Step 1: Leverage Spring Boot Auto-Configuration for Data Sources

Spring Boot provides auto-configuration for data sources. We can leverage this feature to configure our data sources.

application.properties:

spring.datasource.url=jdbc:h2:mem:employee\_management\_system

spring.datasource.username=sa

spring.datasource.password=

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

spring.datasource.employee.url=jdbc:h2:mem:employee\_db

spring.datasource.employee.username=sa

spring.datasource.employee.password=

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

spring.datasource.department.url=jdbc:h2:mem:department\_db

spring.datasource.department.username=sa

spring.datasource.department.password=

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

Step 2: Externalize Configuration with application.properties

We can externalize the configuration by using separate property files for each data source.

application-employee.properties:

spring.datasource.url=jdbc:h2:mem:employee\_db

spring.datasource.username=sa

spring.datasource.password=

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

application-department.properties:

spring.datasource.url=jdbc:h2:mem:department\_db

spring.datasource.username=sa

spring.datasource.password=

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

Step 3: Manage Multiple Data Sources within Your Application

We can manage multiple data sources within our application by creating separate configuration classes for each data source.

EmployeeDataSourceConfig.java:

@Configuration

@PropertySource("classpath:application-employee.properties")

public class EmployeeDataSourceConfig {

@Bean

@Primary

public DataSource employeeDataSource() {

return DataSourceBuilder.create()

.driverClassName("org.h2.Driver")

.url("jdbc:h2:mem:employee\_db")

.username("sa")

.password("")

.build();

}

}

DepartmentDataSourceConfig.java:

@Configuration

@PropertySource("classpath:application-department.properties")

public class DepartmentDataSourceConfig {

@Bean

public DataSource departmentDataSource() {

return DataSourceBuilder.create()

.driverClassName("org.h2.Driver")

.url("jdbc:h2:mem:department\_db")

.username("sa")

.password("")

.build();

}

}

Step 4: Use Qualifiers to Inject Data Sources

We can use qualifiers to inject the correct data source into our repositories.

EmployeeRepository.java:

@Repository

public class EmployeeRepository {

@Autowired

@Qualifier("employeeDataSource")

private DataSource dataSource;

// ...

}

DepartmentRepository.java:

@Repository

public class DepartmentRepository {

@Autowired

@Qualifier("departmentDataSource")

private DataSource dataSource;

// ...

}