前端全栈必须会的Prisma！

什么是Prisma?

在这个快速迭代的数字时代，**开发者们始终在寻找能够提高数据库操作效率和准确性的工具**。

Prisma，应运而生。Prisma最初是作为一款图形QL数据库工具诞生的，随着时间的推移，它演变成了一个强大的数据库工具集，旨在解决传统ORM工具中存在的性能瓶颈和复杂性问题。

与传统的ORM工具相比，Prisma在易用性、性能和类型安全上提供了显著的优势。它不仅简化了数据库操作，而且提高了开发效率和应用性能。

Prisma提供了一系列令人印象深刻的特性，包括但不限于：

- **自动生成的数据库访问客户端：** 这让开发者可以避免编写重复的数据库CRUD操作代码。
- **直观的数据模型设计：** 使用Prisma Schema Language，开发者可以轻松定义应用数据模型。
- **智能的类型安全：** 它利用强大的类型系统，确保代码的稳定性和减少运行时错误。
- **迁移系统：** Prisma Migrate允许开发者安全地变更数据库结构。

上手

接下来我以用的最多的Mysql数据库为例开始演示，如何在项目里使用Prisma，请先确保安装好了nodejs环境。

依赖

进入代码库，安装prisma依赖，并生成prisma文件夹和.env文件：

```
npm install prisma
npx prisma init
```

连接数据库

打开.env设置对应数据库链接和用户名密码等信息：

```
DATABASE_URL="mysql://johndoe:randompassword@localhost:5432/mydb?schema=public"
```

在**prisma/schema.prisma中修改：**

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

模型映射

假设我们正在开发一款博客应用，有以下结构体：

```
model Post {
  id        Int      @id @default(autoincrement())
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  title     String   @db.VarChar(255)
  content   String?
  published Boolean  @default(false)
  author    User     @relation(fields: [authorId], references: [id])
  authorId  Int
}

model Profile {
  id     Int     @id @default(autoincrement())
  bio    String?
  user   User    @relation(fields: [userId], references: [id])
  userId Int     @unique
}

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

我们可以直接将上述模型定义，转换为数据库指令，进行数据库表的修改：

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

等同于如下SQL表：

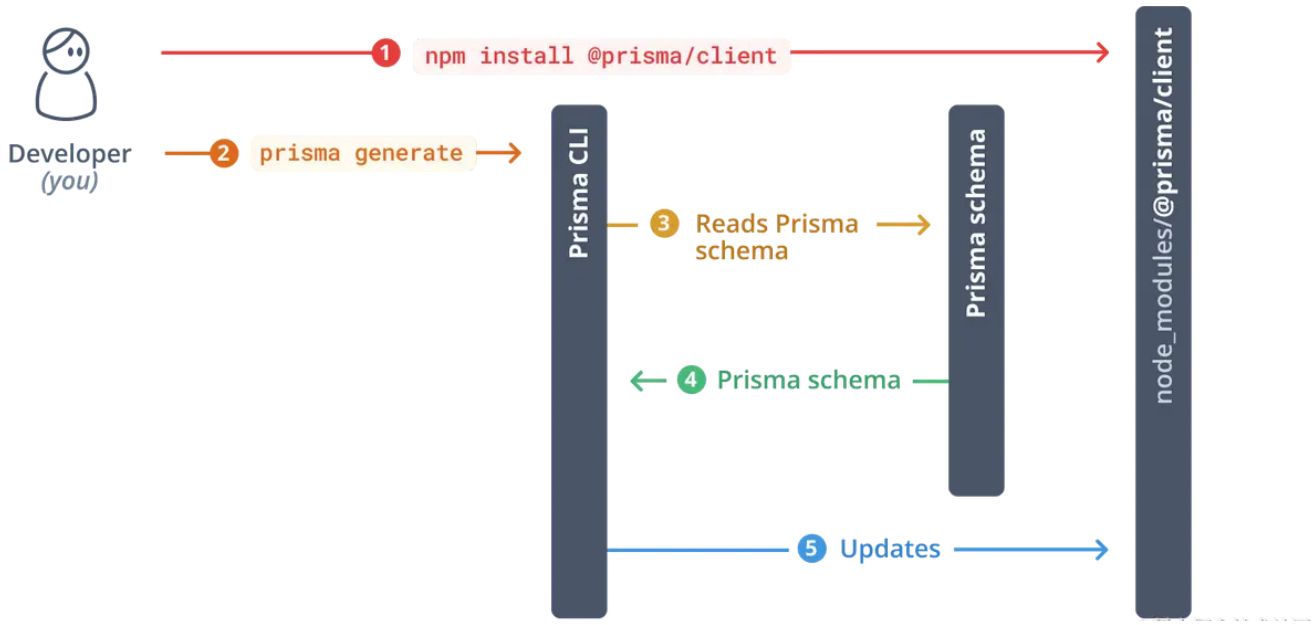
```sql
CREATE TABLE "Post" (
  "id" SERIAL,
  "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  "updatedAt" TIMESTAMP(3) NOT NULL,
  "title" VARCHAR(255) NOT NULL,
  "content" TEXT,
  "published" BOOLEAN NOT NULL DEFAULT false,
  "authorId" INTEGER NOT NULL,
  PRIMARY KEY ("id")
);

CREATE TABLE "Profile" (
  "id" SERIAL,
  "bio" TEXT,
  "userId" INTEGER NOT NULL,
  PRIMARY KEY ("id")
);

CREATE TABLE "User" (
  "id" SERIAL,
  "email" TEXT NOT NULL,
  "name" TEXT,
  PRIMARY KEY ("id")
);

CREATE UNIQUE INDEX "Profile.userId_unique" ON "Profile"("userId");
CREATE UNIQUE INDEX "User.email_unique" ON "User"("email");
ALTER TABLE "Post" ADD FOREIGN KEY("authorId")REFERENCES "User"("id") ON DELETE
 CASCADE ON UPDATE CASCADE;
ALTER TABLE "Profile" ADD FOREIGN KEY("userId")REFERENCES "User"("id") ON
DELETE CASCADE ON UPDATE CASCADE;
```

- 生成客户端

现在我们要做的是，生成prisma客户端，你可以理解为官方提供的调用实例。

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

安装命令会调用 **prisma generate** ，它会读取 Prisma schema并生成适合scheme的 Prisma 客户端版本。



注意，每次修改prisma schema的时候，都需要执行**prisma migrate dev 或 prisma db push。来保证云端数据库和代码里schema结构的同步。**

CRUD

直接开始演示怎么进行增删改查吧!

新建一个index.ts文件：

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

const prisma = new PrismaClient()

async function main() {
  // ... you will write your Prisma Client queries here
    const allUsers = await prisma.user.findMany()
  console.log(allUsers)
}

main()
  .then(async () => {
    await prisma.$disconnect()
  })
  .catch(async (e) => {
    console.error(e)
    await prisma.$disconnect()
    process.exit(1)
  })
```

可以执行这个文件试试：

```
npx ts-node index.ts
```

返回的是一个空数组，因为数据库里什么都没有····

我们来尝试写入数据。

```
async function main() {
  await prisma.user.create({
    data: {
      name: 'Alice',
      email: 'alice@prisma.io',
      posts: {
        create: { title: 'Hello World' },
      },
      profile: {
        create: { bio: 'I like turtles' },
      },
    },
  })

  const allUsers = await prisma.user.findMany({
    include: {
      posts: true,
      profile: true,
    },
  })
  console.dir(allUsers, { depth: null })
}
```

还可以尝试更新：

```
async function main() {
  const post = await prisma.post.update({
    where: { id: 1 },
    data: { published: true },
  })
  console.log(post)
}
```

基础的CRUD就完成啦！是不是很简单？

感受

在我个人使用Prisma的过程中，我深切体会到了它的便利性和高效性。

当然，任何技术都不是完美无缺的，我也遇到了一些挑战，但通过社区的帮助和官方文档，我总能找到解决方案。Prisma是面向未来的数据库工具集，它通过简化数据库操作，提高开发效率，并且不断创新，已经成为了许多现代应用开发的首选工具。

无论你是一个新手还是经验丰富的开发者，Prisma都值得一试。