# Hands-on 1: Spring Data JPA Quick Example

Setup Spring Boot project, datasource, repository, and basic test method.

## Code

// application.properties  
spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn  
spring.datasource.username=root  
spring.datasource.password=root  
spring.jpa.hibernate.ddl-auto=validate  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect  
logging.level.org.springframework=info  
logging.level.com.cognizant=debug  
logging.level.org.hibernate.SQL=trace  
logging.level.org.hibernate.type.descriptor.sql=trace  
  
// OrmLearnApplication.java  
@SpringBootApplication  
public class OrmLearnApplication {  
 private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);  
 private static CountryService countryService;  
 public static void main(String[] args) {  
 ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);  
 countryService = context.getBean(CountryService.class);  
 LOGGER.info("Inside main");  
 testGetAllCountries();  
 }  
 private static void testGetAllCountries() {  
 LOGGER.info("Start");  
 List<Country> countries = countryService.getAllCountries();  
 LOGGER.debug("countries={}", countries);  
 LOGGER.info("End");  
 }  
}

## Sample Output

06-07-25 10:45:12.345 main INFO com.cognizant.ormlearn.OrmLearnApplication main 12 Inside main  
06-07-25 10:45:12.347 main INFO com.cognizant.ormlearn.OrmLearnApplication testGetAllCountries 16 Start  
06-07-25 10:45:12.456 main DEBUG com.cognizant.ormlearn.OrmLearnApplication testGetAllCountries 18 countries=[Country [code=IN, name=India], Country [code=US, name=United States of America]]  
06-07-25 10:45:12.456 main INFO com.cognizant.ormlearn.OrmLearnApplication testGetAllCountries 19 End

# Hands-on 2: Hibernate XML Config

Shows XML-based mapping of Employee object to DB table.

## Code

// hibernate.cfg.xml mapping  
<mapping resource="Employee.hbm.xml"/>  
// Employee.hbm.xml  
<class name="Employee" table="EMPLOYEE">  
 <id name="id" type="int" column="ID">  
 <generator class="native"/>  
 </id>  
 <property name="firstName" column="FIRST\_NAME" type="string"/>  
 <property name="lastName" column="LAST\_NAME" type="string"/>  
</class>

## Sample Output

Hibernate: insert into EMPLOYEE (FIRST\_NAME, LAST\_NAME) values (?, ?)  
Hibernate: select \* from EMPLOYEE

# Hands-on 3: Hibernate Annotations

Uses @Entity, @Table, @Id, @Column to map Employee class to table.

## Code

@Entity  
@Table(name = "EMPLOYEE")  
public class Employee {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private int id;  
 @Column(name = "FIRST\_NAME")  
 private String firstName;  
 @Column(name = "LAST\_NAME")  
 private String lastName;  
}

## Sample Output

Hibernate: insert into EMPLOYEE (FIRST\_NAME, LAST\_NAME) values (?, ?)  
Hibernate: select \* from EMPLOYEE

# Hands-on 4: JPA vs Hibernate vs Spring Data JPA

Comparison between manual Hibernate and Spring Data JPA code.

## Code

// Hibernate  
session = factory.openSession();  
tx = session.beginTransaction();  
session.save(employee);  
tx.commit();  
  
// Spring Data JPA  
employeeRepository.save(employee);

## Sample Output

Hibernate: insert into EMPLOYEE (FIRST\_NAME, LAST\_NAME) values (?, ?)

# Hands-on 5: Populate Country Table

Insert initial set of countries into database.

## Code

insert into country (co\_code, co\_name) values ('IN', 'India');  
insert into country (co\_code, co\_name) values ('US', 'United States of America');  
insert into country (co\_code, co\_name) values ('FR', 'France');

## Sample Output

mysql> select \* from country;  
co\_code | co\_name   
  
FR | France   
IN | India   
US | United States of America

# Hands-on 6: Find Country by Code

Service method to find a country by code and handle not found exception.

## Code

@Transactional  
public Country findCountryByCode(String code) throws CountryNotFoundException {  
 Optional<Country> result = countryRepository.findById(code);  
 if (!result.isPresent()) throw new CountryNotFoundException("Country not found");  
 return result.get();  
}

## Sample Output

DEBUG: Country: Country [code=IN, name=India]

# Hands-on 7: Add New Country

Service method to add a new country.

## Code

@Transactional  
public void addCountry(Country country) {  
 countryRepository.save(country);  
}

## Sample Output

mysql> select \* from country where co\_code='JP';  
  
 co\_code | co\_name  
  
 JP | Japan

# Hands-on 8: Update Country

Service method to update an existing country by code.

## Code

@Transactional  
public void updateCountry(String code, String name) throws CountryNotFoundException {  
 Optional<Country> result = countryRepository.findById(code);  
 if (!result.isPresent()) throw new CountryNotFoundException("Country not found");  
 Country country = result.get();  
 country.setName(name);  
 countryRepository.save(country);  
}

## Sample Output

mysql> select \* from country where co\_code='JP';

co\_code | co\_name   
JP | Japan Updated

# Hands-on 9: Delete Country

Service method to delete a country by code.

## Code

@Transactional  
public void deleteCountry(String code) {  
 countryRepository.deleteById(code);  
}

## Sample Output

mysql> select \* from country where co\_code='JP';  
Empty set (0.00 sec)

**HANDS-ON 1: Country Queries**

// Country.java

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

// Getters, setters, toString()

}

// CountryRepository.java

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContaining(String keyword);

List<Country> findByNameContainingOrderByNameAsc(String keyword);

List<Country> findByNameStartingWith(String prefix);

}

// OrmLearnApplication.java

private static void testFindByNameContaining() {

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

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

}

private static void testFindByNameContainingOrderByNameAsc() {

List<Country> countries = countryRepository.findByNameContainingOrderByNameAsc("ou");

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

}

private static void testFindByNameStartingWith() {

List<Country> countries = countryRepository.findByNameStartingWith("Z");

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

}

// Sample Output:

BV - Bouvet Island

DJ - Djibouti

GP - Guadeloupe

...

ZW - Zimbabwe

**HANDS-ON 2: Stock Queries**

// Stock.java

@Entity

@Table(name = "stock")

public class Stock {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name = "st\_code")

private String code;

@Column(name = "st\_date")

private LocalDate date;

@Column(name = "st\_open")

private BigDecimal open;

@Column(name = "st\_close")

private BigDecimal close;

@Column(name = "st\_volume")

private long volume;

// Getters, setters, toString()

}

// StockRepository.java

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);

}

// OrmLearnApplication.java

private static void testFacebookSept2019() {

List<Stock> stocks = stockRepository.findByCodeAndDateBetween("FB",

LocalDate.of(2019, 9, 1), LocalDate.of(2019, 9, 30));

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

}

private static void testGoogleCloseAbove1250() {

List<Stock> stocks = stockRepository.findByCodeAndCloseGreaterThan("GOOGL",

BigDecimal.valueOf(1250));

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

}

private static void testTop3ByVolume() {

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

}

private static void testNetflixLowestPrices() {

stockRepository.findTop3ByCodeOrderByCloseAsc("NFLX").forEach(System.out::println);

}

// Sample Output: (see prompt)

**HANDS-ON 3: Bean Mapping**

// Employee.java

@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 = "em\_date\_of\_birth")

private Date dateOfBirth;

// Getters, setters, toString()

}

// Department.java and Skill.java are similar

**HANDS-ON 4: Many-to-One**

// Employee.java (Add this)

@ManyToOne

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

private Department department;

// OrmLearnApplication.java

private static void testGetEmployee() {

Employee employee = employeeService.get(1);

LOGGER.debug("Employee: {}", employee);

LOGGER.debug("Department: {}", employee.getDepartment());

}

private static void testAddEmployee() {

Employee emp = new Employee();

emp.setName("Test Emp");

emp.setSalary(50000);

emp.setPermanent(true);

emp.setDateOfBirth(Date.valueOf("2000-01-01"));

emp.setDepartment(departmentService.get(1));

employeeService.save(emp);

}

**HANDS-ON 5: One-to-Many**

// Department.java (Add this)

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

private Set<Employee> employeeList;

// OrmLearnApplication.java

private static void testGetDepartment() {

Department department = departmentService.get(1);

LOGGER.debug("Department: {}", department);

LOGGER.debug("Employees: {}", department.getEmployeeList());

}

**HANDS-ON 6: Many-to-Many**

// 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;

// Skill.java

@ManyToMany(mappedBy = "skillList")

private Set<Employee> employeeList;

// OrmLearnApplication.java

private static void testAddSkillToEmployee() {

Employee employee = employeeService.get(1);

Skill skill = skillService.get(2);

employee.getSkillList().add(skill);

employeeService.save(employee);

}

private static void testGetEmployeeWithSkills() {

Employee employee = employeeService.get(1);

LOGGER.debug("Skills: {}", employee.getSkillList());

}

**Hands-on 2: Get all permanent employees using HQL**

// EmployeeRepository.java

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

@Query("SELECT e FROM Employee e WHERE e.permanent = 1")

List<Employee> getAllPermanentEmployees();

@Query("SELECT e FROM Employee e LEFT JOIN FETCH e.department d LEFT JOIN FETCH e.skillList WHERE e.permanent = 1")

List<Employee> getAllPermanentEmployeesOptimized();

}

// EmployeeService.java

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

public List<Employee> getAllPermanentEmployees() {

return employeeRepository.getAllPermanentEmployeesOptimized();

}

}

// OrmLearnApplication.java

private static void testGetAllPermanentEmployees() {

LOGGER.info("Start");

List<Employee> employees = employeeService.getAllPermanentEmployees();

for (Employee e : employees) {

LOGGER.debug("Employee: {}", e);

LOGGER.debug("Skills: {}", e.getSkillList());

}

LOGGER.info("End");

}

Sample Output (Console)

Employee: Employee(id=2, name=John, ...)

Skills: [Java, Spring Boot]

**Hands-on 3: Fetch Quiz Attempt**

// AttemptRepository.java

@Repository

public interface AttemptRepository extends JpaRepository<Attempt, Integer> {

@Query("SELECT a FROM Attempt a JOIN FETCH a.user u JOIN FETCH a.attemptQuestions aq JOIN FETCH aq.question q JOIN FETCH aq.attemptOptions ao JOIN FETCH ao.option o WHERE u.id = :userId AND a.id = :attemptId")

Attempt getAttempt(@Param("userId") int userId, @Param("attemptId") int attemptId);

}

// AttemptService.java

@Service

public class AttemptService {

@Autowired

private AttemptRepository attemptRepository;

public Attempt getAttempt(int userId, int attemptId) {

return attemptRepository.getAttempt(userId, attemptId);

}

}

// OrmLearnApplication.java

private static void testGetAttempt() {

Attempt attempt = attemptService.getAttempt(1, 1);

System.out.println("Username: " + attempt.getUser().getUsername());

System.out.println("Attempt Date: " + attempt.getAttemptDate());

for (AttemptQuestion aq : attempt.getAttemptQuestions()) {

System.out.println(aq.getQuestion().getText());

for (AttemptOption ao : aq.getAttemptOptions()) {

System.out.printf("%d) %s\t%.1f\t%b\n",

ao.getOption().getId(),

ao.getOption().getText(),

ao.getOption().getScore(),

ao.isSelected()

);

}

}

}

Sample Output:

What is the extension of the hyper text markup language file?

1) .xhtm 0.0 false

2) .ht 0.0 false

3) .html 1.0 true

4) .htmx 0.0 false

**Hands-on 4: Get average salary using HQL**

// EmployeeRepository.java

@Query("SELECT AVG(e.salary) FROM Employee e WHERE e.department.id = :id")

double getAverageSalary(@Param("id") int deptId);

// EmployeeService.java

public double getAverageSalary(int deptId) {

return employeeRepository.getAverageSalary(deptId);

}

// OrmLearnApplication.java

private static void testGetAverageSalary() {

double avg = employeeService.getAverageSalary(2);

LOGGER.debug("Average Salary: {}", avg);

}

Sample Output:

Average Salary: 75000.0

**Hands-on 5: Native Query to get all employees**

// EmployeeRepository.java

@Query(value = "SELECT \* FROM employee", nativeQuery = true)

List<Employee> getAllEmployeesNative();

// EmployeeService.java

public List<Employee> getAllEmployeesNative() {

return employeeRepository.getAllEmployeesNative();

}

// OrmLearnApplication.java

private static void testGetAllEmployeesNative() {

List<Employee> list = employeeService.getAllEmployeesNative();

list.forEach(e -> LOGGER.debug("Employee: {}", e));

}

**Hands-on 6: Criteria Query**

// ProductRepositoryImpl.java

public class ProductRepositoryImpl {

@PersistenceContext

private EntityManager em;

public List<Product> searchProducts(String cpu, String os, int ramSize) {

CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<Product> cq = cb.createQuery(Product.class);

Root<Product> root = cq.from(Product.class);

List<Predicate> predicates = new ArrayList<>();

if (cpu != null) predicates.add(cb.equal(root.get("cpu"), cpu));

if (os != null) predicates.add(cb.equal(root.get("os"), os));

if (ramSize > 0) predicates.add(cb.ge(root.get("ram"), ramSize));

cq.where(predicates.toArray(new Predicate[0]));

return em.createQuery(cq).getResultList();

}

}

Sample Output (Console)

Product: Dell, i7, 16GB, Windows

Product: HP, i5, 8GB, Windows

**Exercise 1: Employee Management System - Overview and Setup**

Business Scenario:  
You are developing an employee management system that will manage employee data, departments, and their relationships.

Instructions:

1. Creating a Spring Boot Project:

* Initialize a new Spring Boot project named EmployeeManagementSystem.
* Add dependencies: Spring Data JPA, H2 Database, Spring Web, Lombok.

1. Configuring Application Properties:

* Configure application.properties for H2 database connection.

spring.datasource.url=jdbc:h2:mem:testdb  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=password  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
spring.h2.console.enabled=true

**Exercise 2: Employee Management System - Creating Entities**

Business Scenario:  
Define JPA entities for Employee and Department with appropriate relationships.

Instructions:

1. Creating JPA Entities:

Department.java

@Entity  
public class Department {  
@Id  
@GeneratedValue(strategy = GenerationType.IDENTITY)  
private Long id;  
private String name;

@OneToMany(mappedBy = "department")

private List<Employee> employees;

}

Employee.java

@Entity  
public class Employee {  
@Id  
@GeneratedValue(strategy = GenerationType.IDENTITY)  
private Long id;  
private String name;  
private String email;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

}

**Exercise 3: Employee Management System - Creating Repositories**

Business Scenario:  
Create repositories for Employee and Department entities to perform CRUD operations.

Instructions:

EmployeeRepository.java

@Repository  
public interface EmployeeRepository extends JpaRepository<Employee, Long> {  
List findByDepartmentId(Long departmentId);  
}

DepartmentRepository.java

@Repository  
public interface DepartmentRepository extends JpaRepository<Department, Long> {  
}

**Exercise 4: Employee Management System - Implementing CRUD Operations**

Business Scenario:  
Implement CRUD operations for managing employees and departments.

Instructions:

EmployeeController.java

@RestController  
@RequestMapping("/employees")  
public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

@GetMapping

public List<Employee> getAll() {

return employeeRepository.findAll();

}

@PostMapping

public Employee save(@RequestBody Employee employee) {

return employeeRepository.save(employee);

}

@DeleteMapping("/{id}")

public void delete(@PathVariable Long id) {

employeeRepository.deleteById(id);

}

}

**Exercise 5: Employee Management System - Defining Query Methods**

Business Scenario:  
Enhance your repository to support custom queries.

Instructions:

Derived Method:  
List findByNameContaining(String keyword);

Using @Query:

@Query("SELECT e FROM Employee e WHERE e.email LIKE %:email%")  
List searchByEmail(@Param("email") String email);

**Exercise 6: Employee Management System - Implementing Pagination and Sorting**

Business Scenario:  
Add pagination and sorting capabilities to your employee search functionality.

Instructions:

@GetMapping("/page")  
public Page getPagedEmployees(Pageable pageable) {  
return employeeRepository.findAll(pageable);  
}

Example Request:  
GET /employees/page?page=0&size=5&sort=name,asc

**Exercise 7: Employee Management System - Enabling Entity Auditing**

Business Scenario:  
Implement auditing to track the creation and modification of employees and departments.

Instructions:

Enable JPA Auditing in Main Class:

@EnableJpaAuditing  
@SpringBootApplication  
public class EmployeeManagementSystemApplication {  
public static void main(String[] args) {  
SpringApplication.run(EmployeeManagementSystemApplication.class, args);  
}  
}

Add Audit Fields:

@Entity  
@EntityListeners(AuditingEntityListener.class)  
public class Employee {

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime modifiedDate;

}

application.properties:  
spring.jpa.properties.hibernate.jdbc.time\_zone=UTC

**Exercise 8: Employee Management System - Creating Projections**

Business Scenario:  
Create projections to fetch specific data subsets from the employee and department entities.

Instructions:

Interface-Based Projection:

public interface EmployeeNameOnly {  
String getName();  
}

Repository Method:

List findByDepartmentId(Long id);

**Exercise 9: Employee Management System - Customizing Data Source Configuration**

Business Scenario:  
Customize your data source configuration and manage multiple data sources.

Instructions:

application.properties

spring.datasource.url=jdbc:h2:mem:mainDB  
spring.secondary-datasource.url=jdbc:h2:mem:secondaryDB

Configuration Classes (Advanced):  
Use @Primary and @Qualifier for managing multiple datasources.

**Exercise 10: Employee Management System - Hibernate-Specific Features**

Business Scenario:  
Leverage Hibernate-specific features to enhance your application's performance and capabilities.

Instructions:

Use Hibernate-Specific Annotations:

@org.hibernate.annotations.DynamicUpdate  
@org.hibernate.annotations.Cache(usage = CacheConcurrencyStrategy.READ\_WRITE)

Configure Hibernate Properties:

spring.jpa.properties.hibernate.format\_sql=true  
spring.jpa.properties.hibernate.order\_inserts=true  
spring.jpa.properties.hibernate.jdbc.batch\_size=30

Batch Processing Example:

for (int i = 0; i < 1000; i++) {  
Employee emp = new Employee("Name" + i, "email" + i + "@test.com");  
employeeRepository.save(emp);  
}

Sample Output (Postman / Console):

Get All Employees:  
[  
{  
"id": 1,  
"name": "John",  
"email": "[john@example.com](mailto:john@example.com)",  
"department": {  
"id": 1,  
"name": "IT"  
}  
}  
]

Pagination Output:  
GET /employees/page?page=0&size=2  
Content: [  
{ "id": 1, "name": "John" },  
{ "id": 2, "name": "Emma" }  
]