**WEEK 3**

**EXERCISE 10**

**Employee Management System - Hibernate-Specific Features**

**1. Hibernate-Specific Annotations**

**a. Using Hibernate-Specific Annotations to Customize Entity Mappings**

* **Purpose:** Hibernate provides additional annotations beyond the standard JPA annotations, allowing you to customize how your entities are mapped to the database, thereby enhancing performance and functionality.

**Examples of Hibernate-Specific Annotations:**

1. **@Type:**
   * Used to specify a custom Hibernate type for a particular entity attribute.

package com.example.employeemanagementsystem.entity;

import org.hibernate.annotations.Type;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import java.util.UUID;

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@Type(type = "uuid-char")

private UUID employeeCode;

// Getters and Setters

}

1. **@NaturalId:**

Used to specify a unique, non-primary key identifier for your entity that can be used for lookups.

package com.example.employeemanagementsystem.entity;

import org.hibernate.annotations.NaturalId;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NaturalId

private String departmentCode;

private String name;

// Getters and Setters

}

**3.@Formula:**

* Allows you to define a SQL formula that is treated as a read-only property in your entity.

package com.example.employeemanagementsystem.entity;

import org.hibernate.annotations.Formula;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@Formula("(select count(\*) from employee e where e.department\_id = id)")

private int numberOfEmployeesInDepartment;

// Getters and Setters

}

**2. Configuring Hibernate Dialect and Properties**

**a. Configuring Hibernate Dialect and Properties for Optimal Performance**

* **Purpose:** Hibernate requires a dialect to generate SQL optimized for the target database. Additionally, configuring Hibernate properties can significantly impact performance.

**Example: Configuring Hibernate Dialect in application.properties:**

# Hibernate Dialect Configuration for MySQL

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

# Additional Hibernate Properties for Performance Optimization

spring.jpa.properties.hibernate.show\_sql=true

spring.jpa.properties.hibernate.format\_sql=true

spring.jpa.properties.hibernate.use\_sql\_comments=true

spring.jpa.properties.hibernate.cache.use\_second\_level\_cache=true

spring.jpa.properties.hibernate.cache.use\_query\_cache=true

spring.jpa.properties.hibernate.cache.region.factory\_class=org.hibernate.cache.jcache.JCacheRegionFactory

spring.jpa.properties.javax.cache.provider=org.ehcache.jsr107.EhcacheCachingProvider

**3. Batch Processing**

**a. Implementing Batch Processing with Hibernate for Bulk Operations**

* **Purpose:** Batch processing allows you to perform bulk operations efficiently, reducing the number of SQL statements and optimizing performance.

**Steps to Implement Batch Processing:**

1. **Enable Batch Processing in application.properties:**

# Enable Hibernate Batch Processing

spring.jpa.properties.hibernate.jdbc.batch\_size=20

spring.jpa.properties.hibernate.order\_inserts=true

spring.jpa.properties.hibernate.order\_updates=true

2.  **Example: Batch Processing in Code:**

package com.example.employeemanagementsystem.service;

import com.example.employeemanagementsystem.entity.Employee;

import com.example.employeemanagementsystem.repository.EmployeeRepository;

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

import org.springframework.stereotype.Service;

import javax.transaction.Transactional;

import java.util.List;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void saveAllEmployees(List<Employee> employees) {

for (int i = 0; i < employees.size(); i++) {

employeeRepository.save(employees.get(i));

// Flush and clear the session periodically to avoid memory issues

if (i % 20 == 0) {

employeeRepository.flush();

employeeRepository.clear();

}

}

}

}