

UNISENAC-PELOTAS

MODELAGEM DE DADOS COM PRISMA

PROF ALINE TIMM



OBJETIVOS DE APRENDIZAGEM:



Objetivos da Aula:

Ao final desta aula, o aluno será capaz de:

1. Entender os conceitos fundamentais de um banco de dados relacional.
2. Explicar o que é um ORM (Object-Relational Mapper) e por que usá-lo.
3. Modelar entidades de dados usando a sintaxe do Prisma.
4. Implementar os três tipos de relacionamentos: One-to-One, One-to-Many e Many-to-Many.
5. Entender e utilizar o sistema de migrations para versionar o banco de dados de forma segura.
6. Realizar operações básicas de CRUD (Create, Read, Update, Delete) usando o Prisma Client.



VAMOS REVISAR JUNTOS O CÓDIGO DO AUTORESROUTER.JS QUE VOCÊS CRIARAM.

- Analisar uma query como: `pool.query('UPDATE AUTOR SET nome = ?, nacionalidade = ? WHERE autor_id = ?', [nome, nacionalidade, id]).`
- O que acontece se errarmos o nome de uma coluna na string?
- Como o editor de código poderia nos ajudar aqui?
- Isso é seguro?
- É fácil de ler e dar manutenção?



APRESENTANDO O ORM COMO SOLUÇÃO

- **O que é um ORM (Object-Relational Mapper)?** É uma ponte que nos permite usar objetos e métodos (como `prisma.autor.update()`) em vez de escrever SQL.
- **Benefícios diretos para nosso projeto:**
- **Segurança e Produtividade:** Fim dos erros de digitação em SQL e autocompletar no VS Code!
- **Código Mais Limpo:** Lógica de banco em JavaScript/TypeScript, não em strings.
- **Gerenciamento de Schema:** Chega de rodar ALTER TABLE manualmente. As migrations cuidam disso.
- **Prisma:** A ferramenta moderna que usaremos para fazer esse "upgrade".



O QUE É O PRISMA

- Pense no Prisma como um tradutor e assistente inteligente para o seu banco de dados. Ele é um ORM (Object-Relational Mapper) de última geração para Node.js e TypeScript.
- Em vez de você escrever SQL manualmente em formato de texto (strings), o Prisma permite que você interaja com seu banco de dados usando objetos e métodos JavaScript, de forma muito mais segura e intuitiva.



ELE É COMPOSTO POR TRÊS FERRAMENTAS PRINCIPAIS:

- **Prisma Client:** Um construtor de consultas gerado automaticamente para o seu projeto. É com ele que você vai escrever `prisma.autor.create(...)` em vez de `INSERT INTO AUTOR....`
- **Prisma Migrate:** Uma ferramenta para versionar a estrutura (schema) do seu banco de dados. É como um "Git para o banco de dados", que substitui a necessidade de rodar `ALTER TABLE` manualmente.
- **Prisma Studio:** Uma interface gráfica e visual para ver e editar os dados do seu banco, excelente para desenvolvimento e depuração.



POR QUE USAR O PRISMA EM VEZ DO SQL MANUAL QUE JÁ USAMOS?



POR QUE USAR O PRISMA EM VEZ DO SQL MANUAL QUE JÁ USAMOS?

No nosso projeto, estamos usando um pool de conexões para executar SQL, o que funciona, mas tem algumas desvantagens que o Prisma resolve:

Erros de Digitação: Escrever `SELECT * FROM AUTORE` (erro de digitação) só quebra a aplicação quando o código é executado.

Código Verboso: `INSERT INTO AUTOR (nome, nacionalidade) VALUES (?, ?)` é um texto que não se integra bem ao JavaScript.

Relacionamentos Complexos: Buscar um autor e todos os seus livros exige a escrita de JOINS em SQL, que podem ficar complexos.



PRINCIPAIS CONCEITOS

- **schema.prisma:** É o coração do Prisma. Um único arquivo que se torna a fonte da verdade sobre a estrutura do seu banco. Nele, você define os models.
- **Modelos (Models):** Um model no schema representa uma tabela no seu banco de dados. `model AUTOR { ... }` mapeia diretamente para a sua tabela `AUTOR`.
- **Migrations:** São os arquivos gerados pelo Prisma que registram cada passo da evolução da estrutura do seu banco de dados, permitindo que você aplique ou reverta mudanças de forma segura.



INSTALANDO E CONHECENDO O PRISMA

1

```
npm install prisma --save-dev.
```

2

```
npx prisma init
```

O ARQUIVO PRISMA/SCHEMA.PRISMA

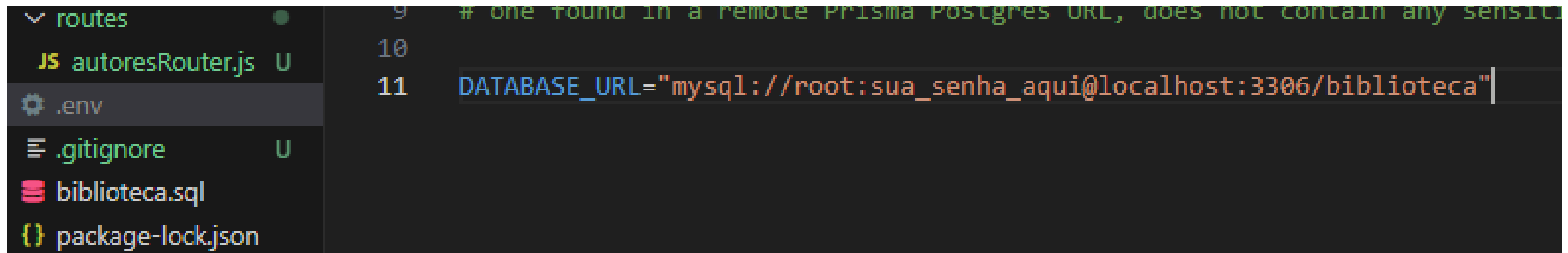
O que é? Este é o coração do Prisma. É um arquivo de modelo (schema) onde você irá definir três coisas principais:

1. **Fonte de Dados (datasource):** Como o Prisma deve se conectar ao seu banco de dados. Ele lê a DATABASE_URL do arquivo .env.
2. **Gerador (generator):** Qual cliente de banco de dados o Prisma deve gerar. Quase sempre será o prisma-client-js.
3. **Modelos de Dados (model):** A parte mais importante. Aqui você descreve a estrutura das suas tabelas (como LIVRO, AUTOR, GENERO) usando uma sintaxe simples e legível.



INSTALANDO E CONHECENDO O PRISMA

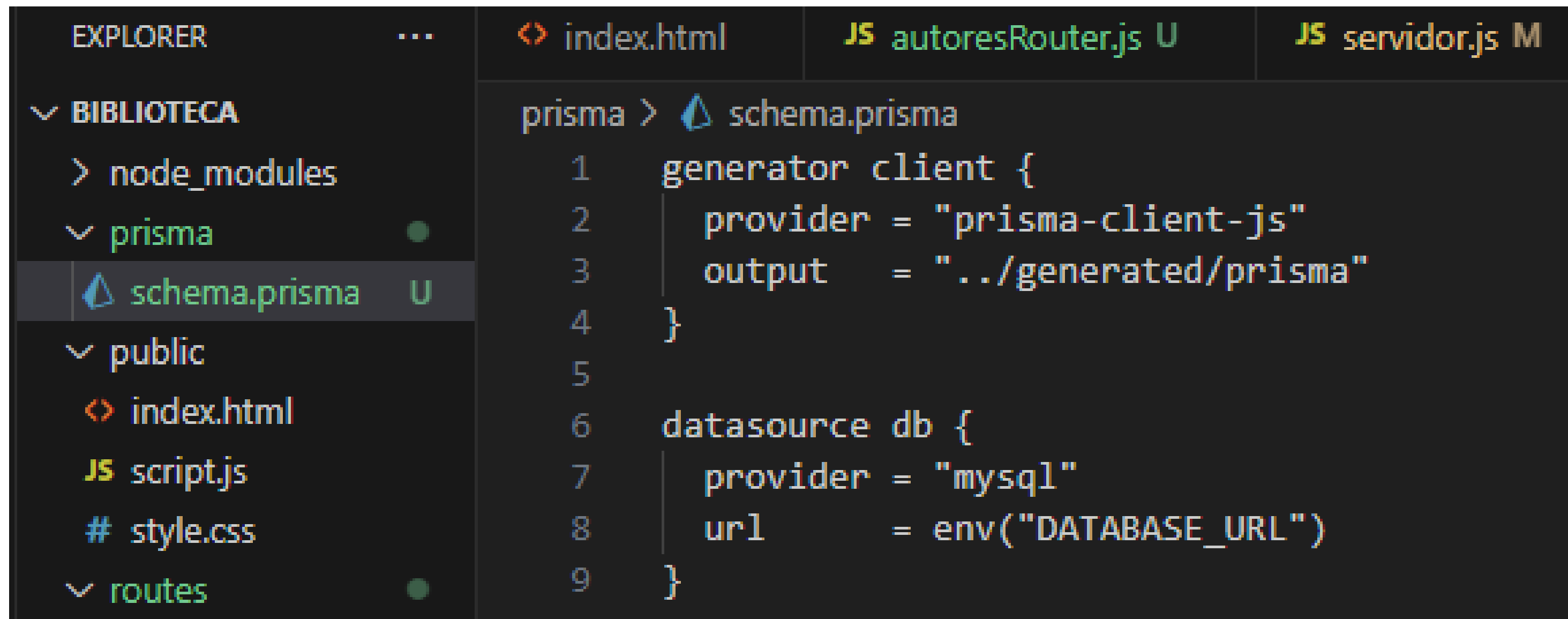
Configure seu arquivo .env da seguinte forma:

A screenshot of a code editor with a dark theme. On the left, a file explorer shows a project structure with files: routes, JS autoresRouter.js, .env (selected), .gitignore, biblioteca.sql, and package-lock.json. The main editor area shows the content of .env. Line 9 has a comment in green: '# one found in a remote Prisma Postgres URL, does not contain any sensitive'. Line 10 is empty. Line 11 contains the text 'DATABASE_URL="mysql://root:sua_senha_aqui@localhost:3306/biblioteca"' in a monospace font, with the cursor at the end of the string.

```
9 # one found in a remote Prisma Postgres URL, does not contain any sensitive
10
11 DATABASE_URL="mysql://root:sua_senha_aqui@localhost:3306/biblioteca"
```

INSTALANDO E CONHECENDO O PRISMA

E o Schema assim:



The image shows a screenshot of a code editor (VS Code) with a dark theme. On the left, the 'EXPLORER' sidebar is visible, showing a file tree. The tree is expanded to show the 'prisma' directory, which contains a file named 'schema.prisma'. The file is highlighted with a blue background. To the right of the sidebar, the main editor area displays the content of 'schema.prisma'. The code is as follows:

```
prisma > schema.prisma
1  generator client {
2    provider = "prisma-client-js"
3    output   = "../generated/prisma"
4  }
5
6  datasource db {
7    provider = "mysql"
8    url      = env("DATABASE_URL")
9  }
```

Em vez de escrever o schema do zero, vamos pedir para o Prisma ler nosso banco de dados biblioteca existente!

Ajuste a DATABASE_URL no .env primeiro, entre na pasta e diga onde o prisma vai encontrar o banco de dados, depois use o comando abaixo para migrar os dados do prisma para a tabela:

3

```
npx prisma db pull
```



RESULTADO:

```
prisma > schema.prisma
1  generator client {
2    provider = "prisma-client-js"
3    output   = "../generated/prisma"
4  }
5
6  datasource db {
7    provider = "mysql"
8    url      = env("DATABASE_URL")
9  }
10
11 model autor {
12   autor_id      Int      @id @default(autoincrement())
13   nome          String    @db.VarChar(255)
14   nacionalidade String?  @db.VarChar(100)
15   livro_autor   livro_autor[]
16 }
17
18 model editora {
19   Cod_Editora  Int      @id
20   Nome_Editora String    @db.VarChar(100)
21   Endereco     String?  @db.VarChar(255)
22 }
23
24 model genero {
25   genero_id Int      @id @default(autoincrement())
26   nome      String   @unique(map: "nome") @db.VarChar(100)
27   livro      livro[]
28 }
29
30 model livro {
31   livro_id      Int      @id @default(autoincrement())
32   titulo        String    @db.VarChar(255)
33   ano_publicacao Int?
34   genero        String?   @db.VarChar(100)
35   excluido      Boolean?  @default(false)
36   genero_id     Int?
37   genero        genero?   @relation(fields: [genero_id], references: [genero_id], onDelete: NoAction, onUpdate: NoAction, map: "livro_ibfk_1")
38   livro_autor   livro_autor[]
39
40   @@index([genero_id], map: "genero_id")
41 }
```

GERAR O PRISMA CLIENT

Agora que o schema está pronto, precisamos gerar o cliente que usaremos no nosso código para fazer as consultas.

5

```
npx prisma generate
```

Pronto! O Prisma já foi configurado!

AGORA VAMOS FAZER NOSSA PRIMEIRA MODIFICAÇÃO COM O PRISMA: ADICIONAR UMA COLUNA NOVA COM MIGRATE

- Vamos aplicar a opção do livro estar ou não disponível. No Model livros, vamos fazer a seguinte modificação:

```
model livro {  
  livro_id      Int      @id @default(autoincrement())  
  titulo        String   @db.VarChar(255)  
  ano_publicacao Int?  
  disponivel    Boolean  @default(true) // Por padrão, um livro novo está disponível.  
  excluido      Boolean? @default(false)  
  genero_id     Int?  
  genero        genero?  @relation(fields: [genero_id], references: [genero_id], onDelete  
  livro_autor   livro_autor[]  
  @@index([genero_id], map: "genero_id")  
}
```

RODE A MIGRATION: NO TERMINAL, EXECUTE:

```
npx prisma migrate dev --name "schema01"
```

ATENÇÃO

Caso você esteja com versões diferentes entre seu banco e seu Schema,
rode:

```
> npx prisma migrate reset
```

E depois sim rode:

```
npx prisma migrate dev --name "schema01"
```

ENTENDENDO O PRISMA CLIENT

Ao rodar `prisma migrate dev`, o Prisma fez duas coisas:

1. Alterou o banco de dados: Ele executou os comandos SQL para criar tabelas e colunas.
2. Gerou o Prisma Client: Ele leu seu `schema.prisma` e criou um conjunto de funções e objetos JavaScript feitos sob medida para o seu banco. É um cliente de banco de dados 100% tipado e com autocompletar, que "sabe" que você tem autores, livros e os relacionamentos entre eles.

VAMOS PRATICAR?

[Acesse o link](#)



AGORA É SUA VEZ!

OBRIGADA!

Unisenac-Pelotas

