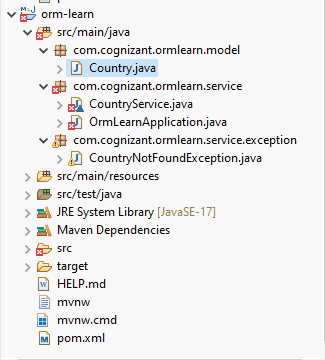
**Docx 1**

**Hands on 1: Spring Data JPA**



**Hands on 2: Hibernate XML Config implementation walk through**

**Explanation Topics**

**Explain how object to relational database mapping done in hibernate xml configuration file**

In Hibernate XML configuration, object-to-relational mapping is done by defining a .hbm.xml file for each entity class. This file declares how a Java class and its fields map to a specific database table and its columns.

**Explain about following aspects of implementing the end to end operations in Hibernate:**

**SessionFactory**

A heavyweight object created once per application. It is configured using hibernate.cfg.xml, and it creates Session objects to interact with the database.

**Session**

the main interface for CRUD operations. It represents a single unit of work and acts like a connection between the application and the database.

**Transaction**

handles commit and rollback logic. Hibernate requires explicit transactions for any DML operation even in non-transactional databases.

**beginTransaction()**

It begins a new database transaction:

Transaction tx = session.beginTransaction();

**commit()**

Commits the current transaction — persisting all changes to the database.

tx.commit();

**rollback()**

Reverts all changes made during the current transaction if an error occurs:

tx.rollback();

**session.save()**

Saves a new persistent object to the database:

session.save(employee);

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

Executes an HQL (Hibernate Query Language) query and returns results as a list:

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

**session.get()**

Fetches an object using its primary key:

Employee emp = session.get(Employee.class, 101);

**session.delete()**

Deletes a persistent object from the database:

session.delete(employee);

**Hands on 3: Hibernate Annotation Config implementation walk through**

**Explanation Topics:**

**Explain how object to relational database mapping done in persistence class file Employee**

In Hibernate, the entire concept is to map Java class attributes to relational database table columns. This is done by defining persistent classes, such as Employee, which follow the POJO (Plain Old Java Object) model.

A persistent class is a Java class whose instances are stored in a database table. Hibernate maps the fields of this class to the columns of the table either through XML mapping files or annotations.

The Employee class below is a POJO that follows key Hibernate mapping rules:

public class Employee {  
 private int id;  
 private String firstName;  
 private String lastName;  
 private int salary;  
  
 public Employee() {} // Required default constructor  
  
 public Employee(String fname, String lname, int salary) {  
 this.firstName = fname;  
 this.lastName = lname;  
 this.salary = salary;  
 }  
  
 // JavaBean-style getters and setters  
 public int getId() { return id; }  
 public void setId(int id) { this.id = id; }  
  
 public String getFirstName() { return firstName; }  
 public void setFirstName(String fname) { this.firstName = fname; }  
  
 public String getLastName() { return lastName; }  
 public void setLastName(String lname) { this.lastName = lname; }  
  
 public int getSalary() { return salary; }  
 public void setSalary(int salary) { this.salary = salary; }  
}

Hibernate uses this class to understand how to **persist objects to a table**. The field id typically maps to the **primary key**, and other fields map to corresponding **columns**.

**Explain about following aspects of implementing the end to end operations in Hibernate:**

**@Entity**

In Hibernate, an entity is a Java class annotated with @Entity that is mapped to a table in the database. It is a persistent object — a representation of a table row.

@Entity

@Table(name = "employee")

public class Employee {

@Id

private int id;

private String name;

private double salary;

}

**@Table**

The @Table annotation in Hibernate (JPA) is used to map a Java entity class to a specific table in the database.

It works together with the @Entity annotation.

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

}

**@Id**

Marks the primary key field of the entity.

**@GeneratedValue**

Tells Hibernate to auto-generate primary key values.

**@Column**

Maps the Java field to a specific database column.

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

**Dialect, Driver, Connection URL, Username, Password**

<hibernate-configuration>

<session-factory>

<property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>

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

...

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

</session-factory>

</hibernate-configuration>

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

|  |  |  |
| --- | --- | --- |
| **JPA** | **Hibernate** | **Spring Data JPA** |
| |  | | --- | |  |  |  | | --- | | Specification / Interface | | Implementation of JPA and also non-JPA | Abstraction layer over JPA & Hibernate |
| Standard way to map Java objects to database tables | Real working ORM that maps Java objects to DB tables | Reduce boilerplate code when using JPA |
| Use JPA to write standard DB code that works across providers | Use directly if you want full control and advanced features | Recommended in Spring Boot apps to quickly interact with the database |

**Hands on 5: Implement services for managing Country**

**Populate\_country**

DELETE FROM country;

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"); INSERT INTO country (co\_code, co\_name) VALUES ("JP", "Japan"); INSERT INTO country (co\_code, co\_name) VALUES ("BR", "Brazil");

**CountryService.java**

**package** com.cognizant.ormlearn.service;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** com.cognizant.ormlearn.model.Country;

**import** com.cognizant.ormlearn.repository.CountryRepository;

**import** jakarta.transaction.Transactional;

@Service

**public** **class** CountryService {

@Autowired

**private** CountryRepository countryRepository;

// 1. get countries

@Transactional

**public** List<Country> getAllCountries() {

**return** countryRepository.findAll();

}

@Transactional

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

**return** countryRepository.findById(code)

.orElseThrow(() -> **new** RuntimeException("Country not found"));

}

// 2. Add New Country

@Transactional

**public** **void** addCountry(Country country) {

countryRepository.save(country);

}

// 3. Update Country

@Transactional

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

Country country = findCountryByCode(code);

country.setName(name);

countryRepository.save(country);

}

// 4. Delete Country

@Transactional

**public** **void** deleteCountry(String code) {

countryRepository.deleteById(code);

}

// 6. Search by Partial Name

@Transactional

**public** List<Country> searchCountries(String partialName) {

**return** countryRepository.findByNameContainingIgnoreCase(partialName);

}

}

**CountryRepository.java**

**package** com.cognizant.ormlearn.repository;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** org.springframework.stereotype.Repository;

**import** com.cognizant.ormlearn.model.Country;

**import** java.util.List;

@Repository

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

List<Country> findByNameContainingIgnoreCase(String namePart);

}

**OrmLearnApplication.java**

**package** com.cognizant.ormlearn.service;

**import** com.cognizant.ormlearn.service.CountryService;

**import** com.cognizant.ormlearn.model.Country;

**import** com.cognizant.ormlearn.service.exception.CountryNotFoundException;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.context.ApplicationContext;

@SpringBootApplication

**public** **class** OrmLearnApplication {

**private** **static** CountryService *countryService*;

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(OrmLearnApplication.**class**);

**public** **static** **void** main(String[] args) **throws** CountryNotFoundException {

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

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

*testFindCountryByCode*();

}

**private** **static** **void** testFindCountryByCode() **throws** CountryNotFoundException {

***LOGGER***.info("Start");

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

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

***LOGGER***.info("End");

}

**private** **static** **void** testCountryService() {

***LOGGER***.info("Start");

// 1. Add new country

Country newCountry = **new** Country();

newCountry.setCode("AU");

newCountry.setName("Australia");

*countryService*.addCountry(newCountry);

// 2. Update country

*countryService*.updateCountry("AU", "Australia Updated");

// 3. Find by code

**try** {

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

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

} **catch** (CountryNotFoundException e) {

***LOGGER***.error("Error finding country: ", e);

}

// 4. Search by partial name

*countryService*.searchCountries("Uni").forEach(c -> ***LOGGER***.debug("Match: {}", c));

// 5. Delete country

*countryService*.deleteCountry("AU");

***LOGGER***.info("End");

}

}

**Hands on 6: Find a country based on country code**

**CountryService.java**

**package** com.cognizant.ormlearn.service;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** com.cognizant.ormlearn.model.Country;

**import** com.cognizant.ormlearn.repository.CountryRepository;

**import** com.cognizant.ormlearn.service.exception.CountryNotFoundException;

**import** jakarta.transaction.Transactional;

@Service

**public** **class** CountryService {

@Autowired

**private** CountryRepository countryRepository;

// 1. Get all countries

@Transactional

**public** List<Country> getAllCountries() {

**return** countryRepository.findAll();

}

// 2. Find country by code

@Transactional

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

**return** countryRepository.findById(code)

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

}

// 3. Add new country

@Transactional

**public** **void** addCountry(Country country) {

countryRepository.save(country);

}

// 4. Update country

@Transactional

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

Country country = findCountryByCode(code);

country.setName(name);

countryRepository.save(country);

}

// 5. Delete country

@Transactional

**public** **void** deleteCountry(String code) {

countryRepository.deleteById(code);

}

// 6. Search countries by partial name

@Transactional

**public** List<Country> searchCountries(String partialName) {

**return** countryRepository.findByNameContainingIgnoreCase(partialName);

}

}

**OrmLearnApplication.java**

**package** com.cognizant.ormlearn;

**import** com.cognizant.ormlearn.model.Country;

**import** com.cognizant.ormlearn.service.CountryService;

**import** com.cognizant.ormlearn.service.exception.CountryNotFoundException;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.context.ApplicationContext;

@SpringBootApplication

**public** **class** OrmLearnApplication {

**private** **static** CountryService *countryService*;

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(OrmLearnApplication.**class**);

**public** **static** **void** main(String[] args) {

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

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

// New test method

*testAddCountry*();

*testFindCountryByCode*();

// testCountryService(); // Uncomment to test full CRUD

}

**private** **static** **void** testFindCountryByCode() {

***LOGGER***.info("Start");

**try** {

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

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

} **catch** (CountryNotFoundException e) {

***LOGGER***.error("Exception: {}", e.getMessage());

}

***LOGGER***.info("End");

}

@SuppressWarnings("unused")

**private** **static** **void** testCountryService() {

***LOGGER***.info("Start");

**try** {

Country newCountry = **new** Country();

newCountry.setCode("AU");

newCountry.setName("Australia");

*countryService*.addCountry(newCountry);

*countryService*.updateCountry("AU", "Australia Updated");

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

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

*countryService*.searchCountries("Uni").forEach(c -> ***LOGGER***.debug("Match: {}", c));

*countryService*.deleteCountry("AU");

} **catch** (CountryNotFoundException e) {

***LOGGER***.error("Error: ", e);

}

***LOGGER***.info("End");

}

@SuppressWarnings("unused")

**private** **static** **void** testAddCountry() {

***LOGGER***.info("Start");

}

}

**Find\_country.java**

DEBUG Country: Country{code='IN', name='India'}

**Hands on 7: Add a new country**

**CountryService.java**

**package** com.cognizant.ormlearn.service;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** com.cognizant.ormlearn.model.Country;

**import** com.cognizant.ormlearn.repository.CountryRepository;

**import** com.cognizant.ormlearn.service.exception.CountryNotFoundException;

**import** jakarta.transaction.Transactional;

@Service

**public** **class** CountryService {

@Autowired

**private** CountryRepository countryRepository;

// 1. Get all countries

@Transactional

**public** List<Country> getAllCountries() {

**return** countryRepository.findAll();

}

// 2. Find country by code

@Transactional

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

**return** countryRepository.findById(code)

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

}

// 3. Add new country

@Transactional

**public** **void** addCountry(Country country) {

countryRepository.save(country);

}

// 4. Update country

@Transactional

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

Country country = findCountryByCode(code);

country.setName(name);

countryRepository.save(country);

}

// 5. Delete country

@Transactional

**public** **void** deleteCountry(String code) {

countryRepository.deleteById(code);

}

// 6. Search countries by partial name

@Transactional

**public** List<Country> searchCountries(String partialName) {

**return** countryRepository.findByNameContainingIgnoreCase(partialName);

}

}

**OrmLearnApplication.java**

**package** com.cognizant.ormlearn;

**import** com.cognizant.ormlearn.model.Country;

**import** com.cognizant.ormlearn.service.CountryService;

**import** com.cognizant.ormlearn.service.exception.CountryNotFoundException;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.context.ApplicationContext;

@SpringBootApplication

**public** **class** OrmLearnApplication {

**private** **static** CountryService *countryService*;

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(OrmLearnApplication.**class**);

**public** **static** **void** main(String[] args) {

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

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

//testGetAllCountries();

//getAllCountriesTest();

// New test method

*testAddCountry*();

*testFindCountryByCode*();

// testCountryService(); // Uncomment to test full CRUD

}

**private** **static** **void** testFindCountryByCode() {

***LOGGER***.info("Start");

**try** {

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

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

} **catch** (CountryNotFoundException e) {

***LOGGER***.error("Exception: {}", e.getMessage());

}

***LOGGER***.info("End");

}

@SuppressWarnings("unused")

**private** **static** **void** testCountryService() {

***LOGGER***.info("Start");

**try** {

Country newCountry = **new** Country();

newCountry.setCode("AU");

newCountry.setName("Australia");

*countryService*.addCountry(newCountry);

*countryService*.updateCountry("AU", "Australia Updated");

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

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

*countryService*.searchCountries("Uni").forEach(c -> ***LOGGER***.debug("Match: {}", c));

*countryService*.deleteCountry("AU");

} **catch** (CountryNotFoundException e) {

***LOGGER***.error("Error: ", e);

}

***LOGGER***.info("End");

}

@SuppressWarnings("unused")

**private** **static** **void** testAddCountry() {

***LOGGER***.info("Start");

// 1. Create new country object

Country country = **new** Country();

country.setCode("XY");

country.setName("Xyland");

// 2. Add to database

*countryService*.addCountry(country);

// 3. Retrieve the newly added country

**try** {

Country addedCountry = *countryService*.findCountryByCode("XY");

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

} **catch** (CountryNotFoundException e) {

***LOGGER***.error("Country not found", e);

}

***LOGGER***.info("End");

}

}

**Find\_country**

SELECT \* FROM country WHERE co\_code = 'XY';

**Hands on 8: Update a country based on code**

**CountryService.java**

**package** com.cognizant.ormlearn.service;

**import** java.util.List;

**import** java.util.Optional;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** com.cognizant.ormlearn.model.Country;

**import** com.cognizant.ormlearn.repository.CountryRepository;

**import** com.cognizant.ormlearn.service.exception.CountryNotFoundException;

**import** jakarta.transaction.Transactional;

@Service

**public** **class** CountryService {

@Autowired

**private** CountryRepository countryRepository;

// 1. Get all countries

@Transactional

**public** List<Country> getAllCountries() {

**return** countryRepository.findAll();

}

// 2. Find country by code

@Transactional

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

**return** countryRepository.findById(code)

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

}

// 3. Add new country

@Transactional

**public** **void** addCountry(Country country) {

countryRepository.save(country);

}

// 4. Update country

@Transactional

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

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

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

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

}

Country country = result.get();

country.setName(name); // Update the name

countryRepository.save(country); // Persist updated country

}

// 5. Delete country

@Transactional

**public** **void** deleteCountry(String code) {

countryRepository.deleteById(code);

}

// 6. Search countries by partial name

@Transactional

**public** List<Country> searchCountries(String partialName) {

**return** countryRepository.findByNameContainingIgnoreCase(partialName);

}

}

**OrmLearnApplication.java**

**package** com.cognizant.ormlearn;

**import** com.cognizant.ormlearn.model.Country;

**import** com.cognizant.ormlearn.service.CountryService;

**import** com.cognizant.ormlearn.service.exception.CountryNotFoundException;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.context.ApplicationContext;

@SpringBootApplication

**public** **class** OrmLearnApplication {

**private** **static** CountryService *countryService*;

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(OrmLearnApplication.**class**);

**public** **static** **void** main(String[] args) {

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

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

//testGetAllCountries();

//getAllCountriesTest();

// New test method

*testAddCountry*();

*testFindCountryByCode*();

*testCountryService*(); // Uncomment to test full CRUD

*testUpdateCountry*();

}

**private** **static** **void** testFindCountryByCode() {

***LOGGER***.info("Start");

**try** {

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

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

} **catch** (CountryNotFoundException e) {

***LOGGER***.error("Exception: {}", e.getMessage());

}

***LOGGER***.info("End");

}

@SuppressWarnings("unused")

**private** **static** **void** testCountryService() {

***LOGGER***.info("Start");

**try** {

Country newCountry = **new** Country();

newCountry.setCode("AU");

newCountry.setName("Australia");

*countryService*.addCountry(newCountry);

*countryService*.updateCountry("AU", "Australia Updated");

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

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

*countryService*.searchCountries("Uni").forEach(c -> ***LOGGER***.debug("Match: {}", c));

*countryService*.deleteCountry("AU");

} **catch** (CountryNotFoundException e) {

***LOGGER***.error("Error: ", e);

}

***LOGGER***.info("End");

}

@SuppressWarnings("unused")

**private** **static** **void** testAddCountry() {

***LOGGER***.info("Start");

// 1. Create new country object

Country country = **new** Country();

country.setCode("XY");

country.setName("Xyland");

// 2. Add to database

*countryService*.addCountry(country);

// 3. Retrieve the newly added country

**try** {

Country addedCountry = *countryService*.findCountryByCode("XY");

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

} **catch** (CountryNotFoundException e) {

***LOGGER***.error("Country not found", e);

}

***LOGGER***.info("End");

}

@SuppressWarnings("unused")

**private** **static** **void** testUpdateCountry() {

***LOGGER***.info("Start");

**try** {

// Update existing country

*countryService*.updateCountry("XY", "Xyland Updated");

// Verify update

Country updatedCountry = *countryService*.findCountryByCode("XY");

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

} **catch** (CountryNotFoundException e) {

***LOGGER***.error("Country not found", e);

}

***LOGGER***.info("End");

}

}

**Sql code**

USE ormlearn;

CREATE TABLE country (

co\_code VARCHAR(2) PRIMARY KEY,

co\_name VARCHAR(50)

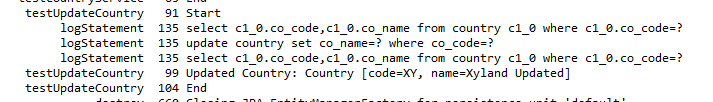
);

INSERT INTO country VALUES ('IN', 'India');

INSERT INTO country VALUES ('XY', 'Xyland');

SELECT \* FROM country WHERE co\_code = 'XY';

**Output:**



**Hands on 9: Delete a country based on code**

**CountryService.java**

**package** com.cognizant.ormlearn.service;

**import** java.util.List;

**import** java.util.Optional;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** com.cognizant.ormlearn.model.Country;

**import** com.cognizant.ormlearn.repository.CountryRepository;

**import** com.cognizant.ormlearn.service.exception.CountryNotFoundException;

**import** jakarta.transaction.Transactional;

@Service

**public** **class** CountryService {

@Autowired

**private** CountryRepository countryRepository;

// 1. Get all countries

@Transactional

**public** List<Country> getAllCountries() {

**return** countryRepository.findAll();

}

// 2. Find country by code

@Transactional

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

**return** countryRepository.findById(code)

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

}

// 3. Add new country

@Transactional

**public** **void** addCountry(Country country) {

countryRepository.save(country);

}

// 4. Update country

@Transactional

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

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

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

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

}

Country country = result.get();

country.setName(name); // Update the name

countryRepository.save(country); // Persist updated country

}

// 6. Search countries by partial name

@Transactional

**public** List<Country> searchCountries(String partialName) {

**return** countryRepository.findByNameContainingIgnoreCase(partialName);

}

@Transactional

**public** **void** deleteCountry(String code) {

countryRepository.deleteById(code);

}

}

**OrmLearnApplication.java**

**package** com.cognizant.ormlearn.service;

**import** java.util.List;

**import** java.util.Optional;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** com.cognizant.ormlearn.model.Country;

**import** com.cognizant.ormlearn.repository.CountryRepository;

**import** com.cognizant.ormlearn.service.exception.CountryNotFoundException;

**import** jakarta.transaction.Transactional;

@Service

**public** **class** CountryService {

@Autowired

**private** CountryRepository countryRepository;

// 1. Get all countries

@Transactional

**public** List<Country> getAllCountries() {

**return** countryRepository.findAll();

}

// 2. Find country by code

@Transactional

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

**return** countryRepository.findById(code)

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

}

// 3. Add new country

@Transactional

**public** **void** addCountry(Country country) {

countryRepository.save(country);

}

// 4. Update country

@Transactional

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

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

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

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

}

Country country = result.get();

country.setName(name); // Update the name

countryRepository.save(country); // Persist updated country

}

// 6. Search countries by partial name

@Transactional

**public** List<Country> searchCountries(String partialName) {

**return** countryRepository.findByNameContainingIgnoreCase(partialName);

}

@Transactional

**public** **void** deleteCountry(String code) {

countryRepository.deleteById(code);

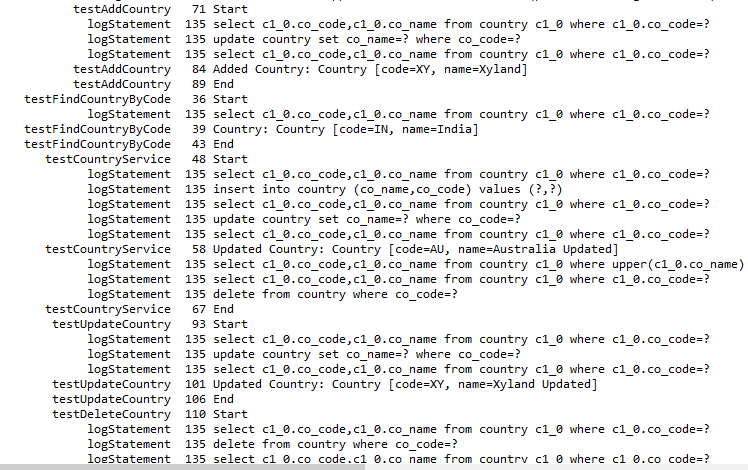
}

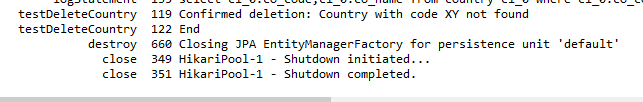
}

**Sql code**

SELECT \* FROM country WHERE co\_code = 'XY';

**Output:**





**Docx 2**

**Hands on 1: Write queries on country table using Query Methods**

**CountryService.java**

@Transactional

**public** List<Country> searchCountries(String partialName) {

**return** countryRepository.findByNameContainingIgnoreCase(partialName);

}

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

**return** countryRepository.findByNameContainingOrderByNameAsc(keyword);

}

**public** List<Country> searchCountriesStartingWith(String prefix) {

**return** countryRepository.findByNameStartingWith(prefix);

}

**OrmLearnApplication.java**

*testSearchCountries*();

@SuppressWarnings("unused")

**private** **static** **void** testSearchCountries() {

***LOGGER***.info("Start");

// 1. Contains "ou"

List<Country> result = *countryService*.searchCountries("ou");

result.forEach(c -> ***LOGGER***.debug("Contains ou: {}", c));

// 2. Contains "ou" sorted

List<Country> sortedResult = *countryService*.searchCountriesSorted("ou");

sortedResult.forEach(c -> ***LOGGER***.debug("Sorted: {}", c));

// 3. Starts with "Z"

List<Country> startsWithZ = *countryService*.searchCountriesStartingWith("Z");

startsWithZ.forEach(c -> ***LOGGER***.debug("StartsWith Z: {}", c));

***LOGGER***.info("End");

}

**Sql code**

INSERT INTO country (co\_code, co\_name) VALUES

('BV', 'Bouvet Island'),

('DJ', 'Djibouti'),

('GP', 'Guadeloupe'),

('GS', 'South Georgia and the South Sandwich Islands'),

('LU', 'Luxembourg'),

('SS', 'South Sudan'),

('TF', 'French Southern Territories'),

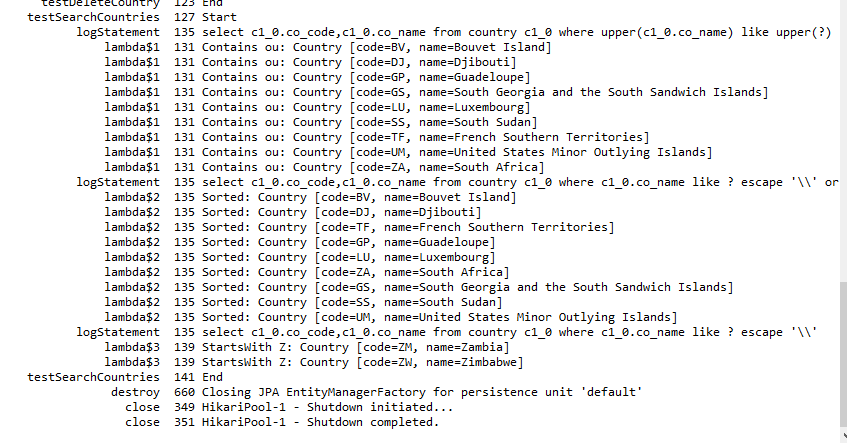
('UM', 'United States Minor Outlying Islands'),

('ZA', 'South Africa'),

('ZM', 'Zambia'),

('ZW', 'Zimbabwe');

**Output**



**Hands on 2: Write queries on stock table using Query Methods**

**Sql code**

CREATE TABLE IF NOT EXISTS stock

( st\_id INT NOT NULL AUTO\_INCREMENT,

st\_code VARCHAR(10),

st\_date DATE, st\_open DECIMAL(10,2),

st\_close DECIMAL(10,2),

st\_volume BIGINT,

PRIMARY KEY (st\_id) );

**Stock.java**

**package** com.cognizant.ormlearn.model;

**import** jakarta.persistence.\*;

**import** java.math.BigDecimal;

**import** java.sql.Date;

@Entity

@Table(name = "stock")

**public** **class** Stock {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "st\_id")

**private** **int** id;

@Column(name = "st\_code")

**private** String code;

@Column(name = "st\_date")

**private** Date date;

@Column(name = "st\_open")

**private** BigDecimal open;

@Column(name = "st\_close")

**private** BigDecimal close;

@Column(name = "st\_volume")

**private** Long volume;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getCode() {

**return** code;

}

**public** **void** setCode(String code) {

**this**.code = code;

}

**public** Date getDate() {

**return** date;

}

**public** **void** setDate(Date date) {

**this**.date = date;

}

**public** BigDecimal getOpen() {

**return** open;

}

**public** **void** setOpen(BigDecimal open) {

**this**.open = open;

}

**public** BigDecimal getClose() {

**return** close;

}

**public** **void** setClose(BigDecimal close) {

**this**.close = close;

}

**public** Long getVolume() {

**return** volume;

}

**public** **void** setVolume(Long volume) {

**this**.volume = volume;

}

@Override

**public** String toString() {

**return** "Stock [id=" + id + ", code=" + code + ", date=" + date +

", open=" + open + ", close=" + close + ", volume=" + volume + "]";

}

}

**StockRepository.java**

**package** com.cognizant.ormlearn.repository;

**import** com.cognizant.ormlearn.model.Stock;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** java.math.BigDecimal;

**import** java.sql.Date;

**import** java.util.List;

**public** **interface** StockRepository **extends** JpaRepository<Stock, Integer> {

// 1.stock details of Facebook in Sep 2019

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

// 2. Get all Google stock where close > 1250

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

// 3. Get top 3 days with high vol

List<Stock> findTop3ByOrderByVolumeDesc();

// 4. Get top 3 Netflix stocks with low closing price

List<Stock> findTop3ByCodeOrderByCloseAsc(String code);

}

**OrmLearnApplication.java**

**private** **static** StockRepository *stockRepository*;

*stockRepository* = context.getBean(StockRepository.**class**);

*testStockQueries*();

@SuppressWarnings("unused")

**private** **static** **void** testStockQueries() {

***LOGGER***.info("Start testStockQueries");

// 1. Facebook stock in Sep 2019

Date start = Date.*valueOf*("2019-09-01");

Date end = Date.*valueOf*("2019-09-30");

*stockRepository*.findByCodeAndDateBetween("FB", start, end)

.forEach(stock -> ***LOGGER***.debug("FB Sept 2019: {}", stock));

// 2. Google stock price > 1250

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

.forEach(stock -> ***LOGGER***.debug("GOOGL > 1250: {}", stock));

// 3. Top 3 volume days

*stockRepository*.findTop3ByOrderByVolumeDesc()

.forEach(stock -> ***LOGGER***.debug("Top Volume: {}", stock));

// 4. Netflix lowest 3 close prices

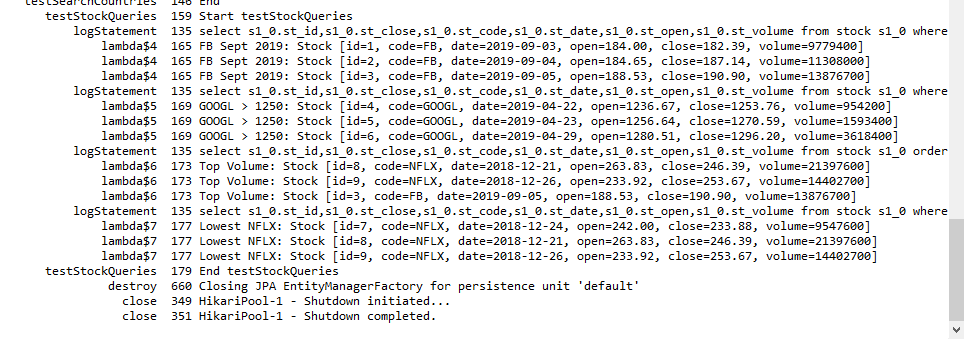
*stockRepository*.findTop3ByCodeOrderByCloseAsc("NFLX")

.forEach(stock -> ***LOGGER***.debug("Lowest NFLX: {}", stock));

***LOGGER***.info("End testStockQueries");

}

**Output**



**Hands on 3: Create payroll tables and bean mapping**

**Sql code**

CREATE TABLE department (

dp\_id INT PRIMARY KEY,

dp\_name VARCHAR(50)

);

CREATE TABLE employee (

em\_id INT PRIMARY KEY,

em\_name VARCHAR(30),

em\_salary DECIMAL(10,2),

em\_permanent BOOLEAN,

em\_date\_of\_birth DATE,

em\_dp\_id INT,

FOREIGN KEY (em\_dp\_id) REFERENCES department(dp\_id) );

CREATE TABLE skill ( sk\_id INT PRIMARY KEY, sk\_name VARCHAR(30)

);

CREATE TABLE employee\_skill ( es\_em\_id INT, es\_sk\_id INT, PRIMARY KEY (es\_em\_id, es\_sk\_id), FOREIGN KEY (es\_em\_id) REFERENCES employee(em\_id),

FOREIGN KEY (es\_sk\_id) REFERENCES skill(sk\_id) );

Department.java

**package** com.cognizant.ormlearn.model;

**import** jakarta.persistence.\*;

@Entity

@Table(name = "department")

**public** **class** Department {

@Id

@Column(name = "dp\_id")

**private** **int** id;

@Column(name = "dp\_name")

**private** String name;

**public** **int** getId() { **return** id; }

**public** **void** setId(**int** id) { **this**.id = id; }

**public** String getName() { **return** name; }

**public** **void** setName(String name) { **this**.name = name; }

@Override

**public** String toString() {

**return** "Department [id=" + id + ", name=" + name + "]";

}

}

**Employee.java**

**package** com.cognizant.ormlearn.model;

**import** jakarta.persistence.\*;

**import** java.math.BigDecimal;

**import** java.sql.Date;

@Entity

@Table(name = "employee")

**public** **class** Employee {

@Id

@Column(name = "em\_id")

**private** **int** id;

@Column(name = "em\_name")

**private** String name;

@Column(name = "em\_salary")

**private** BigDecimal salary;

@Column(name = "em\_permanent")

**private** **boolean** permanent;

@Column(name = "em\_date\_of\_birth")

**private** Date dateOfBirth;

**public** **int** getId() { **return** id; }

**public** **void** setId(**int** id) { **this**.id = id; }

**public** String getName() { **return** name; }

**public** **void** setName(String name) { **this**.name = name; }

**public** BigDecimal getSalary() { **return** salary; }

**public** **void** setSalary(BigDecimal salary) { **this**.salary = salary; }

**public** **boolean** isPermanent() { **return** permanent; }

**public** **void** setPermanent(**boolean** permanent) { **this**.permanent = permanent; }

**public** Date getDateOfBirth() { **return** dateOfBirth; }

**public** **void** setDateOfBirth(Date dateOfBirth) { **this**.dateOfBirth = dateOfBirth; }

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", name=" + name + ", salary=" + salary +

", permanent=" + permanent + ", dateOfBirth=" + dateOfBirth + "]";

}

}

**Skill.java**

**package** com.cognizant.ormlearn.model;

**import** jakarta.persistence.\*;

@Entity

@Table(name = "skill")

**public** **class** Skill {

@Id

@Column(name = "sk\_id")

**private** **int** id;

@Column(name = "sk\_name")

**private** String name;

**public** **int** getId() { **return** id; }

**public** **void** setId(**int** id) { **this**.id = id; }

**public** String getName() { **return** name; }

**public** **void** setName(String name) { **this**.name = name; }

@Override

**public** String toString() {

**return** "Skill [id=" + id + ", name=" + name + "]";

}

}

**Hands on 4: Implement many to one relationship between Employee and Department**

**Sql code**

INSERT INTO department VALUES (1, 'HR'), (2, 'IT');

INSERT INTO employee (em\_id, em\_name, em\_salary, em\_permanent, em\_date\_of\_birth, em\_dp\_id) VALUES

(1, 'John', 50000.00, true, '1990-01-01', 1),

(2, 'Jane', 70000.00, false, '1992-06-15', 2);

**EmployeeRepository.java**

**package** com.cognizant.ormlearn.repository;

**import** com.cognizant.ormlearn.model.Employee;

**import** org.springframework.data.jpa.repository.JpaRepository;

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

}

**DepartmentRepository.java**

**package** com.cognizant.ormlearn.repository;

**import** com.cognizant.ormlearn.model.Department;

**import** org.springframework.data.jpa.repository.JpaRepository;

**public** **interface** DepartmentRepository **extends** JpaRepository<Department, Integer> {

}

**OrmLearnApplication.java**

@SuppressWarnings("unused")

**private** **static** EmployeeRepository *employeeRepository*;

**Hands on 5: Implement one to many relationship between Employee and Department**

**Department.java**

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

**private** List<Employee> employeeList;

@SuppressWarnings("unchecked")

**public** List<Employee> getEmployeeList() {

**return** (List<Employee>) employeeList;

}

**public** **void** setEmployeeList(List<Employee> employeeList) {

**this**.employeeList = employeeList;

}

**Employee.java**

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", name=" + name + ", salary=" + salary +

", permanent=" + permanent + ", dateOfBirth=" + dateOfBirth +

", department=" + (department != **null** ? department.getName() : "null") + "]";

}

**OrmLearnApplication.java**

*departmentRepository* = context.getBean(DepartmentRepository.**class**);

*testGetDepartment*();

@SuppressWarnings("unused")

**private** **static** DepartmentRepository *departmentRepository*;

**private** **static** **void** testGetDepartment() {

***LOGGER***.info("Start");

Department department = *departmentRepository*.findById(1).get();

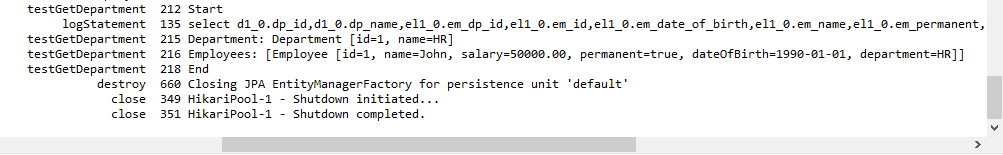
***LOGGER***.debug("Department: {}", department);

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

***LOGGER***.info("End");

}

**Output**



**Hands on 6: Implement many to many relationship between Employee and Skill**

**EmployeeSkill.java**

**package** com.cognizant.ormlearn.model;

**import** jakarta.persistence.\*;

**import** java.time.LocalDate;

@Entity

@Table(name = "employee\_skill\_map")

**public** **class** EmployeeSkill {

@EmbeddedId

**private** EmployeeSkillId id = **new** EmployeeSkillId();

@ManyToOne

@MapsId("employeeId")

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

**private** Employee employee;

@ManyToOne

@MapsId("skillId")

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

**private** Skill skill;

**private** **int** proficiency;

**private** LocalDate endorsedOn;

**public** EmployeeSkillId getId() { **return** id; }

**public** **void** setId(EmployeeSkillId id) { **this**.id = id; }

**public** Employee getEmployee() { **return** employee; }

**public** **void** setEmployee(Employee employee) {

**this**.employee = employee;

**this**.id.setEmployeeId(employee.getId());

}

**public** Skill getSkill() { **return** skill; }

**public** **void** setSkill(Skill skill) {

**this**.skill = skill;

**this**.id.setSkillId(skill.getId());

}

**public** **int** getProficiency() { **return** proficiency; }

**public** **void** setProficiency(**int** proficiency) { **this**.proficiency = proficiency; }

**public** LocalDate getEndorsedOn() { **return** endorsedOn; }

**public** **void** setEndorsedOn(LocalDate endorsedOn) { **this**.endorsedOn = endorsedOn; }

}

**EmployeeSkill.java**

**package** com.cognizant.ormlearn.model;

**import** jakarta.persistence.\*;

**import** java.time.LocalDate;

@Entity

@Table(name = "employee\_skill\_map")

**public** **class** EmployeeSkill {

@EmbeddedId

**private** EmployeeSkillId id = **new** EmployeeSkillId();

@ManyToOne

@MapsId("employeeId")

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

**private** Employee employee;

@ManyToOne

@MapsId("skillId")

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

**private** Skill skill;

**private** **int** proficiency;

**private** LocalDate endorsedOn;

**public** EmployeeSkillId getId() { **return** id; }

**public** **void** setId(EmployeeSkillId id) { **this**.id = id; }

**public** Employee getEmployee() { **return** employee; }

**public** **void** setEmployee(Employee employee) {

**this**.employee = employee;

**this**.id.setEmployeeId(employee.getId());

}

**public** Skill getSkill() { **return** skill; }

**public** **void** setSkill(Skill skill) {

**this**.skill = skill;

**this**.id.setSkillId(skill.getId());

}

**public** **int** getProficiency() { **return** proficiency; }

**public** **void** setProficiency(**int** proficiency) { **this**.proficiency = proficiency; }

**public** LocalDate getEndorsedOn() { **return** endorsedOn; }

**public** **void** setEndorsedOn(LocalDate endorsedOn) { **this**.endorsedOn = endorsedOn; }

}

**EmployeeSkillService.java**

**package** com.cognizant.ormlearn.service;

**import** com.cognizant.ormlearn.model.\*;

**import** com.cognizant.ormlearn.repository.\*;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** org.springframework.transaction.annotation.Transactional;

**import** java.time.LocalDate;

**import** java.util.List;

@Service

**public** **class** EmployeeSkillService {

@Autowired

**private** EmployeeRepository employeeRepository;

@Autowired

**private** SkillRepository skillRepository;

@Autowired

**private** EmployeeSkillRepository employeeSkillRepository;

@Transactional

**public** **void** assignSkillWithProficiency(**int** empId, **int** skillId, **int** proficiency) {

Employee employee = employeeRepository.findById(empId).orElseThrow();

Skill skill = skillRepository.findById(skillId).orElseThrow();

EmployeeSkill es = **new** EmployeeSkill();

es.setEmployee(employee);

es.setSkill(skill);

es.setProficiency(proficiency);

es.setEndorsedOn(LocalDate.*now*());

employeeSkillRepository.save(es);

}

**public** List<EmployeeSkill> getSkillsForEmployee(**int** empId) {

**return** employeeSkillRepository.findByEmployeeId(empId);

}

}

**OrmLearnApplication.java**

employeeSkillService.assignSkillToEmployee(employeeId, skillId);

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

employeeSkillService = context.getBean(EmployeeSkillService.**class**);

*testAssignSkillWithProficiency*();

**private** **static** **void** testAddSkillToEmployee() {

***LOGGER***.info("Start");

Employee employee = employeeService.get(1);

Skill skill = skillService.get(3);

Set<Skill> skills = employee.getSkillList();

skills.add(skill);

employee.setSkillList(skills);

employeeService.save(employee);

***LOGGER***.debug("Updated Employee with Skills: {}", employee.getSkillList());

***LOGGER***.info("End");

}

**private** **static** **void** testAssignSkillWithProficiency() {

***LOGGER***.info("Start");

employeeSkillService.assignSkillWithProficiency(10, 2, 8);

***LOGGER***.info("End");

}

**private** **static** **void** testGetSkillsForEmployee() {

***LOGGER***.info("Skills for Employee 10:");

employeeSkillService.getSkillsForEmployee(10).forEach(es ->

LOGGER.info("Skill: {}, Proficiency: {}, Endorsed: {}",

es.getSkill().getName(), es.getProficiency(), es.getEndorsedOn())

);

}

**Sql code**

CREATE TABLE employee\_skill\_map (

es\_em\_id INT, es\_sk\_id INT,

proficiency INT, endorsed\_on DATE,

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

FOREIGN KEY (es\_em\_id) REFERENCES employee(em\_id), FOREIGN KEY (es\_sk\_id) REFERENCES skill(sk\_id) );

INSERT INTO employee (em\_id, em\_name, em\_salary, em\_permanent, em\_date\_of\_birth, em\_dp\_id) VALUES

(10, 'Amit', 65000, true, '1993-04-15', 1);

SELECT \* FROM employee\_skill\_map;

**Docs 3**

**Hands on 1: Introduction to HQL and JPQL**

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

**EmployeeService.java**

**package** com.cognizant.ormlearn.service;

**import** com.cognizant.ormlearn.model.Employee;

**import** com.cognizant.ormlearn.repository.EmployeeRepository;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** java.util.List;

@Service

**public** **class** EmployeeService {

@Autowired

**private** EmployeeRepository employeeRepository;

**public** List<Employee> getAllPermanentEmployees() {

**return** employeeRepository.getAllPermanentEmployees();

}

}

**EmployeeRepository.java**

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

List<Employee> getAllPermanentEmployees();

**OrmLearnApplication.java**

*employeeService* = context1.getBean(EmployeeService.**class**);

*testGetAllPermanentEmployees*();

@Autowired

**private** **static** EmployeeService *employeeService*;

**private** **static** **void** testGetAllPermanentEmployees() {

***LOGGER***.info("Start");

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

**for** (Employee e : employees) {

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

***LOGGER***.debug("Department: {}", e.getDepartment().getName());

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

.map(skill -> skill.getName())

.toList());

}

***LOGGER***.info("End");

}

**Hands on 3**

**Fetch quiz attempt details using HQL**

**Sql code**

-- USER TABLE

CREATE TABLE user (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL );

-- QUESTION TABLE

CREATE TABLE question (

id INT PRIMARY KEY AUTO\_INCREMENT,

text VARCHAR(255) NOT NULL );

CREATE TABLE options (

id INT PRIMARY KEY AUTO\_INCREMENT,

question\_id INT,

text VARCHAR(255) NOT NULL,

score DECIMAL(3,1) DEFAULT 0.0,

is\_correct BOOLEAN DEFAULT FALSE,

FOREIGN KEY (question\_id) REFERENCES question(id) );

-- ATTEMPT TABLE

CREATE TABLE attempt (

id INT PRIMARY KEY AUTO\_INCREMENT,

user\_id INT,

attempted\_date DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES user(id) );

-- ATTEMPT\_QUESTION TABLE

CREATE TABLE attempt\_question (

id INT PRIMARY KEY AUTO\_INCREMENT,

attempt\_id INT,

question\_id INT,

FOREIGN KEY (attempt\_id) REFERENCES attempt(id),

FOREIGN KEY (question\_id) REFERENCES question(id) );

CREATE TABLE attempt\_option (

id INT PRIMARY KEY AUTO\_INCREMENT,

attempt\_question\_id INT,

option\_id INT,

is\_selected BOOLEAN DEFAULT FALSE,

FOREIGN KEY (attempt\_question\_id) REFERENCES attempt\_question(id), FOREIGN KEY (option\_id) REFERENCES options(id) );

**User.java**

**package** com.cognizant.ormlearn.model;

**import** java.util.List;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.GeneratedValue;

**import** jakarta.persistence.GenerationType;

**import** jakarta.persistence.Id;

**import** jakarta.persistence.OneToMany;

**import** jakarta.persistence.Table;

@Entity

@Table(name = "user")

**public** **class** User {

@Id @GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

@SuppressWarnings("unused")

**private** String name;

@OneToMany(mappedBy = "user")

**private** List<Attempt> attempts;

}

**Question.java**

**package** com.cognizant.ormlearn.model;

**import** com.cognizant.ormlearn.model.Option;

**import** java.util.List;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.GeneratedValue;

**import** jakarta.persistence.GenerationType;

**import** jakarta.persistence.Id;

**import** jakarta.persistence.OneToMany;

**import** jakarta.persistence.Table;

@SuppressWarnings("unused")

@Entity

@Table(name = "question")

**public** **class** Question {

@Id @GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

**private** String text;

@OneToMany(mappedBy = "question")

**private** List<Option> options;

**public** String getText() {

**return** text;

}

}

**Option.java**

**package** com.cognizant.ormlearn.model;

**import** java.math.BigDecimal;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.GeneratedValue;

**import** jakarta.persistence.GenerationType;

**import** jakarta.persistence.Id;

**import** jakarta.persistence.JoinColumn;

**import** jakarta.persistence.ManyToOne;

**import** jakarta.persistence.Table;

@Entity

@Table(name = "options")

**public** **class** Option {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

**private** String text;

**private** BigDecimal score;

**private** **boolean** isCorrect;

@ManyToOne

@JoinColumn(name = "question\_id")

**private** Question question;

**public** String getText() {

**return** text;

}

**public** BigDecimal getScore() {

**return** score;

}

**public** **boolean** isCorrect() {

**return** isCorrect;

}

**public** Question getQuestion() {

**return** question;

}

}

**Attempt.java**

**package** com.cognizant.ormlearn.model;

**import** java.time.LocalDateTime;

**import** java.util.List;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.GeneratedValue;

**import** jakarta.persistence.GenerationType;

**import** jakarta.persistence.Id;

**import** jakarta.persistence.JoinColumn;

**import** jakarta.persistence.ManyToOne;

**import** jakarta.persistence.OneToMany;

**import** jakarta.persistence.Table;

@Entity

@Table(name = "attempt")

**public** **class** Attempt {

@Id @GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

@ManyToOne

@JoinColumn(name = "user\_id")

**private** User user;

@SuppressWarnings("unused")

**private** LocalDateTime attemptedDate;

@OneToMany(mappedBy = "attempt")

**private** List<AttemptQuestion> attemptQuestions;

**public** List<AttemptQuestion> getAttemptQuestions() {

**return** attemptQuestions;

}

@ManyToOne

@JoinColumn(name = "question\_id")

**private** Question question;

**public** Question getQuestion() {

**return** question;

}

**private** List<AttemptOption> attemptOptions;

**public** List<AttemptOption> getAttemptOptions() {

**return** attemptOptions;

}

}

**AttemptQuestion.java**

**package** com.cognizant.ormlearn.model;

**import** java.util.List;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.GeneratedValue;

**import** jakarta.persistence.GenerationType;

**import** jakarta.persistence.Id;

**import** jakarta.persistence.JoinColumn;

**import** jakarta.persistence.ManyToOne;

**import** jakarta.persistence.OneToMany;

**import** jakarta.persistence.Table;

@Entity

@Table(name = "attempt\_question")

**public** **class** AttemptQuestion {

@Id @GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

@ManyToOne

@JoinColumn(name = "attempt\_id")

**private** Attempt attempt;

@ManyToOne

@JoinColumn(name = "question\_id")

**private** Question question;

@OneToMany(mappedBy = "attemptQuestion")

**private** List<AttemptOption> attemptOptions;

**public** List<AttemptOption> getAttemptOptions() {

**return** attemptOptions;

}

**public** Question getQuestion() {

**return** question;

}

}

**AttemptOption.java**

**package** com.cognizant.ormlearn.model;

**import** java.util.List;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.GeneratedValue;

**import** jakarta.persistence.GenerationType;

**import** jakarta.persistence.Id;

**import** jakarta.persistence.JoinColumn;

**import** jakarta.persistence.ManyToOne;

**import** jakarta.persistence.OneToMany;

**import** jakarta.persistence.Table;

@Entity

@Table(name = "attempt\_question")

**public** **class** AttemptQuestion {

@Id @GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

@ManyToOne

@JoinColumn(name = "attempt\_id")

**private** Attempt attempt;

@ManyToOne

@JoinColumn(name = "question\_id")

**private** Question question;

@OneToMany(mappedBy = "attemptQuestion")

**private** List<AttemptOption> attemptOptions;

**public** List<AttemptOption> getAttemptOptions() {

**return** attemptOptions;

}

**public** Question getQuestion() {

**return** question;

}

}

**AttemptRespository.java**

**package** com.cognizant.ormlearn.repository;

**import** com.cognizant.ormlearn.model.Attempt;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** org.springframework.data.jpa.repository.Query;

**import** org.springframework.data.repository.query.Param;

**public** **interface** AttemptRepository **extends** JpaRepository<Attempt, Integer> {

@Query("SELECT a FROM Attempt a " +

"JOIN FETCH a.user " +

"JOIN FETCH a.attemptQuestions aq " +

"JOIN FETCH aq.question q " +

"JOIN FETCH aq.attemptOptions ao " +

"JOIN FETCH ao.option o " +

"WHERE a.user.id = :userId AND a.id = :attemptId")

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

}

**DepartmentRepository.java**

**package** com.cognizant.ormlearn.service;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** com.cognizant.ormlearn.model.Skill;

**import** com.cognizant.ormlearn.repository.SkillRepository;

@Service

**public** **class** SkillService {

@Autowired

**private** SkillRepository skillRepository;

**public** Skill get(**int** id) {

**return** skillRepository.findById(id).orElseThrow();

}

}

**SkillRepository.java**

**package** com.cognizant.ormlearn.repository;

**import** com.cognizant.ormlearn.model.Skill;

**import** org.springframework.data.jpa.repository.JpaRepository;

**public** **interface** SkillRepository **extends** JpaRepository<Skill, Integer> {

}

**AttemptService.java**

**package** com.cognizant.ormlearn.service;

**import** com.cognizant.ormlearn.model.Attempt;

**import** com.cognizant.ormlearn.repository.AttemptRepository;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

@Service

**public** **class** AttemptService {

@Autowired

**private** AttemptRepository attemptRepository;

**public** Attempt getAttempt(**int** userId, **int** attemptId) {

**return** attemptRepository.getAttempt(userId, attemptId);

}

}

**SkillService.java**

**package** com.cognizant.ormlearn.service;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** com.cognizant.ormlearn.model.Skill;

**import** com.cognizant.ormlearn.repository.SkillRepository;

@Service

**public** **class** SkillService {

@Autowired

**private** SkillRepository skillRepository;

**public** Skill get(**int** id) {

**return** skillRepository.findById(id).orElseThrow();

}

}

**Hands on 4**

**Get average salary using HQL**

**EmployeeService.java**

**public** List<Object[]> getAverageSalaryByDepartment() {

**return** employeeRepository.getAverageSalaryByDepartment();

}

**EmployeeRepository.java**

@Query("SELECT e.department.name, AVG(e.salary) FROM Employee e GROUP BY e.department.name")

List<Object[]> getAverageSalaryByDepartment();

**OrmLearnApplication.java**

TestGetAverageSalaryByDepartment();

**private** **static** **void** testGetAverageSalaryByDepartment() {

***LOGGER***.info("Start");

List<Object[]> result = *employeeService*.getAverageSalaryByDepartment();

**for** (Object[] row : result) {

String deptName = (String) row[0];

Double avgSalary = (Double) row[1];

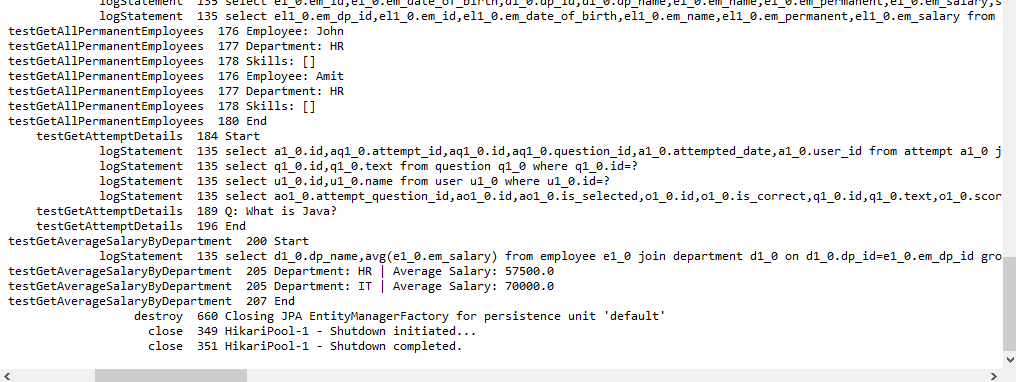
***LOGGER***.info("Department: {} | Average Salary: {}", deptName, avgSalary);

}

***LOGGER***.info("End");

}

**Output:**



**Hands on 5**

**Get all employees using Native Query**

**EmployeeService.java**

**public** List<Employee> getAllEmployeesNative() {

**return** employeeRepository.getAllEmployeesNative();

}

**EmployeeRepository.java**

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

List<Employee> getAllEmployeesNative();

**OrmLearnApplication.java**

**import** org.springframework.beans.factory.annotation.Autowired;

*testGetAllEmployeesNative*();

@Autowired

**private** **static** EmployeeService *employeeService*;

**public** **static** **void** testGetAllEmployeesNative() {

***LOGGER***.info("Start");

List<Employee> employees = *employeeService*.getAllEmployeesNative();

***LOGGER***.debug("All Employees using Native Query: {}", employees);

***LOGGER***.info("End");

}

**Sql code**

-- Ensure a user exists

INSERT INTO user (id, name) VALUES (1, 'Lakshmi')

ON DUPLICATE KEY UPDATE name = 'Lakshmi';

-- Create a valid attempt

INSERT INTO attempt (id, user\_id, attempted\_date) VALUES (2, 1, NOW());

INSERT INTO question (id, text) VALUES (100, 'What is Java?')

ON DUPLICATE KEY UPDATE text = 'What is Java?';

-- attempt\_question

INSERT INTO attempt\_question (id, attempt\_id, question\_id) VALUES (10, 2, 100);

-- option

INSERT INTO option (id, text, score) VALUES (1000, 'A programming language', 1)

ON DUPLICATE KEY UPDATE text = 'A programming language', score = 1;

-- attempt\_option

INSERT INTO attempt\_option (id, is\_selected, attempt\_question\_id, option\_id) VALUES (200, true, 10, 1000);

UPDATE attempt

SET user\_id = 1, attempted\_date = NOW()

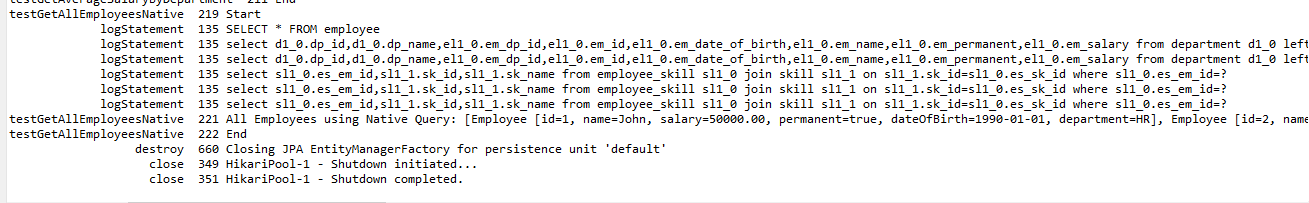
WHERE id = 2; SELECT a.id AS attempt\_id, a.user\_id, aq.id AS aq\_id

FROM attempt a

JOIN attempt\_question aq ON a.id = aq.attempt\_id

WHERE a.user\_id = 1 AND a.id = 2;

**Output:**



**Hands on 6**

**Criteria Query**

**Sql code**

INSERT INTO employee (em\_id, em\_date\_of\_birth, em\_dp\_id, em\_name, em\_permanent, em\_salary)

VALUES (14, '1990-06-05', 2, 'Karthick', true, 50000);

**EmployeeService.java**

@PersistenceContext

**private** EntityManager entityManager;

**public** List<Employee> getEmployeesByCriteria(String name, Integer departmentId, Boolean permanent) {

CriteriaBuilder cb = entityManager.getCriteriaBuilder();

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

Root<Employee> employee = cq.from(Employee.**class**);

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

**if** (name != **null** && !name.isEmpty()) {

predicates.add(cb.like(employee.get("name"), "%" + name + "%"));

}

**if** (departmentId != **null**) {

predicates.add(cb.equal(employee.get("department").get("id"), departmentId));

}

**if** (permanent != **null**) {

predicates.add(cb.equal(employee.get("permanent"), permanent));

}

// predicates

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

// query

**return** entityManager.createQuery(cq).getResultList();

}

**OrmLearnApplication.java**

*testCriteriaQuery*();

**public** **static** **void** testCriteriaQuery() {

***LOGGER***.info("Start");

// Eg: Filter employees with name 'John', departmentId = 2, permanent = true

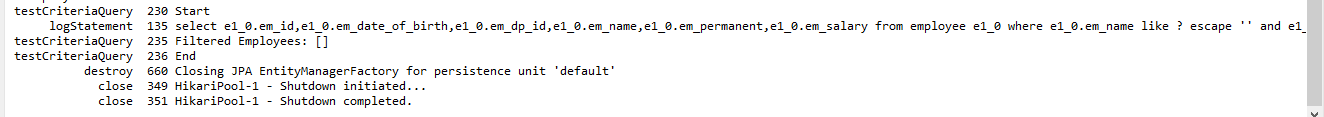
List<Employee> employees = *employeeService*.getEmployeesByCriteria("John", 2, **true**);

***LOGGER***.debug("Filtered Employees: {}", employees);

***LOGGER***.info("End");

}

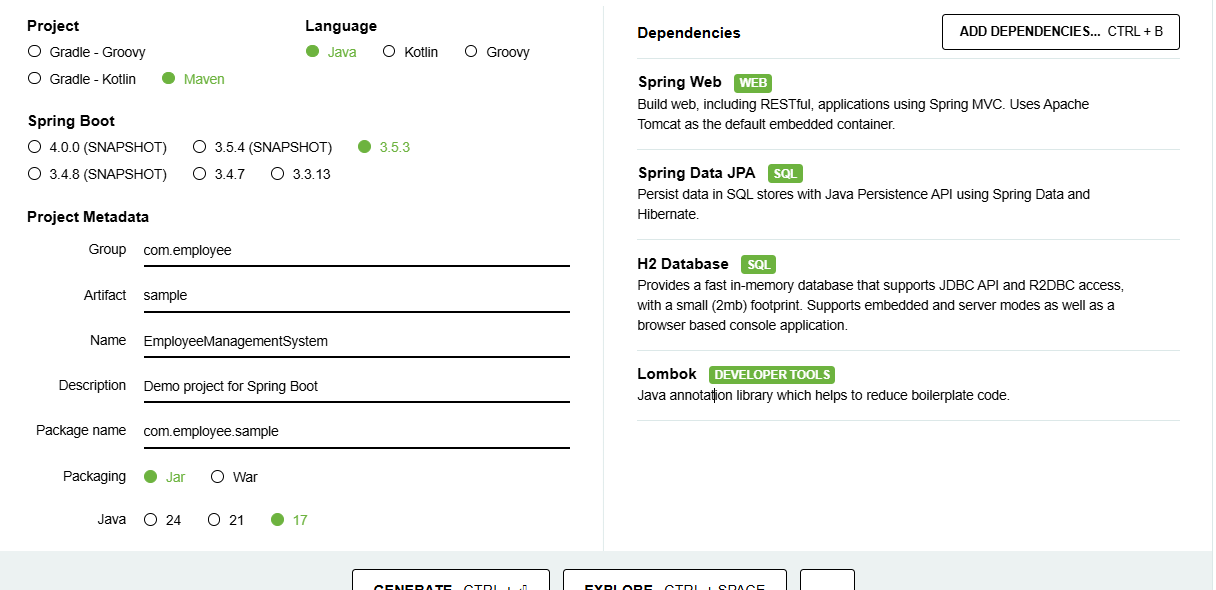
**Output**

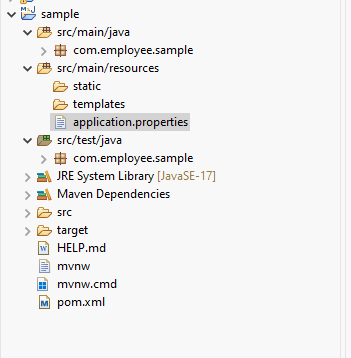


**Docs 4**

**Spring Data JPA and Hibernate**

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





**Application.properties**

spring.application.name=EmployeeManagementSystem

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=root

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.h2.console.enabled=true

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

**Employee.java**

**package** com.employee.sample.model;

**import** jakarta.persistence.\*;

**import** lombok.\*;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

**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;

}

**Department.java**

**package** com.employee.sample.model;

**import** jakarta.persistence.\*;

**import** lombok.\*;

**import** java.util.List;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

**public** **class** Department {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Long id;

**private** String name;

@OneToMany(mappedBy = "department", cascade = CascadeType.***ALL***)

**private** List<Employee> employees;

}

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

**EmployeeRepository.java**

**package** com.employee.sample.repository;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** com.employee.sample.model.Employee;

**public** **interface** EmployeeRepository **extends** JpaRepository<Employee, Long> {}

**DepartmentRepository.java**

**package** com.employee.sample.repository;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** com.employee.sample.model.Department;

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

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

**EmployeeController.java**

**package** com.employee.sample.controller;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

**import** com.employee.sample.model.Employee;

**import** com.employee.sample.repository.EmployeeRepository;

@RestController

@RequestMapping("/employees")

**public** **class** EmployeeController {

@Autowired

**private** EmployeeRepository employeeRepository;

@GetMapping

**public** List<Employee> getAllEmployees() {

**return** employeeRepository.findAll();

}

@PostMapping

**public** Employee createEmployee(@RequestBody Employee employee) {

**return** employeeRepository.save(employee);

}

}

**DepartmentController.java**

**package** com.employee.sample.controller;

**import** com.employee.sample.model.Department;

**import** com.employee.sample.repository.DepartmentRepository;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.web.bind.annotation.\*;

**import** java.util.List;

@RestController

@RequestMapping("/departments")

**public** **class** DepartmentController {

@Autowired

**private** DepartmentRepository departmentRepository;

// GET all departments

@GetMapping

**public** List<Department> getAllDepartments() {

**return** departmentRepository.findAll();

}

// GET department by ID

@GetMapping("/{id}")

**public** Department getDepartmentById(@PathVariable Long id) {

**return** departmentRepository.findById(id)

.orElseThrow(() -> **new** RuntimeException("Department not found"));

}

// POST create new department

@PostMapping

**public** Department createDepartment(@RequestBody Department department) {

**return** departmentRepository.save(department);

}

// PUT update department

@PutMapping("/{id}")

**public** Department updateDepartment(@PathVariable Long id, @RequestBody Department departmentDetails) {

Department department = departmentRepository.findById(id)

.orElseThrow(() -> **new** RuntimeException("Department not found"));

department.setName(departmentDetails.getName());

**return** departmentRepository.save(department);

}

// DELETE department

@DeleteMapping("/{id}")

**public** **void** deleteDepartment(@PathVariable Long id) {

departmentRepository.deleteById(id);

}

}

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

**EmployeeRepository.java**

**package** com.employee.sample.repository;

**import** com.employee.sample.model.Employee;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** java.util.List;

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

List<Employee> findByName(String name); // Find by name

List<Employee> findByNameContaining(String keyword); // Partial match

List<Employee> findByDepartmentId(Long departmentId); // Employees department

}

**DepartmentRepository.java**

**package** com.employee.sample.repository;

**import** com.employee.sample.model.Department;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** java.util.Optional;

**public** **interface** DepartmentRepository **extends** JpaRepository<Department, Long> {

Optional<Department> findByName(String name); // Find department by name

}

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

**EmployeeController.java**

**package** com.employee.sample.controller;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestParam;

**import** org.springframework.web.bind.annotation.RestController;

**import** com.employee.sample.model.Employee;

**import** com.employee.sample.repository.EmployeeRepository;

@RestController

@RequestMapping("/employees")

**public** **class** EmployeeController {

@Autowired

**private** EmployeeRepository employeeRepository;

@GetMapping

**public** List<Employee> getAllEmployees() {

**return** employeeRepository.findAll();

}

@PostMapping

**public** Employee createEmployee(@RequestBody Employee employee) {

**return** employeeRepository.save(employee);

}

// docx 4 hands-on 6

@GetMapping("/search")

**public** List<Employee> searchByName(@RequestParam String name) {

**return** employeeRepository.findByNameContaining(name);

}

@GetMapping("/page")

**public** Page<Employee> getEmployeesWithPagination(

@RequestParam(defaultValue = "0") **int** page,

@RequestParam(defaultValue = "5") **int** size,

@RequestParam(defaultValue = "id,asc") String[] sort) {

// Create sort object

Sort sortOrder = Sort.by(Sort.Direction.fromString(sort[1]), sort[0]);

Pageable pageable = PageRequest.*of*(page, size, sortOrder);

**return** employeeRepository.findAll(pageable);

}

**DepartmentController.java**

**package** com.employee.sample.controller;

**import** com.employee.sample.model.Department;

**import** com.employee.sample.repository.DepartmentRepository;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.web.bind.annotation.\*;

**import** java.util.List;

@RestController

@RequestMapping("/departments")

**public** **class** DepartmentController {

@Autowired

**private** DepartmentRepository departmentRepository;

// GET all departments

@GetMapping

**public** List<Department> getAllDepartments() {

**return** departmentRepository.findAll();

}

// GET department by ID

@GetMapping("/{id}")

**public** Department getDepartmentById(@PathVariable Long id) {

**return** departmentRepository.findById(id)

.orElseThrow(() -> **new** RuntimeException("Department not found"));

}

// POST create new department

@PostMapping

**public** Department createDepartment(@RequestBody Department department) {

**return** departmentRepository.save(department);

}

// PUT update department

@PutMapping("/{id}")

**public** Department updateDepartment(@PathVariable Long id, @RequestBody Department departmentDetails) {

Department department = departmentRepository.findById(id)

.orElseThrow(() -> **new** RuntimeException("Department not found"));

department.setName(departmentDetails.getName());

**return** departmentRepository.save(department);

}

// DELETE department

@DeleteMapping("/{id}")

**public** **void** deleteDepartment(@PathVariable Long id) {

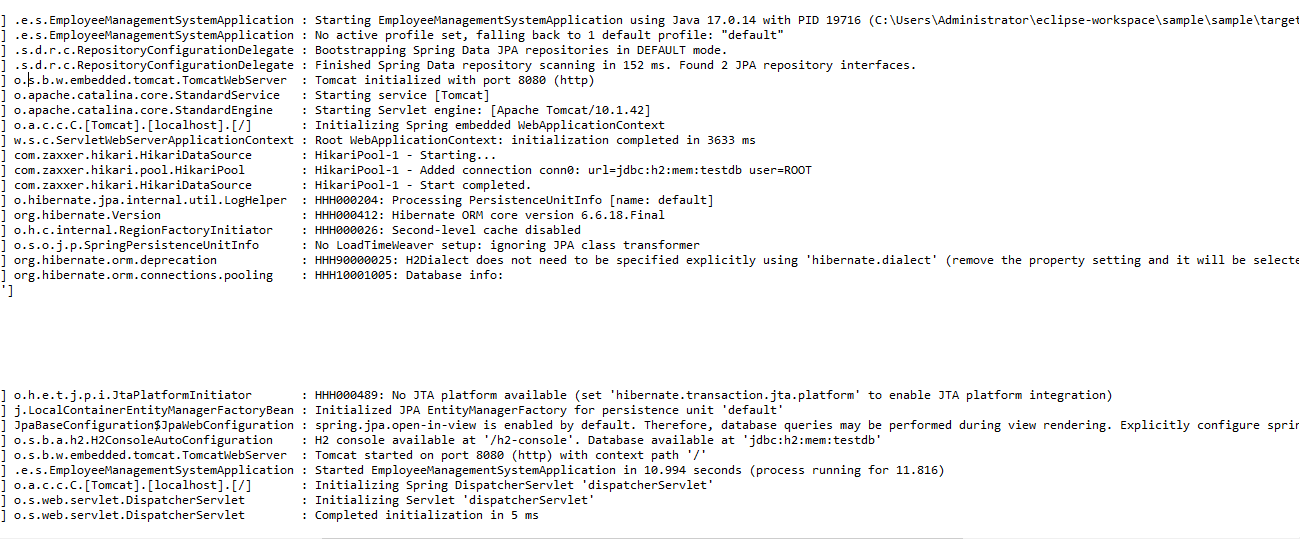
departmentRepository.deleteById(id);

}

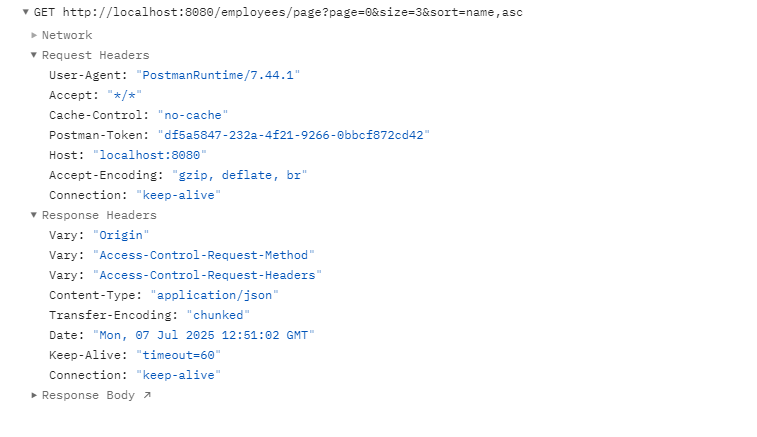
}

**Application.properties**  
server.port=8080

**Output**







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

**EmployeeManagementSystemApplication.java**

**import** org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@EnableJpaAuditing(auditorAwareRef = "auditorProvider")

**AuditorAwareImpl.java**

**package** com.employee.sample.config;

**import** org.springframework.data.domain.AuditorAware;

**import** org.springframework.stereotype.Component;

**import** java.util.Optional;

@Component("auditorProvider")

**public** **class** AuditorAwareImpl **implements** AuditorAware<String> {

@Override

**public** Optional<String> getCurrentAuditor() {

// fetch from Spring Security. For now, static.

**return** Optional.*of*("SystemAdmin");

}

}

**Auditable.java**

**package** com.employee.sample.model;

**import** jakarta.persistence.\*;

**import** lombok.Getter;

**import** lombok.Setter;

**import** org.springframework.data.annotation.CreatedDate;

**import** org.springframework.data.annotation.LastModifiedDate;

**import** org.springframework.data.jpa.domain.support.AuditingEntityListener;

**import** java.time.LocalDateTime;

@MappedSuperclass

@EntityListeners(AuditingEntityListener.**class**)

@Getter

@Setter

**public** **abstract** **class** Auditable {

@CreatedDate

@Column(updatable = **false**)

**private** LocalDateTime createdDate;

@LastModifiedDate

**private** LocalDateTime lastModifiedDate;

}

**Employee.java**

**package** com.employee.sample.model;

**import** jakarta.persistence.\*;

**import** lombok.\*;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

**public** **class** Employee **extends** Auditable {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Long id;

**private** String name;

**private** String email;

@ManyToOne

@JoinColumn(name = "department\_id")

**private** Department department;

}

**Department.java**

**package** com.employee.sample.model;

**import** jakarta.persistence.\*;

**import** lombok.\*;

**import** java.util.List;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

**public** **class** Department **extends** Auditable {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Long id;

**private** String name;

@OneToMany(mappedBy = "department", cascade = CascadeType.***ALL***)

**private** List<Employee> employees;

}

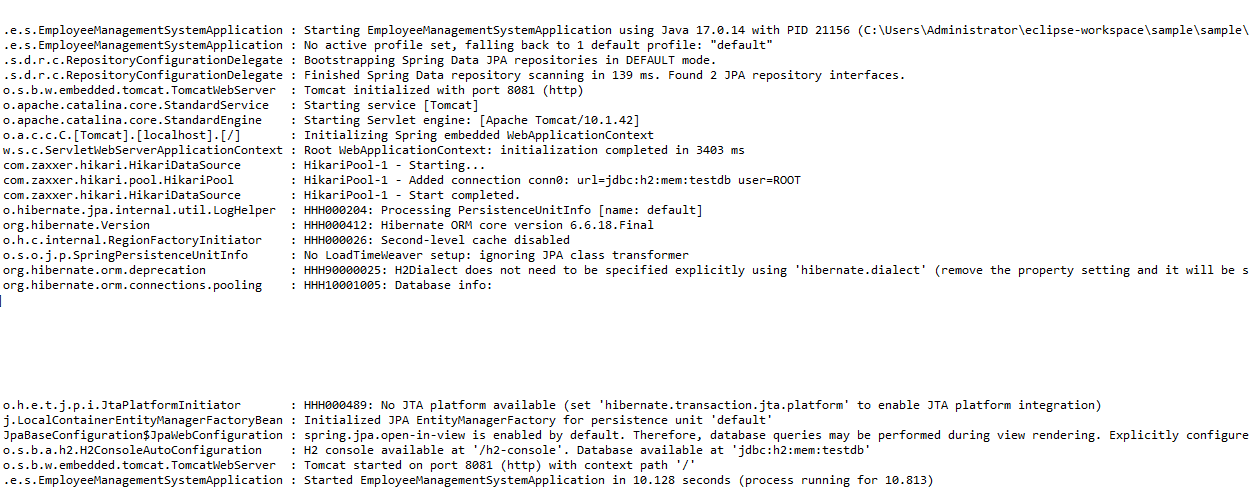
**Employee and Department.java**

@EntityListeners(AuditingEntityListener.**class**)

**Application.properties**

server.port=8081

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



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

**Projection.java**

**package** com.employee.sample;

**public** **interface** Projection {

String getName();

String getEmail();

}

**EmployeeRepository.java**

List<Projection> findByDepartment\_Name(String departmentName);

**EmployeeDTO.java**

**package** com.employee.sample.model;

**public** **class** EmployeeDTO {

**private** String name;

**private** String email;

**public** EmployeeDTO(String name, String email) {

**this**.name = name;

**this**.email = email;

}

// Getters

**public** String getName() {

**return** name;

}

**public** String getEmail() {

**return** email;

}

}

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

**Application.properties**

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

**PrimaryDataSourceConfig.java**

**package** com.employee.sample.config;

**import** javax.sql.DataSource;

**import** org.hibernate.jpa.boot.spi.EntityManagerFactoryBuilder;

**import** org.springframework.beans.factory.annotation.Qualifier;

**import** org.springframework.boot.autoconfigure.jdbc.DataSourceProperties;

**import** org.springframework.boot.context.properties.ConfigurationProperties;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.context.annotation.Primary;

**import** org.springframework.data.jpa.repository.config.EnableJpaRepositories;

**import** org.springframework.orm.jpa.JpaTransactionManager;

**import** org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

**import** org.springframework.transaction.PlatformTransactionManager;

**import** org.springframework.transaction.annotation.EnableTransactionManagement;

**import** jakarta.persistence.EntityManagerFactory;

@Configuration

@EnableTransactionManagement

@EnableJpaRepositories(

basePackages = "com.employee.sample.repository",

entityManagerFactoryRef = "primaryEntityManagerFactory",

transactionManagerRef = "primaryTransactionManager"

)

**public** **class** PrimaryDataSourceConfig {

@Bean

@Primary

@ConfigurationProperties("spring.datasource")

**public** DataSourceProperties primaryProps() {

**return** **new** DataSourceProperties();

}

@Bean

@Primary

**public** DataSource primaryDataSource() {

**return** primaryProps().initializeDataSourceBuilder().build();

}

@Bean

@Primary

**public** LocalContainerEntityManagerFactoryBean primaryEntityManagerFactory(EntityManagerFactoryBuilder builder) {

**return** builder

.dataSource(primaryDataSource())

.packages("com.employee.sample.model")

.persistenceUnit("primary")

.build();

}

@Bean

@Primary

**public** PlatformTransactionManager primaryTransactionManager(EntityManagerFactory emf) {

**return** **new** JpaTransactionManager(emf);

}

}

**SecondaryDataSourceConfig.java**

**package** com.employee.sample.config;

**import** javax.sql.DataSource;

**import** org.springframework.boot.autoconfigure.jdbc.DataSourceProperties;

**import** org.springframework.boot.context.properties.ConfigurationProperties;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.data.jpa.repository.config.EnableJpaRepositories;

**import** org.springframework.orm.jpa.JpaTransactionManager;

**import** org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

**import** org.springframework.transaction.PlatformTransactionManager;

**import** org.springframework.transaction.annotation.EnableTransactionManagement;

**import** jakarta.persistence.EntityManagerFactory;

**import** org.springframework.boot.orm.jpa.EntityManagerFactoryBuilder;

@Configuration

@EnableTransactionManagement

@EnableJpaRepositories(

basePackages = "com.employee.sample.repository.secondary",

entityManagerFactoryRef = "secondaryEntityManagerFactory",

transactionManagerRef = "secondaryTransactionManager"

)

**public** **class** SecondaryDataSourceConfig {

@Bean

@ConfigurationProperties("spring.secondary-datasource")

**public** DataSourceProperties secondaryProps() {

**return** **new** DataSourceProperties();

}

@Bean

**public** DataSource secondaryDataSource() {

**return** secondaryProps().initializeDataSourceBuilder().build();

}

@Bean

**public** LocalContainerEntityManagerFactoryBean secondaryEntityManagerFactory(EntityManagerFactoryBuilder builder) {

**return** builder

.dataSource(secondaryDataSource())

.packages("com.employee.sample.model.secondary")

.persistenceUnit("secondary")

.build();

}

@Bean

**public** PlatformTransactionManager secondaryTransactionManager(EntityManagerFactory emf) {

**return** **new** JpaTransactionManager(emf);

}

}

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

**Application.properties**

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect

spring.jpa.properties.hibernate.show\_sql=true

spring.jpa.properties.hibernate.format\_sql=true

spring.jpa.properties.hibernate.jdbc.batch\_size=30

spring.jpa.properties.hibernate.order\_inserts=true

spring.jpa.properties.hibernate.order\_updates=true

spring.jpa.properties.hibernate.generate\_statistics=true

**Department.java**

**public** String getName() {

**return** name;

}

**Employee.java**

@ManyToOne

@JoinColumn(name = "department\_id")

**private** Department department;

**EmployeeControllers.java**

@GetMapping("/page")

**public** Page<Employee> getEmployeesWithPagination(

@RequestParam(defaultValue = "0") **int** page,

@RequestParam(defaultValue = "5") **int** size,

@RequestParam(defaultValue = "id,asc") String[] sort) {

String sortBy = sort[0];

String direction = sort[1];

Sort sortObj = Sort.*by*(Direction.*fromString*(direction), sortBy);

Pageable pageable = PageRequest.*of*(page, size, sortObj);

**return** employeeRepository.findAll(pageable);

}

