**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.

**1. Hibernate-Specific Annotations**

java

Copy code

import org.hibernate.annotations.GenericGenerator;

import javax.persistence.\*;

@Entity

@Table(name = "employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.AUTO, generator = "uuid-generator")

@GenericGenerator(name = "uuid-generator", strategy = "uuid2")

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

private String id;

@Column(name = "first\_name", nullable = false)

private String firstName;

@Column(name = "last\_name", nullable = false)

private String lastName;

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

private String email;

@Column(name = "salary")

private Double salary;

// Getters and setters

}

**2. Configuring Hibernate Dialect and Properties**

properties

Copy code

# hibernate.cfg.xml

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- JDBC Database connection settings -->

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/your\_database</property>

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

<property name="hibernate.connection.password">password</property>

<!-- Specify dialect -->

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

<!-- Enable Hibernate's automatic session context management -->

<property name="hibernate.current\_session\_context\_class">thread</property>

<!-- Echo all executed SQL to stdout -->

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

<!-- Drop and re-create the database schema on startup -->

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

<!-- Naming strategy for Hibernate -->

<property name="hibernate.physical\_naming\_strategy">org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl</property>

<!-- Batch size -->

<property name="hibernate.jdbc.batch\_size">50</property>

</session-factory>

</hibernate-configuration>

**3. Batch Processing**

java

Copy code

import org.hibernate.Session;

import org.hibernate.Transaction;

import java.util.List;

public class EmployeeBatchProcessor {

public void batchInsertEmployees(List<Employee> employees) {

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction transaction = null;

try {

transaction = session.beginTransaction();

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

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

// Batch process every 50 records

if (i % 50 == 0) {

session.flush();

session.clear();

}

}

transaction.commit();

} catch (Exception e) {

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

}

}

In the batchInsertEmployees method, we use session.flush() and session.clear() to manage the batch processing efficiently. This approach reduces memory consumption and improves performance for bulk insert operations.

Top of Form

Bottom of Form