Spring Boot multi-db example

spring boot data jpa example for 3 data bases like postgress, mysql and mongodb, give code from scratch

Here's a complete **Spring Boot** example from scratch that demonstrates how to configure **3 databases**:

- **PostgreSQL**
- **MySQL**
- **MongoDB**

We will use **Spring Data JPA** for PostgreSQL and MySQL, and **Spring Data MongoDB** for MongoDB.

✅ Project Structure

```css
spring-multi-db-example/
├── src/
│   └── main/
│       ├── java/
│       │   └── com/example/multidb/
│       │       ├── config/
│       │       │   ├── MySQLConfig.java
│       │       │   └── PostgresConfig.java
│       │       ├── controller/
│       │       ├── entity/
│       │       │   ├── mysql/
│       │       │   │   └── MySQLUser.java
│       │       │   ├── postgres/
│       │       │   │   └── PostgresUser.java
│       │       │   └── mongo/
│       │       │       └── MongoUser.java
│       │       ├── repository/
│       │       │   ├── mysql/
│       │       │   │   └── MySQLUserRepository.java
│   │   │   ├── postgres/
│   │   │   │   └── PostgresUserRepository.java
│   │   │   └── mongo/
│   │   │       └── MongoUserRepository.java
│   │   └── MultiDbApplication.java
│   └── resources/
│       └── application.yml
├── pom.xml
```

✅ `pom.xml`

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>multi-db-example</artifactId>
  <version>1.0.0</version>
  <dependencies>
    <!-- Spring Boot Starter -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
    </dependency>

    <!-- Spring Data JPA -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- Spring Data MongoDB -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-mongodb</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>

    <!-- Spring Boot Test -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <properties>
    <java.version>17</java.version>
    <spring-boot.version>3.1.0</spring-boot.version>
  </properties>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

✅ `application.yml`

yaml

```yaml
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/postgres_db
    username: postgres_user
    password: postgres_pass
    driver-class-name: org.postgresql.Driver
  jpa:
    database-platform: org.hibernate.dialect.PostgreSQLDialect
    show-sql: true
    hibernate:
      ddl-auto: update

mysql:
  datasource:
    url: jdbc:mysql://localhost:3306/mysql_db
    username: mysql_user
    password: mysql_pass
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    show-sql: true
    hibernate:
      ddl-auto: update
    database-platform: org.hibernate.dialect.MySQLDialect

spring:
  data:
    mongodb:
      uri: mongodb://localhost:27017/mongo_db
```

✅ PostgreSQL Entity + Repository

`entity/postgres/PostgresUser.java`

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

import jakarta.persistence.*;
import lombok.*;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class PostgresUser {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
}
```

`repository/postgres/PostgresUserRepository.java`

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

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

public interface PostgresUserRepository extends JpaRepository<PostgresUser, Long> {}
```

`config/PostgresConfig.java`

```java
package com.example.multidb.config;

import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@Configuration
@EntityScan(basePackages = "com.example.multidb.entity.postgres")
@EnableJpaRepositories(
    basePackages = "com.example.multidb.repository.postgres"
)
public class PostgresConfig {}
```

✅ MySQL Entity + Repository

`entity/mysql/MySQLUser.java`

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

import jakarta.persistence.*;
import lombok.*;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class MySQLUser {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String email;
}
```

`repository/mysql/MySQLUserRepository.java`

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

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

public interface MySQLUserRepository extends JpaRepository<MySQLUser, Long> {}
```

`config/MySQLConfig.java`

```java
 package com.example.multidb.config;

import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

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

> You can enhance MySQLConfig to use a `LocalContainerEntityManagerFactoryBean` and `PlatformTransactionManager` for advanced customization.

✅ MongoDB Document + Repository

`entity/mongo/MongoUser.java`

```java
package com.example.multidb.entity.mongo;

import lombok.*;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document("mongo_users")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class MongoUser {
    @Id
    private String id;
    private String username;
}
```

`repository/mongo/MongoUserRepository.java`

```java
package com.example.multidb.repository.mongo;

import com.example.multidb.entity.mongo.MongoUser;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface MongoUserRepository extends MongoRepository<MongoUser, String> {}
```

> No extra MongoDB config required for default `spring.data.mongodb.uri` setup.

✅ Controller (Test All 3 DBs)

`controller/UserController.java`

```java
package com.example.multidb.controller;

import com.example.multidb.entity.mysql.MySQLUser;
import com.example.multidb.entity.mongo.MongoUser;
import com.example.multidb.entity.postgres.PostgresUser;
import com.example.multidb.repository.mysql.MySQLUserRepository;
import com.example.multidb.repository.mongo.MongoUserRepository;
import com.example.multidb.repository.postgres.PostgresUserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/users")
public class UserController {

    @Autowired private PostgresUserRepository postgresRepo;
    @Autowired private MySQLUserRepository mysqlRepo;
    @Autowired private MongoUserRepository mongoRepo;

    @PostMapping("/postgres")
    public PostgresUser savePostgres(@RequestBody PostgresUser user) {
        return postgresRepo.save(user);
    }

    @PostMapping("/mysql")
    public MySQLUser saveMySQL(@RequestBody MySQLUser user) {
        return mysqlRepo.save(user);
    }

    @PostMapping("/mongo")
    public MongoUser saveMongo(@RequestBody MongoUser user) {
        return mongoRepo.save(user);
    }
}
```

✅ Main Class

`MultiDbApplication.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);
    }
}
```

✅ Run Your Application

Make sure:

- PostgreSQL is running on port `5432`
- MySQL is running on port `3306`
- MongoDB is running on port `27017`

Use Postman or cURL to test:

```bash
curl -X POST http://localhost:8080/users/postgres -H "Content-Type: application/json" -d '{"name": "Postgres User"}'

curl -X POST http://localhost:8080/users/mysql -H "Content-Type: application/json" -d '{"email": "mysql@example.com"}'

curl -X POST http://localhost:8080/users/mongo -H "Content-Type: application/json" -d '{"username": "mongoUser"}'
```

🧠 Notes

- You can use `@Primary` on one `DataSource` bean if you go the manual bean config route.
- MongoDB configuration is automatic with just the URI.
- More complex cases may involve separate `TransactionManager`, `EntityManagerFactory` for JPA configs.

Would you like me to generate this code as a downloadable ZIP or GitHub repo structure for quick use?