## 📌 Example: Spring Data JPA Query Generation

### Repository

@Repository

public interface ProductRepo extends JpaRepository<Product, Integer> {

// 1. Find by name (exact match)

List<Product> findByName(String name);

// 2. Find by name containing substring (LIKE %name%)

List<Product> findByNameContaining(String keyword);

// 3. Find by price greater than

List<Product> findByPriceGreaterThan(BigDecimal price);

// 4. Find by price between two values

List<Product> findByPriceBetween(BigDecimal min, BigDecimal max);

// 5. Find by name and price

List<Product> findByNameAndPrice(String name, BigDecimal price);

// 6. Find top 5 cheapest products

List<Product> findTop5ByOrderByPriceAsc();

// 7. Find all products ordered by price descending

List<Product> findAllByOrderByPriceDesc();

}

### Service Example

@Service

public class ProductService {

@Autowired

private ProductRepo productRepo;

public void demoQueries() {

// Save example product

productRepo.save(new Product("Laptop", new BigDecimal("50000")));

// Query examples

System.out.println(productRepo.findByName("Laptop"));

System.out.println(productRepo.findByNameContaining("top"));

System.out.println(productRepo.findByPriceGreaterThan(new BigDecimal("1000")));

System.out.println(productRepo.findByPriceBetween(new BigDecimal("1000"), new BigDecimal("60000")));

System.out.println(productRepo.findByNameAndPrice("Laptop", new BigDecimal("50000")));

System.out.println(productRepo.findTop5ByOrderByPriceAsc());

}

}

### Generated SQL (behind the scenes)

For example,

productRepo.findByPriceBetween(new BigDecimal("1000"), new BigDecimal("60000"));

will generate something like:

select \* from product

where price between 1000 and 60000;

✅ With this, you don’t need to write queries manually — Spring Data generates them by parsing method names.

Do you want me to also show you an example of **custom JPQL query with** @Query **annotation** (in case method names become too complex)?

##### You said:

give me some example of @Query annotation with above example

###### ChatGPT said:

Great 👍 let’s extend your ProductRepo with **custom queries** using @Query.

Spring Data JPA lets you define queries in **JPQL** (object-oriented, works on entities) or **native SQL**.

## 📌 Examples with @Query

@Repository

public interface ProductRepo extends JpaRepository<Product, Integer> {

// 1. JPQL: Find product by exact name

@Query("SELECT p FROM Product p WHERE p.name = :name")

List<Product> findByNameJPQL(@Param("name") String name);

// 2. JPQL: Find products with price greater than

@Query("SELECT p FROM Product p WHERE p.price > :price")

List<Product> findExpensiveProducts(@Param("price") BigDecimal price);

// 3. JPQL: Find products in a price range

@Query("SELECT p FROM Product p WHERE p.price BETWEEN :min AND :max")

List<Product> findProductsInRange(@Param("min") BigDecimal min, @Param("max") BigDecimal max);

// 4. JPQL: Search by name containing (LIKE)

@Query("SELECT p FROM Product p WHERE p.name LIKE %:keyword%")

List<Product> searchByName(@Param("keyword") String keyword);

// 5. JPQL: Sort by price descending

@Query("SELECT p FROM Product p ORDER BY p.price DESC")

List<Product> findAllOrderByPriceDesc();

// 6. Native SQL: Top N cheapest products

@Query(value = "SELECT \* FROM product ORDER BY price ASC LIMIT :count", nativeQuery = true)

List<Product> findTopNByPrice(@Param("count") int count);

}

## 📌 Usage Example (Service Layer)

@Service

public class ProductService {

@Autowired

private ProductRepo productRepo;

public void demoQueries() {

productRepo.save(new Product("Mobile", new BigDecimal("15000")));

productRepo.save(new Product("Laptop", new BigDecimal("60000")));

productRepo.save(new Product("Headphones", new BigDecimal("2000")));

System.out.println(productRepo.findByNameJPQL("Laptop"));

System.out.println(productRepo.findExpensiveProducts(new BigDecimal("10000")));

System.out.println(productRepo.findProductsInRange(new BigDecimal("2000"), new BigDecimal("20000")));

System.out.println(productRepo.searchByName("top"));

System.out.println(productRepo.findAllOrderByPriceDesc());

System.out.println(productRepo.findTopNByPrice(2));

}

}

## 🔑 Key Points

* @Query with **JPQL** → works with entity field names (Product, name, price)
* @Query(nativeQuery = true) → works with actual DB table/column names (product, price)
* Use @Param("...") to bind method parameters to query parameters