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

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContaining(String keyword); List<Country>

findByNameContainingOrderByNameAsc(String keyword); List<Country> findByNameStartingWith(String prefix);

}

private static CountryRepository countryRepository; private static void testCountrySearch() {

List<Country> result = countryRepository.findByNameContaining("ou");

result.forEach(System.out::println); List<Country> sortedResult =

countryRepository.findByNameContainingOrderByNameAsc( "ou");

sortedResult.forEach(System.out::println); List<Country> startsWithZ =

countryRepository.findByNameStartingWith("Z"); startsWithZ.forEach(System.out::println);

}

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

public interface StockRepository extends JpaRepository<Stock, Integer> {

List<Stock> findByCodeAndDateBetween(String code, LocalDate start, LocalDate end);

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

List<Stock> findTop3ByOrderByVolumeDesc();

List<Stock> findTop3ByCodeOrderByCloseAsc(String code);

}

private static StockRepository stockRepository; private static void testStockQueries() {

List<Stock> fbSept = stockRepository.findByCodeAndDateBetween("FB", LocalDate.of(2019, 9, 1), LocalDate.of(2019, 9, 30));

fbSept.forEach(System.out::println); List<Stock> googleHigh =

stockRepository.findByCodeAndCloseGreaterThan("GOOGL ", new BigDecimal("1250"));

googleHigh.forEach(System.out::println); List<Stock> topVolumes =

stockRepository.findTop3ByOrderByVolumeDesc(); topVolumes.forEach(System.out::println);

List<Stock>netflixLow=stockRepository.findTop3ByCodeOr derByCloseAsc("NFLX");

netflixLow.forEach(System.out::println);

}

# Hands-on 3: Create Payroll Tables and Bean Mapping:

@Entity

@Table(name = "employee") public class Employee {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

private int id; private String name;

private double salary; private boolean permanent;

@Column(name = "date\_of\_birth") private Date dateOfBirth;

// + department (ManyToOne)

}

@Entity

@Table(name = "department")

public class Department {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

private int id; private String name;

// + employeeList (OneToMany)

}

@Entity

@Table(name = "skill") public class Skill {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

private int id; private String name;

// + employeeList (ManyToMany)

}

# Hands-on 4: Many-to-One (Employee ↔ Department):

@ManyToOne

@JoinColumn(name = "em\_dp\_id") private Department department; private static void testGetEmployee() {

Employee e = employeeService.get(1); System.out.println(e); System.out.println(e.getDepartment());

}

private static void testAddEmployee() { Employee e = new Employee(); e.setName("John"); e.setSalary(50000); e.setPermanent(true); e.setDateOfBirth(new Date());

Department d = departmentService.get(1); e.setDepartment(d); employeeService.save(e); System.out.println(e);

}

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

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

private Set<Employee> employeeList; private static void testGetDepartment() {

Department d = departmentService.get(1); System.out.println(d); d.getEmployeeList().forEach(System.out::println);

}

**Hands-on 6: Many-to-Many (Employee ↔ Skill):** @ManyToMany(fetch = FetchType.EAGER) @JoinTable(name = "employee\_skill",

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

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

private static void testAddSkillToEmployee() { Employee e = employeeService.get(1);

Skill s = skillService.get(2); e.getSkillList().add(s); employeeService.save(e);

}

