## Spring Data JPA Additional Hands-On Implementation

**Demonstrate implementation of Query Methods feature of Spring Data JPA:**

> Search by containing text

> Sorting

> Filter with starting text

> Fetch between dates

> Greater than or lesser than

> Top N records

· Demonstrate implementation of Object-Relational Mapping (ORM):

>@ManyToOne, @JoinColumn, @OneToMany, FetchType.EAGER, FetchType.LAZY

@ManyToMany, @JoinTable, mappedBy

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

**Search by Containing Text**

List<Country> findByNameContaining(String name);

Example search: "ou"

Output:

BV Bouvet Island

DJ Djibouti

TF French Southern Territories

GP Guadeloupe

LU Luxembourg

ZA South Africa

GS South Georgia and the South Sandwich Islands

SS South Sudan

UM United States Minor Outlying Islands

**Sort by Ascending Name**

List<Country> findByNameContainingOrderByNameAsc(String name);

**Filter by Starting Letter**

List<Country> findByNameStartingWith(String prefix);

Example: "Z"

Output:

ZM Zambia

ZW Zimbabwe

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

**Get Facebook stock data for September 2019**

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

Output (sample):

FB | 2019-09-03 | 184.00 | 182.39 | 9779400 FB | 2019-09-04 | 184.65 | 187.14 | 11308000 ... FB | 2019-09-27 | 180.49 | 177.10 | 14656200

**Get Google stock data with closing price > 1250**

List<Stock> findByCodeAndCloseGreaterThan(String code, BigDecimal close);

**Find Top 3 Dates with Highest Volume**

List<Stock> findTop3ByOrderByVolumeDesc();

Output:

FB | 2019-01-31 | 165.60 | 166.69 | 77233600 FB | 2018-10-31 | 155.00 | 151.79 | 60101300 FB | 2018-12-19 | 141.21 | 133.24 | 57404900

**Lowest 3 Netflix Prices**

List<Stock> findTop3ByCodeOrderByCloseAsc(String code);

Output:

NFLX | 2018-12-24 | 242.00 | 233.88 | 9547600 NFLX | 2018-12-21 | 263.83 | 246.39 | 21397600 NFLX | 2018-12-26 | 233.92 | 253.67 | 14402700

### Hands-on 3: ORM Mapping - Setup and Annotations

**Entities created:**

Employee

Department

Skill

Each entity uses:

@Entity@Table(name = "table\_name")@Id@GeneratedValue(strategy = GenerationType.IDENTITY)@Column(name = "column\_name")

**Fields in Employee:**

> id, name, salary, permanent, dateOfBirth

**Fields in Department:**

> id, name

**Fields in Skill:**

> id, name

### Object-Relational Mapping (ORM)

**@ManyToOne and @JoinColumn**  
Used in Employee.java

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

**@OneToMany with FetchType.EAGER**  
Used in Department.java

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

**@ManyToMany and @JoinTable**  
In Employee.java:

@ManyToMany@JoinTable(

name = "employee\_skill",

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

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

)private Set<Skill> skillList;

In Skill.java:

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

### Hands-on 4: Many-to-One Relationship - Employee and Department

**Service Implementation**

@Transactionalpublic Employee get(int id) {

return employeeRepository.findById(id).get();

}

@Transactionalpublic void save(Employee employee) {

employeeRepository.save(employee);

}

**Main Application Logs:**

Employee: Employee{id=1, name='John', salary=75000.0}

Department: Department{id=1, name='IT'}

**Add Employee to Department:**

Department dept = departmentService.get(1);Employee emp = new Employee();

emp.setName("Alice");

emp.setDepartment(dept);

employeeService.save(emp);

**Update Employee’s Department:**

Employee emp = employeeService.get(2);Department newDept = departmentService.get(3);

emp.setDepartment(newDept);

employeeService.save(emp);

**Hands-on 5: One-to-Many Relationship - Department to Employees**

**In Department.java:**

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

**Output Example:**

Department: HREmployees: [Alice, Bob, Clara]

### Hands-on 6: Many-to-Many Relationship - Employee and Skill

**In Employee.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;

**In Skill.java:**

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

**Add Skill to Employee:**

Employee emp = employeeService.get(1);Skill skill = skillService.get(3);

emp.getSkillList().add(skill);

employeeService.save(emp);

**Output Log:**

Employee: JohnSkills: [Java, Python, Docker]Hibernate: insert into employee\_skill values (1, 3)