

1



JPARepository API

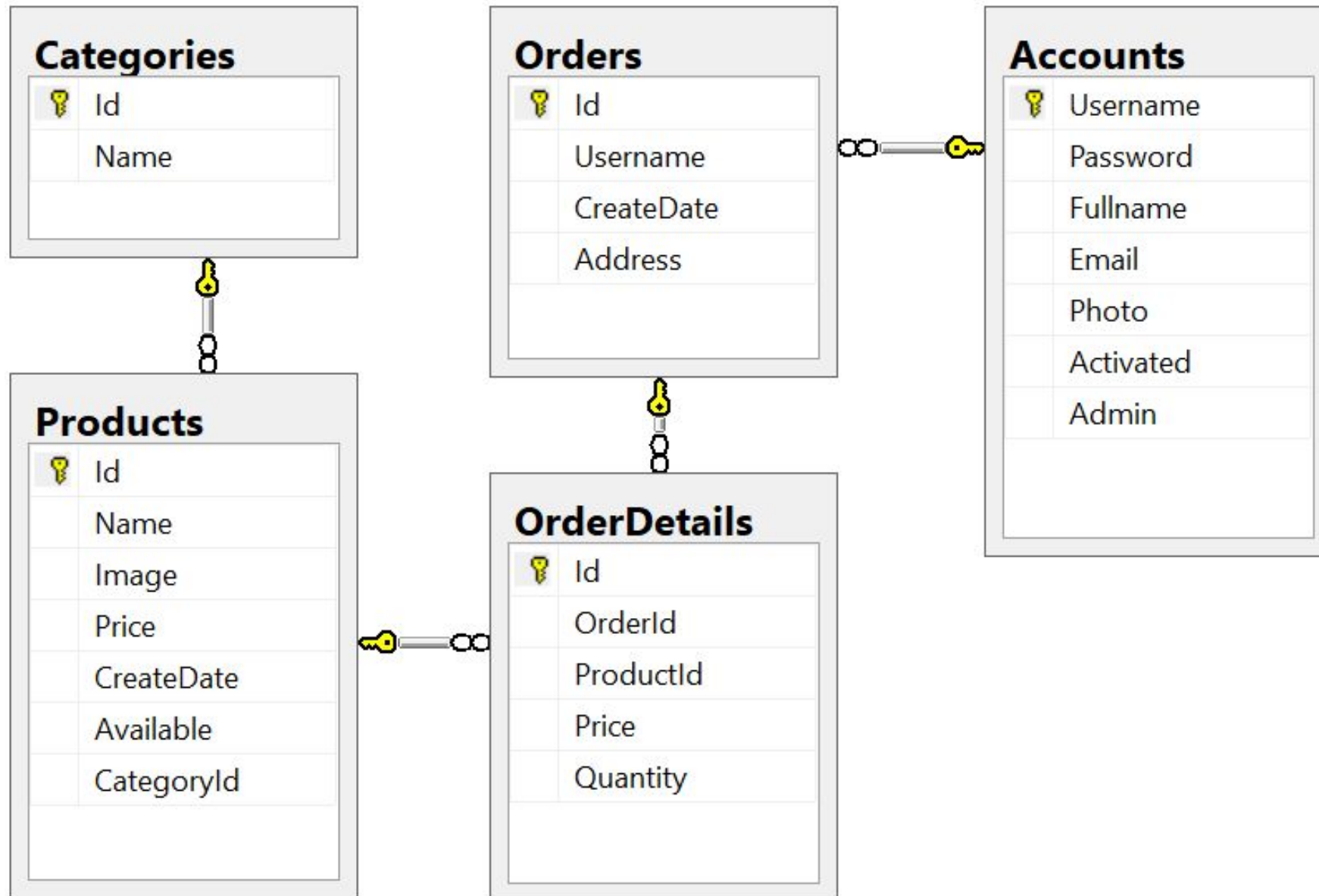
GIẢNG VIÊN:

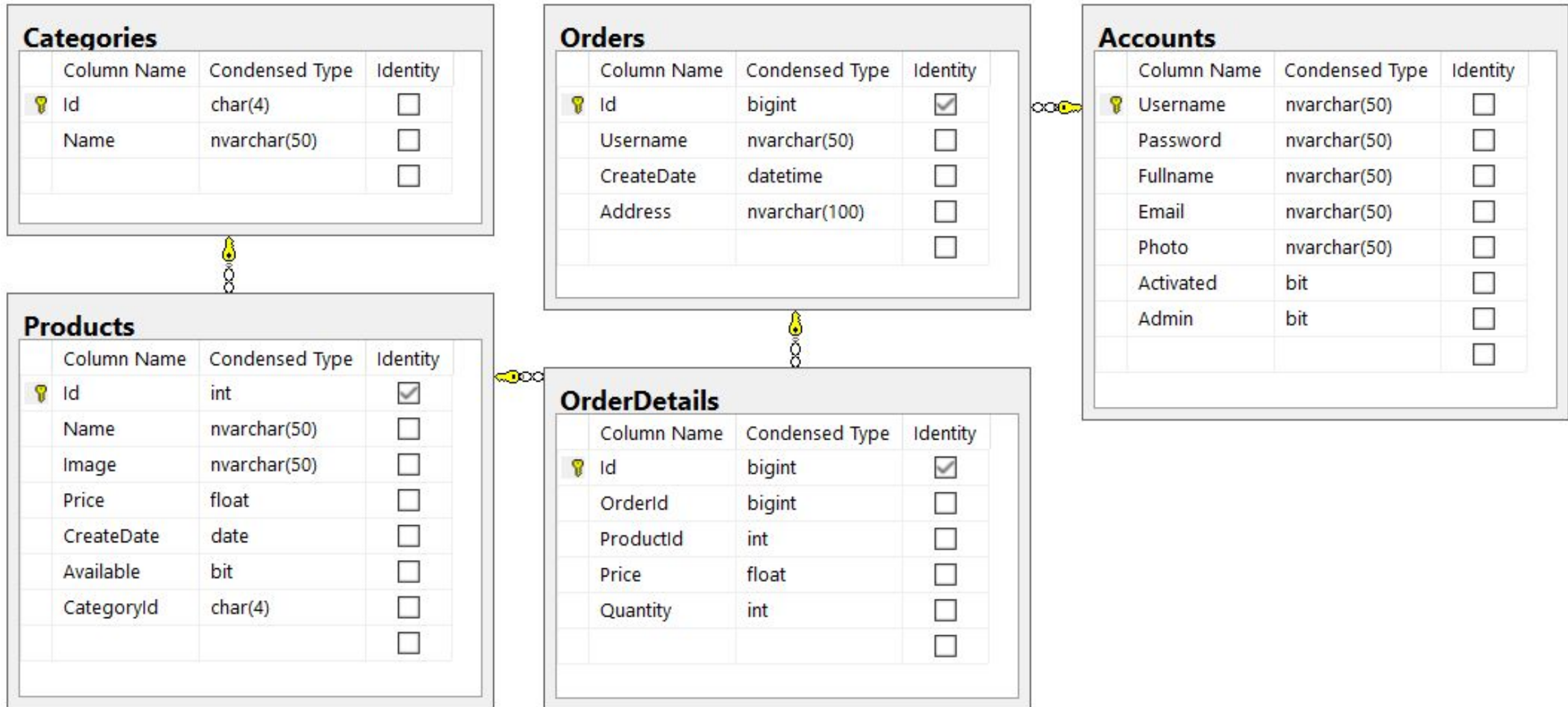
- JPARepository API
 - GIỚI THIỆU CSDL MẪU J5SHOP
 - TẠO CÁC LỚP THỰC THỂ
 - SƠ ĐỒ PHÂN CẤP THỪA KẾ JPARepository
 - TẠO CÁC DAO LÀM VIỆC VỚI J5SHOP
 - XÂY DỰNG ỨNG DỤNG CRUD
- SẮP XẾP VÀ PHÂN TRANG VỚI JPARepository
 - TRUY VẤN CÓ SẮP XẾP
 - TRUY VẤN CÓ PHÂN TRANG
 - SẮP XẾP VÀ PHÂN TRANG SẢN PHẨM





GIỚI THIỆU CSDL MẪU







CẤU HÌNH JPARepository

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>
```

```
<dependency>  
  <groupId>com.microsoft.sqlserver</groupId>  
  <artifactId>mssql-jdbc</artifactId>  
  <scope>runtime</scope>  
</dependency>
```



```
spring.datasource.url=<dburl>  
spring.datasource.username=<username>  
spring.datasource.password=<password>  
spring.datasource.driverClassName=<driver>
```

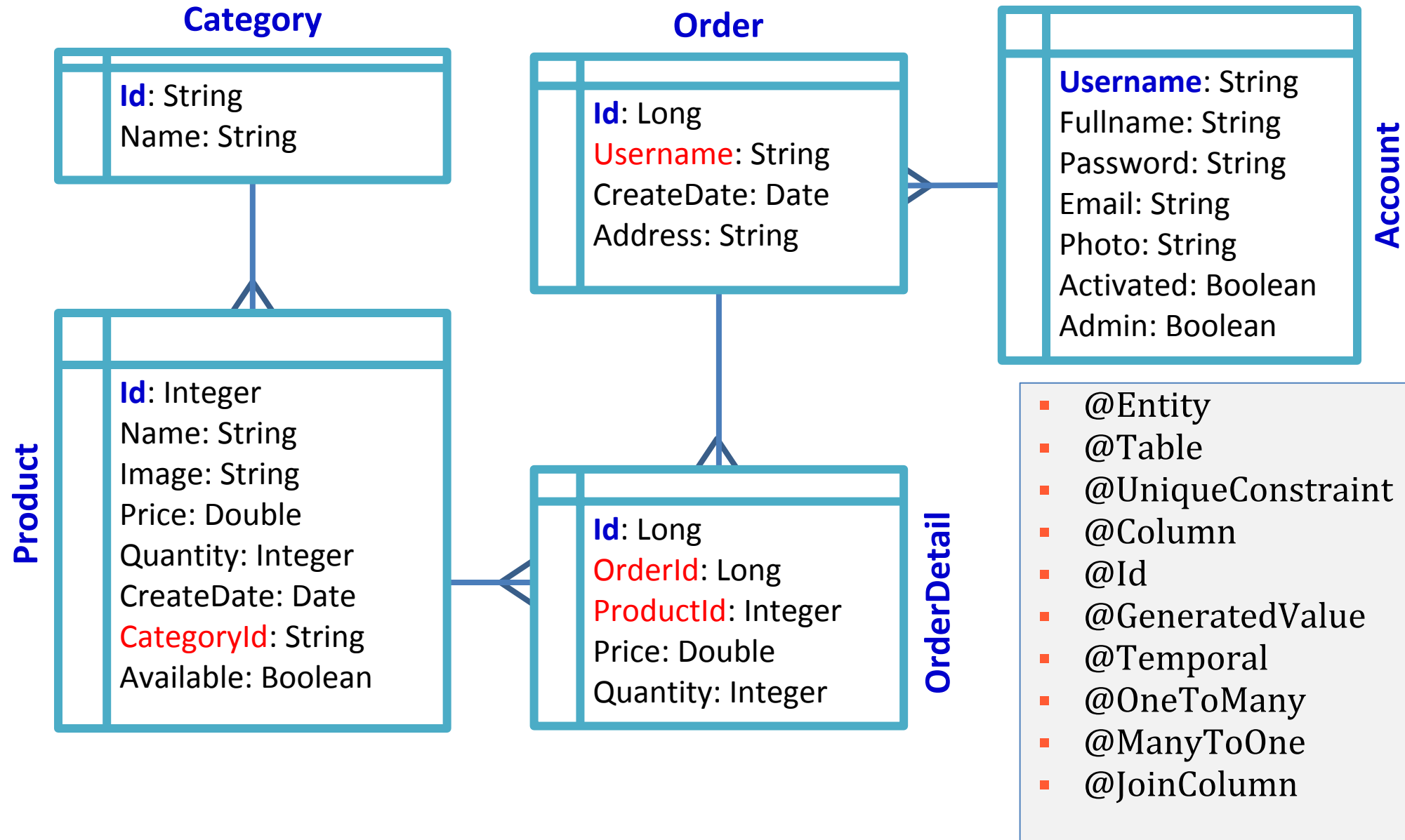
```
spring.jpa.hibernate.dialect=<dialect>  
spring.jpa.hibernate.ddl-auto=<none|create|create-drop|validate|update>
```

```
spring.jpa.show-sql=<true|false>  
spring.jpa.properties.hibernate.format_sql=<true|false>
```




TẠO CÁC LỚP THỰC THỂ

ENTITY RELATIONAL DIAGRAM



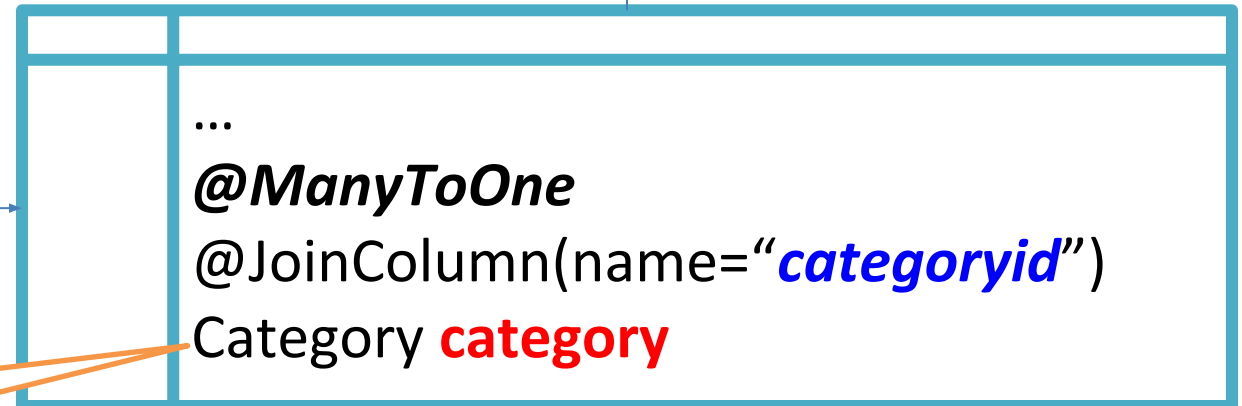
Category



2. @OneToMany
Map mới @ManyToOne

@OneToMany

Product



@ManyToOne

1. @ManyToOne
Thay foreign key bằng
entity

```
@Data
```

```
@Entity
```

```
@Table(name = "Orderdetails", uniqueConstraints = {  
    @UniqueConstraint(columnNames = {"Orderid", "Productid"})  
})
```

```
public class OrderDetail implements Serializable{  
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)  
    Long id;  
    Double price;  
    Integer quantity;  
    @ManyToOne @JoinColumn(name = "Productid")  
    Product product;  
    @ManyToOne @JoinColumn(name = "Orderid")  
    Order order;  
}
```

- ☐ @Entity
- ☐ @Table
- ☐ @UniqueConstraint
- ☐ @Id
- ☐ @GeneratedValue
- ☐ @Column
- ☐ @ManyToOne
- ☐ @JoinColumn

@Data

@Entity

@Table(name = "Orders")

public class Order **implements** Serializable{

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

 Long id;

 String address;

 @Temporal(TemporalType.**DATE**)

 @Column(name = "Createdate")

 Date createDate = **new** Date();

 @ManyToOne @JoinColumn(name = "Username")

 Account account;

 @OneToMany(mappedBy = "order")

 List<OrderDetail> orderDetails;

}

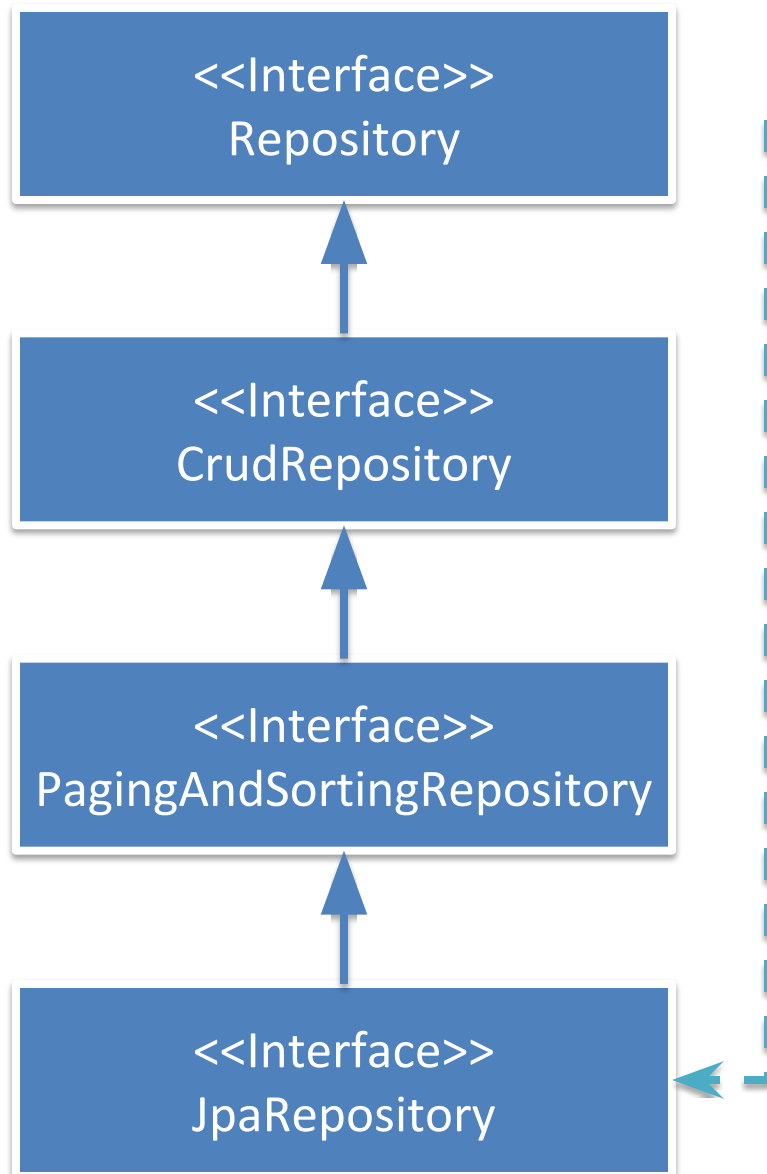
☐ @Temporal

☐ @OneToMany



JPARepository API

JPARepository INHERITANCE HIERARCHY



- ❑ Thông qua sơ đồ phân cấp thừa kế này, có 2 lựa chọn để lập trình với JpaRepository
 - 1. Sử dụng **SimpleJpaRepository**.
 - 2. Xây dựng lớp DAO implements **JpaRepository**
- ❑ Với Spring Boot, thay vì xây dựng các lớp DAO, ta **chỉ việc tạo các interface thừa kế từ JpaRepository và viết thêm các phương thức truy vấn cần thiết**, hệ thống sẽ tự động tạo các Spring Bean thực thi theo interface của chúng ta.

<<Interface>> CrudRepository<T, ID>

<S extends T> S **save**(S entity)
 void **delete**(T entity)
 T **getOne**(ID id)
 Optional<T> **findById**(ID id)
 Iterable<T> **findAll**()
 Long **count**()
 boolean **exists**(ID id)

<<Interface>> JpaRepository<T, ID>

List<T> **findAll**()
 List<T> **findAll**(Sort sort)
 List<T> **save**(Iterable<? extends T> entities)
 void **flush**()
 T **saveAndFlush**(T entity)
 void **deleteInBatch**(Iterable<T> entities)

<<Interface>> PagingAndSortingRepository<T, ID>

Iterable<T> **findAll**(Sort sort)
 Page<T> **findAll**(Pageable pageable)

- ❑ Generic Types
 - ◆ **T**: Entity Class
 - ◆ **ID**: Primary Key Class

- ❑ <S extends T> S **save**(S entity)
 - ❖ Tạo mới hoặc update nếu entity đã tồn tại
- ❑ void **delete**(T entity), **deleteById**(ID id)
 - ❖ Xóa thực thể
- ❑ T **getOne**(ID id), Optional<T> **findById**(ID id)
 - ❖ Truy vấn thực thể theo id
- ❑ Iterable<T> **findAll**()
 - ❖ Truy vấn tất cả thực thể
- ❑ Long **count**()
 - ❖ Lấy số lượng của tất cả thực thể
- ❑ boolean **exists**(ID id)
 - ❖ Kiểm tra sự tồn tại theo id

❑ Iterable<T> **findAll**(**Sort** sort)

❖ Truy vấn tất cả có sắp xếp

❑ Page<T> **findAll**(**Pageable** pageable)

❖ Truy vấn phân trang có sắp xếp

❑ Trong đó:

❖ **Sort**

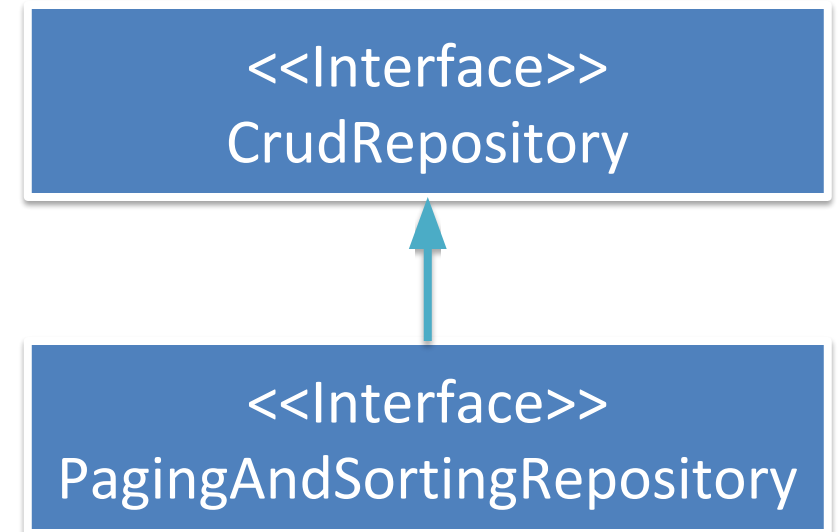
❑ Cung cấp tiêu chí sắp xếp

❖ **Pageable**

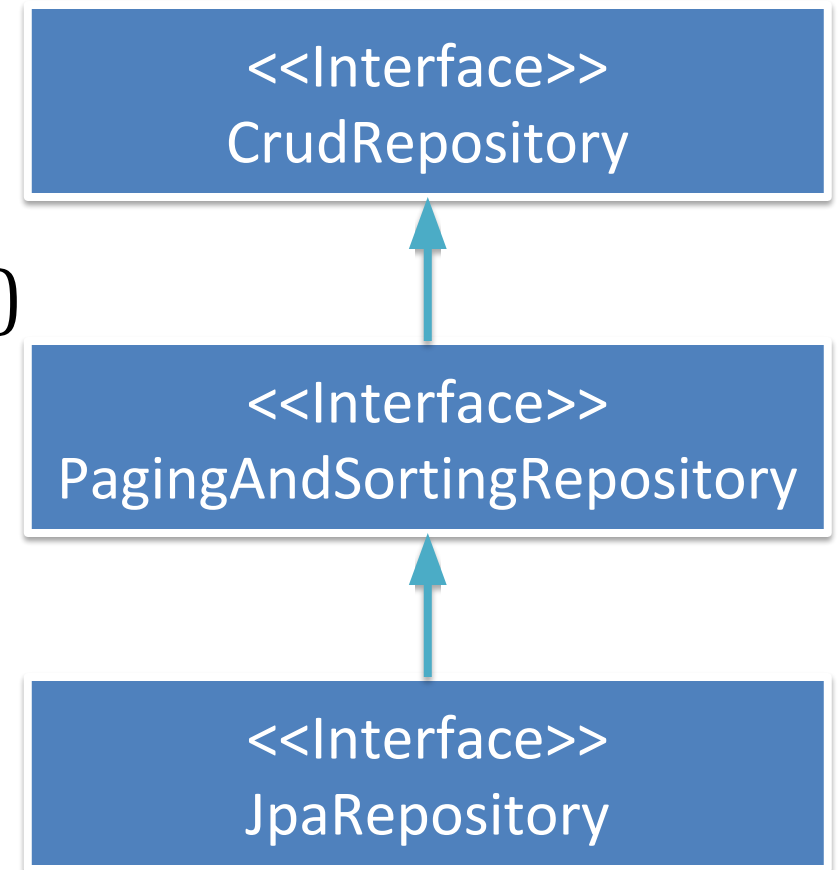
❑ Cung cấp tiêu chí sắp xếp và phân trang (nếu cần)

❖ **Page**

❑ Chứa dữ liệu của trang đã được truy vấn và các thông số kết quả phân trang



- ❑ List<T> **findAll()**
 - ❖ Truy vấn tất cả
- ❑ List<T> **findAll(Sort sort)**
 - ❖ Truy vấn tất cả có sắp xếp
- ❑ List<T> **save(Iterable<? extends T> entities)**
 - ❖ Lưu (tạo hoặc cập nhật) nhiều thực thể
- ❑ void **flush()**
 - ❖ Đồng bộ với CSDL
- ❑ T **saveAndFlush(T entity)**
 - ❖ Lưu và đồng bộ với CSDL
- ❑ void **deleteInBatch(Iterable<T> entities)**
 - ❖ Xóa nhiều thực thể





LẬP TRÌNH JPARepository

CÁCH 1: SỬ DỤNG `SIMPLEJPARepository<T, ID>`

```
@Autowired
```

```
EntityManager em;
```

```
@ResponseBody
```

```
@RequestMapping("/category/list")
```

```
public List<Category> list() {
```

```
    SimpleJpaRepository<Category, String> repo
```

```
        = new SimpleJpaRepository<>(Category.class, em);
```

```
    return repo.findAll();
```

```
}
```

<<Interface>>
`JpaRepository<T, ID>`

implements

<<Class>>
`SimpleJpaRepository<T, ID>`

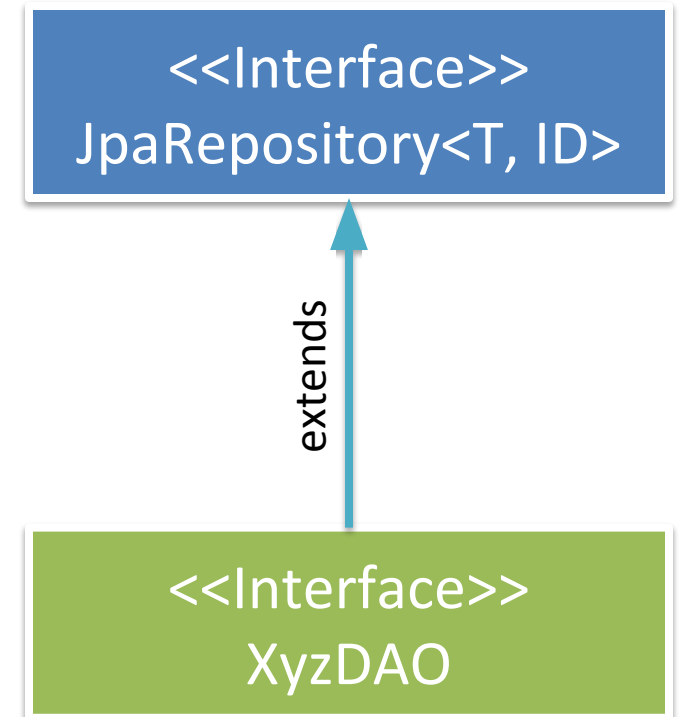
❑ *`SimpleJpaRepository(Class<Entity>, EntityManager)`*

- ❖ Phương thức khởi dựng này tạo một đối tượng DAO tương ứng với một Entity. Từ đó có thể gọi các phương thức phù hợp để làm việc với CSDL.

CÁCH 2: KẾ THỪA JpaRepository<T, ID>

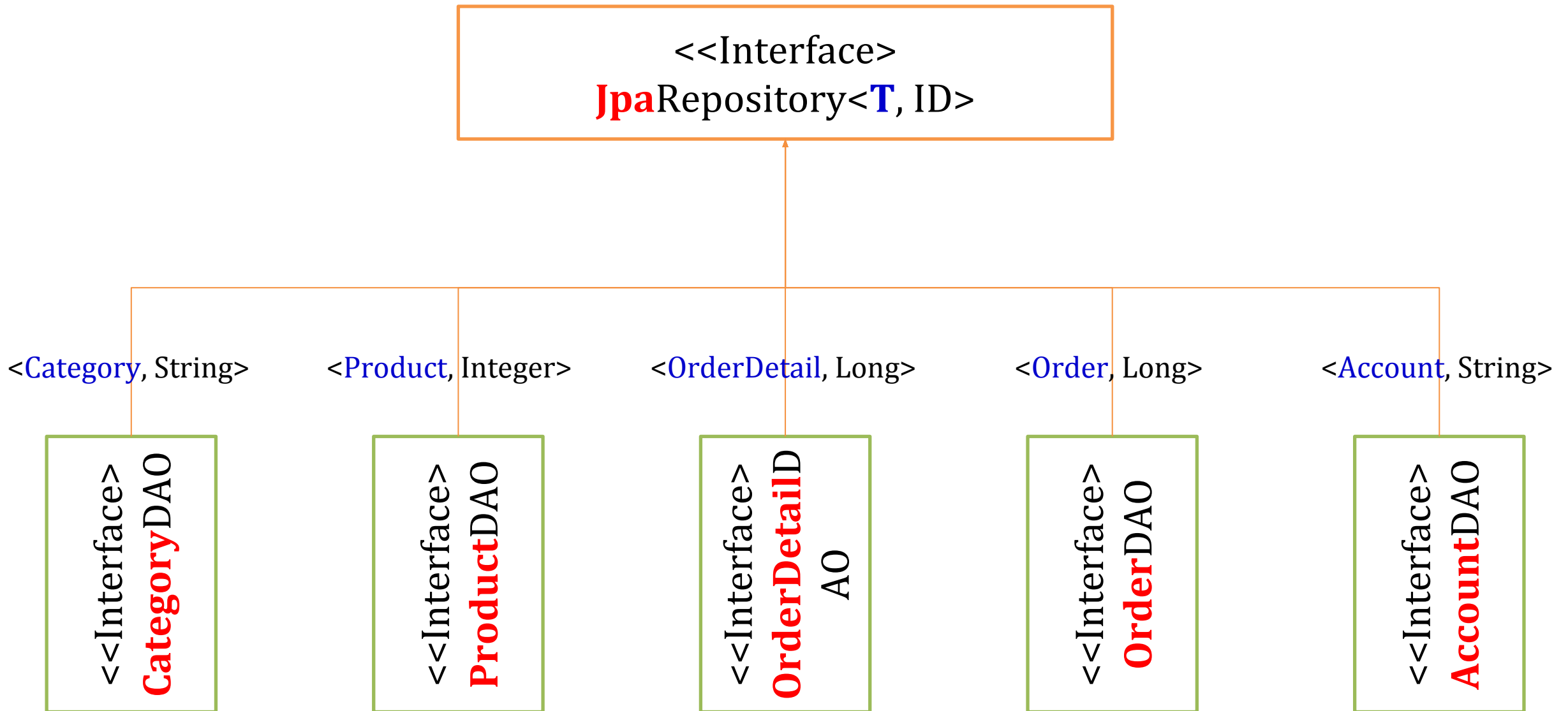
```
public interface CategoryDAO  
    extends JpaRepository<Category, String> {  
}
```

```
@Autowired  
CategoryDAO repo;  
  
@ResponseBody  
@RequestMapping("/category/list")  
public List<Category> list() {  
    return repo.findAll();  
}
```





XÂY DỰNG J5SHOP REPOSITORY



```
public interface CategoryDAO extends JpaRepository<Category, String>{
```

```
public interface ProductDAO extends JpaRepository<Product, Integer>{
```

```
public interface AccountDAO extends JpaRepository<Account, String>{
```

```
public interface OrderDAO extends JpaRepository<Order, Long>{
```

```
public interface OrderDetailDAO extends JpaRepository<OrderDetail, Long>{
```

@Controller

```
public class CrudController {  
    @Autowired  
    AccountDAO dao;  
    @RequestMapping("/account/list")  
    public List<Account> list() {...}  
    @RequestMapping("/account/edit/{username}")  
    public Account edit(@PathVariable("username") String username) {...}  
    @RequestMapping("/account/create")  
    public Account create(Account item) {...}  
    @RequestMapping("/account/update")  
    public Account update(Account item) {...}  
    @RequestMapping("/account/delete/{username}")  
    public void delete(@PathVariable("username") String username) {...}  
}
```

Tiêm DAO cần thiết vào
Controller

```
@RequestMapping("/account/list")
```

```
public List<Account> list() {
```

```
    List<Account> list = dao.findAll();
```

```
    return list;
```

```
}
```

```
@RequestMapping("/account/edit/{username}")
```

```
public Account edit(@PathVariable("username") String username) {
```

```
    Account account = dao.getOne(username);
```

```
    return account;
```

```
}
```

```
@RequestMapping("/account/create")
public Account create(Account item) {
    if(!dao.existsById(item.getUsername())) {
        dao.save(item);
        return item;
    }
    throw new RuntimeException("Mã tài khoản này đã tồn tại.");
}

@RequestMapping("/account/delete/{username}")
public void delete(@PathVariable("username") String username) {
    if(dao.existsById(username)) {
        dao.deleteById(username);
    }
    throw new RuntimeException("Không tìm thấy tài khoản này.");
}
```


2



TRUY VẤN CÓ SẮP XẾP VÀ PHÂN TRẠNG

GIẢNG VIÊN:

❑ **List<Entity> findAll(*Sort*)**

- ❖ Cho phép chúng ta cung cấp tiêu chuẩn sắp xếp trong việc truy vấn tất cả các thực thể.

❑ **Sort**


- ❖ `Sort.by(Direction, String...properties)`: Tạo Sort
- ❖ `Sort.by(String...properties)`: tạo Sort tăng dần

❑ Ví dụ:

```
@ResponseBody
@RequestMapping("/product/list")
public List<Product> list() {
    Sort sort = Sort.by(Direction.DESC, "unitPrice");
    return dao.findAll(sort);
}
```

- ❑ Hiển thị sản phẩm theo cấu trúc giao diện như sau.
- ❑ Thực hiện sắp xếp tăng dần hoặc giảm dần khi click chuột vào ô tiêu đề của mỗi cột

Click để sắp xếp tăng/giảm theo cột



Name	Price	Date	Category

❑ **Page<Entity> findAll(Pageable)**

- ❖ Cho phép thực hiện truy vấn có sắp xếp và phân trang. Trong đó Pageable cung cấp tiêu chí sắp xếp và phân trang

❑ Pageable có thể tạo ra theo các cách sau

- ❖ PageRequest.of(int page, int size)
- ❖ PageRequest.of(int page, int size, Sort sort)
- ❖ PageRequest.of(int page, int size, Direction direction, String...properties)

❑ Page<Entity> chứa kết quả truy vấn gồm

- ❖ Trang dữ liệu List<Entity>
- ❖ Thông tin tổng hợp phân trang và các hoạt động điều hướng khác

❑ Dữ liệu trang

- ❖ getContent(): List<Entity>

❑ Thông tin phân trang

- ❖ getNumber(): int
 - ❑ Trang số
- ❖ getSize(): int
 - ❑ Kích thước trang
- ❖ getTotalPages(): int
 - ❑ Tổng số trang
- ❖ getTotalElements(): int
 - ❑ Tổng số thực thể
- ❖ getNumberOfElements(): int
 - ❑ Số thực thể hiện có

❑ Điều hướng

- ❖ nextPageable(): Pageable
- ❖ nextOrLastPageable(): Pageable
- ❖ previousPageable(): Pageable
- ❖ previousOrFirstPageable(): Pageable

❑ Kiểm tra

- ❖ hasNext(): boolean
- ❖ hasPrevious() : boolean
- ❖ isFirst() : boolean
- ❖ isLast() : boolean

```
@Autowired
CategoryDAO dao;

@ResponseBody
@RequestMapping("/pager/list")
public Page<Category> list(
    @RequestParam("pageNo") Optional<Integer> pageNo) {
    Pageable pageable = PageRequest.of(pageNo.orElse(0), 3);
    Page<Category> page = dao.findAll(pageable);
    return page;
}
```

❑ <http://localhost:8080/pager/list>

❑ <http://localhost:8080/pager/list?pageNo=2>

```
{
  "content": [...],
  "pageable": {
    "offset": 6,
    "pageNumber": 2,
    "pageSize": 3,
  },
  "totalElements": 8,
  "last": true,
  "totalPages": 3,
  "size": 3,
  "number": 2,
  "numberOfElements": 2,
  "first": false
}
```

```
"content": [
  {
    "id": "1006",
    "name": "Nón thời trang"
  },
  {
    "id": "1007",
    "name": "Túi xách du lịch"
  }
],
```

- **Content:** dữ liệu truy vấn được
- **Pageable:** thông tin phân trang đầu vào
- Các thuộc tính còn lại là các thông số được tính từ kết quả của việc phân trang

❑ Duyệt dữ liệu

```
<c:forEach var="item" items="${page.content}">  
    //....  
</c:forEach>
```

❑ Hiển thị kết quả phân trang

- ❖ Trang số: `${page.number}`
- ❖ Kích thước trang: `${page.size}`
- ❖ Tổng số trang: `${page.totalPages}`
- ❖ Số phần tử hiện tại: `${page.numberOfElements}`
- ❖ Tổng số phần tử: `${page.totalElements}`

□ Tìm giải pháp thực hiện phân trang sản phẩm như hình sau

Name	Price	Date	Category

- Trang: ?
- Tổng số trang: ?
- Số thực thể: ?
- Tổng số thực thể: ?



```
<a href="/pager/list?page=0"> |&lt; </a>
<a href="/pager/list?page=${page.number-1}"> &lt;&lt; </a>
<a href="/pager/list?page=${page.number+1}"> &gt;&gt; </a>
<a href="/pager/list?page=${page.totalPages-1}"> &gt;| </a>
```

- JpaRepository API
 - Giới thiệu CSDL mẫu J5Shop
 - Tạo các lớp thực thể
 - Sơ đồ phân cấp thừa kế JpaRepository
 - Tạo các DAO làm việc với J5Shop
 - Xây dựng ứng dụng CRUD
- Sắp xếp và phân trang với JpaRepository
 - Truy vấn có sắp xếp
 - Truy vấn có phân trang
 - Sắp xếp và phân trang sản phẩm





FPT Education

FPT POLYTECHNIC

Thank you