

Lập trình web SPRING MVC



BÀI 6: Tích hợp Hibernate

Mục tiêu :

1. Hiểu Hibernate
2. Cấu tình tích hợp Hibernate
3. Ánh xạ thực thể
4. Lập trình Hibernate
 - ❖ Truy vấn
 - ❖ Thao tác
5. Hiểu thêm ngôn ngữ HQL





01

Hiểu Hibernate

GIỚI THIỆU HIBERNATE

- ❑ Hibernate là framework hỗ trợ lập trình với CSDL trong các ứng dụng Java được ưa chuộng nhất hiện nay.
- ❑ Hibernate đóng vai trò là tầng trung gian giữa các đối tượng và CSDL để điều khiển các công việc quản lý lưu trữ trạng thái của các đối tượng đó dựa trên cơ sở ánh xạ.



GIỚI THIỆU HIBERNATE

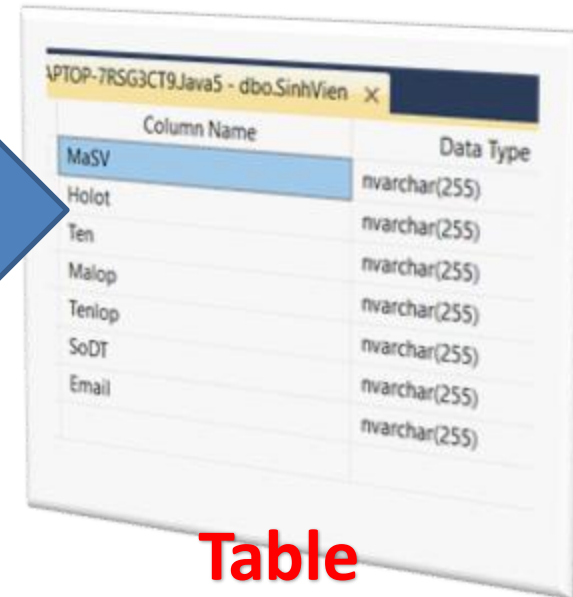
- ❑ Hibernate ánh xạ các lớp thực thể vào các bảng của CSDL quan hệ thông qua XML hoặc annotation.
- ❑ Hibernate tương thích với ngôn ngữ SQL, nó sử dụng HQL để truy vấn đối tượng.
- ❑ Truy vấn các thực thể kết hợp một cách dễ dàng thông qua mối quan hệ giữa các thực thể.
- ❑ Hibernate không những ổn định, tin cậy mà còn giúp giảm giảm thiểu công việc của người lập trình CSDL.

GIỚI THIỆU HIBERNATE

```
entity
Table(name="SinhVien")
public class SinhVien {
    @Id
    @Column(name="MaSV")
    private String maSV;
    @Column(name="Holot")
    private String holot;
    @Column(name="Ten")
    private String ten;
    @Column(name="Malop")
    private String malop;
    @Column(name="Tenlop")
    private String tenlop;
    @Column(name="SoDT")
    private String soDT;
    @Column(name="Email")
    private String email;
}
```

Thực thể
SinhVien

ORM/HIBERNATE



Column Name	Data Type
MaSV	nvarchar(255)
Holot	nvarchar(255)
Ten	nvarchar(255)
Malop	nvarchar(255)
Tenlop	nvarchar(255)
SoDT	nvarchar(255)
Email	nvarchar(255)

Table
SinhVien

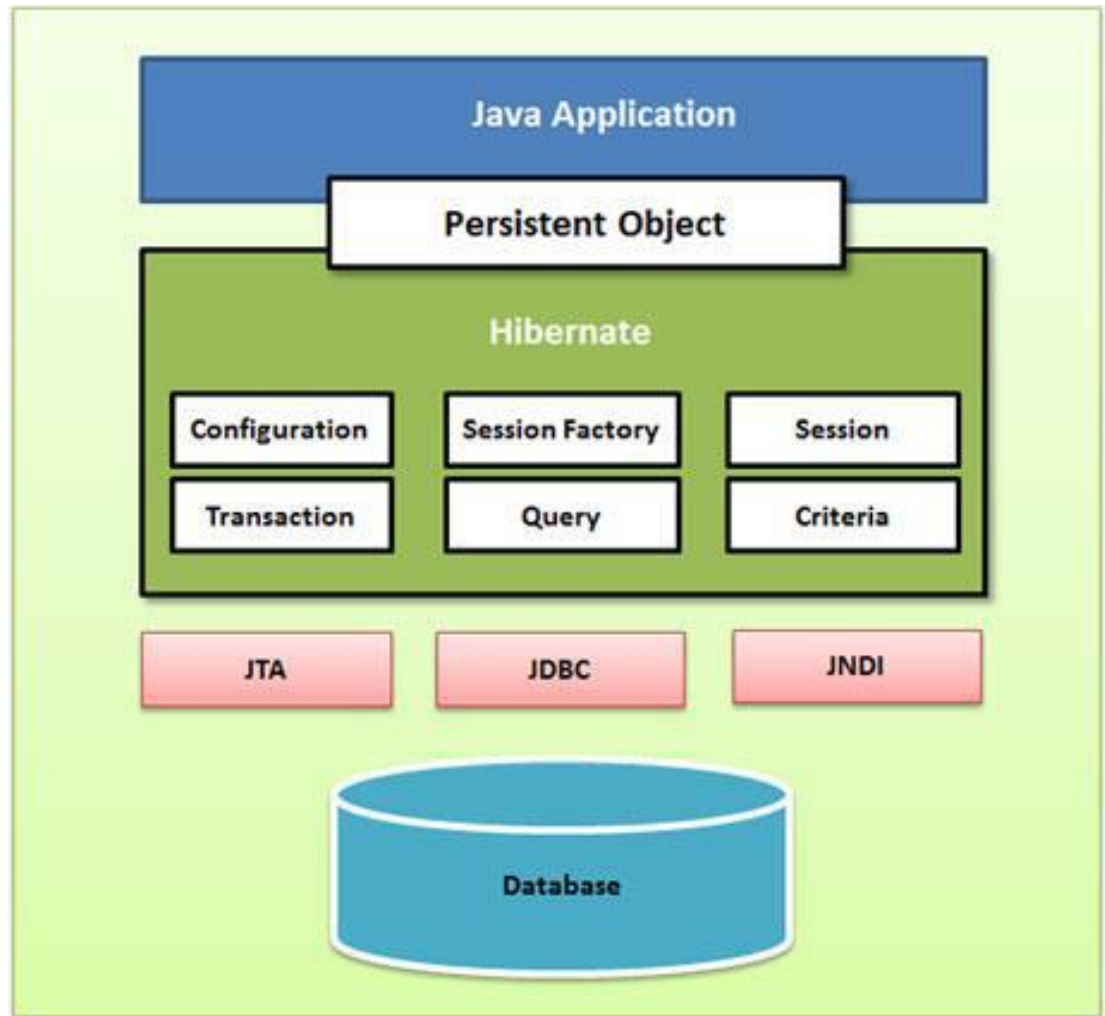
Hibernate ánh xạ các lớp thực thể vào các bảng của CSDL quan hệ thông qua XML hoặc annotation.

❑ Hibernate hỗ trợ hầu hết các CSDL phổ thông nhất hiện nay


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

CÁC THÀNH PHẦN HIBERNATE




















- ❑ Ứng dụng Java chỉ làm việc với các đối tượng
- ❑ Hibernate có trách nhiệm chuyển đổi các đối tượng vào các CSDL và ngược lại



CÁC THÀNH PHẦN HIBERNATE

- ❑ **Configuration**: được sử dụng để quản lý thông tin cấu hình kết nối đến CSDL và ánh xạ thực thể vào CSDL.
- ❑ **SessionFactory**: cho phép sản sinh ra nhiều session khác nhau từ thông tin cấu hình.
- ❑ **Session**: là một phiên làm việc được tạo từ SessionFactory
- ❑ **Transaction**: sử dụng để điều khiển các giao dịch làm thay đổi dữ liệu 
- ❑ **Query**: sử dụng để thực hiện truy vấn đối tượng
- ❑ **Criteria**: sử dụng để xây dựng câu lệnh truy vấn bằng lập trình thay cho câu lệnh HQL hay SQL.

- ❑ Trong số các thư viện này thì **sqljdbc4.jar** là JDBC driver làm việc với SQL Server.
- ❑ Nếu bạn muốn làm việc với một CSDL cụ thể nào đó (Oracle, MySQL, DB2...) thì phải bổ sung JDBC driver tương ứng với CSDL đó vào ứng dụng của bạn.

-  antlr-2.7.7.jar
-  aopalliance-1.0.jar
-  c3p0-0.9.2.1.jar
-  dom4j-1.6.1.jar
-  ehcache-core-2.4.3.jar
-  hibernate-c3p0-4.3.1.Final.jar
-  hibernate-commons-annotations-4.0.4.Final.jar
-  hibernate-core-4.3.1.Final.jar
-  hibernate-ehcache-4.3.1.Final.jar
-  hibernate-entitymanager-4.3.1.Final.jar
-  hibernate-jpa-2.1-api-1.0.0.Final.jar
-  javassist-3.18.1-GA.jar
-  javax.persistence-2.1.0-doc.zip
-  jboss-logging-3.1.3.GA.jar
-  jboss-transaction-api_1.2_spec-1.0.0.Final.jar
-  mchange-commons-java-0.2.3.4.jar
-  slf4j-api-1.6.1.jar
-  slf4j-simple-1.6.1.jar
-  sqljdbc4.jar



02

Cấu hình tích hợp Hibernate

CẤU HÌNH TÍCH HỢP HIBERNATE

❑ Để tích hợp Hibernate vào Spring bạn cần cấu hình 3 bean

❖ `org.springframework.jdbc.datasource.DriverManagerDataSource`

➤ Kết nối đến CSDL 

❖ `org.springframework.orm.hibernate4.LocalSessionFactoryBean`

➤ Tạo session factory từ kết nối CSDL 

❖ `org.springframework.orm.hibernate4.HibernateTransactionManager`

➤ Quản lý transaction 

```
<!-- Cấu hình Hibernate -->
<bean id="sessionFactory" |
    class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</prop>
            <prop key="hibernate.show_sql">true</prop>

        </props>
    </property>
```

CẤU HÌNH DATASOURCE

```
<!-- Kết nối CSDL -->  
<bean id="dataSource"  
    class="org.springframework.jdbc.datasource.DriverManagerDataSource"  
    p:driverClassName="com.microsoft.sqlserver.jdbc.SQLServerDriver"  
    p:url="jdbc:sqlserver://localhost:1433; Database=QLSP"  
    p:username="sa"  
    p:password="123">  
</bean>
```

- ❑ DataSource khai báo các thông số kết nối đến CSDL sẽ được sử dụng bởi SessionFactory
- ❑ Các thông số kết nối CSDL
 - ❖ JDBC Driver trong môn học này sử dụng SQL Server
 - ❖ Server nơi cài đặt SQL Server
 - ❖ Username và password đăng nhập vào CSDL thông qua TCP/IP.
- ❑ *Chú ý: phải kích hoạt kết nối TCP/IP cho CSDL SQL Server*

CẤU HÌNH SESSIONFACTORY

```
<!-- Cấu hình Hibernate -->
<bean id="sessionFactory"
      class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
  <property name="dataSource" ref="dataSource"/>
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</prop>
      <prop key="hibernate.show_sql">>false</prop>
    </props>
  </property>
  <property name="packagesToScan" value="ptithcm.entity"/>
</bean>
```

- ❑ SessionFactory sẽ được tiêm vào Controller khi làm việc với Hibernate
 - ❖ Phải tiêm data source đã khai báo trước đó vào session factory thông qua XML
 - ❖ Chỉ ra package nơi chứa các entity class ánh xạ vào các bảng trong CSDL

CẤU HÌNH TRANSACTION

```
<!-- Transaction Manager -->  
<bean id="transactionManager"  
      class="org.springframework.orm.hibernate4.HibernateTransactionManager"  
      p:sessionFactory-ref="sessionFactory" />  
<tx:annotation-driven transaction-manager="transactionManager" />
```

- ❑ TransactionManager được Spring nạp vào để quản lý transaction tự động.
- ❑ Spring sẽ tự tạo một session và commit hoặc rollback tự động để tăng hiệu suất trong việc xử lý thao tác CSDL
- ❑ <tx:annotation> sẽ cho phép bạn sử dụng transaction thông qua việc khai báo @Transaction





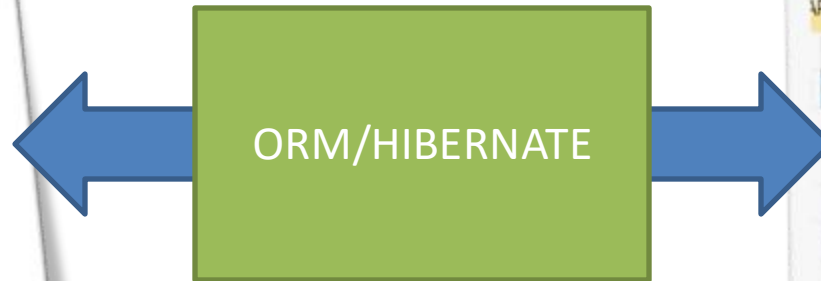
03

Ảnh xạ thực thể

ẢNH XẠ THỰC THỂ

```
entity
Table(name="SinhVien")
public class SinhVien {
    @Id
    @Column(name="MaSV")
    private String maSV;
    @Column(name="Holot")
    private String holot;
    @Column(name="Ten")
    private String ten;
    @Column(name="Malop")
    private String malop;
    @Column(name="Tenlop")
    private String tenlop;
    @Column(name="SoDT")
    private String soDT;
    @Column(name="Email")
    private String email;
}
```

Thực thể
SinhVien



Column Name		Data Type
MaSV		nvarchar(255)
Holot		nvarchar(255)
Ten		nvarchar(255)
Malop		nvarchar(255)
Tenlop		nvarchar(255)
SoDT		nvarchar(255)
Email		nvarchar(255)

Table
SinhVien

Hibernate ánh xạ các lớp thực thể vào các bảng của CSDL quan hệ thông qua XML hoặc annotation.

ÁNH XẠ THỰC THỂ



- ❑ Ánh xạ là sự mô tả việc kết hợp giữa
 - ❖ Lớp Entity và bảng
 - ❖ Các trường và các cột trong bảng
 - ❖ Thực thể kết với Relationship
- ❑ Hibernate chấp nhận 2 phương pháp ánh xạ
 - ❖ Sử dụng XML
 - ❖ Sử dụng Annotation. *Trong môn học này sử dụng Annotation*

Thông tin
chuyên ngành

Users

	Column Name	Condensed Type	Description
🔑	Id	nvarchar(50)	Tên đăng nhập
	Password	nvarchar(50)	Mật khẩu
	Fullname	nvarchar(50)	Họ và tên
	Photo	nvarchar(50)	Hình ảnh
	Email	nvarchar(50)	Email

Tài khoản người
sử dụng

Majors

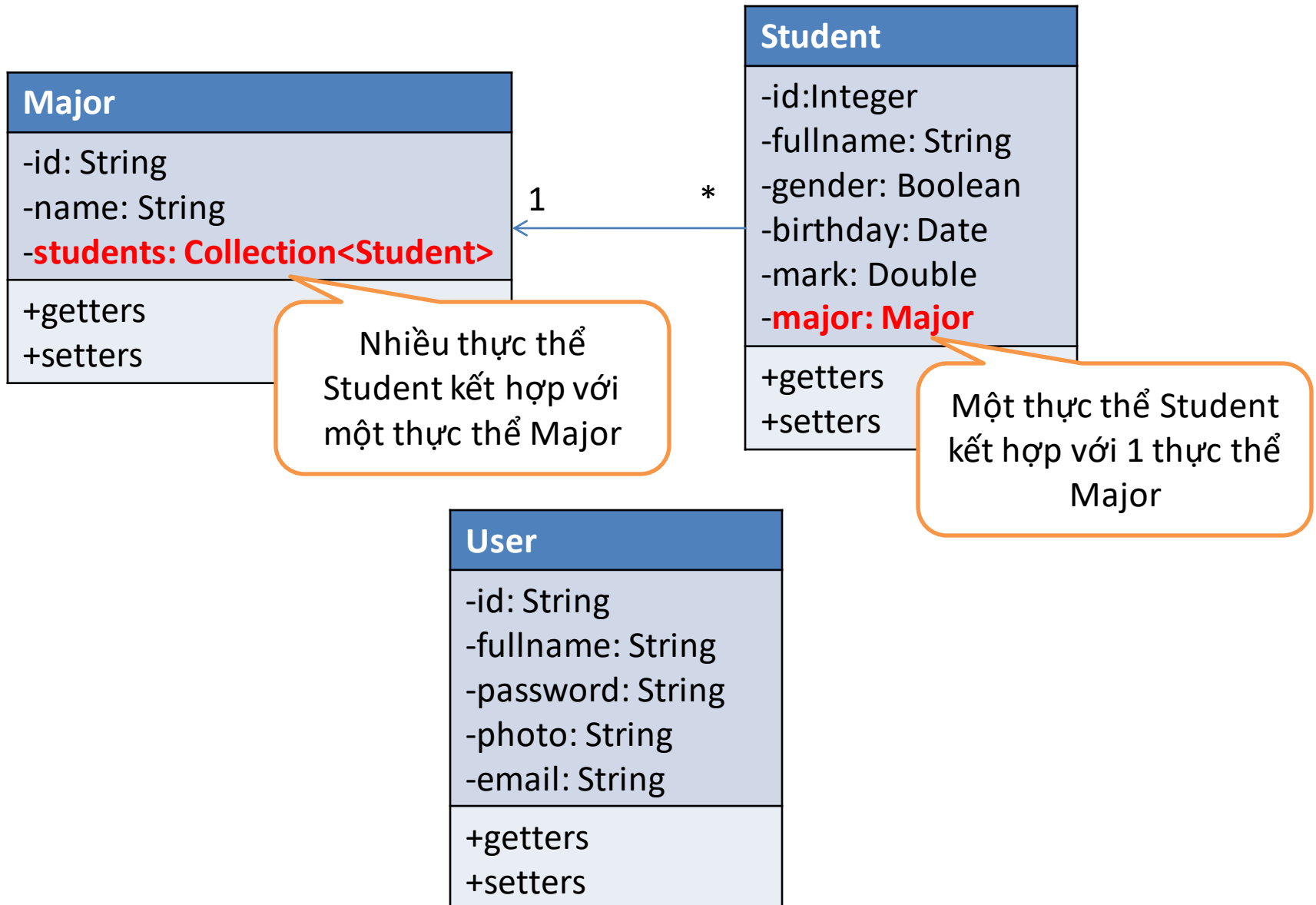
	Column Name	Condensed Type	Description
🔑	Id	char(3)	Mã chuyên ngành
	Name	nvarchar(50)	Tên chuyên ngành

Students

	Column Name	Condensed Type	Description
🔑	Id	int	Mã sinh viên
	Fullname	nvarchar(50)	Họ và tên
	Mark	float	Điểm trung bình
	Gender	bit	Giới tính
	Birthday	date	Ngày sinh
	MajorId	char(3)	Mã chuyên ngành

Thông tin sinh
viên

MÔ HÌNH THỰC THỂ



XÂY DỰNG LỚP THỰC THỂ USER

Users		
	Column Name	Condensed Type
🔑	Id	nvarchar(50)
	Password	nvarchar(50)
	Fullname	nvarchar(50)
	Photo	nvarchar(50)
	Email	nvarchar(50)

@Entity

→ @Table(name="Users")

public class User {

→ @Id

@Column(name="Id")

private String id;

@Column(name="Password")

private String password;

@Column(name="Fullname")

private String fullname;

@Column(name="Email")

private String email;

@Column(name="Photo")

private String photo;

getters/setters

}

- ❑ @Entity đánh dấu lớp thực thể
- ❑ @Table ánh xạ thực thể với bảng
- ❑ @Column ánh xạ trường với cột
- ❑ @Id đánh dấu trường ánh xạ với cột khóa chính

QUI ƯỚC ÁNH XẠ


- ❑ Có thể bỏ **@Table** nếu tên lớp thực thể và tên bảng giống nhau
- ❑ Có thể bỏ **@Column** nếu tên trường và tên cột giống nhau
- ❑ Nếu muốn một trường không ánh xạ vào một cột nào cả hãy sử dụng modifier **transient**
- ❑ Tất cả các trường phải có getter/setter.

```
@Entity
@Table(name="Users")
public class User {
    @Id
    private String id;
    private String password;
    private String fullname;
    private String email;
    private String photo;

    getters/setters
}
```

XÂY DỰNG THỰC THỂ STUDENT

Students

	Column Name	Condensed Type
	Id	int
	Fullname	nvarchar(50)
	Mark	float
	Gender	bit
	Birthday	date
	MajorId	char(3)

Chú ý sử dụng **kiểu dữ liệu** hợp lý

@Entity

@Table(name="Students")

public class Student {

@Id

@GeneratedValue

private Integer id;

private String fullname;

private Boolean gender;

@Temporal(TemporalType.**DATE**)

@DateTimeFormat(pattern="MM/dd/yyyy")

private Date birthday;

private Double mark;

@ManyToOne

@JoinColumn(name="majorId")

private Major major;

getters/setters

}

Trường id ánh xạ
với cột tự tăng

Dữ liệu kiểu
thời gian

Thay khóa ngoại
bằng **thực thể
kết hợp** (N-1)

CÁC ANNOTATION ẢNH XẠ

Annotation	Mô tả	Ví dụ
@Entity	Chỉ ra class là thực thể	@Entity @Table(name="Courses") public class Course{...}
@Table	Ánh xạ lớp thực thể với bảng	
@Id	Chỉ ra cột khóa chính	@Id @GeneratedValue int id;
@GeneratedValue	Chỉ ra cột tự tăng	
@Column	Ánh xạ thuộc tính với cột	@Column(name = "Name") String name
@Temporal	Chỉ ra kiểu dữ liệu chuyển đổi	@Temporal(TemporalType.DATE) Date startDate
@ManyToOne	Ánh xạ quan hệ nhiều-1	@ManyToOne @JoinColumn(name="CategoryId") Category category;
@JoinColumn	Chỉ ra cột kết nối	
@OneToMany	Ánh xạ quan hệ 1-nhiều	@OneToMany(mappedBy="category") Collection<Product> products;

ẢNH XẠ THỰC THỂ KẾT HỢP

❑ 1-N: một ngành có nhiều sinh viên

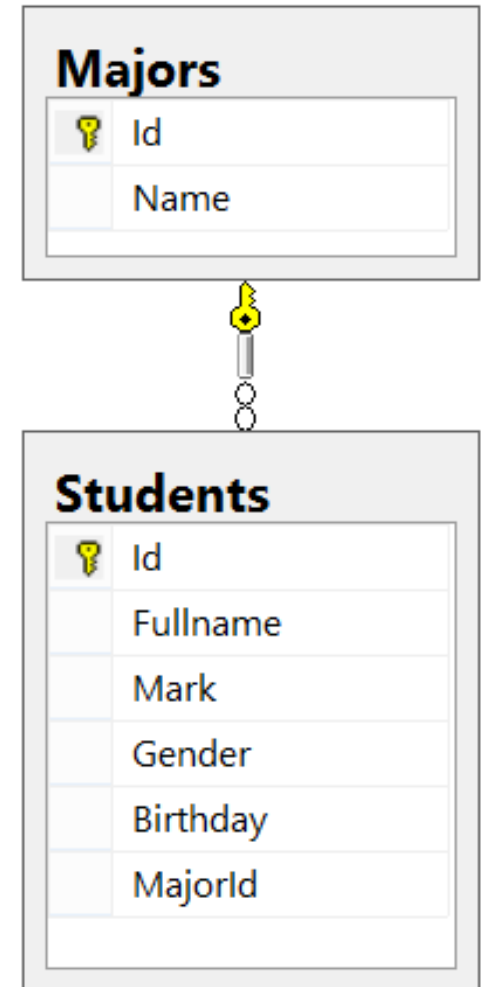
❖ Sử dụng **@OneToMany**

```
@OneToMany(mappedBy="major", fetch=FetchType.EAGER)  
private Collection<Student> students;
```

❑ N-1: nhiều sinh viên thuộc một ngành

❖ Sử dụng **@ManyToOne** và **@JoinColumn**

```
@ManyToOne  
@JoinColumn(name="majorId")  
private Major major;
```



ẢNH XẠ THỰC THỂ KẾT HỢP 1-N

```
@Entity
@Table(name="Majors")
public class Major {
    @Id
    private String id;
    private String name;

    @OneToMany(mappedBy="major", fetch=FetchType.EAGER)
    private Collection<Student> students;

    getters/setters...
}
```

❑ @OneToMany(mappedBy, fetch)

- ❖ **mappedBy** chỉ ra tên trường thực thể kết hợp
- ❖ **fetch** chế độ nạp kèm thực thể kết hợp khi thực thể chính được nạp vào bộ nhớ.
 - **FetchType.LAZY**: không nạp kèm thực thể kết hợp
 - **FetchType.EAGER**: nạp kèm thực thể kết hợp



04

Lập trình Hibernate

LẬP TRÌNH HIBERNATE

- ❑ Trước khi lập trình hibernate bạn phải tiêm **SessionFactory** vào @Controller bằng cách sử dụng @Autowired
- ❑ Trong môi trường Spring bạn có thể sử dụng session được tạo sẵn (**factory.getCurrentSession()**) hoặc mở một session mới (**factory.openSession()**)
- ❑ Nếu sử dụng session được mở sẵn thì Spring tự động commit và rollback, tuy nhiên bạn phải khai báo thêm với @**Transactional**
- ❑ Đối với session tạo mới thì bạn phải commit, rollback và đóng lại khi không cần nữa.

MÔ HÌNH LẬP TRÌNH HIBERNATE

```
@Controller
@RequestMapping("/hibernate/")
public class HibernateController{
```

```
    @Autowired
    SessionFactory factory;
```

```
    @Transactional
    @RequestMapping("query")
    public String query() {
        Session session = factory.getCurrentSession();
        // Truy vấn đặt ở đây
        return "demo1";
    }
```

```
    @RequestMapping("execute")
    public String execute() {
        Session session = factory.openSession();
        // Thao tác thực thể đặt ở đây
        return "demo1";
    }
```

```
}
```

Tiêm **SessionFactory** vào trước khi lập trình Hibernate

Session được **mở sẵn** nên được sử dụng trong truy vấn thực thể.

Chú ý: đính kèm **@Transactional** có thể đặt trên Controller để áp dụng cho tất cả các phương thức action

Session **mới** nên sử dụng trong thao tác thực thể (**save, update, delete**)

TRUY VẤN THỰC THỂ

```
Session session = factory.getCurrentSession();  
String hql = "FROM Major";  
Query query = session.createQuery(hql);  
List<Major> list = query.list();
```

❑ Lấy session được Spring tạo sẵn

- ❖ Session session = factory.**getCurrentSession()**;

❑ Tạo đối tượng Query

- ❖ String hql = "FROM Major";

- ❖ Query query = session.**createQuery(hql)**;

- ❖ *Chú ý: HQL là câu lệnh truy vấn đối tượng. Trong đó chỉ có thực thể và thuộc tính mà không có bảng và cột*

❑ Truy vấn danh sách đối tượng

- ❖ **List<Major>** list = **query.list()**;



TRUY VẤN CÓ THAM SỐ

```
Session session = factory.getCurrentSession();  
String hql = "FROM Student s WHERE s.mark BETWEEN :min AND :max";  
Query query = session.createQuery(hql);  
query.setParameter("min", 6.5);  
query.setParameter("max", 7.5);  
List<Major> list = query.list();
```

- ❑ Tham số bắt đầu bởi dấu hai chấm (:)
- ❑ `Query.setParameter(name, value)` được sử dụng để cấp giá trị cho các tham số trước khi thực hiện truy vấn

TRUY VẤN PHÂN TRANG

- ❑ Truy vấn phân trang là truy vấn một đoạn đối tượng từ kho dữ liệu

```
Session session = factory.getCurrentSession();  
String hql = "FROM Student";  
Query query = session.createQuery(hql);  
query.setFirstResult(5);  
query.setMaxResults(10);  
List<Major> list = query.list();
```

- ❑ `Query.setFirstResult(startIndex)` chỉ ra vị trí bắt đầu truy vấn
- ❑ `Query.setMaxResults(size)` chỉ ra số thực thể tối đa truy vấn được

TRUY VẤN MỘT SỐ THUỘC TÍNH

```
Session session = factory.getCurrentSession();  
String hql = "SELECT name, mark FROM Student";  
Query query = session.createQuery(hql);  
List<Object[]> list = query.list();
```

```
Session session = factory.getCurrentSession();  
String hql = "SELECT s.major.name, COUNT(s.id) "  
            + " FROM Student s GROUP BY s.major.name";  
Query query = session.createQuery(hql);  
List<Object[]> list = query.list();
```

- ❑ Sử dụng mệnh đề SELECT để chọn một số thuộc tính
- ❑ Kết quả nhận được là **danh sách mảng đối tượng**. Số phần tử trong mảng là số các thuộc tính truy vấn trong mệnh đề SELECT.

TRUY VẤN MỘT GIÁ TRỊ

```
Session session = factory.getCurrentSession();  
String hql1 = "SELECT MAX(mark) FROM Student";  
Query query1 = session.createQuery(hql1);  
Double max = (Double) query1.uniqueResult();
```

```
String hql2 = "SELECT COUNT(s) FROM Student s";  
Query query2 = session.createQuery(hql2);  
Long count = (Long) query2.uniqueResult();
```

- ❑ Hql1 và hql2 là 2 câu lệnh chỉ cho 1 giá trị
- ❑ Sử dụng phương thức `query.uniqueResult()` sau đó ép sang kiểu thích hợp để nhận kết quả.

TRUY VẤN MỘT THỰC THỂ

```
Session session = factory.getCurrentSession();  
Major major = (Major) session.get(Major.class, "MOB");
```

```
Major major2 = new Major();  
major2.setId("MOB");  
session.refresh(major2);
```

- ❑ Nếu bạn chỉ có Id thì sử dụng phương thức `session.get(Class, Id)` để nạp thực thể từ CSDL
- ❑ Nếu bạn đang có một đối tượng và muốn nạp lại thông tin từ CSDL thì sử dụng phương thức `session.refresh(Object)`

THÊM MỚI

```
Session session = factory.openSession();
Transaction t = session.beginTransaction();
try {
    session.save(major);
    t.commit();
    model.addAttribute("message", "Thêm mới thành công !");
}
catch (Exception e) {
    t.rollback();
    model.addAttribute("message", "Thêm mới thất bại !");
}
finally {
    session.close();
}
```

CẬP NHẬT

```
Session session = factory.openSession();
Transaction t = session.beginTransaction();
try {
    session.update(major);
    t.commit();
    model.addAttribute("message", "Cập nhật thành công !");
}
catch (Exception e) {
    t.rollback();
    model.addAttribute("message", "Cập nhật thất bại !");
}
finally {
    session.close();
}
```

```
Session session = factory.openSession();
Transaction t = session.beginTransaction();
try {
    session.delete(major);
    t.commit();
    model.addAttribute("message", "Xóa thành công !");
}
catch (Exception e) {
    t.rollback();
    model.addAttribute("message", "Xóa thất bại !");
}
finally {
    session.close();
}
```


THAO TÁC THỰC THỂ

- ❑ Thao tác thực thể là làm thay đổi dữ liệu trong các thực thể như thêm, sửa, xóa.
- ❑ Hibernate cần phải sử dụng transaction để điều khiển các hành động này

```
Session session = factory.openSession();
```

Mở một session mới

```
Transaction t = session.beginTransaction();
```

Bắt đầu điều khiển transaction

```
try {
```

```
// save(), update() và delete()
```

Các hoạt động thao tác dữ liệu

```
t.commit();
```

Chấp nhận thay đổi dữ liệu

```
}  
catch (Exception e) {
```

```
t.rollback();
```

Hủy bỏ thay đổi dữ liệu

```
}
```

```
finally {
```

```
session.close();
```

Đóng session

```
}
```



 05

Ngôn ngữ HQL

- ❑ HQL (Hibernate Query Language) được thiết kế để truy vấn đối tượng được ánh xạ vào CSDL quan hệ
- ❑ HQL trong suốt CSDL – có thể sử dụng để truy vấn bất kỳ CSDL nào mà không cần thay đổi mã truy vấn.
- ❑ HQL gần giống với SQL tuy nhiên nó sử dụng
 - ❖ Thực thể thay bảng
 - ❖ Thuộc tính thay cột
 - ❖ Toán tử và hàm của hibernate nhiều hơn SQL và tập trung vào xử lý đối tượng
- ❑ Khi học HQL người ta chỉ chú trọng vào việc truy vấn vì thao tác đã có API đảm trách.

NGÔN NGỮ HQL - TRUY VẤN ĐỐI TƯỢNG

SELECT <danh sách thuộc tính>

FROM <thực thể>

WHERE <điều kiện>

GROUP BY <biểu thức nhóm>

HAVING <điều kiện nhóm>

ORDER BY <biểu thức sắp xếp>

- ❑ HQL cũng cho phép sử dụng JOIN nhưng thông thường người ta sử dụng thực thể kết hợp thay cho JOIN.

❑ FROM – Tất cả thực thể

- ❖ FROM Course
- ❖ FROM Course **as c**
- ❖ FROM Course **c**

❑ WHERE – Lọc theo điều kiện

- ❖ FROM Course WHERE name **LIKE** 'Nguyễn%'
- ❖ FROM Course WHERE schoolfee **BETWEEN** 100 **AND** 250
- ❖ FROM Product WHERE description **IS NOT NULL**
- ❖ FROM Product p WHERE p.category.id **IN** (1, 3, 5, 7)

❑ SELECT – Một số thuộc tính

- ❖ SELECT **name, schoolfee** FROM Course
- ❖ SELECT c.name, c.schoolfee FROM Course c

❑ Sắp xếp

❖ FROM Course **ORDER BY** startDate **DESC**, schoolfee

❑ Thống kê

❖ SELECT **AVG**(unitPrice), **MAX**(discount) FROM Product
p **GROUP BY** p.category

❖ SELECT AVG(unitPrice), MAX(discount) FROM Product
p GROUP BY p.category **HAVING** AVG(unitPrice) >
100

NGÔN NGỮ HQL – TOÁN TỬ

□ Toán tử

- ❖ Số học: +, -, *, /
- ❖ So sánh: =, >=, <=, <>, !=
- ❖ Logic: AND, OR, NOT

□ Toán tử đặc biệt:

- ❖ NOT
- ❖ IN,
- ❖ BETWEEN
- ❖ IS NULL
- ❖ LIKE
- ❖ IS EMPTY

❑ Tổng hợp thống kê

- ❖ avg(...), sum(...), min(...), max(...)
- ❖ count(*)
- ❖ count(...), count(distinct ...), count(all...)

❑ Xử lý chuỗi

- ❖ concat(...,...): ghép chuỗi
- ❖ substring(): lấy chuỗi con
- ❖ trim(): cắt bỏ ký tự trắng 2 đầu chuỗi
- ❖ lower(): chuyển in thường
- ❖ upper(): chuyển in hoa
- ❖ length(): lấy độ dài chuỗi
- ❖ locate(): tìm vị trí chuỗi con

❑ Hàm xử lý thời gian

- ❖ `current_date()`: lấy ngày, tháng năm
- ❖ `current_time()`: lấy giờ, phút và giây
- ❖ `current_timestamp()`: lấy ngày giờ
- ❖ `second(...)`: lấy giây
- ❖ `minute(...)`: lấy phút
- ❖ `hour(...)`: lấy giờ trong ngày
- ❖ `day(...)`: lấy ngày trong tháng
- ❖ `month(...)`: lấy tháng
- ❖ `year(...)`: lấy năm

□ Các hàm khác

- ❖ `abs()`: lấy giá trị tuyệt đối
- ❖ `sqrt()`: tính căn bậc 2
- ❖ `str()`: chuyển số/ngày sang chuỗi
- ❖ `cast(... as ...)`: ép kiểu

TỔNG KẾT NỘI DUNG BÀI HỌC

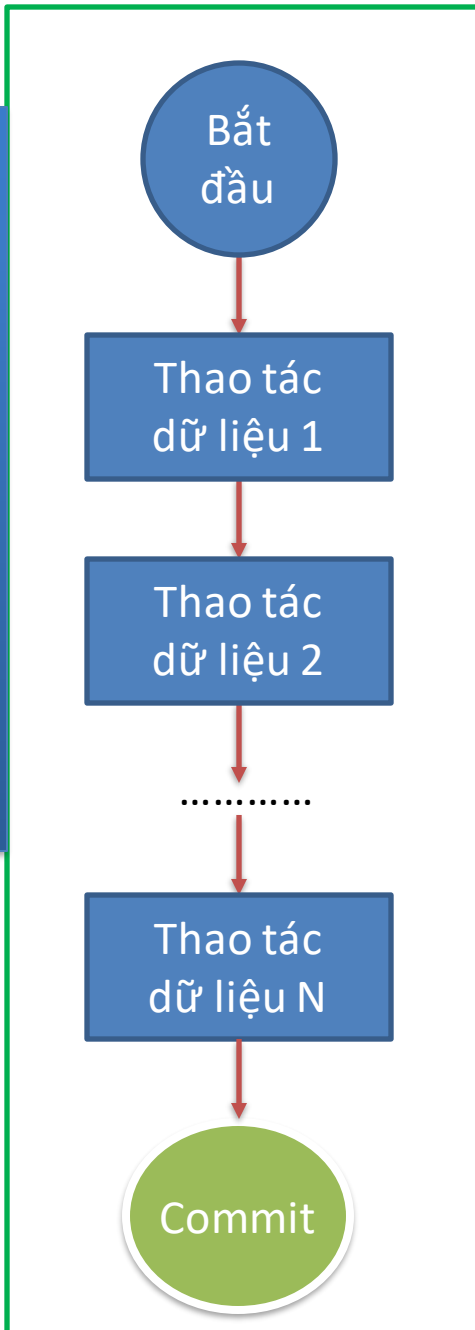
- ☑ Tìm hiểu Hibernate
- ☑ Cấu hình tích hợp Hibernate vào môi trường Spring
- ☑ Ánh xạ thực thể
- ☑ Lập trình Hibernate
 - ☑ Truy vấn
 - ☑ Thao tác
- ☑ Tham khảo thêm ngôn ngữ HQL

TRANSACTION

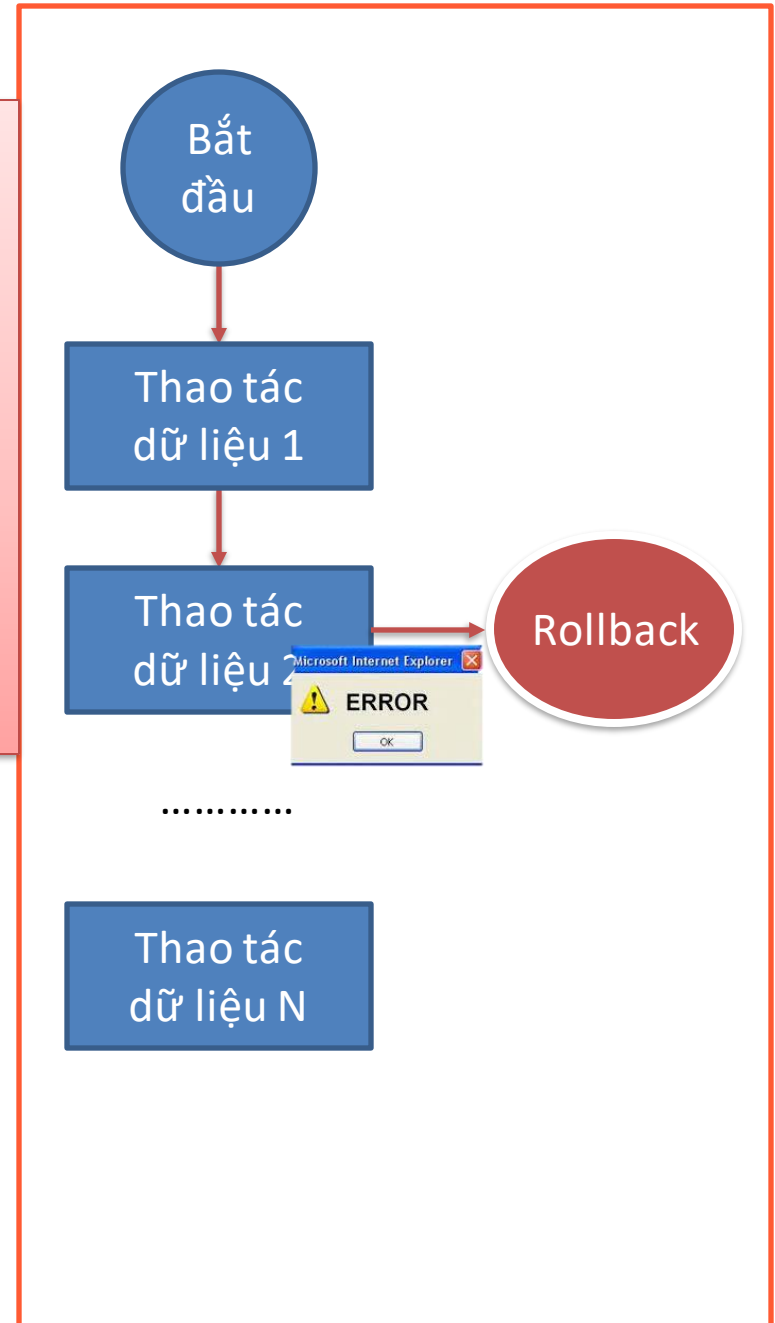
- ❑ **Transaction** là một chủ đề được đề cập đến rất nhiều trong các ứng dụng doanh nghiệp, để đảm bảo tính toàn vẹn của dữ liệu ngay cả trong các hệ thống lớn phát sinh một lớn các thay đổi lên database và đồng thời.
- ❑ **Transaction** là một tập hợp các hoạt động đọc/ghi xuống database hoặc là chúng đều thực thi thành công hết hoặc không có hoạt động nào được thực thi xuống database.

TRANSACTION

Qui
trình
tác
động
dữ
liệu
hoàn
chỉnh



Qui
trình
tác
động
dữ
liệu
bị
lỗi



TRANSACTION

Before: X : 500

Y: 200

Transaction T

T1

X=500

Read (X)

Y=200

X=400

X := X - 100

Y=200

Write (X)

After: X : 400

Microsoft Internet Explorer



ERROR

OK

X=400

Y=200

TRANSACTION

X=50 – Y=50

T			T''		
X=50	<u>Read (X)</u>	Y=50	X=500	<u>Read (X)</u>	Y=50
X=500	<u>X: = X*100</u>	Y=50	X=500	<u>Read (Y)</u>	Y=0
X=500	<u>Write (X)</u>	Y=50	X=500	<u>Z: = X + Y</u>	Y=0
X=500	<u>Read (Y)</u>	Y=50	<u>Write (Z)</u>		
X=500	<u>Y: = Y – 50</u>	Y=0	Z=500		
	<u>Write</u>				

bộ xương database

TRANSACTION

❑ Một transaction được đặc trưng bởi 4 yếu tố (thường được gọi là **ACID**):

4. **Durability**: đảm bảo một transaction thực thi thành công thì tất cả những thay đổi trong transaction phải được đồng bộ xuống database bất kể cả khi hệ thống xảy ra lỗi hoặc bị mất điện. Các transaction thành công nhưng chưa được đồng bộ xuống database phải được đồng bộ lại khi hệ thống hoạt động trở lại.

Tóm lược:

- ❖ Các thuộc tính ACID về cơ bản nó đảm bảo tính đúng đắn và nhất quán của dữ liệu khi trải qua các transaction khác nhau. Tuy nhiên các thuộc tính này có thể sẽ không thể áp dụng với các hệ thống phân tán đang phát triển rất nhanh trong những năm gần đây, chúng ta nên tìm hiểu nhiều hơn nữa.

