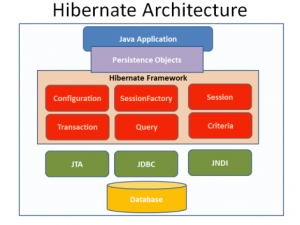
**1. Hibernate framework là gì?**

Hibernate là 1 [ORM Framework](https://stackjava.com/uncategorized/orm-la-gi-tong-quan-ve-orm-framework.html): thực hiện mapping cơ sở dữ liệu quan hệ sang các object trong ngôn ngữ hướng đối tượng.

Hibernate là 1 Framework cho persistence layer: Thực hiện giao tiếp giữa tầng ứng dụng với tầng dữ liệu (kết nối, truy xuất, lưu trữ…)

**2. Kiến trúc hibernate framework.**

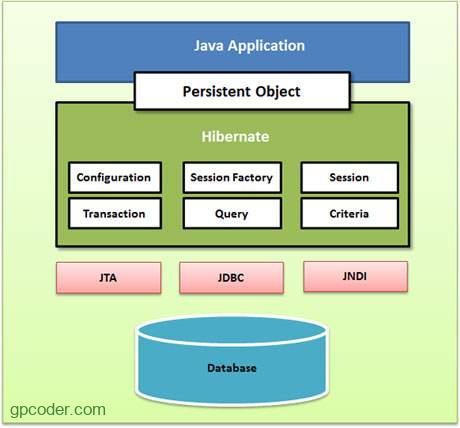


Mỗi table trong database là một object trong Hibernate. Do đó, bạn cần có một java bean cho mỗi table trong database. Các java bean này sẽ có các getters/setters và một số ngoại lệ theo quy ước của Hibernate.

Mỗi file mapping (ánh xạ) có dạng \*\*\*.hbm.xml có nhiệm vụ đặc tả mỗi quan hệ giữa các thuộc tính của đối tượng và các trường trong bảng CSDL (cột nào ứng với thuộc tính nào của class, kiểu giá trị là gì, quan hệ giữa các bảng chuyển thành quan hệ giữa các lớp ra sao…).

## Kiến trúc Hibernate

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

[](https://gpcoder.com/wp-content/uploads/2019/10/hibernate-architecture.jpg)

### **Persistence object**

Chính là các POJO object map với các table tương ứng của cơ sở dữ liệu quan hệ. Nó như là những container chứa dữ liệu từ ứng dụng để lưu xuống database, hay chứa dữ liệu tải lên ứng dụng từ database.

### **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.

### **Session Factory**

Là một interface giúp tạo ra session kết nối đến database bằng cách đọc các cấu hình trong Hibernate configuration.

SessionFactory là đối tượng nặng (heavy weight object) nên thường nó được tạo ra trong quá trình khởi động ứng dụng và lưu giữ để sử dụng sau này.

SessionFactory là một đối tượng luồng an toàn (Thread-safe) và được sử dụng bởi tất cả các luồng của một ứng dụng.

Mỗi một database phải có một session factory. Vì vậy, nếu bạn đang sử dụng nhiều cơ sở dữ liệu thì bạn sẽ phải tạo nhiều đối tượng SessionFactory. Giả sử ta sử dụng MySQL và Oracle cho ứng dụng Java của mình thì ta cần có một session factory cho MySQL, và một session factory cho Oracle.

### **Hibernate 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 instance 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 (thread-unsafe) và chúng cần được tạo ra và được đóng khi cần thiết.

Mỗi một đối tượng session được Session factory tạo ra sẽ tạo một kết nối đến database.

### **Transation**

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).

Transaction đảm bảo tính toàn vẹn của phiên làm việc với cớ sở dữ liệu. Tức là nếu có một lỗi xảy ra trong transaction thì tất cả các tác vụ thực hiện sẽ thất bại.

Transaction 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.

### **Query**

Các đối tượng Query sử dụng chuỗi truy vấn SQL (Native SQL) hoặc Hibernate Query Language (HQL) để lấy dữ liệu từ cơ sở dữ liệu và tạo các đối tượng.

### Criteria

Đối tượng Criteria được sử dụng để tạo và thực hiện các tiêu chí truy vấn để lấy các đối tượng từ database.

## Tại sao nên dùng Hibernate thay vì JDBC

### Object Mapping

Với [JDBC](https://gpcoder.com/6230-su-dung-jdbc-api-thuc-thi-cau-lenh-truy-van-du-lieu/) ta phải map các trường trong bảng với các thuộc tính của Java object một cách “thủ công”. Với Hibernate sẽ hỗ trợ ta map một cách “tự động” thông qua các file cấu hình map XML hay sử dụng các anotation.

JDBC sẽ map Java object với table như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | //rs là ResultSet trả về từ câu query get dữ liệu bảng user.  List<User> users = new ArrayList<>();  while(rs.next()) {       User user = new User();       user.setId(rs.getInt("id"));       user.setUsername(rs.getString("username"));       user.setPassword(rs.getString("password"));       user.setCreatedDate(rs.getDate("createdDate"));       users.add(user);   } |

Cũng với table user đó sử dụng các anotaion để Hibernate có thể map một cách “tự động” như sau.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | @Entity // cho biet đây là một thực thể liên kết với cơ sở dữ liệu  @Table(name = "user")// mapping vơi table user  @Data  public class UserModel {      @Id      @GeneratedValue(strategy = GenerationType.IDENTITY)      private Integer id;        @NotEmpty      @Column(name = "username")      private String username;        @NotEmpty      @Column(name = "password")      private String password;        @NotEmpty      @Column(name = "createdDate")      private Date createdDate;  } |

## **Ví dụ về lớp được chú thích**

Như tôi đã đề cập ở trên khi làm việc với Hibernate Annotation, tất cả siêu dữ liệu được gộp chung vào tệp POJO java cùng với mã, điều này giúp người dùng hiểu cấu trúc bảng và POJO đồng thời trong quá trình phát triển.

Hãy xem xét chúng ta sẽ sử dụng bảng EMPLOYEE sau đây để lưu trữ các đối tượng của chúng ta -

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 Nhân viên với các chú thích 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() {}

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;

}

}

Hibernate phát hiện rằng chú thích @Id nằm trên một trường và giả định rằng nó sẽ truy cập trực tiếp vào các thuộc tính của một đối tượng thông qua các trường trong thời gian chạy. Nếu bạn đặt chú thích @Id trên phương thức getId (), bạn sẽ cho phép truy cập vào các thuộc tính thông qua các phương thức getter và setter theo mặc định. Do đó, tất cả các chú thích khác cũng được đặt trên các trường hoặc các phương thức getter, tuân theo chiến lược đã chọn.

Phần sau sẽ giải thích các chú thích được sử dụng trong lớp trên.

## **Chú thích @Entity**

Các chú thích chuẩn EJB 3 được chứa trong gói **javax.persistence** , vì vậy chúng tôi nhập gói này như bước đầu tiên. Thứ hai, chúng tôi đã sử dụng chú thích **@Entity** cho lớp Employee, lớp này đánh dấu lớp này là một bean thực thể, vì vậy nó phải có một phương thức khởi tạo không đối số có thể nhìn thấy với ít nhất là phạm vi được bảo vệ.

## **@Table Annotation**

Chú thích @Table cho phép bạn chỉ định các chi tiết của bảng sẽ được sử dụng để duy trì thực thể trong cơ sở dữ liệu.

Chú thích @Table cung cấp bốn thuộc tính, cho phép bạn ghi đè tên của bảng, danh mục và lược đồ của nó, đồng thời thực thi các ràng buộc duy nhất trên các cột trong bảng. Hiện tại, chúng tôi chỉ sử dụng tên bảng, là NHÂN VIÊN.

## **Chú thích @Id và @GeneratedValue**

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

Theo mặc định, chú thích @Id sẽ tự động xác định chiến lược tạo khóa chính thích hợp nhất được sử dụng nhưng bạn có thể ghi đè điều này bằng cách áp dụng chú thích **@GeneratedValue** , có hai **chiến lược** và trình **tạo** tham số mà tôi sẽ không thảo luận ở đây, vì vậy chúng ta hãy chỉ sử dụng chiến lược tạo khóa mặc định. Cho phép Hibernate xác định loại trình tạo sẽ sử dụng giúp mã của bạn có thể di động giữa các cơ sở dữ liệu khác nhau.

## **Chú thích @Column**

Chú thích @Column được sử dụng để chỉ định chi tiết của cột mà trường hoặc thuộc tính sẽ được ánh xạ tới. Bạn có thể sử dụng chú thích cột với các thuộc tính được sử dụng phổ biến nhất sau đây:

* **thuộc tính name** cho phép tên của cột được chỉ định rõ ràng.
* **thuộc tính length** cho phép kích thước của cột được sử dụng để ánh xạ một giá trị đặc biệt cho một giá trị Chuỗi.
* **thuộc tính nullable** cho phép cột được đánh dấu KHÔNG ĐẦY ĐỦ khi lược đồ được tạo.
* **thuộc tính duy nhất** cho phép cột được đánh dấu là chỉ chứa các giá trị duy nhất.

## **Tạo lớp ứng dụng**

Cuối cùng, chúng 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 một số hồ sơ của Nhân viên và sau đó chúng tôi sẽ áp dụng các hoạt động CRUD trên các hồ sơ 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.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();

}

}

}

## **Cấu hình cơ sở dữ liệu**

Bây giờ chúng ta hãy tạo tệp cấu hình **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>

## **Biên dịch và Thực hiện**

Dưới đây là các bước để biên dịch và chạy ứng dụng được đề cập ở trên. Đảm bảo rằng bạn đã đặt PATH và CLASSPATH thích hợp trước khi tiếp tục biên dịch và thực thi.

* Xóa tệp ánh xạ Employee.hbm.xml khỏi đường dẫn.
* Tạo tệp nguồn Employee.java như hình trên và biên dịch nó.
* Tạo tệp nguồn ManageEaffee.java như được hiển thị ở trên và biên dịch nó.
* Thực thi lệnh nhị phân ManageEaffee để chạy chương trình.

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

$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

Nếu bạn kiểm tra bảng NHÂN VIÊN của mình, bảng này sẽ có các bản ghi sau:

mysql> select \* from EMPLOYEE;

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

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

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

| 29 | Zara | Ali | 5000 |

| 31 | John | Paul | 10000 |

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

2 rows in set (0.00 sec

mysql>

Hibenate cung cấp Criteria API chứa các phương thức cho phép bạn xây dựng một đối tượng truy vấn tiêu chuẩn theo yêu cầu của chương trình, nơi bạn có thể áp dụng các quy tắc lọc và điều kiện hợp lý.

Giao tiếp Session trong Hibernate cung cấp phương thức createCriteria() có thể được sử dụng để tạo một đối tượng Criteria mà trả về các thể hiện của lớp đối tượng persistence khi ứng dụng của bạn thi hành một truy vấn tiêu chuẩn.

Sau đây là ví dụ đơn giản nhất của một truy vấn tiêu chuẩn, chỉ đơn giản là trả về mọi đối tượng tương ứng với lớp Employee.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1  2 | Criteria cr = session.createCriteria(Employee.class);  List results = cr.list(); **Restrictions với Criteria** Bạn có thể sử dụng phương thức **add()** có sẵn cho đối tượng Criteria để thêm các điều kiện lọc cho truy vấn tiêu chuẩn. Sau đây là ví dụ để bổ sung điều kiện để trả các record có salary là 2000:   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 1  2  3 | 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 từng yêu cầu:   |  |  | | --- | --- | | 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  26  27  28 | 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")); |   Bạn có thể tạo các điều kiện AND và OR sử dụng các hạn chế LogicExpression như sau:  [?](https://viettuts.vn/hibernate/hibernate-criteria-queries)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | 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(); |   Tuy nhiên tất cả các điều kiện trên có thể được sử dụng trực tiếp với HQL như đã giải thích trong hướng dẫn trước.   **Phân trang bằng việc sử dụng Criteria** Có hai phương thức của giao tiếp Criteria để phân trang.   |  |  | | --- | --- | | **No.** | **Phương thức & Mô tả** | | 1 | **public Criteria setFirstResult(int firstResult)**  Phương thức này lấy một số nguyên đại diện cho hàng đầu tiên trong tập kết quả của bạn, bắt đầu với hàng 0. | | 2 | **public Criteria setMaxResults(int maxResults)**  Phương thức này cho Hibernate tìm kiếm một số **maxResults của các đối tượng.** |   Sử dụng hai phương thức trên, chúng ta có thể xây dựng một thành phần phân trang trong ứng dụng Web hoặc Swing. Sau đây là ví dụ mà bạn có thể mở rộng để lấy 10 hàng một lúc:  [?](https://viettuts.vn/hibernate/hibernate-criteria-queries)   |  |  | | --- | --- | | 1  2  3  4 | Criteria cr = session.createCriteria(Employee.class);  cr.setFirstResult(1);  cr.setMaxResults(10);  List results = cr.list(); |  **Sắp xếp kết quả** Criteria API cung cấp lớp org.hibernate.criterion.Order để sắp xếp kết quả trả về theo thứ tự tăng dần hoặc giảm dần, theo một trong các thuộc tính của đối tượng. Ví dụ này thể hiện cách sử dụng lớp Order để sắp xếp tập kết quả:  [?](https://viettuts.vn/hibernate/hibernate-criteria-queries)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12 | 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(); | | |