**Spring Data JPA 1 – Hands-On Assignment**

Name: Divyansh Tiwari

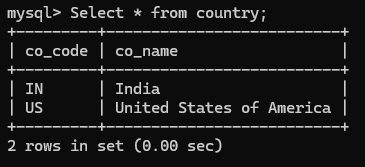
Superset ID: 6394658

**Hands-on 1: Introduction to Spring Data JPA – Country Entity & Basic CRUD**

Goal: To set up a Spring Boot application integrated with Spring Data JPA and perform basic database operations (read all countries) using the Country entity.

Steps followed:

1. Created Country Table in SQL database.



Created table

1. Created Country Entity

This is done via the following:

*@Entity*

*@Table(name = "country")*

*public class Country {*

*@Id*

*@Column(name = "co\_code")*

*private String code;*

*@Column(name = "co\_name")*

*private String name;*

*}*

1. Created Repository Interface

*@Repository*

*public interface CountryRepository extends JpaRepository<Country, String> {*

*}*

1. Created Service Layer

*@Service*

*public class CountryService {*

*@Autowired*

*private CountryRepository countryRepository;*

*public List<Country> getAllCountries() {*

*return countryRepository.findAll();*

*}*

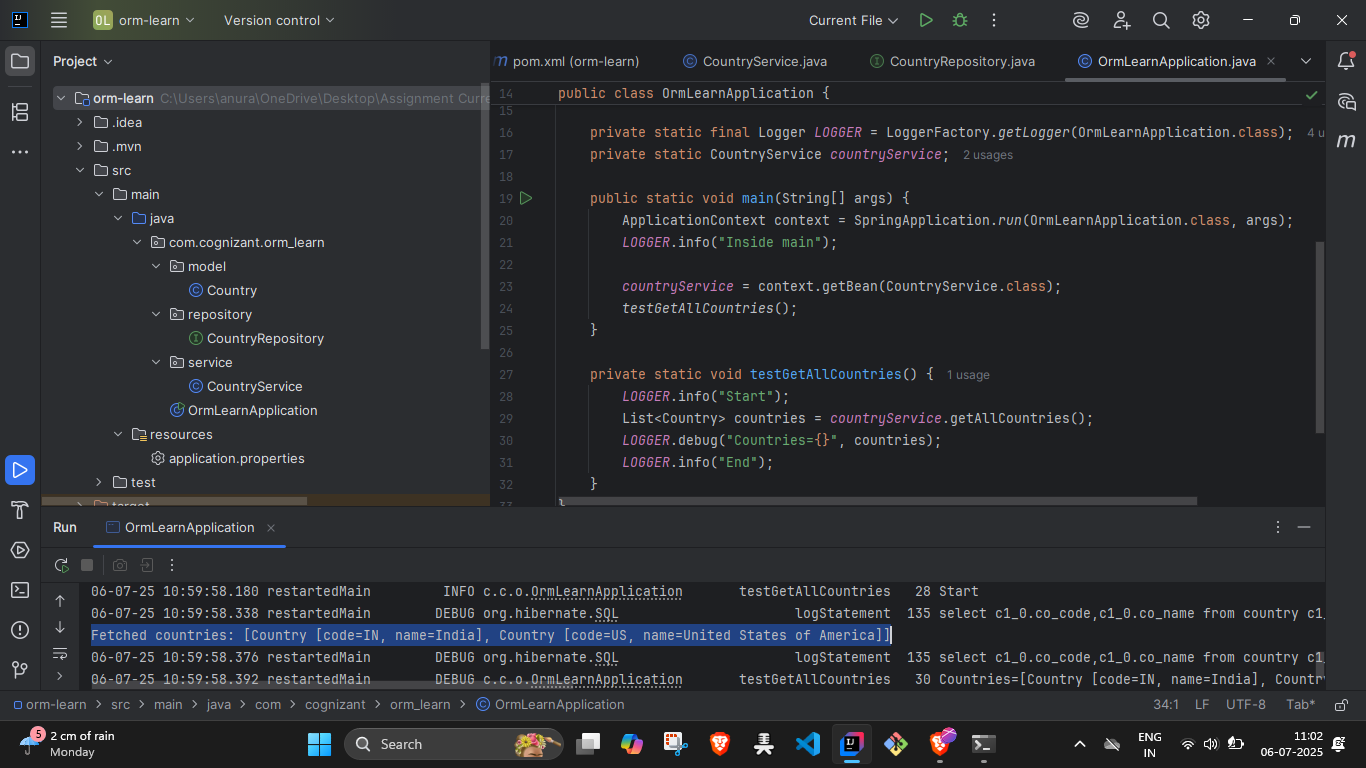
*}*

1. Tested in OrmLearnApplication.java

*ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);*

*countryService = context.getBean(CountryService.class);*

*testGetAllCountries();*

OUTPUT: 

Output after testing, also testing method used is included in the screenshot.

**Hands-On 2: Hibernate XML Configuration – Conceptual Implementation.**

Goal: To understand the **fundamentals of Hibernate using XML-based configuration**, and explore the lifecycle of a typical Hibernate application using:

* Object-relational mapping in XML
* Hibernate core components (SessionFactory, Session, Transaction, etc.)
* CRUD operations using the Hibernate API

**1. Object-to-Relational Mapping in Hibernate XML**

In Hibernate ,entity-to-table mapping is handled via an XML file like Employee.hbm.xml.

**Example:**

xml

Code:

*<hibernate-mapping>*

*<class name="com.example.Employee" table="EMPLOYEE">*

*<id name="id" column="ID">*

*<generator class="native"/>*

*</id>*

*<property name="firstName" column="FIRST\_NAME"/>*

*<property name="lastName" column="LAST\_NAME"/>*

*<property name="salary" column="SALARY"/>*

*</class>*

*</hibernate-mapping>*

* <class>: maps a Java class (Employee) to a db table (EMPLOYEE)
* <id>: maps the primary key field, including generation strategy
* <property>: maps each field in the Java class to a db column

**2. Hibernate Configuration File (hibernate.cfg.xml)**

Code:

*<hibernate-configuration>*

*<session-factory>*

*<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>*

*<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>*

*<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/yourdb</property>*

*<property name="hibernate.connection.username">root</property>*

*<property name="hibernate.connection.password">root</property>*

*<mapping resource="Employee.hbm.xml"/>*

*</session-factory>*

*</hibernate-configuration>*

* declares database connection details
* specifies SQL dialect (MySQL, Oracle, etc.)
* registers the XML mapping file

**3. Key Hibernate Concepts**

**SessionFactory**

* A thread-safe, heavyweight object created **once** for the entire application.
* Configured using hibernate.cfg.xml.
* Used to obtain Session objects.

Code

*SessionFactory factory = new Configuration().configure().buildSessionFactory();*

**Session**

* Lightweight, **non-thread-safe** object used to perform CRUD.
* Represents a single **unit of work**.

Code

*Session session = factory.openSession();*

**Transaction**

* Encapsulates one or more operations into a single atomic unit.

code

*Transaction tx = session.beginTransaction();*

**beginTransaction(), commit(), rollback()**

code

* commit(): Saves changes to DB
* rollback(): Undoes if an error occurs

**session.save(Object)**

* persists a **transient object** to the database
* returns generated identifier

code

*Employee emp = new Employee("John", "Doe", 50000);*

*session.save(emp);*

**session.get(Class, id)**

* Fetches an object by primary key (returns null if not found)

code

*Employee emp = session.get(Employee.class, 1);*

**session.createQuery().list()**

* Executes HQL (Hibernate Query Language) and returns results

code

*List<Employee> list = session.createQuery("FROM Employee").list();*

**session.delete(Object)**

* Deletes a persistent object from the database

code

*Employee emp = session.get(Employee.class, 1);*

*session.delete(emp);*

**Hands-On 3: Hibernate Annotation Config implementation**

Goal: Model a **One-to-Many relationship** between Company and Stock, where one company has many stocks, but each stock has only one company.

**Object-to-Relational Mapping with Annotations**

In the annotation-based approach, mapping between the Java class and database table is done **directly in the class file** using JPA annotations.

**The Persistence Class: Employee**

code

*import jakarta.persistence.\*;*

*@Entity*

*@Table(name = "EMPLOYEE")*

*public class Employee {*

*@Id*

*@GeneratedValue(strategy = GenerationType.IDENTITY)*

*@Column(name = "id")*

*private int id;*

*@Column(name = "first\_name")*

*private String firstName;*

*@Column(name = "last\_name")*

*private String lastName;*

*@Column(name = "salary")*

*private int salary;*

*}*

Annotations Explanation:

|  |
| --- |
| @Entity: marks the class as persistent entity, mapped to a db table.  @Table: maps the class to a specific table  @Id: marks primary key of the entity  @GeneratedValue: automatically generates primary key values, with chosen strategies.  @Column: maps a class field to a specific db column |

**Hibernate Configuration – hibernate.cfg.xml**

code

*<hibernate-configuration>*

*<session-factory>*

*<!-- Dialect -->*

*<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>*

*<!-- JDBC Driver -->*

*<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>*

*<!-- DB URL -->*

*<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/test</property>*

*<!-- DB Credentials -->*

*<property name="hibernate.connection.username">root</property>*

*<property name="hibernate.connection.password">1234</property>*

*<!-- Register Annotated Class -->*

*<mapping class="com.example.Employee"/>*

*</session-factory>*

*</hibernate-configuration>*

**Configuration Elements:**

hibernate.dialect

Instructs Hibernate how to generate SQL for your DB (e.g., MySQLDialect)

hibernate.connection.driver\_class

JDBC driver class for the database

hibernate.connection.url

JDBC URL to connect to your DB

hibernate.connection.username

DB login username

hibernate.connection.password

DB login password

Mapping class

Registers your annotated entity class

**Using the Annotated Class in Hibernate Code**

code

*SessionFactory factory = new Configuration().configure().buildSessionFactory();*

*Session session = factory.openSession();*

*Transaction tx = session.beginTransaction();*

*Employee emp = new Employee("John", "Doe", 50000);*

*session.save(emp);*

*tx.commit();*

*session.close();*

*factory.close();*

**Hands-On 4: Difference between JPA, Hibernate and Spring Data JPA**

Goal: understand the layered architecture of persistence in Java applications, and how JPA, Hibernate, and Spring Data JPA are related — including a side-by-side code comparison.

**Java Persistence API (JPA)**

* JPA is a *Java specification* (JSR 338) that defines how to manage relational data using Java objects.
* It is not a framework, but a set of interfaces and annotations (e.g., @Entity, @Id, @OneToMany).
* It does not provide an implementation — you need a provider (like Hibernate) to use it.

**2. Hibernate**

* Hibernate is a popular Object Relational Mapping (ORM) framework in Java.
* It is a concrete implementation of the JPA specification.
* Hibernate also has native APIs (non-JPA), which provide additional functionality.
* Manages DB connections, entity states, transactions, and queries.

**3. Spring Data JPA**

* **Spring Data JPA** is a wrapper/abstraction over JPA and Hibernate.
* It *does not implement* JPA or Hibernate itself — instead, it:
  + Simplifies CRUD operations using repositories
  + Reduces boilerplate code
  + Manages transactions behind the scenes
  + Integrates easily with Spring Boot

**Code Comparison**

**Hibernate (Manual Approach)**

code

*public Integer addEmployee(Employee employee) {*

*Session session = factory.openSession();*

*Transaction tx = null;*

*Integer employeeID = null;*

*try {*

*tx = session.beginTransaction();*

*employeeID = (Integer) session.save(employee);*

*tx.commit();*

*} catch (HibernateException e) {*

*if (tx != null) tx.rollback();*

*e.printStackTrace();*

*} finally {*

*session.close();*

*}*

*return employeeID;*

*}*

**Explanation:**

* Manual session and transaction handling
* Explicit exception management
* More lines of code for simple persistence

**Spring Data JPA (Declarative + Abstracted)**

**EmployeeRepository.java**

code

*public interface EmployeeRepository extends JpaRepository<Employee, Integer> {*

*}*

EmployeeService.java

code

*@Autowired*

*private EmployeeRepository employeeRepository;*

*@Transactional*

*public void addEmployee(Employee employee) {*

*employeeRepository.save(employee);*

*}*

**Explanation:**

* No need to manually open sessions or manage transactions
* Uses Spring’s @Transactional
* JpaRepository provides built-in save(), findAll(), etc.
* Clean, concise, testable

**Hands-On 5: Implementing Services for Managing Country**

To implement full CRUD functionality and search capability on the Country entity using Spring Data JPA's service and repository layers.

Steps followed:

1. Finding a Country by Code

*Optional<Country> findById(String code);*

In CountryService.java

*public Country findCountryByCode(String code) throws CountryNotFoundException {*

*return countryRepository.findById(code)*

*.orElseThrow(() -> new CountryNotFoundException("Country not found: " + code));*

*}*

1. Added a new Country

*public void addCountry(Country country) {*

*countryRepository.save(country);*

*}*

1. Updating an existing country

*CREATE TABLE employee\_skill (*

*es\_em\_id INT,*

*es\_sk\_id INT,*

*FOREIGN KEY (es\_em\_id) REFERENCES employee(em\_id),*

*FOREIGN KEY (es\_sk\_id) REFERENCES skill(sk\_id),*

*PRIMARY KEY (es\_em\_id, es\_sk\_id)*

*);*

1. Deleting a Country

*public void deleteCountry(String code) {*

*countryRepository.deleteById(code);*

*}*

1. Finding a Country by partial name [string]

In CountryRepository:

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

In CountryService.java

*public List<Country> findCountriesByNameContaining(String keyword) {*

*return countryRepository.findByNameContaining(keyword);*

*}*

1. Testing in OrmLearnApplication.java

The output screenshots are included with each operation under the appropriate Hands-On.

**Hands-On 6: Find a Country Based on Country Code**

**Goal**: Implement a service method to retrieve a country using its code, with proper exception handling.

Steps followed:

1. Created custom exception

*package com.cognizant.orm\_learn.service.exception;*

*public class CountryNotFoundException extends Exception {*

*public CountryNotFoundException(String message) {*

*super(message);*

*}*

*}*

1. Updated Service

*@Transactional*

*public Country findCountryByCode(String countryCode) throws CountryNotFoundException {*

*Optional<Country> result = countryRepository.findById(countryCode);*

*if (!result.isPresent()) {*

*throw new CountryNotFoundException("Country not found with code: " + countryCode);*

*}*

*return result.get();*

*}*

1. Tested via OrmLearnApplication

*private static void testFindCountryByCode() throws CountryNotFoundException {*

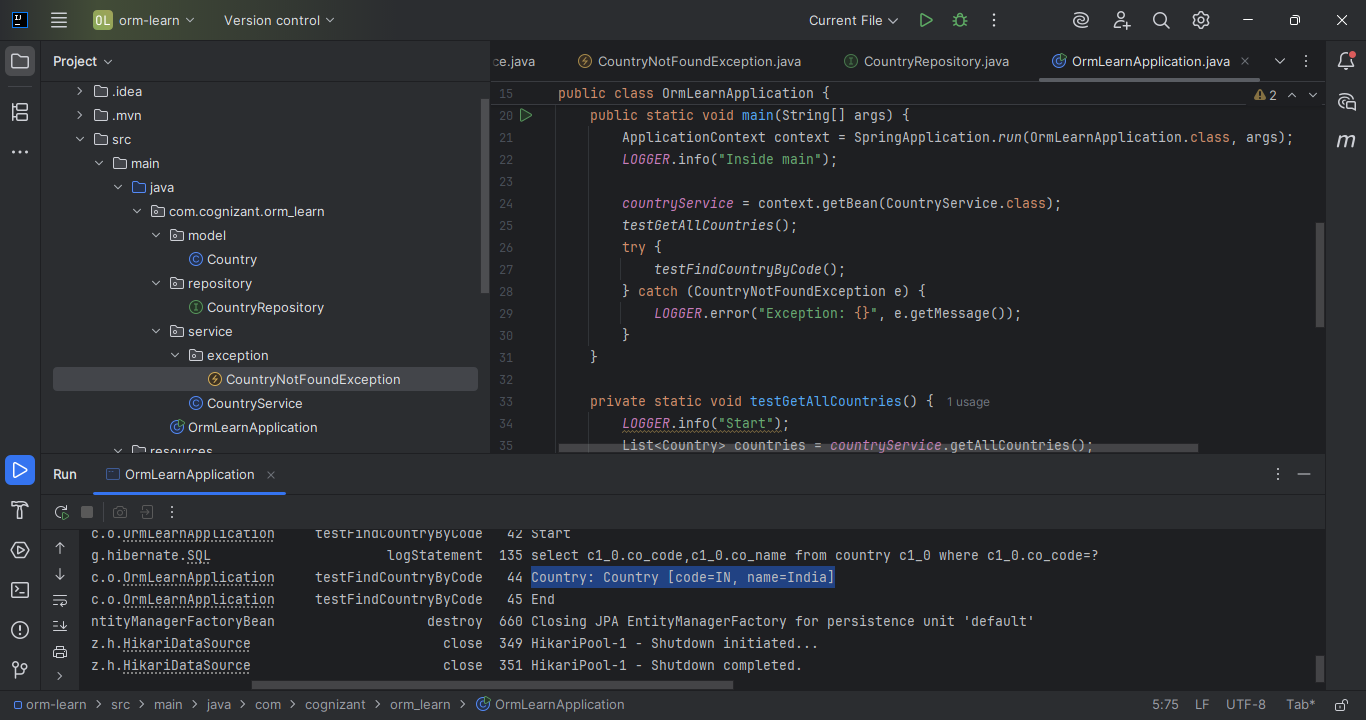
*LOGGER.info("Start");*

*Country country = countryService.findCountryByCode("IN");*

*LOGGER.debug("Country: {}", country);*

*LOGGER.info("End");*

*}*

TEST OUTPUT: **

Test outputs are selected for clarity.

**Hands-On 7: Add a New Country**

Goal: To add a new country record using Spring Data JPA.

Steps followed:

1. Service Method in the service package’s CountryService

*@Transactional*

*public void addCountry(Country country) {*

*countryRepository.save(country);*

*}*

1. Test method

*private static void testAddCountry() {*

*LOGGER.info("Start");*

*Country country = new Country();*

*country.setCode("JP");*

*country.setName("Japan");*

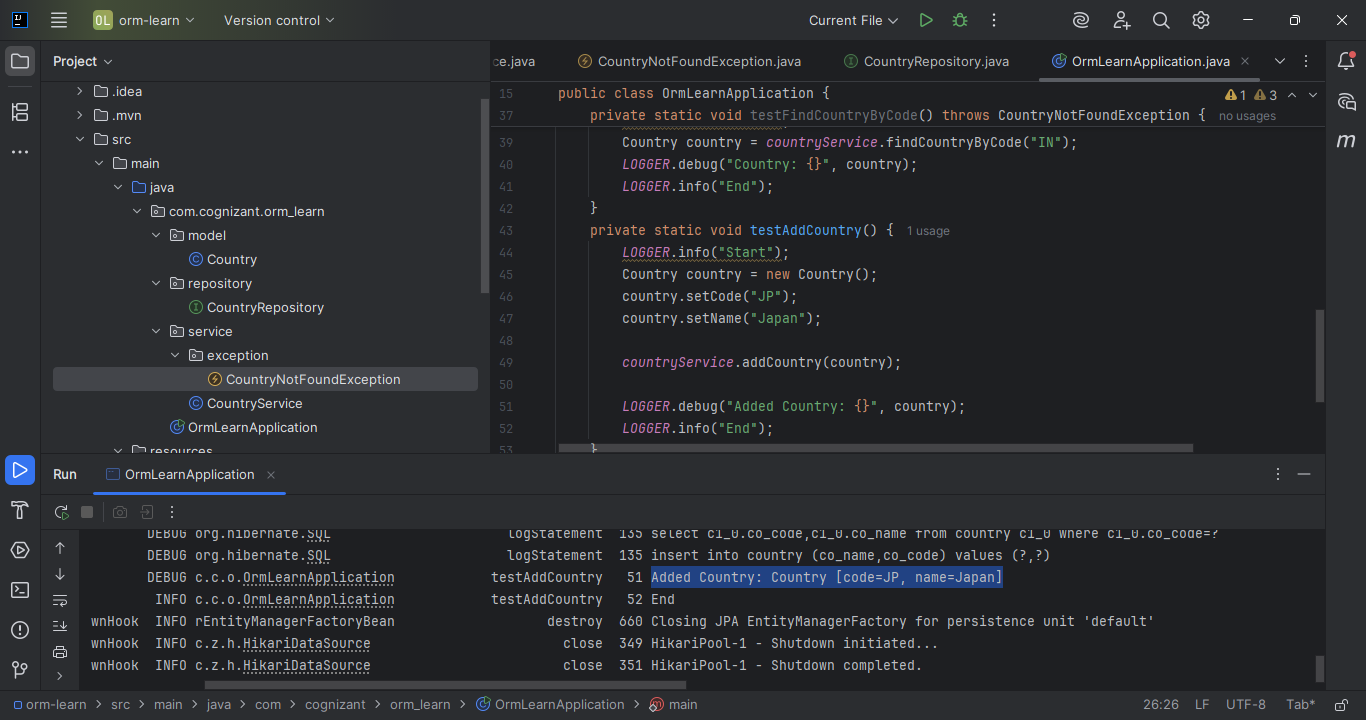
*countryService.addCountry(country);*

*Country added = countryService.findCountryByCode("JP");*

*LOGGER.debug("Added Country: {}", added);*

*LOGGER.info("End");*

*}*

TEST OUTPUT: **

Test outputs are selected for clarity.

**Hands-On 8: Update Country based on Code**

Goal: Update the name of a country based on its code.

Steps followed:

1. Service method updated inside CountryService

*@Transactional*

*public void updateCountry(String code, String name) throws CountryNotFoundException {*

*Country country = findCountryByCode(code);*

*country.setName(name);*

*countryRepository.save(country);*

*}*

1. Test Method in the OrmLearnApplication.java

*private static void testUpdateCountry() throws CountryNotFoundException {*

*LOGGER.info("Start");*

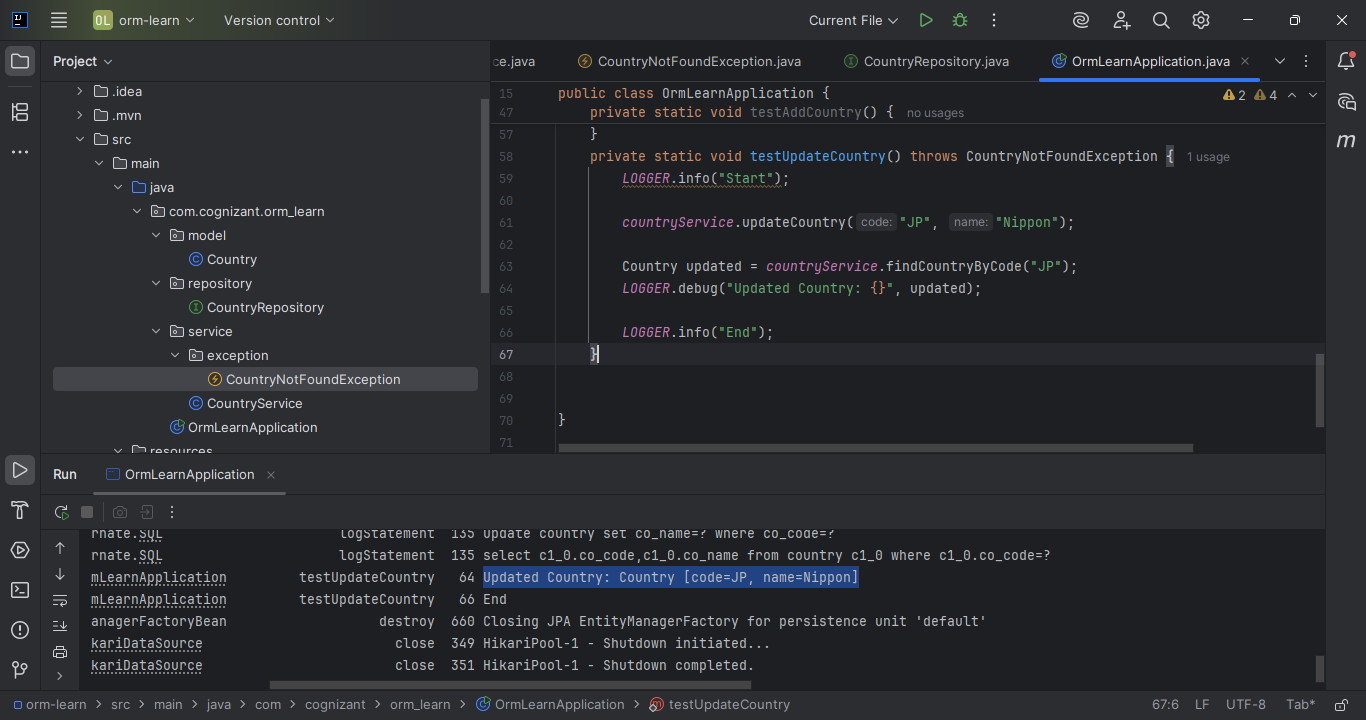
*countryService.updateCountry("JP", "Nippon");*

*Country updated = countryService.findCountryByCode("JP");*

*LOGGER.debug("Updated Country: {}", updated);*

*LOGGER.info("End");*

*}*

TEST OUTPUT: **

Test outputs are highlighted for clarity.

**Hands-On 9: Delete a Country based on Code**

Goal: Delete a country using its code.

Steps followed:

1. Service Method inside CountryService

*@Transactional*

*public void deleteCountry(String code) {*

*countryRepository.deleteById(code);*

*}*

1. Test Method used inside OrmLearnApplication.java

*private static void testDeleteCountry() {*

*LOGGER.info("Start");*

*countryService.deleteCountry("JP");*

*boolean exists = countryService.getAllCountries()*

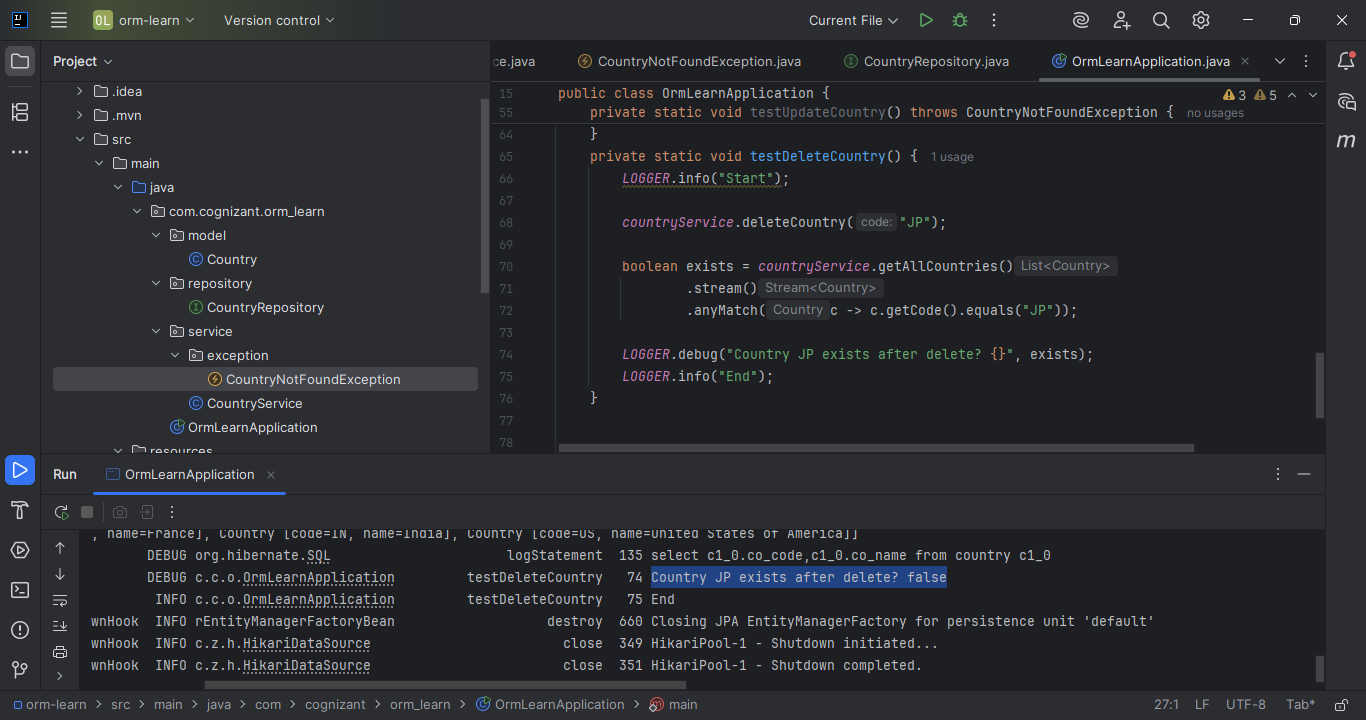
*.stream()*

*.anyMatch(c -> c.getCode().equals("JP"));*

*LOGGER.debug("Country JP exists after delete? {}", exists);*

*LOGGER.info("End");*

*}*

TEST OUTPUT: **

Test outputs are highlighted for clarity.

**

Also deleted from the MySQL table