### 1. Spring Boot Auto-Configuration

Spring Boot provides auto-configuration for a single data source out of the box. To customize and manage multiple data sources, you will need to configure each data source manually and define how they should be used.

#### 1.1 Define Multiple Data Sources

To manage multiple data sources, you need to:

* Configure each data source.
* Set up JPA configurations for each data source.
* Define transaction managers for each data source.

**Example Setup:**

**Add Dependencies:** Make sure you have the necessary dependencies in your pom.xml or build.gradle.

<!-- Spring Data JPA -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

**Define Data Sources in application.properties:**

# Primary DataSource

spring.datasource.primary.url=jdbc:h2:mem:primarydb

spring.datasource.primary.username=sa

spring.datasource.primary.password=password

spring.datasource.primary.driver-class-name=org.h2.Driver

# Secondary DataSource

spring.datasource.secondary.url=jdbc:h2:mem:secondarydb

spring.datasource.secondary.username=sa

spring.datasource.secondary.password=password

spring.datasource.secondary.driver-class-name=org.h2.Driver

**Create Configuration Classes for Each Data Source:**

Define configuration classes for each data source, JPA entity manager, and transaction manager.

**PrimaryDataSourceConfig.java:**

package com.example.employeemanagementsystem.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

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

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import org.springframework.orm.jpa.JpaTransactionManager;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

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

import javax.persistence.EntityManagerFactory;

import javax.sql.DataSource;

@Configuration

@EnableJpaRepositories(

basePackages = "com.example.employeemanagementsystem.repository.primary",

entityManagerFactoryRef = "primaryEntityManagerFactory",

transactionManagerRef = "primaryTransactionManager"

)

public class PrimaryDataSourceConfig {

@Bean(name = "primaryDataSource")

public DataSource dataSource() {

DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setDriverClassName("org.h2.Driver");

dataSource.setUrl("jdbc:h2:mem:primarydb");

dataSource.setUsername("sa");

dataSource.setPassword("password");

return dataSource;

}

@Bean(name = "primaryEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean entityManagerFactory(

@Qualifier("primaryDataSource") DataSource dataSource) {

LocalContainerEntityManagerFactoryBean emf = new LocalContainerEntityManagerFactoryBean();

emf.setDataSource(dataSource);

emf.setPackagesToScan("com.example.employeemanagementsystem.model.primary");

emf.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

return emf;

}

@Bean(name = "primaryTransactionManager")

public JpaTransactionManager transactionManager(

@Qualifier("primaryEntityManagerFactory") EntityManagerFactory emf) {

return new JpaTransactionManager(emf);

}

**}**

**SecondaryDataSourceConfig.java:**

package com.example.employeemanagementsystem.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

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

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import org.springframework.orm.jpa.JpaTransactionManager;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

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

import javax.persistence.EntityManagerFactory;

import javax.sql.DataSource;

@Configuration

@EnableJpaRepositories(

basePackages = "com.example.employeemanagementsystem.repository.secondary",

entityManagerFactoryRef = "secondaryEntityManagerFactory",

transactionManagerRef = "secondaryTransactionManager"

)

public class SecondaryDataSourceConfig {

@Bean(name = "secondaryDataSource")

public DataSource dataSource() {

DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setDriverClassName("org.h2.Driver");

dataSource.setUrl("jdbc:h2:mem:secondarydb");

dataSource.setUsername("sa");

dataSource.setPassword("password");

return dataSource;

}

@Bean(name = "secondaryEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean entityManagerFactory(

@Qualifier("secondaryDataSource") DataSource dataSource) {

LocalContainerEntityManagerFactoryBean emf = new LocalContainerEntityManagerFactoryBean();

emf.setDataSource(dataSource);

emf.setPackagesToScan("com.example.employeemanagementsystem.model.secondary");

emf.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

return emf;

}

@Bean(name = "secondaryTransactionManager")

public JpaTransactionManager transactionManager(

@Qualifier("secondaryEntityManagerFactory") EntityManagerFactory emf) {

return new JpaTransactionManager(emf);

}

}

### Externalizing Configuration

Externalizing configuration allows you to separate your configuration settings from your code, making them easier to manage and modify.

#### 2.1 Update application.properties

In your application.properties or application.yml, define the settings for multiple data sources.

**Example in** application.properties**:**

# Primary DataSource

spring.datasource.primary.url=jdbc:h2:mem:primarydb

spring.datasource.primary.username=sa

spring.datasource.primary.password=password

spring.datasource.primary.driver-class-name=org.h2.Driver

# Secondary DataSource

spring.datasource.secondary.url=jdbc:h2:mem:secondarydb

spring.datasource.secondary.username=sa

spring.datasource.secondary.password=password

spring.datasource.secondary.driver-class-name=org.h2.Driver

#### 2.2 Accessing Configuration Properties

You can use the @Value annotation or @ConfigurationProperties to inject configuration properties into your configuration classes.

**Example using** @Value**:**

package com.example.employeemanagementsystem.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import javax.sql.DataSource;

@Configuration

public class DataSourceConfig {

@Value("${spring.datasource.primary.url}")

private String primaryUrl;

@Value("${spring.datasource.primary.username}")

private String primaryUsername;

@Value("${spring.datasource.primary.password}")

private String primaryPassword;

@Value("${spring.datasource.primary.driver-class-name}")

private String primaryDriverClassName;

@Bean(name = "primaryDataSource")

public DataSource primaryDataSource() {

DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setDriverClassName(primaryDriverClassName);

dataSource.setUrl(primaryUrl);

dataSource.setUsername(primaryUsername);

dataSource.setPassword(primaryPassword);

return dataSource;

}

// Define secondary data source similarly

}