

LẬP TRÌNH JAVA

HIBERNATE

*ThS. Dương Hữu Thành
Khoa CNTT, Đại học Mở Tp.HCM
thanh.dh@ou.edu.vn*



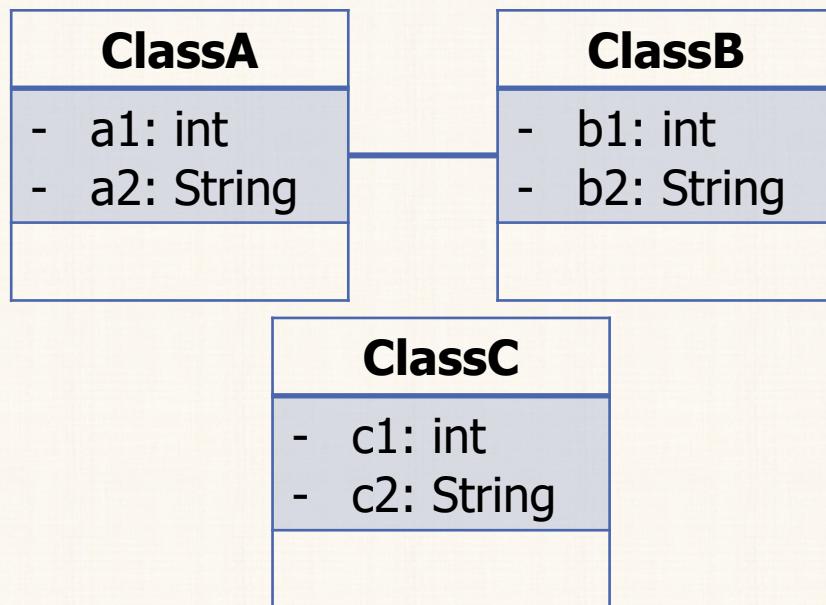


Nội dung chính

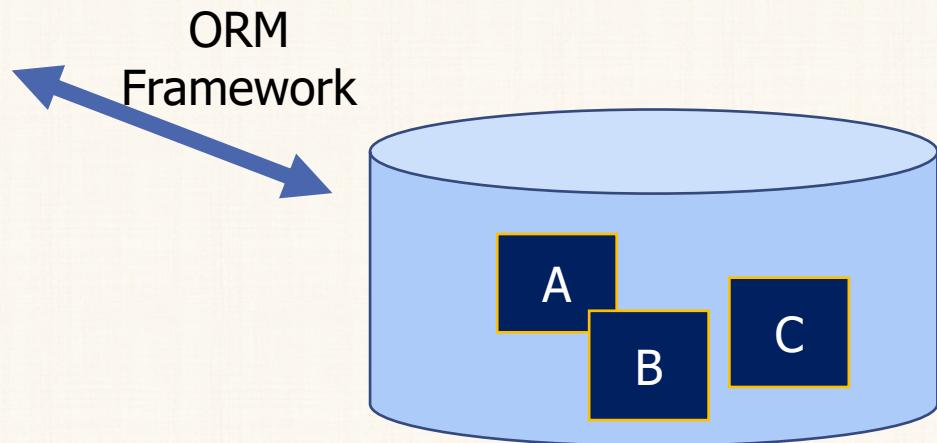
- 1. Giới thiệu ORM**
- 2. Giới thiệu Hibernate**
- 3. Kiến trúc Hibernate**
- 4. HQL Query**
- 5. Sử dụng Annotation**
- 6. Criteria Query API**

Giới thiệu ORM

- ORM (Object-Relational Mapping) là một kỹ thuật lập trình chuyển dữ liệu giữa các cơ sở dữ liệu quan hệ và các ngôn ngữ lập trình hướng đối tượng như Java, C#, v.v.



Các lớp đối tượng



Cơ sở dữ liệu quan hệ



Giới thiệu ORM

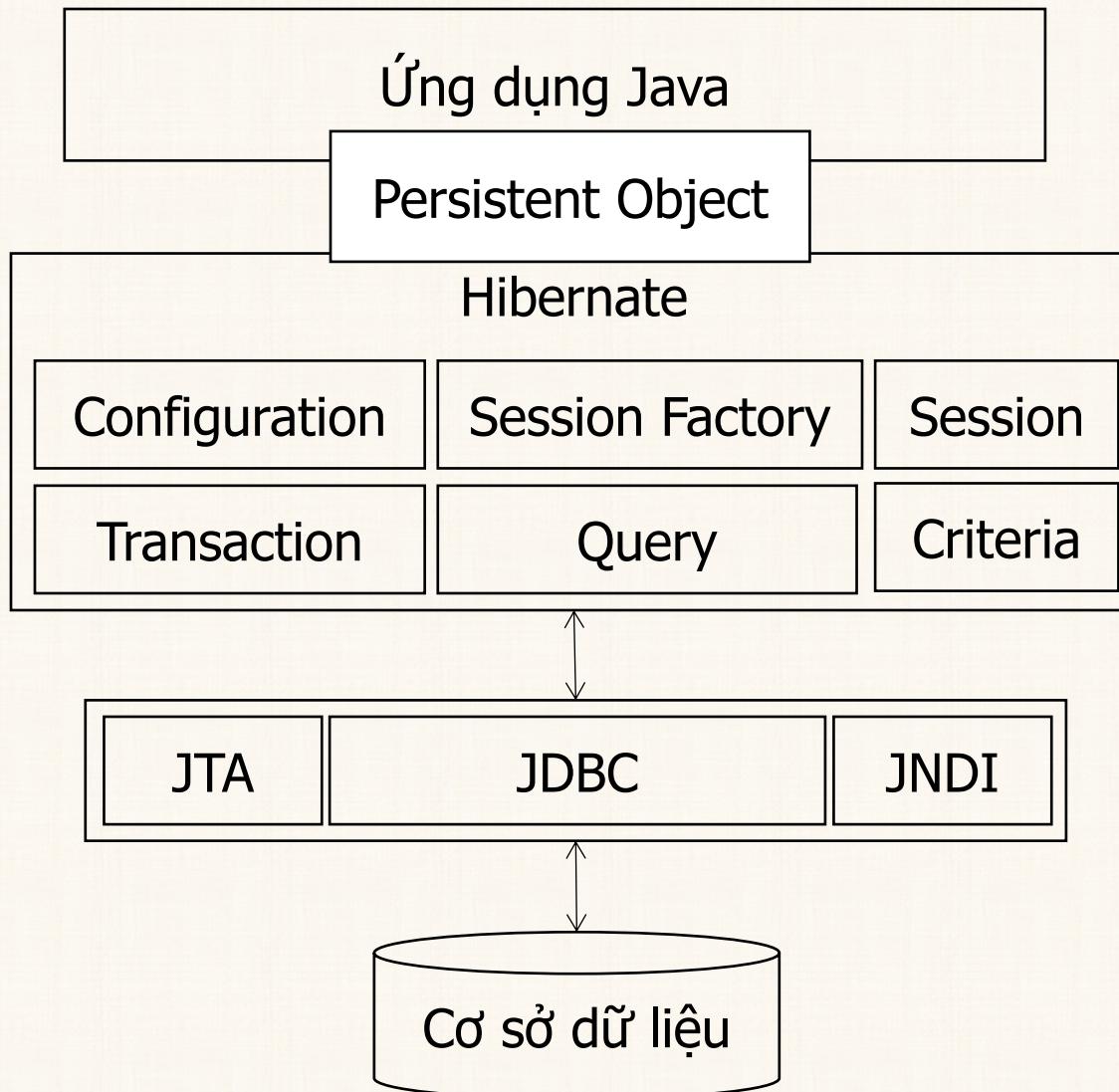
- ORM giúp lập trình viên chỉ tập trung vào vấn đề logic nghiệp vụ cần xử lý.
- Tương tác với đối tượng nhiều hơn là tương tác với các bảng trong cơ sở dữ liệu quan hệ
- Truy vấn dữ liệu thông qua các logic hướng đối tượng, không trực tiếp viết các truy vấn SQL.



Giới thiệu Hibernate

- Hibernate là một framework mã nguồn mở được phát triển bởi **Gavin King** năm 2001.
- Hibernate là một giải pháp ORM mạnh mẽ, hiệu năng cao của Java.
- Hibernate hỗ trợ hầu hết các cơ sở dữ liệu quan hệ như MySQL, PostgreSQL, MS SQL Server, DB2.

Kiến trúc Hibernate





Kiến trúc Hibernate

- Trong lập trình, với những project sử dụng maven ta cần thêm các dependency gồm thư viện JDBC connector tương ứng của hệ quản trị cơ sở dữ liệu đang tương tác và hibernate-core:

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.4.14.Final</version>
</dependency>
```



Configuration

- Đối tượng Configuration đại diện cho các **cấu hình hay thuộc tính** để hibernate sử dụng, nó được tạo **đầu tiên** và chỉ tạo **một lần** lúc khởi động ứng dụng hibernate.
- Tập tin cấu hình chỉ định thông tin liên quan đến cơ sở dữ liệu và một số thông tin liên quan để ánh xạ lớp Java với bảng dữ liệu.
- Tập tin XML: **hibernate.cfg.xml**
- Tập tin này đặt **thư mục gốc** của project.



Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/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.cj.jdbc.Driver
        </property>
        <property name="hibernate.connection.url">
            jdbc:mysql://localhost/bookstore
        </property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">123456</property>
        <mapping resource="Author.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```



Configuration

- Cấu hình bằng mã nguồn Java

```
Configuration conf = new Configuration();
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/bookstore");
props.put(Environment.USER, "root");
props.put(Environment.PASS, "123456");

conf.setProperties(props);
```

Persistent Class

- Các lớp mà các đối tượng của nó sẽ được lưu trữ xuống cơ sở dữ liệu gọi là các persistent class, còn gọi là mô hình POJO (Plain Old Java Object).
- Lớp persistent phải:
 - Có phương thức khởi tạo không tham số.
 - Các thuộc tính nên được khai báo là **private**, và cần có các phương thức **getter** và **setter**.
 - Các lớp **không** kế thừa tường minh lớp khác.



Persistent Class

```
CREATE TABLE `author` (
    `author_code` varchar(15) NOT NULL,
    `first_name` varchar(50) NOT NULL,
    `last_name` varchar(50) NOT NULL,
    `date_of_birth` date NOT NULL,
    `gender` bit(1) NOT NULL
PRIMARY KEY (`author_code`)
);
```

Bảng author trong
cơ sở dữ liệu MySQL

Lớp Author tương
ứng bảng author

```
import java.util.Date;
public class Author {
    private String id;
    private String firstName;
    private String lastName;
    private Date dateOfBirth;
    private boolean gender;

    // Phương thức khởi tạo
    public Author() {

    }

    // getter và setter
}
```



Mapping File

- Hibernate sử dụng tập tin XML để ánh xạ các lớp Java vào các bảng cơ sở dữ liệu.
- Tập tin ánh xạ nên được đặt tên theo quy tắc <tên-lớp>.hbm.xml.



Mapping File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="pojo.Author" table="author">
    <meta attribute="class-description">Author Info</meta>
    <id column="author_code" name="id" type"string"/>
    <property column="first_name"
              name="firstName" type="string"/>
    <property column="last_name"
              name="lastName" type="string"/>
    <property column="date_of_birth"
              name="dateOfBirth" type="date"/>
    <property column="gender"
              name="gender" type="boolean"/>
</class>
</hibernate-mapping>
```

Mapping File

- Ví dụ thiết lập quan hệ many-to-one giữa book và category, quan hệ many-to-many giữa book và author.
 - Một danh mục có thể có nhiều quyển sách, một quyển sách thuộc một danh mục.
 - Một quyển sách có thể có nhiều tác giả, một tác giả có thể là tác giả nhiều quyển sách.



Mapping File

```
<class name="pojo.Book" table="Book">
    <meta attribute="class-description">
        Thông tin sách
    </meta>
    <id column="book_code" name="bookCode" type="string" />
    <property column="title" name="title" type="string" />
    ...
    <many-to-one column="category_id"
        name="category" not-null="true"
        class="pojo.Category" />
    <set name="authors" cascade="save-update"
        table="book_author"many-to-many column="author_code"
            class="pojo.Author" />
    </set>
</class>
```



Session Factory

- Đối tượng SessionFactory là “máy sản xuất” các đối tượng Session tương tác với cơ sở dữ liệu.
- Đối tượng này được tạo thông qua đối tượng Configuration và sẽ **tồn tại xuyên suốt** khi chương trình hoạt động.
- Mỗi cơ sở dữ liệu (có tập tin cấu hình riêng) **cần một thể hiện** của SessionFactory riêng.



Session Factory

```
public class HibernateUtil {  
    private final static SessionFactory FACTORY;  
  
    static {  
        Configuration conf = new Configuration();  
        configure.configure("hibernate.cfg.xml");  
  
        ServiceRegistry registry  
        = new StandardServiceRegistryBuilder()  
            .applySettings(conf.getProperties()).build();  
        FACTORY = conf.buildSessionFactory(registry);  
    }  
  
    public static SessionFactory getFactory() {  
        return FACTORY;  
    }  
}
```



Session

- Đối tượng của Session được sử dụng lấy kết nối vật lý đến cơ sở dữ liệu có **thời gian tồn tại ngắn**.
- Thể hiện của Session được tạo mỗi lúc có tương tác với cơ sở dữ liệu.
- Nhiệm vụ chính của đối tượng Session để thực hiện các thao tác tạo, đọc, cập nhật và xóa các thể hiện của các lớp đối tượng được ánh xạ (các persistent object).



Session

```
SessionFactory factory
        = HibernateUtil.getSessionFactory();
Session session = factory.openSession();

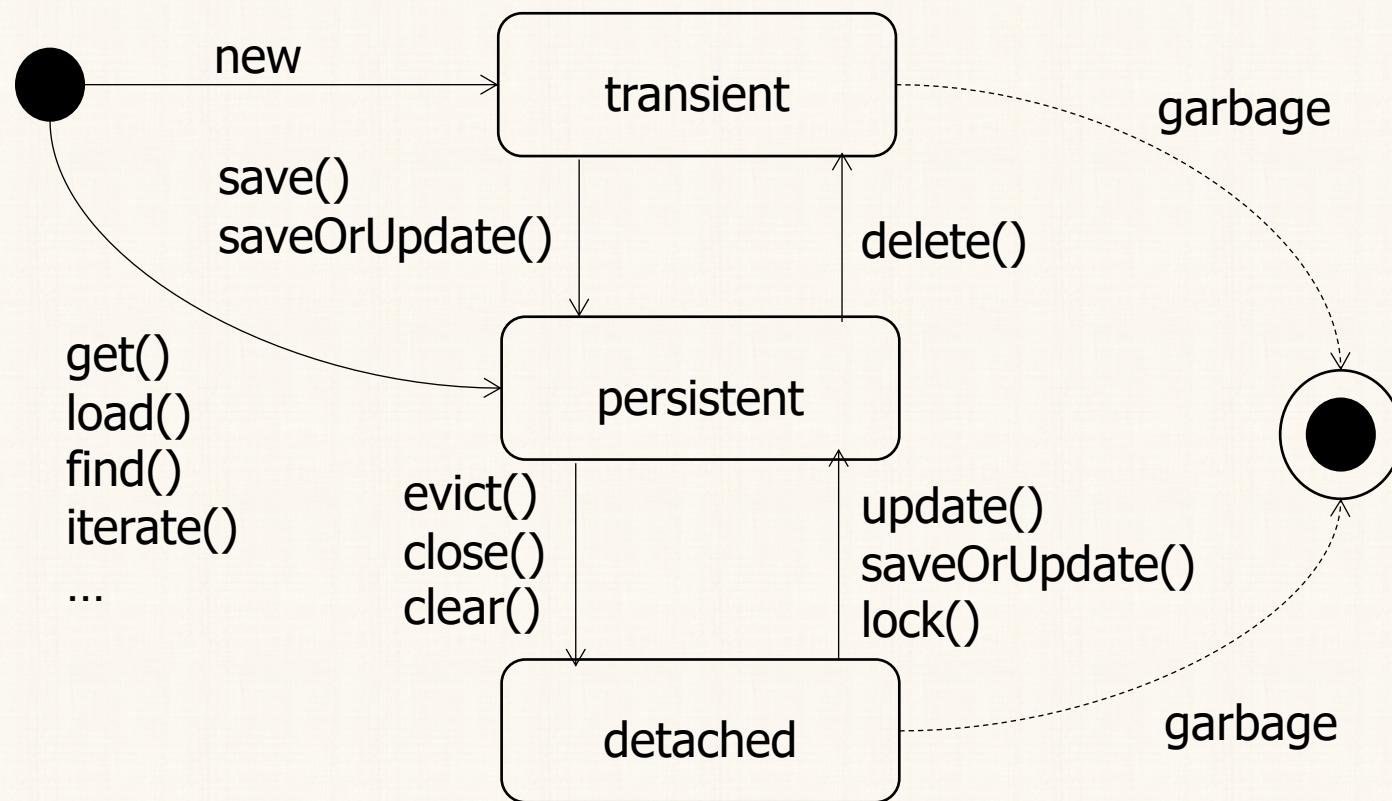
Query q = session.createQuery("FROM Author");
List authors = q.list();

Iterator iterator = authors.iterator();
while (iterator.hasNext()) {
    Author a = (Author) iterator.next();
    System.out.println(a.getFirstName());
    System.out.println(a.getLastName());
}

session.close()
```

- Persistent object có thể có một trong các trạng thái sau:
 - **transient**: persistent object vừa được tạo ra, chưa kết hợp với Session nào, chưa đại diện record nào trong cơ sở dữ liệu.
 - **persistent**: persistent object kết hợp với một Session, đại diện một record trong cơ sở dữ liệu.
 - **detached**: khi đóng Session.

- Chuyển đổi các trạng thái của persistent object



Các thành phần khác

- **Transaction:** hibernate làm việc với các giao tác thông qua đối tượng Transaction.
- **Query:** các đối tượng Query sử dụng SQL hoặc HQL (Hibernate Query Language) để truy vấn dữ liệu từ cơ sở dữ liệu và tạo các đối tượng.
- **Criteria:** các đối tượng Criteria được dùng để tạo và thực thi các câu truy vấn có điều kiện để tìm các đối tượng.



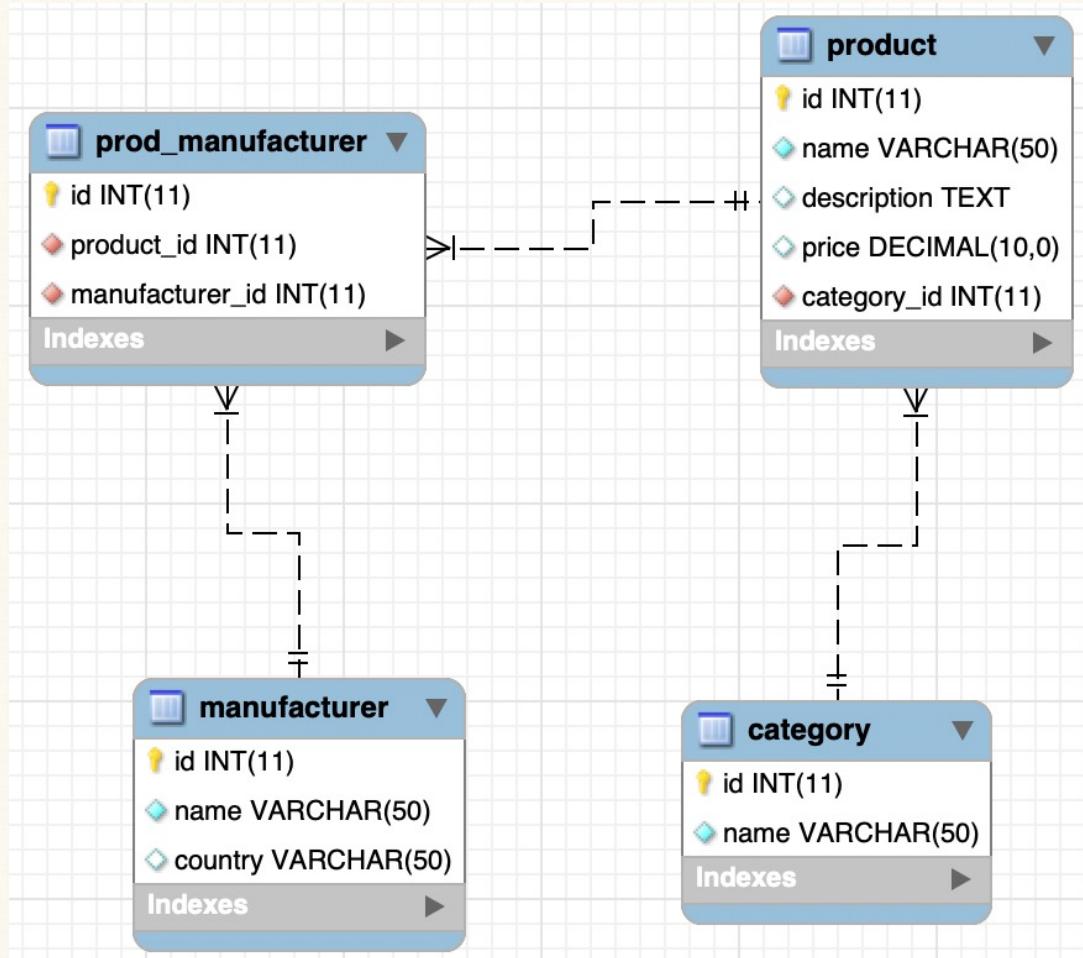
Sử dụng Annotation

- Annotation là cách thức đơn giản và hiệu quả để ánh xạ lớp Java tới các bảng cơ sở dữ liệu mà không cần sử dụng tập tin XML.
- Việc định nghĩa ánh xạ lớp POJO và bảng trong cơ sở dữ liệu được viết trực tiếp trong tập tin lớp POJO.
- Các lớp annotation định nghĩa trong gói

javax.persistence

Sử dụng Annotation

- Giả sử ta có lược đồ cơ sở dữ liệu quan hệ:





Sử dụng Annotation

- Lớp POJO sau định nghĩa cho bảng category.

```
import javax.persistence.*;  
  
@Entity  
@Table(name = "category")  
public class Category implements Serializable {  
    @Id  
    @GeneratedValue(strategy=GenerationType.IDENTITY)  
    @Column(name = "id")  
    private int id;  
    @Column(name = "name")  
    private String name;  
  
    // Các phương thức getter, setter  
}
```



Sử dụng Annotation

- Đối với các thuộc tính lớp có tên trùng với tên trường của bảng cơ sở dữ liệu thì không cần khai báo @Column. Lớp POJO cho bảng manufacturer

```
@Entity  
@Table(name = "manufacturer")  
public class Manufacturer implements Serializable {  
    @Id  
    @GeneratedValue(strategy=GenerationType.IDENTITY)  
    private int id;  
    private String name;  
    private String country;  
  
    // Các phương thức getter/setter  
}
```

@Entity

@Table(name = "product")

public class Product implements Serializable {

@Id

@GeneratedValue(strategy=GenerationType.IDENTITY)

▪ Quan hệ ManyToOne của product và category.

private int id;

private String name;

private String description;

private BigDecimal price;

@ManyToOne

@JoinColumn(name = "category_id")

private Category category;

@ManyToMany

@JoinTable(

name = "prod_manufacturer",

joinColumns = {

 @JoinColumn(name = "product_id")

},

inverseJoinColumns = {

 @JoinColumn(name = "manufacturer_id")

}

)

private Set<Manufacturer> manufacturers;

// Các phương thức getter/setter

Sử dụng Annotation

- Trong đối tượng Configuration chỉ định lớp annotation để ánh xạ bằng phương thức addAnnotatedClass().

```
Configuration configure = new Configuration();  
configure.configure("hibernate.cfg.xml");  
configure.addAnnotatedClass(Category.class);  
configure.addAnnotatedClass(Product.class);  
configure.addAnnotatedClass(Manufacturer.class);  
...
```

Sử dụng Annotation

- Hoặc chỉ định trong tập tin cấu hình

```
<hibernate-configuration>
  <session-factory>
    ...
    <mapping class="com.dht.pojo.Category"/>
    <mapping class="com.dht.pojo.Product"/>
    <mapping class="com.dht.pojo.Manufacturer"/>
  </session-factory>
</hibernate-configuration>
```

Sử dụng Annotation

- Minh họa thêm sản phẩm

```
session.getTransaction().begin();
Manufacturer m = session.get(Manufacturer.class, 1);
Category c = session.get(Category.class, 1);

Product p = new Product();
p.setName("iPhone XR");
p.setPrice(new BigDecimal(28));


.setCategory(c);



Set<Manufacturer> manufacturers = new HashSet<>();
manufacturers.add(m);



.setManufacturers(manufacturers);



session.save(p);
session.getTransaction().commit();
```

Sử dụng Annotation

- Trong lớp Category khai báo thuộc tính products phục vụ truy vấn ngược thông tin các sản phẩm của một danh mục thuận tiện.

```
@Entity  
@Table(name = "category")  
public class Category implements Serializable {  
    ...  
  
    @OneToMany(mappedBy = "category",  
              fetch = FetchType.LAZY)  
    private Set<Product> products;  
  
    // Phương thức getter và setter  
}
```

Sử dụng Annotation

- Tương tự trong quan hệ ManyToMany khi truy vấn ngược thì trong lớp POJO Manufacturer khai báo như sau:

```
@Entity  
@Table(name = "manufacturer")  
public class Manufacturer implements Serializable {  
    ...  
  
    @ManyToMany(mappedBy = "manufacturers")  
    private Set<Product> products;  
  
    // Các phương thức getter/setter  
}
```

Sử dụng Annotation

- Ví dụ: lấy danh sách sản phẩm theo danh mục hoặc theo nhà sản xuất.

```
Category c = session.get(Category.class, 1);
c.getProducts().forEach(
    p -> System.out.println(p.getName()) );

```



```
Manufacturer m = session.get(Manufacturer.class, 1);
m.getProducts().forEach(
    p -> System.out.println(p.getName()) );

```



Sử dụng Annotation

- Kế thừa
- <https://www.baeldung.com/hibernate-inheritance>

- HQL (Hibernate Query Language) là ngôn ngữ **truy vấn hướng đối tượng**.
- HQL làm việc với các **đối tượng persistent** và các thuộc tính của nó, không tương tác trực tiếp với các cột và bảng cơ sở dữ liệu.
- HQL là độc lập với cơ sở dữ liệu, hỗ trợ các đặc trưng của hướng đối tượng như kế thừa, đa hình, các quan hệ kết hợp (association, aggregation, composition).

HQL Query

- Mệnh đề FROM: lấy tất cả các đối tượng của lớp POJO.

```
Query q = session.createQuery("FROM Author");  
List r = q.list();
```

- Mệnh đề SELECT: lấy vài thuộc tính của đối tượng persistent.

```
Query q = session.createQuery(  
        "SELECT A.firstName "  
        + "FROM Author A");  
List r = q.list();
```



HQL Query

- Mệnh đề WHERE: chỉ định điều kiện lọc trong truy vấn.
- Mệnh đề ORDER BY: chỉ định cách sắp xếp kết quả truy vấn.

```
Query q = session.createQuery("FROM Author A "
    + "WHERE A.location=:loc "
    + "ORDER BY A.id DESC");

q.setParameter("loc", "Tp.HCM");
List r = q.list();
```

- **Mệnh đề GROUP BY:** gom nhóm kết quả truy vấn theo một thuộc tính nào đó.
- Một số phương thức thống kê như sau:
 - avg: tính giá trị trung của thuộc tính.
 - count: đếm số lần giá trị thuộc tính xuất hiện.
 - max: tính giá trị lớn nhất trong các giá trị của thuộc tính.
 - min: tính giá trị nhỏ nhất trong các giá trị của thuộc tính.
 - sum: tính tổng các giá trị của thuộc tính.



HQL Query

- Ví dụ

```
Query q = session.createQuery("SELECT A.location, "
                             + "count(A.id) "
                             + "FROM Author A "
                             + "GROUP BY A.location");
List r = q.list();
Iterator<Object[]> a = r.iterator();
while (a.hasNext()) {
    Object[] i = a.next();
    System.out.println(i[0]);
    System.out.println(i[1]);
}
```

- Mệnh đề **INSERT, UPDATE và DELETE**: lần lượt dùng để chèn, cập nhật và xóa thông tin các đối tượng trong cơ sở dữ liệu.
- Để thực thi các truy vấn này dùng phương thức **executeUpdate()** của Query.

```
Query q = session.createQuery("UPDATE Author "
    + "SET firstName=:fn "
    + "WHERE id=:id");
q.setParameter("fn", "Nam");
q.setParameter("id", "AUTHOR0001");
int r = q.executeUpdate();
```



Criteria Query API

- Hibernate cung cấp các Criteria API để thực hiện các thao tác tương tác với cơ sở dữ liệu dễ dàng, nhanh chóng và không cần biết quá nhiều cú pháp truy vấn SQL.
- Nó cho phép xây dựng các đối tượng truy vấn Criteria bằng cách sử dụng phương thức `createCriteria()` của Session.
- Từ Hibernate 5.2 các phương thức của `org.hibernate.Criteria` không còn dùng nữa, phát triển mới tập trung vào `CriteriaQuery API` của JPA (Java Persistence API).



Criteria Query API

- Tạo thể hiện CriteriaBuilder

```
CriteriaBuilder builder  
        = session.getCriteriaBuilder();
```

- Tạo đối tượng truy vấn là thể hiện của CriteriaQuery (T là một lớp POJO)

```
CriteriaQuery<T> query  
        = builder.createQuery(T.class);
```

- Thiết lập truy vấn Root bằng cách gọi phương thức **from()** của đối tượng CriteriaQuery.

```
Root<T> root = query.from(T.class);
```



Criteria Query API

- Chỉ định loại kết quả truy vấn bằng phương thức select() của đối tượng CriteriaQuery

```
query.select(root);
```

- Tạo thể hiện Query để thực thi truy vấn bằng cách gọi phương thức createQuery() của đối tượng Session với đối số là kết quả truy vấn

```
Query<T> q = session.createQuery(query);
```



Criteria Query API

- Thực thi câu truy vấn bằng cách gọi phương thức **getResultSet()** hoặc **getSingleResult()** của đối tượng Query.

```
List<T> list = q.getResultList();
```

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Product> query
        = builder.createQuery(Product.class);
Root<Product> root = query.from(Product.class);
query.select(root);

Query<Product> q = session.createQuery(query);
List<Product> rs = q.getResultList();
rs.forEach(p -> System.out.println(p.getName()));
```



Criteria Query API

Ví dụ: Lấy danh sách các sản phẩm cho phép lọc;

đữ liệu theo tên và mô tả sản phẩm.

= builder.createQuery(Product.class);

Đặt kw là chuỗi từ khóa truy vấn truyền vào hàm

q.select(root);

```
if (!kw.isEmpty()) {  
    String p = String.format("%%%s%%", kw);  
    Predicate p1 = builder.like(  
        root.get("name").as(String.class), p);  
    Predicate p2 = builder.like(  
        root.get("description").as(String.class), p);  
    q = q.where(builder.or(p1, p2));  
}  
List<Product> prods = session.createQuery(q)  
    .getResultList();
```

Criteria Query API

- Ví dụ lấy danh sách các sản phẩm theo khoảng giá chỉ định:
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Product> q
= builder.createQuery(Product.class);
Root<Product> root = q.from(Product.class);
q.select(root);
Đặt `fromPrice`, `toPrice` là khoảng giá `BigDecimal` được truyền vào để lọc giá sản phẩm.

```
Predicate p1 = builder.greaterThanOrEqualTo(  
    root.get("price").as(BigDecimal.class), fromPrice);  
Predicate p2 = builder.lessThanOrEqualTo(  
    root.get("price").as(BigDecimal.class), toPrice);  
  
q = q.where(builder.and(p1, p2));  
  
List<Product> prods = session.createQuery(q)  
    .getResultList();
```



Criteria Query API

• Đếm số lượng giá cao nhất, giá thấp nhất, giá trung bình các sản phẩm.

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Object[]> q = builder.createQuery(Object[].class);
Root<Product> root = q.from(Product.class);
q.multiselect(builder.count(root.get("id")),
               builder.max(root.get("price"))
                         .as(BigDecimal.class)),
               builder.min(root.get("price"))
                         .as(BigDecimal.class)),
               builder.avg(root.get("price"))
                         .as(BigDecimal.class)));
Query<Object[]> query = session.createQuery(q);
Object[] k = query.getSingleResult();
System.out.println("So luong: " + k[0]);
System.out.println("Cao nhat: " + k[1]);
System.out.println("Thap nhat: " + k[2]);
System.out.println("Trung binh: " + k[3]);
```

```
CriteriaBuilder b = session.getCriteriaBuilder();
CriteriaQuery<Object[]> q = b.createQuery(Object[].class);
Root<Product> pRoot = q.from(Product.class);
Root<Category> cRoot = q.from(Category.class);
q.where(b.equal(pRoot.get("category"), cRoot.get("id")));
q.multiselect(cRoot.get("name"),
    b.count(pRoot.get("id")),
    b.max(pRoot.get("price")).as(BigDecimal.class)),
    b.min(pRoot.get("price")).as(BigDecimal.class)),
    b.avg(pRoot.get("price")).as(BigDecimal.class)));
q.groupBy(cRoot.get("name"));
q.orderBy(b.asc(cRoot.get("name")));
Query<Object[]> query = session.createQuery(q);
List<Object[]> rs = query.getResultList();
for (Object[] obj: rs) {
    System.out.printf("Danh muc: %s\nSo luong: %d\n"
        + "Cao nhat: %.1f\nThap nhat: %.1f\n"
        + "Trung binh: %.1f\n",
        obj[0], obj[1], obj[2], obj[3], obj[4]);
}
```



Hibernate Cache

- Cache là cơ chế thường sử dụng để nâng cao hiệu năng hệ thống. Thông thường, ta sẽ có vùng nhớ đệm (buffer) ở giữa ứng dụng và CSDL để hạn chế tối đa số truy vấn xuống CSDL.
- Hibernate thực hiện nhiều cấp độ cache
 - First-level cache
 - Second-level cache
 - Query-level cache



Hibernate Cache



Named Queries

Q&A