Session 02:

JAVA HIBERNATE Programming

# ORM là gì?

ORM là viết tắt của Object-Relational Mapping (ORM) là một kỹ thuật lập trình để chuyển đổi dữ liệu giữa các cơ sở dữ liệu quan hệ và các ngôn ngữ lập trình hướng đối tượng như Java, C# ... Một hệ thống ORM có những ưu điểm so vơi JDBC như sau:

* Ko cần viết SQL mà tương tác thông qua class đối tượng
* Tần ORM chuyển mã nguồn về thành truy vấn SQL gọi JDBC thực hiện
* Từng đối tượng thông qua nền tảng ORM ánh xạ CSDL(nên ko quan tâm, phụ thuộc sử dụng data nào)
* DatabaseFirst(có data trước-> viết code ko tác động đến csdl)
* CodeFirst(ko tạo csdl rỗng ->dựa code class object ánh xạ sinh ra lược đồ csdl)

Diagram

Description automatically generated

|  |  |
| --- | --- |
| **No.** | **Ưu điểm** |
| 1 | Cho phép **đối tượng truy cập mã nghiệp vụ** thay vì bảng DB. |
| 2 | **Ẩn** các chi tiết của các **truy vấn SQL từ logic**. |
| 3 | Dựa trên JDBC 'under the hood' |
| 4 | **Không cần** phải đối phó với việc **thực hiện cơ sở dữ liệu**. |
| 5 | Các thực thể dựa trên các khái niệm nghiệp vụ thay vì cấu trúc cơ sở dữ liệu. |
| 6 | Quản lý Transaction và tạo ra key tự động. |
| 7 | Phát triển ứng dụng nhanh chóng. |

Một giải pháp ORM bao gồm bốn thực thể sau:

|  |  |
| --- | --- |
| **No.** | **Giải pháp** |
| 1 | Một API để thực hiện các thao tác CRUD cơ bản trên **các đối tượng của các lớp persistent.** |
| 2 | Một ngôn ngữ hoặc API để chỉ định các **truy vấn đề cập đến các lớp và thuộc tính của các lớp**. |
| 3 | Một phương tiện có thể cấu hình để chỉ định **siêu dữ liệu ánh xạ**. |
| 4 | Một kỹ thuật tương tác với các đối tượng giao dịch để thực hiện dirty checking, lazy loading, join fetching và các chức năng tối ưu hóa khác. |

# Tại sao ORM (Object Relational Mapping)?

Khi chúng ta làm việc với một hệ thống hướng đối tượng, có một sự không khớp giữa mô hình đối tượng và cơ sở dữ liệu quan hệ. RDBMS thể hiện dữ liệu trong một định dạng dạng bảng, trong khi các ngôn ngữ hướng đối tượng, chẳng hạn như Java hoặc C #, đại diện cho nó như một đồ thị kết nối của các đối tượng. Xem xét lớp Java sau đây với các constructor phù hợp và phưng thức public liên quan:

[?](https://viettuts.vn/hibernate/orm-la-gi)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | **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;     }  } |

Xem xét các đối tượng trên cần phải được lưu trữ và truy xuất vào bảng RDBMS sau:

[?](https://viettuts.vn/hibernate/orm-la-gi)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **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 phải sửa đổi thiết kế của cơ sở dữ liệu của chúng ta sau khi đã phát triển vài trang hoặc ứng dụng?

Thứ hai, Việc tải và lưu trữ các đối tượng trong một cơ sở dữ liệu quan hệ làm cho chúng ta gặp 5 vấn đề không khớp.

|  |  |
| --- | --- |
| **Mismatch** | **Mô tả** |
| Tính chi tiết | Đôi khi bạn sẽ có một **mô hình đối tượng có nhiều lớp** hơn số lượng các bảng tương ứng trong cơ sở dữ liệu. |
| Tính kế thừa | RDBMS không định nghĩa bất cứ điều gì tương tự để kế thừa đó là một mô hình **tự nhiên trong các ngôn ngữ lập trình hướng đối tượng**. |
| Identity | Một RDBMS định nghĩa chính xác một khái niệm về 'sự giống nhau': khóa chính. Java, tuy nhiên, định nghĩa cả nhận dạng đối tượng (a == b) và đối tượng bằng nhau (a.equals (b)). |
| Mối quan hệ kết hợp | Các ngôn ngữ **hướng đối tượng biểu diễn cho các sự kết hợp sử dụng các đối tượng tham chiếu**, trong khi RDBMS biểu diễn mối quan hệ kết hợp bằng một cột khoá ngoại. |
| Điều hướng | Các cách bạn truy cập các đối tượng trong Java và trong một RDBMS về cơ bản là khác nhau. |

ORM (Object Relational Mapping) là một giải pháp để giải quyết các vấn đề trên.

# Giới thiệu về Hibernate

Hibernate ra đời năm 2001 bởi nhà sáng lập Gavin King như một sự thay thế cho EJB2 kiểu thực thể bean. Hiện nay, phiên bản mới nhất của Hibernate là 5.2.10.Final

# Hibernate Framework là gì?

Hibernate framework là một giải pháp ORM (Object Relational Mapping) mã nguồn mở, gọn nhẹ. Hibernate giúp đơn giản hoá sự phát triển của ứng dụng java để tương tác với cơ sở dữ liệu.

Tool ORM giúp đơn giản hoá việc tạo ra dữ liệu, thao tác dữ liệu và truy cập dữ liệu. Đó là một kỹ thuật lập trình để ánh xạ đối tượng vào dữ liệu được lưu trữ trong cơ sở dữ liệu.



Các tool ORM sử dụng JDBC API để tương tác với cơ sở dữ liệu.

# Lợi ích của Hibernate Framework

Hibernate Framework có các lợi ích như dưới đây:

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

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

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

**4. Tạo bảng tự động:** Hibernate framework cung cấp phương tiện để tạo ra các bảng cơ sở dữ liệu tự động. Vì vậy, không cần phải tạo ra các bảng trong cơ sở dữ liệu bằng tay.

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

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

# Kiến trúc Hibernate

## Kiến trúc Hibernate

Kiến trúc Hibernate bao gồm nhiều đối tượng như đối tượng persistent, session factory, transaction factory, connection factory, session, transaction,...

Có 4 tầng trong kiến trúc hibernate đó là tầng ứng dụng java, tầng hibernate framework, tầng backhand api và tầng database. Dưới đây là sơ đồ kiến trúc hibernate:



Dưới đây là một hình ảnh chi tiết về Kiến trúc ứng dụng Hibernate với vài lớp core quan trọng.

## Các thành phần trong kiến trúc Hibernate

Để tạo ứng dụng hibernate đầu tiên, chúng ta phải biết các thành phần của kiến trúc Hibernate. Chúng là như sau:

### **Đối tượng Configuration(chỉ cần gọi 1 lần di nhất)**

Đối tượng Configuration là đối tượng Hibernate đầu tiên bạn tạo trong bất kỳ ứng dụng Hibernate nào và chỉ cần tạo một lần trong quá trình khởi tạo ứng dụng. Nó đại diện cho **một tập tin cấu hình hoặc thuộc tính yêu cầu của Hibernate**. Đối tượng Configuration cung cấp hai thành phần chính:

* **Database Connection:** Thao tác này được xử lý thông qua một hoặc nhiều tệp cấu hình được Hibernate hỗ trợ. Các tệp này là hibernate.properties và hibernate.cfg.xml.
* **Class Mapping Setup:** Thành phần này tạo ra kết nối giữa các lớp Java và các bảng cơ sở dữ liệu.

### **Đối tượng SessionFactory**

Đối tượng **Configuration được sử dụng để tạo ra một đối tượng SessionFactory** mà cấu hình Hibernate cho ứng dụng bằng cách **sử dụng tệp tin cấu hình được cung cấp và cho phép một đối tượng Session được khởi tạo**. SessionFactory là một đối tượng **luồng an toàn và được sử dụng bởi tất cả các luồng** của một ứng dụng.

SessionFactory là đối tượng nặng nên thường nó được tạo ra trong quá trình khởi động ứng dụng và lưu giữ để sử dụng sau này. Bạn sẽ cần một đối tượng SessionFactory cho mỗi cơ sở dữ liệu bằng cách sử dụng một tập tin 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**

Một session được sử dụng để có được một **kết nối vật lý với một cơ sở dữ liệu**. Đối tượng Session là nhẹ và được thiết kế để được tạo ra thể hiện mỗi khi 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 mở trong một thời gian dài bởi vì chúng thường không phải là luồng an toàn và chúng cần được tạo ra và được đóng khi cần thiết.

### **Đối tượng Transaction**

Một Transaction **đại diện cho một đơn vị làm việc với cơ sở dữ liệu** và hầu hết các RDBMS hỗ trợ chức năng transaction. Các transaction trong Hibernate được xử lý bởi trình quản lý transaction và transaction (từ JDBC hoặc JTA).

Đây là một đối tượng tùy chọn và các ứng dụng Hibernate có thể chọn không sử dụng interface này, thay vào đó quản lý transaction trong code ứng dụng riêng.

### **Đối tượng Query**

Các đối tượng Query sử dụng chuỗi truy vấn SQL hoặc **Hibernate Query Language** (HQL) để lấy dữ liệu từ cơ sở dữ liệu và tạo các đối tượng. Ví dụ truy vấn được sử dụng để ràng buộc các tham số truy vấn, giới hạn số lượng kết quả được trả về bởi truy vấn và cuối cùng thực hiện truy vấn.

### **Đối tượng Criteria**

Đối tượng Criteria được **sử dụng để tạo và thực hiện truy vấn** các tiêu chí định hướng đối tượng để lấy các đối tượng.



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

# Môi trường Hibernate

## Download Hibernate

Giả sử bạn đã có phiên bản Java mới nhất được cài đặt trên máy của bạn. Sau đây là các bước đơn giản để tải xuống và cài đặt Hibernate trên máy của bạn.

* Thực hiện một sự lựa chọn cho dù bạn muốn cài đặt Hibernate trên Windows hoặc Unix và sau đó tiến hành bước tiếp theo để tải xuống tệp .zip cho các cửa sổ và tệp .tz cho Unix.
* Tải về phiên bản mới nhất của Hibernate từ http://www.hibernate .org/downloads

## Cài đặt Hibernate

Khi bạn đã tải xuống và giải nén phiên bản mới nhất của tệp Cài đặt Hibernate, bạn cần phải thực hiện hai bước đơn giản sau đây. 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.

* Bây giờ sao chép tất cả các tập tin thư viện từ /lib vào CLASSPATH của bạn, và thay đổi biến classpath của bạn để bao gồm tất cả các JARs:
* Cuối cùng sao chép tập tin hibernate[x].jar vào CLASSPATH của bạn. Tập tin này nằm trong thư mục gốc của tiến trình cài đặt và là tệp JAR chính mà Hibernate cần để thực hiện công việc của nó.

## Cài đặt thư viện liên quan đến Hibernate

Dưới đây là danh sách các gói/thư viện được yêu cầu bởi Hibernate và bạn nên cài đặt chúng trước khi bắt đầu với Hibernate. Để cài đặt những gói này, bạn phải sao chép các tập tin thư viện từ /lib thành CLASSPATH của bạn và thay đổi biến CLASSPATH của bạn cho phù hợp.

|  |  |
| --- | --- |
| **No.** | **Packages/Libraries** |
| 1 | **dom4j** - XML parsing [www.dom4j.org/](http://www.dom4j.org/) |
| 2 | **Xalan** - XSLT Processor <http://xml.apache.org/xalan-j/> |
| 3 | **Xerces** - The Xerces Java Parser <http://xml.apache.org/xerces-j/> |
| 4 | **cglib** - Appropriate changes to Java classes at runtime <http://cglib.sourceforge.net/> |
| 5 | **log4j** - Logging Faremwork <http://logging.apache.org/log4j> |
| 6 | **Commons** - Logging, Email etc. <http://jakarta.apache.org/commons> |
| 7 | **SLF4J** - Logging Facade for Java [http://www.slf4j.org](http://www.slf4j.org/download.html) |

## Cài đặt Hibernate bằng Maven

Nếu dự án của bạn được quản lý bằng maven bạn có thể thêm dependency thích hợp để tải thư viện Hibernate tương ứng.

**Bước 1:** Chuẩn bị cơ sở dữ liệu

**Bước 2:** Tạo Project Maven + Thêm các thư viện:

**Pom.xml**

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

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

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.6.15.Final</version>

</dependency>

<https://mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.32>

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

<dependency>

<groupId>mysql</groupId>

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

<version>8.0.32</version>

</dependency>

# <https://gpcoder.com/6327-cai-dat-va-su-dung-hibernate/>

**Bước 03:** Tạo Entity

Lưu ý: Anotation

@Entity: khai báo đối tượng ánh xạ đến csdl

@Table: thuộc tính bảng

@inheritance: superclass dùng lưu các đối tượng

@Id: khóa chính

@Colomn: cột

@Transient: lưu xuống csdl

@JoinColumn: khai báo khóa ngoại

@OneToMany: quan hệ one to many

@ManyToOne: quan hệ many to one

@ManyToMany: many to many

Lưu ý: anotation

**Persistent Class**

Các class mà đối tượng của nói sẽ được lưu trữ xuống csdl còn gọi POJO(Plain Old Java Object)

Có phương thức khởi tạo ko tham số

Các thuộc tính private, có get và set

**Category.java**

**Graphical user interface, text, application

Description automatically generated**

**Graphical user interface, application

Description automatically generated**

@Entity

@Table(name="category")

**public** **class** Category **implements** Serializable{//đồng bộ Data trên server

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

**private** String name;

**private** String description;

**public** **int** getId() {

**return** id;

}

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

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getDescripttion() {

**return** description;

}

**public** **void** setDescripttion(String description) {

**this**.description = description;

}

}

**Bước 4:** Cấu **hình Configuration (hibernate.cfg.xml)**

**Configuration**: đại diện cho các cấu hình hay thuộc tính để hibernate sử dụng, nó được tạo đầu tiên và chỉ tạo 1 lần lúc khởi động

* C1. Tập tin XML: **hibernate.cfg.xml** đặt thư mục gốc của project (resources).

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

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

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

<hibernate-configuration>

<session-factory>

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

com.mysql.jdbc.Driver</property>

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

jdbc:mysql://localhost:3306/saledb</property>

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

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

org.hibernate.dialect.MySQLDialect</property>

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

</session-factory>

</hibernate-configuration>

HibernateUtils.java

**public** **class** HibernateUtils {

// muốn hằng số tạo ra 1 lần dùng chung static final

**private** **static** **final** SessionFactory ***FACTORY***;//tạo getFactory

//khối tĩnh này chạy 1 lần

**static** {

Configuration conf=**new** Configuration();

//C1. cung cấp thông tin cấu hình file xml

conf.configure("hibernate.cfg.xml");

//Khai bao tồn tại Category

conf.addAnnotatedClass(Category.**class**);

//ServiceRegistry lớp trừ tượng ko thay đổi theo thời gian

ServiceRegistry registry= **new** StandardServiceRegistryBuilder().applySettings(conf.getProperties()).build();

***FACTORY*** =conf.buildSessionFactory(registry);

}

**public** **static** SessionFactory getFactory() {

**return** ***FACTORY***;

}

}

**Bước 5:** Tạo file HibernateUtil

* C2. Cấu hình bằng mã nguồn Java

HibernateUtils.java

**public** **class** HibernateUtils {

// muốn hằng số tạo ra 1 lần dùng chung static final

**private** **static** **final** SessionFactory ***FACTORY***;

//khối tĩnh này chạy 1 lần

**static** {

Configuration conf=**new** Configuration();

//code cung cấp thông tin cấu hình

Properties props =**new** Properties();

props.put(Environment.***DIALECT***,"org.hibernate.dialect.MySQLDialect");

props.put(Environment.***DRIVER***,"com.mysql.cj.jdbc.Driver");

props.put(Environment.***URL***,"jdbc:mysql://localhost:3306/saledb");

props.put(Environment.***USER***,"root");

props.put(Environment.***PASS***,"");

props.put(Environment.***SHOW\_SQL***,"true");

conf.setProperties(props);

//Khai bao tồn tại Category

conf.addAnnotatedClass(Category.**class**);

//ServiceRegistry lớp trừ tượng

ServiceRegistry registry= **new** StandardServiceRegistryBuilder().applySettings(conf.getProperties()).build();

***FACTORY*** =conf.buildSessionFactory(registry);

}

**public** **static** SessionFactory getFactory() {

**return** ***FACTORY***;

}

}

Bước 6: Test

**public** **class** Test {

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

// **TODO** Auto-generated method stub

//Tạo Session cho mỗi truy vấn từ Factory

Session session=HibernateUtils.*getFactory*().openSession();

//Them

// Category c= new Category();//transient

// c.setName("May In");

// c.setDescripttion("Chi tiết cấu hình ");

// session.save(c);

//Update

// Category c=session.get(Category.class, 1); //persistent lấy lên

// c.setDescripttion("Dữ liệu mới cập nhât");

//

// session.getTransaction().begin();//Tranh chấp ghi dữ liệu phải có GIAO TAC (bật tuong tac có 2 client cùng thực hiện)

// session.save(c);

// session.getTransaction().commit();

//Delete

// Category c=session.get(Category.class, 1); //persistent lấy lên

//

// session.getTransaction().begin();//Tranh chấp ghi dữ liệu phải có GIAO TAC (bật tuong tac có 2 client cùng thực hiện)

// session.delete(c);

// session.getTransaction().commit();

//Truy vấn HQL All

// Query q=session.createQuery("FROM Category"); //HQL of Class

// List<Category> cates=q.getResultList();

// cates.forEach(c->System.out.printf("%d - %s \n", c.getId(), c.getName()));

//Truy vấn HQL Id

Query q=session.createQuery("FROM Category WHERE id=5"); //HQL of Class

List<Category> cates=q.getResultList();

cates.forEach(c->System.***out***.printf("%d - %s \n", c.getId(), c.getName()));

session.close();

}

}

**Session**

* Persistent Object có 1 trong các trạng thái
* Transisent: vừa tạo, chưa kết hợp với Sesion nào, chưa đại diện cho record
* Persistent: có kết hợp session và đại diejejn 1 record
* Detached: khi đóng Session
* Vòng đời của Hibernate

Diagram

Description automatically generated

* Một đối tượng trong Hibernate có 1 trong 4 trạng thái:
* **Transient**    (Tạm thời)
* **Persistent**   (Bền vững)
* **Removed**     (Đã bị xóa – dưới DB)
* **Detached**    (Đã bị tách riêng ra – so với session hiện tại)

1. Trường hợp bạn tạo mới một đối tượng java từ một Entity, đối tượng đó có tình trạng là ***Transient***. Hibernate không biết về sự tồn tại của nó. Nó nằm ngoài sự quản lý của Hibernate.
2. Trường hợp bạn lấy ra đối tượng Entity bằng method get, load hoặc find, bạn có được một đối tượng nó tương ứng với 1 record dưới database. Đối tượng này có trạng thái ***Persistent***. Nó được quản lý bởi Hibernate.
3. Session gọi một trong các method save,saveOrUpdate, persist, merge sẽ đẩy đối tượng ***Transient***vào sự quản lý của Hibernate và đối tượng này chuyển sang trạng thái ***Persistent***. Tùy tình huống nó sẽ insert hoặc update dữ liệu vào DB.
4. Session gọi evict(..) hoặc clear() để đuổi các đối tượng có trạng thái persistent (bền vững) ra khỏi sự quản lý của Hibernate, giờ các đối tượng này sẽ có trạng thái mới là Detached (Bị tách ra).  Nếu nó không được đính (Attached) trở lại, nó sẽ bị bộ gom rác của Java quét đi theo cơ chế thông thường.
5. Sử dụng update(..), saveOrUpdate(..), merge(..) sẽ đính trở lại các đối tượng Detached vào lại. Tùy tình huống nó sẽ tạo ra dưới DB câu lệnh update hoặc insert. Các đối tượng sẽ trở về trạng thái Persistent (bền vững).
6. Session gọi method remove(..), delete(..) để xóa một bản ghi, đối tượng persistent giờ chuyển sang trạng thái Removed (Đã bị xóa)

**Các thành phần khác**

* Transaction: giao tác
* Query: truy vấn giwuax csdl và tạo đối tượng (SQL/HQL)
* Criteria: tạo và thực thi các truy vấn có đk

## Annotation: ánh xạ lớp java tới các bảng csdl ko cần tập tin xml, viết trực tiếp POJO, gói javax.persistence

Catology

Graphical user interface, application, website

Description automatically generated

product

Graphical user interface

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated

## OneToMany

Category

**public** **class** Category **implements** Serializable{

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

**private** String name;

**private** String description;

@OneToMany(mappedBy = "category")//mặc định fetch = FetchType.LAZY

**private** List<Product>products;//1 khai bào list product trùng lặp lại

chuyển List->Set: ko lặp lại product

@Entity

@Table(name="category")

**public** **class** Category **implements** Serializable{

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

**private** String name;

**private** String description;

@OneToMany(mappedBy = "category")//mặc định fetch = FetchType.LAZY

**private** Set<Product>products;//1 khai bào list product

**public** **int** getId() {

**return** id;

}

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

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getDescripttion() {

**return** description;

}

**public** **void** setDescripttion(String description) {

**this**.description = description;

}

**public** String getDescription() {

**return** description;

}

**public** **void** setDescription(String description) {

**this**.description = description;

}

**public** Set<Product> getProducts() {

**return** products;

}

**public** **void** setProducts(Set<Product> products) {

**this**.products = products;

}

}

## ManyToOne

@Entity

@Table(name="product")

**public** **class** Product **implements** Serializable{

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

**private** String name;

**private** String description;

**private** BigDecimal price;

**private** String image;

@Column(name="created\_date")

@Temporal(javax.persistence.TemporalType.***DATE***)

**private** Date createDate;

**private** **boolean** active;

// Many to one

@ManyToOne(fetch = FetchType.***LAZY***)// tối ưu ko join ngươc lại mac định product left out join catology)

@JoinColumn(name="category\_id")

**private** Category category;

**public** **int** getId() {

**return** id;

}

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

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getDescription() {

**return** description;

}

**public** **void** setDescription(String description) {

**this**.description = description;

}

**public** BigDecimal getPrice() {

**return** price;

}

**public** **void** setPrice(BigDecimal price) {

**this**.price = price;

}

**public** String getImage() {

**return** image;

}

**public** **void** setImage(String image) {

**this**.image = image;

}

**public** Date getCreateDate() {

**return** createDate;

}

**public** **void** setCreateDate(Date createDate) {

**this**.createDate = createDate;

}

**public** **boolean** isActive() {

**return** active;

}

**public** **void** setActive(**boolean** active) {

**this**.active = active;

}

**public** Category getCategory() {

**return** category;

}

**public** **void** setCategory(Category category) {

**this**.category = category;

}

}

HibernateUtils.java

**public** **class** HibernateUtils {

// muốn hằng số tạo ra 1 lần dùng chung static final

**private** **static** **final** SessionFactory ***FACTORY***;

//khối tĩnh này chạy 1 lần

**static** {

Configuration conf=**new** Configuration();

//code cung cấp thông tin cấu hình

Properties props =**new** Properties();

props.put(Environment.***DIALECT***,"org.hibernate.dialect.MySQLDialect");

props.put(Environment.***DRIVER***,"com.mysql.cj.jdbc.Driver");

props.put(Environment.***URL***,"jdbc:mysql://localhost:3306/saledb");

props.put(Environment.***USER***,"root");

props.put(Environment.***PASS***,"");

props.put(Environment.***SHOW\_SQL***,"true");

conf.setProperties(props);

////cung cấp thông tin cấu hình file xml

// conf.configure("hibernate.cfg.xml");

conf.addAnnotatedClass(Category.**class**);//Khai bao tồn tại Category

conf.addAnnotatedClass(Product.**class**);//Khai bao tồn tại Product

//ServiceRegistry lớp trừ tượng

ServiceRegistry registry= **new** StandardServiceRegistryBuilder().applySettings(conf.getProperties()).build();

***FACTORY*** =conf.buildSessionFactory(registry);

}

**public** **static** SessionFactory getFactory() {

**return** ***FACTORY***;

}

}

Test.java

**public** **class** Test {

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

// **TODO** Auto-generated method stub

Session session=HibernateUtils.*getFactory*().openSession();

//Add Prodect

// Product p=new Product();

// p.setName("Ipad Pro 2023");

// p.setPrice(new BigDecimal(300000000));

// Category c=session.get(Category.class, 9);

// p.setCategory(c);

//

// session.save(p);

//Prodect for id ( Test @ManyToOne(fetch = FetchType.LAZY))

// Product p=session.get(Product.class, 2);

// System.out.println(p.getName());

//Truy vấn for id test @OneToMany(mappedBy = "category", fetch= FetchType.EAGER).

Category c=session.get(Category.**class**, 9); //persistent lấy lên

c.getProducts().forEach(p->System.***out***.printf("%d - %s \n", p.getId(), p.getName()));

session.close();

}

}

## ManyToMany

<https://stackjava.com/hibernate/code-vi-du-hibernate-manytomany-quan-he-nhieu-nhieu.html>

## HQL Query

Truy vấn hướng đối tượng, là đối tượng persistent: độc lập với csdl, có đtrưng của hướng đối tượng: Kthua, đa hình

**FORM**

String hql = "FROM Employee";

Query query = session.createQuery(hql);

List results = query.list();

**SELECT**

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

Query query = session.createQuery(hql);

List results = query.list();

**WHERE**

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

Query query = session.createQuery(hql);

List results = query.list();

**ORDER BY**

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**

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

             "GROUP BY E.firstName";

Query query = session.createQuery(hql);

List results = query.list();

**Sử dụng các tham số đặt tên**

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**

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**

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**

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);

**Các phương thức tổng hợp**

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

Query query = session.createQuery(hql);

List results = query.list();

**Phân trang bằng việc sử dụng Query**

String hql = "FROM Employee";

Query query = session.createQuery(hql);

query.setFirstResult(1);

query.setMaxResults(10);

List results = query.list();

 xây dựng một thành phần phân trang trong ứng dụng Web hoặc Swing

## [Restrictions với Criteria](https://viettuts.vn/hibernate/hibernate-criteria-queries)

ví dụ để bổ sung điều kiện để trả các record có salary là 2000:

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

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

List results = cr.list();

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

// Lấy các record có salary lớn hơn 2000

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

// Lấy các record có salary nhỏ hơn 2000

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

// Lấy các record có fistName bắt đầu là zara

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

// Dạng không phân biệt chữ hoa và chữ thường cho điều kiện trên.

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

// Lấy các record có salary trong khoảng 1000 và 2000

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

// Check thuộc tính đã cho là null

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

// Check thuộc tính đã cho là khác null

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

// Check thuộc tính đã cho là empty

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

// Check thuộc tính đã cho là khác empty

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

AND và OR

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

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

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

// Lấy các record phù hợp với điều kiện OR

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

cr.add(orExp);

// Lấy các record phù hợp với điều kiện AND

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

cr.add(andExp);

List results = cr.list();

Bạn có thể mở rộng để lấy 10 hàng một lúc

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

cr.setFirstResult(1);

cr.setMaxResults(10);

List results = cr.list();

 lớp Order để sắp xếp tập kết quả

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

// Lấy các record có salary lớn hơn 2000

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

// Sắp xếp các record theo thứ tự giảm dần

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

// Sắp xếp các record theo thứ tự tăng dần

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

List results = cr.list();

giá trị trung bình, maximum và minimum của các giá trị thuộc tính

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

// Lấy tổng số hàng.

cr.setProjection(Projections.rowCount());

// Lấy giá trị trung bình của một thuộc tính.

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

// Để có số lần xuất hiện duy nhât của một thuộc tính.

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

// Lấy giá trị maximum của một thuộc tính.

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

// Lấy giá trị minimum của một thuộc tính.

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

// Lấy tổng các giá trị của một thuộc tính

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