Run Scripts

To use SQL scripts defined under resources in a Spring Boot application using `application.properties`, you typically set up a data source and specify the scripts to be executed during application startup. Here's how you can do it:

1. **Place SQL Scripts in Resources**: Ensure your SQL scripts are located in the `src/main/resources` directory of your Spring Boot project.

2. **Configure Data Source**: Define your data source configuration in the `application.properties` file. Below is an example configuration for MySQL:

   ```properties
   spring.datasource.url=jdbc:mysql://localhost:3306/your_database
   spring.datasource.username=your_username
   spring.datasource.password=your_password
   spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
   ```

   Adjust the URL, username, password, and driver class name according to your database setup.

3. **Specify SQL Scripts**: Define the SQL scripts you want to execute during application startup in `application.properties` using the `spring.datasource.schema` or `spring.datasource.data` properties. The `schema` property is typically used for DDL (Data Definition Language) scripts, while the `data` property is used for DML (Data Manipulation Language) scripts.

   For example:

   ```properties
   spring.datasource.schema=classpath:schema.sql
   spring.datasource.data=classpath:data.sql
   ```

   Here, `schema.sql` and `data.sql` are the SQL scripts located in the `src/main/resources` directory.

4. **Run Application**: Run your Spring Boot application. During startup, Spring Boot will automatically execute the specified SQL scripts against your database.

Make sure that your SQL scripts are compatible with your database and that they are idempotent, meaning they can be safely executed multiple times without causing issues. Also, ensure that your application's database initialization mode is set appropriately to prevent unintentional data loss or schema changes during development or production. You can configure this using the `spring.datasource.initialization-mode` property in `application.properties`.

There is another way to configure

```
@SpringBootApplication
@Sql("/data.sql")
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
```

To run SQL scripts using JPA (Java Persistence API) in a Spring Boot application, you typically leverage JPA's `EntityManager` or Spring Data JPA's `JpaRepository`. Here's how you can do it:

1. **Place SQL Scripts in Resources**: Ensure all your SQL scripts are located in the `src/main/resources` directory of your Spring Boot project.

2. **Configure Data Source**: Set up your data source configuration in the `application.properties` file as previously described.

3. **Create a Service or Component to Execute Scripts**: You can create a service or a component responsible for executing SQL scripts during application startup.

4. **Inject EntityManager or JpaRepository**: Depending on your preference and use case, you can either inject `EntityManager` or `JpaRepository` into your service or component.

5. **Execute SQL Scripts Programmatically**: Inside your service or component, programmatically execute the SQL scripts using the injected `EntityManager` or `JpaRepository`.

Here's an example using `EntityManager`:

```java
import javax.persistence.EntityManager;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.stream.Collectors;

@Service
public class DatabaseInitializationService {

    @Autowired
    private EntityManager entityManager;

    @Transactional
    public void executeSqlScript(String scriptPath) {
        try {
            InputStream inputStream =
getClass().getClassLoader().getResourceAsStream(scriptPath);
            if (inputStream != null) {
                String scriptContent = new BufferedReader(new
InputStreamReader(inputStream))
                        .lines().collect(Collectors.joining("\n"));
                entityManager.createNativeQuery(scriptContent).executeUpdate();
            } else {
                throw new IllegalArgumentException("Script not found: " +
scriptPath);
            }
        } catch (Exception ex) {
            // Handle exception
        }
    }
}
```

And then you can call this service from your main application class or any other appropriate place to execute the scripts:

```java
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.beans.factory.annotation.Autowired;

@SpringBootApplication
public class MyApplication implements CommandLineRunner {

    @Autowired
    private DatabaseInitializationService initializationService;
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        initializationService.executeSqlScript("schema.sql");
        initializationService.executeSqlScript("data.sql");
        initializationService.executeSqlScript("additional.sql");
    }
}
```

Ensure that your SQL scripts are compatible with your database and that they are idempotent to avoid unintended consequences. Additionally, handle exceptions appropriately to ensure graceful error handling during script execution.

Sql Statement:

The SQL statement `CREATE USER 'spring'@'localhost' IDENTIFIED BY 'springstudent'` is used to create a new user in a MySQL database with the username 'spring' and the password 'springstudent'. Let's break down its components:

- **CREATE USER**: This part of the statement indicates that you want to create a new user in the MySQL database.

- **'spring'@'localhost'**: This specifies the username and the host from which the user can connect to the MySQL database. In this case, 'spring' is the username, and 'localhost' is the host. This means that the user 'spring' can only connect to the MySQL database from the local machine (the same machine where the MySQL server is running).

- **IDENTIFIED BY 'springstudent'**: This part sets the password for the user 'spring'. The password 'springstudent' is associated with the user 'spring', and it will be required whenever this user attempts to connect to the MySQL database.

Overall, this SQL statement creates a new user named 'spring' with the password 'springstudent' who can only connect to the MySQL database from the local machine. This is often used in scenarios where you need to set up specific users with restricted access rights for security or application purposes.