JDBC và dọn dẹp. |
| 5 | **Criteria createCriteria(Class persistentClass)**  Tạo một cá thể Criteria mới, cho class thực thể đã cho hoặc superclass của lớp thực thể. |
| 6 | **Criteria createCriteria(String entityName)**  Tạo một thể hiện Criteria mới, cho tên thực thể đã cho. |
| 7 | **Serializable getIdentifier(Object object)**  Trả về giá trị định danh của thực thể đã cho như được liên kết với session |
| 8 | **Query createFilter(Object collection, String queryString)**  Tạo một phiên bản mới của Truy vấn cho bộ sưu tập và chuỗi bộ lọc đã cho. |
| 9 | **Query createQuery(String queryString)**  Tạo một phiên bản mới của Truy vấn cho chuỗi truy vấn HQL đã cho. |
| 10 | **SQLQuery createSQLQuery(String queryString)**  Tạo một phiên bản mới của SQLQuery cho chuỗi truy vấn SQL đã cho. |
| 11 | **void delete(Object object)**  Loại bỏ một cá thể liên tục khỏi kho dữ liệu. |
| 12 | **void delete(String entityName, Object object)**  Loại bỏ một cá thể liên tục khỏi kho dữ liệu. |
| 13 | **Session get(String entityName, Serializable id)**  Trả về thể hiện liên tục của thực thể được đặt tên với định danh đã cho hoặc null nếu không có thể hiện liên tục như vậy. |
| 14 | **SessionFactory getSessionFactory()**  Lấy sessionFactory được tạo bởi phiên này |
| 15 | **void refresh (Đối tượng)**  Đọc lại trạng thái của thể hiện đã cho từ cơ sở dữ liệu cơ bản. |
| 16 | **Transaction getTransaction()**  Lấy Transaction liên quan đến phiên này. |
| 17 | **boolean isConnected ()**  Kiểm tra nếu phiên hiện đang được kết nối. |
| 18 | **boolean isDirty ()**  Phiên này có chứa bất kỳ thay đổi nào phải được đồng bộ hóa với cơ sở dữ liệu không? |
| 19 | **boolean isOpen ()**  Kiểm tra nếu phiên vẫn mở. |
| 20 | **Serializable save(Object object)**  Lưu transient instance đã cho, trước tiên chỉ định một định danh được tạo. |
| 21 | **void saveOrUpdate (Object object)**  Lưu (Đối tượng) hoặc cập nhật (Đối tượng) đối tượng đã cho. |
| 22 | **void update (Object object)**  Cập nhật persistent instance với định danh của detached instance đã cho. |
| 23 | **void update (String entityName, Object object)**  Cập nhật persistent instance với định danh của detached instance đã cho. |

.

***Persistant Class***

Toàn bộ khái niệm của Hibernate là lấy các giá trị từ các thuộc tính lớp Java và thêm chúng vào bảng cơ sở dữ liệu. Một tài liệu ánh xạ giúp Hibernate xác định cách đưa các giá trị từ các lớp và ánh xạ chúng với bảng và các trường liên quan.

Các lớp Java có các đối tượng hoặc cá thể sẽ được lưu trữ trong các bảng cơ sở dữ liệu được gọi là các persistant class trong Hibernate. Hibernate hoạt động tốt nhất nếu các lớp này tuân theo một số quy tắc đơn giản, còn được gọi là mô hình lập trình **Plain Old Object** (POJO).

Có các quy tắc chính của các lớp persistant class

* Tất cả các lớp Java mà được duy trì cần một contructor mặc định.
* Tất cả các lớp nên chứa 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 khóa chính của bảng cơ sở dữ liệu.
* Tất cả các thuộc tính sẽ được duy trì phải được khai báo là private và có các phương thức **getXXX** và **setXXX** được xác định theo kiểu JavaBean.
* Một tính năng trung tâm của Hibernate, proxy, phụ thuộc vào persistant class là không phải là final hoặc implement một interface được khai báo tất cả các public method.
* Tất cả các lớp không mở rộng hoặc triển khai một số class và interface chuyên biệt theo yêu cầu của khung EJB.

Tên POJO được sử dụng để nhấn mạnh rằng một đối tượng đã cho là một Java Object, không phải là một đối tượng đặc biệt và đặc biệt không phải là JavaBean Enterprise.

## *Simple POJO Example*

Dựa trên một vài quy tắc được đề cập ở trên, chúng ta có thể định nghĩa một lớp POJO như sau

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;

}

}

## *Mapping Files*

Ánh xạ đối tượng / quan hệ thường được xác định trong tài liệu XML. Tệp ánh xạ này hướng dẫn Hibernate - làm thế nào để ánh xạ lớp hoặc các lớp được xác định vào các bảng cơ sở dữ liệu?

Nhiều người dùng Hibernate chọn viết XML bằng tay, nhưng một số công cụ tồn tại để tạo tài liệu ánh xạ. Bao gồm **XDoclet, Middlegen** và **AndroMDA** cho người dùng Hibernate tiên tiến.

Chúng ta hãy xem xét lớp POJO được xác định trước đó của chúng ta có các đối tượng sẽ tồn tại trong bảng được định nghĩa trong phần tiếp theo.

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;

}

}

Lớp trên sẽ tương ứng với một bảng mà bạn đã cung cấp.Đối tượng trên cần được lưu trữ và truy suất 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)

);

Dựa trên hai thực thể trên, chúng ta có thể định nghĩa tệp ánh xạ sau, hướng dẫn Hibernate cách ánh xạ lớp hoặc lớp được xác định 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">

This class contains the employee detail.

</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>

## Bạn nên lưu tài liệu ánh xạ dưới dạng <classname>.hbm.xml. Trong trường hợp này ta lưu nó trong tệp Employee.hbm.xml.

## Chi tiết về thành phần ánh xaạ trong file ánh xạ

* Tài liệu ánh xạ là một tài liệu XML có <hibernate-maps> làm phần tử gốc, chứa tất cả các phần tử <class> .
* Các phần tử <class> được sử dụng để xác định ánh xạ cụ thể từ các lớp Java đến các bảng cơ sở dữ liệu. Tên lớp Java được chỉ định bằng cách sử dụng thuộc tính *name* của thành phần lớp và tên bảng cơ sở dữ liệu được chỉ định bằng thuộc tính *table*.
* Phần tử <meta> là phần tử tùy chọn và có thể được sử dụng để tạo mô tả lớp.
* Phần tử <id> ánh xạ thuộc tính ID duy nhất trong lớp vào khóa chính của bảng cơ sở dữ liệu. Các tên thuộc tính của phần tử id đề cập đến tài sản trong lớp và các cột thuộc tính liên quan đến cột trong bảng cơ sở dữ liệu. Các loại thuộc tính nắm giữ các loại bản đồ ngủ đông, điều này các loại bản đồ sẽ chuyển đổi từ Java để kiểu dữ liệu SQL.
* Phần tử <Generator> trong phần tử id được sử dụng để tự động tạo các giá trị khóa chính. Các lớp thuộc tính của phần tử generate được thiết lập để tự nhiên để cho Hibernate sử dụng thuật toán Hilo, identity, sequence để tạo khóa chính phụ thuộc vào khả năng của cơ sở dữ liệu cơ bản.
* Phần tử <property> được sử dụng để ánh xạ một thuộc tính lớp Java thành một cột trong bảng cơ sở dữ liệu. Các tên thuộc tính này liên quan đến lớp và các cột thuộc tính liên quan đến cột trong bảng cơ sở dữ liệu. Các loại thuộc tính nắm giữ các kiểu mapping của Hibernate, kiểu mapping sẽ chuyển đổi từ Java để kiểu dữ liệu SQL.

Còn có nhiều thuộc tính và yếu tố có sẵn khác, mà sử dụng mapping doc và tôi cố gắng bao quát nhât có thể về các chủ đề của Hibernate khác.

***Mapping Types***

Để mapping, bạn phải biết cách map các kiểu dữ liệu Java sang kiểu dữ liệu của RDBMS. ***Kiểu dữ liệu*** được định nghĩa và sử dụng trong file mapping không phải kiểu dữ liệu của Java cũng như RDBMS. Nó được gọi là ***Hibernate mapping types*** – có thể dịch các kiểu dữ liệu Java sang SQL và ngược lại.

Chương này liệt kê tất cả các kiểu dữ liệu nguyên thủy, ngày và giờ, đối tượng lớn và các loại ánh xạ dựng sẵn khác.

***Kiểu dữ liệu nguyên thủy***

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

## *Kiểu dữ liệu Date and Time*

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

## *Kiểu dữ liệu Binary and Large Object*

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

## *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 |

***Example***

Bây giờ chúng ta hãy lấy một ví dụ để hiểu cách chúng ta có thể sử dụng Hibernate để cung cấp tính bền vững của Java trong một ứng dụng độc lập. Chúng tôi sẽ thực hiện các bước khác nhau liên quan đến việc tạo Ứng dụng Java bằng công nghệ Hibernate.

***Tạo lớp POJO***

Bước đầu tiên trong việc tạo một ứng dụng là xây dựng lớp hoặc lớp Java POJO. Chúng ta hãy xem xét lớp **Employee** với các phương thức **getXXX** và **setXXX** để biến nó thành lớp tuân thủ JavaBeans.

POJO (Đối tượng Java cũ thuần túy) là một đối tượng Java không mở rộng hoặc triển khai một số lớp và giao diện chuyên biệt tương ứng theo yêu cầu của khung EJB. Tất cả các đối tượng Java bình thường là POJO.

Khi bạn thiết kế một lớp persisted của Hibernate, điều quan trọng là tuân thủ JavaBeans cũng như một thuộc tính, sẽ hoạt động như chỉ mục như thuộc tính **id** trong lớp Employee.

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 Cơ sở dữ liệu***

Bước thứ hai sẽ tạo các bảng trong cơ sở dữ liệu của bạn tương ứng.

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 tập tin cấu hình ánh xạ*

Bước này là tạo một tệp ánh xạ hướng dẫn Hibernate cách ánh xạ lớp hoặc các lớp đã xác định 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">

This class contains the employee detail.

</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 lớp ứng dụng*

Cuối cùng, ta sẽ tạo lớp ứng dụng của mình với phương thức main () để chạy ứng dụng. Chúng tôi sẽ sử dụng ứng dụng này để lưu vài Employee và sau đó chúng tôi sẽ áp dụng các thao tác CRUD trên các bản ghi đó.

import java.util.List;

import java.util.Date;

import java.util.Iterator;

import org.hibernate.HibernateException;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

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();

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();

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();

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();

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();

}

}

}

Dưới đây là các bước để biên dịch và chạy ứng dụng đã đề cập ở trên. Hãy chắc chắn rằng, bạn đã thiết lập PATH và CLASSPATH một cách thích hợp trước khi tiến hành biên dịch và thực thi.

* Tạo tệp cấu hình hibernate.cfg.xml như được giải thích trong chương cấu hình.
* Tạo tệp ánh xạ Employee.hbm.xml như được hiển thị ở trên.
* Tạo tệp nguồn Employee.java như được hiển thị ở trên và biên dịch nó.
* Tạo tệp nguồn ManageEmployee.java như được hiển thị ở trên và biên dịch nó.
* Thực thi main của ManageEmployee để chạy chương trình.

Bạn sẽ nhận được kết quả sau và các bản ghi sẽ được tạo trong bảng NHÂN VIÊN.

$java ManageEmployee

.......VARIOUS LOG MESSAGES WILL DISPLAY HERE........

First Name: Zara Last Name: Ali Salary: 1000

First Name: Daisy Last Name: Das Salary: 5000

First Name: John Last Name: Paul Salary: 10000

First Name: Zara Last Name: Ali Salary: 5000

First Name: John Last Name: Paul Salary: 10000

***O/R Mapping***

Cho đến nay, chúng ta đã thấy ánh xạ O / R rất cơ bản bằng cách sử dụng chế độ ngủ đông, nhưng có ba chủ đề ánh xạ quan trọng nhất mà chúng ta phải tìm hiểu chi tiết.

## Collections Mappings

## Association Mappings

## Component Mappings

## *Collections Mappings*

Nếu một thực thể hoặc lớp có tập hợp các giá trị cho biến cụ thể, thì chúng ta có thể ánh xạ các giá trị đó bằng cách sử dụng bất kỳ một interface collection có sẵn trong java.

|  |  |
| --- | --- |
| **Sr.No.** | **Collection type & Mapping Description** |
| 1 | [java.util.Set](https://www.tutorialspoint.com/hibernate/hibernate_set_mapping.htm)  This is mapped with a <set> element and initialized with java.util.HashSet  <https://www.tutorialspoint.com/hibernate/hibernate_set_mapping.htm> |
| 2 | [java.util.SortedSet](https://www.tutorialspoint.com/hibernate/hibernate_sortedset_mapping.htm)  This is mapped with a <set> element and initialized with java.util.TreeSet. The **sort** attribute can be set to either a comparator or natural ordering.  <https://www.tutorialspoint.com/hibernate/hibernate_sortedset_mapping.htm> |
| 3 | [java.util.List](https://www.tutorialspoint.com/hibernate/hibernate_list_mapping.htm)  This is mapped with a <list> element and initialized with java.util.ArrayList  <https://www.tutorialspoint.com/hibernate/hibernate_list_mapping.htm> |
| 4 | [java.util.Collection](https://www.tutorialspoint.com/hibernate/hibernate_bag_mapping.htm)  This is mapped with a <bag> or <ibag> element and initialized with java.util.ArrayList  <https://www.tutorialspoint.com/hibernate/hibernate_bag_mapping.htm> |
| 5 | [java.util.Map](https://www.tutorialspoint.com/hibernate/hibernate_map_mapping.htm)  This is mapped with a <map> element and initialized with java.util.HashMap  <https://www.tutorialspoint.com/hibernate/hibernate_map_mapping.htm> |
| 6 | [java.util.SortedMap](https://www.tutorialspoint.com/hibernate/hibernate_sortedmap_mapping.htm)  This is mapped with a <map> element and initialized with java.util.TreeMap. The **sort** attribute can be set to either a comparator or natural ordering.  <https://www.tutorialspoint.com/hibernate/hibernate_sortedmap_mapping.htm> |

Mảng được Hibernate hỗ trợ với <primitive-array> cho các loại giá trị nguyên thủy Java và <array> cho mọi thứ khác. Tuy nhiên, chúng hiếm khi được sử dụng, vì vậy tôi sẽ không nói về chúng trong hướng dẫn này.

Nếu bạn muốn ánh xạ interface collection do người dùng định nghĩa, không được Hibernate hỗ trợ trực tiếp, bạn cần nói với Hibernate về ngữ nghĩa của các collection tùy chỉnh của bạn, điều này không dễ dàng và không nên sử dụng.

## *Association Mappings*

Ánh xạ liên kết giữa các lớp và các mối quan hệ giữa các bảng là linh hồn của ORM. Sau đây là bốn cách mà chính xác của mối quan hệ giữa các đối tượng có thể được thể hiện. Một ánh xạ liên kết có thể là một chiều cũng như hai chiều.

|  |  |
| --- | --- |
| **Sr.No.** | **Mapping type & Description** |
| 1 | [Many-to-One](https://www.tutorialspoint.com/hibernate/hibernate_many_to_one_mapping.htm)  Mapping many-to-one relationship using Hibernate  <https://www.tutorialspoint.com/hibernate/hibernate_many_to_one_mapping.htm> |
| 2 | [One-to-One](https://www.tutorialspoint.com/hibernate/hibernate_one_to_one_mapping.htm)  Mapping one-to-one relationship using Hibernate  <https://www.tutorialspoint.com/hibernate/hibernate_one_to_one_mapping.htm> |
| 3 | [One-to-Many](https://www.tutorialspoint.com/hibernate/hibernate_one_to_many_mapping.htm)  Mapping one-to-many relationship using Hibernate  <https://www.tutorialspoint.com/hibernate/hibernate_one_to_many_mapping.htm> |
| 4 | [Many-to-Many](https://www.tutorialspoint.com/hibernate/hibernate_many_to_many_mapping.htm)  Mapping many-to-many relationship using Hibernate  <https://www.tutorialspoint.com/hibernate/hibernate_many_to_many_mapping.htm> |

## *Component Mappings*

Rất có khả năng một lớp Entity có thể có một tham chiếu đến một lớp khác như một biến thành viên. Nếu lớp được giới thiệu không có vòng đời riêng và hoàn toàn phụ thuộc vào vòng đời của lớp thực thể sở hữu, thì lớp được gọi do đó được gọi là Component Class.

Ánh xạ của Collection of Component cũng có thể theo cách tương tự như ánh xạ của Collection thông thường với sự khác biệt về cấu hình nhỏ. Chúng ta sẽ thấy hai ánh xạ này một cách chi tiết với các ví dụ.

|  |  |
| --- | --- |
| **Sr.No.** | **Mapping type & Description** |
| 1 | [Component Mappings](https://www.tutorialspoint.com/hibernate/hibernate_component_mappings.htm)  Mapping for a class having a reference to another class as a member variable  <https://www.tutorialspoint.com/hibernate/hibernate_component_mappings.htm>. |

***ANNOTATION***

Cho đến nay bạn thường sử dụng tệp ánh xạ XML để chuyển đổi dữ liệu từ POJO sang các bảng cơ sở dữ liệu và ngược lại. Annotation là cách mới nhất để xác định ánh xạ mà không cần sử dụng tệp XML. Bạn có thể sử dụng các chú thích ngoài hoặc thay thế siêu dữ liệu ánh xạ XML.

***Thiết lập môi trường sử dụng Annotation Hibernate***

Trước hết bạn sẽ phải đảm bảo rằng bạn đang sử dụng JDK 5.0 nếu không bạn cần nâng cấp JDK của mình lên JDK 5.0 để tận dụng sự hỗ trợ riêng cho các chú thích.

Thứ hai, bạn sẽ cần cài đặt gói phân phối chú thích Hibernate 3.x.

## *Annotated Class Example*

## Tạo bảng Employee để lưu trữ

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)

);

## Sau đây là ánh xạ của lớp Employee với các Annotation để ánh xạ các đối tượng với bảng Employee được xác định

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() {}

// getter and setter

## *@Entity*

Ta sử dụng **@Entity** cho lớp Nhân viên, đánh dấu lớp này là một entity, do đó nó phải có ít nhất contructor mặc đinh với phạm vi protected là ít nhất.

## *@Table*

Chú thích @Table cho phép bạn chỉ bảng sẽ được sử dụng để map trong cơ sở dữ liệu.

Chú thích @Table cung cấp bốn thuộc tính, cho phép bạn ghi đè tên của bảng, là catalogue, schema, và lược đồ của nó và thực thi các ràng buộc duy nhất trên các cột trong bảng. Hiện tại, chúng tôi đang sử dụng chỉ tên bảng, đó là EMPLOYEE.

## *@Id và @GeneratedValue*

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

Theo mặc định, chú thích @Id sẽ tự tạo khóa chính phù hợp nhất sẽ được sử dụng và bạn có thể thêm **@GeneratedValue** , trong đó có hai tham số **strategy**  và **generator**  mà tôi sẽ không thảo luận ở đây. Để Hibernate xác định loại trình tạo nào sẽ sử dụng làm cho code của bạn có thể tương thích giữa các cơ sở dữ liệu khác nhau.

## *@Column*

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

* **name** cho phép tên của cột trong db được chỉ định rõ ràng.
* **length** cho phép kích thước của cột được sử dụng để ánh xạ một giá trị đặc biệt cho giá trị String.
* **nullable** cho phép cột có thể KHÔNG NULL khi bản ghi được tạo.
* **unique** cho phép cột được đánh dấu là chỉ chứa các giá trị duy nhất.

## *Có thểm tham khảo thêm tại*

## <https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#annotations>

## *Create Application Class*

import java.util.List;

import java.util.Date;

import java.util.Iterator;

import org.hibernate.HibernateException;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.hibernate.cfg.AnnotationConfiguration;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

public class ManageEmployee {

private static SessionFactory factory;

public static void main(String[] args) {

try {

factory = new AnnotationConfiguration().

configure().

//addPackage("com.xyz") //add package if used.

addAnnotatedClass(Employee.class).

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();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

Employee employee = new Employee();

employee.setFirstName(fname);

employee.setLastName(lname);

employee.setSalary(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();

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();

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();

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();

}

}

}

## *Database Configuration*

Tạo **hibernate.cfg.xml** để xác định các tham số liên quan đến cơ sở dữ liệu.

<?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>

<!-- Assume students is the database name -->

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

jdbc:mysql://localhost/test

</property>

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

root

</property>

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

cohondob

</property>

</session-factory>

</hibernate-configuration>

## *Hibernate Query Language*

## Hibernate Query Language - HQL là ngôn ngữ truy vấn hướng đối tượng, tương tự như SQL nhưng HQL hoạt động trên các đối tượng từ đó thực hiện tương tác lên CSDL.

## Mặc dù có thể sử dụng SQL trực tiếp với Hibernate nhưng tôi khuyên bạn nên sử dụng HQL để tránh rắc rối về tính di động của cơ sở dữ liệu và tận dụng sự lưu trữ và bộ đệm của nó.Các từ khóa SELECT, FROM, WHERE,… không phân biệt chữ hoa thường nhưng thuộc tính như tên bảng và cột thì có.

## *FROM CLAUSE*

## Bạn sẽ sử dụng câu lệnh FROM nếu muốn load toàn bộ đối tượng vào bộ nhớ. Dưới đây là cú pháp của nó:

String hql = "FROM Employee";

Query query = session.createQuery(hql);

List results = query.list();

## Nếu muốn sử dụng toàn bộ tên lớp trong HQL, chỉ cần chỉ định gói và tên lớp như sau:

String hql = "FROM com.hibernatebook.criteria.Employee";

Query query = session.createQuery(hql);

List results = query.list();

## *AS CLAUSE*

## Sử dụng AS để gán biệt danh cho một Class trong HQL query, đặc biệt quan trọng nếu bạn có một câu truy vấn dài, cú pháp của nó như sau:

String hql = "FROM Employee AS E";

Query query = session.createQuery(hql);

List results = query.list();

## As ở đây là không bắt buộc vì vậy có thể bỏ đi

String hql = "FROM Employee E";

Query query = session.createQuery(hql);

List results = query.list();

## *SELECT CLAUSE*

## Câu lệnh Select cung cấp cho ta kiểm soát được kết quả trả về từ câu lệnh From. Nếu bạn muốn trả về một vài thuộc tính thay vì cả đối tượng, hãy sử dụng câu lệnh Select như sau:

String hql = "SELECT E.firstName FROM Employee E";

Query query = session.createQuery(hql);

List results = query.list();

## Lưu ý ở đây firstName là thuộc tính của Class Employee chứ không phải của table trong DB

## *WHERE CLAUSE*

## Khi bạn muốn thu hẹp kết quả trả về từ storage, hãy sử dụng mệnh đề WHERE

String hql = "FROM Employee E WHERE E.id = 10";

Query query = session.createQuery(hql);

List results = query.list();

## *ORDER BY CLAUSE*

## Để sắp xếp kết quả truy vấn HQL của bạn, bạn sẽ cần sử dụng mệnh đề ORDER BY. Sắp xếp thuộc tính tăng dần sử dụng ASC, giảm dần là DESC

String hql = "FROM Employee E WHERE E.id > 10 ORDER BY E.salary DESC";

Query query = session.createQuery(hql);

List results = query.list();

## Nếu muốn sắp xếp theo nhiều thuộc tính, thêm thuộc tính bổ sung vào sau dấu phẩy:

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();

## *GROUP BY CLAUSE*

Mệnh đề này cho phép Hibernate lấy thông tin từ cơ sở dữ liệu và nhóm nó lai dựa trên giá trị của một thuộc tính và thông thường để trả về một giá trị tổng hợp.

String hql = "SELECT SUM(E.salary), E.firtName FROM Employee E " +

"GROUP BY E.firstName";

Query query = session.createQuery(hql);

List results = query.list();

## *USING NAME BY PARAMETERS*

## Hibernate hỗ trợ các tham số được đặt tên trong HQL query. Điều này làm cho việc viết các truy vấn HQL chấp nhận đầu vào từ người dùng dễ dàng.

String hql = "FROM Employee E WHERE E.id = :employee\_id";

Query query = session.createQuery(hql);

query.setParameter("employee\_id",10);

List results = query.list();

## *UPDATE CLAUSE*

## Query Interface chứa method có tên executeUpdate() để xử lí lệnh HQL UPDATE hoặc DELETE.

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();

System.out.println("Rows affected: " + result);

## *DELETE CLAUSE*

## Câu lệnh Delete sử dụng để xóa 1 hoặc n đối tượng. Dưới đay là cú pháp:

String hql = "DELETE FROM Employee " +

"WHERE id = :employee\_id";

Query query = session.createQuery(hql);

query.setParameter("employee\_id", 10);

int result = query.executeUpdate();

System.out.println("Rows affected: " + result);

## *INSERT CLAUSE*

## HQL hỗ trợ Insert into chỉ với bản ghi có thể được thêm từ một obj đến một obj khác.

String hql = "INSERT INTO Employee(firstName, lastName, salary)" +

"SELECT firstName, lastName, salary FROM old\_employee";

Query query = session.createQuery(hql);

int result = query.executeUpdate();

System.out.println("Rows affected: " + result);

## *AGGREGATE METHODS*

## HQL cung cấp một số hàm tổng hợp tương tự với SQL và hoạt động tương tự như vậy ở trong HQL như trong SQL

|  |  |
| --- | --- |
| **Sr.No.** | **Functions & Description** |
| 1 | **avg(property name)**  The average of a property's value |
| 2 | **count(property name or \*)**  The number of times a property occurs in the results |
| 3 | **max(property name)**  The maximum value of the property values |
| 4 | **min(property name)**  The minimum value of the property values |
| 5 | **sum(property name)**  The sum total of the property values |

## Từ khóa distinct để trả về giá trị duy nhất của cột. Câu lệnh dưới đây sẽ trả về tổng số cột có giá trị duy nhất:

String hql = "SELECT count(distinct E.firstName) FROM Employee E";

Query query = session.createQuery(hql);

List results = query.list();

## *PAGINATION USING QUERY*

Có hai phương thức của Interface Query để phân trang.

|  |  |
| --- | --- |
| **Sr.Không.** | **Phương pháp & Mô tả** |
| 1 | **Truy vấn setFirstResult (int startPocation)**  Phương thức này lấy một số nguyên biểu thị hàng đầu tiên trong tập kết quả của bạn, bắt đầu bằng hàng 0. |
| 2 | **Truy vấn setMaxResults (int maxResult)**  Phương thức này báo cho Hibernate để lấy một số **maxResults** cố định của các đối tượng. |

## Kết hợp 2 phương thức này, ta dễ dàng phân trang trong web hoặc SWING app

String hql = "FROM Employee";

Query query = session.createQuery(hql);

query.setFirstResult(1);

query.setMaxResults(10);

List results = query.list();

## *Criteria Query*

## Hibernate cung cấp các cách khác nhau để thao tác các đối tượng và lần lượt chuyển đổi dữ liệu có sẵn trong các bảng RDBMS. Một trong những phương pháp là API Criteria, cho phép bạn xây dựng một đối tượng truy vấn tiêu chí theo chương trình, nơi bạn có thể áp dụng các quy tắc lọc và điều kiện logic.

## Giao diện Session Hibernate cung cấp phương thức createdCriteria() , có thể được sử dụng để tạo đối tượng Criteria trả về các thể hiện của lớp đối tượng presistent khi ứng dụng của bạn thực hiện Criteria query. Đây là ví dụ đơn giản:

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

List results = cr.list();

## *Restrictions with Criteria*

## Bạn có thể sử dụng phương thức add() có sẵn cho đối tượng Criteria để thêm hạn chế cho truy vấn tiêu chí. Sau đây là ví dụ để thêm một hạn chế để trả lại các bản ghi với mức lương bằng 2000:

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

cr.add(Restrictions.eq("salary", 2000));

List results = cr.list();

## Sau đây là một vài ví dụ khác bao gồm các kịch bản khác nhau và có thể được sử dụng theo yêu cầu:

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

// To get records having salary more than 2000

cr.add(Restrictions.gt("salary", 2000));

// To get records having salary less than 2000

cr.add(Restrictions.lt("salary", 2000));

// To get records having fistName starting with zara

cr.add(Restrictions.like("firstName", "zara%"));

// Case sensitive form of the above restriction.

cr.add(Restrictions.ilike("firstName", "zara%"));

// To get records having salary in between 1000 and 2000

cr.add(Restrictions.between("salary", 1000, 2000));

// To check if the given property is null

cr.add(Restrictions.isNull("salary"));

// To check if the given property is not null

cr.add(Restrictions.isNotNull("salary"));

// To check if the given property is empty

cr.add(Restrictions.isEmpty("salary"));

// To check if the given property is not empty

cr.add(Restrictions.isNotEmpty("salary"));

## Bạn có thể tạo điều kiện AND và OR bằng cách sử dụng hạn chế LogicalExpression như sau:

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

Criterion salary = Restrictions.gt("salary", 2000);

Criterion name = Restrictions.ilike("firstNname","zara%");

// To get records matching with OR conditions

LogicalExpression orExp = Restrictions.or(salary, name);

cr.add( orExp );

// To get records matching with AND conditions

LogicalExpression andExp = Restrictions.and(salary, name);

cr.add( andExp );

List results = cr.list();

## *Pagination Using Criteria*

## Có 2 method trong Criteria hỗ trợ phân trang

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **public Criteria setFirstResult(int firstResult)**  This method takes an integer that represents the first row in your result set, starting with row 0. |
| 2 | **public Criteria setMaxResults(int maxResults)**  This method tells Hibernate to retrieve a fixed number **maxResults** of objects. |

## Được sử dụng tương tự hql

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

cr.setFirstResult(1);

cr.setMaxResults(10);

List results = cr.list();

## *Sorting the Result*

## API Criteria chí cung cấp lớp org.hibernate.criterion.Order để sắp xếp tập kết quả của bạn theo thứ tự tăng dần hoặc giảm dần, theo một trong các thuộc tính của đối tượng của bạn. Ví dụ này cho thấy cách bạn sẽ sử dụng lớp Order để sắp xếp tập kết quả :

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

// To get records having salary more than 2000

cr.add(Restrictions.gt("salary", 2000));

// To sort records in descening order

cr.addOrder(Order.desc("salary"));

// To sort records in ascending order

cr.addOrder(Order.asc("salary"));

List results = cr.list();

## *Projections & Aggregations*

API Criteria cung cấp lớp **org.hibernate.criterion.Projections** , có thể được sử dụng để lấy giá trị trung bình, max hoặc min của các thuộc tính. Lớp Projection tương tự như lớp Restriction, ở chỗ nó cung cấp một số phương thức nhà máy tĩnh để lấy các phiên bản ***Projections***.

Sau đây là một vài ví dụ bao gồm các kịch bản khác nhau và có thể được sử dụng theo yêu cầu

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

// To get total row count.

cr.setProjection(Projections.rowCount());

// To get average of a property.

cr.setProjection(Projections.avg("salary"));

// To get distinct count of a property.

cr.setProjection(Projections.countDistinct("firstName"));

// To get maximum of a property.

cr.setProjection(Projections.max("salary"));

// To get minimum of a property.

cr.setProjection(Projections.min("salary"));

// To get sum of a property.

cr.setProjection(Projections.sum("salary"));

***Criteria Query Example***

Tạo lớp POJO

public class Employee {

private int id;

private String firstName;

private String lastName;

private int salary;

// getter-setter contructor

}

Tạo bảng EMPLOYEE

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)

);

Mapping file

<?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">

This class contains the employee detail.

</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 hàm main sử dụng Criteria queries

import java.util.List;

import java.util.Date;

import java.util.Iterator;

import org.hibernate.HibernateException;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.hibernate.SessionFactory;

import org.hibernate.Criteria;

import org.hibernate.criterion.Restrictions;

import org.hibernate.criterion.Projections;

import org.hibernate.cfg.Configuration;

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", 2000);

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

Integer empID3 = ME.addEmployee("John", "Paul", 5000);

Integer empID4 = ME.addEmployee("Mohd", "Yasee", 3000);

/\* List down all the employees \*/

ME.listEmployees();

/\* Print Total employee's count \*/

ME.countEmployee();

/\* Print Total salary \*/

ME.totalSalary();

}

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

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

Session session = factory.openSession();

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 having salary more than 2000 \*/

public void listEmployees( ) {

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

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

// Add restriction.

cr.add(Restrictions.gt("salary", 2000));

List employees = cr.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 print total number of records \*/

public void countEmployee(){

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

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

// To get total row count.

cr.setProjection(Projections.rowCount());

List rowCount = cr.list();

System.out.println("Total Coint: " + rowCount.get(0) );

tx.commit();

} catch (HibernateException e) {

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

e.printStackTrace();

} finally {

session.close();

}

}

/\* Method to print sum of salaries \*/

public void totalSalary(){

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

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

// To get total salary.

cr.setProjection(Projections.sum("salary"));

List totalSalary = cr.list();

System.out.println("Total Salary: " + totalSalary.get(0) );

tx.commit();

} catch (HibernateException e) {

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

e.printStackTrace();

} finally {

session.close();

}

}

}

Kết quả console:

$java ManageEmployee

.......VARIOUS LOG MESSAGES WILL DISPLAY HERE........

First Name: Daisy Last Name: Das Salary: 5000

First Name: John Last Name: Paul Salary: 5000

First Name: Mohd Last Name: Yasee Salary: 3000

Total Coint: 4

Total Salary: 15000

Kết quả bảng EMPLOYEE ở DB

+----+------------+-----------+--------+

| id | first\_name | last\_name | salary |

+----+------------+-----------+--------+

| 14 | Zara | Ali | 2000 |

| 15 | Daisy | Das | 5000 |

| 16 | John | Paul | 5000 |

| 17 | Mohd | Yasee | 3000 |

+----+------------+-----------+--------+

4 rows in set (0.00 sec)

mysql>

***Native SQL***

Bạn có thể sử dụng SQL gốc để thể hiện các truy vấn cơ sở dữ liệu nếu bạn muốn sử dụng các tính năng dành riêng cho cơ sở dữ liệu như gợi ý truy vấn hoặc từ khóa CONNECT trong Oracle. Ứng dụng của bạn sẽ tạo một truy vấn SQL gốc từ Session với phương thức **createSQLQuery()** trên Session interface.

public SQLQuery createSQLQuery(String sqlString) throws HibernateException

Sau khi bạn chuyển một chuỗi chứa truy vấn SQL sang phương thức createSQLQuery(), bạn có thể liên kết kết quả SQL với một thực thể Hibernate hiện có, join hoặc tạo kết quả vô hướng bằng cách sử dụng các phương thức addEntity(), addJoin() và addScalar() tương ứng.

## *Scalar Queries*

Truy vấn SQL cơ bản nhất là lấy danh sách vô hướng(giá trị) từ 1 hoặc n bảng. Sau đây là cú pháp sử dụng SQL gốc cho các giá trị vô hướng

String sql = "SELECT first\_name, salary FROM EMPLOYEE";

SQLQuery query = session.createSQLQuery(sql);

query.setResultTransformer(Criteria.ALIAS\_TO\_ENTITY\_MAP);

List results = query.list();

***Entity Queries***

Các truy vấn trên là tất cả về việc trả về giá trị vô hướng, về cơ bản trả về giá trị "thô" từ tập kết quả. Sau đây là cú pháp để lấy toàn bộ các đối tượng thực thể từ truy vấn sql gốc thông qua addEntity ().

String sql = "SELECT \* FROM EMPLOYEE";

SQLQuery query = session.createSQLQuery(sql);

query.addEntity(Employee.class);

List results = query.list();

## *Named SQL Queries*

Sau đây là cú pháp để lấy các đối tượng thực thể từ truy vấn sql gốc thông qua addEntity () và set tham số cho query.

String sql = "SELECT \* FROM EMPLOYEE WHERE id = :employee\_id";

SQLQuery query = session.createSQLQuery(sql);

query.addEntity(Employee.class);

query.setParameter("employee\_id", 10);

List results = query.list();

***Ví dụ Native SQL***

Tạo lớp POJO

public class Employee {

private int id;

private String firstName;

private String lastName;

private int salary;

// contructor & getter - setter

}

Bảng EMPLOYEE

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)

);

Tập tin ánh xạ

<?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">

This class contains the employee detail.

</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>

Phương thức main

import java.util.\*;

import org.hibernate.HibernateException;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.hibernate.SessionFactory;

import org.hibernate.SQLQuery;

import org.hibernate.Criteria;

import org.hibernate.Hibernate;

import org.hibernate.cfg.Configuration;

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", 2000);

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

Integer empID3 = ME.addEmployee("John", "Paul", 5000);

Integer empID4 = ME.addEmployee("Mohd", "Yasee", 3000);

/\* List down employees and their salary using Scalar Query \*/

ME.listEmployeesScalar();

/\* List down complete employees information using Entity Query \*/

ME.listEmployeesEntity();

}

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

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

Session session = factory.openSession();

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 using Scalar Query \*/

public void listEmployeesScalar( ){

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

String sql = "SELECT first\_name, salary FROM EMPLOYEE";

SQLQuery query = session.createSQLQuery(sql);

query.setResultTransformer(Criteria.ALIAS\_TO\_ENTITY\_MAP);

List data = query.list();

for(Object object : data) {

Map row = (Map)object;

System.out.print("First Name: " + row.get("first\_name"));

System.out.println(", Salary: " + row.get("salary"));

}

tx.commit();

} catch (HibernateException e) {

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

e.printStackTrace();

} finally {

session.close();

}

}

/\* Method to READ all the employees using Entity Query \*/

public void listEmployeesEntity( ){

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

String sql = "SELECT \* FROM EMPLOYEE";

SQLQuery query = session.createSQLQuery(sql);

query.addEntity(Employee.class);

List employees = query.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();

}

}

}

Kết quả trên Console:

$java ManageEmployee

.......VARIOUS LOG MESSAGES WILL DISPLAY HERE........

First Name: Zara, Salary: 2000

First Name: Daisy, Salary: 5000

First Name: John, Salary: 5000

First Name: Mohd, Salary: 3000

First Name: Zara Last Name: Ali Salary: 2000

First Name: Daisy Last Name: Das Salary: 5000

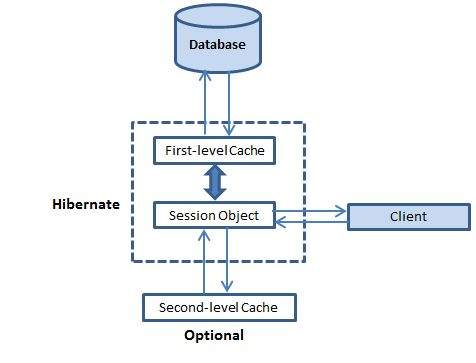
First Name: John Last Name: Paul Salary: 5000

First Name: Mohd Last Name: Yasee Salary: 3000

***Hibernate Caching***

Bộ nhớ đệm là một cơ chế để tăng cường hiệu suất của một hệ thống. Nó là một bộ nhớ đệm nằm giữa ứng dụng và cơ sở dữ liệu. Bộ nhớ cache lưu trữ các mục dữ liệu được sử dụng gần đây để giảm số lần truy cập cơ sở dữ liệu càng nhiều càng tốt.

Bộ nhớ đệm cũng quan trọng đối với Hibernate. Nó sử dụng một sơ đồ bộ nhớ đệm đa cấp như được giải thích dưới đây



***First-level Cache***

Bộ đệm cấp đầu tiên là Bộ đệm Session và là bộ đệm bắt buộc mà qua đó tất cả các yêu cầu phải đi qua. Đối tượng Session giữ một đối tượng dưới sự quản lí của chính nó trước khi đưa nó vào cơ sở dữ liệu.

Nếu bạn update nhiều lần cho một đối tượng, Hibernate cố gắng trì hoãn việc cập nhật càng lâu càng tốt để giảm số lượng các câu lệnh SQL cập nhật. Nếu bạn đóng Session, tất cả các đối tượng được lưu trong bộ nhớ cache sẽ bị mất và cả duy trì hoặc cập nhật trong cơ sở dữ liệu.

***Second-level Cache***

Bộ đệm cấp hai là bộ đệm tùy chọn và bộ đệm cấp một sẽ luôn cân nhắc trước khi xác định vị trí của đối tượng trong bộ đệm cấp hai. Bộ đệm cấp hai có thể được cấu hình trên cơ sở mỗi lớp và mỗi bộ sưu tập và chủ yếu chịu trách nhiệm lưu trữ các đối tượng trong các phiên.

Bất kỳ bộ nhớ cache của bên thứ ba đều có thể được sử dụng với Hibernate. Một giao diện **org.hibernate.cache.CacheProvider** được cung cấp, phải được triển khai để cung cấp cho Hibernate để thực hiện bộ đệm.

***Query-level Cache***

Hibernate cũng thực hiện bộ đệm cho các kết quả truy vấn được tích hợp chặt chẽ với bộ đệm cấp hai. Đây là một tính năng không bắt buộc và yêu cầu hai vùng bộ đệm vật lý bổ sung chứa kết quả truy vấn được lưu trong bộ nhớ cache và thời gian khi bảng được cập nhật lần cuối. Điều này chỉ hữu ích cho các truy vấn được chạy thường xuyên với cùng tham số.

***The Second-level Cache***

Hibernate sử dụng bộ đệm cấp một theo mặc định và bạn không sử dụng bộ đệm cấp một. Vậy hãy đi thẳng đến bộ đệm cấp hai tùy chọn. Không phải tất cả các lớp đều được hưởng lợi từ bộ nhớ đệm, vì vậy điều quan trọng là có thể vô hiệu hóa bộ đệm cấp hai.

Bộ đệm cấp hai của Hibernate được thiết lập theo hai bước. Đầu tiên, bạn phải quyết định sử dụng chiến lược tương tranh nào. Sau đó, bạn định cấu hình hết hạn bộ đệm và thuộc tính bộ đệm vật lý bằng cách sử dụng nhà cung cấp bộ đệm.

## *Concurrency Strategies*

Cách tương tranh là một trung gian hòa giải, chịu trách nhiệm lưu trữ các dữ liệu trong bộ đệm và truy xuất chúng từ bộ đệm. Nếu bạn định kích hoạt bộ đệm cấp hai, bạn sẽ phải quyết định, đối với mỗi class và collection, nên sử dụng cách tương tranh bộ đệm nào.

* **Transactional** - Sử dụng cách này cho dữ liệu chủ yếu là đọc trong đó rất quan trọng để ngăn chặn dữ liệu cũ trong các giao dịch đồng thời, trong trường hợp hiếm hoi của update.
* **Read-Write** - Một lần nữa sử dụng cách này cho dữ liệu chủ yếu là đọc trong đó rất quan trọng để ngăn chặn dữ liệu cũ trong các giao dịch đồng thời, trong trường hợp hiếm hoi của update.
* **Nonstrict-read-write** - cách này không đảm bảo tính nhất quán giữa bộ đệm và cơ sở dữ liệu. Sử dụng cách này nếu dữ liệu hầu như không thay đổi và khả năng dữ liệu cũ không phải là vấn đề quan trọng.
* **Chỉ đọc** - Một cách tương tranh phù hợp với dữ liệu, không bao giờ thay đổi. Chỉ sử dụng nó cho dữ liệu tham khảo.

Nếu chúng ta sẽ sử dụng bộ nhớ đệm cấp hai cho lớp **Nhân viên của** mình, chúng ta hãy thêm phần tử ánh xạ cần thiết để nói với Hibernate để lưu trữ Employee bằng cách sử dụng cách đọc-ghi.

<?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">

This class contains the employee detail.

</meta>

<cache usage = "read-write"/>

<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>

Thuộc tính used = "read-write" cho Hibernate sử dụng cách tương tranh đọc-ghi cho bộ đệm được xác định.

## *Cache Provider*

Bước tiếp theo của bạn sau khi xem xét các chiến lược tương tranh, bạn sẽ sử dụng các lớp bộ đệm nào để chọn nhà cung cấp bộ đệm. Hibernate buộc bạn phải chọn một nhà cung cấp bộ đệm duy nhất cho toàn bộ ứng dụng.

|  |  |
| --- | --- |
| **Sr.No.** | **Cache Name & Description** |
| 1 | **EHCache**  It can cache in memory or on disk and clustered caching and it supports the optional Hibernate query result cache. |
| 2 | **OSCache**  Supports caching to memory and disk in a single JVM with a rich set of expiration policies and query cache support. |
| 3 | **warmCache**  A cluster cache based on JGroups. It uses clustered invalidation, but doesn't support the Hibernate query cache. |
| 4 | **JBoss Cache**  A fully transactional replicated clustered cache also based on the JGroups multicast library. It supports replication or invalidation, synchronous or asynchronous communication, and optimistic and pessimistic locking. The Hibernate query cache is supported. |

Mọi nhà cung cấp bộ đệm không tương thích với mọi chiến lược tương tranh. Ma trận tương thích sau đây sẽ giúp bạn chọn một sự kết hợp phù hợp.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Chiến lược / Nhà cung cấp** | **Chỉ đọc** | **Nonstrictread-write** | **Đọc viết** | **Giao dịch** |
| EHCache | X | X | X |  |
| OSCache | X | X | X |  |
| SwarmCache | X | X |  |  |
| Bộ nhớ cache JBoss | X |  |  | X |

Bạn sẽ chỉ định một nhà cung cấp bộ đệm trong tệp cấu hình hibernate.cfg.xml. Chúng tôi chọn EHCache làm nhà cung cấp bộ đệm cấp hai của mình -

<?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>

<!-- Assume students is the database name -->

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

jdbc:mysql://localhost/test

</property>

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

root

</property>

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

root123

</property>

<property name = "hibernate.cache.provider\_class">

org.hibernate.cache.EhCacheProvider

</property>

<!-- List of XML mapping files -->

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

</session-factory>

</hibernate-configuration>

Bây giờ, bạn cần chỉ định các thuộc tính của các vùng bộ đệm. EHCache có tệp cấu hình riêng, **ehcache.xml** , tệp này phải có trong CLASSPATH của ứng dụng. Một cấu hình bộ đệm trong ehcache.xml cho lớp Employee có thể trông như thế này

<diskStore path="java.io.tmpdir"/>

<defaultCache

maxElementsInMemory = "1000"

eternal = "false"

timeToIdleSeconds = "120"

timeToLiveSeconds = "120"

overflowToDisk = "true"

/>

<cache name = "Employee"

maxElementsInMemory = "500"

eternal = "true"

timeToIdleSeconds = "0"

timeToLiveSeconds = "0"

overflowToDisk = "false"

/>

Chúng ta đã bật bộ nhớ đệm cấp hai cho lớp Employee và Hibernate, giờ đây sẽ truy cập bộ đệm cấp hai bất cứ khi nào bạn điều hướng đến Employee hoặc khi bạn load Employee bằng định danh.

Bạn nên phân tích tất cả các lớp của bạn và chọn chiến lược bộ nhớ đệm phù hợp cho từng lớp. Đôi khi, bộ nhớ đệm cấp hai có thể làm giảm hiệu suất của ứng dụng. Vì vậy, bạn nên đánh giá ứng dụng của mình trước, không cần bật bộ đệm và sau đó bật bộ đệm phù hợp tốt của bạn và kiểm tra hiệu suất. Nếu bộ nhớ đệm không cải thiện hiệu năng hệ thống, thì không nên bật bất kỳ loại bộ đệm.

## *Cache cấp truy vấn*

Để sử dụng bộ đệm truy vấn, trước tiên bạn phải kích hoạt nó bằng thuộc tính **hibernate.cache.use\_query\_cache="true"** trong tệp cấu hình. Bằng cách đặt thuộc tính này thành true, bạn tạo Hibernate tạo bộ nhớ cache cần thiết trong bộ nhớ để giữ bộ truy vấn và bộ định danh.

Tiếp theo, để sử dụng bộ đệm truy vấn, bạn sử dụng phương thức setCachizable (Boolean) của lớp Truy vấn. 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 đệm rất chi tiết thông qua khái niệm vùng đệm. Vùng bộ đệm là một phần của bộ đệm 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();

Mã này sử dụng phương thức để báo cho Hibernate lưu trữ và tìm kiếm truy vấn trong vùng employee của bộ đệm.

***Batch Processing***

Cân nhắc trường hợp bạn cần upload lượng lớn các bản ghi vào db sử dụng Hibernate. Như trường hợp dưới đây:

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 tất cả các đối tượng tồn tại trong bộ đệm Session và cuối cùng, ứng dụng của bạn sẽ ném **OutOfMemoryException** quanh hàng thứ 50000. Bạn có thể xử lí điều này nếu bạn sử dụng **batch processing.**

Để sử dụng tính năng này, đầu tiên hãy set **hibernate.jdbc.batch\_size** làm kích thước batch và khoảng 20 – 50 tùy thuộc vào đối tượng. Để thực hiện điều này, ta phải sửa đổi như sau

Session session = SessionFactory.openSession();

Transaction tx = session.beginTransaction();

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

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

session.save(employee);

if( i % 50 == 0 ) { // Same as the JDBC batch size

//flush a batch of inserts and release memory:

session.flush();

session.clear();

}

}

tx.commit();

session.close();

Đoạn mã trên sẽ ổn khi bạn sử dụng **Insert**, nhưng nếu bạn sử dụng **Update** thì nên sử dụng đoạn code dưới 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();

## *Batch Processing Example*

## Thêm thuộc tính hibernate.jdbc.batch\_size vào file config.

<?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>

<!-- Assume students is the database name -->

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

jdbc:mysql://localhost/test

</property>

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

root

</property>

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

root123

</property>

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

50

</property>

<!-- List of XML mapping files -->

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

</session-factory>

</hibernate-configuration>

## Lớp POJO

public class Employee {

private int id;

private String firstName;

private String lastName;

private int salary;

// contructor – getter & setter

}

## Bảng Employee

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)

);

## File Mapping Employee OBJ với EMPLOYEE ở Table

<?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">

This class contains the employee detail.

</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>

Cuối cùng, chúng ta sẽ tạo main () để chạy ứng dụng nơi chúng ta sẽ sử dụng các phương thức **flush()** và **clear()** có sẵn với đối tượng Session để Hibernate tiếp tục ghi các bản ghi này vào cơ sở dữ liệu thay vì lưu trữ chúng vào memory.