**Week-3 – Spring Data JPA With Spring Boot Hibernate**

**Spring Data JPA – Quick Setup and Project Overview**

1. **A Spring Boot project was initialized using Spring Initializr with the following configurations:**
   * **Group: com.cognizant**
   * **Artifact: orm-learn**
   * **Description: Demo project for Spring Data JPA and Hibernate**
   * **Dependencies Added: Spring Boot DevTools, Spring Data JPA, and MySQL Driver**
2. **The generated project was extracted and imported into Eclipse IDE as a Maven project, ensuring seamless dependency management and build integration.**
3. **A MySQL schema named ormlearn was created to act as the database for the application. This schema is used to store and retrieve persistent objects.**
4. **Configuration for database connectivity and logging was added in the application.properties file located in the src/main/resources directory:**
   * **JDBC connection URL, username, and password were provided for MySQL.**
   * **Hibernate dialect was specified.**
   * **SQL and log tracing were enabled to track queries and debug application flow.**
5. **The main application class, OrmLearnApplication.java, includes a logging mechanism using SLF4J to verify the application startup and track execution logs.**
6. **Folder structure in the project:**
   * **src/main/java: Contains the main application code and business logic**
   * **src/main/resources: Holds application configuration files like application.properties**
   * **src/test/java: Reserved for unit and integration tests**
7. **The @SpringBootApplication annotation in the main class enables component scanning, auto-configuration, and configuration properties support, simplifying setup.**
8. **A database table named country was created with the following schema:**
   * **co\_code (Primary Key, VARCHAR(2))**
   * **co\_name (VARCHAR(50))  
     Sample data inserted:**
   * **('IN', 'India')**
   * **('US', 'United States of America')**
9. **A JPA entity class Country was created inside the com.cognizant.ormlearn.model package:**
   * **Annotated with @Entity and @Table(name = "country")**
   * **Contains two fields: code and name**
   * **Each field is mapped to its corresponding database column using @Column**
   * **Includes getters, setters, and a toString() method**
10. **A repository interface CountryRepository was created in the com.cognizant.ormlearn.repository package:**
    * **Extends JpaRepository<Country, String>**
    * **Annotated with @Repository to mark it as a Spring-managed component**

> mysql -u root -p

mysql> create schema ormlearn;

* In orm-learn Eclipse project, open src/main/resources/application.properties and include the below database and log configuration.

# Spring Framework and application log

logging.level.org.springframework=info

logging.level.com.cognizant=debug

# Hibernate logs for displaying executed SQL, input and output

logging.level.org.hibernate.SQL=trace

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

# Log pattern

logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger{25} %25M %4L %m%n

# Database configuration

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 configuration

spring.jpa.hibernate.ddl-auto=validate

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

* Build the project using ‘mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456’ command in command line
* Include logs for verifying if main() method is called.

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

  LOGGER.info("Inside main");

}

* Execute the OrmLearnApplication and check in log if main method is called.

SME to walk through the following aspects related to the project created:

1. src/main/java - Folder with application code
2. src/main/resources - Folder for application configuration
3. src/test/java - Folder with code for testing the application
4. OrmLearnApplication.java - Walkthrough the main() method.
5. Purpose of @SpringBootApplication annotation
6. pom.xml
   1. Walkthrough all the configuration defined in XML file
   2. Open 'Dependency Hierarchy' and show the dependency tree.

**Country table creation**

* Create a new table country with columns for code and name. For sample, let us insert one country with values 'IN' and 'India' in this table.

create table country(co\_code varchar(2) primary key, co\_name varchar(50));

* Insert couple of records into the table

insert into country values ('IN', 'India');

insert into country values ('US', 'United States of America');

**Persistence Class - com.cognizant.orm-learn.model.Country**

* Open Eclipse with orm-learn project
* Create new package com.cognizant.orm-learn.model
* Create Country.java, then generate getters, setters and toString() methods.
* Include @Entity and @Table at class level
* Include @Column annotations in each getter method specifying the column name.

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name="country")

public class Country {

  @Id

    @Column(name="code")

    private String code;

    @Column(name="name")

    private String name;

// getters and setters

  // toString()

}

*Notes:*

* @Entity is an indicator to Spring Data JPA that it is an entity class for the application
* @Table helps in defining the mapping database table
* @Id helps is defining the primary key
* @Column helps in defining the mapping table column

**Repository Class - com.cognizant.orm-learn.CountryRepository**

* Create new package com.cognizant.orm-learn.repository
* Create new interface named CountryRepository that extends JpaRepository<Country, String>
* Define @Repository annotation at class level

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

import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**Service Class - com.cognizant.orm-learn.service.CountryService**

* Create new package com.cognizant.orm-learn.service
* Create new class CountryService
* Include @Service annotation at class level
* Autowire CountryRepository in CountryService
* Include new method getAllCountries() method that returns a list of countries.
* Include @Transactional annotation for this method
* In getAllCountries() method invoke countryRepository.findAll() method and return the result

**Testing in OrmLearnApplication.java**

* Include a static reference to CountryService in OrmLearnApplication class

private static CountryService countryService;

* Define a test method to get all countries from service.

    private static void testGetAllCountries() {

        LOGGER.info("Start");

        List<Country> countries = countryService.getAllCountries();

        LOGGER.debug("countries={}", countries);

        LOGGER.info("End");

    }

* Modify SpringApplication.run() invocation to set the application context and the CountryService reference from the application context.

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

        countryService = context.getBean(CountryService.class);

        testGetAllCountries();

* Execute main method to check if data from ormlearn database is retrieved.

**OUTPUT :**

A blue and white screen

AI-generated content may be incorrect.

**Difference Between JPA, Hibernate, and Spring Data JPA**

**Java Persistence API (JPA)**

* **It is a specification (JSR 338) that defines how Java objects should be stored in a database.**
* **No concrete implementation—just a set of interfaces and rules.**
* **Think of it like a blueprint or contract.**
* **Hibernate is one of the most popular implementations of JPA.**

**Hibernate**

* **Hibernate is an ORM (Object Relational Mapping) framework.**
* **It provides the actual implementation of JPA interfaces.**
* **Allows you to map Java classes to database tables and persist them using SQL under the hood.**
* **Manages session handling, transactions, and queries manually (or semi-automatically).**

**Spring Data JPA**

* **Spring Data JPA is not a JPA provider, but a helper layer built on top of JPA (and typically Hibernate).**
* **Eliminates boilerplate code (e.g., CRUD operations, transaction handling).**
* **Uses Spring’s dependency injection and abstraction.**
* **You mostly just define interfaces and annotate — Spring handles the rest.**

**Refer code snippets below on how the code compares between Hibernate and Spring Data JPA  
Hibernate**

   /\* Method to CREATE an employee in the database \*/

   public Integer createUser(User userData) {

Session dbSession = sessionFactory.openSession();

Transaction transaction = null;

Integer userId = null;

try {

transaction = dbSession.beginTransaction();

userId = (Integer) dbSession.save(userData);

transaction.commit();

} catch (HibernateException ex) {

if (transaction != null) transaction.rollback();

ex.printStackTrace();

} finally {

dbSession.close();

}

return userId;

}**Spring Data JPA**  
EmployeeRespository.java

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

EmployeeService.java

@Service

public class UserService {

@Autowired

private UserRepository userRepo;

@Transactional

public void registerUser(User user) {

userRepo.save(user); // That's it!

}

}

**Output:**

**Code Comparison**

**Hibernate:**

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

session.save(employee);

tx.commit();

session.close();

**Spring Data JPA:**

employeeRepository.save(employee);

**A black screen with blue and white text

AI-generated content may be incorrect.**