**Question 1) Simple JPA example**

**EmployeeManagementSystemApplication.java**

package com.example.ems;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class EmployeeManagementSystemApplication {  
  
public static void main(String[] args) {  
SpringApplication.*run*(EmployeeManagementSystemApplication.class, args);  
 }  
  
}

**EmployeeController.java**

package com.example.ems.controller;  
  
import com.example.ems.model.Employee;  
import com.example.ems.repository.EmployeeRepository;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/employees")  
public class EmployeeController {  
  
private final EmployeeRepositoryrepository;  
  
public EmployeeController(EmployeeRepository repository) {  
this.repository= repository;  
 }  
  
@GetMapping  
public List<Employee>getAllEmployees() {  
return repository.findAll();  
 }  
  
@PostMapping  
public Employee createEmployee(@RequestBody Employee employee) {  
return repository.save(employee);  
 }  
}

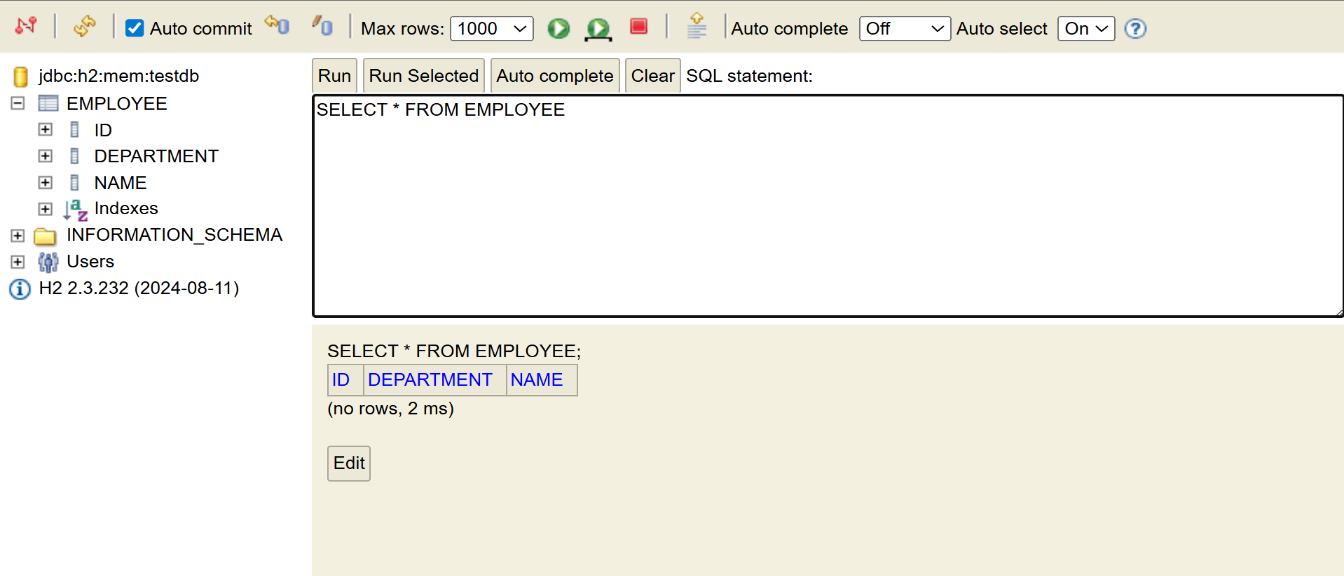
**Employee.java**

package com.example.ems.model;  
  
import jakarta.persistence.\*;  
import lombok.\*;  
  
@Entity  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Employee {  
@Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
private Long id;  
  
private String name;  
private String department;  
}

**EmployeeRepository.java**

package com.example.ems.repository;  
  
import com.example.ems.model.Employee;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface EmployeeRepositoryextends JpaRepository<Employee, Long> {  
}

**Output**



**Question 2) Difference between JPA, Hibernate and Spring Data JPA**

**JPA (Java Persistence API)**

JPA is a **specification** — it defines how Java objects should be mapped to database tables. It gives us annotations like @Entity, @Id, and @OneToMany, and it defines interfaces like EntityManager for basic operations. But JPA doesn’t actually do anything on its own — it's just a set of guidelines. To make it work, you need a concrete implementation (like Hibernate).

### Hibernate

Hibernate is a **JPA implementation** — it takes what JPA defines and brings it to life. It's responsible for generating SQL, managing database sessions, and handling features like caching and lazy loading. It supports all JPA features and also adds some advanced stuff of its own (like its own query language: HQL). When you use JPA in a typical Spring project, Hibernate is usually the engine doing the work behind the scenes.

### Spring Data JPA

Spring Data JPA is a **higher-level abstraction** built on top of JPA and Hibernate. It makes working with databases even easier by removing boilerplate code. Instead of writing your own DAO classes, you just create an interface (like EmployeeRepository) and extend something like JpaRepository. Spring automatically gives you the basic CRUD operations and lets you define queries by method name, so you write a lot less code.