**Exercise 9: Employee Management System – Customizing Data Source Configuration**

To customize the data source configuration in your Employee Management System and manage multiple data sources, follow these steps:

1. **Spring Boot Auto-Configuration**

Spring Boot simplifies data source configuration with auto-configuration. By default, Spring Boot can auto-configure a DataSource bean based on the properties defined in application.properties or

application.yml.

* 1. **Default Data Source Configuration**

If you only have one data source, you can configure it in the application.properties file: properties

Copy code

spring.datasource.url=jdbc:mysql://localhost:3306/employee\_db

spring.datasource.username=root

spring.datasource.password=yourpassword

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

With this configuration, Spring Boot automatically creates a DataSource bean and uses it for all database operations.

1. **Externalizing Configuration**

Externalizing configuration means moving your configurations, such as database credentials, out of your codebase. This is typically done using application.properties or application.yml files.

* 1. **Externalize Configuration with application.properties**

You can externalize the configuration as shown above in application.properties. Additionally, you can use environment variables or externalized configuration files to override these values at runtime.

For example, you can define placeholders:

properties Copy code

spring.datasource.url=${DB\_URL:jdbc:mysql://localhost:3306/employee\_db} spring.datasource.username=${DB\_USERNAME:root}

spring.datasource.password=${DB\_PASSWORD:yourpassword}

You can then provide the actual values via environment variables or an external properties file, allowing different environments (dev, test, prod) to use different configurations.

1. **Managing Multiple Data Sources**

In some applications, you may need to manage multiple data sources, such as one for employee data and another for department data.

* 1. **Define Multiple Data Sources in application.properties**

First, define properties for each data source in application.properties: properties

Copy code

# Primary Data Source (Employee)

spring.datasource.employee.url=jdbc:mysql://localhost:3306/employee\_db spring.datasource.employee.username=root

spring.datasource.employee.password=yourpassword

spring.datasource.employee.driver-class-name=com.mysql.cj.jdbc.Driver # Secondary Data Source (Department)

spring.datasource.department.url=jdbc:mysql://localhost:3306/department\_db spring.datasource.department.username=root

spring.datasource.department.password=yourpassword

spring.datasource.department.driver-class-name=com.mysql.cj.jdbc.Driver

* 1. **Create Configuration Classes for Multiple Data Sources**

Next, create configuration classes to define the DataSource beans:

import org.springframework.boot.context.properties.ConfigurationProperties; import org.springframework.boot.jdbc.DataSourceBuilder;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Primary;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import org.springframework.orm.jpa.JpaTransactionManager;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;

import org.springframework.transaction.PlatformTransactionManager;

import javax.persistence.EntityManagerFactory;

import javax.sql.DataSource;

@Configuration

@EnableJpaRepositories(

basePackages = "com.example.employee.repository", // Specify the repository package for employee

entityManagerFactoryRef = "employeeEntityManagerFactory", transactionManagerRef = "employeeTransactionManager"

)

public class EmployeeDataSourceConfig {

@Primary

@Bean(name = "employeeDataSource")

@ConfigurationProperties(prefix = "spring.datasource.employee")

public DataSource employeeDataSource() {

return DataSourceBuilder.create().build();

}

@Primary

@Bean(name = "employeeEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean employeeEntityManagerFactory() { LocalContainerEntityManagerFactoryBean em = new

LocalContainerEntityManagerFactoryBean();

em.setDataSource(employeeDataSource());

em.setPackagesToScan(new String[]{"com.example.employee.entity"}); // Entity package

HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter(); em.setJpaVendorAdapter(vendorAdapter);

return em;

}

@Primary

@Bean(name = "employeeTransactionManager")

public PlatformTransactionManager employeeTransactionManager(EntityManagerFactory employeeEntityManagerFactory) {

return new JpaTransactionManager(employeeEntityManagerFactory);

}

}

Then, create a similar configuration for the Department data source: java

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.boot.jdbc.DataSourceBuilder;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import org.springframework.orm.jpa.JpaTransactionManager;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;

import org.springframework.transaction.PlatformTransactionManager;

import javax.persistence.EntityManagerFactory;

import javax.sql.DataSource;

@Configuration

@EnableJpaRepositories(

basePackages = "com.example.department.repository", // Specify the repository package for department

entityManagerFactoryRef = "departmentEntityManagerFactory", transactionManagerRef = "departmentTransactionManager"

)

public class DepartmentDataSourceConfig {

@Bean(name = "departmentDataSource")

@ConfigurationProperties(prefix = "spring.datasource.department") public DataSource departmentDataSource() {

return DataSourceBuilder.create().build();

}

@Bean(name = "departmentEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean departmentEntityManagerFactory() { LocalContainerEntityManagerFactoryBean em = new

LocalContainerEntityManagerFactoryBean();

em.setDataSource(departmentDataSource());

em.setPackagesToScan(new String[]{"com.example.department.entity"}); // Entity package

HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter(); em.setJpaVendorAdapter(vendorAdapter);

return em;

}

@Bean(name = "departmentTransactionManager")

public PlatformTransactionManager departmentTransactionManager(EntityManagerFactory departmentEntityManagerFactory) {

return new JpaTransactionManager(departmentEntityManagerFactory);

}

}

**4.Using Multiple Data Sources in Your Application**

With the above configurations, Spring Boot knows which DataSource to use for each set of

repositories and entities. The @Primary annotation is used to indicate the primary DataSource when only one is needed in a particular context.