New chat

spring boot data jpa example for 2 data bases connect, give code fro scratch

Here's a **complete example from scratch** of how to connect **Spring Boot** with **two databases (e.g., PostgreSQL and MySQL)** using **Spring Data JPA**, with **separate configuration for each**:

✅ Project Overview

- **Tech stack**: Spring Boot, Spring Data JPA
- **DBs**:
  - `PostgreSQL` as `primary`
  - `MySQL` as `secondary`

✅ Project Structure

```swift
src/main/java/com/example/multidb/
├── config/
│   ├── MySQLConfig.java
│   └── PostgresConfig.java
├── entity/
│   ├── mysql/
│   │   └── MySQLEmployee.java
│   └── postgres/
│       └── PostgresCustomer.java
├── repository/
│   ├── mysql/
│   │   └── MySQLEmployeeRepository.java
│   └── postgres/
│       └── PostgresCustomerRepository.java
└── MultiDbApplication.java
```

✅ Step-by-Step Code

1. `pom.xml`

xml

```xml
<dependencies>
    <!-- Spring Boot Starter Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- PostgreSQL driver -->
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
    </dependency>

    <!-- MySQL driver -->
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
    </dependency>
</dependencies>
```

2. `application.properties`

properties

```properties
# PostgreSQL (Primary)
spring.datasource.postgres.jdbc-url=jdbc:postgresql://localhost:5432/postgresdb
spring.datasource.postgres.username=postgres
spring.datasource.postgres.password=postgres
spring.datasource.postgres.driver-class-name=org.postgresql.Driver

# MySQL (Secondary)
spring.datasource.mysql.jdbc-url=jdbc:mysql://localhost:3306/mysqldb
spring.datasource.mysql.username=root
```

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

# JPA
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

---

3. PostgreSQL Config (`PostgresConfig.java`)

java

```
@Configuration
@EnableTransactionManagement
@EnableJpaRepositories(
        basePackages = "com.example.multidb.repository.postgres",
        entityManagerFactoryRef = "postgresEntityManager",
        transactionManagerRef = "postgresTransactionManager"
)
public class PostgresConfig {

    @Primary
    @Bean
    @ConfigurationProperties(prefix = "spring.datasource.postgres")
    public DataSourceProperties postgresDataSourceProperties() {
        return new DataSourceProperties();
    }

    @Primary
    @Bean
    public DataSource postgresDataSource() {
        return postgresDataSourceProperties().initializeDataSourceBuilder().build();
    }

    @Primary
    @Bean
    public LocalContainerEntityManagerFactoryBean
postgresEntityManager(EntityManagerFactoryBuilder builder) {
        return builder
                .dataSource(postgresDataSource())
                .packages("com.example.multidb.entity.postgres")
                .persistenceUnit("postgres")
```

```
                .build();
    }

    @Primary
    @Bean
    public PlatformTransactionManager postgresTransactionManager(
            @Qualifier("postgresEntityManager") EntityManagerFactory emf) {
        return new JpaTransactionManager(emf);
    }
}
```

---

4. MySQL Config (`MySQLConfig.java`)

java

```
@Configuration
@EnableTransactionManagement
@EnableJpaRepositories(
        basePackages = "com.example.multidb.repository.mysql",
        entityManagerFactoryRef = "mysqlEntityManager",
        transactionManagerRef = "mysqlTransactionManager"
)
public class MySQLConfig {

    @Bean
    @ConfigurationProperties(prefix = "spring.datasource.mysql")
    public DataSourceProperties mysqlDataSourceProperties() {
        return new DataSourceProperties();
    }

    @Bean
    public DataSource mysqlDataSource() {
        return mysqlDataSourceProperties().initializeDataSourceBuilder().build();
    }

    @Bean
    public LocalContainerEntityManagerFactoryBean
mysqlEntityManager(EntityManagerFactoryBuilder builder) {
        return builder
                .dataSource(mysqlDataSource())
                .packages("com.example.multidb.entity.mysql")
```

```
                .persistenceUnit("mysql")
                .build();
    }

    @Bean
    public PlatformTransactionManager mysqlTransactionManager(
            @Qualifier("mysqlEntityManager") EntityManagerFactory emf) {
        return new JpaTransactionManager(emf);
    }
}
```

---

5. PostgreSQL Entity

java

```
package com.example.multidb.entity.postgres;

import jakarta.persistence.*;

@Entity
public class PostgresCustomer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;

    // getters & setters
}
```

---

6. MySQL Entity

java

```
package com.example.multidb.entity.mysql;

import jakarta.persistence.*;

@Entity
```

```
public class MySQLEmployee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String empName;
    private String empDept;

    // getters & setters
}
```

---

7. PostgreSQL Repository

java

```
package com.example.multidb.repository.postgres;

import com.example.multidb.entity.postgres.PostgresCustomer;
import org.springframework.data.jpa.repository.JpaRepository;

public interface PostgresCustomerRepository extends JpaRepository<PostgresCustomer,
Long> {
}
```

---

8. MySQL Repository

java

```
package com.example.multidb.repository.mysql;

import com.example.multidb.entity.mysql.MySQLEmployee;
import org.springframework.data.jpa.repository.JpaRepository;

public interface MySQLEmployeeRepository extends JpaRepository<MySQLEmployee, Long>
{
}
```

---

9. Main Application

java

```java
package com.example.multidb;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MultiDbApplication {
    public static void main(String[] args) {
        SpringApplication.run(MultiDbApplication.class, args);
    }
}
```

✅ Test with Sample Data

You can inject both repositories in a controller or service and test:

java

```java
@RestController
@RequestMapping("/api/test")
public class TestController {

    private final PostgresCustomerRepository customerRepo;
    private final MySQLEmployeeRepository employeeRepo;

    public TestController(PostgresCustomerRepository customerRepo,
MySQLEmployeeRepository employeeRepo) {
        this.customerRepo = customerRepo;
        this.employeeRepo = employeeRepo;
    }

    @GetMapping("/add")
    public String test() {
        customerRepo.save(new PostgresCustomer(null, "John", "john@gmail.com"));
        employeeRepo.save(new MySQLEmployee(null, "Alice", "IT"));
        return "Saved to both DBs!";
    }
}
```

✅ Summary

- Two datasources configured via `@EnableJpaRepositories` with `persistenceUnit`
- Each DB has its own entity, repository, and configuration
- Primary = PostgreSQL, Secondary = MySQL

Let me know if you want:

- ✅ Flyway/Liquibase setup for both DBs
- ✅ Spring Batch copying from one DB to another
- ✅ Scheduling DB sync

Would you like a ZIP or GitHub project for this?