Implementing Query Methods in Spring Data JPA

Spring Data JPA provides a powerful feature called "query methods" that allows to create database queries simply by defining method signatures in your repository interfaces.

1. Basic Setup
First, defining a simple entity and repository:

```
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private Double price;
    private Integer stock;
    private Boolean active;
    private LocalDate createdAt;

    // Constructors, getters, setters
}
```

2. Repository Interface with Query Methods

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    // 1. Basic query derivation
    List<Product> findByName(String name);

    // 2. Query with multiple conditions
    List<Product> findByNameAndPrice(String name, Double price);

    // 3. Query with comparison operators
    List<Product> findByPriceGreaterThan(Double price);
    List<Product> findByPriceBetween(Double minPrice, Double maxPrice);

    // 4. Query with sorting
    List<Product> findByNameContainingOrderByNameAsc(String namePart);

    // 5. Query with OR condition
    List<Product> findByNameOrPrice(String name, Double price);

    // 6. Query with negation
    List<Product> findByNameNot(String name);

    // 7. Query with NULL check
    List<Product> findByNameIsNull();
    List<Product> findByNameIsNotNull();

    // 8. Query with LIKE pattern
    List<Product> findByNameLike(String pattern);
    List<Product> findByNameStartingWith(String prefix);
    List<Product> findByNameEndingWith(String suffix);
    List<Product> findByNameContaining(String infix);

    // 9. Query with boolean condition
    List<Product> findByActiveTrue();
    List<Product> findByActiveFalse();
```

```
    // 10. Query with date comparison
    List<Product> findByCreatedAtAfter(LocalDate date);
    List<Product> findByCreatedAtBefore(LocalDate date);

    // 11. Limiting query results
    Product findFirstByOrderByPriceAsc();
    Product findTopByOrderByPriceDesc();
    List<Product> findTop3ByOrderByPriceDesc();

    // 12. Count queries
    long countByPriceGreaterThan(Double price);

    // 13. Exists queries
    boolean existsByName(String name);

    // 14. Delete queries
    void deleteByStockLessThan(Integer minStock);

    // 15. Custom sorting parameter
    List<Product> findByPriceLessThan(Double price, Sort sort);
}
```

3. Using the Query Methods in Service Layer

```
@Service
public class ProductService {

    private final ProductRepository productRepository;

    public ProductService(ProductRepository productRepository) {
        this.productRepository = productRepository;
    }

    public void demonstrateQueryMethods() {
        // 1. Basic query
        List<Product> productsByName =
productRepository.findByName("Laptop");

        // 2. Multiple conditions
        List<Product> productsByNameAndPrice =
productRepository.findByNameAndPrice("Laptop", 999.99);

        // 3. Comparison operators
        List<Product> expensiveProducts =
productRepository.findByPriceGreaterThan(500.0);
        List<Product> midRangeProducts =
productRepository.findByPriceBetween(300.0, 700.0);

        // 4. Sorting
        List<Product> sortedProducts =
productRepository.findByNameContainingOrderByNameAsc("top");

        // 5. OR condition
        List<Product> productsByNameOrPrice =
productRepository.findByNameOrPrice("Laptop", 499.99);

        // 6. Negation
```

```
        List<Product> productsNotNamedLaptop =
productRepository.findByNameNot("Laptop");

        // 7. NULL check
        List<Product> productsWithNullName =
productRepository.findByNameIsNull();

        // 8. LIKE patterns
        List<Product> productsLike =
productRepository.findByNameLike("%top%");
        List<Product> productsStartingWithLap =
productRepository.findByNameStartingWith("Lap");

        // 9. Boolean conditions
        List<Product> activeProducts =
productRepository.findByActiveTrue();

        // 10. Date comparison
        List<Product> recentProducts =
productRepository.findByCreatedAtAfter(LocalDate.now().minusMonths(1));

        // 11. Limiting results
        Product cheapestProduct =
productRepository.findFirstByOrderByPriceAsc();
        List<Product> top3MostExpensive =
productRepository.findTop3ByOrderByPriceDesc();

        // 12. Count
        long expensiveCount =
productRepository.countByPriceGreaterThan(1000.0);

        // 13. Exists
        boolean laptopExists = productRepository.existsByName("Laptop");

        // 14. Delete
        productRepository.deleteByStockLessThan(5);

        // 15. Custom sorting
        List<Product> cheapProducts =
productRepository.findByPriceLessThan(
            200.0,

Sort.by("price").descending().and(Sort.by("name").ascending())
        );
    }
}
```

4. Custom Queries with @Query
For more complex queries, you can use the @Query annotation:

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    // JPQL query
    @Query("SELECT p FROM Product p WHERE p.price > ?1 AND p.active =
true")
    List<Product> findActiveProductsMoreExpensiveThan(Double minPrice);

    // Native SQL query
```

```
    @Query(value = "SELECT * FROM products WHERE name LIKE %:keyword%",
nativeQuery = true)
    List<Product> searchByKeyword(@Param("keyword") String keyword);

    // Projection query
    @Query("SELECT p.name, p.price FROM Product p WHERE p.stock > 0")
    List<Object[]> findProductNamesAndPricesInStock();
}
```

5. Query by Example
Spring Data JPA also supports Query by Example:

```
public List<Product> findProductsByExample(Product exampleProduct) {
    ExampleMatcher matcher = ExampleMatcher.matching()
        .withIgnoreCase()
        .withStringMatcher(StringMatcher.CONTAINING)
        .withIgnorePaths("price", "createdAt");

    Example<Product> example = Example.of(exampleProduct, matcher);
    return productRepository.findAll(example);
}
```