COURSEWARE

- Professional Skills
- Agile Fundamentals
- Jira
- Git
- Databases Introduction
- Java Beginner
- Maven
- Testing (Foundation)
- Java Intermediate
- HTML
- CSS
- Javascript
- Spring Boot
  - Introduction to Spring Boot
  - Multi-Tier Architecture
  - Beans
  - Bean Scopes
  - Bean Validation
  - Dependency Injection
  - Components
  - Configuration
  - Connecting to a Database
  - Entities
  - Postman
  - Controllers
  - Services
  - Repositories
  - Custom Queries
  - Data Transfer Objects
  - Lombok
  - Custom Exceptions
  - Swagger
  - Profiles
  - Pre-Populating Databases for Testing
  - Unit testing with Mockito

Pre-Populating Databases for Testing

Contents

- Overview
  - data.sql
- Loading specific data
- Tutorial
- Exercises

Overview

When testing our Spring Boot applications it is often useful to "pre-populate" the test database with certain data.
Pre-populating databases can make writing tests quicker and easier because we no longer have to write code in the test to save data to the database - we can just do it directly using SQL.

data.sql

By default, Spring will look for a file named **data.sql** and populate the database by executing the SQL statements inside.

Example **data.sql**:

```
INSERT INTO `person` (`age`, `name`) VALUES (26, 'Jordan Harrison'), (25, 'Nick Johnson'), (25, 'Chris Perrins');
```

The `data.sql` file will typically go in either `src/main/resources` or `src/test/resources`.
Note that when running the application normally only files from the `main` folder will be loaded but when testing the application files will be loaded from *both* locations.
This means that if you had the same `data.sql` in `src/main/resources` and `src/test/resources` then the data would be loaded *twice* during testing!

The same process works with a `schema.sql` file, but this is typically handled by Spring using `spring.jpa.hibernate.ddl-auto`.

Loading specific data

In Spring it is possible to load specific data for a test using the `@Sql` annotation.
Use `scripts` to set which files should be run.

Decide when these scripts are run using `executionPhase` - allows for either running scripts before each test (`BEFORE_TEST_METHOD`) or after each test (`AFTER_TEST_METHOD`).

Tutorial

▶ Example test class
As we can see the *tests* are there, we just need to provide a set of data so the tests will pass.

First, we will create two SQL files:

▶ person-schema.sql
▶ person-data.sql
As these files will be loaded during tests, they should go in `src/test/resources`:

Testing
Selenium
Sonarqube
Advanced Testing (Theory)
Cucumber
MongoDB
Express
NodeJS
React
Express-Testing
Networking
Security
Cloud Fundamentals
AWS Foundations
AWS Intermediate
Linux
DevOps
Jenkins Introduction
Jenkins Pipeline
Markdown
IDE Cheatsheet

```
└──src
    └──test
        ├──java
        │   └──com
        │       └──qa
        │           └──demo
        │               │   PrePopulatingDbsApplicationTests.java
        │               │
        │               └──rest
        │                       PersonControllerIntegrationTest.java
        │
        └──resources
                person-data.sql
                person-schema.sql
```

The first will recreate the `person` table and the second will populate it with our data.
Doing this has the advantage of *guaranteeing* the data present when the tests run, as well as preventing any issues with having auto-generated identifiers mucking up the tests.

Next, we need to tell the test class which files it should load using `@SQL`:

```
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@AutoConfigureMockMvc
@Sql(scripts = { "classpath:person-schema.sql",
        "classpath:person-data.sql" }, executionPhase =
ExecutionPhase.BEFORE_TEST_METHOD)
@ActiveProfiles("test")
public class PersonControllerIntegrationTest {
```

And that's it - now every test in `PersonControllerIntegrationTest` will run against the dataset we've set out.

Note that the SQL files will be read in the order they are listed in the `scripts` array so if you are using a schema file, make sure to put it in *first*.

Link to example repo

Exercises

There are no exercises for this module.