Spring Data: youtubers DB

Ein top-down Ansatz zu Datenbanken und deren Abstraktionslayer in Java mit Spring Boot.

Lernziele

1. JPA und Spring Data Repositories als eine Datenbankabstraktion in Java anwenden
2. Die Begriffe "Datenbank", "JDBC", "JPA", "Connection", "SQL-Driver", "SQL-Dialekt" erklären und darstellen.

Links

1. Spring Data Referenz - https://docs.spring.io/spring-data/data-commons/docs/2.7.2/reference/html/
2. Ein H2 Tutorial: https://attacomsian.com/blog/spring-data-jpa-h2-database
3. H2 SQL Grammar - http://h2database.com/html/grammar.html
4. Top 1000 Youtubers Beispieldatenbank - https://www.kaggle.com/datasets/syedjaferk/top-1000-youtubers-clea–ned

Ablauf

Wie immer ist dies kein step-by-step Tutorial, sondern eine Liste von Hilfestellungen. Ziel ist es, eine Datenbank der Top 1000 Youtubers via Spring Data verfügbar zu machen und auf verschiedene Arten zu nutzen.

Initializr

Wir starten mit einem leeren Spring Boot Projekt.

https://start.spring.io/#!type=gradle-project&language=java&platformVersion=2.7.2&packaging=jar&jvmVersion=17&groupId=ch.bbw.m151&artifactId=youtubers&name=youtubers&description=BBW%20Spring%20Data%20Demo&packageName=ch.bbw.m151.youtubers&dependencies=data-jpa,web,h2

- Gradle Project, Java
- Spring Boot 2.7.2
- ch.bbw.m151:springdata
- Java 17
- Dependencies:
    a. Spring Data JPA - Java-Persistenz mit Hibernate
    b. Spring Web - Standard REST-Services
    c. H2 Database - in-memory Datenbank

IMPORTANT	Öffne das Projekt im (neusten) IntelliJ Ultimate und starte die `YoutubersApplication` ohne Fehler.

Datenbank Verbindung

Als Erstes muss unsere Applikation wissen, wo sie die Datenbank finden kann: "3. Database Configuration"

Unsere H2 DB kann auch via online Konsole erreicht werden. Eine Beispielkonfiguration findet sich in "5. Accessing the H2 Console".

application.properties

```
spring.datasource.url=jdbc:h2:mem:mydb;DB_CLOSE_DELAY=-1
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.h2.console.enabled=true
spring.jpa.show-sql=true
spring.jpa.defer-datasource-initialization=true
spring.jpa.open-in-view=false
```

IMPORTANT Starte die Applikation und verbinde auf die Datenbank via http://localhost:8080/h2-console/ .

Datenbank Verbindung via TCP

Access the Same In-Memory H2 zeigt einen gangbaren Weg die H2 Datenbank auch via TCP zu erreichen. Das hilft uns bei der IntelliJ-Integration.

build.gradle

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'com.h2database:h2' // not `runtimeOnly`
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

YoutubersApplication.java

```
@Bean(initMethod = "start", destroyMethod = "stop")
public Server inMemoryH2DatabaseaServer() throws SQLException {
    // connectable on: jdbc:h2:tcp://localhost:9090/mem:mydb
    return Server.createTcpServer("-tcp", "-tcpAllowOthers", "-tcpPort", "9090");
}
```

IMPORTANT Verbinde auf die H2DB mit der IntelliJ-Database View (View → Tool Windows → Database → +)

Entity erstellen

Eine `@Entity` representiert je eine DB-Tabelle. Anstelle einer `Book`-Entity erstellen wir eine `YoutubersEntity` mit `@Column`s entsprechend dem Youtubers Datensatz (`csv`), ansonsten können wir dem Tutorial "Create an Entity" folgen.

IMPORTANT Starte die Applikation neu und checke, ob die Tabelle existiert wie erwartet.

Initiale Daten Laden

4.1. DataSource Initialization beschreibt einen einfachen Ansatz um initiale Daten beim Programmstart zu laden. Erstelle dazu ein geeignetes File `src/main/resources/data.sql`. Achte auf Datentypen, so eignet sich für `rank` zB `int`.

Konvertiere (Search/Replace/Regex) das csv entsprechend.

Alternativ (advanced) nutze CSVREAD. Optional verwende `enum` Typen wo möglich.

IMPORTANT	Verifiziere mithilfe des IntelliJ Database Viewers, dass alle 1000 Datensätze geladen wurdem.

Queries

Wir wollen nun dieselben Queries auf verschiedene Arten schreiben:

1. Die Anzahl Datensätze in der `Youtubers`-Tabelle (`1000`)
2. Alle Youtuber mit mindestes 100 Millionen Subscribern (drei Rows)
3. Alle Youtuber aus der Kategorie `Sports` nach Username sortiert. (acht Rows, die letzte ist "Red Bull")
4. Alle Youtuber mit einem "a" im Usernamen, sortiert nach Username, zu Pages von jeweils 10 Rows. (total 656 Rows, also 66 Pages)
5. Alle Länder ohne Duplikate und ohne `null`. (eine Liste von 28 Länder)

Native SQL Queries

SQL Kurzrepetition :) .

Eine Beispiellösung als SQL-Query

```
SELECT * FROM YOUTUBERS WHERE SUBSCRIBERS >= 100000000;
```

IMPORTANT	Schreibe und teste native SQL-Queries direkt in der IntelliJ "Query Console".

JPA Repositories

Erstelle ein neues Repository analog zu "Create a Repository". Überfliege auch die offizielle Dokumentation zu "4. Working with Spring Data Repositories".

Repository Deklaration

```
public interface YoutubersRepository extends JpaRepository<YoutubersEntity, Integer>
{

    List<YoutubersEntity> findAllByAudienceCountryIn(List<String> countryFilter);

    @Query("SELECT e FROM youtubers e WHERE e.audienceCountry IN :filter")
    List<YoutubersEntity> findAllByAudienceCountryInNative(@Param("filter") List
<String> countryFilter);
    ...
```

Schreibe JUnit-Tests, um die Methoden zu verifizieren. Ein Start kann so aussehen:

```
@SpringBootTest(properties = {"spring.h2.console.enabled=false"})
class YoutubersApplicationTests implements WithAssertions {

    @Autowired
    YoutubersRepository repository;

    @Test
    void count() {
        assertThat(repository.count()).isEqualTo(1000);
    }
    ...
```

TIP Spring Data supported Paging zB mit https://www.baeldung.com/spring-data-jpa-pagination-sorting

IMPORTANT Schreibe alle Repository Methoden und jeweils ein JUnit-Test.