SAMSUNG SW ACADEMY FOR YOUTH

對外秘

프로젝트 명세서

데이터베이스 형상관리 실습

對外秘

목차

1. 프로젝트 개요

데이터베이스는 대부분의 소프트웨어 시스템에서 핵심적인 역할을 하는 중요한 기술입니다.

이런 데이터 베이스를 사용하여 개발하는 과정에서 여러 명이 하나의 데이터베이스를 동시에 작업할 경우 다양한 문제가 발생할 수 있습니다.

배포 환경을 구분하여, 프로덕션(production), 개발(development), 로컬(local) 등으로 나눈 후 데이터베이스 또한 분리하여 개발하는 것도 도움이 됩니다. 하지만 언젠가는 통합이 필요하며 실제 개발 과정 중 데이터베이스의 스키마나 데이터 등 역시 충돌(conflict) 가능성이 있습니다.

이러한 문제를 해결하기 위해 데이터베이스 형상 관리의 필요성이 대두되었습니다.

Flyway 와 Prisma 는 이러한 데이터베이스 형상관리를 효과적으로 지원하는 도구로서, 각각 Java 와 JavaScript/TypeScript 환경에서 널리 사용됩니다.

- Flyway: SQL 기반의 마이그레이션 스크립트를 관리하여 데이터베이스 변경 사항을 간단하고 신뢰성 있게 적용할 수 있습니다. 개발자들은 버전 관리된 마이그레이션 파일을 통해 스키마 변경을 추적하고 팀 내에서 일관성을 유지할 수 있습니다.

- Prisma: ORM(Object-Relational Mapping)과 마이그레이션 도구를 통합하여 데이터베이스와 애플리케이션 간의 상호 작용을 효율적으로 관리합니다. Prisma 는 스키마 정의, 마이그레이션, 데이터베이스 액세스 계층을 하나의 도구로 제공하여 개발 생산성을 향상시킵니다.

이 외에도 데이터베이스 형상관리를 위한 다양한 솔루션이 존재하며, 각 솔루션마다 특징이 있습니다.

- Alembic: Python SQLAlchemy 를 사용하는 프로젝트를 위한 데이터베이스

마이그레이션 도구로, Python 스크립트를 통해 스키마 변경을 관리합니다.

- EF Migrations: .NET 환경에서 Entity Framework를 사용하는 경우 데이터베이스 마이그레이션을 지원하며, 코드 기반의 마이그레이션을 제공합니다.

- Laravel Migrations: Laravel의 기본 내장 도구로, 데이터베이스 스키마를 버전 관리하고 팀 간에 공유할 수 있게 해줍니다.

각 솔루션은 특정한 환경이나 언어, 프레임워크에 특화되어 있으며, 팀의 요구 사항과 기술 스택에 따라 적합한 도구를 선택하는 것이 중요합니다.과제는 프로젝트에서 Web Storage를 활용하는 이유와 그에 따른 장점을 설명하며, 개발자들에게 해당 기술을 활용하여 프로젝트의 효율성을 향상시킬 수 있는 방법을 안내 합니다.

2. 과제 목표

1) 데이터베이스 형상 관리의 중요성

- 일관성 유지: 여러 개발자가 동시에 작업할 때 데이터베이스 스키마의 일관성을 보장합니다.

- 변경 추적: 누가 언제 어떤 변경을 했는지 추적하여 문제 발생 시 신속하게 대응할 수 있습니다.

- 자동화 배포: 개발, 테스트, 프로덕션 환경에 데이터베이스 변경 사항을 자동으로 배포할 수 있습니다.

- 충돌 방지: 팀원 간의 데이터베이스 스키마 충돌을 방지하고 효율적인 협업을 지원합니다.

- 롤백 지원: 문제 발생 시 이전 버전으로 쉽게 복원할 수 있습니다.

이번 과정에서는 Flyway 와 Prisma 를 활용하여 데이터베이스 형상관리를 실습해보고자 합니다. 이를 통해 데이터베이스 변경 사항을 효율적으로 관리하고 팀 협업을 향상시키는 방법을 배우게 될 것입니다.

2) SQLite

데이터베이스 형상관리를 실습해보려면 데이터베이스가 있어야 합니다. 여기서는 가장 간단한 SQLite 를 사용해서 실습해보겠습니다.

SQLite 는 서버가 필요 없는 경량의 오픈 소스 관계형 데이터베이스 관리 시스템(RDBMS)입니다. 단일 파일로 구성되어 있으며, 애플리케이션과 함께 내장되어 사용할 수 있습니다. 따라서 사실상 데이터베이스를 위한 설치는 별도로 없으며 필요시 클라이언트 설치는 필요할 수 있습니다.

SQLite 는 다음과 같은 특징을 가지고 있습니다.

- 경량성: 설치나 설정이 필요 없으며, 단일 파일로 구성되어 있습니다.
- 서버리스: 별도의 데이터베이스 서버가 필요 없으며, 애플리케이션에서 직접 파일을 읽고 씁니다.
- 크로스 플랫폼: 다양한 운영 체제에서 동일한 방식으로 동작합니다.
- 표준 준수: 대부분의 SQL 표준을 지원합니다.

SQLite 는 임베디드 시스템, 모바일 애플리케이션, 데스크톱 애플리케이션 등에서 널리 사용되며, 학습용이나 프로토타이핑에도 적합합니다.

이 실습에서는 Flyway, Prisma 모두 데이터베이스 설치는 별도로 하지 않으며 결과 확인을 위해서 vscode 의 extension 이나 prisma 자체의 viewer 를 사용하여 결과값을 확인하겠습니다.

3. 학습자료

◆ Flyway

사전 준비

- lyway 설치 (https://flywaydb.org/download) Free Community

- 2 가지 방법 중 택 1₩

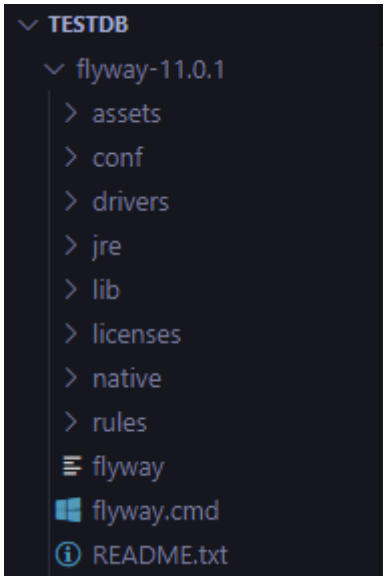
. Installer 사용

. 압축파일 풀기

(https://documentation.red-gate.com/fd/command-line-184127404.html)

1) Flyway 설치

여기서는 압축된 testdb 라는 디렉토리에 Flyway 를 다운로드 받은 후 풀어줍니다.



flyway 의 버전을 확인해봅니다.

```
C:₩Users₩SSAFY₩repo₩testdb₩flyway-11.0.1>flyway --version
Flyway Community Edition 11.0.1 by Redgate

See release notes here: https://rd.gt/416ObMi
Plugin Name     | Version
--------------- | ---------------
DB2 for z/OS    | 10.16.3
OceanBase       | 10.16.0
TiDB            | 10.16.0
YugabyteDB      | 10.16.0
ClickHouse      | 10.16.0
Apache Ignite   | 10.16.0
SQLFluff        | not installed
```

2) Flyway 설정 파일 작성

conf 디렉토리에 flyway.conf 파일을 생성하고 다음과 같이 작성합니다.

```
flyway.url=jdbc:sqlite:sample.db
flyway.driver=org.sqlite.JDBC
flyway.locations=filesystem:migration
```

그런 후 info 명령어로 설정상태를 확인합니다.

```
C:₩Users₩SSAFY₩repo₩testdb₩flyway-11.0.1>flyway info
Flyway Community Edition 11.0.1 by Redgate

See release notes here: https://rd.gt/416ObMi
ERROR: Skipping filesystem location: migration (not found)
Database: jdbc:sqlite:sample.db (SQLite 3.41)
Schema history table "main"."flyway_schema_history" does not exist yet

You are not signed in to Flyway, to sign in please run auth
Schema version: << Empty Schema >>

+----------+---------+-------------+------+--------------+-------+----------+
| Category | Version | Description | Type | Installed On | State | Undoable |
+----------+---------+-------------+------+--------------+-------+----------+
| No migrations found                                                        |
+----------+---------+-------------+------+--------------+-------+----------+
```

이 명령을 하고 나면 testdb 디렉토리에 sample.db 가 생성됩니다. 이것은 sqlite DB 입니다. 만약 다른 관계형 DB 를 사용한다면 생성되지 않습니다.

위에서 에러 부분은 migration 디렉토리가 없어서 발생한 에러입니다.

3) 마이그레이션 파일 생성

마이그레이션 스크립트를 저장할 디렉토리 migration 을 생성합니다.

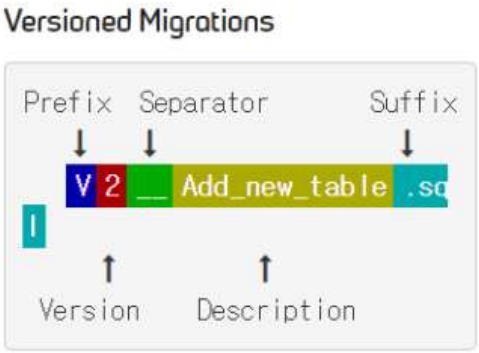
```
C:₩Users₩SSAFY₩repo₩testdb₩flyway-11.0.1>mkdir migration
```

그리고 migration 으로 이동 한 후 아래와 같은 sql 파일을 생성합니다.

migration/V1__create_person_table.sql

```
CREATE TABLE person (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL
);
```

sql 은 id, name 컬럼을 가진 Person 테이블을 생성하는 것입니다. 여기서 추가로 공부해야 할 것은 스크립트 파일명 명명 규칙입니다.



4) Flyway 마이그레이션 실행

이번에는 migrate 명령어를 수행하여 migration 디렉토리에 있는 파일을 실행시킵니다.

```
C:₩Users₩SSAFY₩repo₩testdb₩flyway-11.0.1>flyway migrate
Flyway Community Edition 11.0.1 by Redgate

See release notes here: https://rd.gt/416ObMi

Flyway Pipelines are not active for this project. Learn more here: https://flyway.red-gate.com
Database: jdbc:sqlite:sample.db (SQLite 3.41)
Schema history table "main"."flyway_schema_history" does not exist yet
Successfully validated 1 migration (execution time 00:00.013s)
Creating Schema History table "main"."flyway_schema_history" ...
Current version of schema "main": << Empty Schema >>
Migrating schema "main" to version "1 - create person table"
Successfully applied 1 migration to schema "main", now at version v1 (execution time 00:00.004s)

You are not signed in to Flyway, to sign in please run auth
```

5. 결과 확인

SQLite 데이터베이스가 생성되었는지 확인하고, 테이블이 올바르게 생성되었는지 확인합니다.

sqlite는 뷰어가 많습니다. 각자의 뷰어로 확인해서 V1 파일 내 테이블 생성 sql이 잘 실행되었는지 확인해 봅니다. (아래 화면은 vscode의 SQLite Viewer extension 화면)

◆ Prisma

사전 준비

- Node.js 설치
- npm 또는 yarn 패키지 매니저 사용 가능

1) 프로젝트 생성 및 초기화

```
mkdir prisma-sqlite
cd prisma-sqlite
npm init -y
```

2) Prisma 설치 및 초기화

```
npm install prisma --save-dev
npx prisma init
```

3) 데이터베이스 설정

prisma/schema.prisma 파일을 열어 SQLite 데이터베이스를 설정합니다.

```
datasource db {
  provider = "sqlite"
  url      = "file:./dev.db"
}

generator client {
  provider = "prisma-client-js"
}
```

4) 스키마 모델 정의

prisma/schema.prisma 파일에 Person 모델을 추가합니다.

```
model Person {
  id    Int    @id @default(autoincrement())
  name String
}
```

5) 마이그레이션 생성 및 적용

마이그레이션을 생성하고 데이터베이스에 적용합니다.

```
npx prisma migrate dev --name init
```

이 명령은 다음을 수행합니다.

- 마이그레이션 파일 생성
- 데이터베이스에 스키마 적용
- Prisma Client 업데이트

6) Prisma Client 설치

```
npm install @prisma/client
```

7) 코드 작성 및 실행

index.js 파일을 생성하고 다음 코드를 작성합니다.

```
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

async function main() {
  // 새로운 Person 생성
  const newPerson = await prisma.person.create({
    data: {
      name: '홍길동',
    },
  });
  console.log('Created new person:', newPerson);

  // 모든 Person 조회
  const allPersons = await prisma.person.findMany();
  console.log('All persons:', allPersons);
}

main()
```

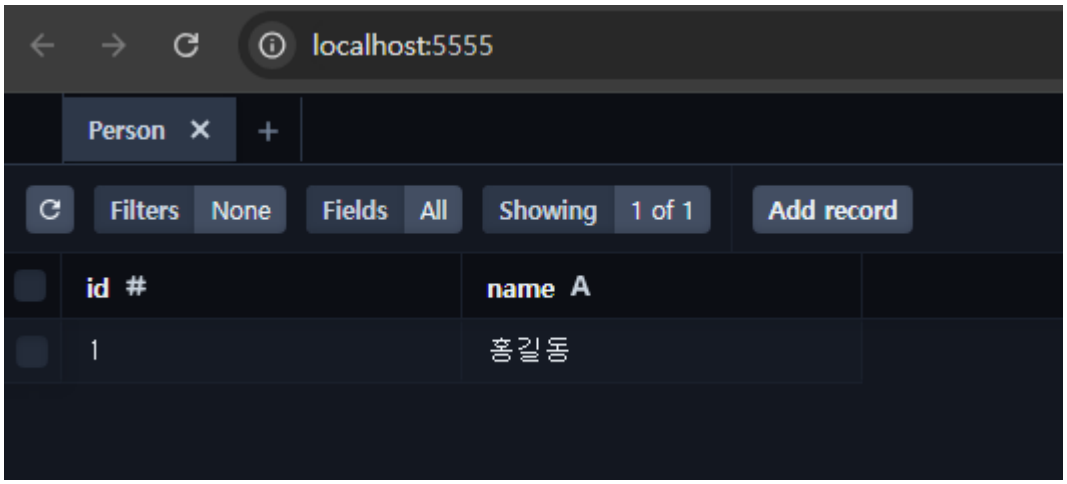
```
  .catch((e) => {
    console.error(e);
  })
  .finally(async () => {
    await prisma.$disconnect();
  });
```

터미널에서 코드를 실행합니다.

```
node index.js
Created new person: { id: 1, name: '홍길동' }
All persons: [ { id: 1, name: '홍길동' } ]
```

출력 결과를 확인하여 데이터베이스 작업이 정상적으로 수행되었는지 확인합니다.

```
npx prisma studio
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma
Prisma Studio is up on http://localhost:555
```

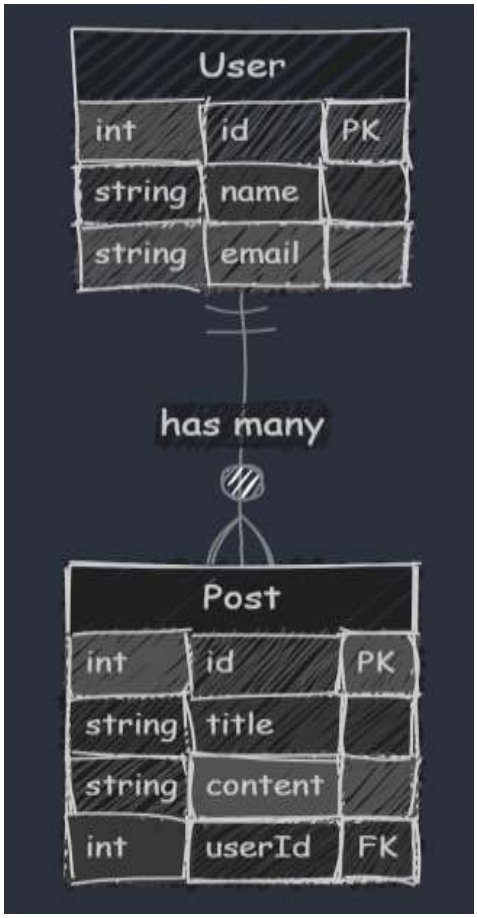


①

4. 과 제

1) 기본 과제

데이터베이스 형상관리 도구(Flyway 또는 Prisma)를 활용하여 간단한 블로그 구현을 위한 User, Post 테이블 생성 및 초기 데이터 삽입을 수행합니다.



- 제시한 다이어그램의 User 와 Post 테이블을 생성하고, User:Post = 1:N 관계를 설정해봅니다.
- 샘플 데이터를 마이그레이션 스크립트를 통해 삽입해 봅니다.

2) 심화 과제

기존 테이블에 추가 요청이 발생하였다고 가정하겠습니다. 새로운 칼럼을 추가하고 마이그레이션을 통해 반영하는 연습을 합니다.

- Post 테이블에 slug 컬럼을 추가하겠습니다.
- slug : TEXT NOT NULL UNIQUE

  slug는 글 제목을 기반으로 한 URL-friendly한 문자열을 말합니다. 보통 블로그 게시글, 제품 페이지 등에서 사용되며, 글 제목이나 콘텐츠 제목에서 자동으로 생성됩니다. 예를 들어, 아래와 같은 URL이 있을 때:

  ○ 일반 URL: https://example.com/posts/123

○ Slug를 사용한 URL: https://example.com/posts/how-to-use-slugs

● 컬럼 추가를 위한 sql 스크립트 제출해 주세요.

5. 산출물 제출

1) https://lab.ssafy.com/s11-study/self-project/ 의 "산출물 제출 가이드" 참조

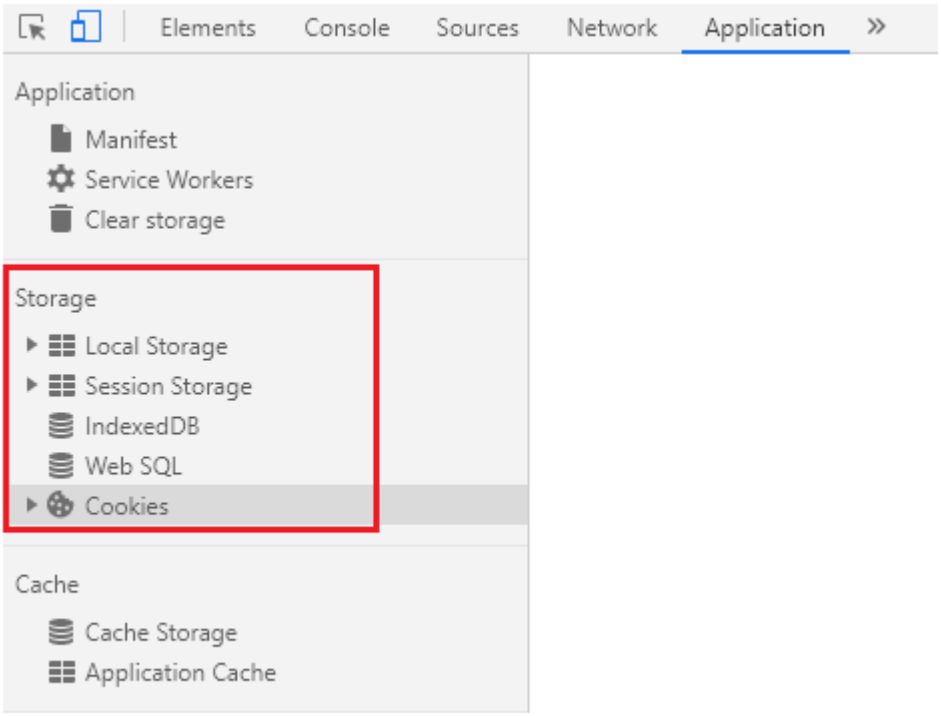
2) 제출물

A. 데이터 분석 정보

- 전체 데이터와 분석한 결과를 포함하는 list
- 본 과제로 인해 이관된 데이터를 위 list에 추가 포함

B. 결과물

- Web storage에 저장된 데이터를 스크린샷으로 포함



개발자 도구의 Application탭에 들어가보자

6. 기대 결과

클라이언트 자원을 활용함으로써, 응답속도를 향상 시키고 서버와의 통신을 최소화하여 네트워크 부하를 감소시킬 수 있습니다. 또한 간편한 사용성으로 코드의 가독성과 유지보수성을 향상시키고, user 의 경험을 향상 시킵니다.

다만, 클라이언트 측에서 관리되는 특성상 민감 정보는 암호화되어야 하며, 보안 취약점을 방어하기 위한 추가 매커니즘이 필요해질 수 있습니다. 그리고 여러 탭이나 창을 열었을 때 데이터의 일관성을 유지하기 위해 데이터 동기화를 고려해야 합니다.

위의 장단점을 명확히 이해하고 프로젝트에 반영한다면, 보다 나은 품질의 프로젝트를 만드는데 큰 도움이 됩니다.