**ORM (Object-Relational Mapping)**

**Need and Benefits**

- Simplifies database interactions by mapping Java objects to database tables

- Abstracts database system specifics

- Handles transaction management automatically

- Reduces boilerplate JDBC code

- Provides database independence

**Spring Data JPA**

**Evolution**

- From Hibernate XML configuration → Hibernate annotations → Spring Data JPA

**Benefits**

- Higher abstraction over Hibernate

- Reduces boilerplate code

- Built-in CRUD operations

- Easy query creation from method names

- Transaction management

**Hibernate Core Objects**

**1. SessionFactory:** Creates Session instances (thread-safe)

**2. Session:** Main runtime interface between Java application and Hibernate

**3. Transaction Factory/Transaction:** Defines unit of work

**4. Connection Provider:** Manages database connections

**JPA vs Hibernate vs Spring Data JPA**

**- JPA:** Specification (JSR 338) with no implementation

**- Hibernate:** ORM tool that implements JPA specification

**- Spring Data JPA:** Abstraction layer over JPA providers (like Hibernate) that reduces boilerplate code

**Hands-on Implementation**

**Hands-on 1: Basic Setup**

**1.Project Creation:**

- Use Spring Initializr with:

- Group: `com.cognizant`

- Artifact: `orm-learn`

- Dependencies: Spring Boot DevTools, Spring Data JPA, MySQL Driver

**2. Database Configuration(`application.properties`):**

**# Database**

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn

spring.datasource.username=root

spring.datasource.password=root

**# Hibernate**

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

**# Logging**

logging.level.org.hibernate.SQL=trace

logging.level.org.hibernate.type.descriptor.sql=trace

```

**3. Country Table Creation:**

**```sql**

CREATE TABLE country(co\_code VARCHAR(2) PRIMARY KEY, co\_name VARCHAR(50));

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

INSERT INTO country VALUES ('US', 'United States of America');

```

**Entity, Repository, and Service Implementation**

**1. Entity Class (`Country.java`):**

```java

@Entity

@Table(name="country")

public class Country {

@Id

@Column(name="co\_code")

private String code;

@Column(name="co\_name")

private String name;

// Getters, setters, toString()

}

```

**2. Repository Interface (`CountryRepository.java`):**

```java

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

```

**3. Service Class (`CountryService.java`):**

```java

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

```

**Hands-on 6: Find Country by Code**

```java

// In CountryService

@Transactional

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

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

if (!result.isPresent()) {

throw new CountryNotFoundException();

}

return result.get();

}

// In OrmLearnApplication

private static void testGetCountryByCode() {

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

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

}

```

**Hands-on 7: Add Country**

```java

// In CountryService

@Transactional

public void addCountry(Country country) {

countryRepository.save(country);

}

// Test method

private static void testAddCountry() {

Country newCountry = new Country();

newCountry.setCode("DE");

newCountry.setName("Germany");

countryService.addCountry(newCountry);

Country found = countryService.findCountryByCode("DE");

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

}

```

**Hands-on 8: Update Country**

```java

// In CountryService

@Transactional

public void updateCountry(String code, String name) {

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

if (result.isPresent()) {

Country country = result.get();

country.setName(name);

countryRepository.save(country);

}

}

// Test method

private static void testUpdateCountry() {

countryService.updateCountry("DE", "Germany (Updated)");

Country updated = countryService.findCountryByCode("DE");

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

}

```

**Hands-on 9: Delete Country**

```java

// In CountryService

@Transactional

public void deleteCountry(String code) {

countryRepository.deleteById(code);

}

// Test method

private static void testDeleteCountry() {

countryService.deleteCountry("DE");

try {

Country deleted = countryService.findCountryByCode("DE");

} catch (CountryNotFoundException e) {

LOGGER.debug("Country deleted successfully");

}

}

**Query Methods for Partial Name Matching**

```java

// In CountryRepository

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContaining(String partialName);

List<Country> findByNameContainingOrderByNameAsc(String partialName);

List<Country> findByNameStartingWithOrderByNameAsc(String startingLetter);

}

// In CountryService

@Transactional

public List<Country> findCountriesByNameContaining(String partialName) {

return countryRepository.findByNameContainingOrderByNameAsc(partialName);

}

```

**Key Annotations**

**- `@Entity`:** Marks class as JPA entity

**- `@Table`:** Specifies the mapped table

**- `@Id`:** Marks field as primary key

**- `@Column`**: Maps field to table column

**- `@Repository`:** Marks interface as Spring Data repository

**- `@Service`:** Marks class as Spring service component

**- `@Transactional`:** Defines transaction boundaries

**Hibernate Configuration Options**

**`ddl-auto` values:**

**- `validate`:** Validates schema, doesn't make changes

**- `update`:** Updates schema if needed

**- `create`:** Creates schema, destroys previous data

**- `create-drop`:** Creates on startup, drops on exit

**- `none`:** No action