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

**Hibernate:**

Code:

package com.cognizant.hibernate;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.HibernateException;

import org.hibernate.cfg.Configuration;

import com.cognizant.hibernate.model.Employee;

public class App {

private static SessionFactory *factory*;

public static void main(String[] args) {

*factory* = new Configuration().configure().buildSessionFactory();

Employee emp = new Employee();

emp.setId(1);

emp.setName("Rohith");

emp.setSalary(65000.0);

Integer empId = *addEmployee*(emp);

System.***out***.println("Inserted Employee with ID: " + empId);

*factory*.close();

}

public static 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;

}

}

<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 5.3//EN"

"http://hibernate.org/dtd/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/hibernate</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">Root@123</property>

<property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>

<property name="hibernate.hbm2ddl.auto">update</property>

<property name="hibernate.show\_sql">true</property>

<mapping class="com.cognizant.hibernate.model.Employee"/>

</session-factory>

</hibernate-configuration>

package com.cognizant.hibernate.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

*@Entity*

public class Employee {

*@Id*

private int id;

private String name;

private double salary;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

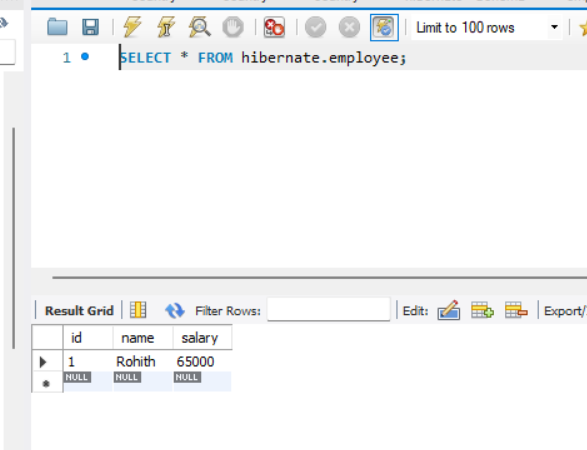
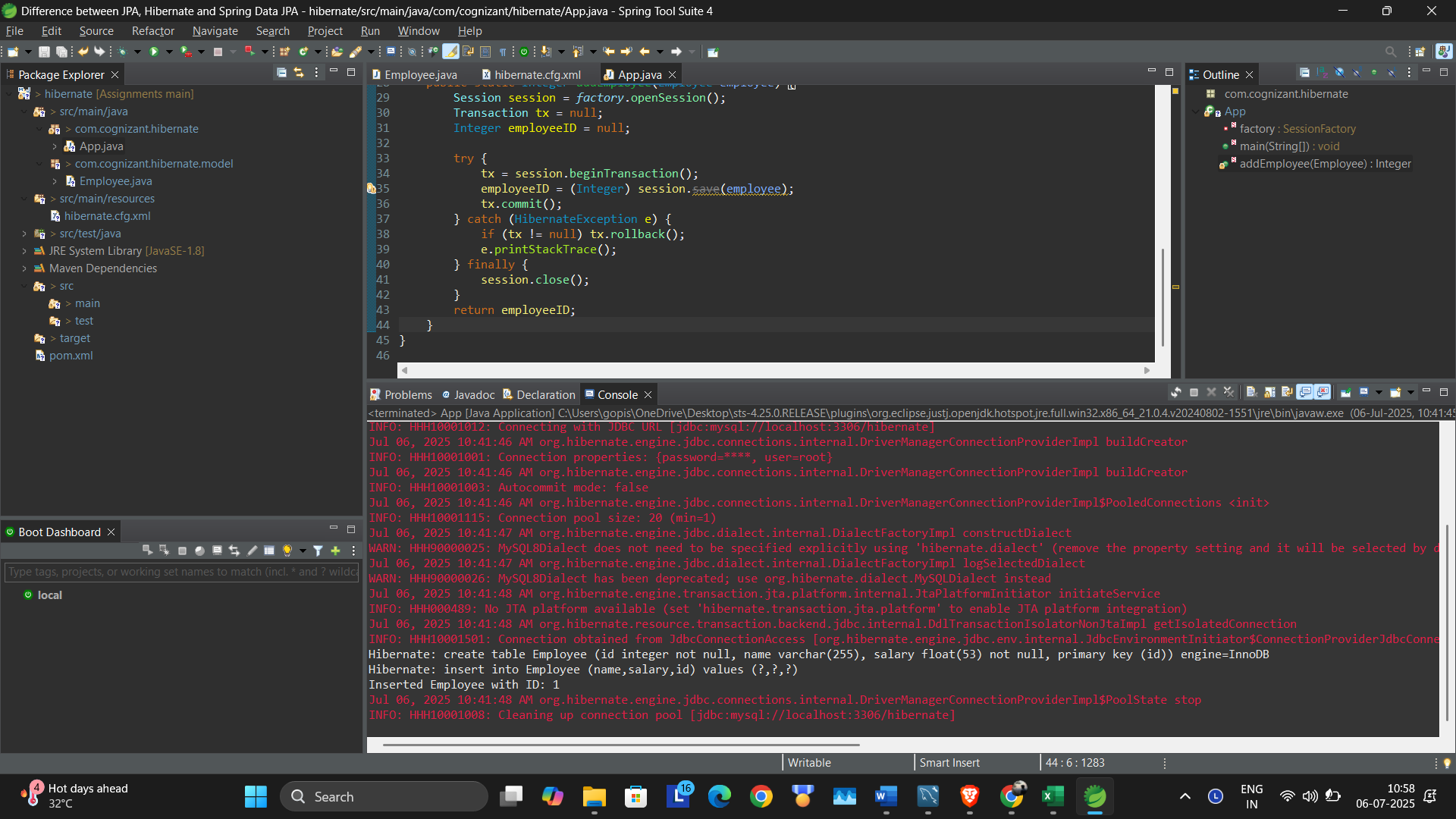
public String toString() {

return "Employee ID"+id+ "\nEmployee Name: "+name+"\nEmployee Salary: "+ salary;

}

}

Outputs:



**Spring Data JPA:**

**Code:**

package com.cognizant;

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.model.Employee;

import com.cognizant.service.EmployeeService;

*@SpringBootApplication*

public class DataJpaApplication {

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

private static EmployeeService *empService*;

public static void main(String[] args) {

SpringApplication.*run*(DataJpaApplication.class, args);

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

*empService* = context.getBean(EmployeeService.class);

***LOGGER***.info("Started");

Employee emp = new Employee();

emp.setId(2);

emp.setName("Gopisetty Rohith");

emp.setSalary(150000);

*empService*.addEmployee(emp);

***LOGGER***.info("END");

}

}

package com.cognizant.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

*@Entity*

public class Employee {

*@Id*

private int id;

private String name;

private double salary;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

public String toString() {

return "Employee ID"+id+ "\nEmployee Name: "+name+"\nEmployee Salary: "+ salary;

}

}

package com.cognizant.repository;

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

import com.cognizant.model.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

package com.cognizant.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.cognizant.model.Employee;

import com.cognizant.repository.EmployeeRepository;

import jakarta.transaction.Transactional;

*@Service*

public class EmployeeService {

*@Autowired*

private EmployeeRepository employeeRepository;

*@Transactional*

public void addEmployee(Employee employee) {

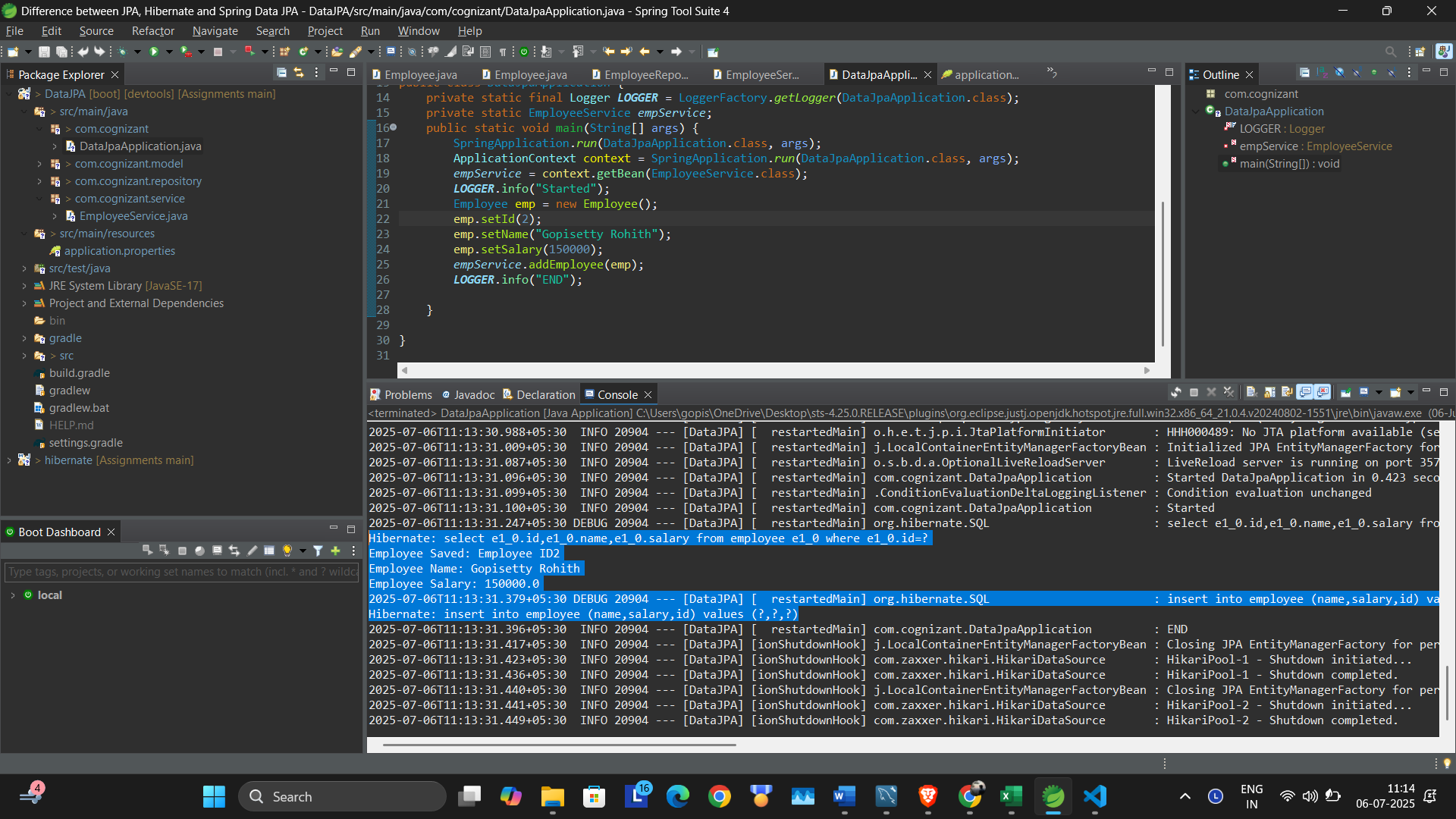
employeeRepository.save(employee);

System.***out***.println("Employee Saved: "+ employee);

}

}

Output:



**Hibernate**

Hibernate is a popular **ORM (Object-Relational Mapping)** tool that directly implements the **Java Persistence API (JPA)** specification. It provides the ability to map Java objects to relational database tables and vice versa.

When using Hibernate, the developer has to manually manage sessions and transactions. For example, you need to open a session using a SessionFactory, begin a transaction, perform database operations (like save() or update()), commit the transaction, and finally close the session. This approach provides **fine-grained control** over the persistence logic, but it also results in **more boilerplate code**.

Hibernate is ideal when you're building an application that needs detailed management of the persistence context or custom optimizations, and when you want to understand how ORM works under the hood.

**Spring Data JPA**

Spring Data JPA is a **high-level abstraction** built on top of JPA. It is part of the larger Spring Data family and is designed to reduce the amount of boilerplate code required to interact with the database.

Spring Data JPA does not implement JPA by itself; instead, it relies on providers like Hibernate. What it offers is a **simplified way to access data**, by generating most of the common CRUD operations for you. You only need to create interfaces that extend JpaRepository, and Spring automatically provides the implementation for methods like save(), findAll(), findById(), and more.

It also handles session and transaction management behind the scenes, so developers can focus on business logic rather than persistence infrastructure. With annotations like @Transactional and simple repository interfaces, data access becomes highly declarative and clean.

Spring Data JPA is the preferred approach for **enterprise application development**, where maintainability, simplicity, and productivity are key.