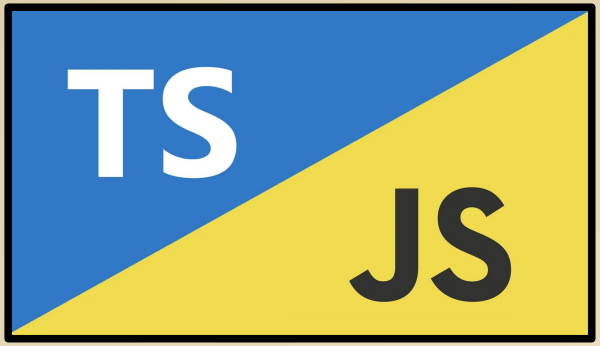
- Introdução
- Classe
- Constraints
- Herança
- Exemplo
- .create({})
- .update({})
- .delete({})



Prisma TS - Parte 1
Mapeamento

Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

Ambiente


```
//gera package.json
npm init -y
//dependências ambiente produção
npm install @prisma/client
//dependências ambiente local
npm install prisma typescript tsx @types/node --save-dev
```

```
//package.json
 "main": "main.js"
"dependencies": {
    "prisma": "^6.6.0"
  },
  "devDependencies": {
    "@types/node": "^22.14.1",
    "tsx": "^4.19.3",
    "typescript": "^5.8.3"
 }
```

Ambiente


```
//npx executa pacotes sem instalar globalmente
//gera tsconfig.json
npx tsc --init
```

```
//ambiente local
//compila e executa
npx tsx main.ts
```

```
//ambiente produção
//compila
tsc -p .

//executa
node .
```

```
{ //tsconfig.json
 "compilerOptions": {
 "target": "es2020",
```

TSX

Introdução

- ➔ Projetado para simplificar sua experiência com TypeScript.
- ➔ Suporta modos CommonJS e ESM (antes e depois de 2015)
- ➔ Oferece suporte ao tsconfig.json
- ➔ Modo de observação para facilitar ainda mais o desenvolvimento.

TS JS

Prisma

Introdução

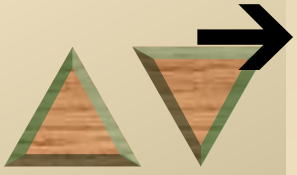
- ORM (Object-Relational Mapping) para Node.js e TypeScript.
- Facilita o trabalho com bancos de dados
- Recursos como migrações de banco de dados, consultas fortemente tipadas e um modelo de dados declarativo.
- Usa os conceitos arquiteturais do Martin Fowler: Data Mapper e Active Record

TS JS

Introdução

docker-compose.yaml

```yaml
services:
   db:
    image: postgres:15.3
    volumes:
      - ./volumes/postgres/data:/var/lib/postgresql/data
    environment:
      POSTGRES_PASSWORD: thor
      POSTGRES_USER: thor
      POSTGRES_DB: thor
    ports:
      - "5432:5432"
```



docker compose up -d

TS JS

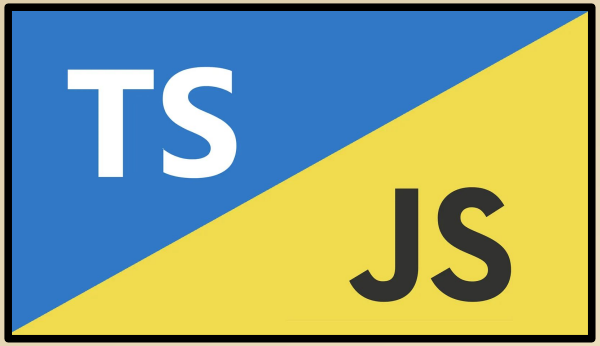
Introdução

```
//prefixar o prisma com o executor de pacotes:
npx prisma
//gera Prisma Client
npx prisma generate
//cria ./prisma/schema.prisma para modelo
//cria .env para variáveis ambiente DATABASE_URL
npx prisma init --datasource-provider postgresql
//string conexão em .env
DATABASE_URL="postgresql://thor:thor@localhost:5432
/thor?schema=public"
//editar ./prisma/schema.prisma com modelo
//migrar banco de dados para ambiente dev
//gera arquivo de migração e executa SQL no banco
npx prisma migrate dev --name init
//Visualizar dados
npx prisma studio
```

.gitignore

```
generated
nodes_modules
volumes
```


- Introdução
- Classe
- Constraints
- Herança
- Exemplo
- .create({})
- .update({})
- .delete({})



Prisma TS - Parte 1
Mapeamento

Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

TS JS

Classe

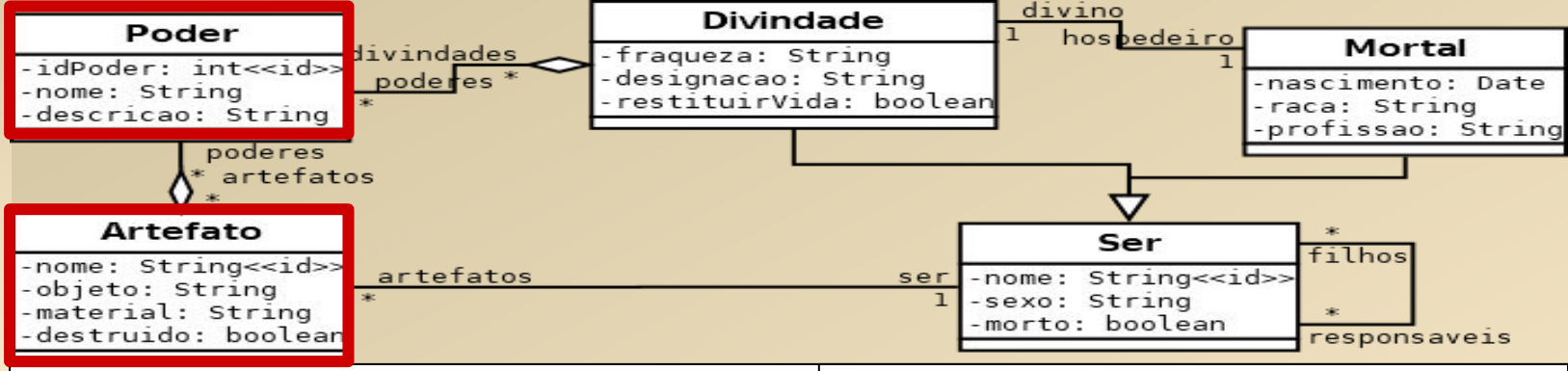
- Classes são mapeadas com o uso da palavra model
- Atributos = Nome + tipo + anotação + modificador

Classe

Tipos	Anotação	Modificador
Int	@id	[]
String	@@id([campo1, campo2])	?
Boolean	@unique	
DateTime	@default(autoincrement())	
	@default(now())	
	@map	

TS JS

Classe




```
model Poder {
 idPoder Int @id
 nome String
 descricao String
}
```

```
model Artefato{
 nome String @id
 objeto String
 material String
 destruido Boolean
}
```

TS JS

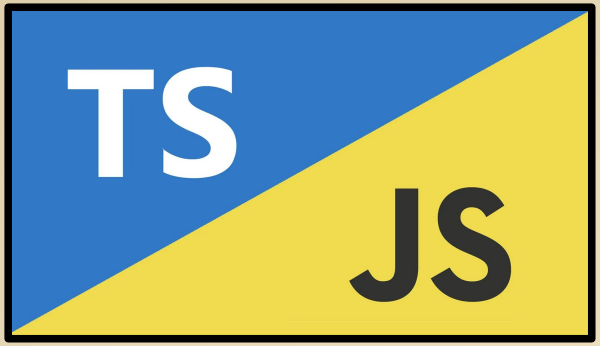
Enumerados

Classe

- São suportados nativamente pelo PostgreSQL e MySQL
- São implementados e aplicados no nível do Prisma ORM no SQLite e MongoDB
- Nome → no singular começando com maiúscula
- Valores → em maiúsculo
- **Exemplo:**

```
enum TipoCliente{
  USUARIO
  ADMINISTRADOR
}
```

- Introdução
- Classe
- Constraints
- Herança
- Exemplo
- .create({})
- .update({})
- .delete({})



Prisma TS - Parte 1
Mapeamento

Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

TS JS

Constraints

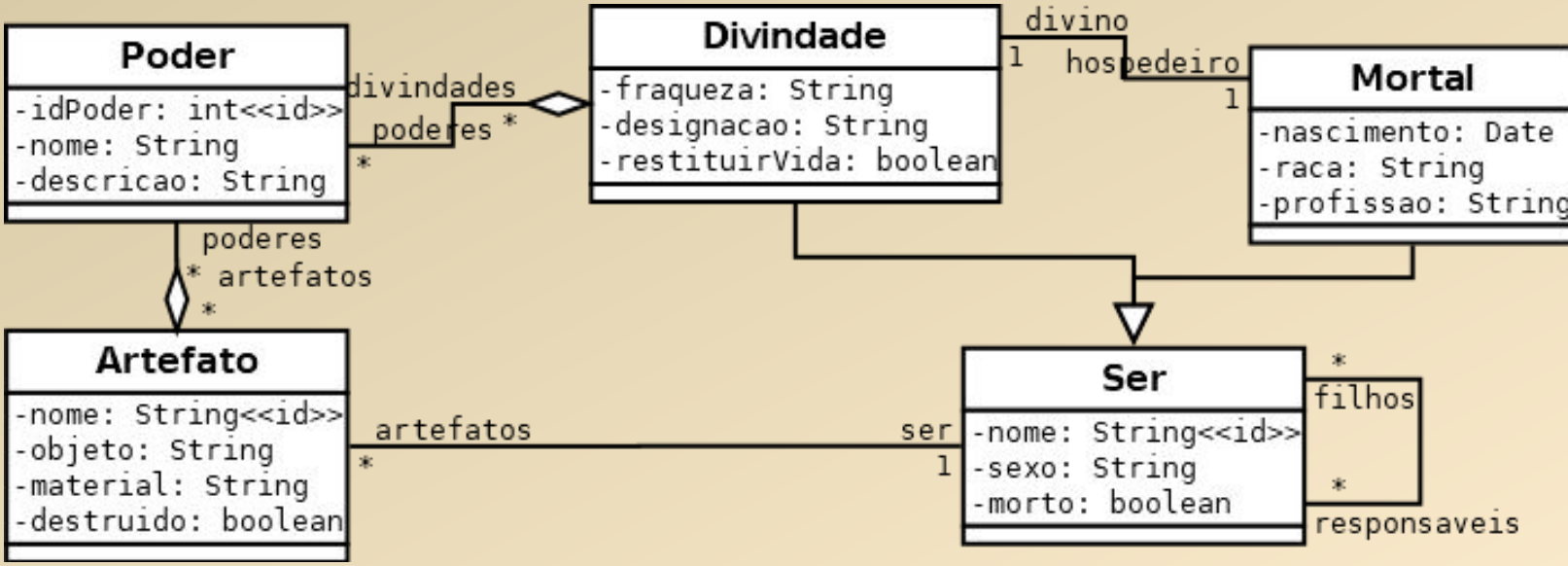
Constraints

- @default
- @unique
- @relation
  - 1:1, 1:N, N:N, unidirecional, reflexivo
  - navegação deve acontecer em ambos os sentidos
  - Relacionamento 1:N pode pode ser definido campos virtuais para obter dados de qualquer lado, mesmo que a no sentido físico da definição.

TS JS

Diagrama Exemplo

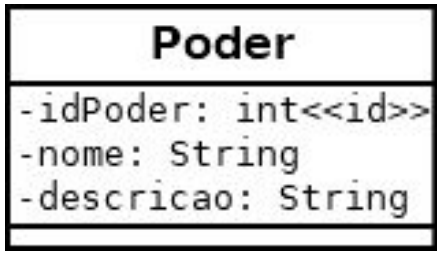
Constraints



@default

Constraints

```
model Poder {
    idPoder Int @id @default(autoincrement())
    nome String
    descricao String
}
```

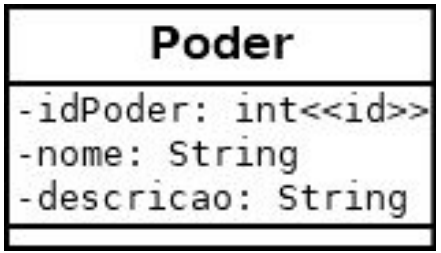


TS JS

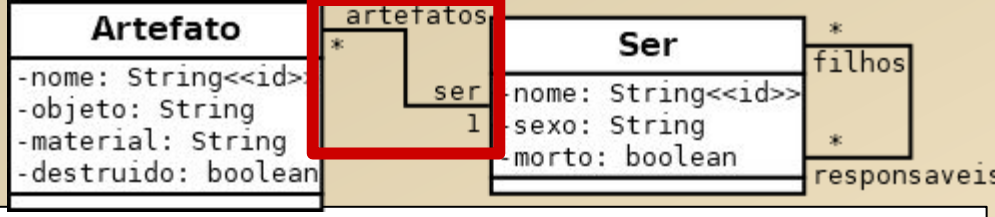
@unique

Constraints

```
model Poder {
    idPoder Int @id @default(autoincrement())
    nome String @unique
    descricao String
}
```



@relation 1:N bidirecional




```
model Artefato{
 nome String @id
 objeto String
 material String
 destruido Boolean
 nomeSer String
 ser Ser @relation(fields: [nomeSer], references: [nome])
}
model Ser {
 nome String @id
 sexo String
 morto Boolean
 artefatos Artefato[]
}
```

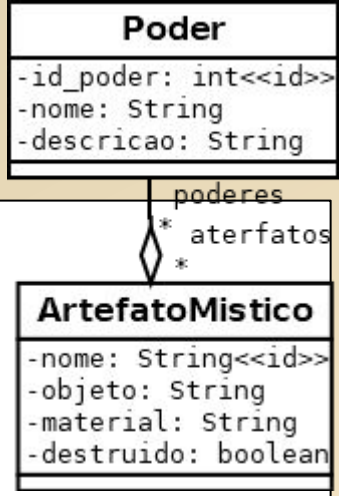
TS JS

Constraints

@relation N:N

- Implícito
  - Mínimo esforço anotações
  - Nomeações automáticas
  - Mapeamento descontrolado (sem semântica)
- Explícito
  - Maior esforço com anotações
  - Controle semântico do mapeamento

@relation N:N bidirecional implícito




```
model Poder {
  idPoder Int @id @default(autoincrement())
  nome String @unique
  descricao String
  artefatos Artefato[] @relation("ArtefatoPoder")
}
model Artefato{
  nome String @id
  objeto String
  material String
  destruido Boolean
  nomeSer String
  ser Ser @relation(fields: [nomeSer], references: [nome])
  poderes Poder[] @relation("ArtefatoPoder")
}
```

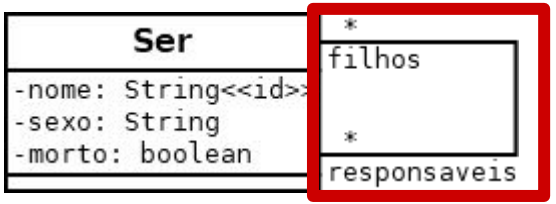
```
model Poder {
  idPoder Int @id @default(autoincrement())
  nome String @unique
  descricao String
  poderes ArtefatoPoder[]
}
model Artefato{
  nome String @id
  objeto String
  material String
  destruido Boolean
  nomeSer String
  ser Ser @relation(fields: [nomeSer], references: [nome])
  artefatos ArtefatoPoder[]
}
model ArtefatoPoder {
  artefatoNome String
  artefato Artefato @relation(fields: [artefatoNome], references: [nome])
  poderId Int
  poder Poder @relation(fields: [poderId], references: [idPoder])
  @@id([artefatoNome, poderId]) // PK composta
  @@map("ArtefatoPoder")        // nome da tabela no banco
}
```

@relation N:N explícito

TS JS

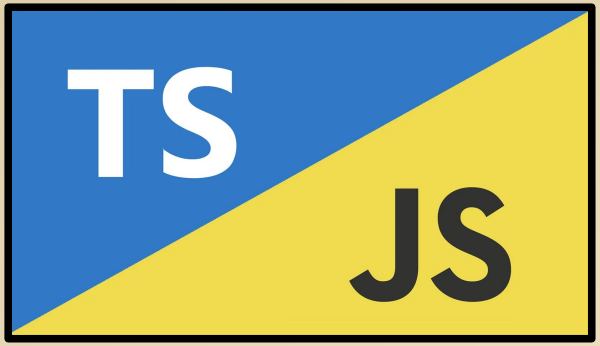
Reflexivo N:N bidirecional

Constraints




```
model Ser {
 nome String @id
 sexo String
 morto Boolean
 artefatos Artefato[]
 responsaveis SerSer[] @relation("serResponsavel")
 filhos SerSer[] @relation("serFilho")
}
model SerSer{
 responsavel Ser @relation("serResponsavel", fields:
[nomeResponsavel], references: [nome])
 nomeResponsavel String
 filho Ser @relation("serFilho", fields: [nomeFilho],
references: [nome])
 nomeFilho String
 @@id([nomeResponsavel, nomeFilho])
}
```

- Introdução
- Classe
- Constraints
- Herança
- Exemplo
- .create({})
- .update({})
- .delete({})



Prisma TS - Parte 1
Mapeamento

Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

TS JS

2 formas de mapeamento

Herança

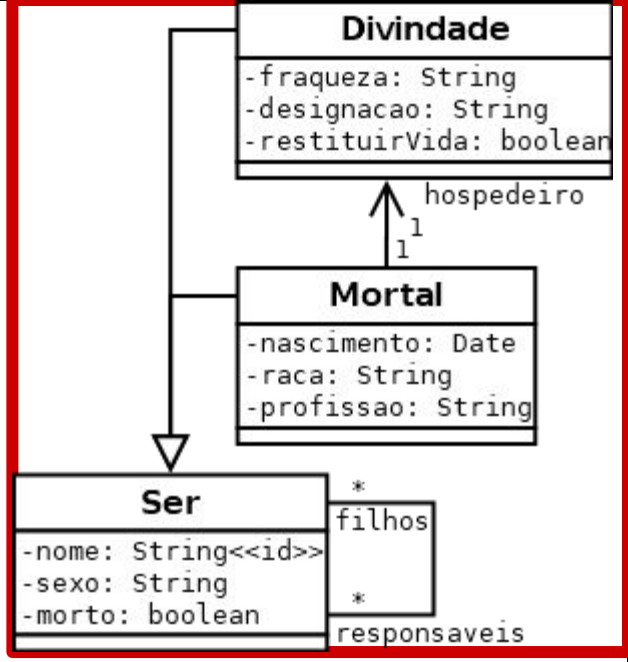
- Único model
  - Recomendado por desempenho
- Um model genérico e vários específicos
  - Recomendado modelagem complexas

TS JS

Herança

```
enum TipoSer {
  SER
  MORTAL
  DIVINDADE
}
model Ser {
 nome String @id
 sexo String
 morto Boolean
 tipo TipoSer
 fraqueza String? //Divindade
 designacao String? //Divindade
 restituirVita Boolean? //Divindade
 nascimento DateTime? //Mortal
 raca String? //Mortal
 profissao String? //Mortal
 artefatos Artefato[]
 responsaveis SerSer[] @relation("serResponsavel")
 filhos SerSer[] @relation("serFilho")
}
```

tabela única
para melhor
desempenho

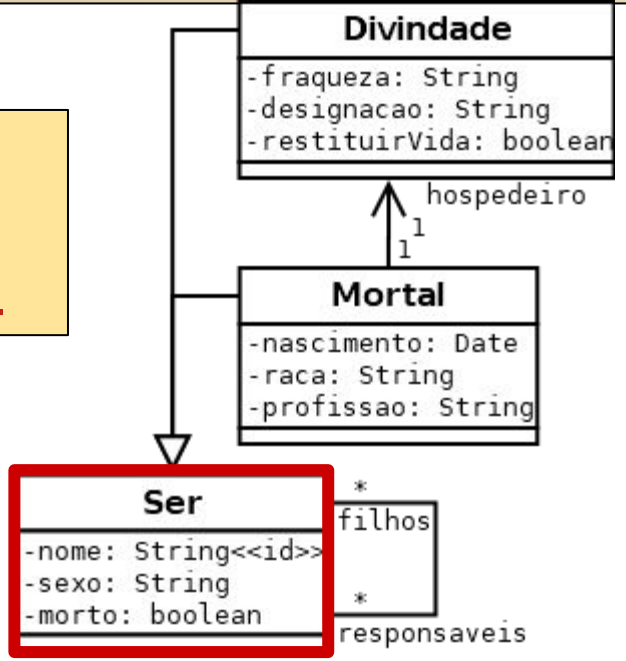


TS JS

Herança

```
enum TipoSer {
  SER
  MORTAL
  DIVINDADE
}
model Ser {
 nome String @id
 sexo String
 morto Boolean
 tipo TipoSer
 artefatos Artefato[]
 responsaveis SerSer[] @relation("serResponsavel")
 filhos SerSer[] @relation("serFilho")
 divindade Divindade? @relation("serDivindade")
 mortal Mortal? @relation("serMortal")
}
```

múltiplas tabelas (genérica)



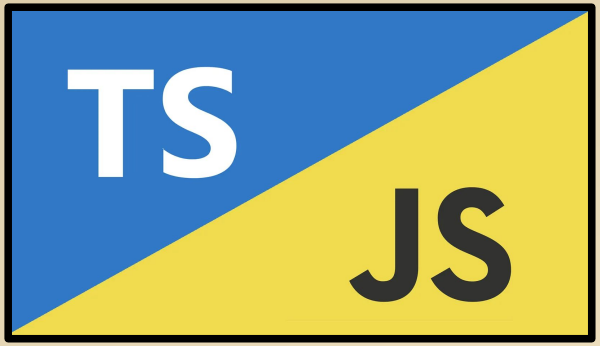
TS JS

Herança

```
model Divindade {
 nomeSer String @id
 ser Ser @relation("serDivindade", fields:
[nomeSer], references: [nome])
 fraqueza String
 designacao String
 restituirVita Boolean
}
model Mortal  {
 nomeSer String @id
 ser Ser @relation("serMortal", fields:
[nomeSer], references: [nome])
 nascimento DateTime
 raca String
 profissao String
}
```

múltiplas tabelas (específica)

- Introdução
- Classe
- Constraints
- Herança
- Exemplo
- .create({})
- .update({})
- .delete({})



Prisma TS - Parte 1
Mapeamento

Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

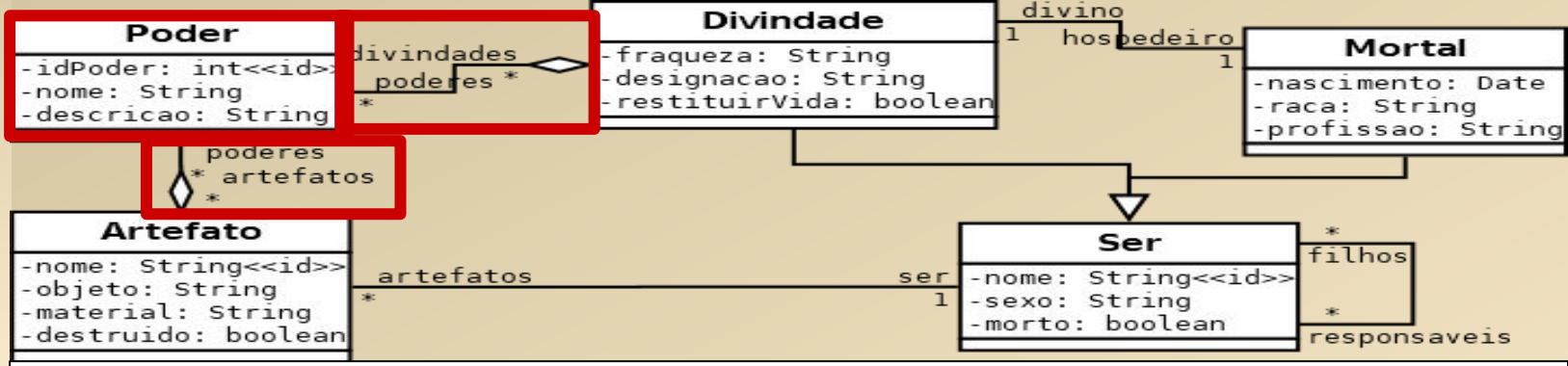
Passo a Passo mapeamento

Exemplo

- model Poder
- model PoderDivindade
- model ArtefatoPoder
- model Artefato
- enum TipoSer
- model Ser
- model SerSer
- model Divindade
- model Mortal

TS JS

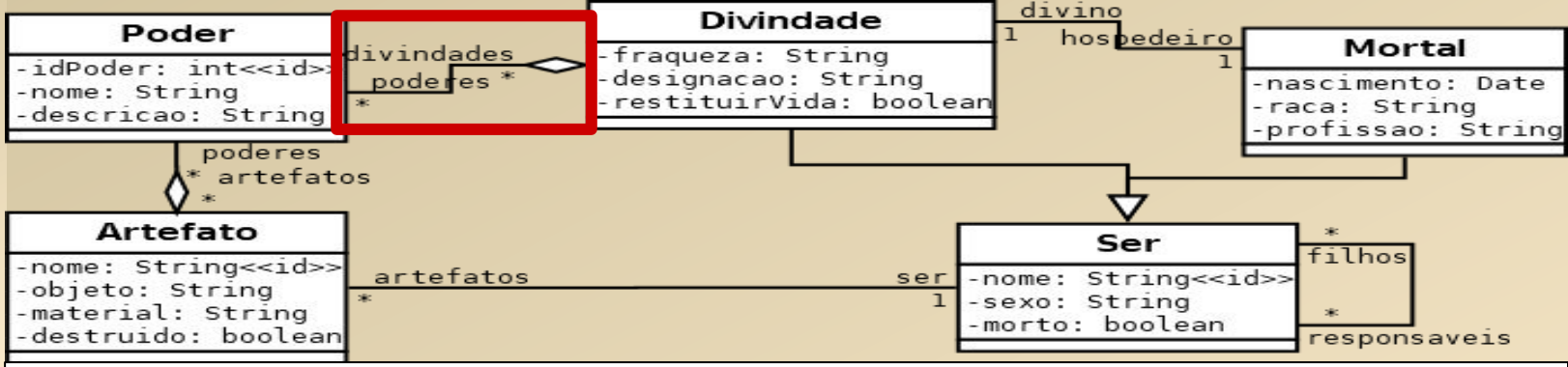
Exemplo



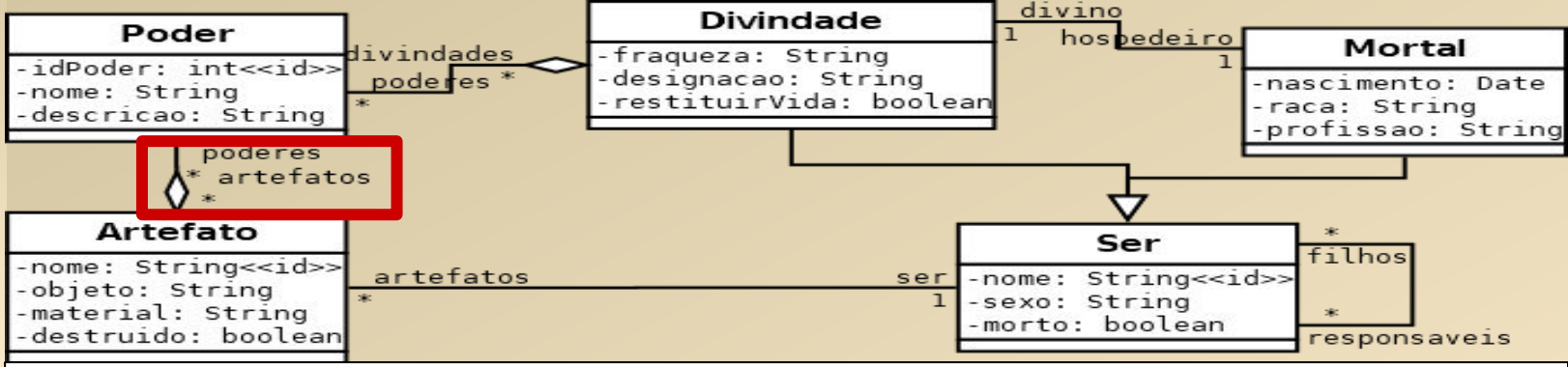

```
model Poder {
  idPoder Int @id @default(autoincrement())
  nome String @unique
  descricao String
  poderes ArtefatoPoder[]
  divindades PoderDivindade []
}
```

TS JS

Exemplo




```
model PoderDivindade {
  serNome String
  divindade Divindade @relation(fields:
[serNome], references: [nomeSer])
  poderId Int
  poder Poder @relation(fields: [poderId],
references: [idPoder])
  @@id([serNome , poderId])
}
```

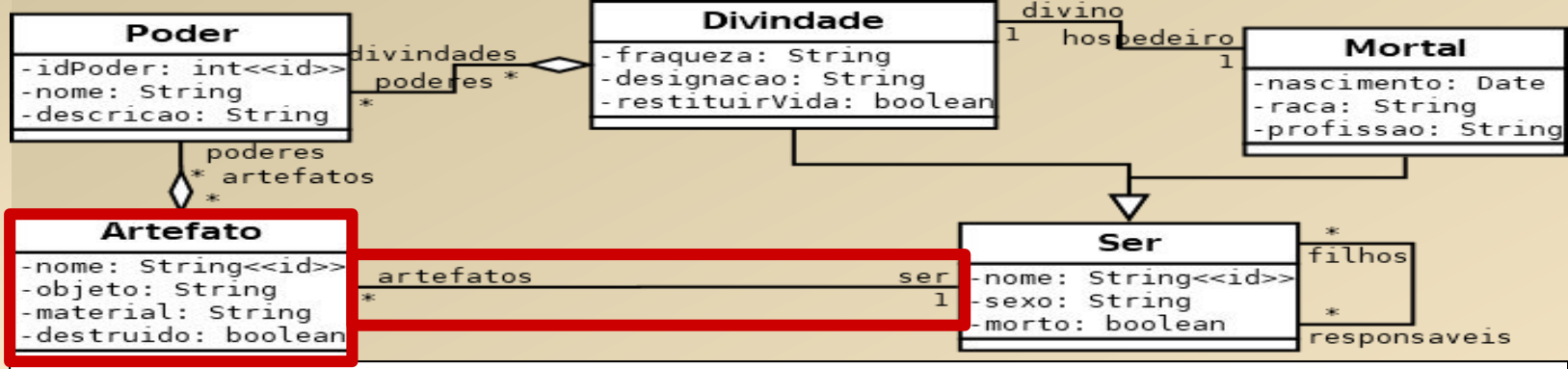


Exemplo

```
model ArtefatoPoder {
  artefatoNome String
  artefato Artefato @relation(fields:
[artefatoNome], references: [nome])
  poderId Int
  poder Poder @relation(fields: [poderId],
references: [idPoder])
  @@id([artefatoNome, poderId])
  @@map("ArtefatoPoder")
}
```

TS JS

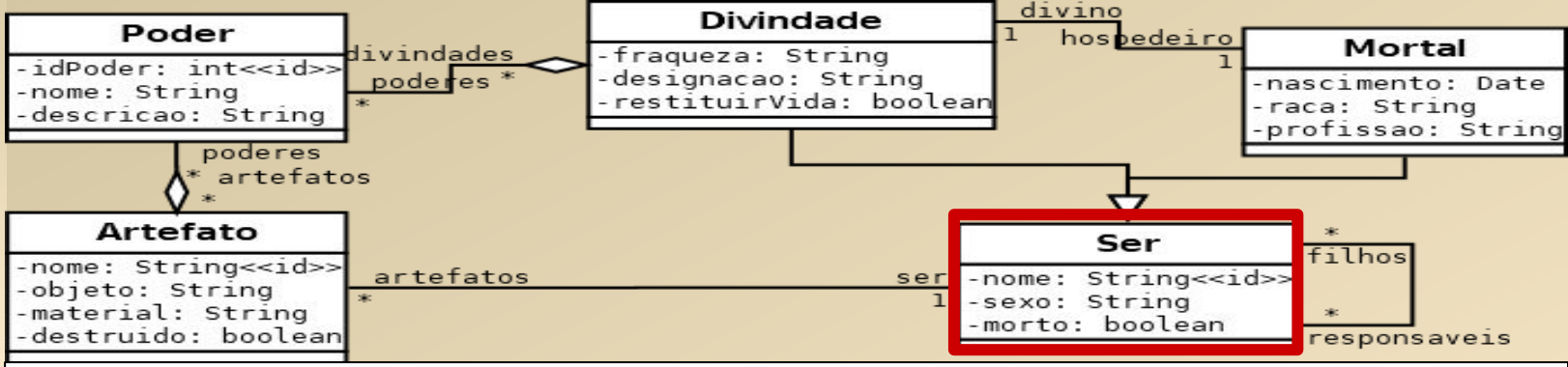
Exemplo




```
model Artefato{
   nome String @id
   objeto String
   material String
   destruido Boolean
   nomeSer String
   ser Ser @relation(fields: [nomeSer],
references: [nome])
   artefatos ArtefatoPoder[]
}
```

TS JS

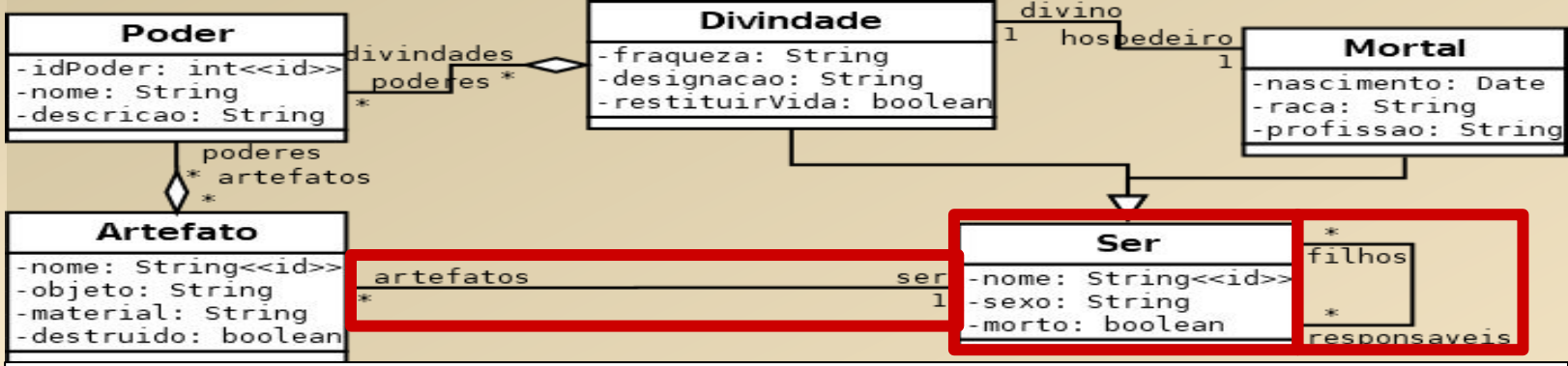
Exemplo



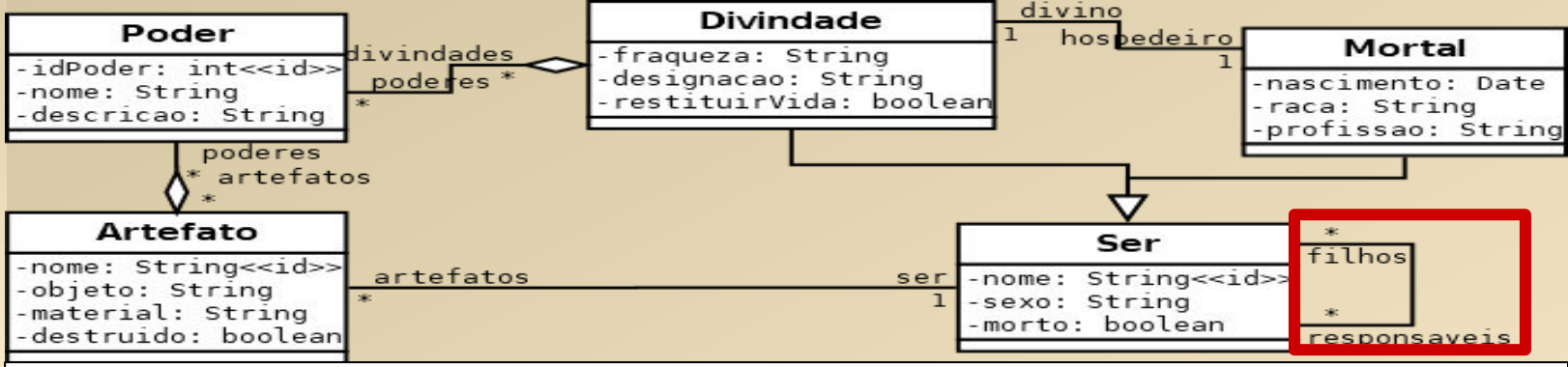

```
enum TipoSer {
   SER
   MORTAL
   DIVINDADE
}
```

TS JS

Exemplo




```
model Ser {
 nome String @id
 sexo String
 morto Boolean
 tipo TipoSer
 artefatos Artefato[]
 responsaveis SerSer[] @relation("serResponsavel")
 filhos SerSer[] @relation("serFilho")
 divindade Divindade? @relation("serDivindade")
 mortal Mortal? @relation("serMortal")
}
```

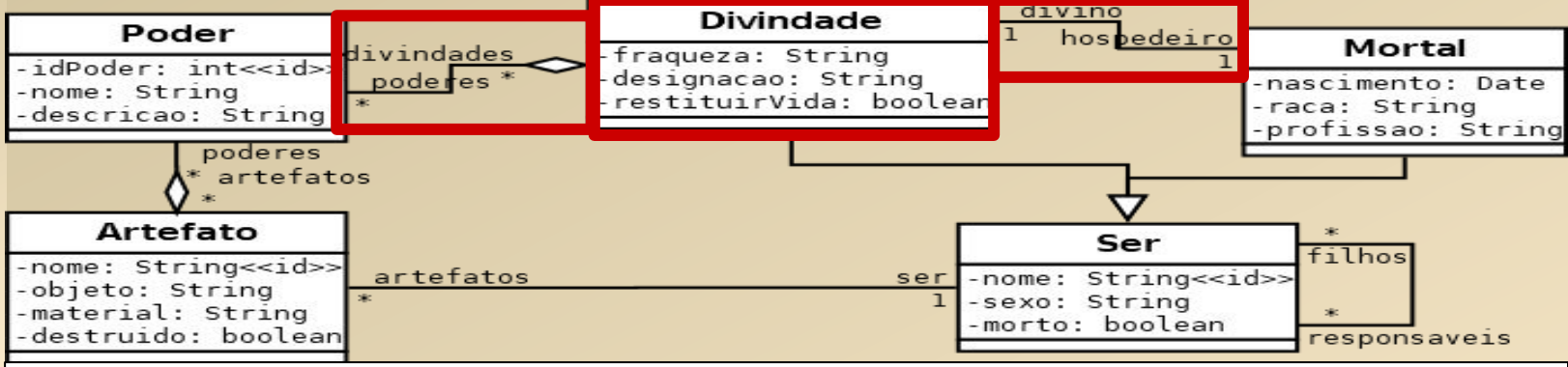


Exemplo

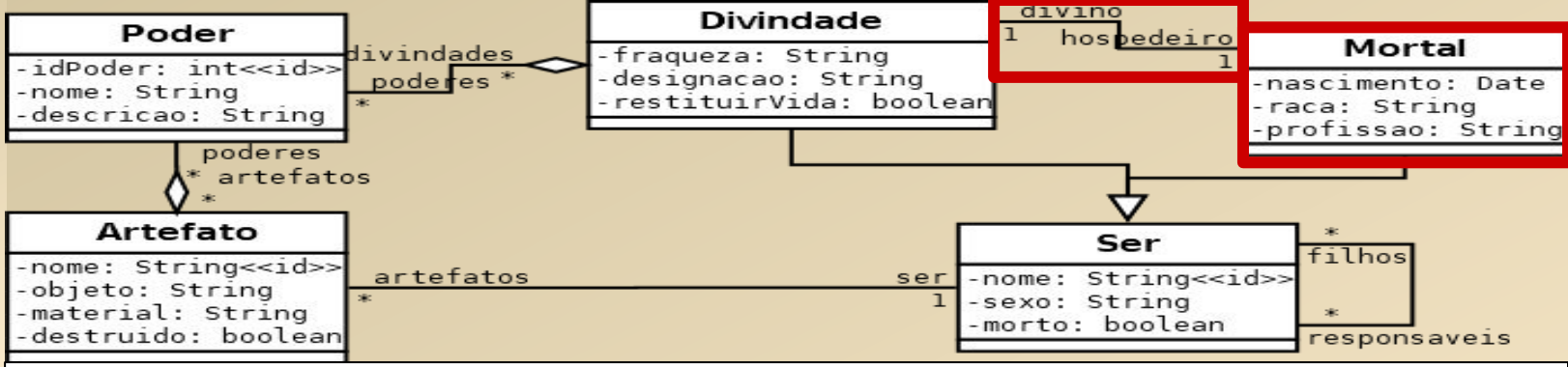
```
model SerSer{
 responsavel Ser @relation("serResponsavel",
fields: [nomeResponsavel], references: [nome])
 nomeResponsavel String
 filho Ser @relation("serFilho", fields:
[nomeFilho], references: [nome])
 nomeFilho String
 @@id([nomeResponsavel, nomeFilho])
}
```

TS JS

Exemplo



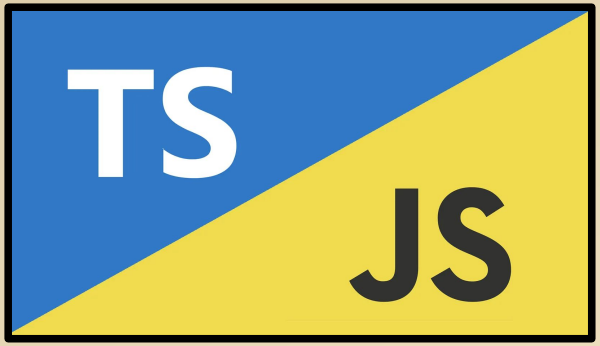

```
model Divindade {
 nomeSer String @id
 ser Ser @relation("serDivindade", fields:
[nomeSer], references: [nome])
 fraqueza String
 designacao String
 restituirVita Boolean
 poderes PoderDivindade[]
 nomeMortal String @unique
 hospedeiro Mortal? @relation(fields: [nomeMortal],
references: [nomeSer])
}
```



Exemplo

```
model Mortal  {
 nomeSer String @id
 ser Ser @relation("serMortal", fields:
[nomeSer], references: [nome])
 nascimento DateTime
 raca String
 profissao String
 divino Divindade?
}
```

- Introdução
- Classe
- Constraints
- Herança
- Exemplo
- .create({})
- .update({})
- .delete({})



Prisma TS - Parte 1
Mapeamento

Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

TS JS

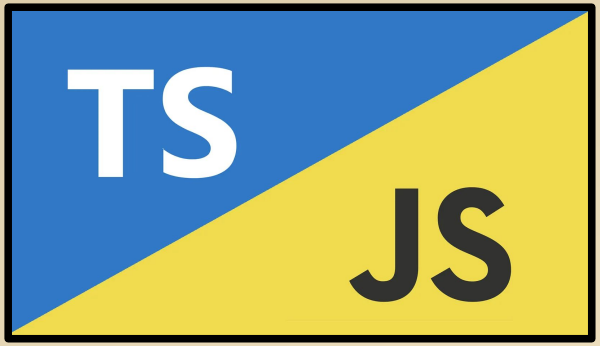
.create{}

```
import { PrismaClient} from './generated/prisma'

const prisma = new PrismaClient()

async function main() {
    const poder = await prisma.poder.create({
        data: {
          nome: 'Eletricidade instantânea',
          descricao: 'concentra raios elétricos',
        },
      })
}

main()

//comentar linha output do schema.prisma
//ou importar from './generated/prisma'
```

Poder
-idPoder: int<<id>> -nome: String -descricao: String

```
import { PrismaClient} from
'./generated/prisma'
const prisma = new PrismaClient()
async function main() {
 const ser = await prisma.ser.create({
  data: {
   nome: 'Loki',
   sexo: 'Masculino',
   morto: false,
   tipo: 'DIVINDADE',
  },
 })
 const poder = await
prisma.poder.create({
  data: {
   nome: 'Telepatia',
   descricao: 'Manipular pensamentos',
  },
 })
 const artefato = await
prisma.artefato.create({
  data: {
   nome: 'Coroa Psíquica',
   objeto: 'Coroa',
   material: 'Cristal Mental',
   destruido: false,
   nomeSer: ser.nome,
  },
 })
 const relacao = await
prisma.artefatoPoder.create({
  data: {
   artefatoNome: artefato.nome,
   poderId: poder.id_poder,
  },
 })
}
main()
```

- Introdução
- Classe
- Constraints
- Herança
- Exemplo
- .create({})
- .update({})
- .delete({})



Prisma TS - Parte 1
Mapeamento

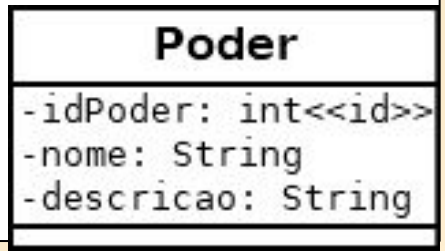
Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

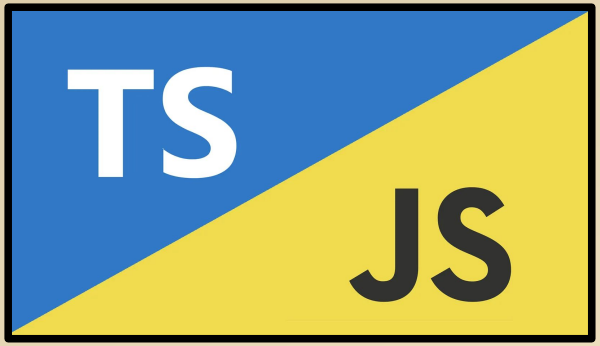
TS JS

.update({})

```
import { PrismaClient} from './generated/prisma'

const prisma = new PrismaClient()

async function main() {
    const poderAtualizado = await prisma.poder.update({
        where: { nome: 'Eletricidade instantânea' },
        data: {
          nome: 'Raio Fulminante',
          descricao: 'Descargas elétricas de alta
voltagem paralisando inimigos instantaneamente.',
        }
    })
}

main()
```



- Introdução
- Classe
- Constraints
- Herança
- Exemplo
- .create({})
- .update({})
- .delete({})



Prisma TS - Parte 1
Mapeamento

Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

TS JS

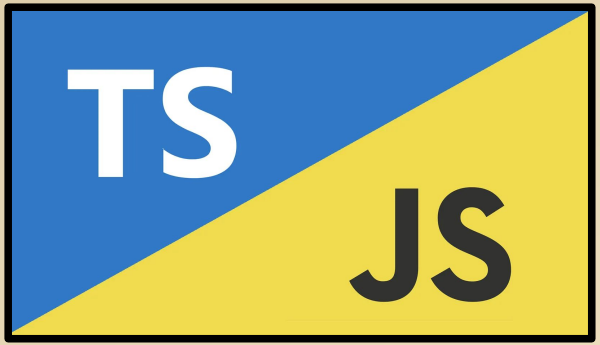
.delete({})

```
import { PrismaClient} from './generated/prisma'

const prisma = new PrismaClient()

async function main() {
    const poderRemovido = await prisma.poder.delete({
        where: {
          nome: 'Raio Fulminante',
        },
    })
}

main()
```

Poder
-idPoder: int<<id>> -nome: String -descricao: String

Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano



Prisma TS - Parte 1
Mapeamento