

<b>1. O QUE É PRISMA.JS?</b>	<b>2</b>
1.1 ESTRUTURANDO O PROJETO	2
<b>2. TRABALHANDO COM TABELAS</b>	<b>5</b>
2.1 CRIANDO TABELA COURSES	5
2.2 CRIANDO TABELA MODULES	7
2.3 ADD REGISTRO NA TABELA COURSES	8
2.4 LISTANDO REGISTROS NA TABELA COURSES	10
2.4.1 CLÁUSULA WHERE - CONDICIONAL	10
2.4.2 CLÁUSULA WHERE - ÚLTIMO REGISTRO	11
2.5 EDITANDO REGISTRO NA TABELA COURSES	11
2.6 EXCLUINDO REGISTRO NA TABELA COURSES	12
<b>3. RELACIONAMENTO ONE-TO-ONE (1:1)</b>	<b>13</b>
3.1 RELACIONAMENTO TABELA TEACHER E COURSES	14
3.2 CRIAR REGISTRO QUE PODE EXISTIR - CONNECT OR CREATE	16
3.2 CRIAR REGISTRO QUE EXISTE - CONNECT	17
3.3 CRIAR REGISTRO QUE NÃO EXISTE - CREATE	18
3.4 CRIAR REGISTRO PELO FK DA CHAVE ESTRANGEIRA	19
3.5 BUSCA EM MÚLTIPLAS TABELAS	20
<b>4. RELACIONAMENTO ONE-TO-MANY (1:N)</b>	<b>22</b>
4.1 RELACIONAMENTO TABELA AUTHORS E BOOKS (1:N)	22
4.2 CRIAR VÁRIOS REGISTROS - CREATE MANY	24
<b>5. RELACIONAMENTO MANY-TO-MANY (N:N)</b>	<b>26</b>
5.1 RELACIONAMENTO COURSES_MODULES (N:N)	26
5.2 CRIANDO REGISTROS EM RELACIONAMENTO MUITO PARA MUITOS (N:N)	28
5.3 BUSCANDO DADOS EM RELACIONAMENTOS N:N	32
5.4 REMOVENDO DADOS EM RELACIONAMENTOS N:N	33
<b>6. IMPORTANDO OS DADOS</b>	<b>34</b>

# 1. O QUE É PRISMA.JS?

O ORM [Prisma](#) é uma ferramenta de mapeamento objeto-relacional (ORM) moderna e poderosa para bancos de dados. Ele facilita a interação entre a aplicação e o banco de dados, permitindo que os desenvolvedores **escrevam consultas de banco de dados usando uma sintaxe familiar de linguagem de programação**, em vez de consultas SQL diretamente.

O Prisma é uma ferramenta de código aberto que oferece suporte a vários bancos de dados populares, como PostgreSQL, MySQL, MongoDB, SQLite e outros. Ele permite que os desenvolvedores definam modelos de dados usando uma linguagem de definição de modelo declarativa e, em seguida, fornece métodos para realizar operações de leitura, gravação, atualização e exclusão (CRUD) nesses modelos de dados.

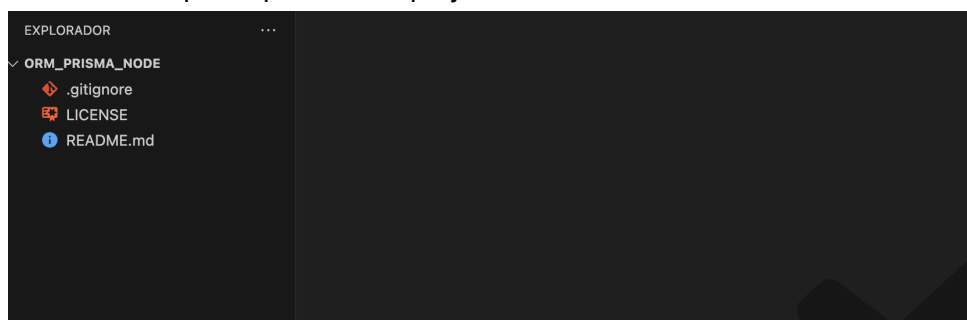
O Prisma também oferece suporte a recursos avançados, como transações, relacionamentos entre tabelas, migrações de banco de dados e consultas complexas. Além disso, ele é compatível com muitos frameworks e tecnologias de back-end populares, o que o torna uma escolha atraente para muitos desenvolvedores e equipes de desenvolvimento.

Saiba mais:

<https://www.prisma.io/docs/concepts/overview/what-is-prisma>

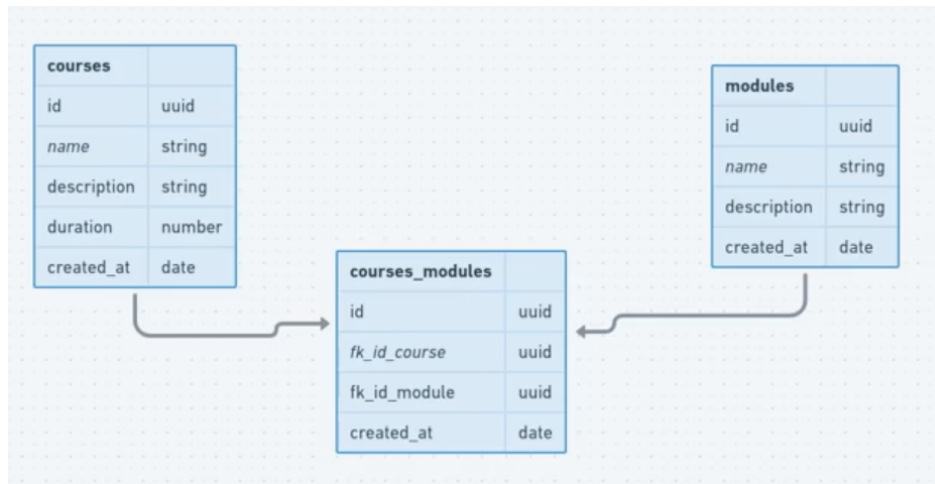
## 1.1 ESTRUTURANDO O PROJETO

Inicialmente, crie uma pasta para o seu projeto.



No VScode, instale as extensões: [Prisma](#) e [Prisma-Insider](#) e siga as instruções de configurações.

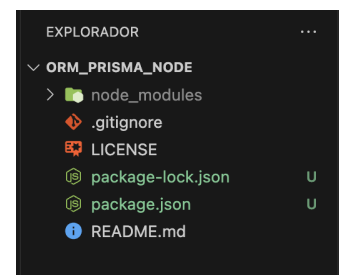
Para essa aula, inicialmente iremos utilizar o modelo de DB abaixo:



Instale o Prisma como dependência de desenvolvimento

**`npm i prisma -D`**

Após a instalação do Prisma será criado a pasta `node_modules` com todas as dependências e os arquivos `package.json` e `package-lock.json`.



Faça a inicialização do Prisma.

**`npx prisma init`**

*As instruções abaixo serão apresentadas como guia inicial.*

✓ Your Prisma schema was created at `prisma/schema.prisma`  
You can now open it in your favorite editor.

warn You already have a `.gitignore` file. Don't forget to add ``.env`` in it to not commit any private information.

Next steps:

1. Set the `DATABASE_URL` in the `.env` file to point to your existing database. If your database has no tables yet, read <https://pris.ly/d/getting-started>
2. Set the provider of the `datasource` block in `schema.prisma` to match your database: `postgresql`, `mysql`, `sqlite`, `sqlserver`, `mongodb` or `cockroachdb`.
3. Run `prisma db pull` to turn your database schema into a Prisma schema.
4. Run `prisma generate` to generate the Prisma Client. You can then start querying your database.

More information in our documentation:  
<https://pris.ly/d/getting-started>

Observe que uma pasta denominada Prisma é criada na raiz do projeto, contendo um arquivo chamado `schema.prisma`. Neste arquivo é possível definir o tipo de banco de dados e um link para uma variável de ambiente contendo o endereço do banco de dados utilizado. Logo abaixo é apresentado o conteúdo do arquivo `.env` gerado.

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

Realizei a seguinte alteração:

```
DATABASE_URL="mysql://root:123456789@localhost:3306/ormprisma?schema=public"
```

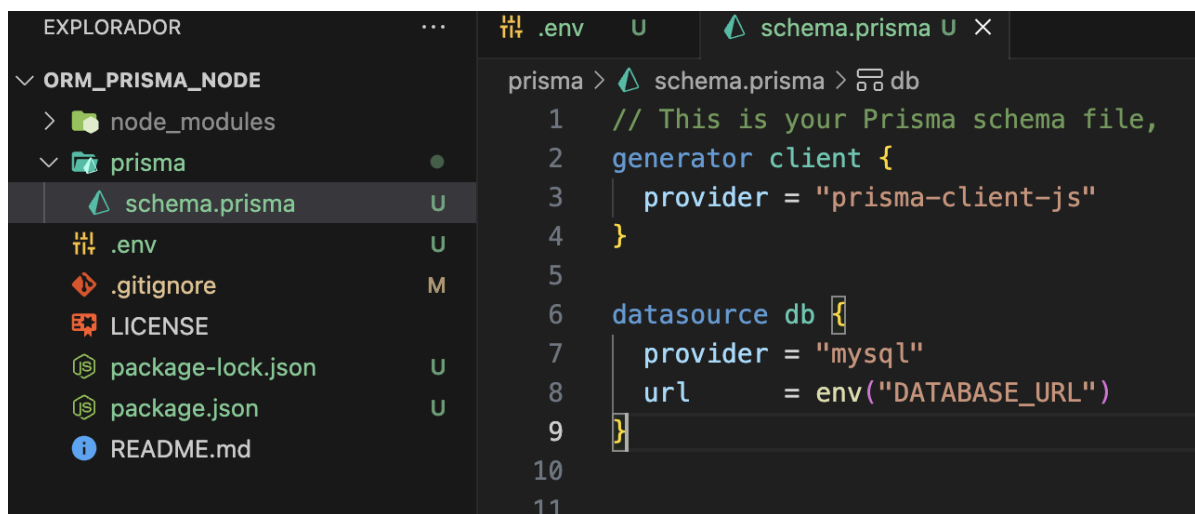
Tipo de banco: MySQL

Usuário: root

Senha: 123456789

Porta: 3306

Nome do banco: prismafundamentals



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files like .env, .gitignore, LICENSE, package-lock.json, package.json, and README.md. The code editor shows the content of the schema.prisma file, which is a Prisma schema file. The schema defines a client and a data source named 'db'.

```
prisma > schema.prisma > db
1 // This is your Prisma schema file,
2 generator client {
3   provider = "prisma-client-js"
4 }
5
6 datasource db {
7   provider = "mysql"
8   url      = env("DATABASE_URL")
9 }
10
11
```

Arquivo schema.prisma:

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

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

Crie o Banco de Dados:

**create database prismafundamentals**

Link do Github:

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/aa676483d99387d674e32dc429baddf18b97291b](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/aa676483d99387d674e32dc429baddf18b97291b)

## 2. TRABALHANDO COM TABELAS

### 2.1 CRIANDO TABELA COURSES

Vamos criar a tabela **Courses**!

```
model Courses {  
  id String @id @default(uuid())  
  name String @unique  
  description String?  
  duration Int  
  created_at DateTime @default(now())  
  
  @@map("courses")  
}
```

**@unique** significa que o tipo é único, ou seja, UNIQUE.

**UUID** significa "Universally Unique Identifier" (Identificador Único Universal) e é um identificador padrão utilizado em software e sistemas para identificar de forma única informações, componentes ou recursos. O UUID é um identificador de 128 bits frequentemente representado por 32 dígitos hexadecimais, geralmente exibidos em cinco grupos separados por hifens, na forma 8-4-4-4-12, por exemplo, "550e8400-e29b-41d4-a716-446655440000". Esses identificadores são gerados de forma que a probabilidade de gerar o mesmo identificador em outro sistema é extremamente baixa.

A interrogação significa que o campo é opcional. **@default** significa que o campo será um valor padrão, quando não atribuído. Com o uso de **@@map**, você pode especificar explicitamente o nome da tabela no banco de dados para a qual um modelo do Prisma deve ser mapeado.

Para gerar a tabela e seus respectivos campos no bd, iremos utilizar o comando abaixo:

```
npx prisma migrate dev
```

Será solicitado um nome para a *migrate* (coloquei create\_courses) e depois gerado a tabela no bd.

```

● brunobandeirafernandes@MacBook-Pro-de-Bruno ORM_Prisma_Node % npx prisma migrate dev

Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma
Datasource "db": MySQL database "ormprisma" at "localhost:3306"

✓ Enter a name for the new migration: ... create_courses
Applying migration `20231023021055_create_courses`

The following migration(s) have been created and applied from new schema changes:

migrations/
├─ 20231023021055_create_courses/
│   └─ migration.sql

Your database is now in sync with your schema.

Running generate... (Use --skip-generate to skip the generators)

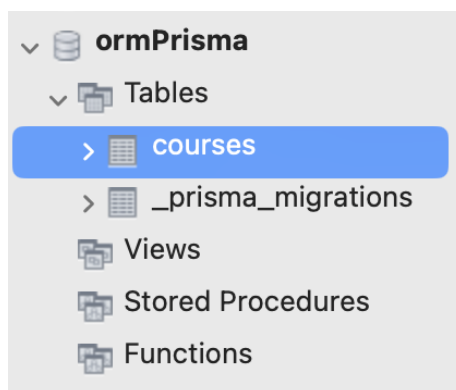
added 2 packages, and audited 5 packages in 6s

found 0 vulnerabilities

✓ Generated Prisma Client (v5.4.2) to ./node_modules/@prisma/client in 39ms

```

No db foi criado a tabela Courses, conforme especificação.



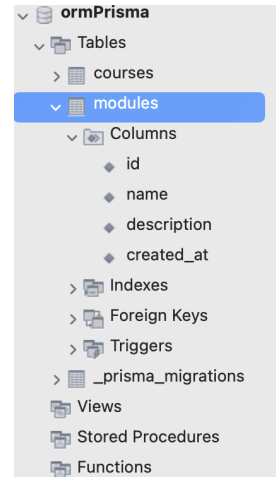
**Link do Github:**

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/aa676483d99387d674e32dc429baddf18b97291b](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/aa676483d99387d674e32dc429baddf18b97291b)

## 2.2 CRIANDO TABELA MODULES

Seguindo o mesmo procedimento para criação da tabela Courses, agora vamos criar a tabela **Modules**!

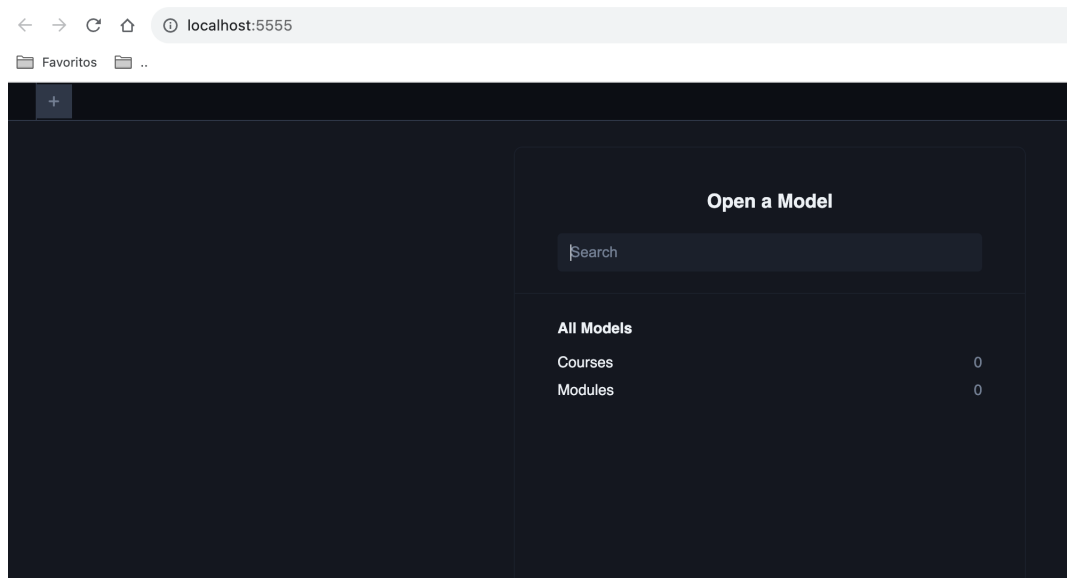
```
model Modules {  
  id String @id @default(uuid())  
  name String @unique  
  description String  
  created_at DateTime @default(now())  
  
  @@map("modules")  
}
```



Para gerar utilize o comando `npx prisma migrate dev` novamente. Coloquei o nome `create_table_modules` como nome da migrate gerada.

Para visualizar os dados de maneira mais simplificada, iremos utilizar o Prisma Studio, responsável por manipular os dados diretamente no navegador.

**`npx prisma studio`**



Com o Prisma Studio é possível visualizar as tabelas criadas, bem como realizar manipulação dos dados nas tabelas.

**Link do Github:**

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/ea12ce4c967cf9377af3e992825bfe36f0cc671c](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/ea12ce4c967cf9377af3e992825bfe36f0cc671c)

## 2.3 ADD REGISTRO NA TABELA COURSES

Crie uma pasta **src**, em seguida **Courses** e um arquivo chamado **create.ts**.

```
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

async function main() {
  const result = await prisma.courses.create({
    data: {
      name: "Curso de React Native",
      duration: 200,
      description: "Curso de Apps com React Native",
    },
  });

  console.log(result);
}

main();
```

Para executar o arquivo criado, utilize o comando:

```
npx ts-node-dev src/Courses/create.ts
```

Para executar o arquivo em JS, primeiramente configure o arquivo package.json para aceitar importações via module:

```
"type": "module"
```

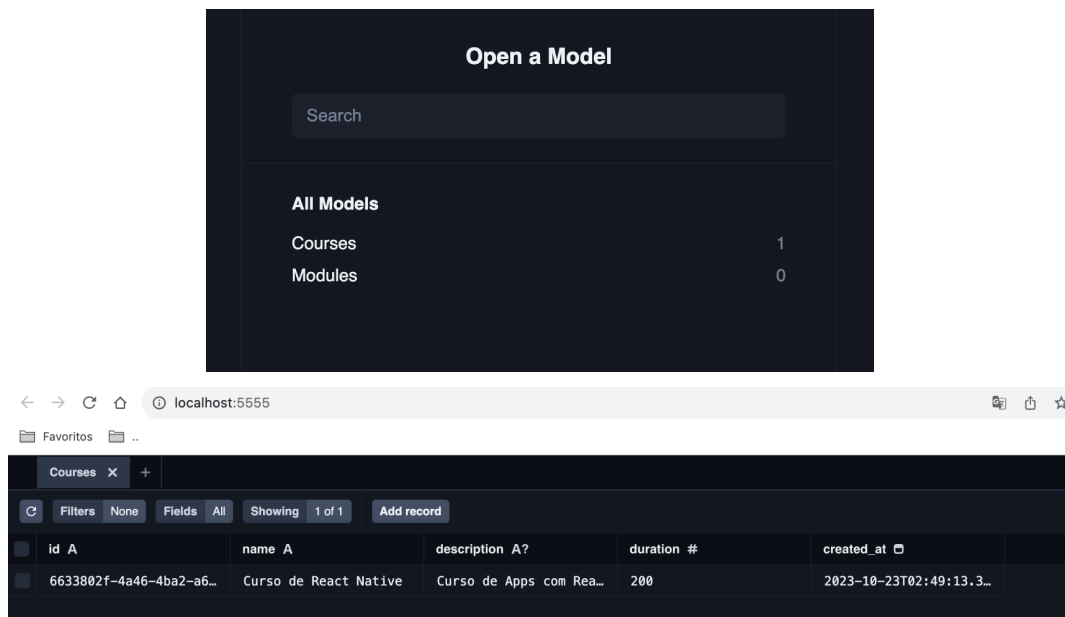
Para executar o arquivo, utilize o comando:

```
node src/Courses/create.ts
```

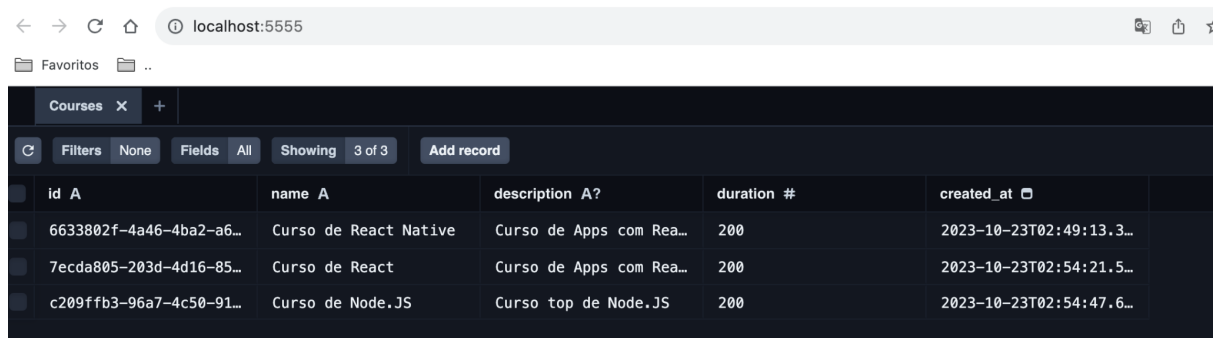
```
● brunobandeirafernandes@MacBook-Pro-de-Bruno ORM_Prisma_Node % npx ts-node-dev src/Courses/create.ts
[INFO] 23:49:12 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.1, typescript ver. 5.1.3)
{
  id: '6633802f-4a46-4ba2-a6cf-5ff5b88624ac',
  name: 'Curso de React Native',
  description: 'Curso de Apps com React Native',
  duration: 200,
  created_at: 2023-10-23T02:49:13.331Z
}
```



Realmente, os dados foram inseridos na tabela Courses.



Execute o arquivo novamente e faça a inserção de novos registros!



Link do Github:

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/c99e574dbab0ebdb7ee181289c602d6a67b5804f](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/c99e574dbab0ebdb7ee181289c602d6a67b5804f)

## 2.4 LISTANDO REGISTROS NA TABELA COURSES

Na pasta **Courses**, crie um arquivo chamado **findMany.ts**.

```
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

async function main() {
  const courses = await prisma.courses.findMany();
  console.log(courses);
}

main();
```

O resultado no console:

```
● brunobandeirafernandes@MacBook-Pro-de-Bruno ORM_Prisma_Node % npx ts-node-dev src/Courses/findMany.ts
[INFO] 00:04:25 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.1, typescript ver. 5.1.3)
[
  {
    id: '6633802f-4a46-4ba2-a6cf-5ff5b88624ac',
    name: 'Curso de React Native',
    description: 'Curso de Apps com React Native',
    duration: 200,
    created_at: 2023-10-23T02:49:13.331Z
  },
  {
    id: '7ecda805-203d-4d16-850a-30f72715403b',
    name: 'Curso de React',
    description: 'Curso de Apps com React',
    duration: 200,
    created_at: 2023-10-23T02:54:21.548Z
  },
  {
    id: 'c209ffb3-96a7-4c50-91d6-38ad4248bdc2',
    name: 'Curso de Node.JS',
    description: 'Curso top de Node.JS',
    duration: 200,
    created_at: 2023-10-23T02:54:47.677Z
  }
]
```

Link do Github:

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/61a372b7b86bf475977a20b35dee6f1c09176b08](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/61a372b7b86bf475977a20b35dee6f1c09176b08)

### 2.4.1 CLÁUSULA WHERE - CONDICIONAL

Retornar as informações do curso que tenha o nome "Curso de React Native"

```
async function main() {
  const courses = await prisma.courses.findMany({
```

```

    where: {
      name: {
        equals: 'Curso de React Native',
      }
    }
  });
  console.log(courses);
}

```

## 2.4.2 CLÁUSULA WHERE - ÚLTIMO REGISTRO

Para mais informações, consulta a documentação oficial

<https://www.prisma.io/docs/concepts/components/prisma-client/crud#read>

## 2.5 EDITANDO REGISTRO NA TABELA COURSES

Na pasta **Courses**, crie um arquivo chamado **update.ts**.

```

import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

async function main() {
  const result = await prisma.courses.update({
    where: {
      id: "29d984de-3e8c-4df7-ae1-ac5554156c5a",
    },
    data: {
      duration: 300,
      name: "Curso de React Native - v2",
      description: "Curso muito bom de React Native",
    },
  });

  console.log(result);
}

```

```
}  
  
main();
```

**Link do Github:**

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/fbae08334a1b4e0214dcd3952002f96420ee7210](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/fbae08334a1b4e0214dcd3952002f96420ee7210)

## 2.6 EXCLUINDO REGISTRO NA TABELA COURSES

Na pasta **Courses**, crie um arquivo chamado **delete.ts**.

```
import { PrismaClient } from "@prisma/client";  
  
const prisma = new PrismaClient();  
  
async function main() {  
  const result = await prisma.courses.delete({  
    where: {  
      id: "29d984de-3e8c-4df7-ae1-ac5554156c5a",  
    },  
  });  
  
  console.log(result);  
}  
  
main();
```

**Link do Github:**

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/191ce349172ff1abbd4455b49c636b508d663782](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/191ce349172ff1abbd4455b49c636b508d663782)

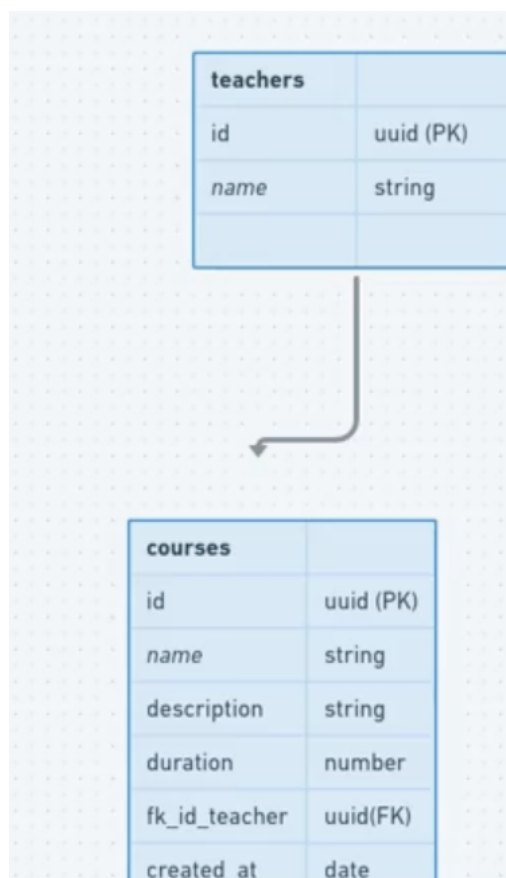
### 3.RELACIONAMENTO ONE-TO-ONE (1:1)

Agora iremos trabalhar com relacionamento um para um. Um relacionamento um para um em bancos de dados relacionais é um tipo de relação entre duas entidades em que uma instância de uma entidade está associada a no máximo uma instância de outra entidade. Isso significa que cada registro na tabela A está relacionado a no máximo um registro na tabela B e vice-versa.

Um exemplo comum é um relacionamento entre duas entidades, como "Pessoa" e "Passaporte". Cada pessoa pode ter no máximo um passaporte, e cada passaporte está associado a apenas uma pessoa.

Geralmente, esse tipo de relacionamento é representado através de chaves estrangeiras. Em um banco de dados relacional, a chave estrangeira em uma tabela aponta para a chave primária em outra tabela, estabelecendo assim o vínculo entre os dados de ambas as tabelas.

No nosso contexto, utilizaremos a ideia de que um professor pode ser vinculado a somente um curso e um curso terá somente um professor, conforme como segue a imagem abaixo:



Neste caso, o id do professor terá que estar vinculado a tabela Curso (fk\_id\_teacher). Aqui utilizaremos a ideia de chave estrangeira.

## 3.1 RELACIONAMENTO TABELA TEACHER E COURSES

Primeira iremos criar a tabela **Teacher**:

```
model Teachers {  
  id String @id @default(uuid())  
  name String @unique  
  
  course Courses?  
  
  @@map("teachers")  
}
```

A linha `course Courses?` informa que o campo é opcional e que faz referência a tabela `Courses`.

Na tabela `Courses` iremos adicionar o campo `fk_id_teacher`:

```
teacher Teachers @relation(fields: [fk_id_teacher], references: [id])  
fk_id_teacher String @unique
```

O código completo das tabelas `Teacher` e `Courses` segue abaixo para consulta:

```
model Courses {  
  id String @id @default(uuid())  
  name String @unique  
  description String?  
  duration Int  
  created_at DateTime @default(now())  
  
  teacher Teachers @relation(fields: [fk_id_teacher], references: [id])  
  fk_id_teacher String @unique  
  
  @@map("courses")  
}  
  
model Teachers {  
  id String @id @default(uuid())  
  name String @unique  
  
  course Courses?
```

```

@@map("teachers")
}

```

Para gerar a tabela e seus respectivos campos no bd, iremos utilizar o comando abaixo:

**npx prisma migrate dev**

The screenshot shows the Prisma Studio interface. On the left, a file explorer displays the project structure, including the Prisma schema file at `src/Courses/schema.prisma`. The main editor shows the Prisma schema for the `Courses` model, which includes a `teachers` relationship. The schema is as follows:

```

model Courses {
  id          VARCHAR(191) NOT NULL,
  name        VARCHAR(191) NOT NULL,
  description  TEXT
  duration    INT
  created_at  DATETIME
  teachers    Teachers[]
  fk_id_teacher VARCHAR(191) NOT NULL
  @@id(id)
  @@defaultCharacterSet(utf8mb4)
  @@collate(utf8mb4_unicode_ci)
}

model Teachers {
  id          VARCHAR(191) NOT NULL,
  name        VARCHAR(191) NOT NULL,
  @@id(id)
  @@defaultCharacterSet(utf8mb4)
  @@collate(utf8mb4_unicode_ci)
}

Courses } Teachers

```

The right sidebar shows the migration files, including `migration.sql` and `migration_lock.toml`. The `migration.sql` file contains the following SQL commands:

```

ALTER TABLE `courses` ADD COLUMN `fk_id_teacher` VARCHAR(191) NOT NULL;

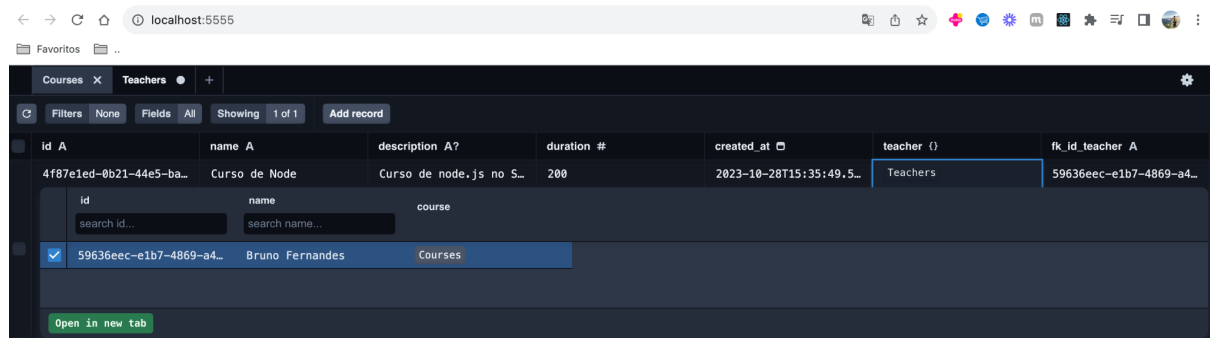
-- CreateTable
CREATE TABLE `teachers` (
  `id` VARCHAR(191) NOT NULL,
  `name` VARCHAR(191) NOT NULL,
  PRIMARY KEY (`id`),
  DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
);

-- CreateIndex
CREATE UNIQUE INDEX `courses_fk_id_teacher_key` ON `courses` (`fk_id_teacher`);

-- AddForeignKey
ALTER TABLE `courses` ADD CONSTRAINT `courses_fk_id_teacher_fkey` FOREIGN KEY (`fk_id_teacher`) REFERENCES `teachers` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

```

Observe no Prisma Studio que é possível adicionar um professor. Após adicionado o professor, é possível adicionar um novo curso e selecionar facilmente os professores cadastrados.



Link do Github:

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/0889f88f7730f5692b0df595934a959e856cbb48](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/0889f88f7730f5692b0df595934a959e856cbb48)

## 3.2 CRIAR REGISTRO QUE PODE EXISTIR - CONNECT OR CREATE

Como alteramos a estrutura da tabela, os comandos criados anteriormente não irão mais funcionar corretamente, isso porque agora na Tabela Course possui um relacionamento com a tabela Teacher. Dessa forma, irei renomear a pasta Courses para Old\_Courses e criar uma nova pasta Courses para os novos comandos.

Para fazer a inserção de registros vinculados a chave estrangeira, podemos utilizar quatro abordagens:

- **ConnectOrCreate:** se a chave estrangeira não existe, será criada.
- **Connect:** conecta uma chave estrangeira existente.
- **Create:** cria uma nova chave estrangeira
- **Vinculando a fk:** vincula a uma chave estrangeira existente.

O primeiro comando para criar registros utilizando o Prisma é utilizando o comando connectOrCreate, onde será criado os registros em ambas as tabelas, ou seja, nas tabelas Teacher e Courses. Porém, caso não exista na tabela Teacher, será criado

Observe o campo abaixo:

```
teacher: {
  connectOrCreate: {
    where: {
      name: "Bruno Fernandes",
    },
    create: {
      name: "Bruno Bandeira",
    },
  },
},
```

O comando está incompleto, pois é para inserir os dados na tabela Courses. Porém observe que será criado um usuário de nome "Bruno Fernandes". Ou seja, os dados serão inseridos na tabela Courses e se existir um usuário de nome Bruno Fernandes, porém se não estiver o nome informado do parâmetro, será criado o registro na outra tabela. Por isso o comando connect or create (conecte ou crie).

Crie um arquivo chamado **create.ts** com seguinte código (completo).

```
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

async function main() {
```



```

const result = await prisma.courses.create({
  data: {
    name: "Curso de React Native",
    duration: 300,
    description: "Curso com 300h de React Native",

    teacher: {
      connectOrCreate: {
        where: {
          name: "Bruno Fernandes",
        },
        create: {
          name: "Bruno Fernandes",
        },
      },
    },
  },
});

console.log(result);
}

main();

```

Muita atenção ao adicionar um registro utilizando connectOrCreate, pois definimos na tabela Courses que a chave estrangeira fk\_id\_teacher possui valor único. Portanto, o comando de create só deverá funcionar se for criar um registro que não existe, ou seja, um novo Teacher.

#### Link do Github:

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/b0628906298039332f1f61c02455c48664ca46e5](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/b0628906298039332f1f61c02455c48664ca46e5)

## 3.2 CRIAR REGISTRO QUE EXISTE - CONNECT

Para criar um registro, partindo do princípio que a chave estrangeira já existe, utilizaremos o comando connect. Observe o código abaixo:

```

import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

```

```

async function main() {
const result = await prisma.courses.create({
  data: {
    name: "Curso de Prisma",
    duration: 50,
    description: "Curso sobre como utilizar ORM Prisma.js",
    teacher: {
      connect: {
        id: "51af9063-88a0-4980-83a1-3c9d4c26a4a5",
      },
    },
  },
});

console.log(result);
}

main();

```

Lembre-se que definimos o relacionamento entre Teacher e Courses como sendo one-to-one, ou seja, um professor só poderá estar vinculado a um único curso, portanto, se deseja inserir um teacher com este método, lembre-se de registrar um teacher e não vincular a nenhum outro curso.

A imagem abaixo apresenta o código de sucesso ao inserir um professor sem vinculo com outro curso.

```

● brunobandeirafernandes@MacBook-Pro-de-Bruno ORM_Prisma_Node % npx ts-node-dev src/Courses/create_exist_teacher.ts
[INFO] 18:57:28 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.1, typescript ver. 5.1.3)
{
  id: '593037f0-e666-4716-9e5e-4d859c1cdc3c',
  name: 'Curso de Prisma',
  description: 'Curso sobre como utilizar ORM Prisma.js',
  duration: 50,
  created_at: 2023-10-29T21:57:28.846Z,
  fk_id_teacher: '51af9063-88a0-4980-83a1-3c9d4c26a4a5'
}

```

**Link do Github:**

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/827699f04c5d28b1fc0610dda66adcc7a149d710](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/827699f04c5d28b1fc0610dda66adcc7a149d710)

### 3.3 CRIAR REGISTRO QUE NÃO EXISTE - CREATE

Para criar um registro onde a chave estrangeira não existe, utilizaremos o comando create. Observe o código abaixo:

```

import { PrismaClient } from "@prisma/client";

```

```

const prisma = new PrismaClient();

async function main() {
  const result = await prisma.courses.create({
    data: {
      name: "Curso de ORM com Node.js",
      duration: 100,
      description: "Curso excelente sobre os ORM com Node.js",
      teacher: {
        create: {
          name: "Fulano Ciclano",
        },
      },
    },
  });

  console.log(result);
}

main();

```

Com o comando create é possível criar um novo registro na tabela Teacher.

```

● brunobandeirafernandes@MacBook-Pro-de-Bruno ORM_Prisma_Node % npx ts-node-dev src/Courses/create_non_exist_teacher.ts
[INFO] 19:07:29 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.1, typescript ver. 5.1.3)
{
  id: '939867d1-ab6a-4ede-9eb2-5a41e2ce2cac',
  name: 'Curso de ORM com Node.js',
  description: 'Curso excelente sobre os ORM com Node.js',
  duration: 100,
  created_at: 2023-10-29T22:07:29.971Z,
  fk_id_teacher: 'd504f8ea-85e2-44ab-89e5-8f3b8de1684c'
}

```

**Link do Github:**

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/925afb20f9bc62f2c65005527f6159ba1e328445](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/925afb20f9bc62f2c65005527f6159ba1e328445)

## 3.4 CRIAR REGISTRO PELO FK DA CHAVE ESTRANGEIRA

Para criar um registro que existe, outra abordagem existente é vincular diretamente à chave estrangeira. Note que não é uma abordagem tão convencional, porque é preciso ter acesso direto a chave estrangeira a ser criada. Observe o código abaixo:

```

import { PrismaClient } from "@prisma/client";

```

```
const prisma = new PrismaClient();

async function main() {
  const result = await prisma.courses.create({
    data: {
      name: "Curso de CSS",
      duration: 50,
      description: "Curso top de CSS",
      fk_id_teacher: "f5277fb5-4c40-49ec-a62f-8559adec5a67",
    },
  });

  console.log(result);
}

main();
```

No atributo `fk_id_teacher` é informado diretamente o id do teacher a ser inserido.

```
● brunobandeirafernandes@MacBook-Pro-de-Bruno ORM_Prisma_Node % npx ts-node-dev src/Courses/create_fk_teacher.ts
[INFO] 19:18:07 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.1, typescript ver. 5.1.3)

{
  id: 'a40da5b7-de2e-4a10-b018-d427122faec8',
  name: 'Curso de CSS',
  description: 'Curso top de CSS',
  duration: 50,
  created_at: 2023-10-29T22:18:08.243Z,
  fk_id_teacher: 'f5277fb5-4c40-49ec-a62f-8559adec5a67'
}
```

a40da5b7-de2e-4a10-b0...	Curso de CSS	Curso top de CSS	50	2023-10-29T22:18:08.2...	Teachers	f5277fb5-4c40-49ec-a6...
id	name	course				
search id...	search name...					
<input checked="" type="checkbox"/>	f5277fb5-4c40-49ec-a6...	Ciclane Fulano	Courses			

Link do Github:

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/c91f8ef04a3444c744d439ba6156bf8e3ea41dde](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/c91f8ef04a3444c744d439ba6156bf8e3ea41dde)

## 3.5 BUSCA EM MÚLTIPLAS TABELAS

Como realizar uma busca na tabela Courses e retornar os dados do professor vinculado ao respectivo curso?

Utilizaremos o **findMany** para buscar todos os dados e como solução para retornar todos os dados dos professores, precisaremos adicionar um booleano `true` no `include` do teacher. Veja como é simples:

```
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

async function main() {
  const result = await prisma.courses.findMany({
    include: {
      teacher: true,
    },
  });

  console.log(result);
}

main();
```

Como resultado será retornado todos os atributos das tabelas Courses e Teacher:

```
● brunobandeirafernandes@MacBook-Pro-de-Bruno ORM_Prisma_Node % npx ts-node-dev src/Courses/findRelation.ts
[INFO] 19:27:23 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.1, typescript ver. 5.1.3)
[
  {
    id: '483c3366-9240-4c6a-b6a0-6b568ccb5d1a',
    name: 'Curso de React Native',
    description: 'Curso com 300h de React Native',
    duration: 300,
    created_at: 2023-10-29T21:39:49.831Z,
    fk_id_teacher: '5c80c233-2ca1-4801-9780-56b0142d2ab3',
    teacher: {
      id: '5c80c233-2ca1-4801-9780-56b0142d2ab3',
      name: 'Bruno Bandeira'
    }
  },
  {
    id: '4f87e1ed-0b21-44e5-ba5a-52925bb9dbd3',
    name: 'Curso de Node',
    description: 'Curso de node.js no Senai',
    duration: 200,
    created_at: 2023-10-28T15:35:49.563Z,
    fk_id_teacher: '59636eec-e1b7-4869-a4fb-5e154e3464ef',
    teacher: {
      id: '59636eec-e1b7-4869-a4fb-5e154e3464ef',
      name: 'Bruno Fernandes'
    }
  }
]
```

**Link do Github:**

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/1d8c7ec3310f95e529e7c46c4e0648056cb19001](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/1d8c7ec3310f95e529e7c46c4e0648056cb19001)

## 4.RELACIONAMENTO ONE-TO-MANY (1:N)

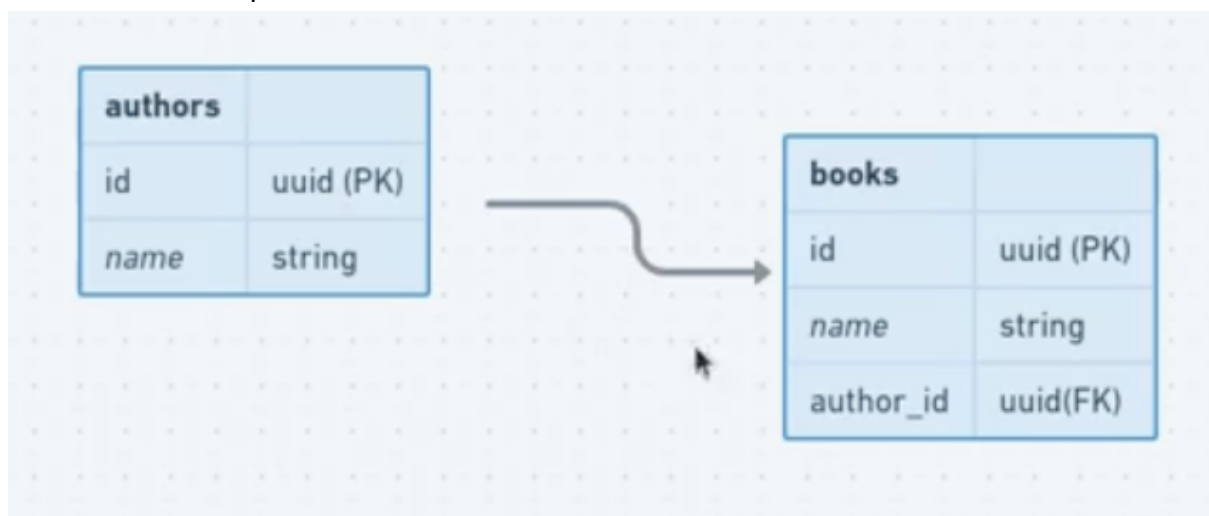
Agora iremos trabalhar com relacionamento um para muitos. O relacionamento "um para muitos" em bancos de dados relacionais é um tipo de relação entre duas entidades, onde uma entidade está relacionada a várias instâncias da outra entidade.

Por exemplo, considere as entidades "Departamento" e "Funcionário". Um departamento pode ter vários funcionários, mas um funcionário só pode pertencer a um único departamento. Nesse caso, você poderia ter uma tabela "Departamento" com uma chave primária única para cada departamento e uma tabela "Funcionário" que inclui uma coluna de chave estrangeira que referencia a chave primária do departamento a que o funcionário pertence.

Para criar esse relacionamento, você pode usar uma chave estrangeira na tabela "Funcionário" que faz referência à chave primária na tabela "Departamento". Isso garante que cada funcionário só possa estar associado a um único departamento, enquanto um departamento pode ter vários funcionários.

### 4.1 RELACIONAMENTO TABELA AUTHORS E BOOKS (1:N)

Utilizaremos o exemplo de Autores e Livros. Cada Livro será escrito por apenas um único autor e cada autor poderá escrever vários livros.



Para criar as tabelas Autor e Livros, seguindo o relacionamento 1:N, utilizaremos o código abaixo:

```
model Authors {  
    id String @id @default(uuid())  
    name String @unique  
  
    books Books[]  
}
```

```

    @@map("authors")
  }

  model Books {
    id String @id @default(uuid())
    name String @unique

    author_id String
    author Authors @relation(fields: [author_id], references: [id])

    @@map("books")
  }

```

Um Autor poderá ter muitos Livros, por isso colocamos o atributo

```
books Books[]
```

Um Livro poderá ser de muitos Autores, por isso iremos criar o atributo de chave estrangeira `author_id String` e iremos criar a referência a tabela Autor, conforme segue:

```
author Authors @relation(fields: [author_id], references: [id])
```

Utilize o comando abaixo para gerar as migrations:

**`npx prisma migrate dev`**

```

● brunobandeirafernandes@MacBook-Pro-de-Bruno ORM_Prisma_Node % npx prisma migrate dev
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma
Datasource "db": MySQL database "ormprisma" at "localhost:3306"

✓ Enter a name for the new migration: ... create_author_books
Applying migration `20231029230820_create_author_books`

The following migration(s) have been created and applied from new schema changes:

migrations/
├─ 20231029230820_create_author_books/
└─ migration.sql

Your database is now in sync with your schema.

✓ Generated Prisma Client (v5.4.2) to ./node_modules/@prisma/client in 60ms

Update available 5.4.2 -> 5.5.2
Run the following to update
  npm i --save-dev prisma@latest
  npm i @prisma/client@latest

```

Link do Github:

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/1785100d259945e63edf44da8b9a24f6c6173e33](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/1785100d259945e63edf44da8b9a24f6c6173e33)

## 4.2 CRIAR VÁRIOS REGISTROS - CREATE MANY

Para realizar o registros utilizando o relacionamento 1:N, utilizaremos os 4 modelos apresentados anteriormente, mais o formato create many, responsável por cadastrar simultaneamente muitos registros de uma única vez.

Crie a pasta **Author**. Em seguida, crie o arquivo chamado **create\_many.ts**.

```
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

async function main() {
  const result = await prisma.authors.create({
    data: {
      name: "Machado de Assis",
      Books: {
        createMany: {
          data: [
            { name: "Dom Casmurro" },
            { name: "Memórias Póstumas de Brás Cubas" },
          ],
        },
      },
    },
  });

  console.log(result);
}

main();
```

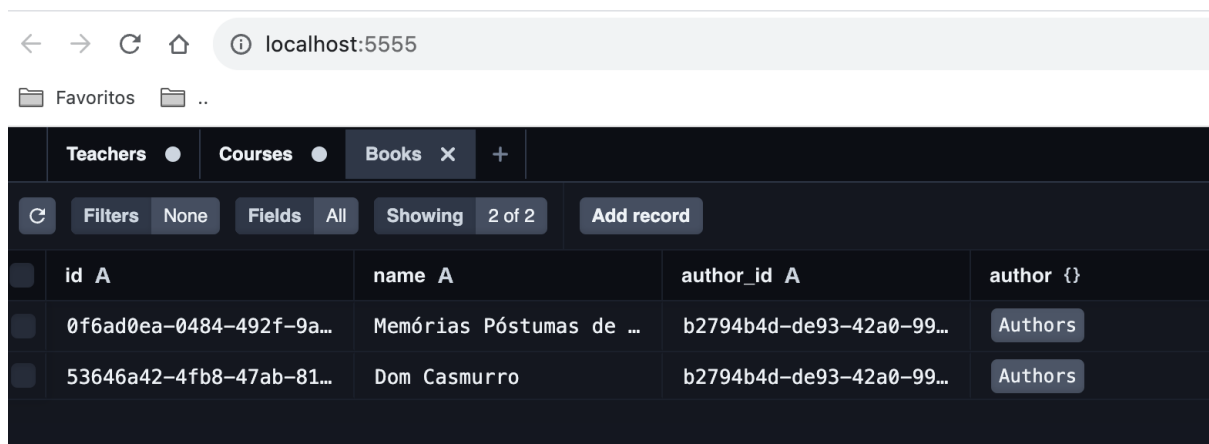
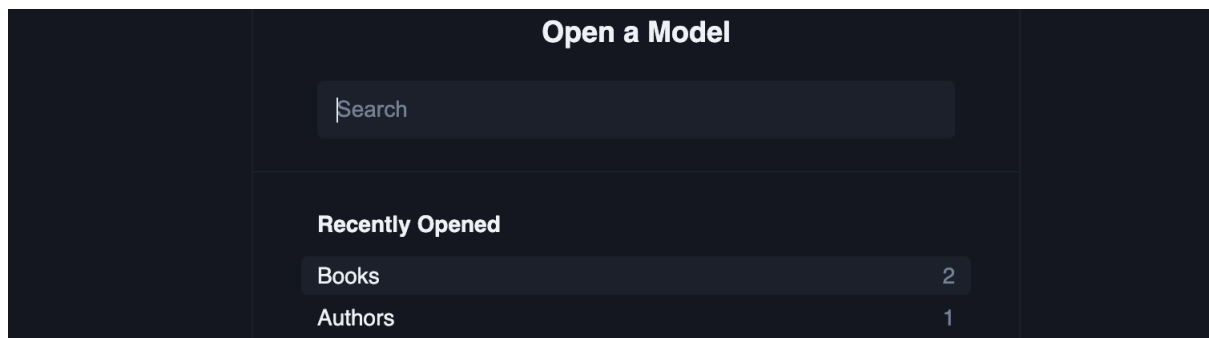
No código acima, iremos criar um autor chamado Machado de Assis e dois livros vinculados a este autor.



```

● brunobandeirafernandes@MacBook-Pro-de-Bruno ORM_Prisma_Node % npx ts-node-dev src/Author/create_many.ts
[INFO] 20:23:04 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.1, typescript ver. 5.1.3)
{
  id: 'b2794b4d-de93-42a0-998f-babb037c679c',
  name: 'Machado de Assis'
}

```



### Link do Github:

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/5922f4b2321d14bd08ceedd65ffc7124c2491546](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/5922f4b2321d14bd08ceedd65ffc7124c2491546)

## 5.RELACIONAMENTO MANY-TO-MANY (N:N)

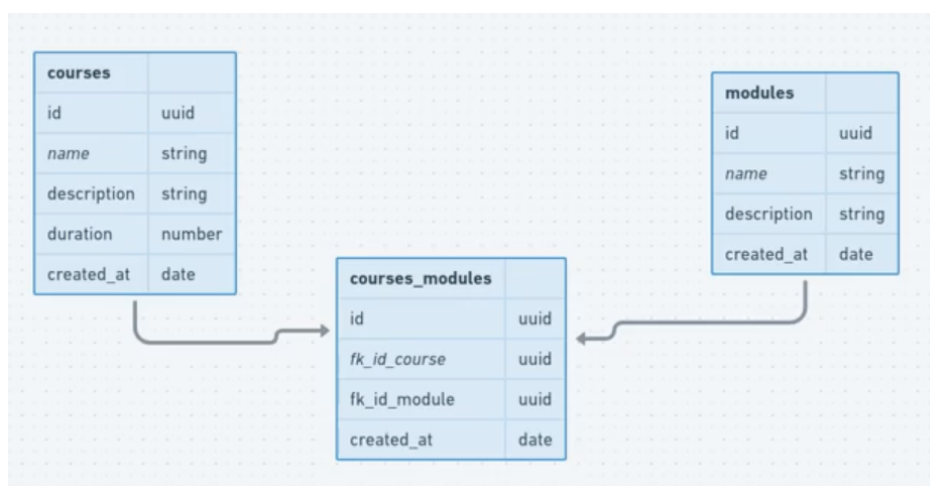
O relacionamento "muitos para muitos" em bancos de dados relacionais é um tipo de relação em que uma entidade pode estar associada a várias instâncias de outra entidade e vice-versa. Esse tipo de relacionamento é implementado usando tabelas de associação, também conhecidas como tabelas de junção ou tabelas de ligação.

Por exemplo, considere as entidades "Estudante" e "Disciplina". Um estudante pode se inscrever em várias disciplinas e uma disciplina pode ter vários estudantes matriculados nela. Neste caso, seria necessário criar uma terceira tabela para representar a relação entre estudantes e disciplinas, e essa tabela de associação conteria as chaves estrangeiras das tabelas de estudantes e disciplinas.

A tabela gerada pode se chamar "Estudante\_Disciplina". Essa tabela é usada como uma tabela de associação para representar o relacionamento "muitos para muitos" entre estudantes e disciplinas. Ela possui chaves estrangeiras que referenciam as chaves primárias das tabelas "Estudante" e "Disciplina". Cada linha nessa tabela representa uma associação entre um estudante e uma disciplina, permitindo que um estudante esteja matriculado em várias disciplinas e uma disciplina tenha vários estudantes matriculados nela.

### 5.1 RELACIONAMENTO COURSES\_MODULES (N:N)

Para esse exemplo, voltaremos ao exemplo utilizado na seção 2 (tabela Curso e Módulos), porém iremos realizar ajustes. Considere recomençar a implementação, conforme código que será apresentado.



Porém agora, iremos implementar o relacionamento N:N, criando o tabela CoursesModules.

Crie a tabela Courses. Observe o array modules do tipo CoursesModules:

```
model Courses {
  id String @id @default(uuid())
  name String @unique
  description String?
  duration Int
  created_at DateTime @default(now())

  modules CoursesModules[]

  @@map("courses")
}
```

Crie a tabela Modules. Observe o array courses do tipo CoursesModules:

```
model Modules {
  id String @id @default(uuid())
  name String @unique
  description String
  created_at DateTime @default(now())

  courses CoursesModules[]

  @@map("modules")
}
```

Por fim, crie a tabela CoursesModules:

```
model CoursesModules {
  id String @id @default(uuid())

  // chave estrangeira de Courses
  course Courses @relation(fields: [fk_id_course], references: [id])
  fk_id_course String

  // chave estrangeira de modules
  module Modules @relation(fields: [fk_id_module], references: [id])
  fk_id_module String

  created_at DateTime @default(now())

  @@map("courses_modules")
}
```

```
}
```

Observe que foi criado o atributo `course` do tipo `Courses` e criada a sua relação com a tabela `Courses`. Ou seja, o atributo `course` será a chave estrangeira da tabela `Courses`.

Utilize o comando abaixo para gerar as migrations:

**npx prisma migrate dev**

```
brunobandeirafernandes@MacBook-Pro-de-Bruno ORM_Prisma_Node % npx prisma migrate dev
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma
Datasource "db": MySQL database "ormprisma" at "localhost:3306"

⚠ Warnings for the current datasource:

  • You are about to drop the column `fk_id_teacher` on the `courses` table, which still contains 6 non-null values.
  • You are about to drop the `teachers` table, which is not empty (6 rows).

✓ Are you sure you want to create and apply this migration? ... yes
✓ Enter a name for the new migration: ... create_courses_modules
Applying migration `20231030025838_create_courses_modules`

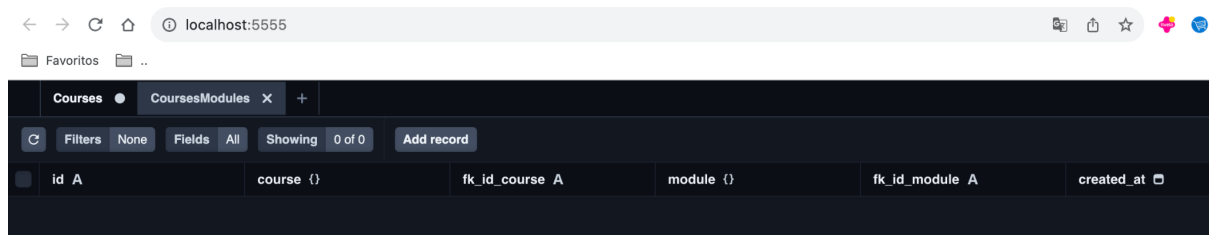
The following migration(s) have been created and applied from new schema changes:

migrations/
├─ 20231030025838_create_courses_modules/
└─ migration.sql

Your database is now in sync with your schema.

✓ Generated Prisma Client (v5.4.2) to ./node_modules/@prisma/client in 65ms
```

As três tabelas e suas respectivas relações foram criadas com sucesso. Abaixo a tabela `CoursesModules`.



id	course	fk_id_course	module	fk_id_module	created_at
----	--------	--------------	--------	--------------	------------

Link do Github:

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/2d374cc083734066357b231fcee51538a113ced1](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/2d374cc083734066357b231fcee51538a113ced1)

## 5.2 CRIANDO REGISTROS EM RELACIONAMENTO MUITO PARA MUITOS (N:N)

Para fazer a inserção de registros vinculados a chave estrangeira, podemos utilizar as mesmas quatro abordagens apresentadas na seção one-to-one:

- **ConnectOrCreate:** se a chave estrangeira não existe, será criada.
- **Connect:** conecta uma chave estrangeira existente.
- **Create:** cria uma nova chave estrangeira
- **Vinculando a fk:** vincula a uma chave estrangeira existente.

Para a tabela Modules, iremos demonstrar considerando que não existe um Módulo cadastrado, porém existe um curso cadastrado (utilizaremos o **connect**). Crie o arquivo **create\_exist\_relation.ts**

```
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

async function main() {
  const result = await prisma.modules.create({
    data: {
      description: "Aprendendo firebase do zero",
      name: "Aprendendo Firebase",
      courses: {
        // Criar um novo relacionamento
        create: {
          // Abrir o relacionamento
          course: {
            // Conectar ao relacionamento existente
            connect: {
              id: "e27d03d2-f5ca-4f0c-908e-1d1bdda7a83e",
            },
          },
        },
      },
    },
  });

  console.log(result);
}

main();
```

Agora considerando que existe registro nas tabelas Courses e Modules (utilizaremos diretamente os ids das chaves primárias). Crie o arquivo **createManyToMany.ts**

```
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

async function main() {
```

```

const result = await prisma.coursesModules.create({
  data: {
    fk_id_course: "e27d03d2-f5ca-4f0c-908e-1d1bdda7a83e",
    fk_id_module: "81d0aea2-a1b2-48bd-b377-f8085fe7e2b9",
  },
});

console.log(result);
}

main();

```

Por fim, considerando que não existe registro em nenhum das tabelas (Courses e Modules). Crie o arquivo **create\_non\_exist\_both.ts**

```

import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

async function main() {
  const result = await prisma.coursesModules.create({
    data: {

      course: {
        create: {
          duration: 200,
          name: "Curso de ABC",
          description: "Curso completo de ABC",
        },
      },

      module: {
        create: {
          description: "Curso completo de ABC",
          name: "Módulo A",
        },
      },

    },
  });
}

```

```
console.log(result);
}

main();
```

Abaixo o registro de sucesso na execução:

```
brunobandeirafernandes@MacBook-Pro-de-Bruno ORM_Prisma_Node % npx ts-node-dev src/Modules/create_non_exist_both.ts
[INFO] 00:33:48 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.1, typescript ver. 5.1.3)
{
  id: 'afee881f-f585-4552-b6db-f5b088674891',
  fk_id_course: '49c72d72-2d8e-481c-aae8-cf0e19ce3c15',
  fk_id_module: 'ddc0bf60-15d4-4c74-9ee4-0be26884e94c',
  created_at: 2023-10-30T03:33:50.172Z
}
```

Observe que no Prisma Studio que foram realizados os registros nas três tabelas.

The top screenshot shows the 'CoursesModules' table with one record:

id	course {}	fk_id_course A	module {}	fk_id_module A	created_at
afee881f-f585-4552-b6...	Courses	49c72d72-2d8e-481c-aa...	Modules	ddc0bf60-15d4-4c74-9e...	2023-10-30T03:33:50.1...

The bottom screenshot shows two tables: 'Courses' and 'Modules'.

id	name	description	duration	created_at
49c72d72-2d8e-481c-aa...	Curso de ABC	Curso completo de ABC	200	2023-10-30T03:33:50.1...

id	name	description	created_at
ddc0bf60-15d4-4c74-9e...	Módulo A	Curso completo de ABC	2023-10-30T03:33:50.1...

Link do Github:

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/dff51f460fbdc9d3f5d95a3582f8e8e60125b415](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/dff51f460fbdc9d3f5d95a3582f8e8e60125b415)

## 5.3 BUSCANDO DADOS EM RELACIONAMENTOS N:N

Para demonstrar a busca na tabela Courses, crie a pasta search e o arquivo findBycourse.ts. Faremos uma busca de todos os atributos de um curso, baseado em seu id e apresentando todos os atributos vinculados ao módulo.

```
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

async function main() {
  const result = await prisma.courses.findMany({
    where: {
      id: "593037f0-e666-4716-9e5e-4d859c1cdc3c"
    },

    include: {
      modules: true,
    },
  });

  //console.log(result);
  console.log(JSON.stringify(result));
}

main();
```

Por fim, para buscar pela tabela CoursesModules todas os atributos das tabelas Courses e Modules, podemos utilizar da forma que segue (arquivo chamado findByRelation.ts):

```
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

async function main() {
  const result = await prisma.coursesModules.findMany({
    include: {
      course: true,
      module: true,
    },
  },
```



```

    });

    console.log(JSON.stringify(result));
  }

  main();

```

Os dados retornados estarão em formato json.

```

● brunobandeirafernandes@MacBook-Pro-de-Bruno ORM_Prisma_Node % npx ts-node-dev src/search/findByRelation.ts
[INFO] 00:58:06 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.1, typescript ver. 5.1.3)
[{"id":"afee881f-f585-4552-b6db-f5b088674891","fk_id_course":"49c72d72-2d8e-481c-aae8-cf0e19ce3c15","fk_id_module":"ddc0bf60-15d4-4c74-9ee4-0be26884e94c","created_at":"2023-10-30T03:33:50.172Z","course":{"id":"49c72d72-2d8e-481c-aae8-cf0e19ce3c15","name":"Curso de ABC","description":"Curso completo de ABC","duration":200,"created_at":"2023-10-30T03:33:50.172Z"},"module":{"id":"ddc0bf60-15d4-4c74-9ee4-0be26884e94c","name":"Módulo A","description":"Curso completo de ABC","created_at":"2023-10-30T03:33:50.172Z"}}]

```

Link do Github:

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/69225c92236d463a1c52e9e3f485d516c36050d6](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/69225c92236d463a1c52e9e3f485d516c36050d6)

## 5.4 REMOVENDO DADOS EM RELACIONAMENTOS N:N

Para remover um item na tabela CoursesModules, utilizaremos o método remove do Prisma, semelhante a seção 2.6.

```

import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

async function main() {
  const result = await prisma.coursesModules.delete({
    where: {
      id: "afee881f-f585-4552-b6db-f5b088674891",
    },
  });

  console.log(result);
}

main();

```

```
● brunobandeirafernandes@MacBook-Pro-de-Bruno ORM_Prisma_Node % npx ts-node-dev src/modules/delete_coursesModules.ts
[INFO] 01:06:59 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.1, typescript ver. 5.1.3)
{
  id: 'afee881f-f585-4552-b6db-f5b088674891',
  fk_id_course: '49c72d72-2d8e-481c-aae8-cf0e19ce3c15',
  fk_id_module: 'ddc0bf60-15d4-4c74-9ee4-0be26884e94c',
  created_at: 2023-10-30T03:33:50.172Z
}
```

Link do Github:

[https://github.com/brunobandeiraf/ORM\\_Prisma\\_Node/commit/eb6822e1f9997b87e3251637fc633a95e1ac856c](https://github.com/brunobandeiraf/ORM_Prisma_Node/commit/eb6822e1f9997b87e3251637fc633a95e1ac856c)

## 6.IMPORTANDO BANCO DE DADOS

Para importar os dados

Para gerar a tabela e seus respectivos campos no bd, iremos utilizar o comando abaixo:

**npx prisma db pull**