**Spring with Multiple DB Connection**

When connecting two databases with the Springboot Application, there are many steps which are included in the process. If there are two databases, then there will be two datasources which should be connected to them. The object of datasources is injected to Entity Manager and the object of Entity Manager is injected to TransactionManager.

A diagram of a software company

Description automatically generated

Demo Project was done to illustrate the two different database connections to a single Spring boot Application.

**Dependencies used for the Project:**

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

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

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

**Application Properties file:**

# MySQL-User

spring.user.datasource.url=jdbc: mysql://localhost:3306/userdb

spring.user.datasource.username=root

spring.user.datasource.password=root

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

spring.jpa.show-sql=true

# MySQL-Book

spring.book.datasource.url=jdbc: mysql://localhost:3306/bookdb

spring.book.datasource.username=root

spring.book.datasource.password=root

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

server.port=9090

In our project, there are two entity classes and their two repositories which corresponds two databases with one table at each database. The Database config classes are written for connecting two databases. The two config classes are BookDBConfig Class and UserDBConfig Class

**BookDBConfig Class:**

EntityManagerFactory: JpaTransactionManager requires an EntityManagerFactory as input. The EntityManagerFactory is responsible for creating EntityManager instances, which are used for interacting with the JPA provider (like Hibernate) and managing entities.

Transaction Management: JpaTransactionManager starts and manages transactions. It participates in the Spring transaction management infrastructure, allowing you to use declarative transaction management with annotations like @Transactional.

The @EnableTransactionManagement annotation is used to enable transaction management in Spring applications.

The @Configuration annotation is used to indicate that a class provides bean definitions to the Spring application context.

The @EnableJpaRepositories annotation is used to enable JPA repository support in a Spring application. It tells Spring to scan specified packages for interfaces that extend JpaRepository and create Spring Data repository beans for them.

The names which are specified in the @EnableJpaRepositories should matched with the method names.

@EnableTransactionManagement

@EnableJpaRepositories (

entityManagerFactoryRef = "bookEntityManagerFactory",

transactionManagerRef = "bookTransactionManager",

basePackages = {

"com.demo.book.repository"

}

}

The bean name is given as “bookDataSource” for this method. The values from the application properties file are fetched through the instance of Environment env. The env.getProperty() method will give the values of the application properties and the parameters in this method should match with that values of application properties.

@Autowired

private Environment env;

@Primary

@Bean (name = "bookDataSource")

public DataSource dataSource() {

return DataSourceBuilder.create().url(env. getProperty("spring.book.datasource.url"))

. driverClassName(env. getProperty("spring.book.datasource.driver-class-name"))

. username(env.getProperty("spring.book.datasource.username"))

. password (env. getProperty("spring.book.datasource.password")).build();

}

The bean object bookDataSource is injected in the bookEntityManagerFactory method.We are specifying the properties and packages that it should scan in this method

@Primary

@Bean (name = "bookEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean bookEntityManagerFactory(EntityManagerFactoryBuilder builder,

@Qualifier("bookDataSource") DataSource dataSource) {

HashMap<String, Object> properties = new HashMap<> ();

properties.put("hibernate.hbm2ddl.auto", "update");

return builder.dataSource(dataSource)

. properties(properties)

. packages("com.demo.book.model")

. persistence Unit("book”) //set of entity classes which are mapped to database.

. build ();

}

The object bookEntityManagerFactory method is injected in the bookTransactionManager method.

@Primary

@Bean (name = "bookTransactionManager")

public PlatformTransactionManager bookTransactionManager(@Qualifier("bookEntityManagerFactory") EntityManagerFactory bookEntityManagerFactory) {

return new JpaTransactionManager(bookEntityManagerFactory); //Implmentation of PlatformTransactionManager,used to manage transactions when working with JPA-based data access in a Spring application.

}

**UserDBConfig Class:**

The @EnableJpaRepositories annotation is used to enable JPA repository support in a Spring application. It tells Spring to scan specified packages for interfaces that extend JpaRepository and create Spring Data repository beans for them.

The names which are specified in the @EnableJpaRepositories should matched with the method names.

@Configuration

@EnableTransactionManagement

@EnableJpaRepositories (

entityManagerFactoryRef = "entityManagerFactory",

basePackages = {

"com.demo.user.repository"

},

transactionManagerRef="transactionManager"

)

The bean name is given as “userDataSource” for this method. The values from the application properties file are fetched through the instance of Environment env. The env.getProperty() method will give the values of the application properties and the parameters in this method should match with that values of application properties.

private Environment env;

@Bean (name = "userDataSource")

public DataSource dataSource() {

return DataSourceBuilder.create().url(env. getProperty("spring.user.datasource.url"))

. driverClassName(env. getProperty("spring.user.datasource.driver-class-name"))

. username(env.getProperty("spring.user.datasource.username"))

. password (env. getProperty("spring.user.datasource.password")).build();

}

The bean object userDataSource is injected in the entityManagerFactory method. We are specifying the properties and packages that it should scan in this method.

@Bean (name = "entityManagerFactory")

public LocalContainerEntityManagerFactoryBean entityManagerFactory(EntityManagerFactoryBuilder builder,

@Qualifier("userDataSource") DataSource dataSource) {

HashMap<String, Object> properties = new HashMap<> ();

properties.put("hibernate.hbm2ddl.auto", "update");

return builder.dataSource(dataSource)

. properties(properties)

. packages("com.demo.user.model")

. persistenceUnit("user")

. build ();

}

The object entityManagerFactory method is injected in the transactionManager method.

@Bean (name = "transactionManager")

public PlatformTransactionManager transactionManager(@Qualifier("entityManagerFactory") EntityManagerFactory entityManagerFactory) {

return new JpaTransactionManager(entityManagerFactory);

}

DemoRestController Class:

Whenever we run the application, The database will be connected and when we hit the /addData endpoint, The data will be saved. When we hit the /getUsers endpoint,The list of users which are saved in the table will be displayed. When we hit the /getBooks endpoint, The list of books which are saved in the table will be displayed.

@RestController

public class DemoRestController {

@Autowired

private BookRepository bookRepository;

@Autowired

private UserRepository userRepository;

@GetMapping("/addData")

public String addData2DB () {

userRepository.saveAll(Stream.of(new User (744, "John"), new User (455, "Smith")). collect (Collectors.toList()));

bookRepository.saveAll(Stream.of(new Book (121, "HarryPotter"), new Book (223, "Rich Dad, Poor Dad”). collect (Collectors.toList()));

return "Data Added Successfully";

}

@GetMapping("/getUsers")

public List<User> getUsers() {

return userRepository.findAll();

}

@GetMapping("/getBooks")

public List<Book> getBooks() {

return bookRepository.findAll();

}