### **Introduction**

#### Xin chào, Tên tôi là Groovy, con nhím ma thuật.

#### Có một quan niệm sai lầm rằng chúng ta không phải là những sinh vật thông minh, nhưng thực tế là chúng ta nhận thức được môi trường xung quanh.

#### Tôi ngủ đông trong một thời gian dài, và tỉnh táo cho phần còn lại.

#### Trong khi tôi thức, tôi học.

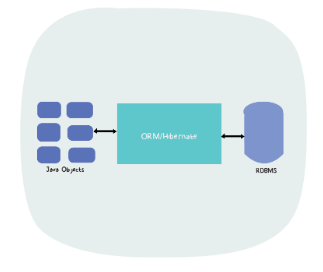


#### Gần đây tôi đã xem qua một khóa học được gọi là Java Hibernate.

#### Đó là dữ liệu hoặc đối tượng trong cơ sở dữ liệu được đặt 'ở chế độ ngủ' và được gọi lại khi cần.

#### Tương tự như Hibernate của tôi, nhưng thú vị hơn.

#### Hãy bắt đầu trước khi tôi ngủ đông một lần nữa.



#### Hibernate là một giải pháp Object-Relational Mapping (ORM) cho JAVA.

#### Nó là một khung liên tục mã nguồn mở.

#### Đây là một dịch vụ truy vấn và bền bỉ quan hệ đối tượng mạnh mẽ, hiệu suất cao cho bất kỳ Ứng dụng Java nào.

#### Nó ánh xạ các lớp Java vào các bảng cơ sở dữ liệu và các kiểu dữ liệu Java thành các kiểu dữ liệu SQL.

#### Điều này giúp nhà phát triển giảm bớt 95% các tác vụ liên quan đến tính bền bỉ của dữ liệu.

### **Ưu điểm của Hibernate**

* Hibernate đảm nhận việc ánh xạ các lớp Java vào các bảng cơ sở dữ liệu. Chúng tôi không phải thay đổi bất kỳ dòng nào trong mã.
* Cung cấp các API đơn giản để lưu trữ và truy xuất các đối tượng Java.
* Hibernate không sử dụng bất kỳ máy chủ ứng dụng nào để hoạt động.
* Thao tác Các liên kết phức tạp của các đối tượng trong cơ sở dữ liệu của bạn.
* Giảm thiểu quyền truy cập cơ sở dữ liệu bằng các chiến lược tìm nạp thông minh.
* Cung cấp truy vấn dữ liệu đơn giản.

#### Hibernate hỗ trợ hầu hết tất cả các RDBMS chính.

#### Nó bao gồm:

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

### **Hibernate Architecture**

#### Hibernate có kiến trúc nhiều lớp.

#### Điều này giúp ngay cả người dùng ngây thơ cũng có thể hoạt động mà không có kiến thức về API.

#### Hibernate sử dụng cơ sở dữ liệu và dữ liệu cấu hình để cung cấp các dịch vụ bền bỉ cho ứng dụng.

#### Chúng ta hãy thảo luận chi tiết từng đối tượng lớp liên quan đến Hibernate Application Architecture.

### **Configuration Object**

#### Đối tượng Configuration là đối tượng Hibernate đầu tiên mà chúng ta tạo ra.

#### Nó thường chỉ được tạo một lần trong quá trình khởi tạo ứng dụng.

#### Nó đại diện cho một sự sắp xếp hoặc tệp thuộc tính cần thiết cho Hibernate.

#### Đối tượng Cấu hình cung cấp hai thành phần chính:

* Kết nối cơ sở dữ liệu - Các tệp này là hibernate.properties và hibernate.cfg.xml
* Class Mapping Setup – Điều này tạo ra kết nối giữa các lớp Java và các bảng cơ sở dữ liệu.

### **Đối tượng SessionFactory**

#### Đối tượng cấu hình được sử dụng để tạo một đối tượng SessionFactory.

#### Điều này được sử dụng để cấu hình Hibernate cho ứng dụng bằng cách sử dụng tệp cấu hình bên ngoài.

#### SessionFactory là một đối tượng thread-safe và được sử dụng bởi tất cả các thread của một ứng dụng.

#### SessionFactory là một vật nặng.

#### Nó thường được tạo trong quá trình khởi động ứng dụng và được giữ lại để sử dụng sau này.

#### Nếu bạn đang sử dụng nhiều cơ sở dữ liệu, thì bạn phải tạo nhiều đối tượng SessionFactory.

### **Transaction Object**

#### Giao dịch là bất kỳ công việc đầu vào / đầu ra nào được thực hiện trên Cơ sở dữ liệu.

#### Các giao dịch trong Hibernate được quản lý bởi một người quản lý giao dịch và giao dịch cơ bản.

#### Giao dịch là một đối tượng tùy chọn.

#### Các ứng dụng ngủ đông có thể chọn không sử dụng giao diện này.

#### Thay vào đó, họ có thể chọn quản lý các giao dịch trong mã ứng dụng của mình.

### **Query Object**

#### Đối tượng truy vấn sử dụng chuỗi SQL hoặc Ngôn ngữ truy vấn ngủ đông (HQL) để truy xuất dữ liệu từ cơ sở dữ liệu và tạo mục.

#### Một phiên bản truy vấn được sử dụng để phân bổ các tham số truy vấn.

#### Nó cũng giới hạn số lượng kết quả được trả về bởi truy vấn.

#### Sau đó, cuối cùng nó thực hiện truy vấn.

### **Criteria Object**

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

### **Preparing Hibernate Environment**

#### Chương này giải thích cách cài đặt Hibernate.

#### Làm thế nào để chuẩn bị một môi trường cho các ứng dụng Hibernate.

#### Chúng tôi sẽ làm việc với cơ sở dữ liệu MySQL để thử nghiệm với các ví dụ Hibernate.

#### Trước tiên chúng ta hãy tìm hiểu cách thiết lập cơ sở dữ liệu MySQL

mysqld.exe --console

#### MySQL được đóng gói gọn gàng với một trình cài đặt.

#### Chỉ cần tải xuống gói trình cài đặt, giải nén nó ở bất cứ đâu và chạy tệp .exe thiết lập.

#### Thiết lập trình cài đặt mặc định.exe sẽ hướng dẫn bạn qua quy trình tầm thường.

#### Theo mặc định, nó sẽ cài đặt mọi thứ dưới C: \ mysql.

#### Kiểm tra máy chủ bằng cách kích hoạt nó từ dấu nhắc lệnh lần đầu tiên, viết mã đã cho ở trên.

[root@host]# mysqladmin -u root password "new\_password";

#### Ngay sau khi bạn đã cài đặt thành công cơ sở dữ liệu,

#### Bạn cần đặt mật khẩu root như trên trong khối mã.

[root@host]# mysql -u root -p

Enter password:\*\*\*\*\*\*\*

#### Bây giờ để tạo kết nối với máy chủ MySQL của bạn, hãy sử dụng đoạn mã trên.

#### Bây giờ chúng ta đã hoàn tất cài đặt và thiết lập MySQL, chúng ta có thể bắt đầu với Hibernate.

### **Downloading Hibernate**

#### Bạn nên cài đặt Java trên hệ thống của mình.

#### Sau đây là các bước để tải xuống và cài đặt Hibernate trên hệ thống của bạn:

* Hãy lựa chọn xem bạn muốn cài đặt Hibernate trên Windows hay Unix và sau đó tiến hành bước tiếp theo để tải xuống tệp .zip cho windows và tệp .tz cho Unix.
* Tải xuống phiên bản mới nhất của Hibernate từ http://www.hibernate.org/downloads.
* Khi bạn giải nén tệp đã tải xuống, nó sẽ cung cấp cho bạn cấu trúc thư mục như thể hiện trong hình trên.

### **Cài đặt Hibernate**

#### Hãy chắc chắn rằng bạn đang thiết lập biến CLASSPATH của bạn đúng cách nếu không bạn sẽ phải đối mặt với vấn đề trong khi biên dịch ứng dụng của bạn.

#### Sao chép tất cả các tệp thư viện từ /lib vào CLASSPATH của bạn và thay đổi biến classpath của bạn để bao gồm tất cả các JAR.

#### Cuối cùng, sao chép tập tin hibernate3.jar vào CLASSPATH của bạn.

#### Tệp này nằm trong thư mục gốc của quá trình cài đặt và là JAR chính mà Hibernate cần để thực hiện công việc của nó.

#### Có một danh sách các gói / thư viện được yêu cầu bởi Hibernate, Bạn nên cài đặt chúng trước khi bắt đầu với Hibernate.

#### Để cài đặt các gói này, bạn sẽ phải sao chép các tệp thư viện từ /lib vào CLASSPATH của bạn và thay đổi biến CLASSPATH của bạn cho phù hợp.

#### Dưới đây là các gói mà bạn cần tải xuống:

* dom4j: www.dom4j.org/ phân tích cú pháp XML
* Xalan: Bộ xử lý XSLT https://xml.apache.org/xalan-j/
* Xerces: Trình phân tích cú pháp Java Xerces https://xml.apache.org/xerces-j/
* Cglib: Thay đổi thích hợp cho các lớp Java trong thời gian chạy http://cglib.sourceforge.net/
* log4j: Ghi nhật ký Faremwork https://logging.apache.org/log4j
* Commons: Ghi nhật ký, Email v.v. https://jakarta.apache.org/commons

#### Sau khi hoàn tất cài đặt, chúng ta có thể tạo một dự án Hibernate trong Eclipse.

1. Tạo dự án java
2. Thêm tệp jar cho chế độ ngủ đông
3. Tạo lớp Persistent
4. Tạo tệp ánh xạ cho lớp Persistent
5. Tạo tệp Cấu hình
6. Tạo lớp truy xuất hoặc lưu trữ đối tượng persistent
7. Chạy ứng dụng

#### Chúng ta sẽ tìm hiểu chi tiết về các bước trong chương tiếp theo.

### **Properties of Hibernate**

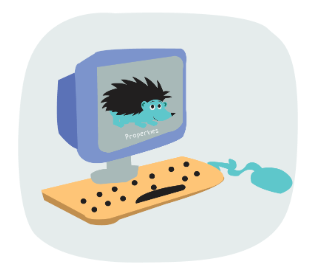
#### Hibernate cần biết trước nơi tìm thông tin bản đồ.

#### Nó cần biết các lớp Java của bạn liên quan đến các bảng cơ sở dữ liệu như thế nào.

#### Điều này thường xuất hiện trong tệp thuộc tính Java tiêu chuẩn hibernate.properties hoặc dưới dạng tệp XML ngủ đông.cfg.xml.

#### Chúng ta sẽ xem xét tệp được định dạng XML ngủ đông.cfg.xml để chỉ định các thuộc tính Hibernate cần thiết trong các ví dụ của tôi.

#### Bây giờ chúng ta hãy nhìn vào các thuộc tính Hibernate.

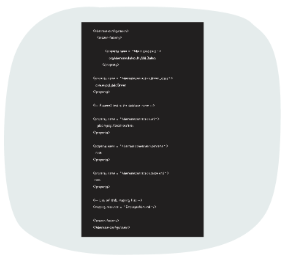


1. hibernate.dialect: Thuộc tính này làm cho biểu mẫu Hibernate trở thành SQLQuery chính xác cho cơ sở dữ liệu đã chọn.
2. hibernate.connection.driver\_class: Lớp trình điều khiển JDBC.
3. hibernate.connection.url: URL kết nối JDB kết nối với phiên bản cơ sở dữ liệu.
4. hibernate.connection.username: Chứa tên người dùng cơ sở dữ liệu.

#### 5. hibernate.connection.password: Chứa mật khẩu cơ sở dữ liệu.

#### 6. hibernate.connection.pool\_size: Nó chứa giới hạn tối đa của số lượng kết nối đang chờ trong nhóm kết nối cơ sở dữ liệu Hibernate.

#### 7. hibernate.connection.autocommit: Cho phép sử dụng chế độ tự động cam kết cho kết nối JDBC. Autocomit có nghĩa là tự động lưu dữ liệu trong cơ sở dữ liệu



#### Đây là cách tệp cấu hình cho ngủ đông sẽ trông như thế nào.

#### Ở đây chúng tôi đã giả định rằng hệ thống cơ sở dữ liệu là MySQL.

#### Bạn sẽ phải đảm bảo rằng bạn có cơ sở dữ liệu testdb có sẵn trong MySQL của bạn.

#### Bạn phải có sẵn kiểm tra người dùng để truy cập cơ sở dữ liệu.

#### Tệp cấu hình trên bao gồm các thẻ <ánh xạ>.

#### Điều này có vẻ mới đối với bạn, Nó liên quan đến tệp hibernatemapping.

#### Chúng ta sẽ thấy điều này trong các chương tiếp theo.

### **Concept of Sessions**

#### Phiên được sử dụng để có được kết nối đáng kể với cơ sở dữ liệu.

#### Đối tượng Session có trọng lượng nhẹ.

#### Nó được thiết kế để được 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 Phiên.

#### Các đối tượng phiên không nên được giữ mở trong một thời gian dài.

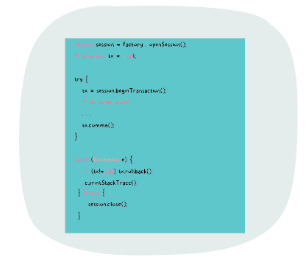
#### Đây thường không phải là sợi an toàn.

#### Chúng nên được tạo ra và phá hủy khi cần thiết.

#### Chức năng chính của Session là cung cấp, tạo, đọc và xóa các hoạt động cho các thể hiện của các lớp thực thể được ánh xạ.

#### Bây giờ, các trường hợp có thể tồn tại ở bất kỳ trạng thái nào trong ba tiểu bang:

* Thoáng qua - Đây là một ví dụ mới của một lớp dai dẳng. Nó không liên minh với Phiên và không có đại diện trong cơ sở dữ liệu.
* dai dẳng - Liên kết một phiên với một đối tượng thoáng qua và bạn có một đối tượng liên tục. Một phiên bản liên tục có một đại diện trong cơ sở dữ liệu.
* tách rời - Khi chúng ta kết thúc Phiên Hibernate, phiên bản liên tục sẽ trở thành một thể hiện tách rời.



#### Nếu một phiên ném một ngoại lệ, giao dịch phải được khôi phục và phiên phải được kết thúc.

#### Ở trên là một ví dụ về cách các phiên có thể được tạo.

#### Cho đến nay chúng tôi đã tìm hiểu về các phiên họp.

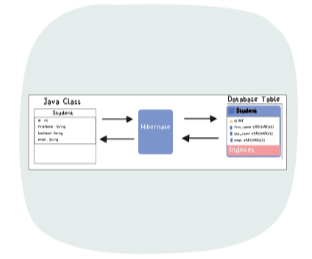
#### Nhưng các buổi học chỉ có ý nghĩa với Lớp học kiên trì.

#### Trong bài học tiếp theo, chúng ta sẽ tìm hiểu Khái niệm về các lớp học liên tục.

### **Persistence Class theory**

#### Mục tiêu chính của Hibernate là lấy các giá trị từ các thuộc tính lớp Java và duy trì / duy trì chúng vào bảng cơ sở dữ liệu.

#### Tài liệu ánh xạ giúp Hibernate xác định cách kéo các giá trị, Nơi ánh xạ chúng với bảng và các trường liên quan.



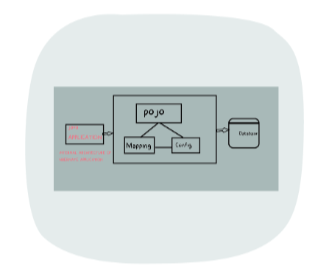
#### Các lớp Java có đối tượng sẽ được lưu trữ trong các bảng cơ sở dữ liệu được gọi là các lớp liên tục.

#### Điều này có nghĩa là lớp java mà từ đó các thuộc tính java đang được ánh xạ, được gọi là lớp liên tục.

#### Hibernate hoạt động tốt nhất nếu các lớp này tuân theo một số quy tắc đơn giản.

#### Chúng được gọi là mô hình lập trình Plain Old Java Object (POJO).

#### Hãy cùng tìm hiểu thêm về các mô hình POJO.



#### Một số quy tắc của mô hình lập trình POJO như sau:

* Tất cả các lớp Java sẽ là các lớp liên tục cần một hàm tạo mặc định.
* Tất cả các lớp phải chứa ID để cho phép dễ dàng xác định các đối tượng của bạn trong Hibernate và cơ sở dữ liệu. Về cơ bản, nó phải có một khóa chính.
* Tất cả các thuộc tính sẽ được duy trì phải được khai báo là riêng tư.
* Một thuộc tính chính của Hibernate, proxy, phụ thuộc vào lớp persistent không phải là cuối cùng hoặc việc triển khai một giao diện khai báo tất cả các phương thức công khai.
* Tên POJO được sử dụng để tập trung rằng một đối tượng nhất định là một Đối tượng Java thông thường, không phải là một đối tượng đặc biệt.

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;

}

}

#### Chương trình trên là một chương trình java đơn giản tuân theo hầu hết các quy tắc của POJO.

#### Nó chứa ID là riêng tư, có thể là khóa chính được yêu cầu.

#### Nó có các phương pháp công khai nhận và thiết lập.

#### Chà, đó là khá nhiều thứ bạn cần cho các lớp học Liên tục.

#### Nghe có vẻ phức tạp nhưng là các lớp java đơn giản với một số ràng buộc bổ sung.

#### Bây giờ chúng ta hãy sửa đổi đơn vị và chuyển sang chương tiếp theo của chúng ta Lập bản đồ.

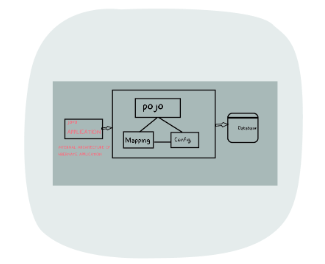
### **Mapping Files**

#### Ánh xạ đối tượng/quan hệ được định nghĩa trong một tài liệu XML.

#### Tệp ánh xạ này hướng dẫn Hibernate cách ánh xạ lớp hoặc lớp đã xác định vào các bảng cơ sở dữ liệu?

#### Mặc dù nhiều người dùng Hibernate chọn viết XML theo cách thủ công, một số công cụ tồn tại để tạo tài liệu ánh xạ.

#### Đây là XDoclet, Middlegen và AndroMDA cho người dùng Hibernate nâng cao.



#### Bạn có nhớ chương trình POJO mà chúng ta đã sử dụng trong chương trước không?

#### Ở đây chúng ta sẽ sử dụng cùng một chương trình ở đây và tìm hiểu về các tệp ánh xạ và cách chúng ta có thể tạo tệp cấu hình.

#### Hãy bắt đầu nào.



#### Đối với mỗi đối tượng sẽ có một bảng tương ứng.

#### Rõ ràng, ở đây chúng ta đang nói về các đối tượng mà bạn muốn làm cho kiên trì.

#### Giả sử rằng các đối tượng đã cho ở trên cần được lưu trữ và truy xuất vào bảng RDBMS sau



#### Dựa trên các thực thể được xác định trước đó, chúng ta có thể định nghĩa tệp ánh xạ ở trên.

#### Điều này sẽ hướng dẫn Hibernate cách ánh xạ lớp hoặc lớp được xác định vào các bảng cơ sở dữ liệu.

#### Khóa chính là đối tượng ID trong ví dụ đã cho.

#### Khi bạn đã hoàn tất việc tạo tài liệu ánh xạ cấu hình, bạn nên lưu tài liệu ánh xạ trong một tệp có định dạng <classname>.hbm.xml.

#### Bây giờ chúng ta hãy hiểu chi tiết về các phần tử ánh xạ được sử dụng trong tệp ánh xạ.

* Tài liệu ánh xạ là một tài liệu XML có <ánh xạ ngủ đông> làm phần tử gốc.
* Điều này 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.
* Phần tử <meta> là một phần tử tùy chọn và có thể được sử dụng để tạo mô tả lớp hoặc dữ liệu meta về lớp được thực hiện liên tục.
* Phần tử <id> ánh xạ thuộc tính ID duy nhất trong lớp tới khóa chính của bảng cơ sở dữ liệu.
* Tương tự, các lĩnh vực khác cũng đang được lập bản đồ.
* Phần tử <generator> trong phần tử id được sử dụng để tự động tạo các giá trị khóa chính.
* Phần tử <property> được sử dụng để ánh xạ một thuộc tính lớp Java tới một cột trong bảng cơ sở dữ liệu.

#### Vâng, đó là nó cho Mapping Files.

#### Nó ít nhiều giống như kết nối các dấu chấm giữa các lớp Java và Cơ sở dữ liệu.

#### Chúng ta hãy xem xét nhanh về những gì chúng ta đọc cho đến nay.

### **O/R Mappings**

#### Cho đến nay, chúng ta đã thấy ánh xạ O / R rất cơ bản bằng cách sử dụng chế độ ngủ đông.

#### Nhưng lập trình thực tế có thể phức tạp hơn.

#### Có ba chủ đề lập bản đồ quan trọng nhất, mà chúng ta cần tìm hiểu chi tiết.

#### Đó là:

* Lập bản đồ các bộ sưu tập,
* Ánh xạ các liên kết giữa các lớp thực thể, và
* Ánh xạ thành phần.

### **Collections Mappings**

#### Nếu trong một thực thể hoặc lớp cho một biến duy nhất có các tập hợp các giá trị.

#### Trong trường hợp đó, chúng tôi ánh xạ các giá trị đó bằng cách sử dụng bất kỳ một trong các giao diện bộ sưu tập có sẵn trong java.

#### Hibernate có thể tồn tại các trường hợp:

* java.util.Map,
* java.util.Set,
* java.util.SortedMap,
* java.util.SortedSet,
* java.util.List,
* array of persistent entities or values.

### **Lập bản đồ hiệp hội**

#### **Ánh xạ các liên kết giữa các lớp thực thể** là mấu chốt của ORM.

#### Chúng ta hãy thảo luận về bốn cách mà tính chính yếu của mối quan hệ giữa các đối tượng có thể được thể hiện.

#### Chúng ta nên nhớ rằng những mối quan hệ này cũng có thể là hai chiều.

#### Hãy thảo luận về những mối quan hệ này trong màn hình tới.

### **Many-to-One**

#### Lập bản đồ các mối quan hệ nhiều-một là loại liên kết phổ biến nhất.

#### Một đối tượng có thể được liên kết với nhiều đối tượng.

#### Ví dụ: cùng một đối tượng địa chỉ có thể được liên kết với nhiều đối tượng nhân viên.

### **One-to-One**

#### Hiệp hội một-một tương tự như hiệp hội nhiều-một.

#### Chỉ khác biệt là cột sẽ được đặt là duy nhất.

#### Ví dụ: một đối tượng địa chỉ có thể được liên kết với một đối tượng nhân viên duy nhất.

### **One-to-Many**

#### Ánh xạ One-to-Many có thể được thực hiện bằng cách sử dụng bộ sưu tập Set java.

#### Nó không chứa bất kỳ yếu tố trùng lặp nào.

### **Many-to-Many**

#### Quy tắc tương tự như One to Many, nhưng ở đây nhiều thực thể có thể được ánh xạ tới nhiều thực thể, miễn là tập hợp không chứa bất kỳ phần tử trùng lặp nào.

### **Component Mappings**

#### Có thể 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 hoặc nó hoàn toàn phụ thuộc vào vòng đời của lớp thực thể, thì lớp được giới thiệu được gọi là lớp Thành phần.

#### Các ánh xạ này có thể giống như ánh xạ tập thể với những thay đổi cấu hình nhỏ.

#### Điều cần lưu ý là lớp component có sự phụ thuộc hoàn toàn vào lớp sở hữu.

#### Chúng tôi đã đề cập đến hầu hết tất cả các khái niệm quan trọng về ánh xạ OR.

#### Bây giờ chúng ta hãy nhìn lại những gì chúng ta đã học được cho đến nay

### **Hibernate Query Language**

#### Ngôn ngữ truy vấn ngủ đông (HQL) là một ngôn ngữ truy vấn hướng đối tượng.

#### Nó rất giống với SQL.

#### Sự khác biệt là SQL hoạt động trên các bảng và cột,

#### HQL hoạt động với các đối tượng bền vững và tính chất của chúng.

#### Các truy vấn HQL được dịch bởi Hibernate thành các truy vấn SQL thông thường, từ đó thực hiện hành động trên cơ sở dữ liệu.

#### Hãy cùng tìm hiểu một số mệnh đề HQL cơ bản và quan trọng.

### **FROM Clause**

#### Chúng ta sử dụng mệnh đề FROM nếu chúng ta muốn tải một đối tượng persistent hoàn chỉnh vào bộ nhớ.

#### Ở đây, chúng tôi lấy các giá trị từ Employee.

#### Nếu chúng ta muốn kéo một lớp hoàn chỉnh trong HQL, chỉ cần viết:

FROM com.hibernate book.criteria.Employee

### **AS Clause**

#### Mệnh đề AS có thể được sử dụng để gán bí danh cho các lớp trong truy vấn HQL của bạn.

#### Điều này được sử dụng đặc biệt khi bạn có truy vấn dài.

#### AS là tùy chọn, chúng ta có thể gán trực tiếp bí danh cho các lớp sau tên lớp.

String hql = "FROM Employee E";

### **SELECT Clause**

#### Mệnh đề SELECT cung cấp nhiều quyền kiểm soát hơn đối với tập kết quả.

#### Nếu chúng ta muốn lấy một số thuộc tính nhất định của các đối tượng thay vì đối tượng hoàn chỉnh, hãy sử dụng mệnh đề SELECT.

#### Từ thực thể Nhân viên, chúng ta kéo firstName trong ví dụ trên.

### **WHERE Clause**

#### Nếu chúng tôi muốn thu hẹp thêm các đối tượng cụ thể được trả về từ bộ nhớ, bạn sử dụng mệnh đề WHERE.

#### Trong ví dụ trên, chúng ta kéo các đối tượng từ Employee, nhưng chỉ các giá trị có id = 10.

### **ORDER BY Clause**

#### Để sắp xếp kết quả của truy vấn HQL tăng dần hoặc giảm dần, chúng ta cần sử dụng mệnh đề ORDER BY.

#### Bạn có thể sắp xếp các kết quả trong bộ kết quả tăng dần (ASC) hoặc giảm dần (DESC) bằng bất kỳ thuộc tính nào trên các đối tượng.

### **GROUP BY Clause**

#### Chúng ta có thể lấy dữ liệu từ cơ sở dữ liệu và nhóm nó dựa trên giá trị của một thuộc tính.

#### Chúng tôi thường sử dụng kết quả để bao gồm một giá trị tổng hợp.

### **UPDATE Clause**

#### Mệnh đề UPDATE có thể được sử dụng để cập nhật một hoặc nhiều thuộc tính của một hoặc nhiều đối tượng.

#### Trong ví dụ trên, chúng tôi đã cập nhật các đối tượng, tiền lương và id nhân viên.

### **DELETE Clause**

#### Mệnh đề DELETE có thể được sử dụng để xóa một hoặc nhiều đối tượng.

#### Ở trên, Chúng tôi đã xóa các đối tượng liên quan đến id nhân viên = 10.

### **INSERT Clause**

#### HQL chỉ hỗ trợ mệnh đề INSERT INTO khi dữ liệu có thể được chèn từ đối tượng này sang đối tượng khác.

### **Native SQL Query**

#### Bạn có thể sử dụng SQL gốc để thể hiện các truy vấn cơ sở dữ liệu.

#### Thường được sử dụng khi bạn muốn sử dụng các tính năng dành riêng cho cơ sở dữ liệu,

#### chẳng hạn như gợi ý truy vấn hoặc từ khóa CONNECT trong Oracle.

#### Hibernate 3.x cho phép bạn chỉ định SQL viết tay.

public SQLQuery createSQLQuery(String sqlString) throws HibernateException

#### Ứng dụng sẽ tạo một truy vấn SQL gốc từ phiên với tệp .createSQLQuery

#### Sau khi chúng ta truyền một chuỗi chứa truy vấn SQL đến createSQLQuery(), chúng ta có thể liên kết kết quả SQL với một thực thể Hibernate hiện có.

### **Scalar Queries**

#### Truy vấn SQL cơ bản nhất là lấy danh sách vô hướng (giá trị) từ một hoặc nhiều bảng. Các truy vấn trên là tất cả về việc trả về các giá trị vô hướng,

#### Về cơ bản trả về các giá trị "thô" từ bộ kết quả.

### **Entity Queries**

#### Ở trên là cú pháp để lấy toàn bộ các đối tượng thực thể từ một truy vấn sql gốc thông qua addEntity().

### **Named SQL Queries**

#### Ở trên là cú pháp để lấy các đối tượng thực thể từ một truy vấn sql gốc thông qua addEntity() và sử dụng truy vấn SQL có tên.

#### Mặc dù nó hiếm khi được sử dụng, SQL gốc đặt cơ sở của HQL.

#### Sau đây là một số lý do tại sao HQL được ưa thích hơn SQL:

* Cung cấp hỗ trợ đầy đủ cho các hoạt động quan hệ.
* Trả về kết quả dưới dạng đối tượng
* Hỗ trợ truy vấn đa hình. Truy vấn đa hình trả về kết quả truy vấn cùng với tất cả các đối tượng con (đối tượng của lớp con), nếu có.
* Dễ học và sử dụng.
* Hỗ trợ nhiều tính năng nâng cao so với SQL, chẳng hạn như phân trang, tìm nạp tham gia với hồ sơ động
* Cung cấp tính độc lập của cơ sở dữ liệu.

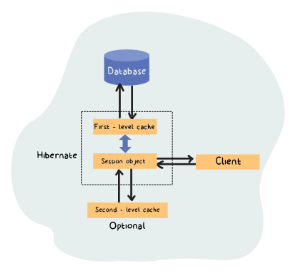
### **Introduction to Caching**

#### Bây giờ bạn đã quen thuộc với Ngôn ngữ truy vấn Hibernate, đã đến lúc chúng ta tìm hiểu cách nâng cao hiệu suất của Ứng dụng của chúng ta.

#### Bộ nhớ đệm là một cơ chế để nâng cao hiệu suất của một hệ thống.

#### Nó là một bộ nhớ đệm tồn tại 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 rất quan trọng đối với Hibernate.

#### Hình trên giải thích cách ngủ đông sử dụng sơ đồ bộ nhớ đệm đa cấp.

#### Bây giờ chúng ta hãy thảo luận chi tiết về từng cấp độ.

### **First-level Cache**

#### Bộ đệm ẩn cấp đầu tiên là bộ đệm ẩn phiên.

#### Đây là một bộ đệm bắt buộc mà qua đó tất cả các truy vấn phải vượt qua.

#### Nếu bạn phát hành nhiều bản Cập Nhật cho một đối tượng, Hibernate cố gắng trì hoãn thực hiện cập nhật càng lâu càng tốt để giảm số lượng câu lệnh SQL cập nhật được phát hành.

#### Nếu bạn kết thúc phiên, tất cả các đối tượng được lưu trong bộ nhớ cache sẽ bị mất và vẫn tồn tại hoặc cập nhật trong cơ sở dữ liệu.

### **Second-level Cache**

#### Bộ nhớ cache cấp hai là một bộ đệm tùy chọn.

#### Bộ đệm cấp một sẽ luôn được giới thiệu trước khi bất kỳ thao tác nào được thực hiện trong bộ đệm cấp hai.

#### Bất kỳ bộ nhớ cache của bên thứ ba nào cũng có thể được sử dụng với Hibernate.

### **Query-level Cache**

#### Hibernate cũng thực hiện một bộ đệm ẩn cho các bộ kết quả truy vấn.

#### Nó tích hợp chặt chẽ với bộ đệm cấp hai.

#### Đây là một tính năng tùy chọn.

#### Nó cần hai vùng bộ nhớ cache vật lý bổ sung lưu trữ:

* Kết quả truy vấn được lưu trong bộ đệm ẩn
* Dấu thời gian khi bảng được cập nhật lần cuối.

#### Điều này chỉ hữu ích cho các truy vấn được chạy thường xuyên với cùng một tham số.

#### Đây là ba bộ nhớ cache mà Hibernate sử dụng để lưu trữ dữ liệu tạm thời.

#### Bây giờ chúng ta hãy xem cách ngủ đông sử dụng những thứ này.

#### Hibernate sử dụng bộ đệm cấp một theo mặc định.

#### Bạn phải mã hóa cứng không có gì để sử dụng bộ nhớ cache cấp một.

#### Vì vậy, hãy tìm hiểu cách sử dụng bộ đệm cấp hai.

#### Bộ đệm ẩn cấp hai Hibernate được thiết lập theo hai bước.

#### Trước tiên, bạn phải quyết định sử dụng chiến lược đồng thời nào.

#### Sau đó, bạn định cấu hình hết hạn bộ nhớ cache và các thuộc tính bộ nhớ cache vật lý bằng cách sử dụng nhà cung cấp bộ đệm.

#### Chúng ta sẽ tìm hiểu chi tiết về chiến lược đồng thời và nhà cung cấp Cache trong đơn vị tiếp theo của chương.

#### Hãy nhìn lại những gì chúng ta đã học được cho đến nay.

### **Concurrency Strategies**

#### Một chiến lược đồng thời hoạt động như một trung gian hòa giải.

#### Nó chịu trách nhiệm lưu trữ các mục dữ liệu trong bộ nhớ cache và truy xuất chúng từ bộ đệm.

#### Bạn có thể tưởng tượng nó như một người gọi tên của những người để trở nên khôn ngoan từ một danh sách chờ.

#### Nếu bạn định kích hoạt bộ đệm cấp hai, cho mỗi lớp và bộ sưu tập liên tục, chiến lược đồng thời bộ nhớ cache nào sẽ sử dụng, bạn cần quyết định.

#### Có 4 loại Chiến lược đồng thời:

Transactional – Chiến lược này được sử dụng chủ yếu để đọc dữ liệu trong đó việc ngăn chặn dữ liệu cũ là rất quan trọng trong các giao dịch đồng thời.

Read-write - Nó tương tự như chiến lược giao dịch. Nơi chúng tôi đọc chủ yếu là dữ liệu và ngăn chặn dữ liệu cũ là rất quan trọng ..

Nonstrict-read-write – Chiến lược này đảm bảo không đảm bảo tính nhất quán giữa cơ sở dữ liệu và bộ nhớ cache. Chúng tôi chỉ có thể sử dụng chiến lược này nếu dữ liệu có thể được sửa đổi và một khả năng nhỏ dữ liệu cũ không phải là mối quan tâm quan trọng.

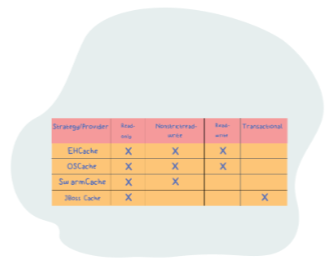
Read-only - Chiến lược này tương thích với dữ liệu không thể sửa đổi. Chúng tôi chỉ có thể sử dụng nó cho dữ liệu tham khảo.

### **Cache Provider**

#### Bước tiếp theo của bạn là sử dụng các lớp ứng cử viên bộ nhớ cache của bạn để chọn nhà cung cấp bộ nhớ cache.

#### Hibernate buộc bạn phải chọn một nhà cung cấp bộ nhớ cache duy nhất cho toàn bộ ứng dụng.

#### Do đó, điều quan trọng là bạn phải chọn cái thích hợp.



#### Không phải tất cả các nhà cung cấp bộ nhớ cache đều đi cùng với tất cả các chiến lược đồng thời.

#### Ma trận tương thích đã cho ở trên sẽ giúp bạn chọn một kết hợp thích hợp.

#### Sau khi thực hiện việc này, chúng tôi đã triển khai thành công bộ đệm cấp hai trong ứng dụng của mình.

### **Query cache**

#### Để sử dụng bộ đệm ẩn truy vấn, trước tiên bạn cần kích hoạt nó bằng cách sử dụng

hibernate.cache.use\_query\_cache="true"

#### tài sản. Điều này có thể được thực hiện trong tệp cấu hình.

#### Bằng cách này, bạn làm cho Hibernate tạo bộ nhớ cache cần thiết trong bộ nhớ để giữ các bộ truy vấn và mã định danh.

#### Tiếp theo, sử dụng phương thức () của lớp Query. setCacheableBoolean

### **How to achieve Batch Processing**

#### Hãy xem xét một tình huống khi bạn cần tải lên nhiều bản ghi vào cơ sở dữ liệu của mình.

#### Tất nhiên chúng ta có thể đạt được nó bằng cách sử dụng mã được đưa ra.

#### Hibernate sẽ lưu trữ tất cả các đối tượng được duy trì trong bộ đệm ẩn cấp phiên.

#### Tại một số điểm, ứng dụng của bạn sẽ rơi xuống với **OutOfMemoryException**.

#### Điều này có thể được giải quyết nếu chúng ta đang sử dụng xử lý hàng loạt với Hibernate.



#### Để sử dụng tính năng xử lý hàng loạt, trước tiên hãy đặt hibernate.jdbc.batch\_size làm kích thước lô.

#### Điều này sẽ cho bộ chứa ngủ đông biết rằng mọi hàng X được chèn dưới dạng một lô.

#### Kích thước lô có thể ở mức 20 hoặc 50 tùy thuộc vào kích thước đối tượng.

#### Các thay đổi sẽ được thực hiện trong hầu hết các bộ tệp cấu hình, tệp ánh xạ và tệp Cấu hình.

#### Và một khi chúng tôi biên dịch tất cả các thay đổi, Hibernate cho phép chúng tôi vượt qua **OutOfMemoryException**.

### **Interceptors**

#### Trong Hibernate, một đối tượng sẽ được tạo ra và tồn tại.

#### Khi đối tượng đã được thay đổi, nó phải được hoàn nguyên về cơ sở dữ liệu.

#### Quá trình này tiếp tục cho đến lần tiếp theo đối tượng là cần thiết.

#### Khi cần, nó có thể được tải từ cửa hàng liên tục.

#### Một vật thể đi qua các giai đoạn khác nhau trong vòng đời của nó.

#### Giao diện đánh chặn cung cấp các phương thức, có thể được gọi ở các giai đoạn khác nhau để thực hiện một số tác vụ cần thiết.

#### Các phương thức này là callback từ session đến ứng dụng.

#### Chúng cho phép ứng dụng kiểm tra và / hoặc thao tác các thuộc tính của một đối tượng liên tục trước khi nó được lưu, cập nhật, xóa hoặc tải.

#### Bây giờ chúng ta hãy xem một số máy bay đánh chặn phổ biến:

* findDirty(): Phương thức này được gọi khi phương thức flush() được gọi trên một đối tượng Session.
* instantiate(): Phương thức này được gọi khi một lớp persisted được khởi tạo.
* isUnsaved(): Phương thức này được gọi khi một đối tượng được truyền vào phương thức saveOrUpdate().
* onDelete(): Phương thức này được gọi trước khi một đối tượng bị xóa.
* onLoad(): Phương thức này được gọi trước khi một đối tượng được khởi tạo.
* onSave(): Phương thức này được gọi trước khi một đối tượng được lưu.
* preFlush(): Phương thức này được gọi trước khi tuôn ra.

#### Hibernate Interceptor cung cấp cho chúng ta toàn quyền kiểm soát cách một đối tượng sẽ trông như thế nào đối với cả ứng dụng và cơ sở dữ liệu.

#### Để chế tạo một máy bay đánh chặn, bạn có thể triển khai trực tiếp một lớp Interceptor hoặc mở rộng lớp EmptyInterceptor.

#### Đó là tất cả những gì bạn cần cho Interceptor.

### **JPA so với Hibernate**

#### Chúng tôi gần như đã đi đến cuối khóa học.

#### Bây giờ chúng ta hãy thảo luận về sự khác biệt chính giữa Java Persistence API và Hibernate.

#### JPA (Java Persistence API) là một đặc tả của Java.

#### Nó được sử dụng để truy cập, quản lý và duy trì dữ liệu giữa các đối tượng Java và cơ sở dữ liệu quan hệ.

#### Nó được coi là một cách tiếp cận tiêu chuẩn cho Object Relational Mapping.

#### Là một đặc điểm kỹ thuật, JPA không tự thực hiện bất kỳ thao tác nào.

#### Vì vậy, nó đòi hỏi phải thực hiện.

#### Vì vậy, các công cụ ORM như Hibernate, TopLink và iBatis thực hiện các thông số kỹ thuật JPA để duy trì dữ liệu.

#### Hibernate là một triển khai của JPA.

#### Vì vậy, nó tuân theo các tiêu chuẩn chung được cung cấp bởi JPA.

### **Need of JPA**

#### JPA là một đặc điểm kỹ thuật.

#### Nó cung cấp nguyên mẫu và chức năng phổ biến cho các công cụ ORM.

#### Tất cả các công cụ ORM (như Hibernate, TopLink, iBatis) đều tuân theo các tiêu chuẩn chung.

#### Nếu chúng ta muốn chuyển ứng dụng của mình từ công cụ ORM này sang công cụ ORM khác, chúng ta có thể thực hiện dễ dàng.

### **JPA so với Hibernate**

#### Java Persistence API (JPA) định nghĩa việc duy trì dữ liệu quan hệ trong các ứng dụng Java.

#### Hibernate, mặt khác, là một công cụ ORM được sử dụng để lưu trạng thái của một đối tượng Java vào cơ sở dữ liệu.

#### Nó chỉ là một đặc điểm kỹ thuật, các công cụ ORM khác nhau thực hiện nó để duy trì dữ liệu.

#### Hibernate là một trong những triển khai JPA được sử dụng thường xuyên nhất.

#### Để tương tác với nhà máy quản lý thực thể, JPA sử dụng giao diện EntityManagerFactory.

#### Mặc dù, Hibernate sử dụng giao diện SessionFactory để tạo các phiên bản Phiên.

#### JPA sử dụng Java Persistence Query Language trong khi Hibernate sử dụng HQL.

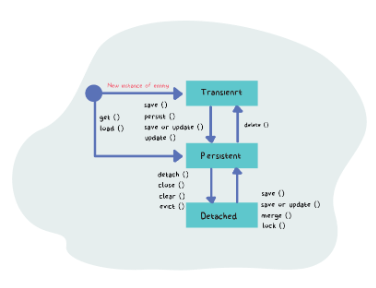
### **Hibernate Lifecycle**

#### Trong Hibernate, hoặc chúng ta tạo một đối tượng của một thực thể và lưu nó vào cơ sở dữ liệu.

#### Chúng tôi lấy dữ liệu của một thực thể từ cơ sở dữ liệu.

#### Ở đây, mỗi thực thể được liên kết với vòng đời.

#### Đối tượng thực thể đi qua các giai đoạn khác nhau của vòng đời.



#### Vòng đời Hibernate chứa các trạng thái sau:

* Transient state
* Persistent state
* Detached state

### **Transient state**

* Trạng thái thoáng qua là trạng thái ban đầu của một đối tượng.
* Khi chúng ta tạo một thể hiện của lớp POJO, sau đó đối tượng đi vào trạng thái quá độ.
* Ở đây, một đối tượng không được liên kết với Phiên. Vì vậy, trạng thái tạm thời không liên quan đến bất kỳ cơ sở dữ liệu nào.
* Do đó, các sửa đổi trong dữ liệu không ảnh hưởng đến bất kỳ thay đổi nào trong cơ sở dữ liệu.
* Các đối tượng thoáng qua tồn tại trong bộ nhớ đống. Chúng độc lập với Hibernate.

### **Persistent state**

* Ngay sau khi đối tượng liên quan đến Phiên, nó đã bước vào trạng thái dai dẳng.
* Do đó, chúng ta có thể nói rằng một đối tượng ở trạng thái bền bỉ khi chúng ta lưu hoặc duy trì nó.
* Ở đây, mỗi đối tượng đại diện cho hàng của bảng cơ sở dữ liệu.
* Vì vậy, các sửa đổi trong dữ liệu tạo ra những thay đổi trong cơ sở dữ liệu.

#### Chúng ta có thể sử dụng bất kỳ phương pháp nào sau đây cho trạng thái liên tục.

### **Detached State**

* Khi chúng ta đóng phiên hoặc xóa bộ nhớ cache của nó, thì đối tượng sẽ chuyển sang trạng thái tách rời.
* Vì một đối tượng không còn được liên kết với Phiên, các sửa đổi trong dữ liệu không ảnh hưởng đến bất kỳ thay đổi nào trong cơ sở dữ liệu.
* Tuy nhiên, đối tượng tách rời vẫn có một đại diện trong cơ sở dữ liệu.
* Nếu chúng ta muốn duy trì các thay đổi được thực hiện cho một đối tượng tách rời, cần phải gắn lại ứng dụng vào một phiên Hibernate hợp lệ.
* Để liên kết đối tượng tách rời với phiên ngủ đông mới, hãy sử dụng bất kỳ phương thức nào sau đây - load(), merge(), refresh(), update() hoặc save() trên một phiên mới với tham chiếu của đối tượng tách rời.

#### Chúng ta có thể sử dụng bất kỳ phương pháp nào sau đây cho trạng thái tách rời.

* session.close();
* session.clear();
* session.detach(e);
* session.evict(e);