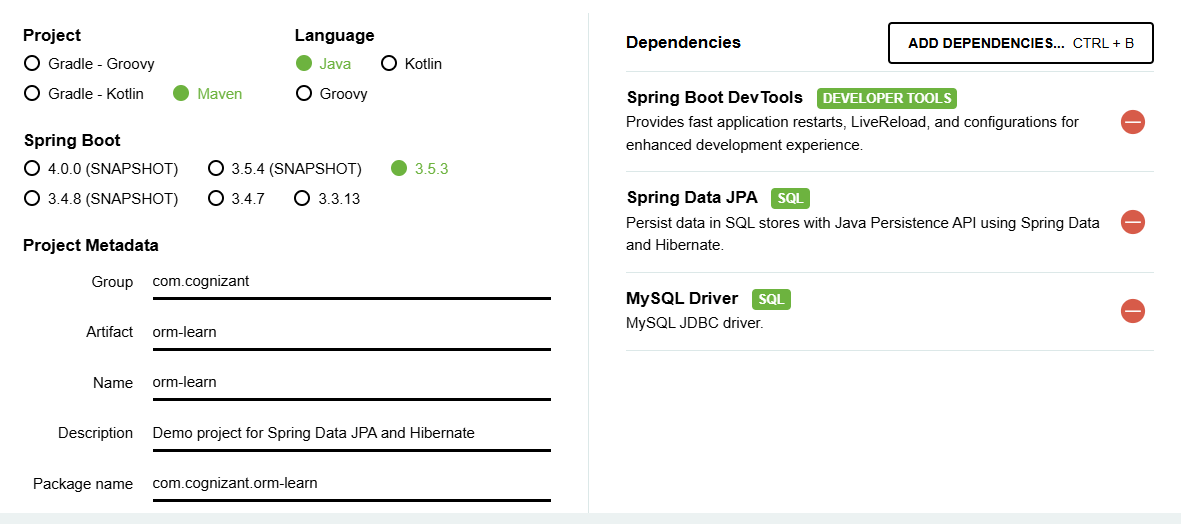
**SKILL LEARNT: SPRING DATA JPA WITH SPRING BOOT, HIBERNATE (WEEK 3)**

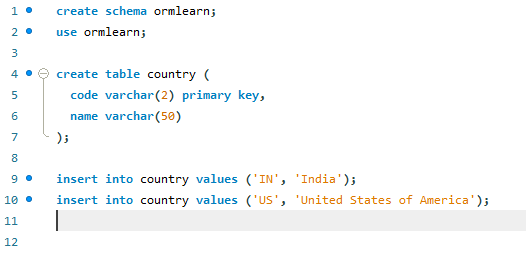
**EXERCISE 1: SPRING DATA JPA- QUICK EXAMPLE**

STEP 1: Firstly I went to Spring Initializr and filled in the details, like project Maven, Language Java, Spring Boot 3.5.3, group com.cognizant, artifact orm-learn, description Demo project for Spring Data JPA and Hibernate, and added some dependencies like Spring Boot Dev Tools, Spring Boot JPA, and MySQL Driver and generated the zip.

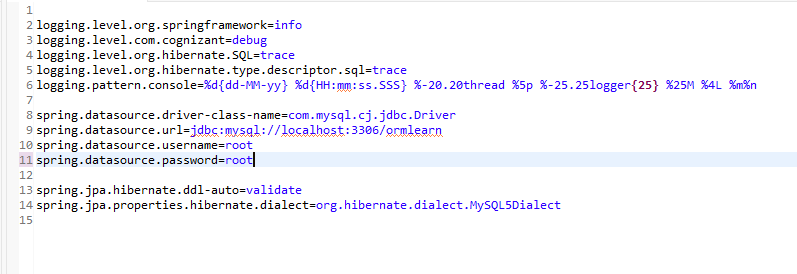


STEP 2: I uploaded the unzipped folder to my Eclipse IDE.

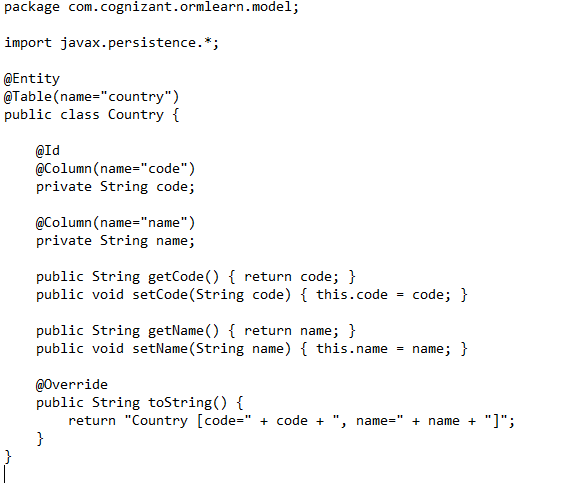
STEP 3: Then I created a new MySql schema



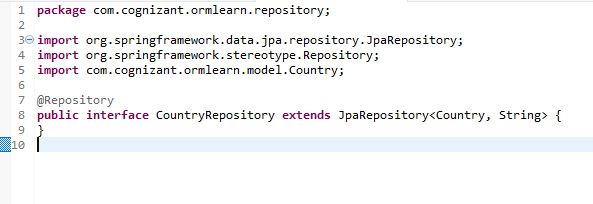
STEP 4: Then I edited the application.properties



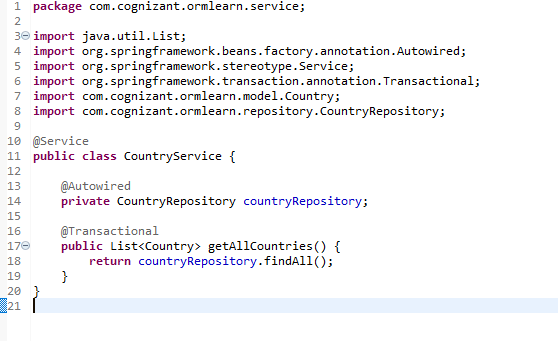
STEP 5: Then under src/main/java/com.cognizant.ormlearn, I created a new package com.cognizant.ormlearn.model, under which I created a class named Country. The code for Country is as follows,



STEP 6: I created a new package com.cognizant.ormlearn.repository under which I created a class named CountryRepository. The code for it is as follows,



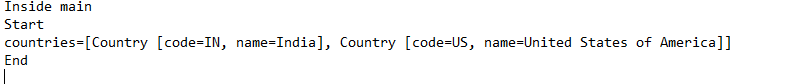
STEP 7: Again I created a new package named com.cognizant.ormlearn.service under which a class named CountryService. The code for it is as follows,



STEP 8: Now I updated the main application named OrmLearnApplication. The code for it is as follows



STEP 9: Runing the OrmLearnApplication application produces the output.



**EXERCISE 4: DIFFERENCE BETWEEN JPA, HIBERNATE AND SPRING DATA JPA**

**JPA:**

**JPA**can be defined as **Java Persistence API**. It is the Java specification that can provide a standardized way to manage the relational data in Java applications. JPA facilitates the management of the database operations and mapping of the Java objects to database tables by defining the concepts and APIs. It serves as the bridge between the object-oriented domain models and [relational database systems](https://www.geeksforgeeks.org/rdbms-full-form/). The Java Persistence API (JPA) is the Java API specification that bridges the gap between relational databases and object-oriented programming by handling the mapping of the Java objects to the database tables and vice-versa. This process is known as the [Object Relational Mapping (ORM)](https://www.geeksforgeeks.org/jpa-object-relational-mapping/). JPA can define the way Java classes (entities) are mapped to the database tables and how they can interact with the database through the EntityManager, the Persistence context, and transactions.

**HIBERNATE:**

Hibernate is an open-source Java Object-Relational Mapping (ORM) framework that streamlines the interaction between Java applications and relational databases. Hibernate uses the Java Persistence API (JPA) specification, but includes many more robust features like caching, lazy loading, and batch processing. Rather than coding intricate SQL queries and manually mapping Java objects to database tables, developers can let Hibernate do this work for them through annotations or XML configuration, allowing them to map Java classes to database tables automatically. Hibernate takes care of database access details, transactions, and Java type-to-SQL type conversions, thus making it significantly simpler to create solid, database-based applications with fewer boilerplate codes.

**SPRING DATA JPA:**

Spring Data JPA is a component of the Spring Framework that offers a higher-level abstraction of JPA implementations such as Hibernate to simplify building data access layers in Java-based applications. Spring Data JPA removes the majority of boilerplate code necessary for database operations by enabling developers to specify repository interfaces that come with CRUD operations, query methods based on method names, and pagination and sorting support. With Spring Data JPA, you don't have to author implementation classes or deal with transactions manually—everything is done declaratively using Spring's dependency injection and annotations. This simplifies development, enhances maintainability, and allows you to concentrate on business logic rather than boilerplate data access code.

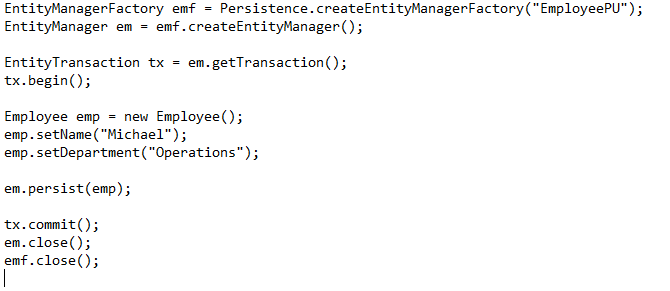
**COMPARISON CHART:**

|  |  |  |  |
| --- | --- | --- | --- |
| **FEATURE** | **JPA** | **HIBERNATE** | **SPRING DATA JPA** |
| 1. TYPE | Standard API | Implementation of JPA plus extra ORM features | Abstraction over JPA implementation |
| 1. Implementation | No | Yes | No |
| 1. Boilerplate code | Medium | Less than JDBC | Very minimal |
| 1. Provider | Oracle | Red Hat | Spring framework |
| 1. Repository Support | No | No | Yes |

**The difference in terms of code will be:**

1. **Employee Insertion Using JPA EntityManager**

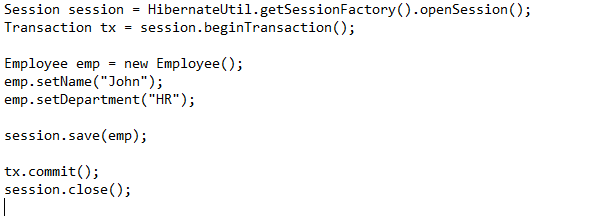
The code is as follows:



**Characteristics:**

1. No Hibernate Session API.
2. Standard JPA EntityManager.
3. Manual transaction handling.
4. **Employee Insertion Using Hibernate Session API**

The code is as follows:

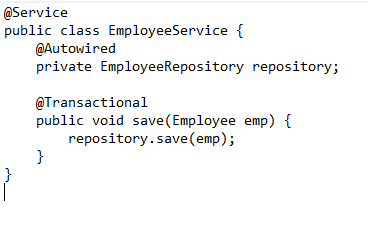


**Characteristics:**

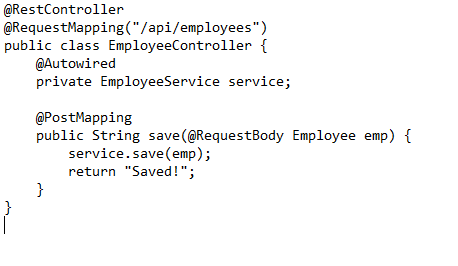
1. Manual session management.
2. Manual transaction handling.
3. Hibernate API (Session, Transaction).
4. **Employee Insertion Using JpaRepository**
5. **Repository Interface**



1. **Service Class**



1. **Controller**



**Characteristics:**

1. No manual session or transaction handling.
2. Uses Spring @Autowired dependency injection.
3. Repository abstraction handles CRUD automatically.

**With respect to the codes given in question the differences are:**

|  |  |  |
| --- | --- | --- |
| **ASPECT** | **HIBERNATE CODE** | **SPRING DATA JPA CODE** |
| 1. SESSION MANAGEMENT | Session session = factory.openSession(); — We **explicitly open** a Hibernate session. | **No session code needed**. Spring Data JPA handles the EntityManager internally. |
| 1. TRANSACTION MANAGEMENT | Transaction tx = session.beginTransaction(); —We **manually begin** a transaction. | @Transactional annotation on the method automatically starts and commits/rolls back the transaction. |
| 1. PERSISTENCE CALL | employeeID = (Integer) session.save(employee); — We **call Hibernate’s** save() method. | employeeRepository.save(employee); — We call **Spring Data JPA’s repository save()** method. |
| 1. EXCEPTION HANDLING | try-catch-finally block needed: you **handle HibernateException, rollback, and close session manually.** | No try-catch-finally required—**Spring handles exceptions and rolls back automatically if needed.** |
| 1. RETURN VALUE | Returns Integer employee ID from save(). | Usually returns the saved entity itself (or void), as save() returns the entity with ID populated. |