**Exercise 10: Employee Management System - Hibernate-Specific Features**

**Business Scenario:**

Leverage Hibernate-specific features to enhance your application's performance and capabilities.

**Instructions:**

1. **Hibernate-Specific Annotations:**
   * Use Hibernate-specific annotations to customize entity mappings.
2. **Configuring Hibernate Dialect and Properties:**
   * Configure Hibernate dialect and properties for optimal performance.
3. **Batch Processing:**

Implement batch processing with Hibernate for bulk operations

**Solution**

**Employee.java**

package com.employee.employeemanagement.entity;

import org.hibernate.annotations.Cache;

import org.hibernate.annotations.CacheConcurrencyStrategy;

import org.hibernate.annotations.DynamicInsert;

import org.hibernate.annotations.DynamicUpdate;

import javax.persistence.\*;

@Entity

@Cacheable

@Cache(usage = CacheConcurrencyStrategy.READ\_WRITE)

@DynamicInsert

@DynamicUpdate

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name = "name", nullable = false)

private String name;

@Column(name = "email", nullable = false, unique = true)

private String email;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "department\_id")

private Department department;

}

**application.properties**

# Hibernate Dialect

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

# Enable Hibernate's second-level cache

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

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

# Batch processing settings

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

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

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

**BookService.java**

package com.employee.employeemanagement.service;

import com.employee.employeemanagement.entity.Employee;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import javax.persistence.EntityManager;

import java.util.List;

@Service

public class BatchService {

@Autowired

private EntityManager entityManager;

@Transactional

public void batchInsertEmployees(List<Employee> employees) {

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

entityManager.persist(employees.get(i));

if (i % 20 == 0) { // Batch size is 20

entityManager.flush();

entityManager.clear();

}

}

}

@Transactional

public void batchUpdateEmployees(List<Employee> employees) {

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

entityManager.merge(employees.get(i));

if (i % 20 == 0) { // Batch size is 20

entityManager.flush();

entityManager.clear();

}

}

}

}

**Main.java**

package com.employee.employeemanagement;

import com.employee.employeemanagement.entity.Employee;

import com.employee.employeemanagement.service.BatchService;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import java.util.ArrayList;

import java.util.List;

@Component

public class Main implements CommandLineRunner {

@Autowired

private BatchService batchService;

@Override

public void run(String... args) throws Exception {

// Prepare a list of employees for batch insert

List<Employee> employees = new ArrayList<>();

for (int i = 1; i <= 100; i++) {

Employee employee = new Employee();

employee.setName("Employee " + i);

employee.setEmail("employee" + i + "@example.com");

// Set department if needed

employees.add(employee);

}

// Perform batch insert

batchService.batchInsertEmployees(employees);

// Perform batch update (for demonstration, reusing the same list)

employees.forEach(emp -> emp.setName(emp.getName() + " Updated"));

batchService.batchUpdateEmployees(employees);

}

}