**Spring Data JPA Hands-on Answers**

**1. Query Methods on Country Table**

**a) Search Countries by Containing Text**

* **Method Name:**  
  List<Country> findByNameContaining(String text);
* **Usage:**  
  Returns all countries whose names contain the given text (case-insensitive).

**b) Search Countries by Containing Text in Ascending Order**

* **Method Name:**  
  List<Country> findByNameContainingOrderByNameAsc(String text);
* **Usage:**  
  Returns all matching countries, sorted by name in ascending order.

**c) Search Countries by Starting Alphabet**

* **Method Name:**  
  List<Country> findByNameStartingWith(String alphabet);
* **Usage:**  
  Returns all countries whose names start with the specified alphabet.

**2. Query Methods on Stock Table**

**a) Get All Facebook Stock Details for September 2019**

* **Method Name:**  
  List<Stock> findByCodeAndDateBetween(String code, LocalDate start, LocalDate end);
* **Usage:**  
  findByCodeAndDateBetween("FB", LocalDate.of(2019, 9, 1), LocalDate.of(2019, 9, 30));
* **Returns:**  
  All Facebook stocks in September 2019.

**b) Get All Google Stock Details Where Price > 1250**

* **Method Name:**  
  List<Stock> findByCodeAndCloseGreaterThan(String code, double price);
* **Usage:**  
  findByCodeAndCloseGreaterThan("GOOGL", 1250);
* **Returns:**  
  All Google stock records with closing price greater than 1250.

**c) Top 3 Dates with Highest Transaction Volume**

* **Method Name:**  
  List<Stock> findTop3ByOrderByVolumeDesc();
* **Usage:**  
  Returns the top 3 stock records with the highest volume.

**d) Three Dates When Netflix Stocks Were the Lowest**

* **Method Name:**  
  List<Stock> findTop3ByCodeOrderByCloseAsc(String code);
* **Usage:**  
  findTop3ByCodeOrderByCloseAsc("NFLX");
* **Returns:**  
  The three lowest closing prices for Netflix.

**3. O/R Mapping and Relationships**

**a) Employee, Department, Skill Entity Mapping**

* **Annotations:**
  + @Entity, @Table for each class.
  + @Id, @GeneratedValue(strategy = GenerationType.IDENTITY) for primary keys.
  + @Column for each field.

**Employee Fields**

* private int id;
* private String name;
* private double salary;
* private boolean permanent;
* private Date dateOfBirth;

**Department Fields**

* private int id;
* private String name;

**Skill Fields**

* private int id;
* private String name;

**4. Many-to-One Relationship: Employee and Department**

* **Employee Entity:**

@ManyToOne  
@JoinColumn(name = "em\_dp\_id")  
private Department department;

* **Repository and Service:**
  + Use EmployeeService and DepartmentService for CRUD operations.
* **Fetching Employee with Department:**  
  Fetching an employee also retrieves the associated department due to the default EAGER fetch type for @ManyToOne[[1]](#fn1)[[2]](#fn2).

**5. One-to-Many Relationship: Department and Employees**

* **Department Entity:**

@OneToMany(mappedBy = "department", fetch = FetchType.EAGER)  
private Set<Employee> employeeList;

* **Usage:**  
  Fetching a department retrieves all associated employees. Default fetch is LAZY, but can be set to EAGER as shown[[3]](#fn3)[[4]](#fn4).

**6. Many-to-Many Relationship: Employee and Skill**

* **Employee Entity:**

@ManyToMany  
@JoinTable(  
 name = "employee\_skill",  
 joinColumns = @JoinColumn(name = "es\_em\_id"),  
 inverseJoinColumns = @JoinColumn(name = "es\_sk\_id")  
)  
private Set<Skill> skillList;

* **Skill Entity:**

@ManyToMany(mappedBy = "skillList")  
private Set<Employee> employeeList;

* **Usage:**  
  Allows each employee to have multiple skills and each skill to belong to multiple employees. Use EAGER fetch for immediate loading if needed[[5]](#fn5)[[6]](#fn6)[[7]](#fn7).

**7. Adding and Updating Employee, Department, Skill**

* **Add Employee:**
  + Create new Employee object.
  + Set properties and associate with a Department.
  + Save using employeeService.save(employee);
* **Update Employee:**
  + Retrieve existing employee.
  + Change department or other fields.
  + Save changes.

**8. Adding Skill to Employee**

* **Steps:**
  + Retrieve employee and skill by ID.
  + Add skill to employee's skill list.
  + Save employee to persist the relationship.

**9. Fetching Employee with Department and Skills**

* **In Service Method:**
  + Fetch employee by ID.
  + Access employee.getDepartment() and employee.getSkillList() to retrieve associated data.

All answers follow the concise, code-oriented format as previously provided.

⁂

1. <https://www.javaguides.net/2023/07/jpa-manytoone-annotation.html>
2. <https://www.geeksforgeeks.org/advance-java/jpa-many-to-one-mapping/>
3. <https://www.geeksforgeeks.org/advance-java/jpa-one-to-many-mapping/>
4. <https://www.bezkoder.com/jpa-one-to-many/>
5. <https://www.geeksforgeeks.org/jpa-many-to-many-mapping/>
6. <https://www.bezkoder.com/jpa-many-to-many/>
7. <https://www.baeldung.com/jpa-many-to-many>