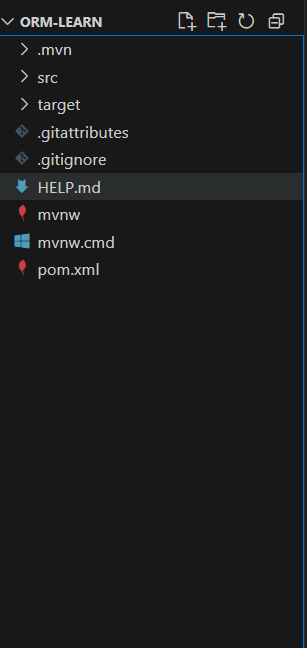
**Week 3-Spring Data JPA with Spring Boot, Hibernate**

**Exercise 1-Spring Data JPA - Quick Example**

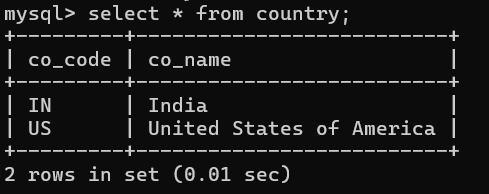
**1.Create a Project using Spring Initializr**

****

|  |
| --- |
| Pom.xml  <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">      <modelVersion>4.0.0</modelVersion>      <parent>          <groupId>org.springframework.boot</groupId>          <artifactId>spring-boot-starter-parent</artifactId>          <version>3.5.3</version>          <relativePath/> <!-- lookup parent from repository -->      </parent>      <groupId>com.cognizant</groupId>      <artifactId>orm-learn</artifactId>      <version>0.0.1-SNAPSHOT</version>      <name>orm-learn</name>      <description>Demo project for Spring Data JPA and Hibernate</description>      <url/>      <licenses>          <license/>      </licenses>      <developers>          <developer/>      </developers>      <scm>          <connection/>          <developerConnection/>          <tag/>          <url/>      </scm>      <properties>          <java.version>17</java.version>      </properties>      <dependencies>          <dependency>              <groupId>org.springframework.boot</groupId>              <artifactId>spring-boot-starter-data-jpa</artifactId>          </dependency>          <dependency>              <groupId>org.springframework.boot</groupId>              <artifactId>spring-boot-devtools</artifactId>              <scope>runtime</scope>              <optional>true</optional>          </dependency>          <dependency>              <groupId>com.mysql</groupId>              <artifactId>mysql-connector-j</artifactId>                <scope>runtime</scope>          </dependency>          <dependency>              <groupId>org.springframework.boot</groupId>              <artifactId>spring-boot-starter-test</artifactId>              <scope>test</scope>          </dependency>      </dependencies>      <build>          <plugins>              <plugin>                  <groupId>org.springframework.boot</groupId>                  <artifactId>spring-boot-maven-plugin</artifactId>              </plugin>          </plugins>      </build>  </project> |

**2. Country table creation**

|  |
| --- |
| mysql> create database ormlearn;  Query OK, 1 row affected (0.09 sec)  mysql> use ormlearn;  Database changed  mysql> create table country(co\_code varchar(2) primary key, co\_name varchar(50));  Query OK, 0 rows affected (0.09 sec)  mysql> insert into country values ('IN', 'India');  Query OK, 1 row affected (0.02 sec)  mysql> insert into country values ('US', 'United States of America');  Query OK, 1 row affected (0.01 sec) |

****

**3.** **Configure application.properties**

|  |
| --- |
| spring.application.name=orm-learn  logging.level.org.springframework=info  logging.level.com.cognizant=debug  logging.level.org.hibernate.SQL=trace  logging.level.org.hibernate.type.descriptor.sql=trace  logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger{25} %25M %4L %m%n  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  spring.jpa.hibernate.ddl-auto=validate  spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect |

**4. Create Model Class**

|  |
| --- |
| Country.java  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="co\_code")      private String code;      @Column(name = "co\_name")      private String name;      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;      }      @Override      public String toString() {          return "Country [code=" + code + ", name=" + name + "]";      }  } |

**5. Create Repository**

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

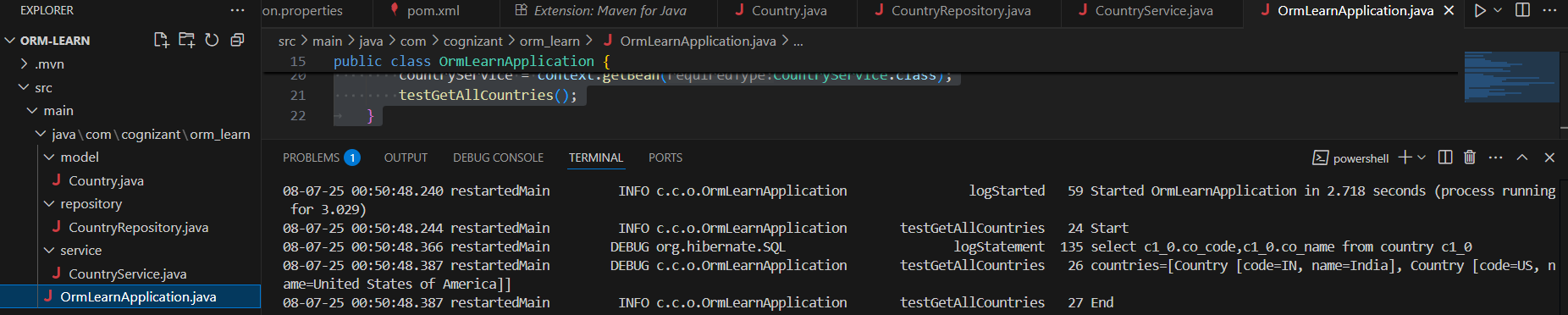
**6.** **Create Service**

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

**7.** **Modify Main Class**

|  |
| --- |
| OrmLearnApplication.java  package com.cognizant.orm\_learn;  import java.util.List;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.context.ApplicationContext;  import com.cognizant.orm\_learn.model.Country;  import com.cognizant.orm\_learn.service.CountryService;  @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:**

****

**Exercise 2: Difference between JPA, Hibernate and Spring Data JPA**

Using Hibernate

1. Create Maven Project  
2. Add Dependencies in pom.xml

|  |
| --- |
| Pom.xml  <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">    <modelVersion>4.0.0</modelVersion>    <groupId>com.hibernate.demo</groupId>    <artifactId>hibernate-demo</artifactId>    <version>1.0-SNAPSHOT</version>    <name>hibernate-demo</name>    <!-- FIXME change it to the project's website -->    <url>http://www.example.com</url>    <properties>      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>      <maven.compiler.source>1.7</maven.compiler.source>      <maven.compiler.target>1.7</maven.compiler.target>    </properties>    <dependencies>      <dependency>        <groupId>junit</groupId>        <artifactId>junit</artifactId>        <version>4.11</version>        <scope>test</scope>      </dependency>      <dependency>          <groupId>org.hibernate</groupId>          <artifactId>hibernate-core</artifactId>          <version>5.6.15.Final</version>      </dependency>        <dependency>          <groupId>mysql</groupId>          <artifactId>mysql-connector-java</artifactId>          <version>8.0.33</version>      </dependency>        <dependency>          <groupId>jakarta.persistence</groupId>          <artifactId>jakarta.persistence-api</artifactId>          <version>2.2.3</version>      </dependency>    </dependencies>    <build>      <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->        <plugins>          <!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean\_Lifecycle -->          <plugin>            <artifactId>maven-clean-plugin</artifactId>            <version>3.1.0</version>          </plugin>          <!-- default lifecycle, jar packaging: see https://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_jar\_packaging -->          <plugin>            <artifactId>maven-resources-plugin</artifactId>            <version>3.0.2</version>          </plugin>          <plugin>            <artifactId>maven-compiler-plugin</artifactId>            <version>3.8.0</version>          </plugin>          <plugin>            <artifactId>maven-surefire-plugin</artifactId>            <version>2.22.1</version>          </plugin>          <plugin>            <artifactId>maven-jar-plugin</artifactId>            <version>3.0.2</version>          </plugin>          <plugin>            <artifactId>maven-install-plugin</artifactId>            <version>2.5.2</version>          </plugin>          <plugin>            <artifactId>maven-deploy-plugin</artifactId>            <version>2.8.2</version>          </plugin>          <!-- site lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#site\_Lifecycle -->          <plugin>            <artifactId>maven-site-plugin</artifactId>            <version>3.7.1</version>          </plugin>          <plugin>            <artifactId>maven-project-info-reports-plugin</artifactId>            <version>3.0.0</version>          </plugin>          <plugin>        <groupId>org.apache.maven.plugins</groupId>        <artifactId>maven-compiler-plugin</artifactId>        <version>3.8.1</version>        <configuration>          <source>17</source>   <!-- or use 11, or 8 depending on your JDK -->          <target>17</target>        </configuration>      </plugin>        </plugins>      </pluginManagement>    </build>  </project> |

3.Create data base

|  |
| --- |
| mysql> create database employee\_db1;  Query OK, 1 row affected (0.01 sec)  mysql> use employee\_db1;  Database changed |

4. Create hibernate configuration

|  |
| --- |
| hibernate.cfg.xml  <!DOCTYPE hibernate-configuration PUBLIC          "-//Hibernate/Hibernate Configuration DTD//EN"          "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">  <hibernate-configuration>      <session-factory>          <property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>          <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/employee\_db1</property>          <property name="hibernate.connection.username">root</property>          <property name="hibernate.connection.password">root</property>            <property name="connection.pool\_size">1</property>            <property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>            <property name="hibernate.hbm2ddl.auto">update</property>            <property name="show\_sql">true</property>            <mapping class="com.hibernate.demo.Employee"/>      </session-factory>  </hibernate-configuration> |

5.Create Employee Entity class

|  |
| --- |
| Employee.java  package com.hibernate.demo;  import javax.persistence.Entity;  import javax.persistence.GeneratedValue;  import javax.persistence.GenerationType;  import javax.persistence.Id;  import javax.persistence.Table;  @Entity  @Table(name = "employee")  public class Employee {      @Id      @GeneratedValue(strategy = GenerationType.IDENTITY)      private Integer id;      private String name;      private String role;      public Employee() {}      public Employee(String name, String role) {          this.name = name;          this.role = role;      }      // Getters and Setters      public Integer getId() { return id; }      public void setId(Integer id) { this.id = id; }      public String getName() { return name; }      public void setName(String name) { this.name = name; }      public String getRole() { return role; }      public void setRole(String role) { this.role = role; }  } |

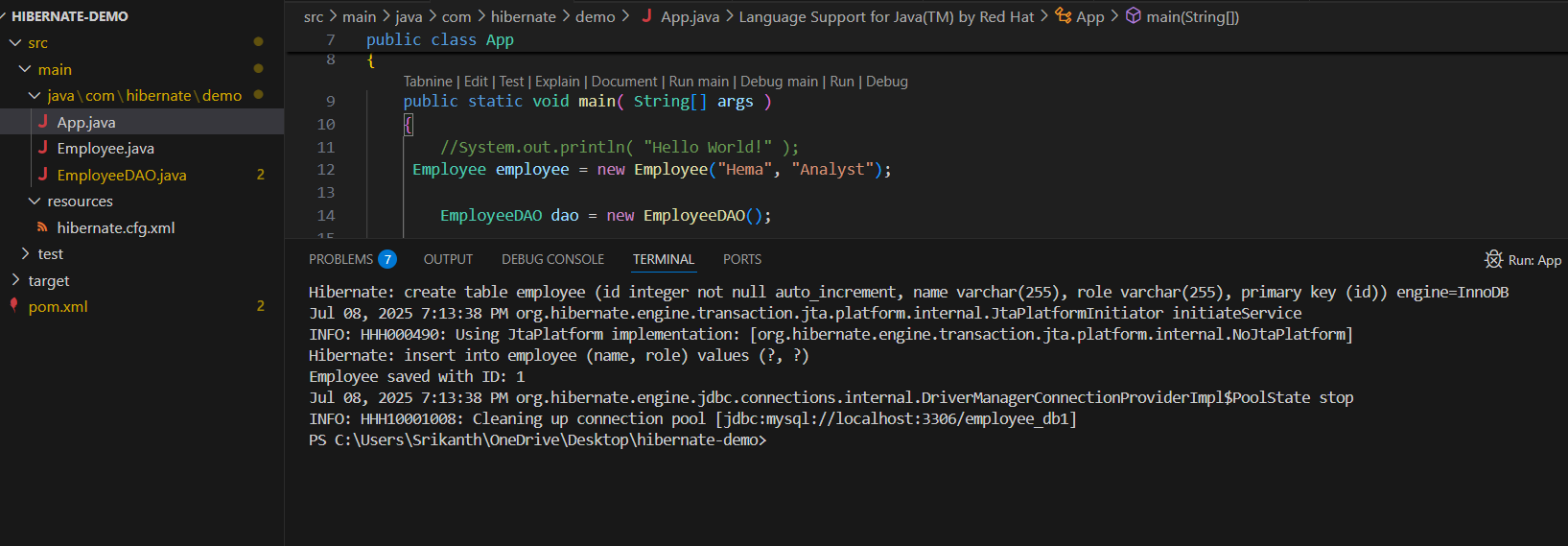
6. Create DAO class

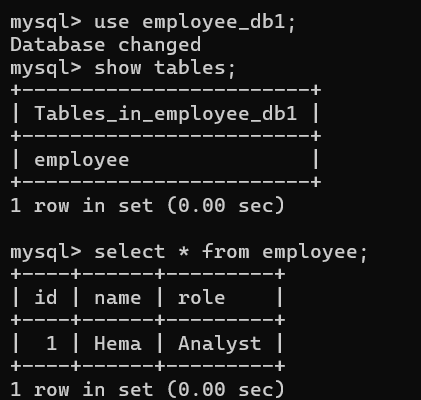
|  |
| --- |
| EmployeeDAO.java  package com.hibernate.demo;  import org.hibernate.HibernateException;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.Transaction;  import org.hibernate.cfg.Configuration;  public class EmployeeDAO {  private static SessionFactory factory;      static {          factory = new Configuration().configure().buildSessionFactory();      }      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;      }      public void shutdown() {          factory.close();      }  } |

7.Update App.java

|  |
| --- |
| package com.hibernate.demo;  public class App  {      public static void main( String[] args )      {       Employee employee = new Employee("Hema", "Analyst");            EmployeeDAO dao = new EmployeeDAO();          Integer id = dao.addEmployee(employee);          System.out.println("Employee saved with ID: " + id);          dao.shutdown();      }  } |

**OUTPUT:**





Using Spring Data JPA

1. Generate Spring Boot Project

2. Create data base & Connect with data base

|  |
| --- |
| mysql> create database employee\_db2;  Query OK, 1 row affected (0.04 sec)  mysql> use employee\_db2;  Database changed |

|  |
| --- |
| application.properties  spring.application.name=demo41  spring.datasource.url=jdbc:mysql://localhost:3306/employee\_db2  spring.datasource.username=root  spring.datasource.password=root  spring.jpa.hibernate.ddl-auto=update  spring.jpa.show-sql=true |

3. Create entity

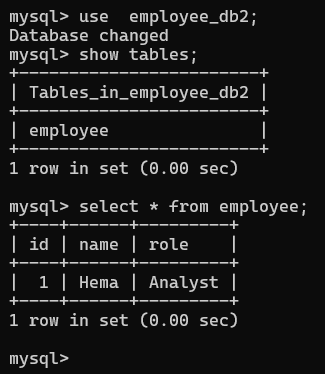
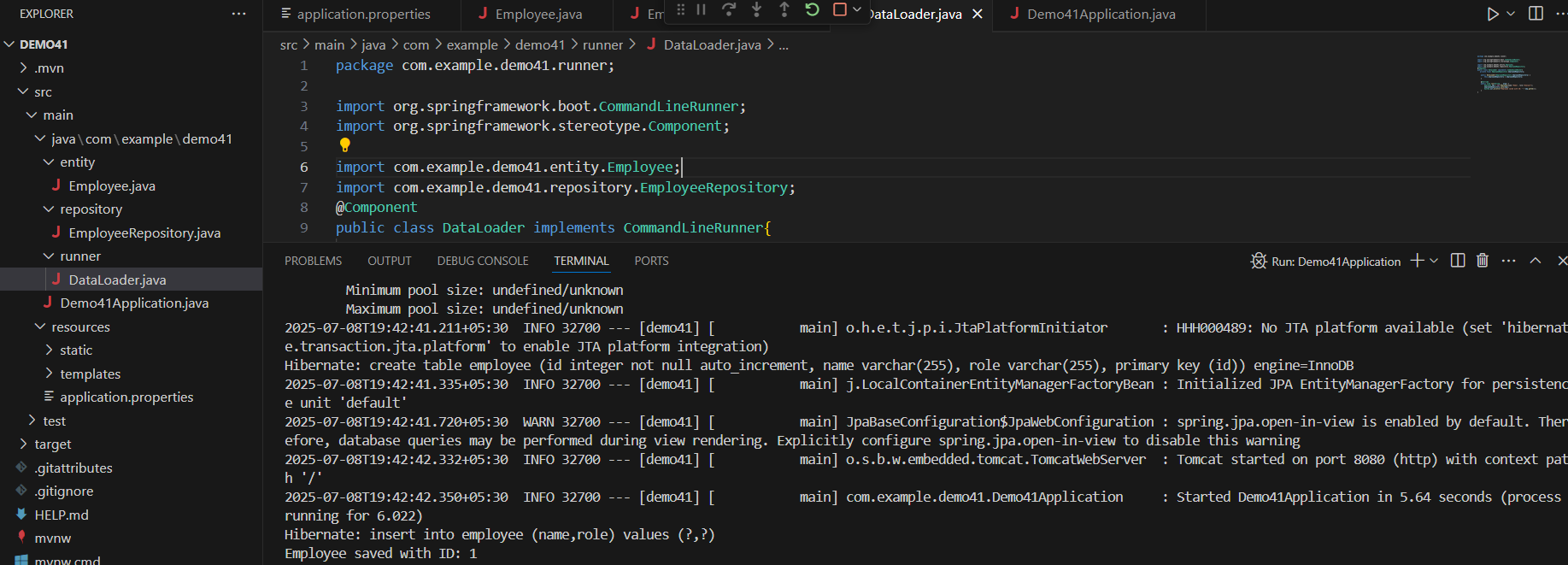
|  |
| --- |
| Employee.java  package com.example.demo41.entity;  import jakarta.persistence.Entity;  import jakarta.persistence.GeneratedValue;  import jakarta.persistence.GenerationType;  import jakarta.persistence.Id;  @Entity  public class Employee {  @Id      @GeneratedValue(strategy = GenerationType.IDENTITY)      private Integer id;      private String name;      private String role;      public Employee() {}      public Employee(String name, String role) {          this.name = name;          this.role = role;      }      public Integer getId() { return id; }      public String getName() { return name; }      public String getRole() { return role; }      public void setId(Integer id) { this.id = id; }      public void setName(String name) { this.name = name; }      public void setRole(String role) { this.role = role; }  } |

4. Create Repository

|  |
| --- |
| EmployeeRepository.java  package com.example.demo41.repository;  import org.springframework.data.jpa.repository.JpaRepository;  import com.example.demo41.entity.Employee;  public interface EmployeeRepository extends JpaRepository<Employee, Integer>{  } |

5.

|  |
| --- |
| DataLoader.java  package com.example.demo41.runner;  import org.springframework.boot.CommandLineRunner;  import org.springframework.stereotype.Component;  import com.example.demo41.entity.Employee;  import com.example.demo41.repository.EmployeeRepository;  @Component  public class DataLoader implements CommandLineRunner{     private final EmployeeRepository employeeRepository;      public DataLoader(EmployeeRepository employeeRepository) {          this.employeeRepository = employeeRepository;      }      @Override      public void run(String... args) {          Employee emp = new Employee("Hema", "Analyst");          employeeRepository.save(emp);          System.out.println("Employee saved with ID: " + emp.getId());      }  } |

**OUTPUT:**

Hibernate lets you connect Java code to the database using objects. Spring Data JPA makes it easier by giving you ready-made methods to save and get data.