Aula 47

Relações em SQL

Labenu_

O que veremos hoje?

- Hoje, vamos finalizar os estudos dos nossos *Banco de Dados* ***Relacionais***: as **relações** entre tabelas

- Falaremos de como criar, ler e inserir informações que sejam representadas como relações **1:1**, **1:N** e **N:M**

Relações

Labenu_

Relações 🔀

- Para entender as relações, vamos considerar um sistema de uma loja simples.

- Ela possui as seguintes entidades
  - Usuários: representam os compradores
  - Fornecedores: representam as empresas que fornecem os produtos
  - Produtos: representam os produtos em si
  - Contas dos usuários: representam a conta do usuário, ou seja, o saldo dele no site

Relações 🔀

- Nos nossos exemplos, vamos sempre considerar que temos 2 usuários

Usuária Alice

Usuário Bob

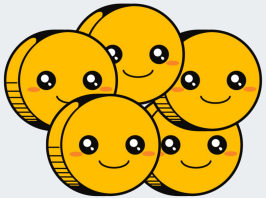
Relações 🔀

```sql
CREATE TABLE relational_user (
    id VARCHAR(255) PRIMARY KEY,
    name VARCHAR(255),
    gender ENUM('male', 'female')
);

INSERT INTO relational_user VALUES
('a', 'Alice', 'female'),
('b', 'Bob', 'male');

SELECT * FROM relational_user;
```

Relações 🔀

- 1ª Relação: Usuários e Contas

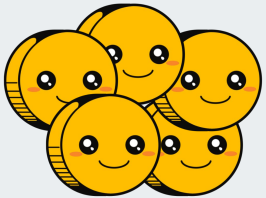
Usuária Alice → Conta da Alice



Relações 🔀

- 1ª Relação: Usuários e Contas

Usuário Bob → Conta do Bob



Relações 🔀

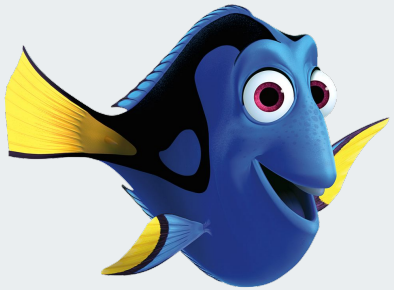
- **1ª Relação: Usuários e Contas**
  - No nosso exemplo, cada usuário tem uma única conta.
  - O oposto, nesse caso, também é verdade. Ou seja uma conta está relacionada a um único usuário
  - Isso caracteriza a relação **1:1**

Relações 🔀

- **2ª Relação: Fornecedores e Produtos**
  - Para prosseguir, vamos dar boas vindas aos fornecedores dos nossos sistemas:

Fornecedor Coragem

Fornecedora Dory

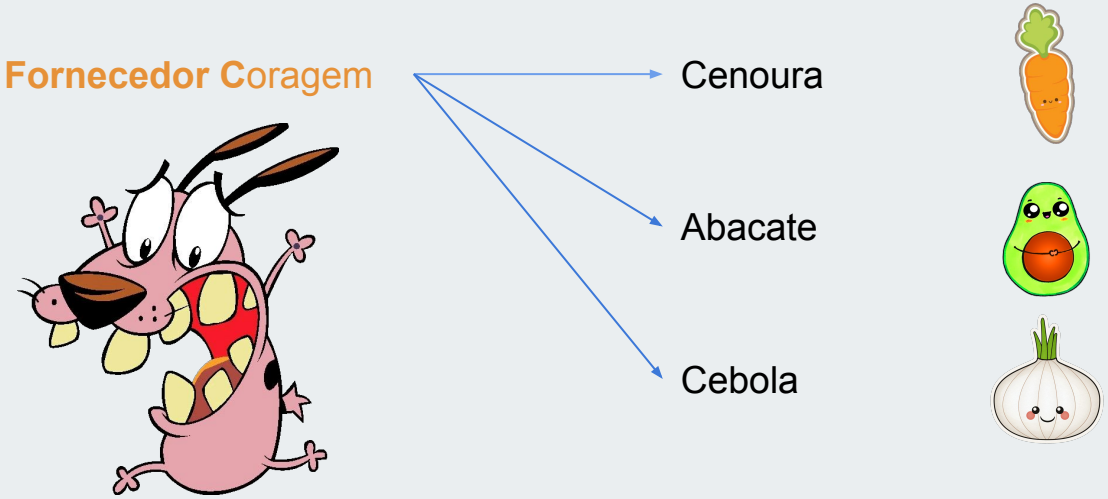


Relações 🔀

```sql
CREATE TABLE relational_supplier (
    id VARCHAR(255) PRIMARY KEY,
    name VARCHAR(255),
    gender ENUM('male', 'female')
);

INSERT INTO relational_supplier
VALUES
('c', 'Coragem', 'male'),
('d', 'Dory', 'female');

SELECT * FROM relational_supplier;
```

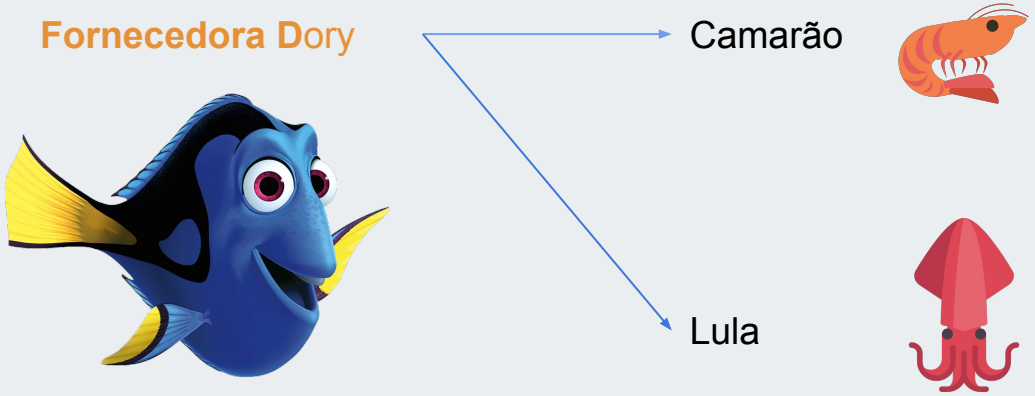
Relações 🔀

- **2ª Relação: Fornecedores e Produtos**
  - O Coragem tem uma fazenda muito grande, então, ele fornece (e só ele), para nossa loja, todos os "produtos da terra"



Relações 🔀

- **2ª Relação: Fornecedores e Produtos**
  - Já a Dory tem uma boa habilidade para caçar frutos do mar. Então, é isso que ela vende para nós, e só ela.



Relações 🔀

- **2ª Relação: Fornecedores e Produtos**
  - Isso significa que cada fornecedor nos vende vários produtos. Ou seja, um fornecedor está relacionado a vários produtos.

  - Só que nós compramos cada produto apenas de um fornecedor. Ou seja, um produto está relacionado a um fornecedor.

  - Isso é chamado de relação **1:N**

Relações 🔀

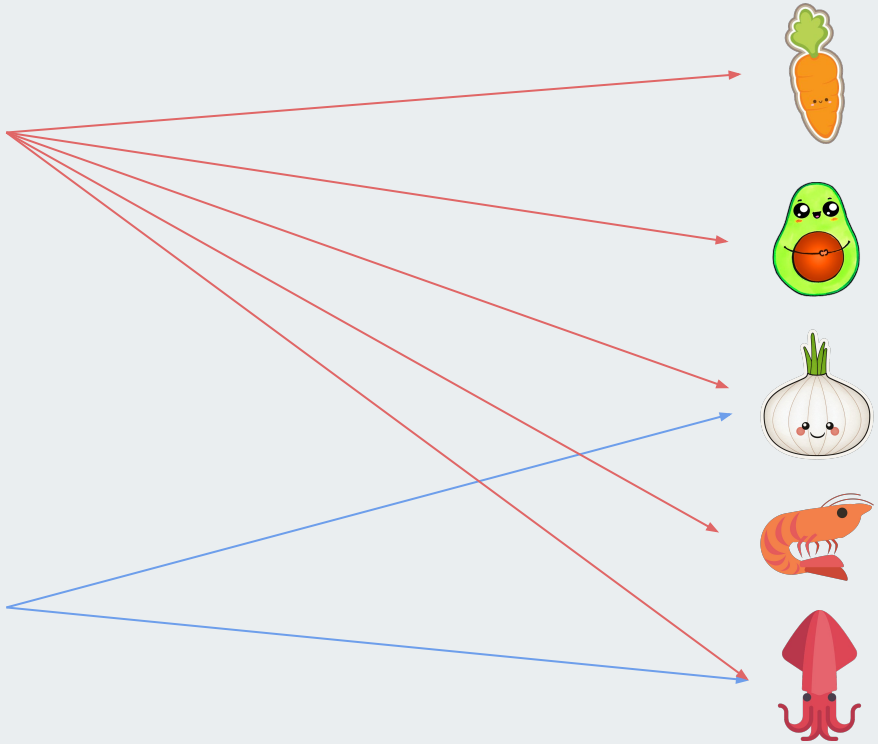
- **3ª Relação: Usuários e Produtos**
  - Bem, tanto a Alice como o Bob colocaram saldos em suas contas
  - Agora, eles começarão a fazer compras. Alice ama comer lula com cebola.
  - Já o Bob, prefere sair de casa uma vez só. Então, vai comprar tudo que tem lá

Relações 🔀

- 3ª Relação: Usuários e Produtos

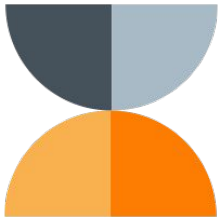
Usuário Bob

Usuária Alice



Relações 🔀

- **3ª Relação: Usuários e Produtos**
  - Cada usuário pode escolher vários produtos. Ou seja, um usuário está relacionado a vários produtos
  - Cada produto pode ser escolhido por vários usuários. Ou seja, um produto está relacionado a vários usuários
  - Isso é chamado de relação **N:M**



- **Relações 1:1**
  - Um item se relaciona com um item
- **Relações 1:N**
  - Um item se relaciona com vários itens
- **Relações N:M**
  - Vários itens se relacionam com vários itens

Criando Relações em MySQL

Labenu_

Criando Relações em MySQL 👷‍♀️

- Para criamos relações, precisaremos dar um jeito de relacionar (ou referenciar) uma tabela com outra, certo?

- Para isso, usamos a *FOREIGN KEY*, que indica que a propriedade em questão é uma chave estrangeira

- *FOREIGN KEY* deve sempre se referenciar a uma *PRIMARY KEY da outra tabela*

Criando Relações em MySQL 👷🏽‍♀️

- Tendo isso em mente, vamos ver, agora, como criamos tabelas e inserimos elementos em relações

- Vamos separar nosso estudo nos três casos que já vimos:
  - Relação 1:1
  - Relação 1:N
  - Relação N:M

Criando Relações em MySQL 

- **Relação 1:1**

```
CREATE TABLE relational_account (
    id INT PRIMARY KEY,
    balance FLOAT,
    user_id VARCHAR(255),
    FOREIGN KEY (user_id) REFERENCES relational_user(id)
);

INSERT INTO relational_account VALUES
(659182, 1000.99, 'a'),
(662834, 1000.99, 'b');

SELECT * FROM relational_user;
```

Criando Relações em MySQL 👷🏼‍♀️

- **Relação 1:1**
  - *Porque guardamos esses valores em tabelas diferentes?*
  - Eventualmente, por *questões de performance:*
    - Se alguma das colunas é lida ou alterada com muitíssima mais frequência do que as demais, talvez valha a pena guardar em tabelas diferentes

Criando Relações em MySQL 

- **Relação 1:N**

```
CREATE TABLE relational_product (
    id VARCHAR(255) PRIMARY KEY,
    name VARCHAR(255) UNIQUE,
    price FLOAT,
    supplier_id VARCHAR(255),
    FOREIGN KEY (supplier_id ) REFERENCES relational_supplier(id)
);

INSERT INTO relational_product VALUES
('okm','cenoura' , 10.00, 'c'),
('uhb','abacate', 11.50, 'c'),
('ygv','cebola', 12.00, 'c'),
('ijn','camarão', 21.50, 'd'),
('tfc','lula', 22.00, 'd');

SELECT * FROM relational_product;
```

Criando Relações em MySQL 👷

- **Relação N:M**
  - Nessas relações, os elementos de uma das tabelas se relacionam com vários da outra tabela
  - Vários produtos se referem a vários usuários
  - Então agora, apenas uma chave estrangeira não resolve

Criando Relações em MySQL 👷

- **Relação N:M**
  - Vamos, então, criar uma tabela intermediária ou de junção. Vamos chamá-la de Compras

```
CREATE TABLE relational_sale (
    user_id VARCHAR(255),
    product_id VARCHAR(255),
    FOREIGN KEY (user_id ) REFERENCES relational_user(id),
    FOREIGN KEY (product_id ) REFERENCES relational_product(id)

);

INSERT INTO relational_sale VALUES
('b', 'okm'), ('b','uhb'), ('a','ygv'), ('b','ygv'),
('b','ijn'), ('b','tfc'), ('a','tfc');


SELECT * FROM relational_sale;
```

Inserindo e Deletando Relações

Labenu_

Inserindo e Deletando Relações 🛠️

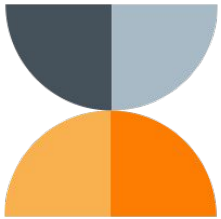
- Para **criar** um elemento que possui uma chave estrangeira, nós temos que passar para ele a chave de um elemento **que já exista na outra tabela**.

- ***Antes de criar uma conta...***
  - Precisamos ter um usuário criado
  - Pegamos o id deste usuário
  - Colocamos na query de criação da conta

Inserindo e Deletando Relações 🛠️

- Para **deletar** um elemento que **foi usado** na criação de um elemento de **outra tabela**, precisamos deletar todas as **referências** dele naquela tabela

- ***Antes de deletar um produto...***
  - Precisamos deletar todas as referências que têm dele no carrinho



- Para a criação de tabelas que utilizam relações, precisamos declarar alguma das **propriedades como chave estrangeira**

- Para criar um item, precisamos **passar a chave estrangeira**

- Para deletar um item, precisamos **deletar todas as referências dele**

Buscando Relações

Labenu_

Buscando Relações 

- Agora que temos tabelas se relacionando, podemos fazer queries que buscam dados de mais de uma tabela
- Para isso, existe o operador **JOIN**
- **Junta os registros** das tabelas em uma única resposta de query
- Podemos passar uma **condição** pela qual as tabelas serão juntadas

Buscando Relações 

- **INNER JOIN:**
  - Retorna registros relacionados nas duas tabelas
  - Se uma condição não é provida, são retornadas combinações de todos os itens das duas tabelas
  - Pode ser abreviado somente para **JOIN**

```
SELECT *
FROM relational_user
JOIN relational_account
ON relational_account.user_id =
relational_user.id;
```

Buscando Relações 

```
SELECT *
FROM facebook
JOIN linkedin
ON facebook.name = linkedin.name
```

facebook

Name	# of Friends
Matt	300
Lisa	500
Jeff	600
Sarah	400

linkedin

Name	# of connections
Matt	500
Lisa	200
Sarah	100
Louis	300

facebook and linkedin JOINed

facebook.Name	facebook.# of Friends	linkedin.Name	linkedin.# of connections

Buscando Relações 🕵🏻

- **LEFT JOIN:**
  - Ele busca os elementos que estão na primeira tabela ("esquerda do JOIN")

  - Então, procura todas as relações desses elementos na segunda tabela. Junta as informações e retorna

```
SELECT colunas
FROM tabela1
LEFT JOIN tabela2 ON condicao;
```

Buscando Relações 

```
SELECT *
FROM facebook
LEFT JOIN linkedin
ON facebook.name = linkedin.name
```

facebook

Name	# of Friends
Matt	300
Lisa	500
Jeff	600
Sarah	400

linkedin

Name	# of connections
Matt	500
Lisa	200
Sarah	100
Louis	300

facebook and linkedin JOINed

facebook.Name	facebook.# of Friends	linkedin.Name	linkedin.# of connections

Buscando Relações 🕵️

- **RIGHT JOIN:**
  - Ele busca os elementos que estão na segunda tabela ("direita do JOIN")

  - Então, procura todas as relações desses elementos na primeira tabela. Junta as informações e retorna

```
SELECT colunas
FROM tabela1
RIGHT JOIN tabela2 ON condicao;
```

Buscando Relações 

- **JOIN na tabela de junção:**

```
SELECT relational_user.name, relational_product.name AS
'product'
FROM relational_sale
JOIN relational_user
ON user_id = relational_user.id
JOIN relational_product
ON product_id = relational_product.id;
```

Resumo

Labenu_

Resumo 

- **Relações 1:1**
  - Um item se relaciona com um item
- **Relações 1:N**
  - Um item se relaciona com vários itens
- **Relações N:M**
  - Vários itens se relacionam com vários itens

Resumo 

- Para a criação de tabelas que utilizam relações, precisamos declarar alguma das **propriedades como chave estrangeira**

- Para criar um item, precisamos **passar a chave estrangeira**

- Para deletar um item, precisamos **deletar todas as referências dele**

Resumo 

- As *Queries* com tabelas que possuem reações podem trazer dados das duas tabelas

- Para isso, usamos o operador **JOIN**, em uma das suas três versões:
  - **INNER** JOIN
  - **LEFT** JOIN
  - **RIGHT** JOIN

Obrigado!