Banco de Dados

O que é?

Um banco de dados é uma coleção organizada de informações - ou dados - estruturadas, normalmente armazenadas eletronicamente em um sistema de computador.

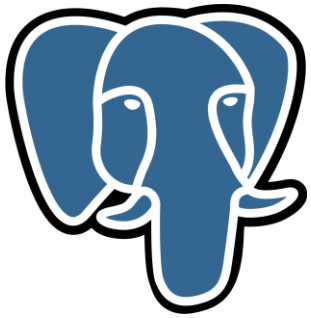
Um banco de dados é geralmente controlado por um sistema de gerenciamento de banco de dados (DBMS). O DBMS gerencia os dados recebidos, organiza-os e fornece maneiras para os dados serem modificados ou extraídos por usuários ou outros programas.

Categorias

Existem duas categorias principais de banco de dados: **Relacionais e Não-relacionais.**

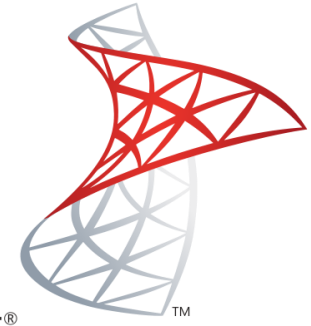
Bancos de dados relacionais (SQL): os dados são armazenados em formatos tabulares, ou seja, o dado fica na coluna, enquanto a descrição fica em linhas e atributos. Uma outra característica importante é sua linguagem, que é baseada no SQL (Structured Query Language).

Bancos de dados não relacionais (NoSQL): os dados não são tabulares e podem ser armazenados de diferentes formas com base no banco NoSQL escolhido. Dentre as principais formas de armazenamento estão o armazenamento em documentos, chave-valor, grafos e colunas.



MySQL®

ORACLE



Microsoft®
SQL Server®

MariaDB

BD Relacionais

BD não relacionais

Modelagem

Tipos de modelagem

Antes de criarmos um banco de dados precisamos fazer a sua modelagem, ou seja, mapear a forma como os dados serão salvos. Existem três etapas nesse processo:

- Modelagem Conceitual
- Modelagem Lógica
- Modelagem Física

Modelagem Conceitual

MER – Modelo Entidade Relacionamento

Modelo conceitual utilizado para descrever os objetos (**entidades**) envolvidos em um domínio de negócios, com suas características (**atributos**) e como elas se relacionam entre si (**relacionamentos**).

MER

Entidades

Uma entidade é uma **Coisa** ou **Objeto** do mundo real.

Essas entidades podem ser:

Físicas: existem no mundo real e são tangíveis.

Ex.: Cliente, Produto, etc.

Lógicas: não tangíveis.

Ex.: Venda, Disciplina, etc.

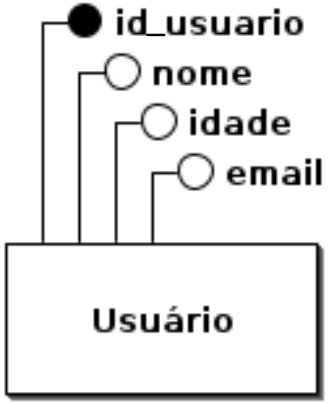
MER

Atributos

As entidades são compostas por **atributos.** Os atributos são as características que descrevem a entidade.

Toda entidade precisa ter um **atributo identificador** (id).

Por exemplo, em uma entidade **usuário** podemos ter os atributos nome, idade e e-mail:



MER

Relacionamentos

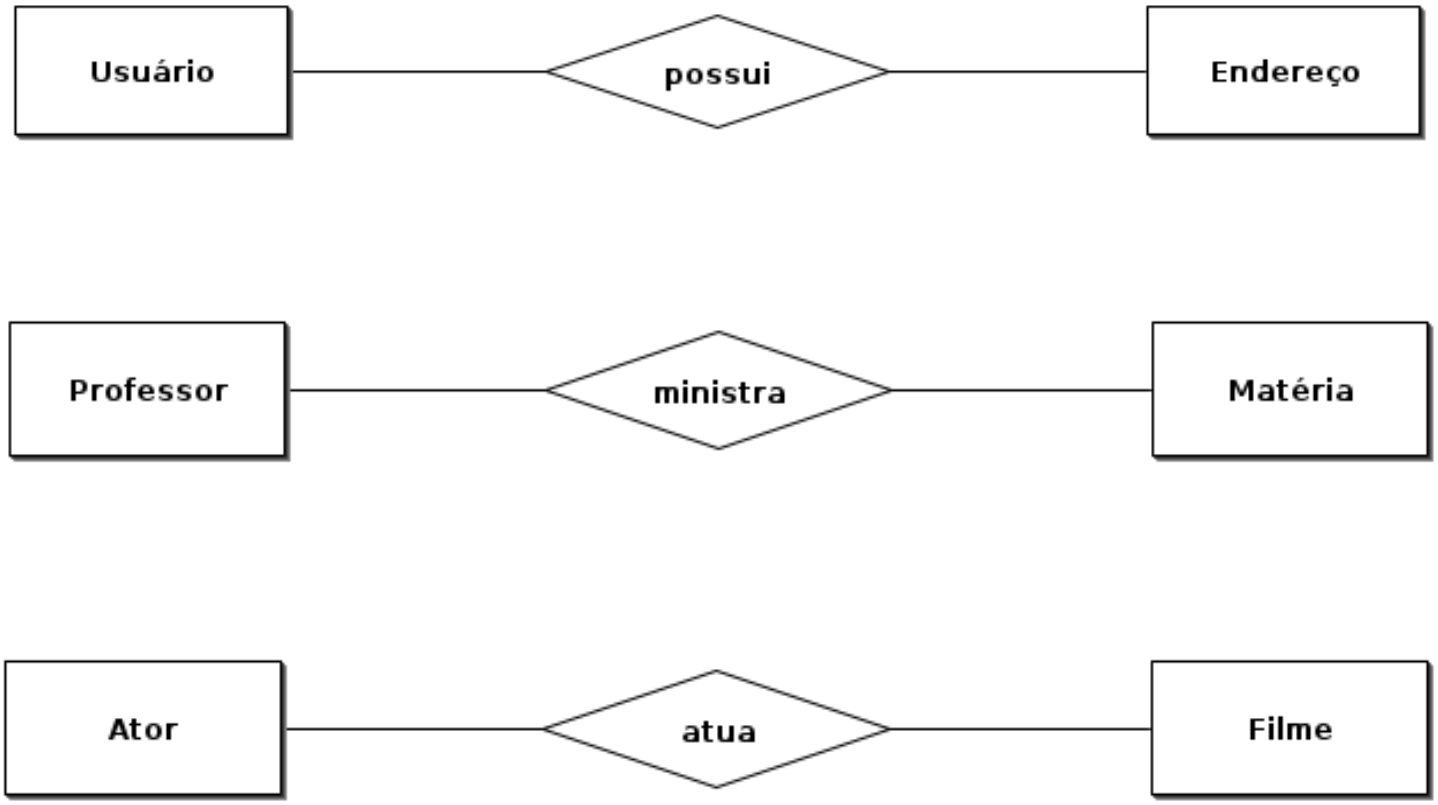
Relacionamento é a relação existente entre entidades, isto é a ligação lógica entre duas entidades que representa uma regra ou restrição de negócio, possibilitando entender como uma entidade se comporta em relação às demais.

Os relacionamentos normalmente são expressos em verbos que indicam uma ação:

- Usuário **possui** um endereço.
- Professor **ministra** uma matéria.
- Ator **atua** em um filme.

MER

Relacionamentos



MER

Mapeamento de cardinalidade

A **cardinalidade** é utilizada para expressar o grau de relação entre duas tabelas.

Cardinalidades:

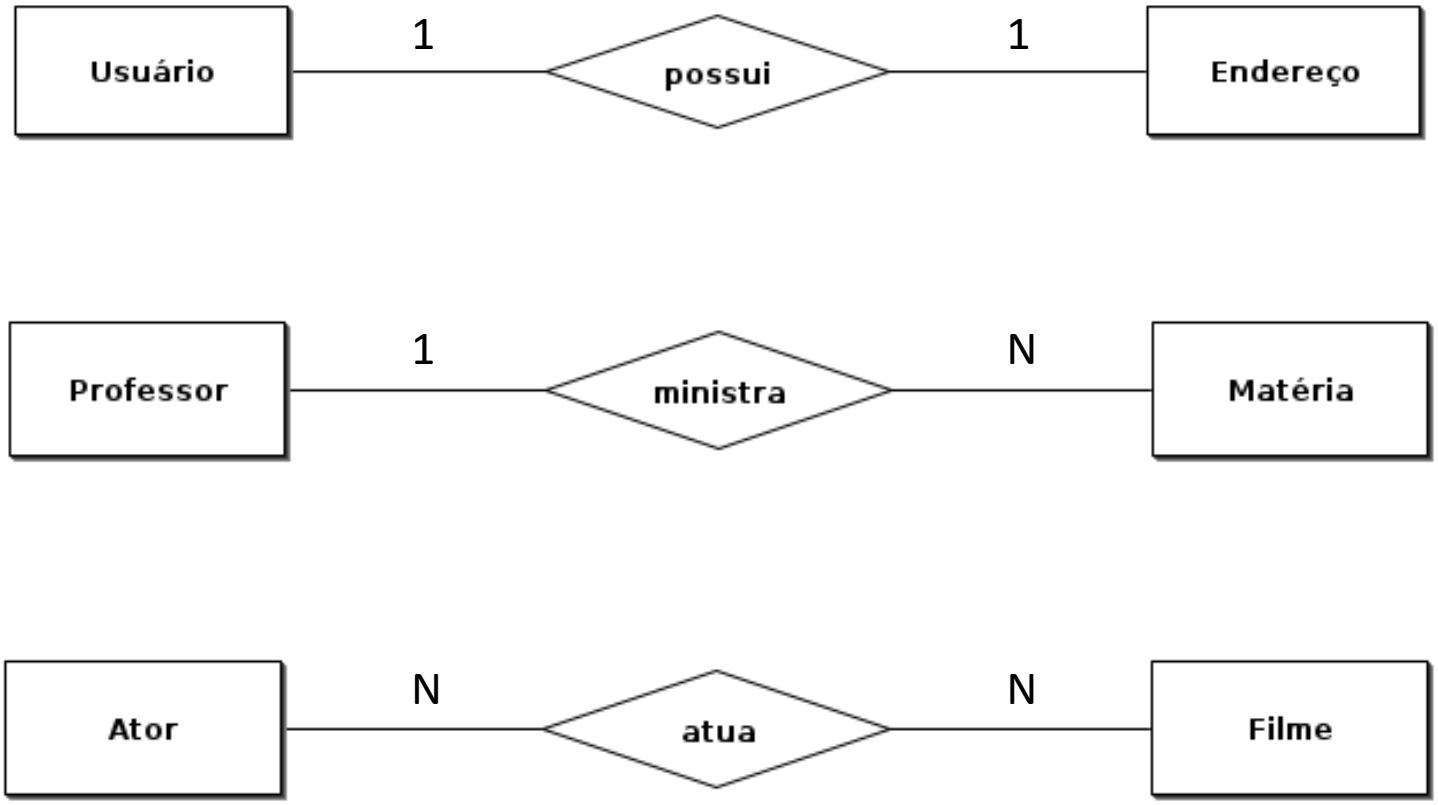
***um-para-um* (1..1):** Quando o registro de uma tabela só pode ter associação com um registro da outra tabela.

um-para-vários (1..N): Quando o registro de uma tabela pode estar associado a vários registros da outra tabela.

vários-para-vários (N..N): Quando vários registros de uma tabela podem estar associados a vários registros da outra tabela. Nesse caso é criado uma **entidade associativa**.

MER

Mapeamento de cardinalidades



MER

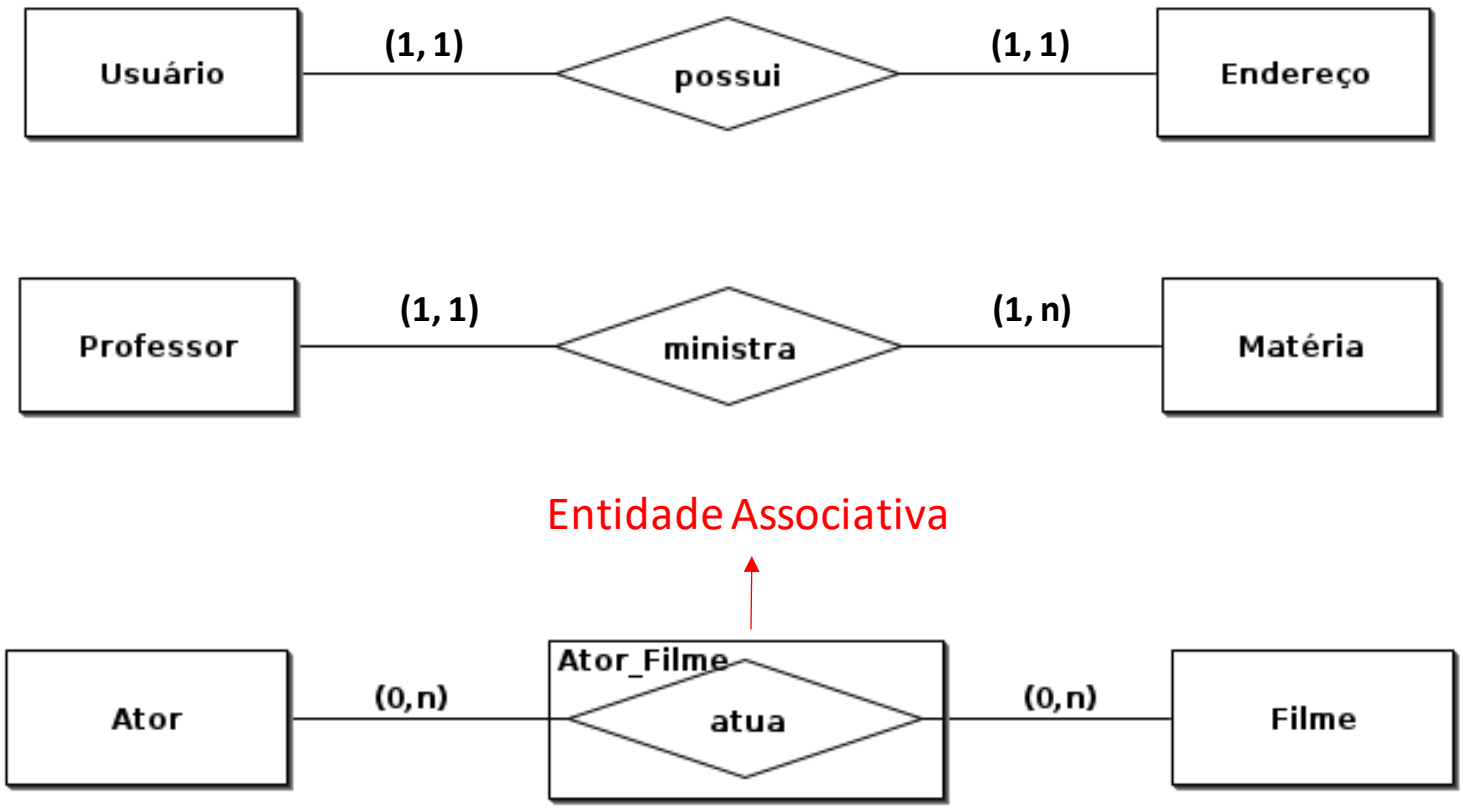
Mapeamento de cardinalidade

Cardinalidade Mínima: número mínimo de vezes em que um registro da entidade A pode ocorrer em B. Pode assumir o valor de 0 ou 1.

Cardinalidade Máxima: número máximo de vezes em que um registro da entidade A pode ocorrer em B. Pode assumir o valor de 1 ou N.

MER

Mapeamento de cardinalidades



MER

Modelagem na prática

Um pequeno país resolveu informatizar sua única delegacia de polícia para criar um banco de dados onde os criminosos deverão ser fichados, sendo que as suas vítimas também deverão ser cadastradas.

No caso de criminosos que utilizem armas, estas deverão ser cadastradas e relacionadas ao crime cometido para possível utilização no julgamento do criminoso.

O sistema, além de fornecer dados pessoais dos criminosos, das vítimas e das armas, também deve possibilitar saber:

- Quais crimes um determinado criminoso cometeu, lembrando que um crime pode ser cometido por mais de um criminoso;
- Quais crimes uma determinada vítima sofreu, lembrando que várias vítimas podem ter sofrido um mesmo crime;

Após o sistema ser colocado em funcionamento, serão definidos relatórios e estatísticas de acordo com a solicitação do chefe da delegacia.

MER

Modelagem na prática

1º Passo: Identificar as entidades.

Um pequeno país resolveu informatizar sua única delegacia de polícia para criar um banco de dados onde os criminosos deverão ser fichados, sendo que as suas vítimas também deverão ser cadastradas.

No caso de criminosos que utilizem armas, estas deverão ser cadastradas e relacionadas ao crime cometido para possível utilização no julgamento do criminoso.

O sistema, além de fornecer dados pessoais dos criminosos, das vítimas e das armas, também deve possibilitar saber:

- Quais crimes um determinado criminoso cometeu, lembrando que um crime pode ser cometido por mais de um criminoso;
- Quais crimes uma determinada vítima sofreu, lembrando que várias vítimas podem ter sofrido um mesmo crime;

Após o sistema ser colocado em funcionamento, serão definidos relatórios e estatísticas de acordo com a solicitação do chefe da delegacia.

MER

Modelagem na prática

2º Passo: Identificar os relacionamentos entre as entidades e definir a cardinalidade.

Criminoso x Vítima:
- Um criminoso pode atacar uma ou mais vítimas;
- Uma vítima pode ser atacada por um ou mais criminosos.

Criminoso x Arma:
- Um criminoso pode utilizar zero ou mais armas;
- Um arma é utilizada por um criminoso.

Criminoso x Crime:
- Um criminoso pode ter cometido um ou vários crimes;
- Um crime pode ter sido cometido por um ou vários criminosos.

Vítima x Crime:
- Uma vítima pode ter sofrido um ou vários crimes;
- Um crime pode ter uma ou várias vítimas.

Arma x Crime:
- Um arma pode ter sido utilizada em um crime;
- Em um crime podem ser utilizadas zero ou várias armas.

MER

Modelagem na prática

3º Passo: Identificar os atributos de cada entidade.

Criminoso: Id, nome e cpf.

Vítima: Id, nome, telefone e cpf.

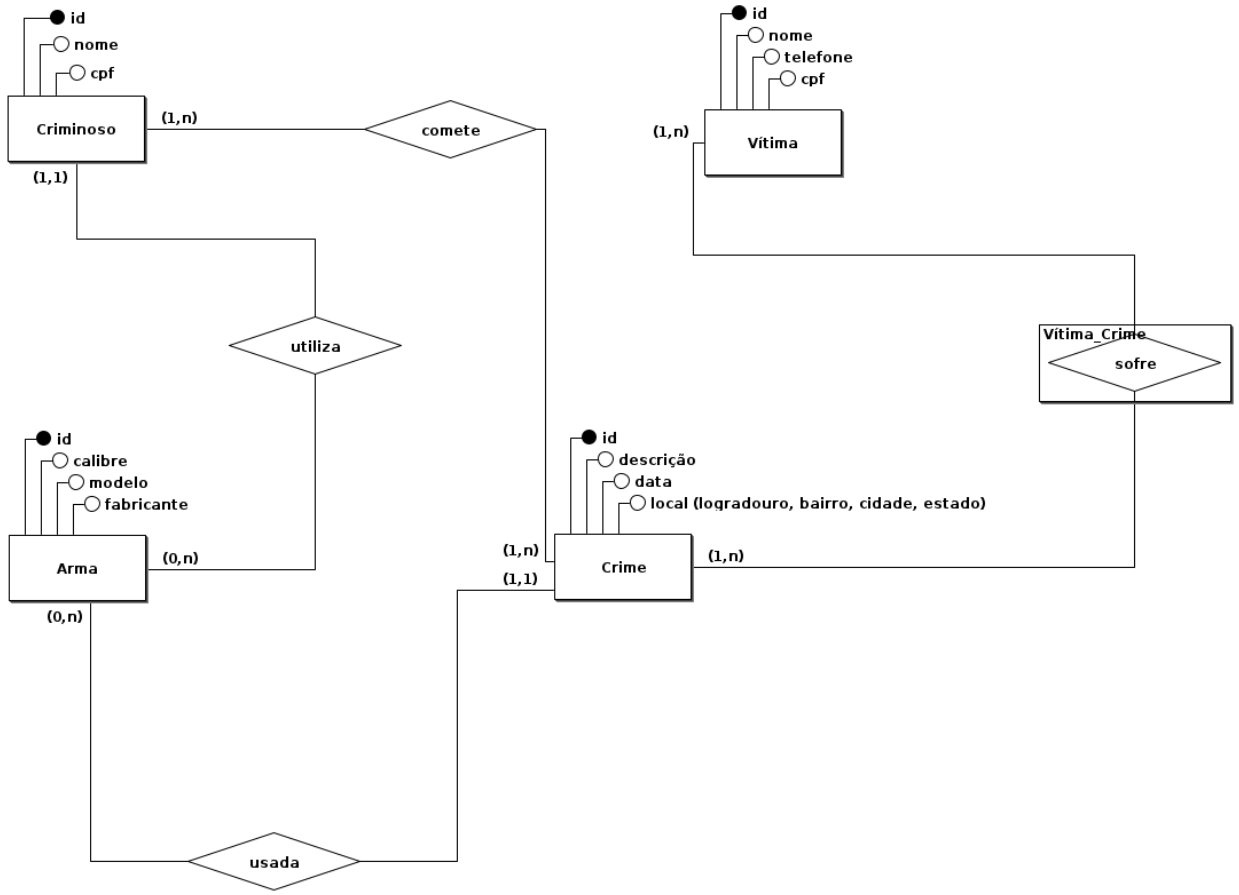
Crime: Id, descrição, local (logradouro, bairro, cidade e estado) e data.

Arma: Id, calibre, modelo e fabricante.

MER

Modelagem na prática

4º Passo: Criar o DER (Diagrama de Entidade-Relacionamento).



Modelagem Lógica

Essa etapa da modelagem contém informações sobre como o modelo deve ser implementado, definindo os **tipos dos atributos** e as **chaves estrangeiras**.

Chave estrangeira é um campo que aponta para a **chave primária** de outra tabela. Ela é utilizada para identificarmos o relacionamento entre as tabelas.

Modelagem Lógica

Mapeamento da chave estrangeira

Regras para utilização da chave estrangeira:

Cardinalidade 1..1: Existe duas possibilidades a primeira é unir as tabelas e a segundo é colocar a chave estrangeira na entidade fraca (Entidade que depende de outra para existir).

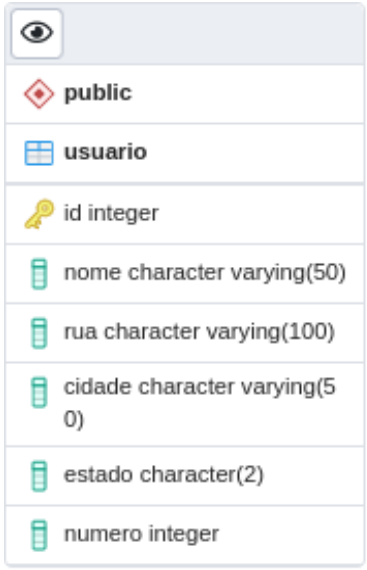
Cardinalidade 1..N: O lado N irá receber a chave estrangeira.

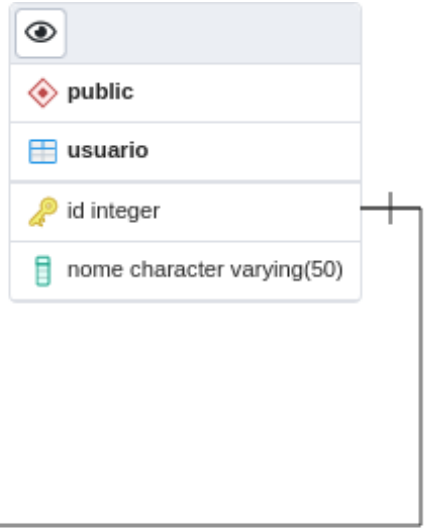
Cardinalidade N..N: É criada uma nova tabela (entidade-associativa) onde a chave primária será composta pelas chaves estrangeiras das duas tabelas.

Modelagem Lógica

Mapeamento da chave estrangeira

Cardinalidade 1..1:





Modelagem Lógica

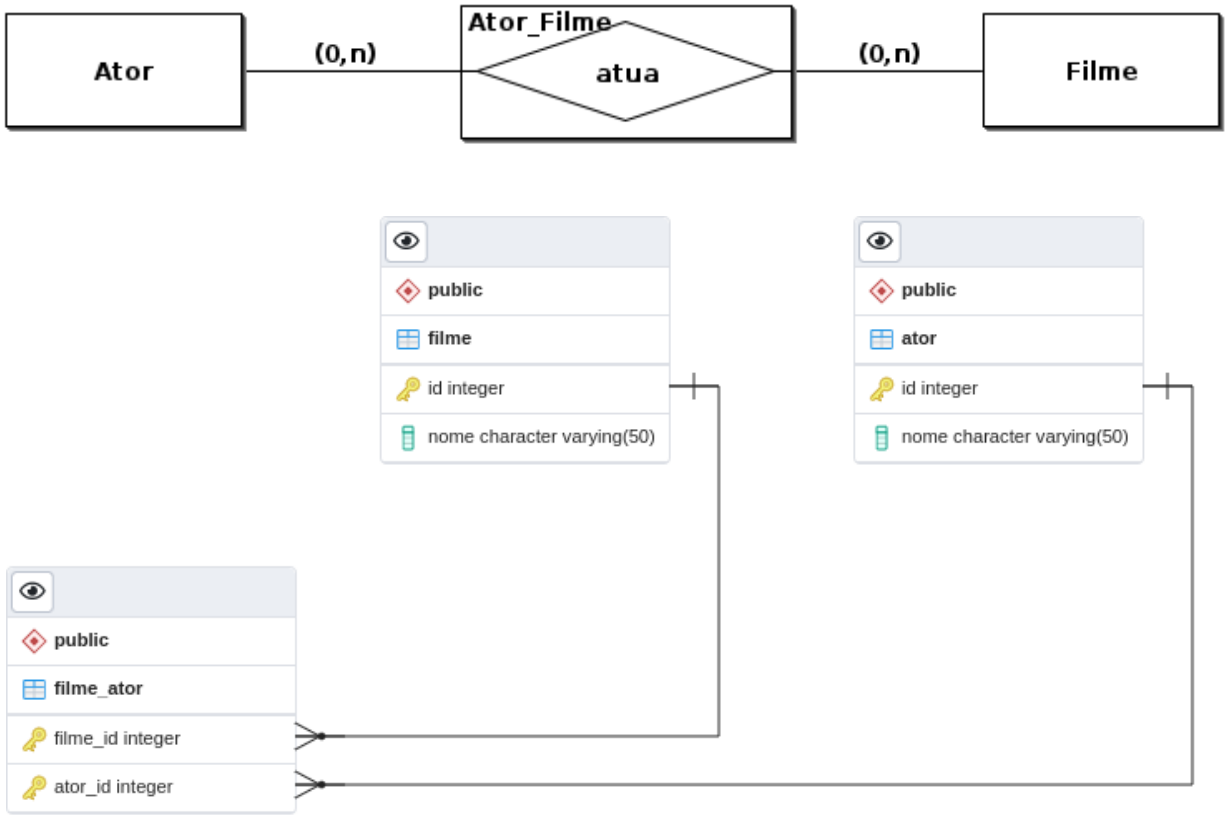
Mapeamento da chave estrangeira

Cardinalidade 1..N:

Modelagem Lógica

Mapeamento da chave estrangeira

Cardinalidade N..N:



Modelagem Lógica

Importância das regras de mapeamento

Marca

id	nome
1	Apple
2	Samsung
3	Xiaomi

Produto

id	nome	id_marca
1	Galaxy A32	2
2	Galaxy A12	2
3	Iphone 11	1

Modelagem Lógica

Importância das regras de mapeamento

Marca

id	nome	id_produto
1	Apple	3
2	Samsung	1
3	Samsung	2

Produto

id	nome
1	Galaxy A32
2	Galaxy A12
3	Iphone 11

Modelagem Lógica

Tipos dos atributos

Nome	Descrição
boolean	usado para armazenar valor lógico (true/false).
character	usado para armazenar caracteres.
text	usado para armazenar texto.
date	usado para armazenar uma data (ano, mês e dia).
integer	usado para armazenar um valor inteiro.
numeric	usado para armazenar um valor com ponto flutuante.
time	usado para armazenar um horário.
timestamp	usado para armazenar data e horário.
uuid	usado para armazenar identificadores universais.

Normalização

Primeira Forma Normal (1FN)

Uma tabela está na 1FN quando respeita as seguintes regras:

- Existe uma chave primária na tabela.
- Colunas contêm valores atômicos (um valor por atributo).

Na primeira forma normal buscamos tratar os atributos multivalorados e compostos.

Atributo multivalorado: conteúdo é formado por mais de um valor.
Ex.: telefone (pode haver mais de um).

Atributo composto: conteúdo é formado por vários itens menores.
Ex.: endereço (rua, número, cidade, estado, etc.).

Normalização

Primeira Forma Normal (1FN)

Pessoa

id	nome	sexo	endereço	telefone
1	Marcos	M	Blumenau, SC, Brasil	999-999, 555-555
2	Maria	F	Rio de Janeiro, RJ, Brasil	123-123
3	Pedro	M	São Paulo, SP, Brasil	345-345, 203-203

Normalização

Primeira Forma Normal (1FN)

Pessoa

id	nome	sexo	cidade	estado	país
1	Marcos	M	Blumenau	SC	Brasil
2	Maria	F	Rio de Janeiro	RJ	Brasil
3	Pedro	M	São Paulo	SP	Brasil

Telefone

id_pessoa	telefone
1	999-999
1	555-555
2	123-123
3	345-345
3	203-203

Chave primária composta: Chave criada a partir de dois campos.

Normalização

Segunda Forma Normal (2FN)

Uma tabela está na 2FN quando respeita as seguintes regras:

- Está na 1FN.
- Não contém **dependências parciais**.

Dependência parcial: quando uma coluna depende de apenas uma parte de uma chave primária composta.

Normalização

Segunda Forma Normal (2FN)

Funcionario_Projeto

id_funcionario	id_projeto	nome_funcionario	nome_projeto	qnt_horas
1	1	Marcos	Projeto 1	10
2	1	Maria	Projeto 1	32
3	2	João	Projeto 2	4

Normalização

Segunda Forma Normal (2FN)

Funcionario

id_funcionario	nome
1	Marcos
2	Maria
3	João

Projeto

id_projeto	nome
1	Projeto 1
2	Projeto 1

Funcionario_Projeto

id_funcionario	id_projeto	qnt_horas
1	1	10
2	1	32
3	2	4

Normalização

Terceira Forma Normal (3FN)

Uma tabela está na 3FN quando respeita as seguintes regras:

- Está na 2FN.
- Não contém **dependências transitivas**.

Dependência transitiva: quando uma coluna depende de outra coluna que não é a chave primária.

Normalização

Terceira Forma Normal (3FN)

Nota_Fiscal

nota_fiscal	cod_vendedor	nome_vendedor	cod_produto	qnt_vendida
751	200	Marcos	111	10
147	201	Maria	112	5
139	200	Marcos	123	8

Normalização

Terceira Forma Normal (3FN)

Nota_Fiscal

nota_fiscal	cod_vendedor	cod_produto	qnt_vendida
751	200	111	10
147	201	112	5
139	200	123	8

Vendedor

cod_vendedor	nome_vendedor
200	Marcos
201	Maria

DDL

Data Definition Language

DDL significa linguagem de definição de dados. Ela é utilizada para definir as estruturas de dados e modificar os dados. Por exemplo, os comandos DDL podem ser utilizados para **adicionar**, **remover** ou **modificar** tabelas dentro de um banco de dados.

DDL

CREATE TABLE

O comando **CREATE TABLE** é utilizado para criar tabelas:

```
CREATE TABLE produtos (
    id SERIAL PRIMARY KEY,
    nome text,
    preco numeric
);
```

A opção **IF NOT EXISTS** pode ser passada para evitar que aconteça um erro caso a tabela já tenha sido criada.

```
CREATE TABLE IF NOT EXISTS produtos (
    id SERIAL PRIMARY KEY,
    nome text,
    preco numeric
);
```

DDL

DEFAULT

Podemos especificar valores padrões para os campos da tabela:

```
CREATE TABLE IF NOT EXISTS produtos (
    id SERIAL PRIMARY KEY,
    nome text,
    preco numeric DEFAULT 9.99
);
```

Dessa forma se não especificarmos nenhum valor para **preco** quando inserirmos valores na tabela, o valor 9.99 será atribuído à essa coluna.

DDL

CONSTRAINTS

Podemos definir algumas restrições para as colunas das tabelas:

- **CHECK:** O valor da coluna deve satisfazer uma expressão booleana.
- **NOT NULL:** O valor da coluna não pode ser nulo.
- **UNIQUE:** O valor da coluna deve ser único.
- **PRIMARY KEY:** Transforma a coluna em chave primária.
- **FOREIGN KEY:** Indica que a coluna é uma chave estrangeira.

DDL

CONSTRAINTS

```sql
CREATE TABLE IF NOT EXISTS produtos (
    id SERIAL PRIMARY KEY,
    nome text NOT NULL UNIQUE,
    preco numeric NOT NULL CHECK (preco > 0)
);

CREATE TABLE IF NOT EXISTS pedidos (
    id SERIAL PRIMARY KEY,
    total_pedido numeric NOT NULL
);

CREATE TABLE IF NOT EXISTS produtos_pedidos (
    id_produto integer,
    id_pedido integer,
    quantidade integer NOT NULL,
    PRIMARY KEY (id_produto, id_pedido),
    FOREIGN KEY (id_produto) REFERENCES produtos (id),
    FOREIGN KEY (id_pedido) REFERENCES pedidos (id)
);
```

DDL

FOREIGN KEY

Existem algumas ações que podem ser específicadas quando deletamos ou atualizamos um registro que está associado a uma FOREIGN KEY:

- SET NULL: todas as linhas que fazem referência ao registro recebem o valor NULL.
- CASCADE: todas as linhas que fazem referência são deletadas junto.

```
CREATE TABLE IF NOT EXISTS produtos_pedidos (
    id_produto integer,
    id_pedido integer,
    quantidade integer NOT NULL,
    PRIMARY KEY (id_produto, id_pedido),
    FOREIGN KEY (id_produto) REFERENCES produtos (id),
    FOREIGN KEY (id_pedido)
        REFERENCES pedidos (id)
        ON DELETE CASCADE
);
```

DDL

Modificando as tabelas

Adicionando colunas:

```
ALTER TABLE produtos ADD COLUMN descricao text;
```

Removendo colunas:

```
ALTER TABLE produtos DROP COLUMN descricao;
```

Renomeando colunas:

```
ALTER TABLE produtos RENAME COLUMN descricao TO informacoes;
```

Renomeando a tabela:

```
ALTER TABLE produtos RENAME TO items;
```

DDL

Removendo tabelas

Para remover as tabelas usamos o comando **DROP TABLE:**

```
DROP TABLE produtos;
```

Podemos utilizar **IF EXISTS** para não gerar algum erro caso a tabela não exista:

```
DROP TABLE IF EXISTS produtos;
```

DML

Data Manipulation Language

DML significa linguagem de manipulação de dados. Ela é utilizada para manipular os dados na tabela.
Os comandos DML podem ser utilizados para **inserir**, **atualizar** e **deletar** registros.

DML

Inserindo dados nas tabelas

Para inserir dados em uma tabela:

```
INSERT INTO produtos VALUES (1, 'caneta', 2.5);
```

Quando temos campos que não precisamos preencher podemos especificar somente os campos que precisam ser inseridos:

```
INSERT INTO produtos (nome, preco) VALUES ('lápis', 1.5);
```

Podemos inserir valores em sequência:

```
INSERT INTO produtos (nome, preco) VALUES
    ('borracha', 0.5),
    ('grafite', 0.5),
    ('régua', 1);
```

DML

Atualizando os dados nas tabelas

Para atualizar um registro na tabela:

```
UPDATE produtos SET preco = 1 WHERE id = 4;
```

```
UPDATE produtos SET preco = 0.25 WHERE nome = 'borracha';
```

Podemos atualizar várias colunas ao mesmo tempo:

```
UPDATE produtos
SET nome = 'caneta bic',
    preco = 3
WHERE nome = 'caneta';
```

DML

Deletando os dados nas tabelas

Para deletar um registro na tabela:

```
DELETE FROM produtos WHERE id = 1;
```

Se não passarmos a clausura WHERE todos os registros da tabela vão ser excluídos:

```
DELETE FROM produtos;
```

DML

Retornando os dados modificados

Para retornar os dados que foram modificados podemos utilizar o **RETURNING:**

```
INSERT INTO produtos (nome, preco) VALUES ('lápis', 2) RETURNING *;
```

Quando utilizamos o * ele irá retornar todos os campos da tabela, porém podemos especificar os campos que queremos no retorno:

```
INSERT INTO produtos (nome, preco) VALUES ('caneta', 3) RETURNING id;
```

DQL

Data Query Language

O último grupo de comandos SQL que veremos é o DQL que significa linguagem de consulta de dados. Esses comandos são utilizados para obter os dados salvos no banco de dados.

DQL

Obtendo os dados

O comando utilizado para obter os dados salvos no banco de dados é o **SELECT:**

```
SELECT * FROM produtos;
```

Quando utilizamos o operador * todas as colunas da tabela serão retornadas. Podemos especificar que colunas queremos na consulta da seguinte forma:

```
SELECT nome, preco FROM produtos;
```

Podemos especificar a quantidade de registros que queremos utilizando o **LIMIT:**

```
SELECT * FROM produtos LIMIT 1;
```

DQL

Juntando tabelas

Quando precisamos obter registros de duas tabelas precisamos utilizar o **INNER JOIN.** Considere as seguintes tabelas com o relacionamento 1:N.

Marca

id	nome
1	Apple
2	Samsung
3	Xiaomi

Produto

id	nome	id_marca
1	Galaxy A32	2
2	Galaxy A12	2
3	Iphone 11	1

DQL

Juntando tabelas

Para obter a seguinte tabela:

nome_produto	nome_marca
Galaxy A32	Samsung
Galaxy A12	Samsung
Iphone 11	Apple

Utilizamos o **INNER JOIN:**

```
SELECT produto.nome AS nome_produto, marca.nome AS nome_marca
FROM produto
INNER JOIN marca ON marca.id = produto.id_marca;
```

DQL

Apelidos de colunas e tabelas

Podemos especificar apelidos para as colunas utilizando a palavra-chave **AS** como mostrado no exemplo anterior. Podemos fazer o mesmo para as tabelas:

```
SELECT p.nome AS nome_produto, m.nome AS nome_marca
FROM produto AS p
INNER JOIN marca AS m ON m.id = p.id_marca;
```

Pode-se omitir a palavra **AS:**

```
SELECT p.nome nome_produto, m.nome nome_marca
FROM produto p
INNER JOIN marca m ON m.id = p.id_marca;
```

DQL

Cláusula WHERE

Podemos utilizar a cláusula WHERE para definir filtros com base em uma expressão booleana:

```
WHERE total_pedido > 10;
```

Pode-se utilizar alguns operadores em conjunto com a cláusula WHERE:

- **BETWEEN:** checa se o valor está em um determinado intervalo.

```
WHERE total_pedido BETWEEN 20 AND 30;
```

- **IN:** checa se o valor está presente em uma lista de valores.

```
WHERE total_pedido IN (10, 20);
```

- **ILIKE:** checa se o valor atende um padrão de string.

```
SELECT * FROM clientes WHERE nome ILIKE 'Jo%';
SELECT * FROM clientes WHERE nome ILIKE 'Jo__';
```

DQL

Funções agregadoras

Existem algumas funções agregadoras que podemos utilizar para obter valores com base nas informações da tabela:

count: Obter a quantidade de registros em uma tabela.

```
SELECT count(id) quantidade_clientes FROM clientes;
```

max/min: Obter o maior/menor valor de determinada coluna.

```
SELECT max(total_pedido) FROM pedidos;
```

avg: Obter a média aritmética de determinada coluna.

```
SELECT ROUND(avg(total_pedido), 2) FROM pedidos;
```

sum: Obter a soma dos valores de determinada coluna.

```
SELECT sum(total_pedido) FROM pedidos;
```

DQL

GROUP BY

As funções agregadoras se aplicam a todos os registros da tabela. Porém existem situações em que queremos agrupar essas funções.

Considere a seguinte tabela de **vendas:**

id	nome_vendedor	quantidade	produto	cidade
1	Jorge	10	Mouse	São Paulo
2	Ana	6	Teclado	Rio de Janeiro
3	Mario	23	Mouse	Blumenau
4	Felipe	25	Webcam	Blumenau
5	João	2	Teclado	Recife
6	Pedro	3	Monitor	São Paulo
7	João	5	Monitor	Rio de Janeiro

DQL

GROUP BY

Para saber a quantidade de vendas por cidade:

```
SELECT cidade, sum(quantidade) FROM vendas GROUP BY cidade;
```

Para saber a quantidade de vendas por produto:

```
SELECT produto, sum(quantidade) FROM vendas GROUP BY produto;
```

Para saber a quantidade de vendas por vendedor:

```
SELECT nome_vendedor, sum(quantidade) FROM vendas GROUP BY nome_vendedor;
```

DQL

ORDER BY

A cláusula **ORDER BY** é utilizada para ordenar os resultados. Essa cláusula aceita duas opções **ASC** (crescente) ou **DESC** (decrescente).

Quantidade de vendas por cidade ordenada pela quantidade de vendas (crescente):

```
SELECT cidade, sum(quantidade) quantidade_vendas
FROM vendas
GROUP BY cidade
ORDER BY quantidade_vendas;
```

Quantidade de vendas por produto ordenada pela quantidade de vendas (decrescente):

```
SELECT produto, sum(quantidade) quantidade_vendas
FROM vendas
GROUP BY produto
ORDER BY quantidade_vendas DESC;
```

Fecomércio SC | Senac

sc.senac.br