

1



## **JPARepository API - @QUERY**

**GIẢNG VIÊN:**

## 📖 TRUY VẤN VỚI JPQL, SQL, NAMEDQUERY

📖 @QUERY

📖 TÙY BIẾN SẮP XẾP VÀ PHÂN TRANG

📖 XÂY DỰNG TRANG TÌM KIẾM SẢN PHẨM

📖 XÂY DỰNG TRANG BÁO CÁO – THỐNG KÊ

## 📖 TRUY VẤN VỚI DSL (DOMAIN SPECIFIC LANGUAGE)

📖 GIỚI THIỆU

📖 TRUY VẤN VỚI DSL

📖 TÌM HIỂU SÂU HƠN VỀ CÚ PHÁP VÀ TỪ KHÓA DSL





@QUERY(VALUE, NAME, NATIVEQUERY)

---

- ❑ JpaRepository API cung cấp **@Query** giúp viết mã truy vấn cho các phương thức khai báo trong interface JpaRepository
  - ❑ Spring sẽ tự động sinh mã thực thi cho phương thức có **@Query()** để thực hiện câu lệnh JPQL
  - ❑ Cú pháp:
    - ❖ **@Query**(*value*, *name*, *nativeQuery*)
    - ❖ Trong đó:
      - Value: câu lệnh JPQL hoặc SQL (nếu nativeQuery=true)
      - Name: tên của @NamedQuery đã được khai báo entity
- Chú ý: value và name không thể xuất hiện đồng thời*

```
public interface ProductDAO extends JpaRepository<Product, Integer> {  
    @Query("SELECT p FROM Product p WHERE p.category.id='1005'")  
    List<Product> findByCategoryId();  
}
```

```
@Autowired  
ProductDAO dao;  
  
@ResponseBody  
@RequestMapping("/account/list")  
public List<Product> list() {  
    List<Product> list = dao.findByCategoryId();  
    return list;  
}
```

- ❑ @Query() được sử dụng để xây dựng câu lệnh JPQL
- ❑ Căn cứ vào JPQL để khai báo cú pháp (*return type và arguments*) cho phương thức

## @QUERY() – JPQL CÓ CHỨA THAM SỐ

```
public interface ProductDAO extends JpaRepository<Product, Integer> {  
    @Query("SELECT p FROM Product p WHERE p.category.id=?1")  
    List<Product> findByCategoryId(String categoryId);  
}
```

```
@Autowired  
ProductDAO dao;  
  
@ResponseBody  
@RequestMapping("/account/list")  
public List<Product> list() {  
    List<Product> list = dao.findByCategoryId("1005");  
    return list;  
}
```

- ❑ ?1, ?2... chỉ ra thứ tự đối số của phương thức được truyền vào
- ❑ Thứ tự của đối số đầu tiên là ?1

## @QUERY() – JPQL CÓ CHỨA THAM SỐ

```
public interface ProductDAO extends JpaRepository<Product, Integer> {  
    @Query("SELECT p FROM Product p WHERE p.category.id=:cid")  
    List<Product> findByCategoryId(@Param("cid") String categoryId);  
}
```

```
@Autowired  
ProductDAO dao;  
  
@ResponseBody  
@RequestMapping("/account/list")  
public List<Product> list() {  
    List<Product> list = dao.findByCategoryId("1005");  
    return list;  
}
```

- ❑ Tên tham số bắt đầu bởi dấu :
- ❑ @Param() xác định tên tham số mà đối số của phương thức được truyền vào

- ❑ SELECT p FROM Product p WHERE p.name LIKE ?1
  - ❖ List<Product> findByKeyword(String keyword)
- ❑ SELECT p FROM Product p WHERE p.name LIKE :keyword
  - ❖ List<Product> findByKeyword(@Param("keyword") String keyword)
- ❑ SELECT p FROM Product p WHERE p.price BETWEEN ?1 AND ?2
  - ❖ List<Product> findByPrice(double min, double max)
- ❑ SELECT p FROM Product p WHERE p.price BETWEEN :min AND :max
  - ❖ List<Product> findByPrice(@Param("min")double min, @Param("max")double max)
- ❑ SELECT p.name FROM Product p WHERE p.price BETWEEN ?1 AND ?2
  - ❖ List<String> findNamesByPrice(double min, double max)
- ❑ SELECT min(p) FROM Product p WHERE p.price BETWEEN ?1 AND ?2
  - ❖ double findMin(double min, double max)



- ❑ @Query(value = "SELECT \* FROM Products WHERE Name LIKE ?1",  
nativeQuery = true)
  - ❖ List<Product> findByKeyword(String keyword);
  
- ❑ @Query(value = "SELECT \* FROM Products WHERE Price BETWEEN  
?1 AND ? 2", nativeQuery = true)
  - ❖ List<Product> findByPrice(double min, double max);
  
- ❑ @Query(value = "SELECT COUNT(\*) FROM Products WHERE Price  
BETWEEN ?1 AND ? 2", nativeQuery = true)
  - ❖ Long countByPrice(double min, double max);

```
@NamedQuery(  
    name="findByKeyword",  
    query="SELECT p FROM Product p WHERE p.name LIKE ?1"  
)  
@Data  
@Entity  
@Table(name = "Products")  
public class Product implements Serializable{...}
```

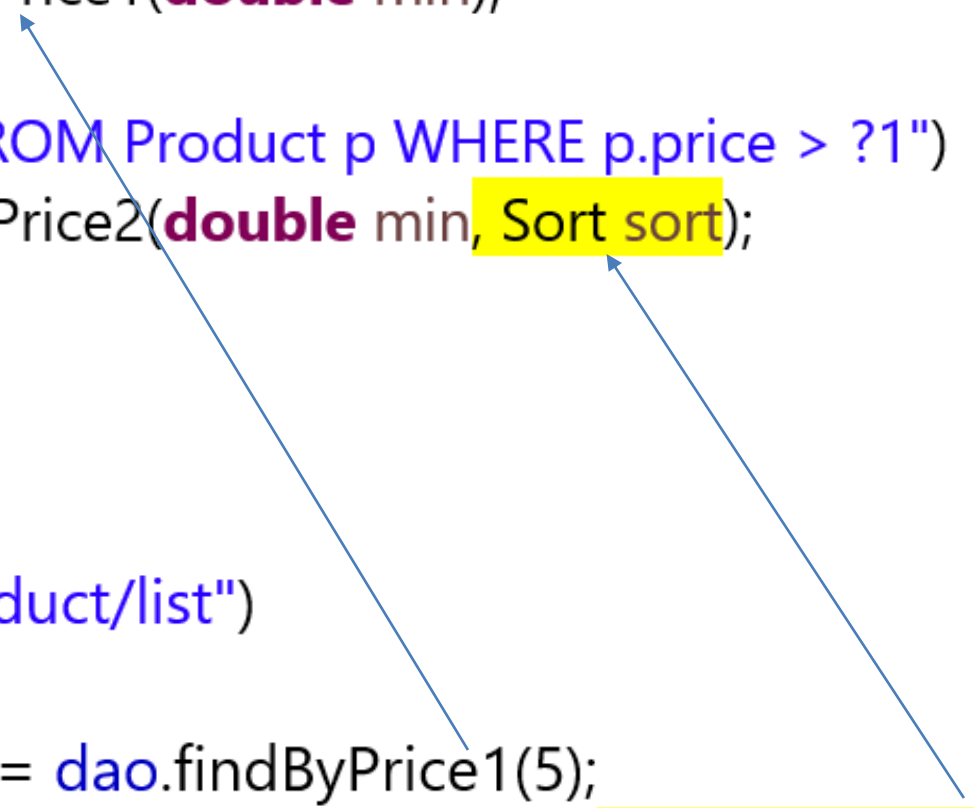
```
public interface ProductDAO extends JpaRepository<Product, Integer>{  
    @Query(name = "findByKeyword")  
    List<Product> findByKeyword(String keyword);  
}
```



# SẮP XẾP VÀ PHÂN TRẠNG

---

```
public interface ProductDAO extends JpaRepository<Product, Integer>{  
    @Query("SELECT p FROM Product p WHERE p.price > ?1 ORDER BY p.price DESC")  
    List<Product> findByPrice1(double min);  
  
    @Query("SELECT p FROM Product p WHERE p.price > ?1")  
    List<Product> findByPrice2(double min, Sort sort);  
}  
  
@RequestMapping("/product/list")  
public void list() {  
    List<Product> list1 = dao.findByPrice1(5);  
    List<Product> list2 = dao.findByPrice2(5, Sort.by(Direction.DESC, "price"));  
}
```

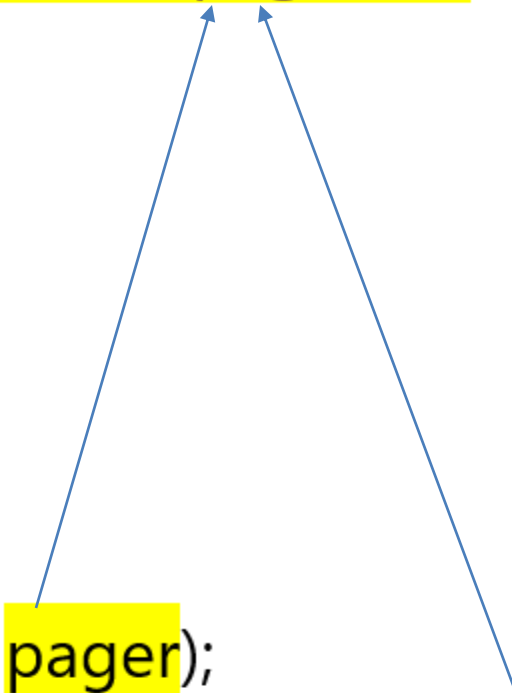


The diagram consists of two blue arrows. One arrow originates from the `dao.findByPrice1(5)` call in the `list()` method and points to the `findByPrice1` method signature in the `ProductDAO` interface. The second arrow originates from the `Sort.by(Direction.DESC, "price")` call in the `list()` method and points to the `Sort sort` parameter in the `findByPrice2` method signature.

```
public interface ProductDAO extends JpaRepository<Product, Integer>{  
    @Query("SELECT p FROM Product p WHERE p.price > ?1")  
    List<Product> findByPrice (double min, Pageable pageable);  
}
```

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

```
public void list() {  
    Pageable pager = PageRequest.of(0, 5);  
    Page<Product> page1 = dao.findByPrice(2, pager);  
    Page<Product> page2 = dao.findByPrice(2, pager.nextOrLastPageable());  
}
```





# TRUY VẤN TỔNG HỢP

---

- ❑ Hãy tổng hợp số liệu bán hàng từ OrderDetail để cung cấp các thông tin có cấu trúc như sau:

LOẠI HÀNG	DOANH THU	SỐ LƯỢNG BÁN

- ❑ Chúng ta cần câu lệnh JPQL như sau

```
SELECT d.product.category, sum(d.price * d.quantity), sum(d.quantity)
FROM OrderDetail d
GROUP BY d.product.category
```

- ❑ Với câu lệnh truy vấn này thì kết quả sẽ là **List<Object[]>**, mỗi phần tử là một mảng đối tượng **[Category, Double, Long]**

@Data

@AllArgsConstructor

@NoArgsConstructor

@Entity

**public class** Report **implements** Serializable{

@Id

**private** Category loại;

**private double** doanhThu;

**private long** soLuong;

}

❑ Để đơn giản hóa trong công việc lập trình xử lý, chúng ta mong muốn nhận được List<Report>. Trong đó Report là một Entity chứa 3 thuộc tính

❖ loại

❖ doanh thu

❖ số lượng bán



- Câu lệnh JPQL cho kết quả List<Report> như sau

SELECT

**new Report**(d.product.category, sum(d.price \* d.quantity), sum(d.quantity))

FROM OrderDetail d

GROUP BY d.product.category

- Phương thức truy vấn dữ liệu tổng hợp với JpaRepository

```
public interface OrderDetailDAO extends JpaRepository<OrderDetail, Long>{  
    @Query("SELECT new Report(d.product.category, sum(d.price*d.quantity), "  
        + " sum(d.quantity)) FROM OrderDetail d GROUP BY d.product.category")  
    List<Report> revenueByCategory();  
}
```

2



## **JPARepository API - DSL**

**GIẢNG VIÊN:**

- ❑ Ngoài @Query(), JpaRepository còn đơn giản hóa việc truy vấn dữ liệu bằng cách khai báo tên phương thức đúng cú pháp quy định mà không cần phải viết câu lệnh JPQL.
- ❑ JpaRepository dựa vào tên của phương thức truy vấn để sinh ra JPQL và mã thực thi truy vấn.
- ❑ Cú pháp đặt tên phương thức truy vấn nói trên được gọi là DSL.
- ❑ Ví dụ:
  - ❖ Với @Query  
@Query("SELECT p FROM Product p WHERE p.name LIKE ?1")  
List<Product> searchProduct(String name)
  - ❖ DSL  
List<Product> **findByNameLike**(String name)

## LIST<PRODUCT> FINDBYNAMELIKE(STRING NAME)

- ❑ JpaRepository xác định miền dữ liệu dựa vào T trong biểu thức tổng quát kế thừa của interface JpaRepository<T, ID>
- ❑ **findByNameLike**(x)
  - ❖ **findBy\_**, **getBy\_**, **countBy\_** là các tiền tố truy vấn được chấp nhận trong JpaRepository
  - ❖ **Name** là tên thuộc tính của miền dữ liệu
  - ❖ **Like** là toán tử áp dụng cho thuộc tính đã được chỉ ra kế trước
- ❑ Ví dụ
  - ❖ findBy**PriceIsNull**() ~ findBy**PriceNull**()
  - ❖ findBy**PriceEqual**(double max) ~ findBy**Price**(double max)
  - ❖ findBy**PriceLessThan**(double max)
  - ❖ findBy**PriceBetween**(double min, double max)
  - ❖ findBy**PriceLessThanEqualAndNameLike**(double max, String name)

TỪ KHÓA	VÍ DỤ	JPQL TƯƠNG ĐƯƠNG
<b>AND</b>	<i>findBy</i> Lastname <b>And</b> Firstname	... <i>WHERE</i> x.lastname = ?1 <b>AND</b> x.firstname = ?2
<b>OR</b>	<i>findBy</i> Lastname <b>Or</b> Firstname	... <i>WHERE</i> x.lastname = ?1 <b>OR</b> x.firstname = ?2
<b>IS, EQUALS</b>	<i>findBy</i> Firstname <b>Equals</b>	... <i>WHERE</i> x.firstname = ?1
<b>BETWEEN</b>	<i>findBy</i> StartDate <b>Between</b>	... <i>WHERE</i> x.startDate <b>BETWEEN</b> ?1 and ?
<b>LESSTHAN</b>	<i>findBy</i> Age <b>LessThan</b>	... <i>WHERE</i> x.age < ?1
<b>LESSTHANEQUAL</b>	<i>findBy</i> Age <b>LessThanEqual</b>	... <i>WHERE</i> x.age <= ?1
<b>GREATERTHAN</b>	<i>findBy</i> Age <b>GreaterThan</b>	... <i>WHERE</i> x.age > ?1
<b>GREATERTHANEQUAL</b>	<i>findBy</i> Age <b>GreaterThanEqual</b>	... <i>WHERE</i> x.age >= ?1
<b>AFTER</b>	<i>findBy</i> StartDate <b>After</b>	... <i>WHERE</i> x.startDate > ?1
<b>BEFORE</b>	<i>findBy</i> StartDate <b>Before</b>	... <i>WHERE</i> x.startDate < ?1
<b>ISNULL</b>	<i>findBy</i> Age <b>IsNull</b>	... <i>WHERE</i> x.age <b>IS NULL</b>
<b>ISNOTNULL, NOTNULL</b>	<i>findBy</i> Age <b>[Is]NotNull</b>	... <i>WHERE</i> x.age <b>IS NOT NULL</b>

TỪ KHÓA	VÍ DỤ	JPQL TƯƠNG ĐƯƠNG
<b>LIKE</b>	<i>findBy</i> Firstname <b>Like</b>	... <b>WHERE</b> x.firstname <b>LIKE</b> ?1
<b>NOT LIKE</b>	<i>findBy</i> Firstname <b>NotLike</b>	... <b>WHERE</b> x.firstname <b>NOT LIKE</b> ?1
<b>STARTING WITH</b>	<i>findBy</i> Firstname <b>StartingWith</b>	... <b>WHERE</b> x.firstname <b>LIKE</b> ?1[%]
<b>ENDING WITH</b>	<i>findBy</i> Firstname <b>EndingWith</b>	... <b>WHERE</b> x.firstname <b>LIKE</b> [%]?1
<b>CONTAINING</b>	<i>findBy</i> Firstname <b>Containing</b>	... <b>WHERE</b> x.firstname <b>LIKE</b> [%]?1[%]
<b>ORDER BY</b>	<i>findBy</i> Age <b>OrderBy</b> Lastname <b>Desc</b>	... <b>WHERE</b> x.age = ?1 <b>ORDER BY</b> x.lastname <b>DESC</b>
<b>NOT</b>	<i>findBy</i> Lastname <b>Not</b>	... <b>WHERE</b> x.lastname <b>&lt;&gt;</b> ?1
<b>IN</b>	<i>findBy</i> Age <b>In</b> (Collection ages)	... <b>WHERE</b> x.age <b>IN</b> ?1
<b>NOT IN</b>	<i>findBy</i> Age <b>NotIn</b> (Collection ages)	... <b>WHERE</b> x.age <b>NOT IN</b> ?1
<b>TRUE</b>	<i>findBy</i> Active <b>True</b> ()	... <b>WHERE</b> x.active = <b>true</b>
<b>FALSE</b>	<i>findBy</i> Active <b>False</b> ()	... <b>WHERE</b> x.active = <b>false</b>
<b>IGNORE CASE</b>	<i>findBy</i> Firstname <b>IgnoreCase</b>	... <b>WHERE</b> <b>upper</b> (x.firstname) = <b>upper</b> (?1)

- ✓ Lambda Expression
- ✓ Stream API
  - ✓ Filter()
  - ✓ Map()
  - ✓ Reduce()
  - ✓ allMatch()/anyMatch()/noneMatch()
- ✓ JSON
- ✓ Jackson API
  - ✓ JsonNode
  - ✓ Jackson with Map
  - ✓ Jackson with Plain Object







**FPT** Education

FPT POLYTECHNIC

**Thank you**