**Spring Data JPA and Hibernate - Hands-on Structured Guide**

**Objectives**

**Query Methods in Spring Data JPA**

* Search by containing text
* Sorting
* Filter with starting text
* Fetch between dates
* Greater than or lesser than
* Top records

**Object-Relational Mapping**

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

**Hands-on 1: Country Table - Query Methods**

**Tasks:**

1. **Search by Partial Text**
   * Query Method: List<Country> findByNameContaining(String namePart);
2. **Search by Partial Text with Sorting**
   * Query Method: List<Country> findByNameContainingOrderByNameAsc(String namePart);
3. **Filter Countries by Starting Alphabet**
   * Query Method: List<Country> findByNameStartingWith(String prefix);

**Hands-on 2: Stock Table - Query Methods**

**Table Schema:**

**sql**

CREATE TABLE IF NOT EXISTS stock (

st\_id INT NOT NULL AUTO\_INCREMENT,

st\_code VARCHAR(10),

st\_date DATE,

st\_open NUMERIC(10,2),

st\_close NUMERIC(10,2),

st\_volume NUMERIC,

PRIMARY KEY (st\_id)

);

**Required Queries:**

1. **All Facebook Stocks in September 2019**
   * List<Stock> findByCodeAndDateBetween(String code, LocalDate startDate, LocalDate endDate);
2. **Google Stocks Where Price > 1250**
   * List<Stock> findByCodeAndCloseGreaterThan(String code, BigDecimal price);
3. **Top 3 Dates with Highest Volume**
   * List<Stock> findTop3ByOrderByVolumeDesc();
4. **Three Lowest Netflix Closing Prices**
   * List<Stock> findTop3ByCodeOrderByCloseAsc(String code);

**Hands-on 3: Payroll Tables and Bean Mapping**

**Entities:**

1. **Employee**
   * Fields: id, name, salary, permanent, dateOfBirth
2. **Department**
   * Fields: id, name
3. **Skill**
   * Fields: id, name

**Common Annotations:**

* @Entity
* @Table
* @Id
* @GeneratedValue(strategy = GenerationType.IDENTITY)
* @Column

**Repositories:**

* EmployeeRepository
* DepartmentRepository
* SkillRepository

**Hands-on 4: Many-to-One Mapping (Employee to Department)**

**Employee.java**

**java**

@ManyToOne

@JoinColumn(name = "em\_dp\_id")

private Department department;

**OrmLearnApplication.java**

* Create testGetEmployee() to fetch and log employee and department.
* Create testAddEmployee() to create and persist a new employee with department.
* Create testUpdateEmployee() to update employee’s department.

**Hands-on 5: One-to-Many Mapping (Department to Employees)**

**Department.java**

**java**

@OneToMany(mappedBy = "department")

private Set<Employee> employeeList;

* Initially fetch will fail (LazyInitializationException).
* Modify to:

**java**

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

**OrmLearnApplication.java**

* testGetDepartment() to log department and its employee list.

**Hands-on 6: Many-to-Many Mapping (Employee and Skill)**

**Employee.java**

**java**

@ManyToMany(fetch = FetchType.EAGER)

@JoinTable(name = "employee\_skill",

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

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

private Set<Skill> skillList;

**Skill.java**

**java**

@ManyToMany(mappedBy = "skillList")

private Set<Employee> employeeList;

**OrmLearnApplication.java**

* Modify testGetEmployee() to include skill list logging.
* Add testAddSkillToEmployee() to link skill to an employee and persist.