**Objectives**

Demonstrate implementation of Query Methods feature of Spring Data JPA

* Query Methods - Search by containing text, sorting, filter with starting text, fetch between dates, greater than or lesser than, top

Query methods - https://docs.spring.io/spring-data/jpa/docs/2.2.0.RELEASE/reference/html/#jpa.query-methods.query-creation

In Spring Data JPA, Query Methods allow us to query the database by simply defining method names in the repository interface. Spring will automatically generate the SQL queries based on the method name.This removes the need to manually write JPQL or SQL queries.

Key Features:

* No JPQL required
* Highly readable and easy to maintain

Examples:

1. **Search by Text (LIKE %text%)**

// Fetch countries containing 'ou' in their names

List<Country> findByNameContainingIgnoreCase(String text);

1. **Sorted Search (Order By)**

// Fetch countries containing 'ou' and sort them ascending by name

List<Country> findByNameContainingIgnoreCaseOrderByNameAsc(String text);

1. **Search by Starting Alphabet**

// Fetch countries starting with 'Z'

List<Country> findByNameStartingWithIgnoreCase(String text);

1. **Fetch Records Between Dates**

// Fetch Facebook stocks in September 2019

List<Stock> findByCodeAndDateBetween(String code, Date startDate, Date endDate);

1. **Greater Than Comparison**

// Fetch Google stocks with closing price greater than 1250

List<Stock> findByCodeAndCloseGreaterThan(String code, double price);

1. **Top N Records**

// Fetch top 3 stocks by trading volume

List<Stock> findTop3ByOrderByVolumeDesc();

Demonstrate implementation of O/R Mapping

* @ManyToOne, @JoinColumn, @OneToMany, FetchType.EAGER, FetchType.LAZY, @ManyToMany, @JoinTable, mappedBy

Relationships reference - <https://www.baeldung.com/spring-data-rest-relationships>

O/R Mapping connects Java objects to relational database tables.  
Hibernate (via JPA annotations) manages this connection using relationships and mappings.

**Relationship Types in Spring Data JPA:**

**a) Many-to-One (Many Employees → One Department)**

When many employees belong to a single department.

**Example:**

@Entity

public class Employee {

@ManyToOne

@JoinColumn(name = "em\_dp\_id") // Foreign key in Employee table

private Department department;

}

**b) One-to-Many (One Department → Many Employees)**

When one department has multiple employees.

**Example:**

@Entity

public class Department {

@OneToMany(mappedBy = "department", fetch = FetchType.EAGER)

private Set<Employee> employeeList;

}

Explanation:

The 'mappedBy' attribute indicates that the 'department' field in the Employee entity is the owner of the relationship.

FetchType.EAGER means that all employees of the department will be loaded immediately when the department is fetched.

**c) Many-to-Many (Many Employees ↔ Many Skills)**

When many employees can have many skills.

**Example (Employee side):**

@Entity

public class Employee {

@ManyToMany(fetch = FetchType.EAGER)

@JoinTable(name = "employee\_skill",

joinColumns = @JoinColumn(name = "es\_em\_id"),

inverseJoinColumns = @JoinColumn(name = "es\_sk\_id"))

private Set<Skill> skillList;

}

**Example (Skill side):**

@Entity

public class Skill {

@ManyToMany(mappedBy = "skillList")

private Set<Employee> employeeList;

}

Explanation:

The @JoinTable creates a join table called 'employee\_skill' to map the many-to-many relationship.

On the Skill side, 'mappedBy = "skillList"' indicates that the ownership of the relationship is in the Employee entity.

FetchType.EAGER is used to immediately load the skills when fetching an employee.

**d) FetchType.EAGER Example**

When related entities must be loaded immediately.

**Example:**

@OneToMany(mappedBy = "department", fetch = FetchType.EAGER)

private Set<Employee> employeeList;

Explanation:

When you load a department, all employees will be fetched immediately in the same query.

**e) FetchType.LAZY Example**

When related entities should be loaded only when accessed.

**Example:**

@OneToMany(mappedBy = "department", fetch = FetchType.LAZY)

private Set<Employee> employeeList;

Explanation:

Employees will be loaded only when you call 'getEmployeeList()'.

This improves performance when you don’t always need the related data.