Лабораторна робота 6: Міграції схем за допомогою Prisma ORM

Контекст

У попередніх лабораторних роботах (Лабораторні роботи 1–5) ви розробили та нормалізували схему бази даних PostgreSQL і написали SQL для її створення. У цій заключній лабораторній роботі ви будете **розвивати** цю схему за допомогою інструментів міграції Prisma ORM, імітуючи зміни в реальній базі даних. Prisma дозволяє підключатися до існуючої бази даних, **аналізувати** її схему, а потім застосовувати поступові зміни за допомогою міграцій. Наприклад, запуск команди `npx prisma db pull` зчитає ваші поточні таблиці бази даних у новий файл схеми Prisma, а запуск команди `npx prisma migrate dev` створить та застосує міграцію SQL для оновлення бази даних. Ця практична практика відображає те, як виробничі бази даних безпечно оновлюються з часом.

Цілі

- Використати Prisma ORM для керування схемами та дослідити, як Prisma може аналізувати та змінювати схему вашої бази даних.
- Застосування міграцій, генерування та застосування змін схеми (таблиць, стовпців, зв'язків) за допомогою `prisma migrate .`
- Моделювання за допомогою файлів схеми Prisma. Визначення таблиць та зв'язків у `schema.prisma` та перегляд їхнього відображення в PostgreSQL.
- Виконати базові запити Prisma, вставити та запитати дані за допомогою клієнта Prisma (через *Prisma Studio* або простий скрипт) для перевірки змін.

Інструкції

1. У папці проекту ініціалізуйте нову конфігурацію Prisma. Наприклад:

   ```
   npm init -y
   npm install prisma --save-dev
   npx prisma init --datasource-provider postgresql
   ```

   Це створить папку `prisma/` з файлом `schema.prisma` та файлом `.env`. Оновіть `.env`, щоб `DATABASE_URL` вказував на вашу існуючу базу даних PostgreSQL з лабораторної роботи 5.

   Порада: Наведена вище команда (`npx prisma init --datasource-provider postgresql`) створює файли схеми Prisma та налаштовує джерело даних.

2. Після встановлення URL-адреси бази даних виконайте:

```
npx prisma db pull
```

Ця команда аналізує вашу існуючу схему бази даних та записує її в prisma/schema.prisma. Тепер ви повинні побачити визначення моделей у schema.prisma, що відповідають вашим таблицям з лабораторної роботи 5. Зафіксуйте цю початкову схему (і зміну .env) у Git.

3. Відредагуйте schema.prisma, щоб внести наступні зміни.

4. Додайте нову таблицю: Наприклад, якщо ваша схема має модель Продукт , ви можете додати нову модель Огляд:

```
model Review {
  id        Int     @id @default(autoincrement())
  rating    Int
  comment   String?
  product   Product @relation(fields: [productId], references: [id])
  productId Int
}
```

5. Зміна існуючої таблиці: додавання або перейменування поля в моделі. Наприклад, щоб додати логічний прапорець:

```
model Product {
  id       Int     @id @default(autoincrement())
  name     String
  price    Float
  inStock  Boolean @default(true)
}
```

Або щоб перейменувати inStock на isAvailable, потрібно видалити inStock та додати isAvailable (Prisma трактуватиме це як drop+add).

6. **Видалення стовпця або зв'язку:** Видалення поля або зв'язку. Наприклад, видалення поля опису :

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

Переконайтеся, що кожна зміна ізольована, щоб ви могли застосовувати міграції одну за одною. Збережіть свої зміни.

7. Для кожної зміни запускайте нову міграцію. Наприклад:

```
npx prisma migrate dev --name add-review-table
npx prisma migrate dev --name add-product-field
npx prisma migrate dev --name drop-old-column
```

Кожна команда генерує нову міграцію SQL у `prisma/migrations/` та застосовує її до бази даних. Prisma виводитиме повідомлення типу *«Створення міграції… Застосування міграції…»* . Ці файли відстежують історію вашої схеми. У файлі markdown (`migration-notes.md`) задокументуйте кожну міграцію: запишіть, що ви змінили (наприклад, «Додано модель огляду»), та додайте короткі знімки моделі Prisma або SQL до/після змін. Зафіксуйте папку `prisma/` (включно з усіма міграціями) у Git.

8. Використайте клієнт Prisma, щоб перевірити, чи працює ваша схема. Ви можете або запустити `npx prisma studio` , щоб відкрити веб-інтерфейс та вставити/запитати дані, або написати короткий скрипт Node.js, використовуючи `@prisma/client`. Наприклад:

```
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();
async function main() {
  await prisma.review.create({ data: { rating: 5, comment: "Great!",
productId: 1 } });

  const products = await prisma.product.findMany({ include: { reviews:
true } });
  console.log(products);
}
main();
```

Тут показано, як вставити рядок та отримати пов'язані записи за допомогою клієнта Prisma. Переконайтеся, що ваші запити працюють без помилок (дані відображаються належним чином). Ви можете зробити скріншот Prisma Studio, на якому відображаються рядки таблиці, або показати вивід терміналу у `migration-notes.md`.

9. Додатково – Повторний аналіз: Щоб перевірити остаточну схему, ви можете ще раз запустити `npx prisma db pull`. Це оновить `schema.prisma` відповідно до бази даних. (У реальному робочому процесі ви б переконалися, що схема Prisma відповідає вашим міграціям.)

Поради

- Синтаксис схеми Prisma: у `schema.prisma` використовуйте ключове слово `model` для визначення таблиць. Наприклад:

```
model Category {
```

```
  id    Int    @id @default(autoincrement())
  name  String @unique
}
```

У зв'язках використовується `@relation`. За потреби ознайомтеся з документацією Prisma щодо синтаксису схеми.

- *Завжди* комітьте файли Prisma. Папка `prisma/schema.prisma` та вся папка `prisma/migrations/` (включно з `migration_lock.toml`) повинні бути в Git. Ці файли міграції є частиною історії вашого проєкту.
- Зосередьтеся на схемі та запитах до даних. Ця лабораторна робота не стосується створення застосунку чи API. Вам не потрібно писати маршрути Express чи код фронтенду. Зосередьтеся на змінах схеми та використанні Prisma Client (або Studio) для перевірки даних.
- Під час запуску `prisma migrate dev --name ...` використовуйте чіткі назви (наприклад, `add-email-to-user`), щоб задокументувати, що робить кожна міграція.

Результати

- **Файл схеми Prisma:** оновлений `prisma/schema.prisma`, що відображає всі зміни.
- **Папки міграції:** вміст `prisma/migrations/` (одна підпапка на міграцію, з файлами SQL).
- **Звіт Markdown** migration-notes.md у вашому репозиторії. Для кожної міграції поясніть, що змінилося, та покажіть фрагменти коду «до»/«після». Додайте будь-які запити клієнта Prisma або знімки екрана, що підтверджують результат. Дозволяється звіт у форматах pdf/docx.
- Надайте або невеликий скрипт (як наведений вище), або знімок екрана з Prisma Studio, який демонструє успішне вставлення/запит даних у нових/змінених таблицях.

Подання

Збережіть усю папку `prisma/` (з `schema.prisma` та підпапками `migrations/`), ваш `migration-notes.md` та будь-які тестові скрипти/скріншоти до вашого групового репозиторію Git. Переконайтеся, що все надіслано, щоб викладачі могли переглянути вашу схему, міграції та докази коректних запитів.

Prisma ORM Guide

https://www.prisma.io/docs/orm

NESTJS BACKEND PROJECT STAGE 1. Step 1: Installing Node.js and NestJS... | by Cansu Sancar | Apr, 2025 | Medium

https://medium.com/@cansusancar01/nestjs-backend-project-stage-1-b73461f0f05e

Elevate Your APIs: Node, Prisma ORM & PostgreSQL

https://www.mindbowser.com/node-prisma-postgres-rest-api-guide/