Hướng dẫn sử dụng Prisma

Mục lục

- Giới thiệu
- Cấu trúc Project
- Kết nối Database
- Tạo Models
- Migrations
- CRUD Operations
- Quan hệ giữa các Models
- Advanced Queries

Giới thiệu

Prisma là ORM (Object-Relational Mapping) hiện đại cho Node.js và TypeScript. Project này sử dụng:

- **Prisma Client**: Để query database
- **Prisma Migrate**: Để quản lý database schema
- **PostgreSQL**: Database (Supabase)

Cấu trúc Project

```
backend/
├── prisma/
│   ├── schema.prisma          # Schema definition
│   └── migrations/            # Migration history
├── src/
│   └── database/
│       └── prisma.js          # Prisma Client instance
└── .env                       # Environment variables
```

Kết nối Database

File `.env`

```
# Connection pooling (dùng cho queries)
DATABASE_URL="postgresql://user:password@host:6543/database?
pgbouncer=true"

# Direct connection (dùng cho migrations)
DIRECT_URL="postgresql://user:password@host:5432/database"
```

File `src/database/prisma.js`

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

let prisma = new PrismaClient()

export default prisma
```

Sử dụng trong code

```
import prisma from './src/database/prisma.js'

// Bây giờ có thể dùng prisma để query
```

Tạo Models

Ví dụ: Model User

Mở file `prisma/schema.prisma` và thêm:

```
model User {
  id        Int      @id @default(autoincrement())
  email     String   @unique
  name      String?
  password  String
  role      String   @default("user")
  createdAt DateTime @default(now())
```

```
  updatedAt DateTime @updatedAt
}
```

Các field types phổ biến:

- `String` - Text
- `Int` - Integer
- `Float` - Số thập phân
- `Boolean` - true/false
- `DateTime` - Ngày giờ
- `Json` - JSON data

Attributes:

- `@id` - Primary key
- `@unique` - Unique constraint
- `@default(value)` - Giá trị mặc định
- `@updatedAt` - Tự động update khi record thay đổi
- `?` - Optional field (có thể null)

Migrations

1. Tạo migration mới

```
npx prisma migrate dev --name ten_migration
```

Ví dụ:

```
npx prisma migrate dev --name add_user_model
npx prisma migrate dev --name add_product_table
```

2. Apply migrations (Production)

```
npx prisma migrate deploy
```

3. Reset database (XÓA TẤT CẢ DỮ LIỆU)

```
npx prisma migrate reset
```

4. Xem trạng thái migrations

```
npx prisma migrate status
```

5. Generate Prisma Client (sau khi thay đổi schema)

```
npx prisma generate
```

CRUD Operations

Import Prisma Client

```
import prisma from './src/database/prisma.js'
```

CREATE - Tạo mới

Tạo 1 record

```
const user = await prisma.user.create({
  data: {
    email: 'test@example.com',
    name: 'John Doe',
    password: 'hashed_password'
  }
})
```

Tạo nhiều records

```
const users = await prisma.user.createMany({
  data: [
    { email: 'user1@example.com', name: 'User 1', password:
'pass1' },
    { email: 'user2@example.com', name: 'User 2', password:
'pass2' }
  ]
})
```

READ - Đọc dữ liệu

Lấy tất cả

```
const users = await prisma.user.findMany()
```

Lấy theo điều kiện

```
const users = await prisma.user.findMany({
  where: {
    role: 'admin'
  }
})
```

Lấy 1 record theo ID

```
const user = await prisma.user.findUnique({
  where: { id: 1 }
})
```

Lấy record đầu tiên

```
const user = await prisma.user.findFirst({
  where: { email: 'test@example.com' }
})
```

Select specific fields

```
const users = await prisma.user.findMany({
  select: {
    id: true,
    email: true,
    name: true
    // không lấy password
  }
})
```

Pagination

```
const users = await prisma.user.findMany({
  skip: 0,    // Bỏ qua bao nhiêu records
  take: 10    // Lấy bao nhiêu records
})
```

Sorting

```
const users = await prisma.user.findMany({
  orderBy: {
    createdAt: 'desc'  // 'asc' hoặc 'desc'
  }
})
```

UPDATE - Cập nhật

Update 1 record

```
const user = await prisma.user.update({
  where: { id: 1 },
  data: {
    name: 'New Name'
  }
})
```

Update nhiều records

```
const result = await prisma.user.updateMany({
  where: { role: 'user' },
  data: {
    role: 'member'
  }
})
```

Upsert (Update nếu có, Create nếu không)

```
const user = await prisma.user.upsert({
  where: { email: 'test@example.com' },
  update: {
    name: 'Updated Name'
  },
  create: {
    email: 'test@example.com',
    name: 'New User',
    password: 'password'
  }
})
```

DELETE - Xóa

Xóa 1 record

```
const user = await prisma.user.delete({
  where: { id: 1 }
})
```

Xóa nhiều records

```
const result = await prisma.user.deleteMany({
  where: {
    createdAt: {
      lt: new Date('2024-01-01')  // Xóa user cũ hơn 2024
    }
```

```
  }
})
```

COUNT - Đếm

```
const count = await prisma.user.count()

// Đếm với điều kiện
const adminCount = await prisma.user.count({
  where: { role: 'admin' }
})
```

Quan hệ giữa các Models

One-to-Many (1-n)

Ví dụ: 1 User có nhiều Posts

```
model User {
  id    Int    @id @default(autoincrement())
  email String @unique
  name  String
  posts Post[] // Quan hệ one-to-many
}

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

Query với quan hệ

```
// Lấy user với tất cả posts
const user = await prisma.user.findUnique({
  where: { id: 1 },
```

```
  include: {
    posts: true
  }
})

// Tạo user và post cùng lúc
const user = await prisma.user.create({
  data: {
    email: 'user@example.com',
    name: 'John',
    posts: {
      create: [
        { title: 'First Post', content: 'Content 1' },
        { title: 'Second Post', content: 'Content 2' }
      ]
    }
  }
})
```

Many-to-Many (n-n)

Ví dụ: Users và Categories (nhiều user có thể thuộc nhiều category)

```
model User {
  id         Int        @id @default(autoincrement())
  email      String     @unique
  categories Category[]
}

model Category {
  id    Int    @id @default(autoincrement())
  name  String
  users User[]
}
```

Query

```
const user = await prisma.user.findUnique({
  where: { id: 1 },
  include: {
    categories: true
  }
})
```

Advanced Queries

Filtering

So sánh

```
const users = await prisma.user.findMany({
  where: {
    age: { gt: 18 }        // greater than
    // lt: less than
    // gte: greater than or equal
    // lte: less than or equal
    // not: not equal
  }
})
```

String operations

```
const users = await prisma.user.findMany({
  where: {
    email: {
      contains: '@gmail.com',   // Chứa
      startsWith: 'admin',       // Bắt đầu với
      endsWith: '.com'           // Kết thúc với
    }
  }
})
```

Logical operators

```
// AND
const users = await prisma.user.findMany({
  where: {
    AND: [
      { role: 'admin' },
      { active: true }
    ]
```

```
  }
})

// OR
const users = await prisma.user.findMany({
  where: {
    OR: [
      { role: 'admin' },
      { role: 'moderator' }
    ]
  }
})

// NOT
const users = await prisma.user.findMany({
  where: {
    NOT: {
      role: 'banned'
    }
  }
})
```

IN / NOT IN

```
const users = await prisma.user.findMany({
  where: {
    role: {
      in: ['admin', 'moderator', 'user']
    }
  }
})
```

Aggregation

```
// Tổng
const result = await prisma.user.aggregate({
  _sum: { age: true },
  _avg: { age: true },
  _min: { age: true },
  _max: { age: true },
  _count: true
})
```

```
// Group by
const result = await prisma.user.groupBy({
  by: ['role'],
  _count: true,
  having: {
    role: {
      _count: {
        gt: 10
      }
    }
  }
})
```

Transactions

```
// Sử dụng khi cần đảm bảo nhiều operations thành công cùng lúc
const result = await prisma.$transaction([
  prisma.user.create({ data: { email: 'user1@example.com', name:
'User 1' } }),
  prisma.user.create({ data: { email: 'user2@example.com', name:
'User 2' } }),
])

// Interactive transaction
const result = await prisma.$transaction(async (tx) => {
  const user = await tx.user.create({
    data: { email: 'test@example.com', name: 'Test' }
  })

  const post = await tx.post.create({
    data: { title: 'Post', content: 'Content', authorId: user.id }
  })

  return { user, post }
})
```

Raw SQL

```
// Nếu cần query phức tạp
const users = await prisma.$queryRaw`
  SELECT * FROM "User" WHERE email LIKE ${'%@gmail.com'}
`
```

```
// Execute SQL
await prisma.$executeRaw`
  UPDATE "User" SET role = 'user' WHERE role IS NULL
`
```

Best Practices

1. Sử dụng 1 Prisma Client instance duy nhất

```
// ☑ ĐÚNG - src/database/prisma.js
import { PrismaClient } from '@prisma/client'
let prisma = new PrismaClient()
export default prisma

// Import và dùng ở mọi nơi
import prisma from './src/database/prisma.js'
```

2. Error Handling

```
try {
  const user = await prisma.user.findUnique({
    where: { id: 1 }
  })

  if (!user) {
    throw new Error('User not found')
  }
} catch (error) {
  console.error('Database error:', error)
  throw error
}
```

3. Đóng connection khi shutdown

```
process.on('beforeExit', async () => {
  await prisma.$disconnect()
})
```

4. Không commit file `.env` lên Git

Thêm vào `.gitignore`:

```
.env
.env.local
```

Debugging

Enable query logging

```
const prisma = new PrismaClient({
  log: ['query', 'info', 'warn', 'error'],
})
```

Prisma Studio (GUI Tool)

```
npx prisma studio
```

Mở browser tại `http://localhost:5555` để xem và edit data

Useful Commands

```
# Generate Prisma Client
npx prisma generate

# Create migration
npx prisma migrate dev --name migration_name

# Apply migrations (production)
npx prisma migrate deploy

# Reset database
npx prisma migrate reset
```

```
# Pull schema từ database có sẵn
npx prisma db pull

# Push schema lên database (không tạo migration)
npx prisma db push

# Format schema file
npx prisma format

# Validate schema
npx prisma validate

# Open Prisma Studio
npx prisma studio
```

Tài liệu tham khảo

- Prisma Docs
- Prisma Schema Reference
- Prisma Client API