Sure! Below is an example of a Spring Boot application that connects to both Oracle and PostgreSQL databases. This example includes the necessary configurations and a simple Order API.

**1. Project Setup**

Add the required dependencies in your pom.xml:

<dependencies>

<!-- Spring Boot Starter Data JPA -->

<dependency>

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

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

</dependency>

<!-- Oracle JDBC Driver -->

<dependency>

<groupId>com.oracle.database.jdbc</groupId>

<artifactId>ojdbc8</artifactId>

<scope>runtime</scope>

</dependency>

<!-- PostgreSQL JDBC Driver -->

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Starter Web -->

<dependency>

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

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

</dependency>

</dependencies>

**2. Configuration**

Create a configuration class to define the data sources:

import javax.sql.DataSource;

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

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.orm.jpa.JpaTransactionManager;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

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

import org.springframework.transaction.PlatformTransactionManager;

@Configuration

public class DataSourceConfig {

@Primary

Source() {

return DataSourceBuilder.create().build();

}

@Bean(name = "postgresDataSource")

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

public DataSource postgresDataSource() {

return DataSourceBuilder.create().build();

}

@Primary

@Bean(name = "oracleEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean oracleEntityManagerFactory(

@Qualifier("oracleDataSource") DataSource dataSource) {

LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();

em.setDataSource(dataSource);

em.setPackagesToScan("com.example.orderapi.oracle");

em.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

return em;

}

@Bean(name = "postgresEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean postgresEntityManagerFactory(

@Qualifier("postgresDataSource") DataSource dataSource) {

LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();

em.setDataSource(dataSource);

em.setPackagesToScan("com.example.orderapi.postgres");

em.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

return em;

}

@Primary

@Bean(name = "oracleTransactionManager")

public PlatformTransactionManager oracleTransactionManager(

@Qualifier("oracleEntityManagerFactory") LocalContainerEntityManagerFactoryBean entityManagerFactory) {

return new JpaTransactionManager(entityManagerFactory.getObject());

}

@Bean(name = "postgresTransactionManager")

public PlatformTransactionManager postgresTransactionManager(

@Qualifier("postgresEntityManagerFactory") LocalContainerEntityManagerFactoryBean entityManagerFactory) {

return new JpaTransactionManager(entityManagerFactory.getObject());

}

}

**3. Application Properties**

Define the data source properties in application.properties:

# Oracle Data Source

spring.datasource.oracle.url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.oracle.username=oracleuser

spring.datasource.oracle.password=oraclepassword

spring.datasource.oracle.driver-class-name=oracle.jdbc.OracleDriver

# PostgreSQL Data Source

spring.datasource.postgres.url=jdbc:postgresql://localhost:5432/postgresdb

spring.datasource.postgres.username=postgresuser

spring.datasource.postgres.password=postgrespassword

spring.datasource.postgres.driver-class-name=org.postgresql.Driver

**4. Entity Classes**

Create entity classes for each database:

**Oracle Entity:**

package com.example.orderapi.oracle;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name = "orders")

public class OracleOrder {

@Id

private Long id;

private String description;

// Getters and Setters

}

**PostgreSQL Entity:**

package com.example.orderapi.postgres;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name = "orders")

public class PostgresOrder {

@Id

private Long id;

private String description;

// Getters and Setters

}

**5. Repositories**

Create repositories for each database:

**Oracle Repository:**

package com.example.orderapi.oracle;

import org.springframework.data.jpa.repository.JpaRepository;

public interface OracleOrderRepository extends JpaRepository<OracleOrder, Long> {

}

**PostgreSQL Repository:**

package com.example.orderapi.postgres;

import org.springframework.data.jpa.repository.JpaRepository;

public interface PostgresOrderRepository extends JpaRepository<PostgresOrder, Long> {

}

**6. Service Layer**

Create a service layer to handle business logic:

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

@Service

public class OrderService {

@Autowired

private OracleOrderRepository oracleOrderRepository;

@Autowired

private PostgresOrderRepository postgresOrderRepository;

@Transactional("oracleTransactionManager")

public OracleOrder saveOrderToOracle(OracleOrder order) {

return oracleOrderRepository.save(order);

}

@Transactional("postgresTransactionManager")

public PostgresOrder saveOrderToPostgres(PostgresOrder order) {

return postgresOrderRepository.save(order);

}

}

**7. Controller**

Create a controller to expose the API endpoints:

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

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/orders")

public class OrderController {

@Autowired

private OrderService orderService;

@PostMapping("/oracle")

public OracleOrder createOrderInOracle(@RequestBody OracleOrder order) {

return orderService.saveOrderToOracle(order);

}

@PostMapping("/postgres")

public PostgresOrder createOrderInPostgres(@RequestBody PostgresOrder order) {

return orderService.saveOrderToPostgres(order);

}

}

This setup will allow your Spring Boot application to connect to both Oracle and PostgreSQL databases and handle orders in both. If you have any specific requirements or need further customization, feel free to ask!

Sure! Below is an updated version of the Spring Boot application using Spring Framework 6 and Spring Boot 3.0, which connects to both Oracle and PostgreSQL databases.

**1. Project Setup**

Add the required dependencies in your pom.xml:

<dependencies>

<!-- Spring Boot Starter Data JPA -->

<dependency>

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

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

</dependency>

<!-- Oracle JDBC Driver -->

<dependency>

<groupId>com.oracle.database.jdbc</groupId>

<artifactId>ojdbc8</artifactId>

<scope>runtime</scope>

</dependency>

<!-- PostgreSQL JDBC Driver -->

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Starter Web -->

<dependency>

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

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

</dependency>

</dependencies>

**2. Configuration**

Create a configuration class to define the data sources:

import javax.sql.DataSource;

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

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.orm.jpa.JpaTransactionManager;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

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

import org.springframework.transaction.PlatformTransactionManager;

@Configuration

public class DataSourceConfig {

@Primary

@Bean(name = "oracleDataSource")

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

public DataSource oracleDataSource() {

return DataSourceBuilder.create().build();

}

@Bean(name = "postgresDataSource")

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

public DataSource postgresDataSource() {

return DataSourceBuilder.create().build();

}

@Primary

@Bean(name = "oracleEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean oracleEntityManagerFactory(

@Qualifier("oracleDataSource") DataSource dataSource) {

LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();

em.setDataSource(dataSource);

em.setPackagesToScan("com.example.orderapi.oracle");

em.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

return em;

}

@Bean(name = "postgresEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean postgresEntityManagerFactory(

@Qualifier("postgresDataSource") DataSource dataSource) {

LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();

em.setDataSource(dataSource);

em.setPackagesToScan("com.example.orderapi.postgres");

em.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

return em;

}

@Primary

@Bean(name = "oracleTransactionManager")

public PlatformTransactionManager oracleTransactionManager(

@Qualifier("oracleEntityManagerFactory") LocalContainerEntityManagerFactoryBean entityManagerFactory) {

return new JpaTransactionManager(entityManagerFactory.getObject());

}

@Bean(name = "postgresTransactionManager")

public PlatformTransactionManager postgresTransactionManager(

@Qualifier("postgresEntityManagerFactory") LocalContainerEntityManagerFactoryBean entityManagerFactory) {

return new JpaTransactionManager(entityManagerFactory.getObject());

}

}

**3. Application Properties**

Define the data source properties in application.properties:

# Oracle Data Source

spring.datasource.oracle.url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.oracle.username=oracleuser

spring.datasource.oracle.password=oraclepassword

spring.datasource.oracle.driver-class-name=oracle.jdbc.OracleDriver

# PostgreSQL Data Source

spring.datasource.postgres.url=jdbc:postgresql://localhost:5432/postgresdb

spring.datasource.postgres.username=postgresuser

spring.datasource.postgres.password=postgrespassword

spring.datasource.postgres.driver-class-name=org.postgresql.Driver

**4. Entity Classes**

Create entity classes for each database:

**Oracle Entity:**

package com.example.orderapi.oracle;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name = "orders")

public class OracleOrder {

@Id

private Long id;

private String description;

// Getters and Setters

}

**PostgreSQL Entity:**

package com.example.orderapi.postgres;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name = "orders")

public class PostgresOrder {

@Id

private Long id;

private String description;

// Getters and Setters

}

**5. Repositories**

Create repositories for each database:

**Oracle Repository:**

package com.example.orderapi.oracle;

import org.springframework.data.jpa.repository.JpaRepository;

public interface OracleOrderRepository extends JpaRepository<OracleOrder, Long> {

}

**PostgreSQL Repository:**

package com.example.orderapi.postgres;

import org.springframework.data.jpa.repository.JpaRepository;

public interface PostgresOrderRepository extends JpaRepository<PostgresOrder, Long> {

}

**6. Service Layer**

Create a service layer to handle business logic:

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

@Service

public class OrderService {

@Autowired

private OracleOrderRepository oracleOrderRepository;

@Autowired

private PostgresOrderRepository postgresOrderRepository;

@Transactional("oracleTransactionManager")

public OracleOrder saveOrderToOracle(OracleOrder order) {

return oracleOrderRepository.save(order);

}

@Transactional("postgresTransactionManager")

public PostgresOrder saveOrderToPostgres(PostgresOrder order) {

return postgresOrderRepository.save(order);

}

}

**7. Controller**

Create a controller to expose the API endpoints:

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

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/orders")

public class OrderController {

@Autowired

private OrderService orderService;

@PostMapping("/oracle")

public OracleOrder createOrderInOracle(@RequestBody OracleOrder order) {

return orderService.saveOrderToOracle(order);

}

@PostMapping("/postgres")

public PostgresOrder createOrderInPostgres(@RequestBody PostgresOrder order) {

return orderService.saveOrderToPostgres(order);

}

}

This setup will allow your Spring Boot 3.0 application using Spring Framework 6 to connect to both Oracle and PostgreSQL databases and handle orders in both. If you have any specific requirements or need further customization, feel free to ask!