HIBERNATE

CYBERSOFT.EDU.VN

Nội dung

❑ **Hibernate**

✓ ORM là gì?

✓ Hibernate là gì?

✓ Cấu trúc của Hibernate.

✓ Ánh xạ dữ liệu trong Hibernate.

✓ Cấu hình Hibernate.

✓ Session method

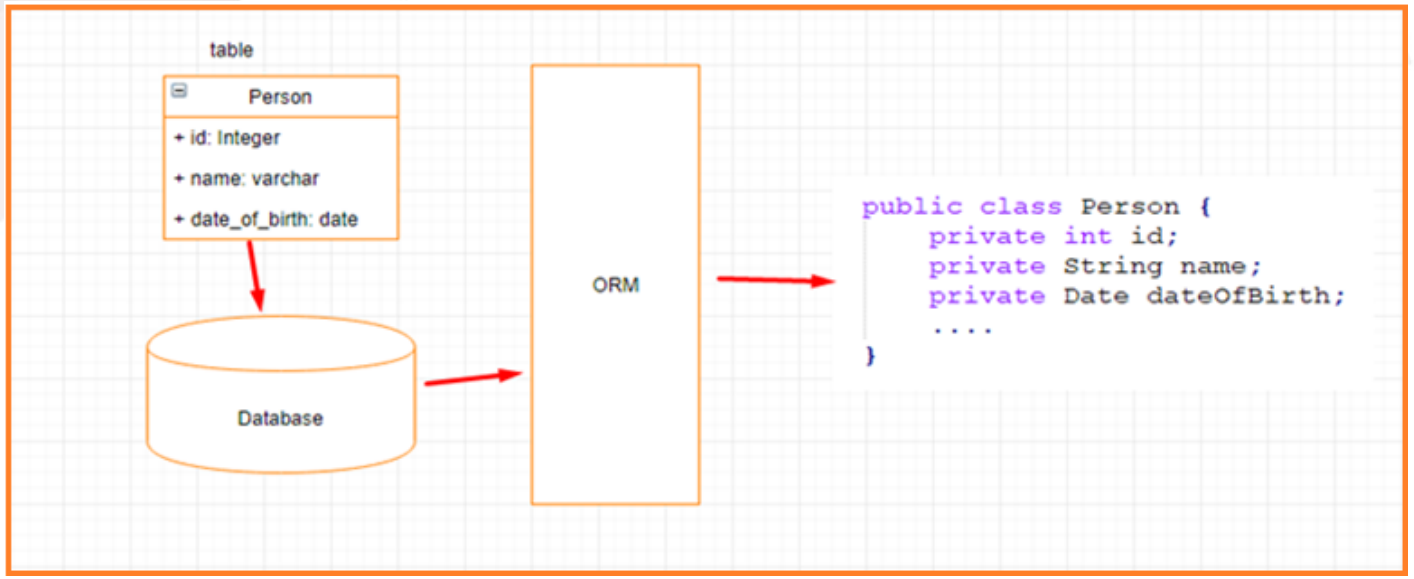
❑ **HQL**

✓ HQL là gì?

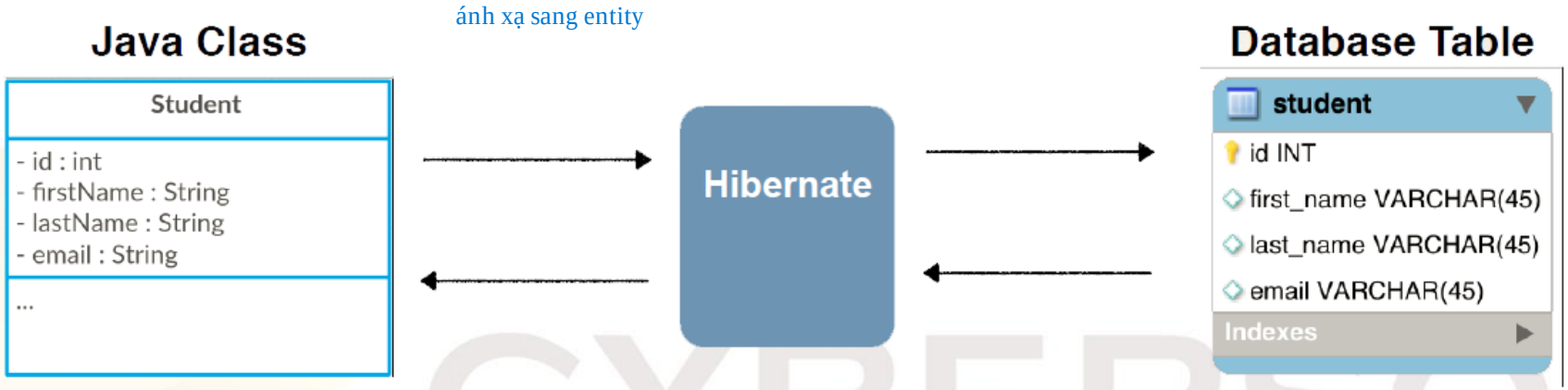
✓ Cú pháp câu lệnh HQL.

ORM là gì?

- **ORM** (Object Relational Mapping) là một kỹ thuật/cơ chế lập trình thực hiện **ánh xạ CSDL sang các đối tượng** trong các ngôn ngữ lập trình hướng đối tượng như Java, C# …
- Các **bảng, cột trong database được ánh xạ sang các đối tượng và các thuộc tính tương ứng trong Java** vì vậy chúng ta có thể **thao tác với database thông qua các đối tượng**.



Hibernate là gì?

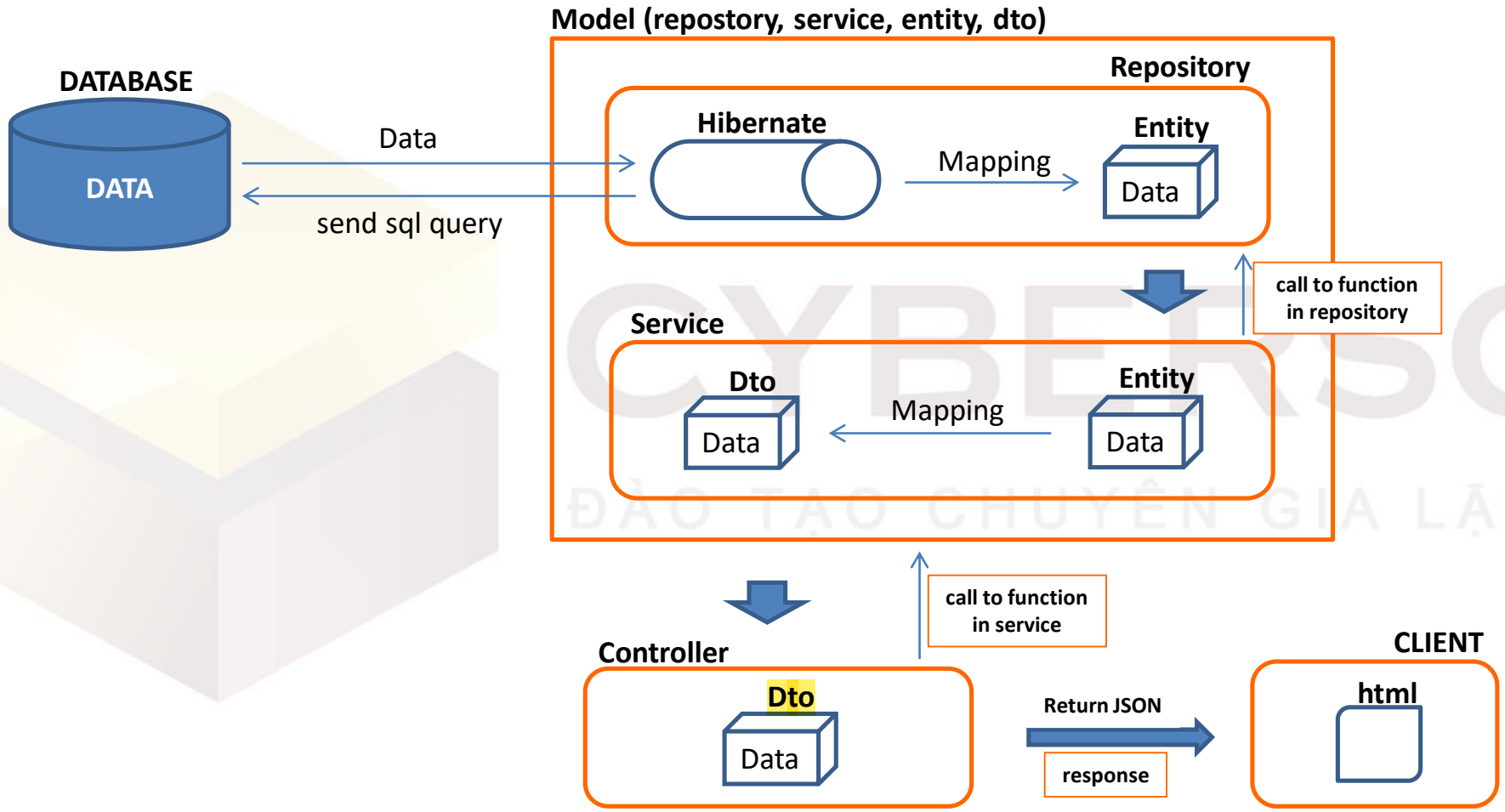


- **Hibernate** là **một thư viện ORM mã nguồn mở** giúp lập trình viên viết ứng dụng Java có thể **map các objects với bảng trong hệ quản trị cơ sở dữ liệu quan hệ**.
- **Hibernate** giúp **thực hiện giao tiếp giữa tầng ứng dụng với tầng dữ liệu** (kết nối, truy xuất, lưu trữ…).
- Mỗi **table trong database là một object trong Hibernate**. Do đó, cần có một java bean cho mỗi table trong database.

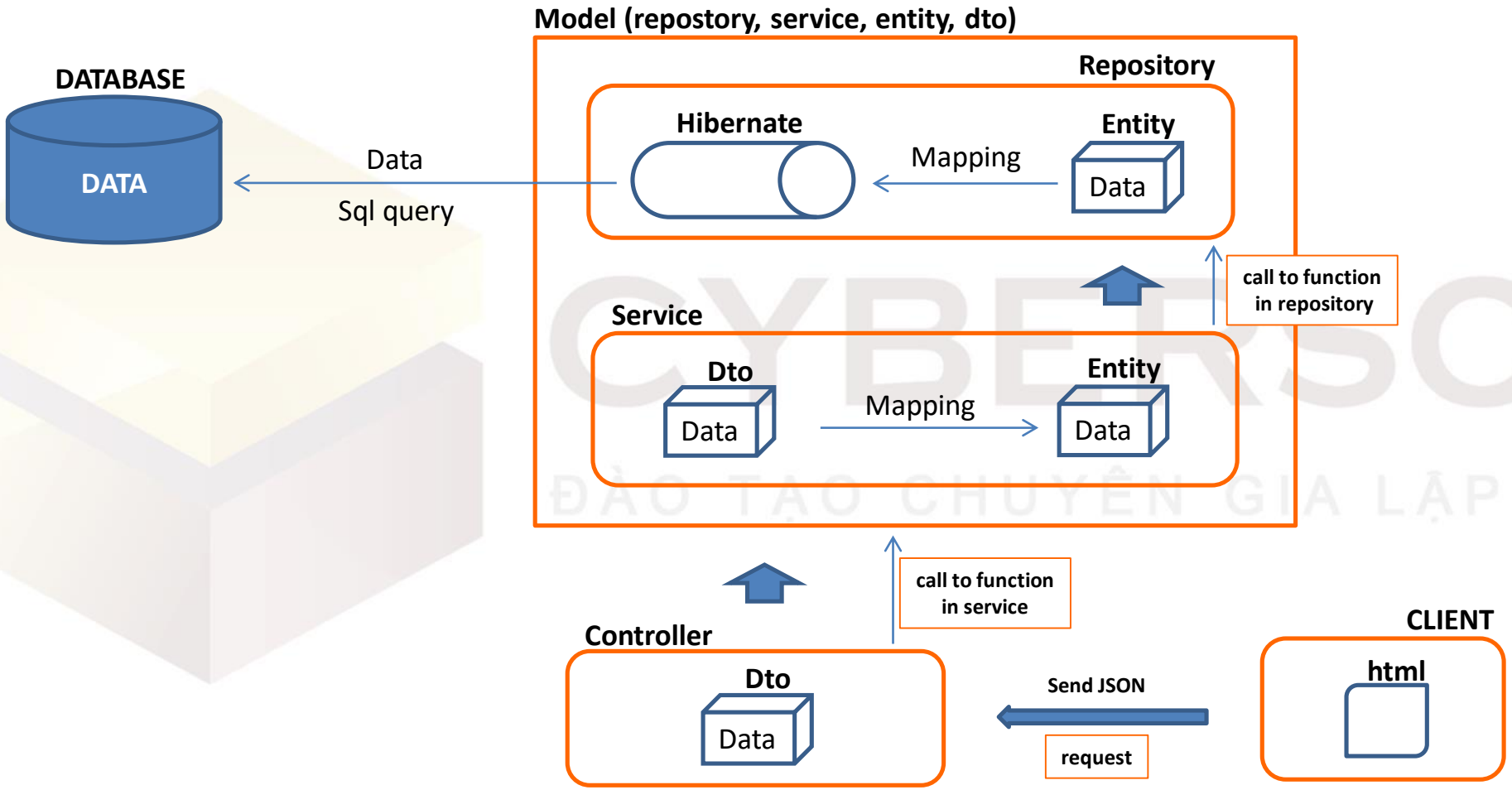
Hibernate cải thiện điều gì?

- **Hibernate** giúp cải thiện một số vấn đề gặp phải khi sử dụng JDBC:
  - ✓Hạn chế **lặp đi lặp lại những dòng code giống nhau** trong ứng dụng chỉ để lấy dữ liệu từ database.
  - ✓Giúp lập trình viên **đỡ phải vất vả với việc map giữa Object Java với các table** tương ứng trong database.
  - ✓Giúp **giảm thiểu công sức để thay đổi từ hệ quản trị CSDL** này sang một hệ quản trị CSDL khác (Hibernate viết một lần, chạy trên mọi loại CSDL).
  - ✓Khắc phục những khó khăn trong việc **tạo các giao tiếp/liên hệ giữa các table**, **lập trình OOPs**.

Lấy dữ liệu


Model (repostory, service, entity, dto)
DATABASE
DATA
Data
send sql query
Repository
Hibernate
Mapping
Entity
Data
call to function
in repository
Service
Dto
Data
Mapping
Entity
Data
call to function
in service
Controller
Dto
Data
Return JSON
response
CLIENT
html

Cập nhật dữ liệu



Ưu và nhược điểm Hibernate

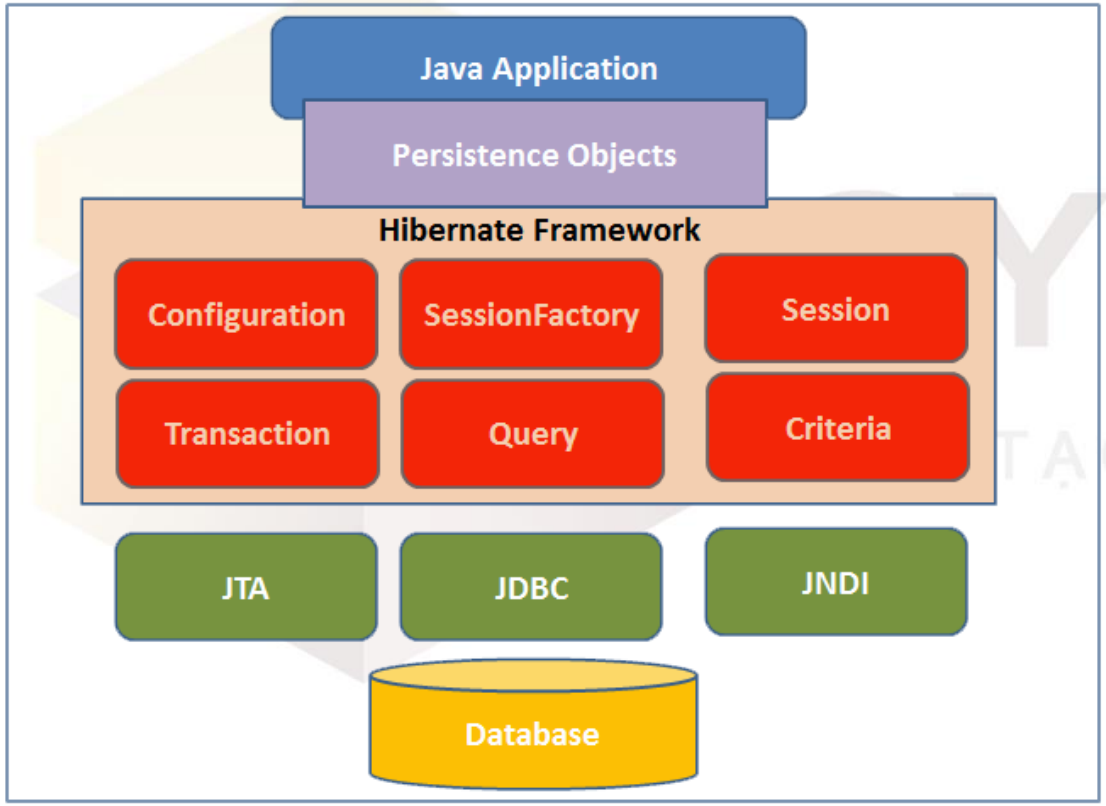
❑ Ưu điểm

- ✓ **Dễ sử dụng**: dễ dàng quản lý các kết nối database và dễ fix bug, cung cấp sẵn nhiều API truy vấn.
- ✓ **Tính độc lập**: không cần quan tâm tới cơ sở dữ liệu sử dụng khi viết câu lệnh SQL.
- ✓ **Tính hướng đối tượng**: tập trung xử lý theo hướng đối tượng, phù hợp sử dụng trong các case Read, Update, Delete.

❑ Nhược điểm

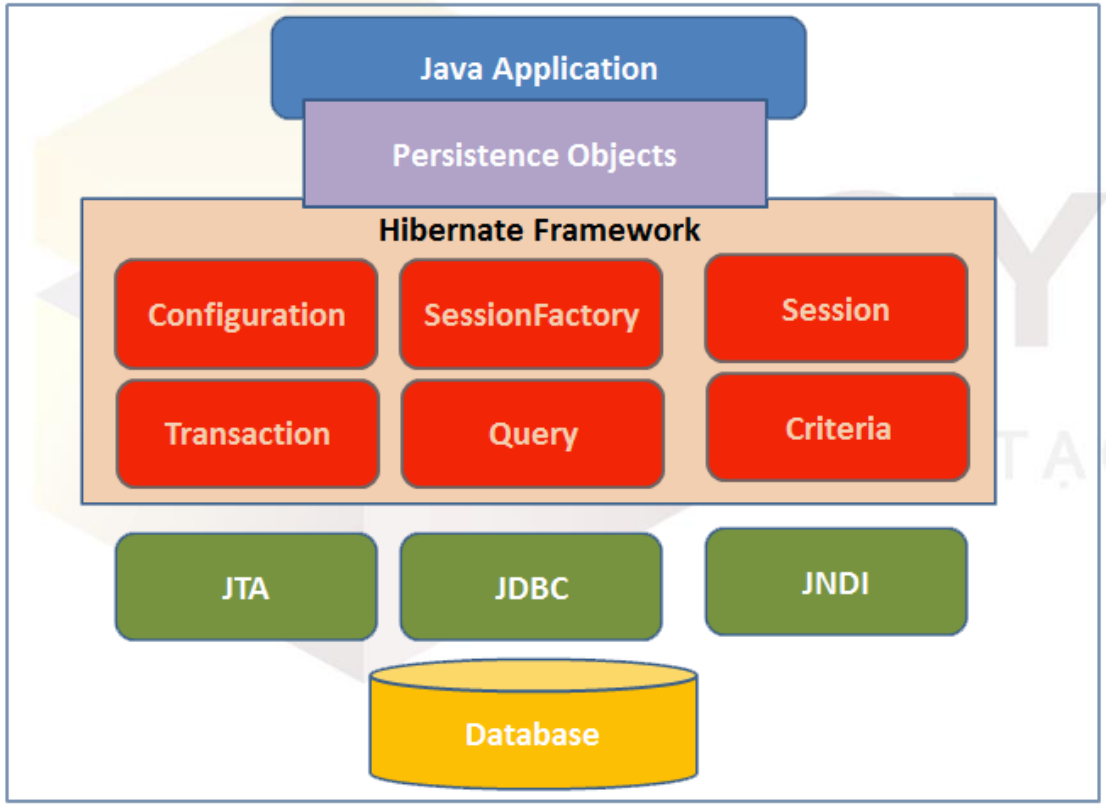
- ✓ Không hỗ trợ các câu truy vấn phức tạp.
- ✓ Một số trường hợp vẫn phải dùng native SQL do Hibernate không thể cover hết tất cả các cú pháp của các hệ quản trị cơ sở dữ liệu.
- ✓ Bị hạn chế sự can thiệp vào câu lệnh SQL do nó được tự động sinh ra.

Cấu trúc Hibernate



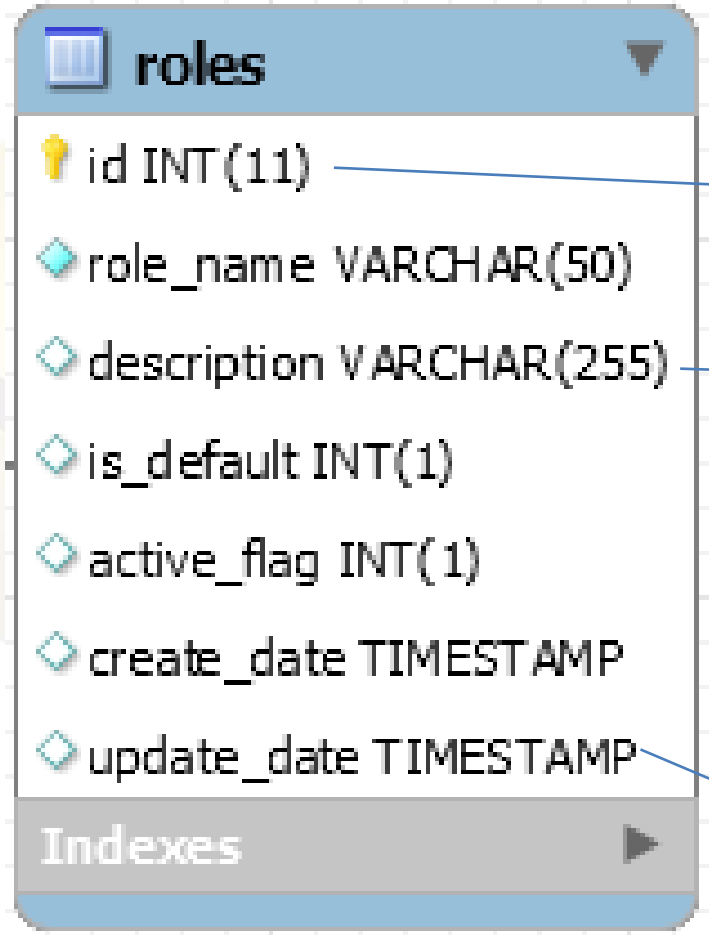
- **Persistence object**: Là các **POJO object map với các table tương ứng** của cơ sở dữ liệu quan hệ.
- **Configuration:** Được sử dụng để **quản lý thông tin cấu hình kết nối** và **ánh xạ thực thể** vào CSDL.
- **Session Factory:** Là một **interface giúp tạo ra session kết nối đến database** bằng cách đọc các cấu hình trong Hibernate configuration.
- **Session:** Mỗi một đối tượng session được Session factory tạo ra sẽ **tạo một kết nối đến database**.

Cấu trúc Hibernate



- **Transation:** Quản lý giao dịch, **đảm bảo thành công hoặc rollback, đảm bảo tính thống nhất và toàn vẹn dữ liệu**. Tức là nếu có một lỗi xảy ra trong transaction thì tất cả các tác vụ thực hiện sẽ thất bại.
- **Query:** **Cung cấp các câu chuy vấn HQL** tới database và **map kết quả trả về với đối tượng tương ứng** của ứng dụng Java.
- **Criteria:** Sử dụng để **xây dựng câu lệnh truy vấn bằng lập trình** thay cho câu lệnh HQL hay SQL.

Ánh xạ thực thể




```java
@Entity(name = "roles")
public class Role {
    @Id
    @GeneratedValue
    private int id;
    @Column(name = "role_name")
    private String name;
    private String description;
    @Column(name = "is_default")
    private boolean isDefault;
    @Column(name = "active_flag")
    private boolean activeFlag;
    @Column(name = "create_date")
    @Temporal(TemporalType.DATE)
    private Date createDate;
    @Column(name = "update_date")
    @Temporal(TemporalType.DATE)
    private Date updateDate;
}
```

Tên bảng

Khóa chính

Tự động tăng

Đặt lại tên nếu thuộc tính không trùng Tên trường Trong CSDL

Kiểu dữ liệu

Ánh xạ quan hệ

❑ Quan hệ n – 1

```java
@Entity(name = "accounts")
public class Account {
    @Id
    private String id;
    private String email;
    private String password;

    @Column(name = "role_id")
    private int roleId;

    @ManyToOne
    @JoinColumn(name = "role_id")
    private Role role;
}
```

@JoinColumn:

Chỉ tên cột khóa ngoại.

❑ Quan hệ 1 – n

```java
@Entity(name = "roles")
public class Role {
    @Id
    @GeneratedValue
    private int id;
    @Column(name = "role_name")
    private String name;

    @OneToMany(mappedBy = "role",
    fetch = FetchType.LAZY)
    private Set<Account> accounts;
}
```

- ❖ **mapBy:** Chỉ ra tên trường thực thể kết hợp.
- ❖ **fetch:** Chế độ nạp kèm thực thể kết hợp
  - ✓ **LAZY:** Không nạp kèm.
  - ✓ **EAGER:** Nạp kèm thực thể.

Ánh xạ quan hệ

❑ Quan hệ n - n

```java
@Entity(name = "accounts")
public class Account {
    @Id
    private String id;
    private String email;
    private String password;

    @OneToMany(mappedBy = "account")
    private Set<AccountCourse> accountCourses;
}
```

```java
@Entity(name = "account_courses")
public class AccountCourse{
    @Id
    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "account_id")
    private Account account;

    @Id
    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "course_id")
    private Course course;
}
```

```java
@Entity(name = "courses")
public class Course {

    @Id
    @GeneratedValue
    private int id;

    @Column(name = "course_name")
    private String name;
    private String description;

    @OneToMany(mappedBy = "course")
    private Set<AccountCourse> accountCourses;
}
```

Annotation Hibernate

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;

Sử dụng **@Transient**
Nếu muốn một thuộc tính của lớp đối tượng không ánh xạ vào bất cứ cột nào trong CSDL.

Cấu hình Hibernate

❑ Cấu hình bean

❖ DriverManagerDataSource

✓ **Kết nối tới CSDL**.

- org.apache.commons.dbcp.BasicDataSource

❖ LocalSessionFactoryBean

✓ **Tạo Session Factory từ kết nối CSDL**.

- org.springframework.orm.hibernate5.LocalSessionFactoryBean

❖ HibernateTransactionManager

✓ **Quản lý Transaction**.

- org.springframework.orm.hibernate5.HibernateTransactionManager

Thư viện sử dụng

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.4.2.Final</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>5.1.5.RELEASE</version>
</dependency>
<dependency>
    <groupId>commons-dbcp</groupId>
    <artifactId>commons-dbcp</artifactId>
    <version>1.4</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>5.1.5.RELEASE</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.15</version>
</dependency>
```

Cấu hình Hibernate XML

❑ DataSource

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="com.mysql.cj.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://localhost:3306/springsecurity"/>
    <property name="username" value="root"/>
    <property name="password" value="123456"/>
</bean>
```

❑ Transaction

```
<bean id="transactionManager"
        class="org.springframework.orm.hibernate5.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
<tx:annotation-driven/>
```

➡ **Cấu hình quản lý transaction tự động**

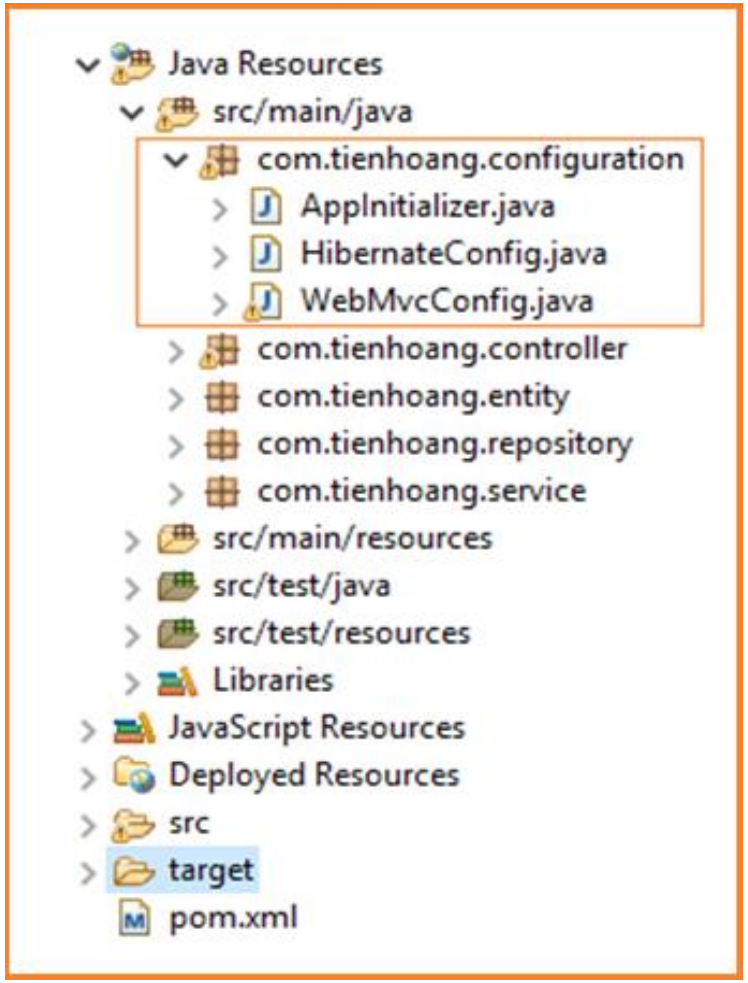
Cấu hình Hibernate XML

❑ SessionFactory

```
<bean id="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
    <property name="dataSource" ref="myDataSource"/>
    <!--Cấu hình package chứa các class mapping dữ liệu-->
    <property name="packagesToScan" value="com.tienhoang.entity" />
    <property name="hibernateProperties">
        <props>
            <!-- Cấu hình hibernate biên dịch ra câu lện MySQL -->
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
            <!--Cấu hình cho phép in câu lệnh SQL ra màn hình console-->
            <prop key="hibernate.show_sql">true</prop>
        </props>
    </property>
</bean>
```

Cấu hình Hibernate Annotation

❑ Cấu trúc thư mục



❑ DispatcherServlet

```
public class AppInitializer extends
    AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        // TODO Auto-generated method stub
        return new Class[] { HibernateConfig.class };
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        // TODO Auto-generated method stub
        return new Class[] { WebMvcConfig.class };
    }

    @Override
    protected String[] getServletMappings() {
        // TODO Auto-generated method stub
        return new String[] { "/" };
    }

}
```

Hibernate Config

Quản lý kết nối database

SessionFactory chịu trách nhiệm tạo và quản lý các Session

```java
@Configuration
public class HibernateConfig {

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
        dataSource.setUrl("jdbc:mysql://localhost:3306/elearning");
        dataSource.setUsername("root");
        dataSource.setPassword("123456");
        return dataSource;
    }

    @Bean
    public LocalSessionFactoryBean sessionFactory() {
        LocalSessionFactoryBean bean = new LocalSessionFactoryBean();
        bean.setDataSource(dataSource());
        bean.setPackagesToScan("com.myclass.entity");
        Properties properties = new Properties();
        properties.put("hibernate.dialect", "org.hibernate.dialect.MySQLDialect");
        properties.put("hibernate.show_sql", true);
        properties.put("hibernate.format_sql", true);
        bean.setHibernateProperties(properties);
        return bean;
    }
}
```

Hibernate Bean

❑ Session Factory

```java
@Bean
public LocalSessionFactoryBean sessionFactory() {
    LocalSessionFactoryBean bean = new LocalSessionFactoryBean();
    // Set datasource cho SessionFactory
    bean.setDataSource(dataSource());
    // Khai báo package chứa các entity mapping với các bảng trong csdl
    bean.setPackagesToScan("com.myclass.entity");
    Properties properties = new Properties();
    // Chỉ định loại SQL đích để Hibernate biên dịch ra các câu lệnh phù hợp với csdl
    properties.put("hibernate.dialect", "org.hibernate.dialect.MySQLDialect");
    // Cấu hình này cho phép in câu lệnh truy vấn trong console
    properties.put("hibernate.show_sql", true);
    // Format các câu truy vấn in ra trong console
    properties.put("hibernate.format_sql", true);
    bean.setHibernateProperties(properties);
    return bean;
}
```

Session trong Hibernate

- **Session** được sử dụng để **tạo một kết nối vật lý với một cơ sở dữ liệu**.
- Các đối tượng **Session** được **khởi tạo mỗi khi cần tương tác với cơ sở dữ liệu**.
- Chức năng chính của session là **cung cấp các thao tác create**, **update**, **read** và **delete** cho các thể hiện các lớp thực thể được ánh xạ.
- Các đối tượng **Session** **không nên giữ mở trong một thời gian dài** bởi vì an toàn và cần được tạo ra và hủy khi cần thiết.
- Đối tượng **Session** **được tạo ra bởi SessionFactoty** vì vậy cần inject **SessionFactory** vào khi muốn sử dụng **Session**.

Sử dụng Session

```java
@Repository
public class RoleRepositoryImpl implements RoleRepository{

    @Autowired
    private SessionFactory sessionFactory;

    public void save(Role role) {
        // Tạo đối tượng session để truy vấn database
        Session session = sessionFactory.openSession();
        Transaction transaction = null;
        try {
            // Khởi tạo transaction để quản lý giao dịch
            transaction = session.beginTransaction();

            // Mở kết nối, tạo câu truy vấn
            session.saveOrUpdate(role);

            // Thực hiện cập nhật dữ liệu
            transaction.commit();
        }
        catch (Exception e) {
            e.printStackTrace();
            // Nếu có lỗi => rollback
            transaction.rollback();
        }
        finally {
            // Đóng session sau khi thực hiện xong
            session.close();
        }
    }
}
```

Inject Session Factory

Tạo một đối tượng Session

Quản lý giao dịch (commit và rollback)

Hủy Session và giải phóng kết nối JDBC

Session method

- **createQuery**() : Tạo câu truy vấn HQL

```
public List<User> findAll(){
    Session session = sessionFactory.getCurrentSession();
    Query<User> query = session.createQuery("FROM users", User.class);
    return query.getResultList();
}
```

- **find**(): Tìm kiếm theo id

```
public User findById(String id){
    Session session = sessionFactory.getCurrentSession();
    return session.find(User.class, id);
}
```

Session method

- **saveOrUpdate**(): Thêm mới hoặc cập nhật.

```
public void save(User model){
    Session session = sessionFactory.getCurrentSession();
    session.saveOrUpdate(model);
}
```

- **delete**(): Xóa một bản ghi.

```
public void delete(String id){
    Session session = sessionFactory.getCurrentSession();
    User model = session.find(User.class, id);
    session.remove(model);
}
```

OpenSession & getCurrentSession

❑ getCurrentSession()

- ✓ Phương thức **getCurrentSession()** sẽ **tự động đẩy dữ liệu đi** (session.flush()) và **tự động đóng session lại** (session.close()).
- ✓ Trường hợp **chỉ sử dụng session một lần** thì ta sử dụng phương thức getCurrentSession().

❑ openSession()

- ✓ Phương thức **openSession()** khi thực hiện xong một hành động nào đó (thêm, xóa, sửa) thì **session vẫn còn giữ và không tự động đẩy dữ liệu đi** mà lập trình viên phải tự làm việc này.

Quản lý Transaction tự động

❑ Để bật chức năng quản lý transaction tự động ta dùng annotation:

@Transactional(rollbackOn = **Exception.class**)

- ✓ **@Transaction** ở đầu class thì tất cả các method trong class đó đều nằm trong một transaction.
- ✓ **@Transaction** ở đầu method thì chỉ các method đó được nằm trong một transaction.
- ✓ **rollbackOn = Exception.class**: Chỉ rõ khi xảy ra loại exception nào thì rollback. Ở đây là khi xảy exception bất kì thì đều rollback lại.
- ✓ Những method được nằm trong một transaction thì chúng ta sẽ gọi **sessionFactory.getCurrentSession()** để lấy session chứ không gọi **sessionFactory.openSession()**.

Cấu hình Transaction

```java
@Configuration
@EnableTransactionManagement
public class HibernateConfig {

    @Bean
    public HibernateTransactionManager transactionManager() {
        HibernateTransactionManager manager = new HibernateTransactionManager();
        // Set SessionFactory cho Transaction
        manager.setSessionFactory(sessionFactory().getObject());
        return manager;
    }

    public DataSource dataSource() {

    public LocalSessionFactoryBean sessionFactory() {
}
```

Kích hoạt quản lý transaction tự động

Cấu hình Transaction

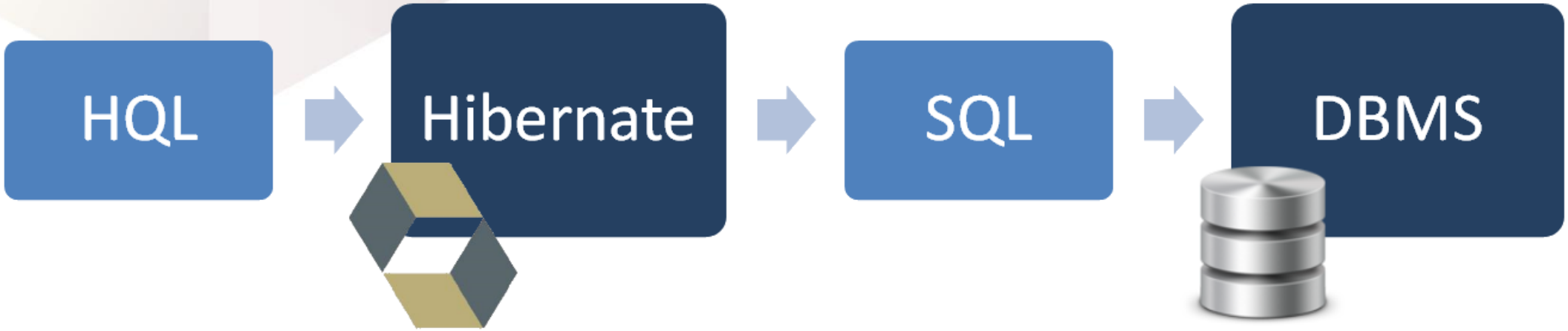
Sử dụngTransaction

Khai báo Transaction tự động Rollback nếu có lỗi xảy ra

```
@Repository
@Transactional(rollbackOn = Exception.class)
public class RoleRepositoryImpl implements RoleRepository{

    @Autowired
    private SessionFactory sessionFactory;

    public void save(Role role) {
        // Tạo đối tượng session để truy vấn database
        Session session = sessionFactory.getCurrentSession();
        try {
            // Mở kết nối, tạo câu truy vấn
            session.saveOrUpdate(role);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

HQL là gì?

- **HQL** (Hibernate Query Language) là một **ngôn ngữ truy vấn hướng đối tượng**.
- **HQL thao tác với thuộc tính của lớp đối tượng** (class) thay vì thao tác với cột/hàng của CSDL như SQL.
- **Các câu truy vấn** được viết bằng HQL sẽ được **biên dịch bởi Hibernate và chuyển thành các câu truy vấn SQL thông thường** để giao tiếp với CSDL.



HQL - Select dữ liệu

- Select tất cả các trường trong bảng.

```
String hql = "FROM users";
Query<User> query = session.createQuery(hql, User.class);
List<User> results = query.list();
```

- Select một số trường trong bảng.

```
String hql = "SELECT id, email, fullname FROM users";
Query<User> query = session.createQuery(hql, User.class);
List<User> results = query.list();
```

HQL - Mệnh đề Where

- Select dữ liệu với mệnh đề where.

```
String hql = "FROM users WHERE email = 'admin@gmail.com'";
Query<User> query = session.createQuery(hql, User.class);
List<User> results = query.list();
```

- Set tham số truyền vào bằng hàm setParameter.

```
String hql = "FROM users WHERE email = :email";
Query<User> query = session.createQuery(hql, User.class);
query.setParameter("email", "admin@gmail.com");
List<User> results = query.list();
```

HQL - Thêm mới dữ liệu

- Thêm mới dữ liệu sử dụng hàm setParameter

```
String hql = "INSERT INTO users(id, email,fullname) " +
             "VALUES (:id, :email, :name)";
Query query = session.createQuery(hql);
query.setParameter("id", 1);
query.setParameter("email", "teo_nguyenvan@gmail.com");
query.setParameter("name", "Nguyễn Văn Tèo");
int result = query.executeUpdate();
```

HQL - Cập nhật - Xóa

- Xóa dữ liệu

```
String hql = "DELETE FROM users WHERE id = :id";
Query query = session.createQuery(hql);
query.setParameter("id", 1);
int result = query.executeUpdate();
```

- Cập nhật dữ liệu

```
String hql = "UPDATE users SET email = :email WHERE id = :id";
Query query = session.createQuery(hql);
query.setParameter("email", "teo_nguyenvan@gmail.com");
query.setParameter("id", 1);
int result = query.executeUpdate();
```

HQL - Truy vấn phân trang

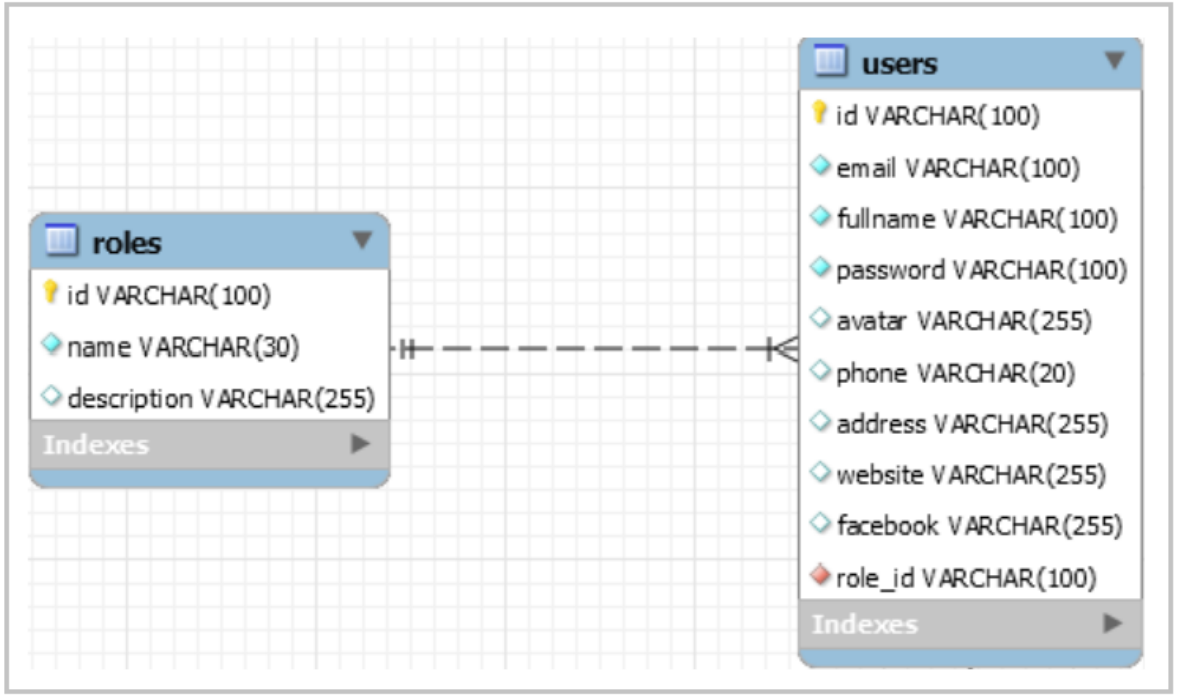
```
String hql = "FROM users";
Query<User> query = session.createQuery(hql, User.class);
query.setFirstResult(1);
query.setMaxResults(10);
List<User> results = query.list();
```

- Mô tả phương thức

1	**Query setFirstResult(int startPosition)** Phương thức này truyền vào một số nguyên là vị trí dòng đầu tiên trong danh sách kết quả, bắt đầu từ 0.
2	**Query setMaxResults(int maxResult)** Phương thức này cố định số lượng lớn nhất trả về.

Bài tập

- ❑ Sử dụng lại bài tập ở buổi hôm trước.
  - ✓ Tạo cơ sở dữ liệu.
  - ✓ Cấu hình Hibernate.
  - ✓ Viết lại các hàm trong tầng Repository sử dụng Hibernate để truy vấn dữ liệu.
  - ✓ Viết thêm tầng Service.



Các bước thực hiện

- ✓ **Bước 1**: Tạo cơ sở dữ liệu.
- ✓ **Bước 2**: Cấu hình Hibernate.
- ✓ **Bước 3**: Ánh xạ quan hệ cho bảng User và Role.
- ✓ **Bước 4**: Định nghĩa các phương thức CRUD cho tầng Repository sử dụng Hibernate.
- ✓ **Bước 5**: Định nghĩa các phương thức cho tầng Service.
- ✓ **Bước 6**: Sửa lại tầng controller.

Tạo cơ sở dữ liệu

❑ Bảng ROLE

```
CREATE TABLE IF NOT EXISTS roles (
    id VARCHAR(100) NOT NULL,
    name VARCHAR(30) NOT NULL,
    description VARCHAR(255),
    CONSTRAINT pk_role PRIMARY KEY(id)
);
```

❑ Bảng USER

```
CREATE TABLE IF NOT EXISTS users (
    id VARCHAR(100) NOT NULL PRIMARY KEY,
    email VARCHAR(100) NOT NULL,
    fullname VARCHAR(100) NOT NULL,
    password VARCHAR(100) NOT NULL,
    avatar VARCHAR(255),
    phone VARCHAR(20),
    address VARCHAR(255),
    website VARCHAR(255),
    facebook VARCHAR(255),
    role_id VARCHAR (100) NOT NULL, CONSTRAINT
            fk_user_role FOREIGN KEY (role_id)
    REFERENCES roles(id) ON DELETE CASCADE
);
```

Entity

User Entity

```
@Entity(name = "users")
public class User {
    @Id
    private String id;
    @NotBlank(message = "Vui lòng nhập email!")
    @Email(message = "Email không đúng định dạng!")
    private String email;
    @NotBlank(message = "Vui lòng nhập họ tên!")
    private String fullname;
    @NotBlank(message = "Vui lòng nhập mật khẩu!")
    @Length(min = 6, max = 12, message = "Mật khẩu từ 6 - 12 ký tự!")
    private String password;
    private String avatar;
    private String phone;
    private String address;
    private String website;
    private String facebook;

    @Column(name = "role_id")
    @NotBlank(message = "Vui lòng chọn loại người dùng!")
    private String roleId;

    @ManyToOne()
    @JoinColumn(name = "role_id",
        insertable = false, updatable = false)
    private Role role;
}
```

Role Entity

```
@Entity(name = "roles")
public class Role {
    @Id
    private String id;
    @NotBlank(message = "Vui lòng nhập tên!")
    private String name;
    @NotBlank(message = "Vui lòng nhập mô tả!")
    private String description;

    @OneToMany(mappedBy = "role", fetch = FetchType.LAZY)
    private List<User> users;
}
```

Role Repository

Interface

```java
public interface RoleRepository {
    public List<Role> findAll();
    public Role findById(String id);
    public Role findByName(String name);
    public void save(Role model);
    public void delete(String id);
}
```

Implement

```java
@Repository
@Transactional(rollbackFor = Exception.class)
public class RoleRepositoryImpl implements RoleRepository{

    @Autowired
    SessionFactory sessionFactory;

    public List<Role> findAll(){
        Session session = sessionFactory.getCurrentSession();
        Query<Role> query = session.createQuery("FROM roles", Role.class);
        return query.getResultList();
    }

    public Role findById(String id){
        Session session = sessionFactory.getCurrentSession();
        return session.find(Role.class, id);
    }

    public void save(Role model){
        Session session = sessionFactory.getCurrentSession();
        session.saveOrUpdate(model);
    }

    public void delete(String id){
        Session session = sessionFactory.getCurrentSession();
        Role role = session.find(Role.class, id);
        session.remove(role);
    }

    public Role findByName(String name){
}
```

User Repository

Interface

```java
public interface UserRepository {
    public List<User> findAll();
    public User findById(String id);
    public User findByEmail(String email);
    public void save(User model);
    public void delete(String id);
}
```

Implement

```java
@Repository
@Transactional(rollbackFor = Exception.class)
public class UserRepositoryImpl implements UserRepository{
    @Autowired
    private SessionFactory sessionFactory;

    public List<User> findAll() {
        Session session = sessionFactory.getCurrentSession();
        Query<User> query = session.createQuery("FROM users", User.class);
        return query.getResultList();
    }

    public User findById(String id) {
        Session session = sessionFactory.getCurrentSession();
        return session.find(User.class, id);
    }

    public User findByEmail(String email) {
        Session session = sessionFactory.getCurrentSession();
        Query<User> query = session.createQuery("FROM users where email = :email", User.class);
        query.setParameter("email", email);
        List<User> result = query.getResultList();
        if(result.size() > 0) {
            return result.get(0);
        }
        return null;
    }

    public void save(User model) {
        Session session = sessionFactory.getCurrentSession();
        session.saveOrUpdate(model);
    }

    public void delete(String id) {
}
```

Database resources

❑ Sử dụng file **properties** để loại bỏ **hash code** khi cấu hình Hibernate.

❖ Các bước thực hiện:

✓ Bước 1: Tải thư viện spring-context.

✓ Bước 2: Tạo file db.properties (nằm trong src/main/resource)

✓ Bước 3: Khai báo file db. properties trong class HibernateConfig.

✓ Bước 4: Tạo bean **PropertySourcesPlaceholderConfigurer** (file HibernateConfig).

✓ Bước 5: Inject Environment vào để lấy các property từ file db. properties.

✓ Bước 6: Sửa lại các cấu hình đang bị hash code.

db.properties

Thư viện sử dụng

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.1.5.RELEASE</version>
</dependency>
```

Khai báo các thuộc tính cho cấu hình Hibernate trong db.properties

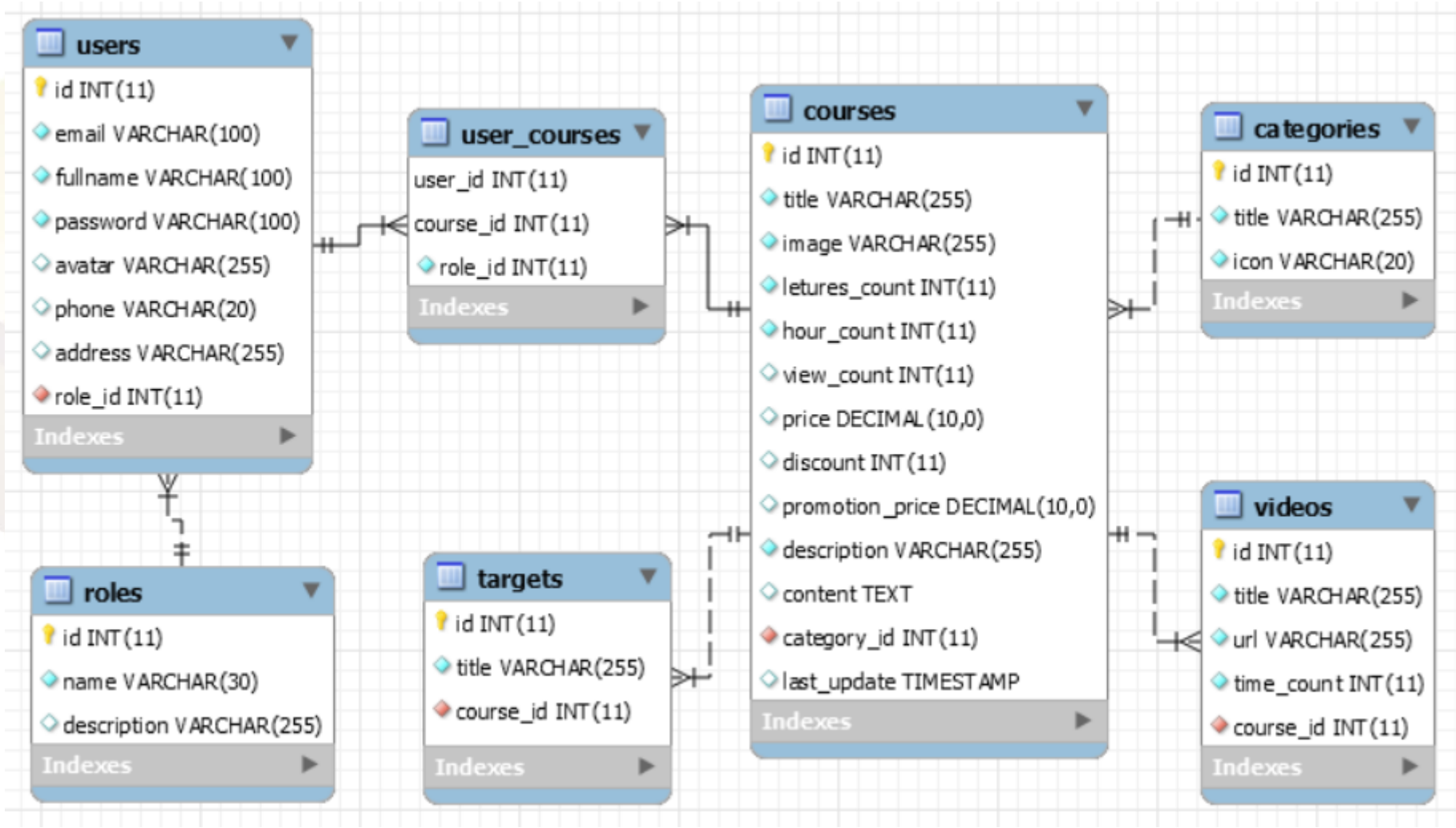
```
mysql.driver=com.mysql.cj.jdbc.Driver
mysql.url=jdbc:mysql://localhost:3306/elearning
mysql.username=root
mysql.password=123456

hibernate.package_scan=com.myclass.entity
hibernate.dialect=org.hibernate.dialect.MySQLDialect
hibernate.show_sql=true
```

HibernateCofig

```java
@Configuration
@EnableTransactionManagement
@PropertySource("classpath:db.properties")
public class HibernateConfig {

    @Autowired
    private Environment environment;

    @Bean
    public static PropertySourcesPlaceholderConfigurer placeholderConfigurer() {
        return new PropertySourcesPlaceholderConfigurer();
    }

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName(environment.getProperty("mysql.driver"));
        dataSource.setUrl(environment.getProperty("mysql.url"));
        dataSource.setUsername(environment.getProperty("mysql.username"));
        dataSource.setPassword(environment.getProperty("mysql.password"));
        return dataSource;
    }
    @Bean
    public LocalSessionFactoryBean sessionFactory() {
        LocalSessionFactoryBean bean = new LocalSessionFactoryBean();
        bean.setDataSource(dataSource());
        bean.setPackagesToScan(environment.getProperty("hibernate.package_scan"));
        Properties properties = new Properties();
        properties.put("hibernate.dialect", environment.getProperty("hibernate.dialect"));
        properties.put("hibernate.show_sql", environment.getProperty("hibernate.show_sql"));
        bean.setHibernateProperties(properties);
        return bean;
    }
    public HibernateTransactionManager transactionManager() {
}
```

Bài tập về nhà

- Hoàn thành các bảng còn lại trong dự án.



Tổng kết

- ✓ ORM là gì?
- ✓ Hibernate là gì?
- ✓ Cấu trúc của Hibernate.
- ✓ Ánh xạ dữ liệu trong Hibernate.
- ✓ Cấu hình Hibernate.
- ✓ Session method
- ✓ HQL là gì?
- ✓ Cú pháp câu lệnh HQL.