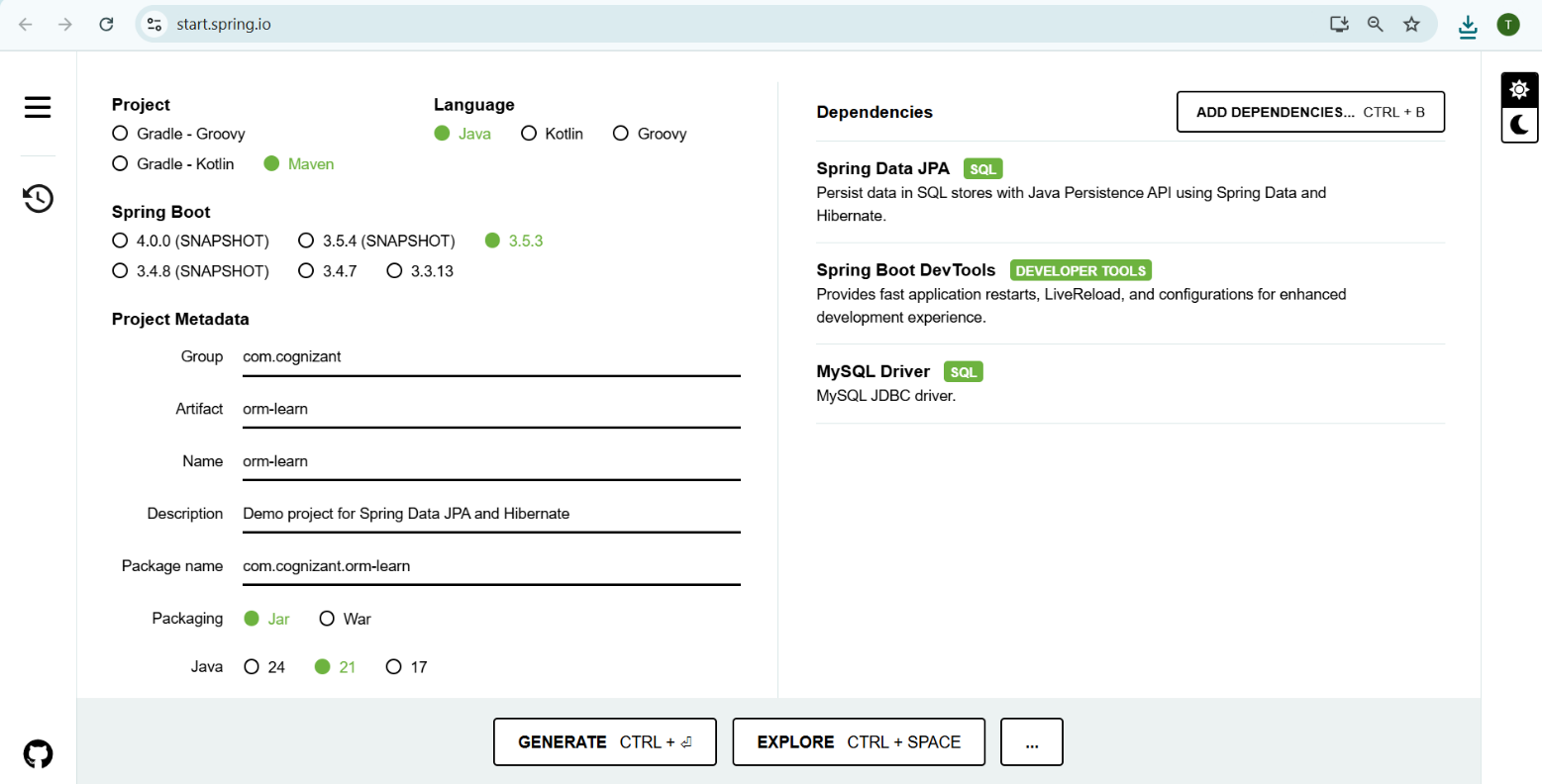
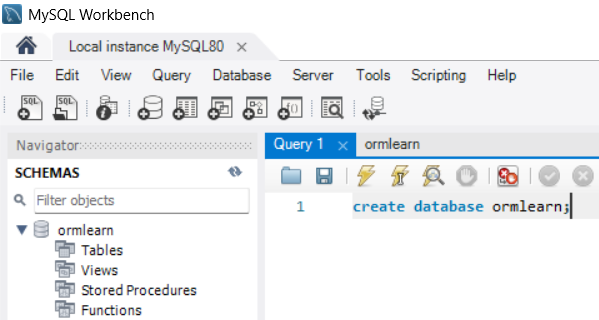
Hands on 1:

**Spring Data JPA - Quick Example**

**1. Created a project using spring initializr**

Import project into IDE

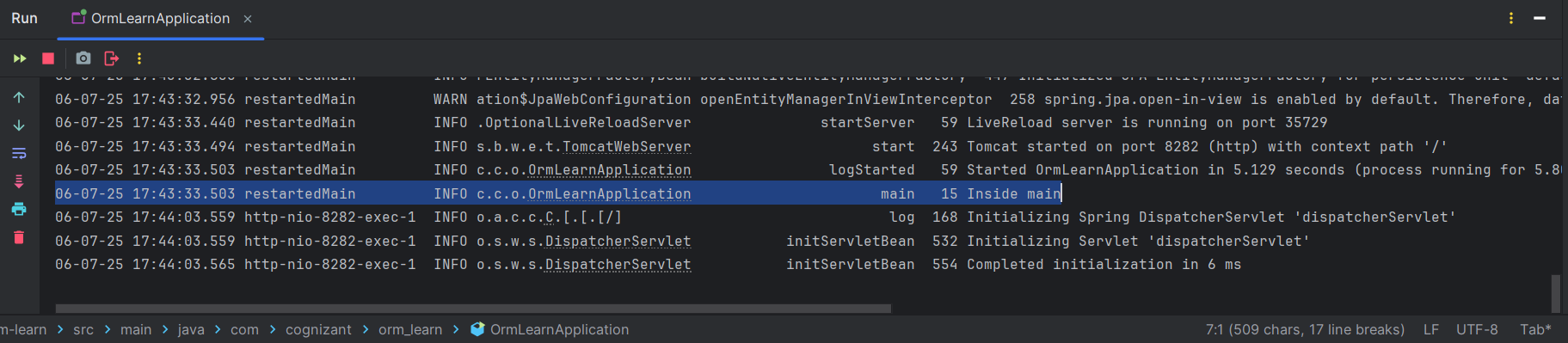
**2. Created MySQL Schema: ormlearn**

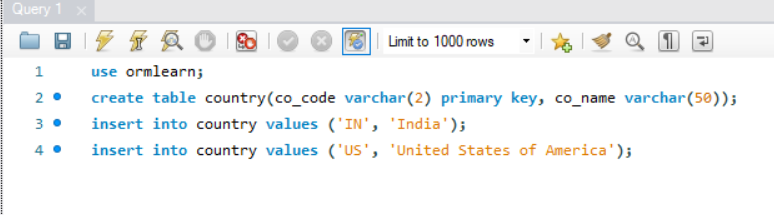
**3. Updated application.properties**

spring.application.name=orm-learn  
  
# Spring Logs  
logging.level.org.springframework=info  
logging.level.com.cognizant=debug  
  
# Hibernate SQL logs  
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 connection  
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=1971  
  
# Hibernate setup  
spring.jpa.hibernate.ddl-auto=validate  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

**4. Include logs for verifying if main() method is called**

package com.cognizant.orm\_learn;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
@SpringBootApplication  
public class OrmLearnApplication {  
  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(OrmLearnApplication.class);  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(OrmLearnApplication.class, args);  
 *LOGGER*.info("Inside main");  
 }  
}

**Output-**

**5. Country table creation in MySQL**

**6. Created Entity Class**

* Create new package com.cognizant.orm-learn.model
* Create Country.java, then generate getters, setters and toString() methods.

Code-

package com.cognizant.orm\_learn.model;  
  
import jakarta.persistence.Column;  
import jakarta.persistence.Entity;  
import jakarta.persistence.Id;  
import jakarta.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  
  
 public String getCode() {  
 return code;  
 }  
  
 public void setCode(String code) {  
 this.code = code;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 // toString()  
 @Override  
 public String toString() {  
 return "Country [code=" + code + ", name=" + name + "]";  
 }  
}

**7. Created Repository Class**

* Create new package com.cognizant.orm-learn.repository
* Create new interface named CountryRepository that extends JpaRepository<Country, String>

Code-

package com.cognizant.orm\_learn.repository;  
  
import com.cognizant.orm\_learn.model.Country;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface CountryRepository extends JpaRepository<Country, String> { }

**8. Created Service Class**

* Create new package com.cognizant.orm-learn.service
* Create new class CountryService

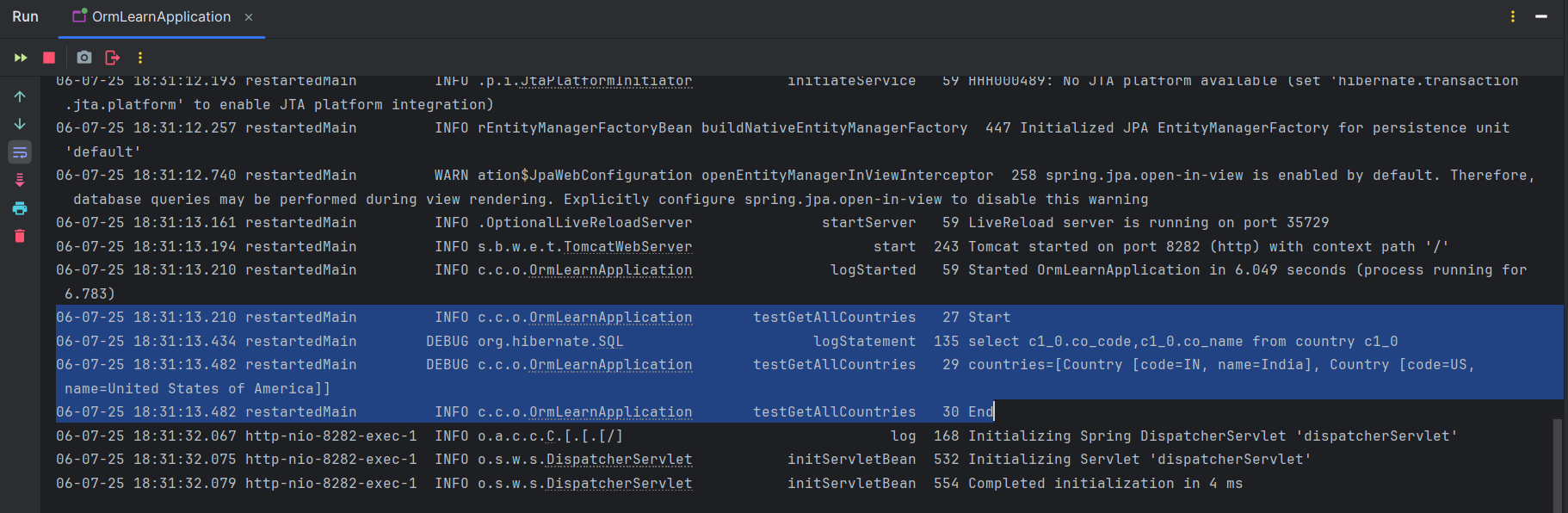
Code-

package com.cognizant.orm\_learn.service;  
  
import com.cognizant.orm\_learn.model.Country;  
import com.cognizant.orm\_learn.repository.CountryRepository;  
import jakarta.transaction.Transactional;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class CountryService {  
  
 @Autowired  
 private CountryRepository countryRepository;  
  
 @Transactional  
 public List<Country> getAllCountries(){  
 return countryRepository.findAll();  
 }  
}

**9. Testing in OrmLearnApplication.java**

Code-

package com.cognizant.orm\_learn;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import com.cognizant.orm\_learn.model.Country;  
import com.cognizant.orm\_learn.service.CountryService;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
  
import java.util.List;  
  
@SpringBootApplication  
public class OrmLearnApplication {  
  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(OrmLearnApplication.class);  
 private static CountryService *countryService*;  
  
 public static void main(String[] args) {  
 ApplicationContext context = SpringApplication.*run*(OrmLearnApplication.class, args);  
 *countryService* = context.getBean(CountryService.class);  
  
 *testGetAllCountries*();  
 }  
  
 private static void testGetAllCountries() {  
 *LOGGER*.info("Start");  
 List<Country> countries = *countryService*.getAllCountries();  
 *LOGGER*.debug("countries={}", countries);  
 *LOGGER*.info("End");  
 }  
  
}

**Output-**

Hands on 4:

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

**Java Persistence API (JPA)**

* JSR 338 Specification for persisting, reading and managing data from Java objects
* Does not contain concrete implementation of the specification
* Hibernate is one of the implementation of JPA

**Hibernate**

* ORM Tool that implements JPA
* Requires more boilerplate code compared to Spring Data JPA.

**Spring Data JPA**

* Does not have JPA implementation, but reduces boiler plate code
* This is another level of abstraction over JPA implementation provider like Hibernate
* Manages transactions

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 addEmployee(Employee employee){

      Session session = factory.openSession();

      Transaction tx = null;

      Integer employeeID = null;

      try {

         tx = session.beginTransaction();

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

         tx.commit();

      } catch (HibernateException e) {

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

         e.printStackTrace();

      } finally {

         session.close();

      }

      return employeeID;

   }

**Spring Data JPA**

**EmployeeRespository.java**

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

**EmployeeService.java**

@Autowire

  private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

  employeeRepository.save(employee);

  }

**Conclusion-**

* Hibernate gives you fine-grained control and flexibility but requires manual session and transaction management.
* Spring Data JPA reduces boilerplate and speeds up development by using built-in repository interfaces.
* In Hibernate, you manage the session (`session.beginTransaction()` / `session.save()` / `session.close()`).
* In Spring Data JPA, you simply autowire a repository and call `repository.save()` without explicitly managing transactions or sessions.