Pemrograman Berbasis Framework

Pertemuan 12-13: **CRUD & Object Relational Mapping (Prisma JS)**

Hendra Permana, M.T
hendrapermana.m@gmail.com
https://hyn.gg

12. CRUD & Object Relational Mapping (Prisma JS)

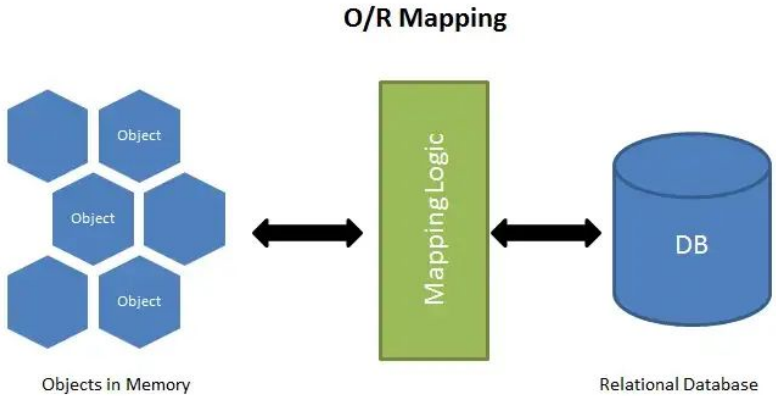
- ORM
- CRUD di Prisma
- Aplikasi sederhana dengan NextJS + Prisma + SQLite

Video Course

- Source code yang digunakan pada video course dapat diakses di: https://codesandbox.io/p/github/hynra/PF-12

ORM

- **ORM** (Object Relational Mapping) merupakan teknik yang merubah suatu **table** menjadi sebuah **object** yang nantinya mudah untuk digunakan
- **Object** yang dibuat memiliki **property** yang sama dengan **field** — field yang ada pada **table** tersebut.
- ORM memungkinkan kita melakukan **query** dan **memanipulasi** data di database menggunakan **object** oriented.



Kenapa ORM?

- Biasanya developer melakukan kesalahan ketika menuliskan query database
- Memerlukan banyak waktu untuk hanya melakukan query.
- ORM mempermudah mengakses database tanpa melakukan query sama sekali.
- ORM Bersifat optional, digunakan tergantung kebutuhan
- Memisahkan kode SQL dari logika aplikasi
- Menghindari ketergantungan aplikasi terhadap vendor database

Cara kerja ORM

- Mendefinisikan suatu object, lalu kita buat field — field pada object sesuai dengan field — field pada table di database
- Misalkan kita akan membuat object bernama makanan. Field yang dimiliki adalah id, nama, harga.
- Selanjutnya object tersebut dapat kita gunakan untuk melakukan CRUD tanpa menggunakan query.

Kelebihan ORM

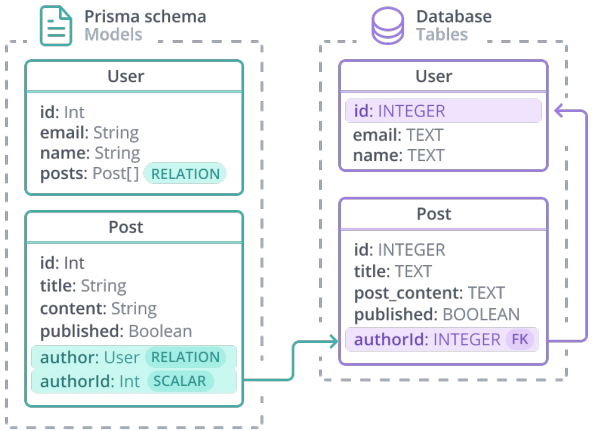
- Terdapat banyak fitur seperti transactions, connection pooling, migrations, seeds, streams, dan lain sebagainya.
- Perintah query memiliki kinerja yang lebih baik, daripada kita menulisnya secara manual.
- Kita menulis model data hanya di satu tempat, sehingga lebih mudah untuk update, maintain, dan reuse the code.
- Membuat akses data menjadi lebih abstrak(kita dapat mengubahnya kapanpun) dan portable.
- Memungkinkan kita memanfaatkan OOP (object oriented programming) dengan baik
- Mempercepat pengembangan program. Contohnya, mengurangi perulangan kode query, memudahkan pemakaian karena tabel-tabel ter-representasikan dalam bentuk objek
- Mengenerate boilerplate code (unit kode yang reusable) untuk fungsi dasar CRUD (Create, Read, Update, Delete).

Kekurangan ORM

- Untuk beberapa query yang sangat kompleks, mungkin ORM tidak dapat meng-handlenya
- konfigurasi awal ORM dapat menjadi sulit. Maka diperlukan untuk mempelajarinya terlebih dahulu sebelum menggunakannya.

Prisma JS

- Prisma JS adalah sebuah perpustakaan JavaScript yang menyediakan antarmuka yang mudah digunakan untuk bekerja dengan database dalam aplikasi JavaScript.
- Ini berfungsi sebagai ORM (Object-Relational Mapper), yang dapat berinteraksi dengan database menggunakan objek dalam kode, bukan menulis query SQL mentah.



Mengapa menggunakan Prisma?

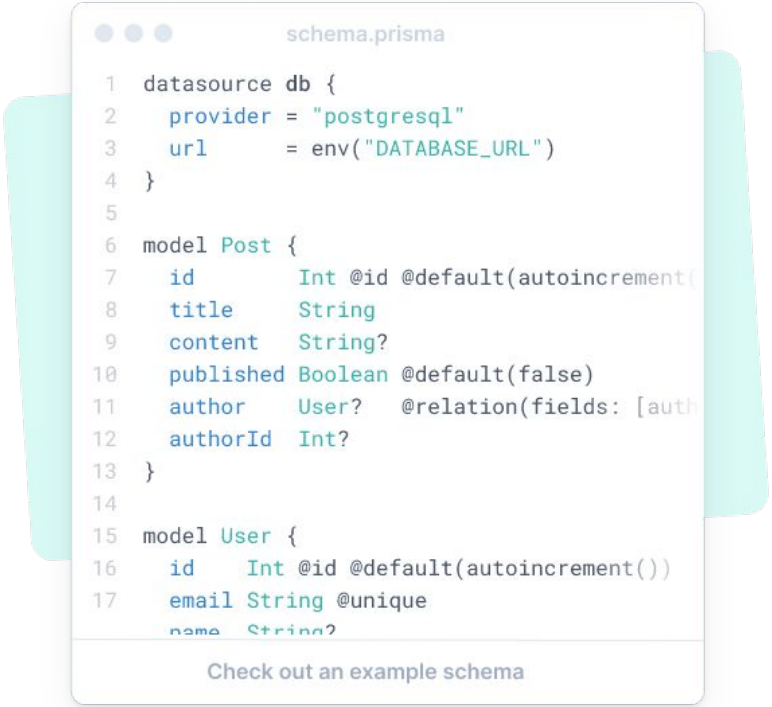
- **Kemudahan**: Prisma menyediakan antarmuka yang sederhana dan elegan untuk bekerja dengan database, sehingga mudah untuk memulai dan cepat memahami cara kerjanya.
- **TypeScript**: Prisma dibangun di atas TypeScript, dapat menggunakan tipe yang kuat dan fitur lanjutan lainnya dalam query database.
- **Model data otomatis**: Prisma menghasilkan model data berdasarkan skema database, tidak perlu menulis model data sendiri. Ini dapat menghemat banyak waktu dan usaha, terutama untuk struktur data yang kompleks.

Fitur Prisma

- **Query dan mutasi generation**: Prisma menghasilkan query dan mutasi berdasarkan model data, sehingga mudah untuk memulai dalam membuat sebuah fitur.
- **Real-time update**: Prisma mendukung pembaruan real-time menggunakan GraphQL subscription, sehingga mudah untuk membangun aplikasi real-time.
- **Database agnostik**: Prisma mendukung beberapa type database, termasuk MySQL, PostgreSQL, dan SQLite, sehingga mudah berganti antar sistem database sesuai kebutuhan.
- **Resolver kustom**: Prisma memungkinkan menulis resolver kustom untuk query dan mutasi GraphQL.

Scheme first

- Prisma memungkinkan kita membuat skema berdasarkan tabel yang ada di dalam Database, termasuk relasi antar tabel dan juga tipe datanya
- Skema bersifat single truth of source, jadi segala bentuk operasi yang dilakukan didasarkan pada skema yang dibuat
- Referensi: https://www.prisma.io/docs/concepts/components/prisma-schema



CRUD

```
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

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

model Post {
  id        Int     @id @default(autoincrement())
  title     String
  content   String?
  published Boolean @default(false)
  author    User?   @relation(fields: [authorId], references: [id])
  authorId  Int?
}

model User {
  id    Int     @id @default(autoincrement())
  email String  @unique
  name  String?
  posts Post[]
}
```

Menggunakan Prisma

```
$ prisma generate
```

```
import { PrismaClient } from '@prisma/client'

const prisma = new PrismaClient()
```

Read operation

Retrieve all `User` records from the database

```
// Run inside `async` function
const allUsers = await prisma.user.findMany()
```

Include the `posts` relation on each returned `User` object

```
// Run inside `async` function
const allUsers = await prisma.user.findMany({
  include: { posts: true },
})
```

Filter all `Post` records that contain `"prisma"`

```
// Run inside `async` function
const filteredPosts = await prisma.post.findMany({
  where: {
    OR: [
      { title: { contains: 'prisma' } },
      { content: { contains: 'prisma' } },
    ],
  },
})
```

Create Operation

Create a new `User` and a new `Post` record in the same query

```
// Run inside `async` function
const user = await prisma.user.create({
  data: {
    name: 'Alice',
    email: 'alice@prisma.io',
    posts: {
      create: { title: 'Join us for Prisma Day 2020' },
    },
  },
})
```

Update operation

Update an existing `Post` record

```
// Run inside `async` function
const post = await prisma.post.update({
  where: { id: 42 },
  data: { published: true },
})
```

Delete Operation

Delete a single record

The following query uses `delete` API to delete a single `User` record:

```
const deleteUser = await prisma.user.delete({
  where: {
    email: 'bert@prisma.io',
  },
})
```

Crud Reference

- https://www.prisma.io/docs/concepts/components/prisma-client/crud

Penutup

- Join Group Telegram mata kuliah PBF sebagai sarana diskusi, pertanyaan seputar mata kuliah PBF: https://t.me/+VYI_dHyFfkdjZmZl

Tugas

- Buat akun di codesandbox.io
- Praktekan dan pahami kembali materi praktikum di pertemuan ke-12 & ke-13 (https://codesandbox.io/p/github/hynra/PF-12), beri komentar setiap baris semampu pemahaman Anda. Submit tugas dalam bentuk link ke project codesanbox di akun Codesandbox Anda