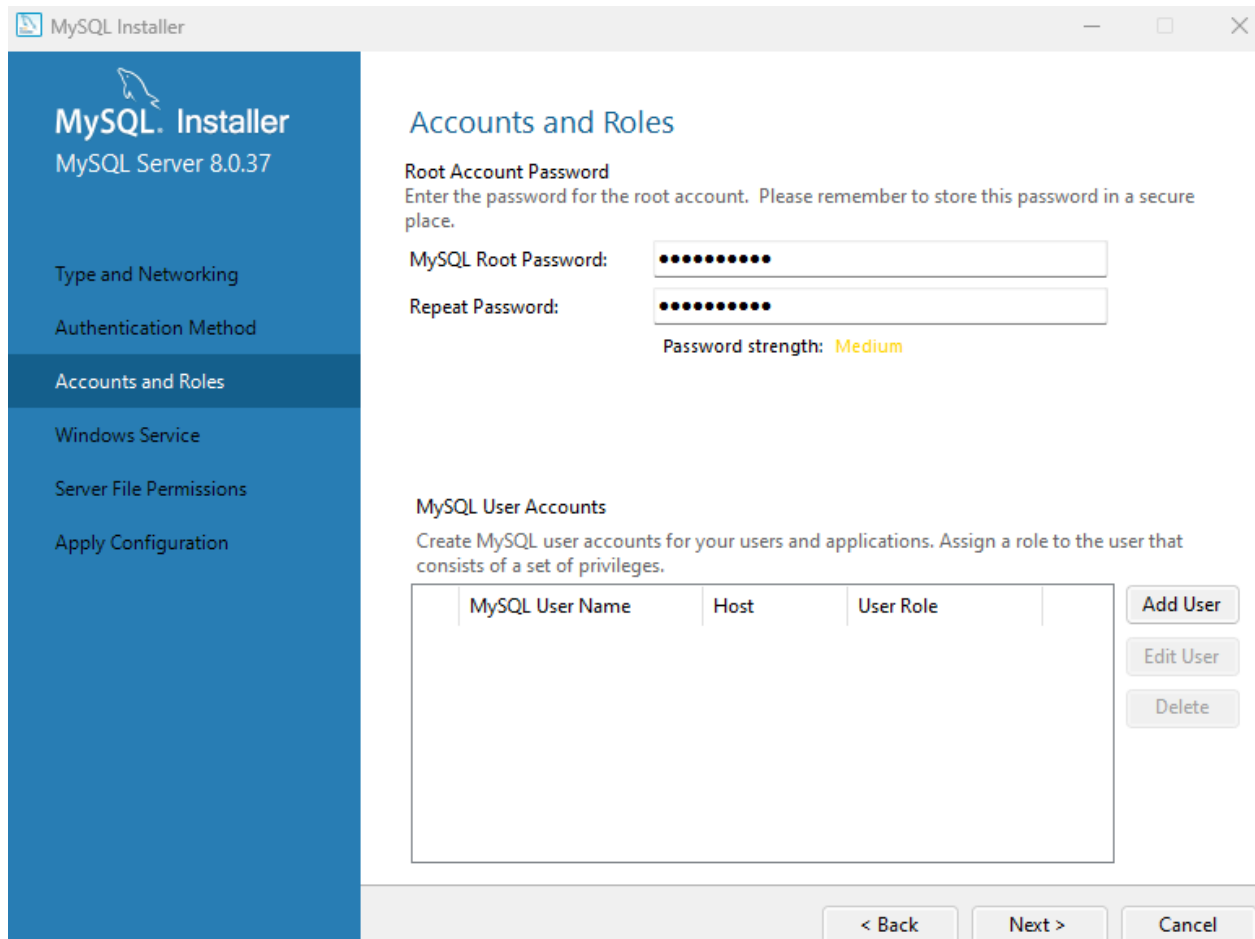


## Vídeos Úteis

[\[FIXED\] XAMPP Error: MySQL shutdown unexpectedly | Repair Corrupted Database](#)

## Login



MySQL Installer

MySQL Server 8.0.37

Type and Networking

Authentication Method

**Accounts and Roles**

Windows Service

Server File Permissions

Apply Configuration

### Accounts and Roles

**Root Account Password**  
Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password:

Repeat Password:

Password strength: **Medium**

**MySQL User Accounts**  
Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.

MySQL User Name	Host	User Role
-----------------	------	-----------

Add User

Edit User

Delete

< Back

Next >

Cancel

123.Arthur

## Introdução

### O que é SQL?

SQL (Structured Query Language) é uma **linguagem de banco de dados** usada para gerenciar e manipular dados em sistemas de bancos de dados relacionais, como **MySQL**, **PostgreSQL** e **SQL Server**. Ela permite que os usuários realizem várias operações, como criar, modificar e consultar dados.

Com SQL, é possível:

- **Criar e alterar** a estrutura dos bancos de dados (tabelas, índices, etc.)
- **Inserir, atualizar e excluir** dados nas tabelas
- **Consultar** (fazer *queries*) para obter informações específicas

## Principais Comandos SQL:

- **INSERT:** Insere novos dados em uma tabela.

Exemplo:

```
INSERT INTO Funcionarios (Nome, Cargo, Salario) VALUES ('Maria', 'Analista', 2500);
```

- **UPDATE:** Atualiza dados existentes.

Exemplo:

```
UPDATE Funcionarios SET Salario = 3000 WHERE Nome = 'Maria';
```

- 
- **SELECT:** Consulta e retorna dados de uma ou mais tabelas.

Exemplo:

sql

Copiar código

```
SELECT Nome, Salario FROM Funcionarios WHERE Cargo = 'Analista';
```

- 

## O que é um Banco de Dados Relacional?

Um **banco de dados relacional** organiza os dados em **tabelas** (linhas e colunas), onde cada linha representa um registro e cada coluna representa um atributo do dado. O SQL é a linguagem padrão para interagir com esses tipos de bancos de dados.

## Diferença entre SQL e MySQL

**SQL (Structured Query Language)**

- **O que é:** SQL é uma **linguagem** usada para gerenciar e manipular dados em um banco de dados relacional. Ou seja, SQL é o meio pelo qual você interage com o banco de dados, criando, consultando, atualizando ou deletando dados.
- **Função principal:** Com SQL, você pode criar tabelas, definir relacionamentos entre dados e executar consultas para extrair informações.
- **Exemplo de uso:**

Criar uma tabela:

```
CREATE TABLE Funcionarios (
    ID INT,
    Nome VARCHAR(100),
    Cargo VARCHAR(50)
);
```

○

Consultar dados:

```
SELECT Nome, Cargo FROM Funcionarios WHERE Cargo = 'Analista';
```

○

## MySQL

- **O que é:** MySQL é um **software** de gerenciamento de banco de dados (SGBD) que utiliza a linguagem SQL para interagir com os dados. Ele é um dos **SGBDs mais populares** e é amplamente usado para gerenciar grandes volumes de dados em aplicações web e sistemas empresariais.
- **Função principal:** O MySQL gerencia o armazenamento físico dos dados e processa as instruções SQL que você escreve. Ele é responsável por garantir a segurança, performance e confiabilidade do banco de dados.
- **Exemplo de uso:** Ao instalar o MySQL, você pode usar SQL para criar, gerenciar e manipular os dados no banco que o MySQL está gerenciando.

## Resumo da Diferença:

- **SQL:** É a **linguagem** usada para interagir com qualquer banco de dados relacional.
- **MySQL:** É um **software** específico de gerenciamento de banco de dados que usa a linguagem SQL para processar dados.

## Comparação

Característica	SQL	MySQL
O que é?	Linguagem de consulta para BD	Sistema de gerenciamento de banco de dados
Função	Manipulação de dados	Armazenamento e gerenciamento de dados
Exemplo de uso	Escrever consultas (SELECT, INSERT)	Executar, armazenar e proteger dados
Tipo	Padrão de linguagem	Software específico

Essa explicação deixa claro que SQL é a linguagem, enquanto MySQL é o software que usa SQL para gerenciar os dados.

## O que é um Banco de Dados?

Um **banco de dados** é um local onde os dados de um sistema são armazenados de forma estruturada. Ele é essencial para gerenciar informações de forma eficiente, permitindo que os dados sejam organizados, acessados e manipulados conforme necessário.

- **Criação:** Um banco de dados é geralmente criado utilizando a linguagem SQL (Structured Query Language), que define sua estrutura e permite manipular os dados.
- **Gerenciamento:** Softwares de gerenciamento de banco de dados, como o **MySQL**, são usados para armazenar, proteger e acessar esses dados de maneira eficiente.

## Principais Elementos de um Banco de Dados (Data Base)

### 1. Diagrama do Banco de Dados

- O **Diagrama do Banco** é um projeto ou um desenho visual que representa como o banco de dados vai funcionar. Ele mostra as tabelas, as relações entre elas e outras estruturas, facilitando a compreensão do modelo de dados.
- **Exemplo:** Em um sistema de gerenciamento de funcionários, o diagrama pode mostrar como a tabela de funcionários se relaciona com a tabela de departamentos.

### 2. Banco de Dados

- O **Banco de Dados** é a **entidade** principal que contém todos os elementos, como tabelas, índices e procedimentos armazenados. Ele é como um "grande container" onde tudo relacionado ao sistema de dados está armazenado.

### 3. Tabelas

- As **tabelas** são o **principal componente** de um banco de dados. Cada tabela armazena uma categoria de dados. Por exemplo, uma tabela pode armazenar dados de **funcionários**, enquanto outra tabela armazena dados de **departamentos**.
- **Exemplo de Tabela (Funcionários):**

4.

ID	Nome		Cargo	Departamento
1	Maria Silva		Analista	TI
2	João Pereira		Gerente	RH

5.

### Colunas

- As **colunas** definem o **tipo de informação** armazenada em uma tabela. Cada coluna representa um campo de dados. Exemplo: Nome, Cargo, Departamento são colunas na tabela Funcionários.
- **Exemplo de Colunas na Tabela Funcionários:**
  - ID: Identificação única do funcionário
  - Nome: Nome do funcionário
  - Cargo: Posição do funcionário na empresa

### 6. Dados

- Os **dados** são os **valores inseridos** nas tabelas pelos usuários ou pelo sistema. São os valores reais que preenchem as colunas de cada tabela. Eles representam a informação que o banco de dados está armazenando.
- **Exemplo de Dados:**
  - "Maria Silva" no campo Nome
  - "Gerente" no campo Cargo

## CRUD: Operações Básicas em um Banco de Dados

As operações básicas para manipular dados em um banco de dados são resumidas pela sigla **CRUD**:

- **Create (Criar)** – Inserir novos dados.
  - Exemplo: `INSERT INTO Funcionarios (Nome, Cargo) VALUES ('Ana Costa', 'Assistente');`
- **Read (Ler)** – Consultar e ler dados existentes.
  - Exemplo: `SELECT * FROM Funcionarios WHERE Cargo = 'Gerente';`
- **Update (Atualizar)** – Modificar dados existentes.
  - Exemplo: `UPDATE Funcionarios SET Cargo = 'Gerente' WHERE Nome = 'Ana Costa';`

- **Delete (Deletar)** – Remover dados.
  - Exemplo: `DELETE FROM Funcionarios WHERE Nome = 'João Pereira';`

## ***Criando um Banco de Dados***

*Ao usar SQL para criar um banco de dados, você começa definindo o nome do banco e, em seguida, pode visualizar todos os bancos criados no sistema. Aqui estão os comandos básicos:*

### ***1. Criar um Banco de Dados***

*Para criar um novo banco de dados, usamos o comando `CREATE DATABASE`. Esse comando define o nome do banco e o cria no servidor de banco de dados.*

**Comando:**

```
CREATE DATABASE nome_do_banco;
```

**Exemplo:**

```
CREATE DATABASE EmpresaDB;
```

*Esse exemplo cria um banco de dados chamado ***EmpresaDB***. Você pode substituir *EmpresaDB* por qualquer nome que faça sentido para o seu projeto.*

### ***2. Mostrar os Bancos de Dados Existentes***

*Depois de criar o banco de dados, você pode listar todos os bancos criados no servidor usando o comando `SHOW DATABASES`.*

**Comando:**

```
SHOW DATABASES;
```

**Exemplo de Retorno:**

```
+-----+
| Database |
```

```
+-----+
| EmpresaDB |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
```

Aqui, ***EmpresaDB*** aparece na lista de bancos de dados, junto com outros bancos padrão do sistema, como ***mysql*** e ***performance\_schema***.

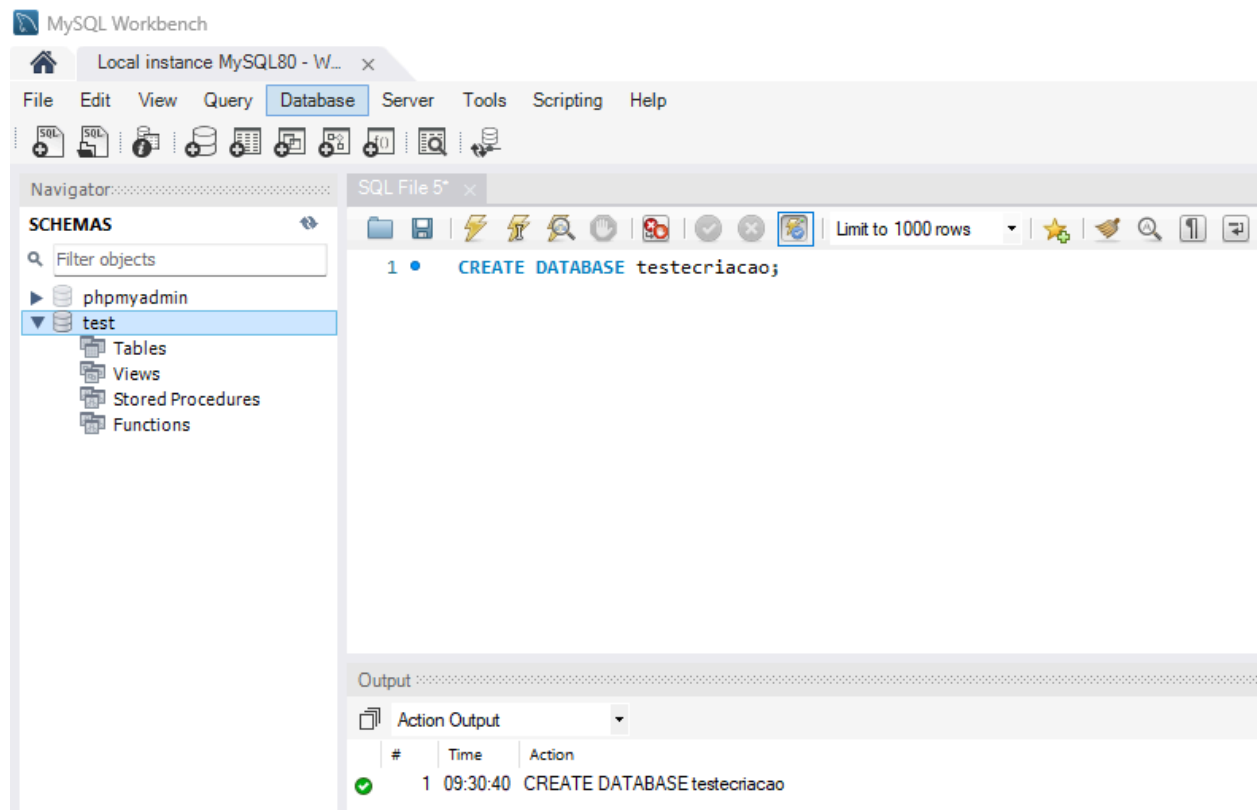
### **Resumo dos Comandos**

Comando	Função
CREATE DATABASE	Cria um novo banco de dados
SHOW DATABASES	Lista todos os bancos de dados no servidor

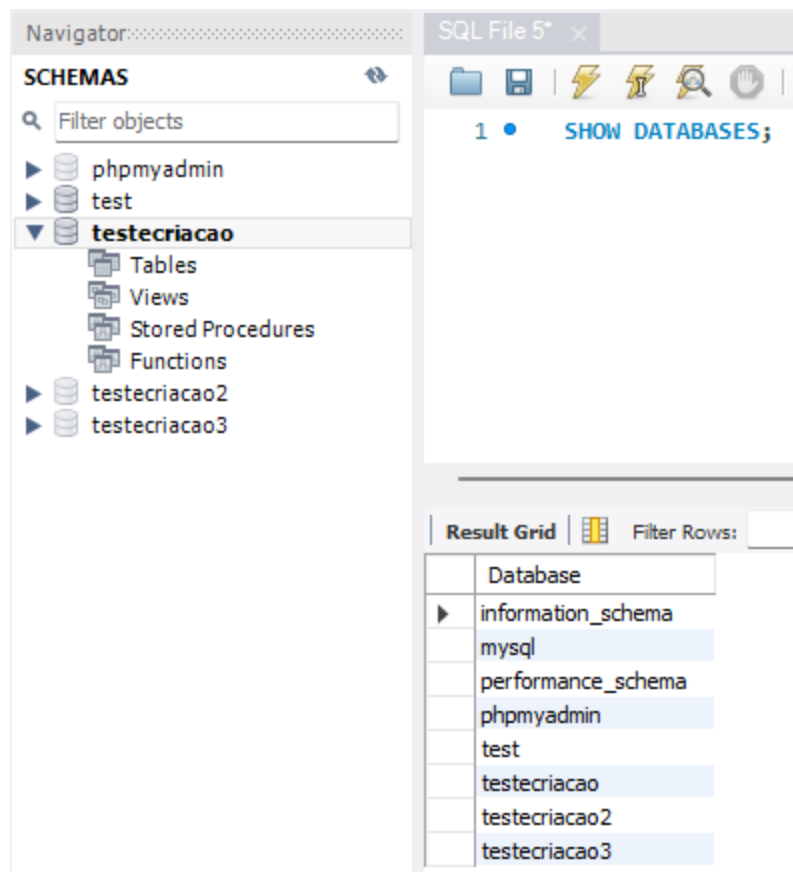
---

Esses são os comandos básicos para começar a trabalhar com bancos de dados no SQL. Com isso, você pode criar e visualizar os bancos de dados no sistema de gerenciamento, como MySQL, PostgreSQL ou SQL Server.

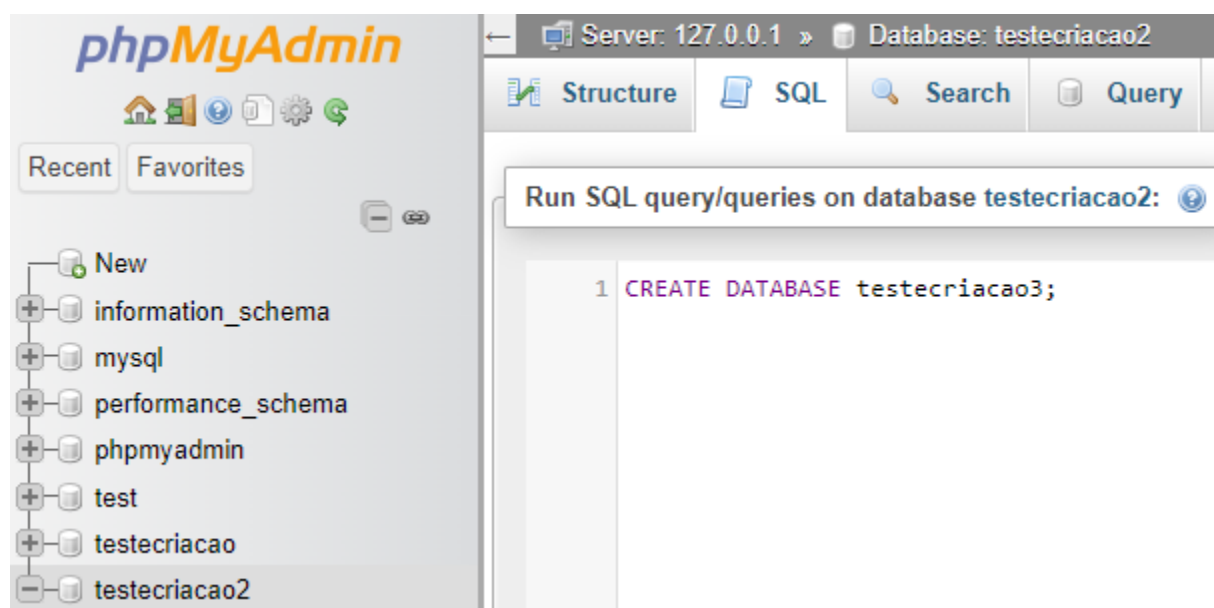
## Workbench







## PHPmyAdmin



phpMyAdmin

Recent Favorites

Server: 127.0.0.1 » Database: testecriacao2

Structure SQL Search Que

Show query box

✓ MySQL returned an empty result set (i.e. zero rows).

```
CREATE DATABASE testecriacao3;
```

[ Edit inline ] [ Edit ] [ Create PHP code ]

New

- information\_schema
- mysql
- performance\_schema
- phpmyadmin
- test
- testecriacao
- testecriacao2
- testecriacao3

phpMyAdmin

Recent Favorites

Server: 127.0.0.1 » Database: testecriacao2

Structure SQL Search Query

Show query box

⚠ Current selection does not contain a unique column. Grid ed

Your SQL query has been executed successfully.

```
SHOW DATABASES;
```

☐ Profiling [ Edit inline ] [ Edit ] [ Create PHP code ] [ Refresh ]

Extra options

Database

- information\_schema
- mysql
- performance\_schema
- phpmyadmin
- test
- testecriacao
- testecriacao2
- testecriacao3

## Sintaxe do SQL

No SQL, há algumas convenções de escrita que ajudam a manter o código organizado e fácil de ler. Aqui estão as principais regras:

1. **Comandos em UPPERCASE:**

- Os comandos SQL, como **SELECT**, **CREATE**, **INSERT**, **UPDATE**, e outros, são escritos **em letras maiúsculas** para facilitar a leitura e diferenciação do que é comando e do que são nomes de tabelas ou colunas.

2. **Nomes de Tabelas e Colunas em lowercase:**

- Os nomes de **tabelas**, **colunas** e outros identificadores geralmente são escritos **em letras minúsculas**. Isso não é uma regra obrigatória, mas é uma boa prática para manter a consistência e evitar confusões.

3. **Uso de ponto e vírgula (;):**

- Cada instrução SQL deve terminar com um **ponto e vírgula (;)**. Embora em alguns casos, dependendo do ambiente, o ponto e vírgula possa não ser obrigatório, **é recomendado sempre colocá-lo** para evitar problemas e garantir que o comando seja finalizado corretamente.

---

## Exemplo de Sintaxe SQL Correta:

```
CREATE DATABASE empresa_db;
```

```
CREATE TABLE funcionarios (  
    id INT PRIMARY KEY,  
    nome VARCHAR(100),  
    cargo VARCHAR(50),  
    salario DECIMAL(10, 2)  
);
```

```
INSERT INTO funcionarios (id, nome, cargo, salario)  
VALUES (1, 'Maria Silva', 'Analista', 2500.00);
```

```
SELECT nome, cargo, salario  
FROM funcionarios  
WHERE cargo = 'Analista';
```

- **Comandos em UPPERCASE:** **CREATE**, **INSERT**, **SELECT**, **WHERE**

- Nomes de tabelas e colunas em lowercase: `empresa_db`, `funcionarios`, `nome`, `cargo`, `salario`
  - Ponto e vírgula no final de cada instrução para garantir que o comando foi finalizado.
- 

Seguindo essa estrutura, seu código SQL fica mais claro, organizado e com menos chances de erro ao ser executado.

## Importação de DB

- Selecionar o banco com o comando **USE**;
- Utilizar com o comando **SOURCE**;

Comandos

`USE<name>`

`source<file>`

INTRO

- empresa.sql
- empresa2.sql

Start

- New File...
- Open File...
- Open Folder...
- Clone Git Repository...

Walkthroughs

- Get Started with VS Code
  - Customize your editor, learn the basics, and
- Learn the Fundamentals

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\arthu\Desktop\Arthur\SQL e MySQL de forma prática e objetiva, e ainda crie projetos com PHP e MySQL\Intro> **mysql** -u root

Welcome to the MariaDB monitor. Commands end with ; or \g.

Your MariaDB connection id is 131

Server version: 10.4.32-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE DATABASE empresa;

Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> SHOW DATABASES;

Database
arthurguilherme
empresa
information_schema
mysql
performance_schema
phpmyadmin
test
testecriacao
testecriacao2
testecriacao3

10 rows in set (0.001 sec)

MariaDB [(none)]> USE empresa;

Database changed

MariaDB [empresa]> SOURCE empresa.sql

> OUTLINE

> TIMELINE

## Workbench

MySQL Workbench

Local instance MySQL80 - W...

File Edit View Query Database Server Tools Scripting Help

Navigator: SCHEMAS

Filter objects

- arthurguilherme
- empresa
  - funcionarios
  - servicos
- Views
- Stored Procedures
- Functions
- phpmyadmin
- test
- testecriacao
- testecriacao2
- testecriacao3

SQL File 5\* funcionarios

```
1 • SELECT * FROM empresa.funcionarios;
```

Result Grid

	id	nome	idade
▶	1	Juliana	23
	2	João	30
	3	Maria	42
	4	Pedro	19
*	NULL	NULL	NULL

## PHPmyAdmin

phpMyAdmin

Server: 127.0.0.1 » Database: empresa

Structure SQL Search Query Export Import Operations Privileges Routines Events

Filters

Containing the word:

Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> funcionarios	Browse  Structure  Search  Insert  Empty  Drop	4	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> servicos	Browse  Structure  Search  Insert  Empty  Drop	5	InnoDB	utf8mb4_general_ci	16.0 KiB	-
2 tables	Sum	9	InnoDB	utf8mb4_general_ci	32.0 KiB	0 B

↑ ☐ Check all With selected: ▼

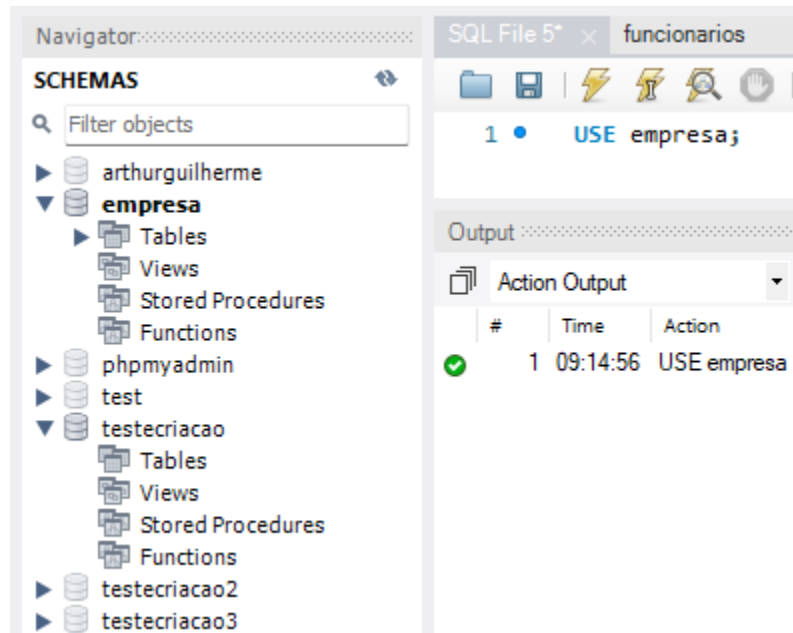
Print Data dictionary

Create new table

Table name:  Number of columns:

## Selecionando Dados de uma Tabela

\*Sempre selecionar DB



## Comandos

*SELECT \* FROM nomeDB*

Navigator

**SCHEMAS**

Filter objects

- arthurguilherme
  - empres
    - Tables
      - funcionarios
      - servicos
    - Views
    - Stored Procedures
    - Functions
  - phpmyadmin
  - test
  - testecriacao
    - Tables
    - Views
    - Stored Procedures
    - Functions
  - testecriacao2
  - testecriacao3

SQL File 5\* x funcionarios

Result Grid

	id	nome	idade
▶	1	Juliana	23
	2	João	30
	3	Maria	42
	4	Pedro	19
*	NULL	NULL	NULL

funcionarios 1 x

Output

Action Output

#	Time	Action
✓ 1	09:14:56	USE empresa
✓ 2	09:20:49	SELECT * FROM funcionarios LIMIT 0, 1000

Recent Favorites

- New
- arthurguilherme
- empres
  - New
  - funcionarios
  - servicos
- information\_schema
- mysql
- performance\_schema
- phpmyadmin
- test
- testecriacao
- testecriacao2
- testecriacao3

Show query box

✓ Showing rows 0 - 3 (4 total, Query took 0.0001 seconds.)

```
SELECT * FROM funcionarios;
```

☐ Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ]

☐ Show all | Number of rows: 25 Filter rows

Extra options

	id	nome	idade
<input type="checkbox"/> Edit Copy Delete	1	Juliana	23
<input type="checkbox"/> Edit Copy Delete	2	João	30
<input type="checkbox"/> Edit Copy Delete	3	Maria	42
<input type="checkbox"/> Edit Copy Delete	4	Pedro	19



## Gerenciamento de DB

Para **remover um banco de dados** em SQL, usamos o comando **DROP DATABASE**. Esse comando exclui completamente o banco de dados, incluindo todas as tabelas e dados armazenados nele. **Atenção:** esse processo é irreversível, ou seja, todos os dados serão perdidos.

### Comando para Remover um Banco de Dados

**Comando:**

```
DROP DATABASE nomedb;
```

- 

**Exemplo:**

```
DROP DATABASE empresa_db;
```

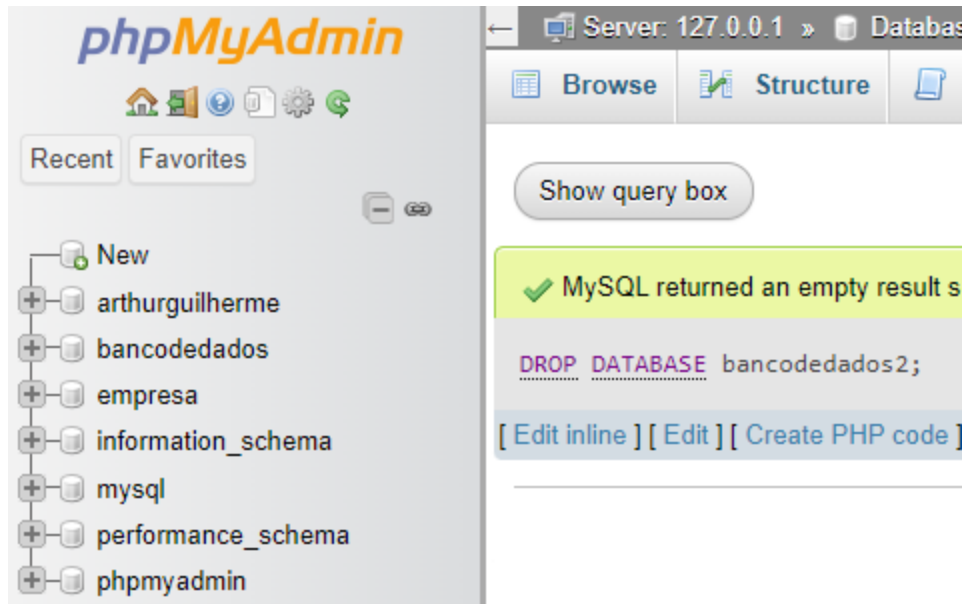
- 

Nesse exemplo, o banco de dados **empresa\_db** será completamente removido do sistema. Após a execução desse comando, não será possível recuperar as informações a menos que haja um backup.

### Resumo

Comando	Função
<b>DROP DATABASE</b> nomedb	Remove um banco de dados e todos os seus dados

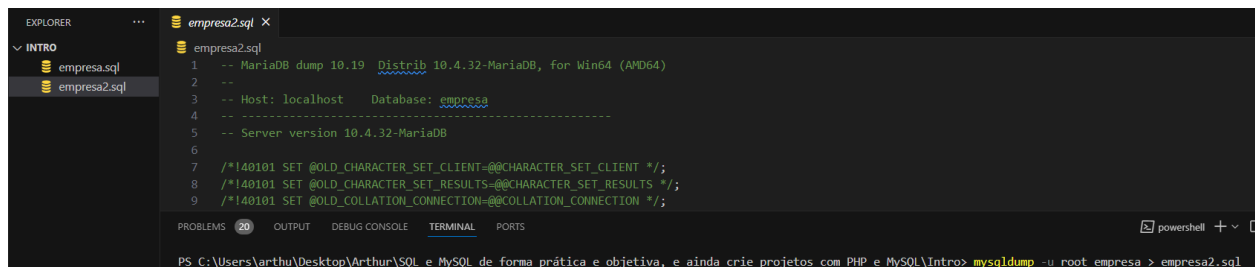
Sempre tenha cuidado ao usar esse comando, especialmente em sistemas de produção!



## Exportar DB

Comando

`mysqldump -u root tablename > filename.sql`



Vai gerar o arquivo cópia.

## Utilizar DB

Comando

*USE name*

DB que não está sendo utilizado

Comando

```
SELECT * FROM bd.table
```

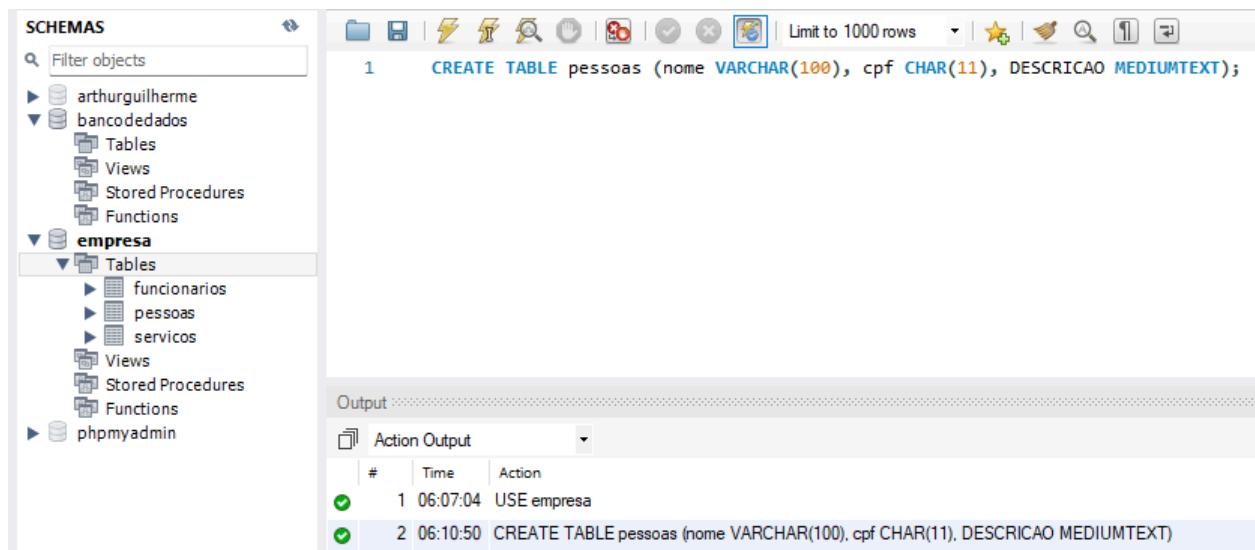
## Criação de Tabelas

### O que é uma tabela?

Entidade responsável por guardar os dados para consulta posterior. Possui colunas (categorias) com tipos determinados e atributos.

### \*Importante

- Se num for preciso realizar uma soma, o número pode ser string. Por exemplo: nº de casa, telefone .



## Criando uma tabela

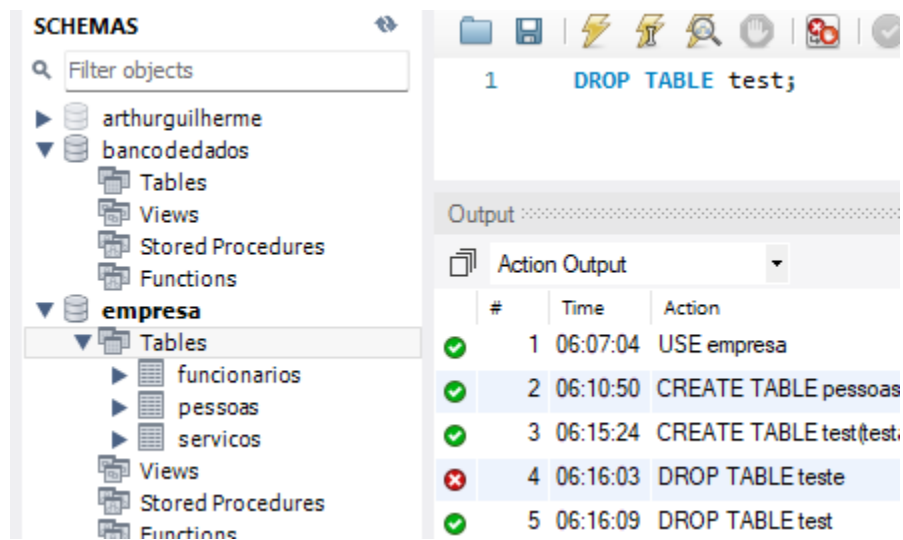
Comando

*CREATE TABLE tablename*

## Removendo Tabela

Comando

*DROP TABLE name*



## Tipos de Dados

Dividem-se em: texto, data, numérico e espacial. Muito importante para criação do DB, permite limites.

## Tipos de Texto

CHAR(x)

0 a 255 caracteres.

VARCHAR(x)

0 A 65535 CARACTERES.

TINYTEXT

0 a 255 caracteres.

MEDIUMTEXT

0 até 16777215.

CHAR e VARCHAR aceitam números e caracteres especiais.

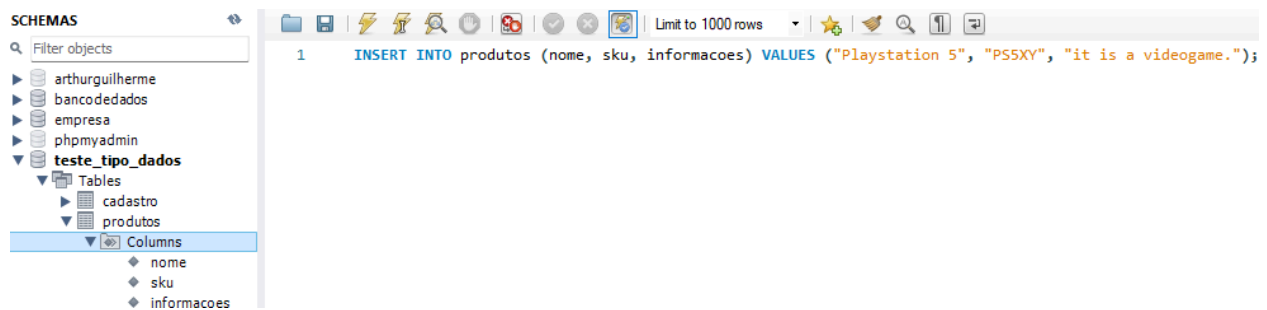
## Inserindo Dados

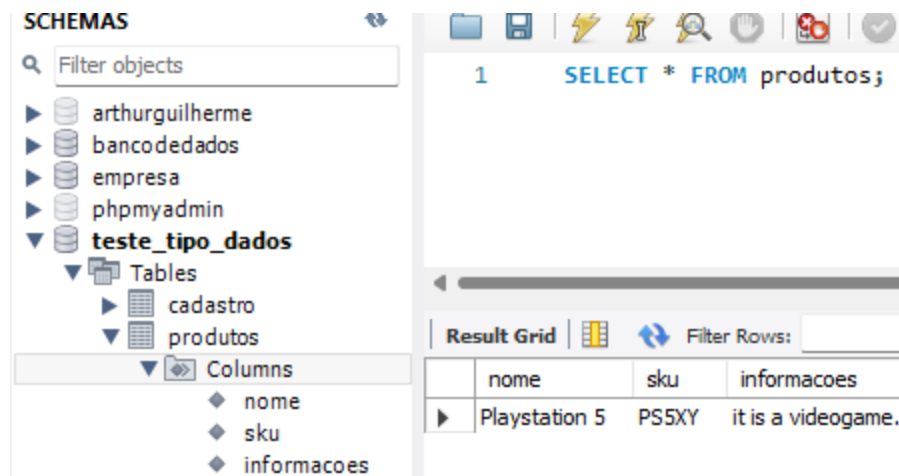
Saiba em qual banco os comandos irão funcionar, depois insira os dados.

Comando

*INSERT INTO tablename (col, col, col) VALUES (valueCOL, valueCOL, valueCOL)*

*SELECT \* FROM produtos;*





## Tipos Numéricos

BIT(x)

1 a 64 caracteres.

TINYINT(X)

1 a 255 caracteres.

BOOL

0 é FALSE e 1 é TRUE

INT(x)

entre -2147483648 a 2147483647

## Criando Tabelas com Tipos Numéricos

```
1 INSERT INTO servidores (nome, espaco_disco, ligado) VALUES ('Servidor 1', 123456, 0);
```

## Tipos de Data

### DATA

Formato yy-mm-dd

### DATETIME

Data com horário no formato YYYY-MM-DD hh:mm:ss

### TIMESTAMP

Aceita data no formato DATETIME, mas apenas entre os anos 1970 e 2038.

\*Observação

**Sempre coloque entre aspas!**

## Criando Tabela com Datas



## Porque Escolher o Tipo de Dado?

Sempre escolha o mais próximo da necessidade e limitar tamanho quando for possível. Isso otimiza o banco e economiza espaço em disco.

## Alteração de Tabelas

Sempre começa com “ALTER TABLE tableName”

### Comandos

ALTER TABLE table ADD COLUMN columnName dataType

ALTER TABLE table DROP COLUMN columnName

ALTER TABLE table MODIFY COLUMN columnName dataType

### Adição de Coluna

```
1 ALTER TABLE funcionarios ADD COLUMN profissao VARCHAR(100);
```

### Remoção de Coluna

```
1 ALTER TABLE funcionarios DROP COLUMN profissao;
```

### Modificar Tipo de Coluna

```
1 ALTER TABLE funcionarios MODIFY COLUMN data_nascimento DATE;
```

## Queries do CRUD

### O que é CRUD?

Ações utilizadas em **todas** as aplicações.

#### Create

Criar, inserir dados. **INSERT**



Read

ler dados. **SELECT**

Update

Atualizar Dados. **UPDATE**

Delete

Deletar, remover dados. **DROP**

Toda Aplicação Web com banco de dados tem pelo menos uma destas operações ou todas elas.

## Selecionar Todos os Dados

Comando

`SELECT * FROM tableName`



Showing rows 0 - 3 (4 total, Query took 0.0001 seconds.)

```
SELECT * FROM pessoas;
```

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [Create PHP](#) ]

☐ Show all | Number of rows: 25

[Extra options](#)

nome	rg	cpf	limite
Arthur Guilherme	1234567	14187714731	2700
Thávea Santos	112112	12277777777	1200
Alexander Junior	321123	87441256325	800
Léa Silva	1012202	32112332112	700

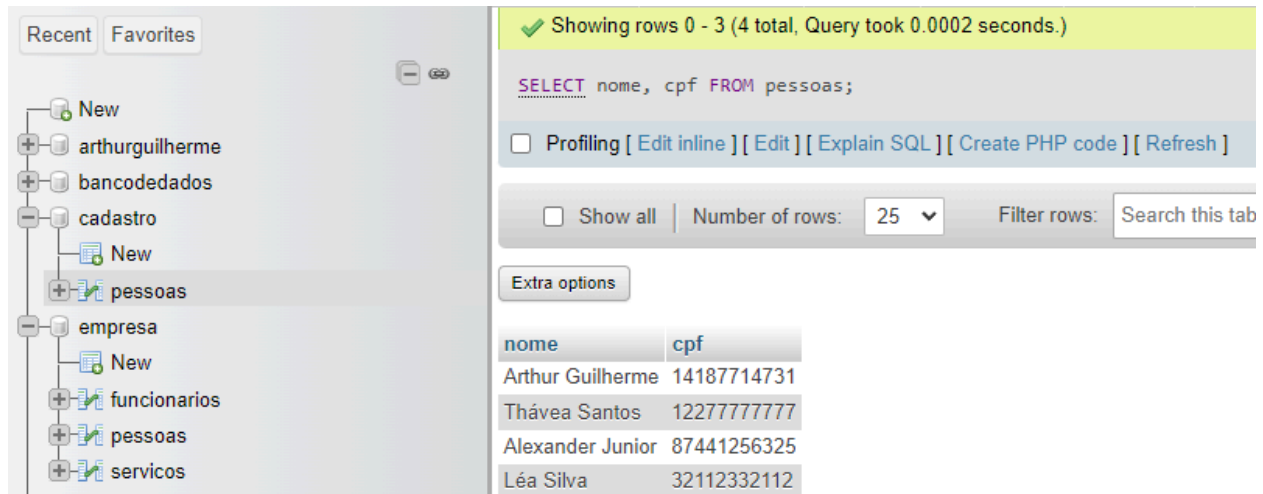
É interessante aprender fazer **SELECTs** mais precisos, para não precisar carregar todos os dados de uma vez.

## Selecionar Colunas Específicas

Isso é um **filtro** para **otimizar** consultas.

### Comando

SELECT coluna1, coluna2 FROM tableName



The screenshot shows a database management interface. On the left is a tree view of the database structure. The main panel displays a SQL query: `SELECT nome, cpf FROM pessoas;`. Below the query are options for Profiling, Edit inline, Edit, Explain SQL, Create PHP code, and Refresh. There are also controls for showing all rows, a dropdown for the number of rows (set to 25), and a filter rows search box. Below these is an 'Extra options' button. The results are shown in a table with two columns: 'nome' and 'cpf'.

nome	cpf
Arthur Guilherme	14187714731
Thávea Santos	12277777777
Alexander Junior	87441256325
Léa Silva	32112332112

## Inserindo Dados na Tabela

Criar dados.

### Comando

INSERT INTO tableName (colunas) VALUES (valores)

```
1 INSERT INTO pessoas VALUES ("Nathan Silva", "7412587", "78965412398", 2200);
```

### Atalho

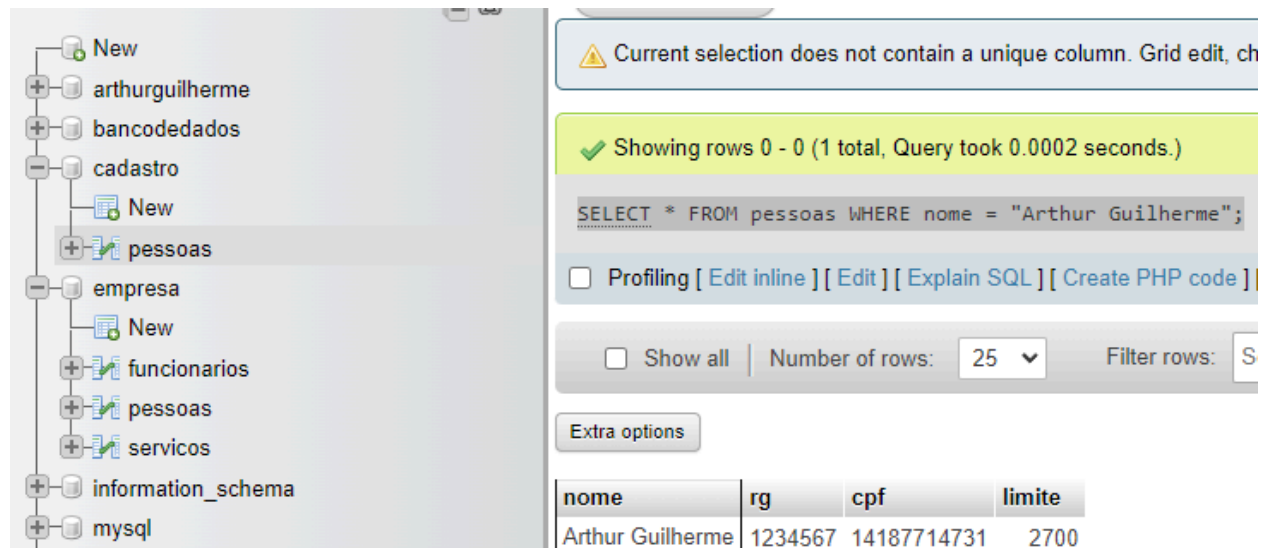
SQL aceita que sejam **omitidas as colunas**, desde que se preencha todas elas ao inserir dados.

## A Impôrtancia do WHERE

Determina **quais registros serão atualizados**.

## Regra

Sempre utilizá-lo quando **Atualizar** ou **Remover** dados.



The screenshot shows a database management tool interface. On the left is a tree view of the database structure. The 'pessoas' table is selected under the 'cadastro' schema. On the right, a query execution panel displays a warning at the top: 'Current selection does not contain a unique column. Grid edit, ch'. Below this, a green status bar indicates 'Showing rows 0 - 0 (1 total, Query took 0.0002 seconds.)'. The SQL query is 'SELECT \* FROM pessoas WHERE nome = "Arthur Guilherme";'. There are links for 'Profiling', 'Edit inline', 'Edit', 'Explain SQL', and 'Create PHP code'. Below these are controls for 'Show all', 'Number of rows' (set to 25), and 'Filter rows'. An 'Extra options' button is also present. At the bottom, a table shows the query results.

nome	rg	cpf	limite
Arthur Guilherme	1234567	14187714731	2700

## Atualizando Dados

Atualize dados de uma ou mais colunas.

### Comando

UPDATE tableName SET coluna=valor WHERE condition

## Tabela

```
SELECT * FROM `pessoas`;
```

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [Create](#) ]

☐ Show all | Number of rows: 25 ▼

Extra options

nome	rg	cpf	limite
Arthur Guilherme	1234567	14187714731	0
Thávea Santos	112112	12277777777	0
Alexander Junior	321123	87441256325	0
Léa Silva	1012202	32112332112	0
Ana Cláudia	1234567	1234567898	0
Hosana Santos	444444	99999999999	0
Nathan Silva	7412587	78965412398	0

## Update Uma Coluna

```
1 UPDATE pessoas SET limite = 10000 WHERE nome = "Arthur Guilherme";
```

## Tabela

```
SELECT * FROM pessoas;
```

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [Create](#) ]

☐ Show all | Number of rows: 25 ▼

Extra options

nome	rg	cpf	limite
Arthur Guilherme	1234567	14187714731	10000
Thávea Santos	112112	12277777777	0
Alexander Junior	321123	87441256325	0
Léa Silva	1012202	32112332112	0
Ana Cláudia	1234567	1234567898	0
Hosana Santos	444444	99999999999	0
Nathan Silva	7412587	78965412398	0

## Update Duas Colunas

```
UPDATE pessoas SET rg = "7777777", nome = "Arthur Silva" WHERE nome = "Arthur Guilherme";
```

## Tabela

```
SELECT * FROM pessoas;
```

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [Cancel](#) ]

☐ Show all | Number of rows: 25 ▼

Extra options

nome	rg	cpf	limite
Arthur Silva	7777777	14187714731	10000
Thávea Santos	112112	12277777777	0
Alexander Junior	321123	87441256325	0
Léa Silva	1012202	32112332112	0
Ana Cláudia	1234567	1234567898	0
Hosana Santos	444444	99999999999	0
Nathan Silva	7412587	78965412398	0

## Deletando Dados

### Comando

DELETE FROM tableName WHERE condição;

```
1 DELETE FROM pessoas WHERE nome = "Hosana Santos";
```

```
SELECT * FROM pessoas;
```

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [C](#) ]

☐ Show all | Number of rows: 25 ▼

Extra options

nome	rg	cpf	limite
Arthur Silva	7777777	14187714731	10000
Thávea Santos	112112	12277777777	0
Alexander Junior	321123	87441256325	0
Léa Silva	1012202	32112332112	1234
Ana Cláudia	1234567	1234567898	0
Nathan Silva	7412587	78965412398	0

## \*\*ATENÇÃO

Caso num utilize WHERE ,**tudo** será deletado!

*Sem WHERE*

```
1 DELETE FROM pessoas;
```


Tabela Vazia:

```
SELECT * FROM pessoas;
```

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ]

nome	rg	cpf	limite
------	----	-----	--------

Query results operations

 Create view

## Como Saber o que Deletar?

Selecione antes para conferir:

```
SELECT * FROM pessoas WHERE nome = "Thávea Santos";
```

☐ Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ]

☐ Show all | Number of rows: 25 ▼ Filter rows

Extra options

nome	rg	cpf	limite
Thávea Santos	112112	12277777777	0

Agora delete:

```
1 DELETE FROM pessoas WHERE nome = "Thávea Santos";
```

```
SELECT * FROM pessoas;
```

☐ Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ]

☐ Show all | Number of rows: 25 ▼ Filter rows

Extra options

nome	rg	cpf	limite
Alexander Junior	321123	87441256325	0
Ana Cláudia	1234567	1234567898	0
Nathan Silva	7412587	78965412398	0

## Avançando em SELECT

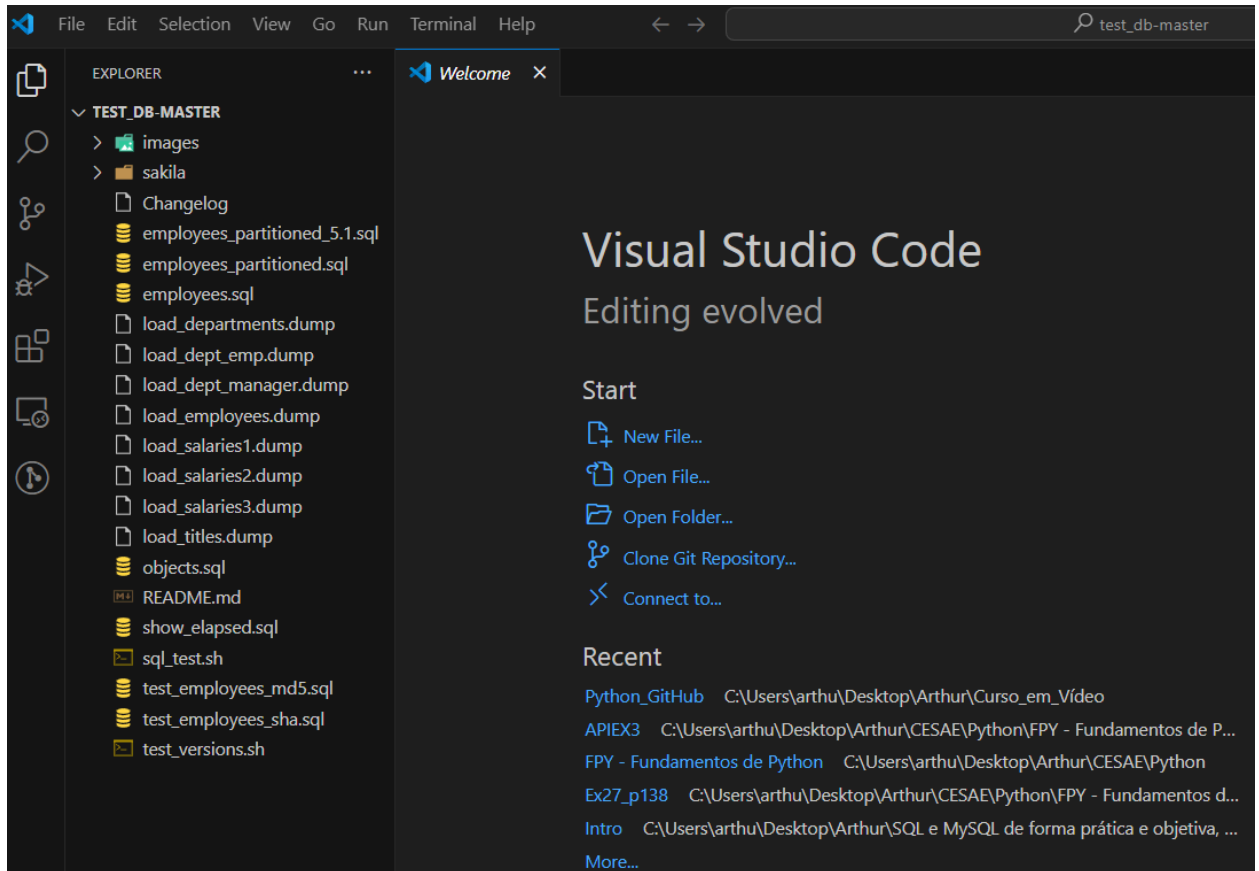
### Instalando um BD

[https://github.com/datacharmer/test\\_db](https://github.com/datacharmer/test_db)



## VSCODE

### 1. Abrir Pasta;

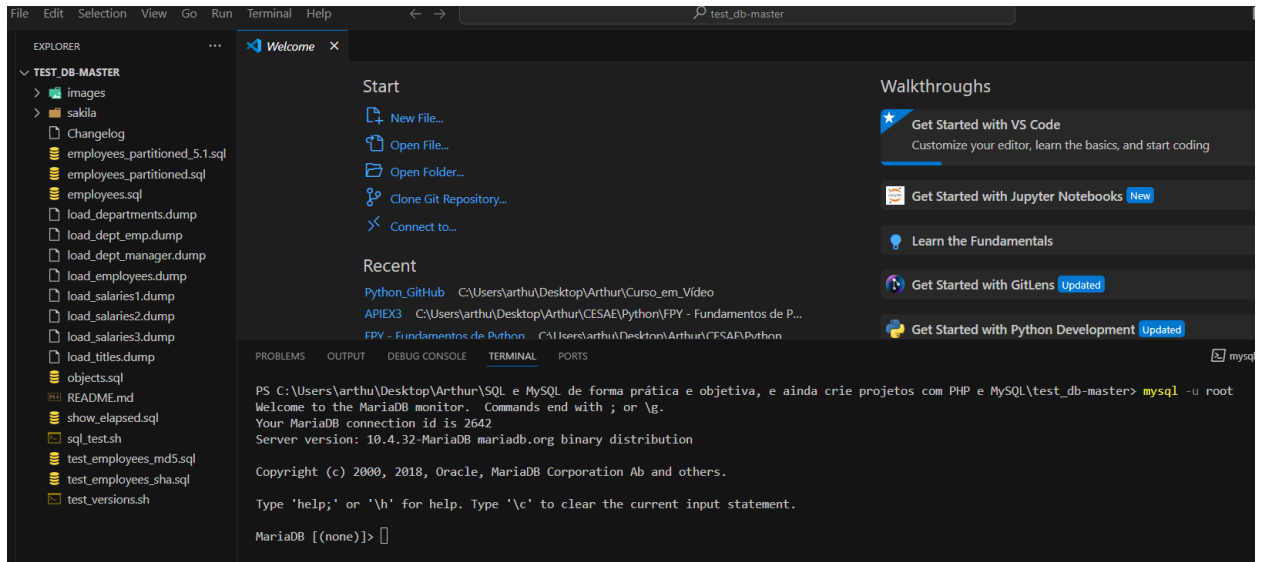


### 2. Acessar Arquivo:

**mysql -u root**

**ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)**

**mysql -u root -p**



### 3. Criar DB

**CREATE DATABASE employees;**

```
PS C:\Users\arthu\Desktop\Arthur\SQL e MySQL de forma prática e objetiva, e ainda crie projetos com PHP e MySQL\test_db-master> mysql -u root
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 2645
Server version: 10.4.32-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE DATABASE employees;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]>
```

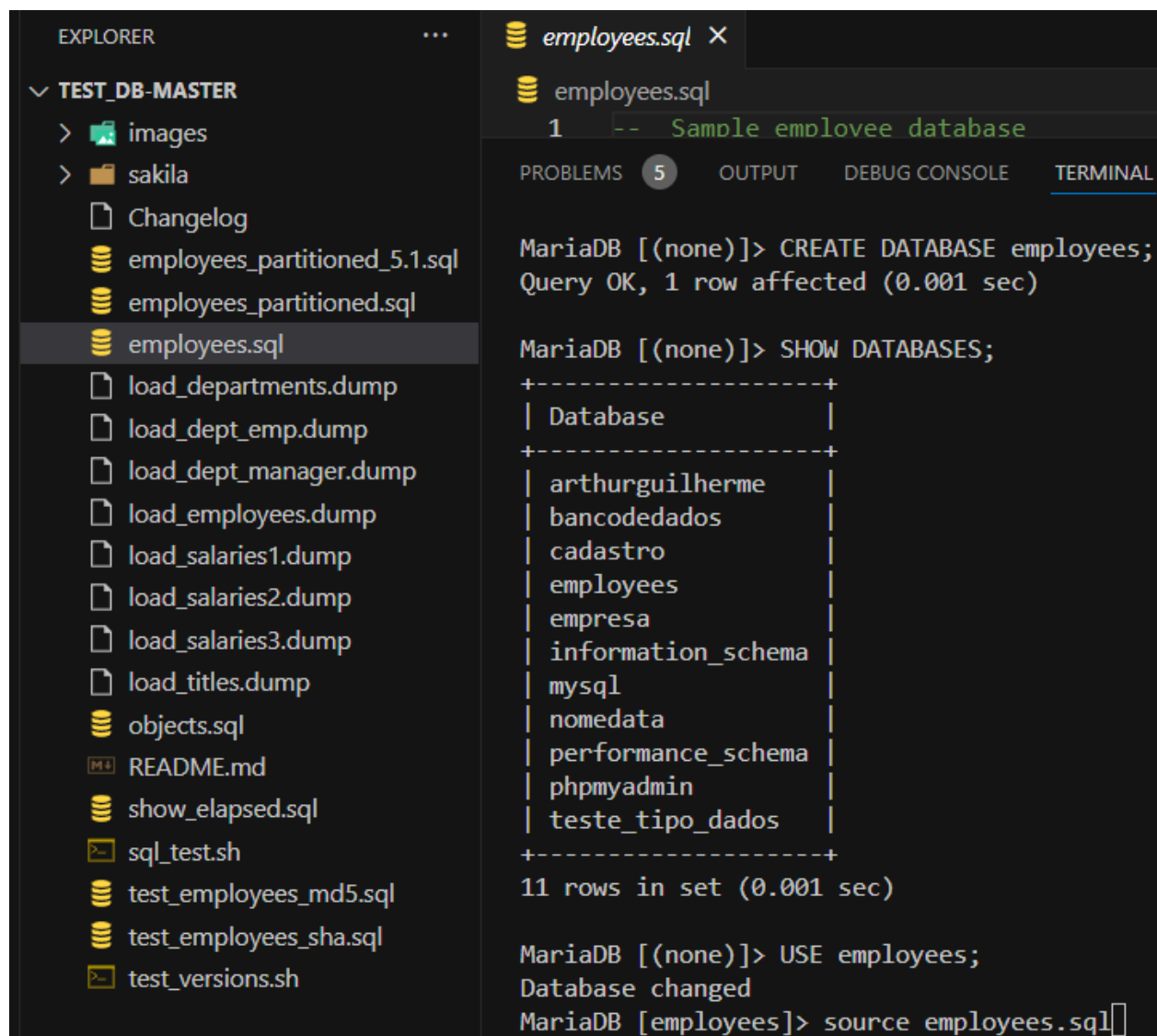
### 4. Confira;

**SHOW DATABASES;**

```
MariaDB [(none)]> CREATE DATABASE employees;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> SHOW DATABASES;
+-----+
| Database |
+-----+
| arthurguilherme |
| bancodedados   |
| cadastro        |
| employees       |
| empresa         |
| information_schema |
| mysql           |
| nomedata        |
| performance_schema |
| phpmyadmin      |
| teste_tipo_dados |
+-----+
11 rows in set (0.001 sec)
```

5. Garanta o Uso do Banco antes de Importar e Importe;  
USE employees;  
source employees.sql;



The screenshot displays a code editor interface with a file explorer on the left and a terminal window on the right.

**File Explorer (Left):**

- TEST\_DB-MASTER
  - images
  - sakila
    - Changelog
    - employees\_partitioned\_5.1.sql
    - employees\_partitioned.sql
    - employees.sql**
    - load\_departments.dump
    - load\_dept\_emp.dump
    - load\_dept\_manager.dump
    - load\_employees.dump
    - load\_salaries1.dump
    - load\_salaries2.dump
    - load\_salaries3.dump
    - load\_titles.dump
    - objects.sql
    - README.md
    - show\_elapsed.sql
    - sql\_test.sh
    - test\_employees\_md5.sql
    - test\_employees\_sha.sql
    - test\_versions.sh

**Terminal Window (Right):**

The terminal window shows the execution of SQL commands in MariaDB:

```
MariaDB [(none)]> CREATE DATABASE employees;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> SHOW DATABASES;
+-----+
| Database |
+-----+
| arthurguilherme |
| bancodedados |
| cadastro |
| employees |
| empresa |
| information_schema |
| mysql |
| nomedata |
| performance_schema |
| phpmyadmin |
| teste_tipo_dados |
+-----+
11 rows in set (0.001 sec)

MariaDB [(none)]> USE employees;
Database changed
MariaDB [employees]> source employees.sql
```

Lembre-se que tudo está sendo executado no **Terminal**, já dentro do local que tem o DB que será importado para o DB a pouco criado.

Query OK, 24918 rows affected (0.215 sec)  
Records: 24918 Duplicates: 0 Warnings: 0

Query OK, 24921 rows affected (0.151 sec)  
Records: 24921 Duplicates: 0 Warnings: 0

Query OK, 24922 rows affected (0.149 sec)  
Records: 24922 Duplicates: 0 Warnings: 0

Query OK, 24918 rows affected (0.165 sec)  
Records: 24918 Duplicates: 0 Warnings: 0

Query OK, 24923 rows affected (0.142 sec)  
Records: 24923 Duplicates: 0 Warnings: 0

Query OK, 24924 rows affected (0.171 sec)  
Records: 24924 Duplicates: 0 Warnings: 0

Query OK, 24920 rows affected (0.231 sec)  
Records: 24920 Duplicates: 0 Warnings: 0

Query OK, 24920 rows affected (0.154 sec)  
Records: 24920 Duplicates: 0 Warnings: 0

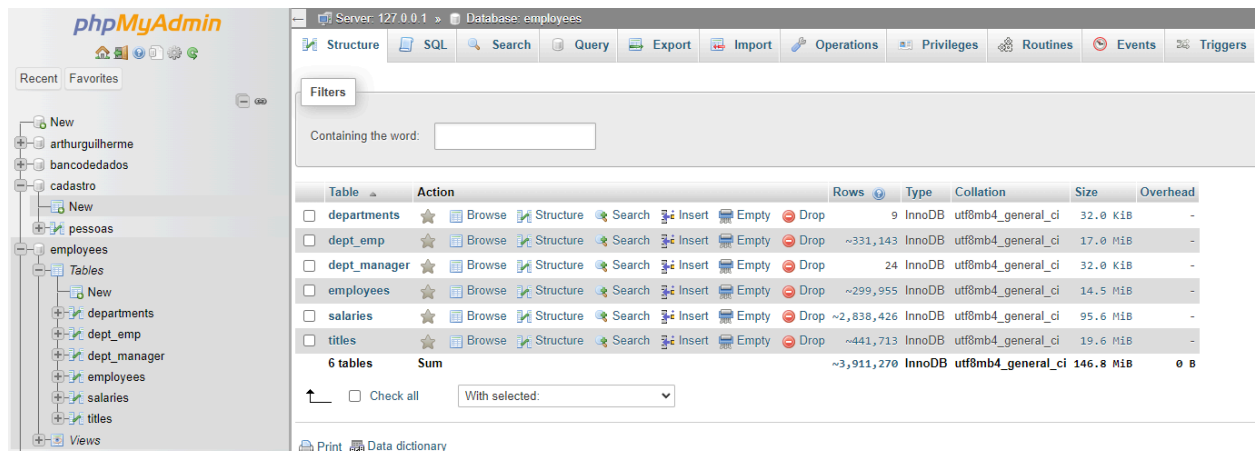
Query OK, 24919 rows affected (0.189 sec)  
Records: 24919 Duplicates: 0 Warnings: 0

Query OK, 7671 rows affected (0.052 sec)  
Records: 7671 Duplicates: 0 Warnings: 0

```
+-----+
| data_load_time_diff |
+-----+
| 00:00:30           |
+-----+
1 row in set (0.001 sec)
```

MariaDB [employees]>

6. Confira;



## Solução para Erro de Importação

### A importância do SELECT

A maioria das **queries** em um DB são de consulta. Esse comando é o que tem **mais variações**, dessa forma cria-se **filtros avançados**.

### \*\*\*\*Operadores do SQL

#### Comparação

Igual a Python.

#### BETWEEN

Selecione em um **intervalo**.

#### LIKE

Literalmente **“COMO”**, irá buscar **por meio** de algum padrão.

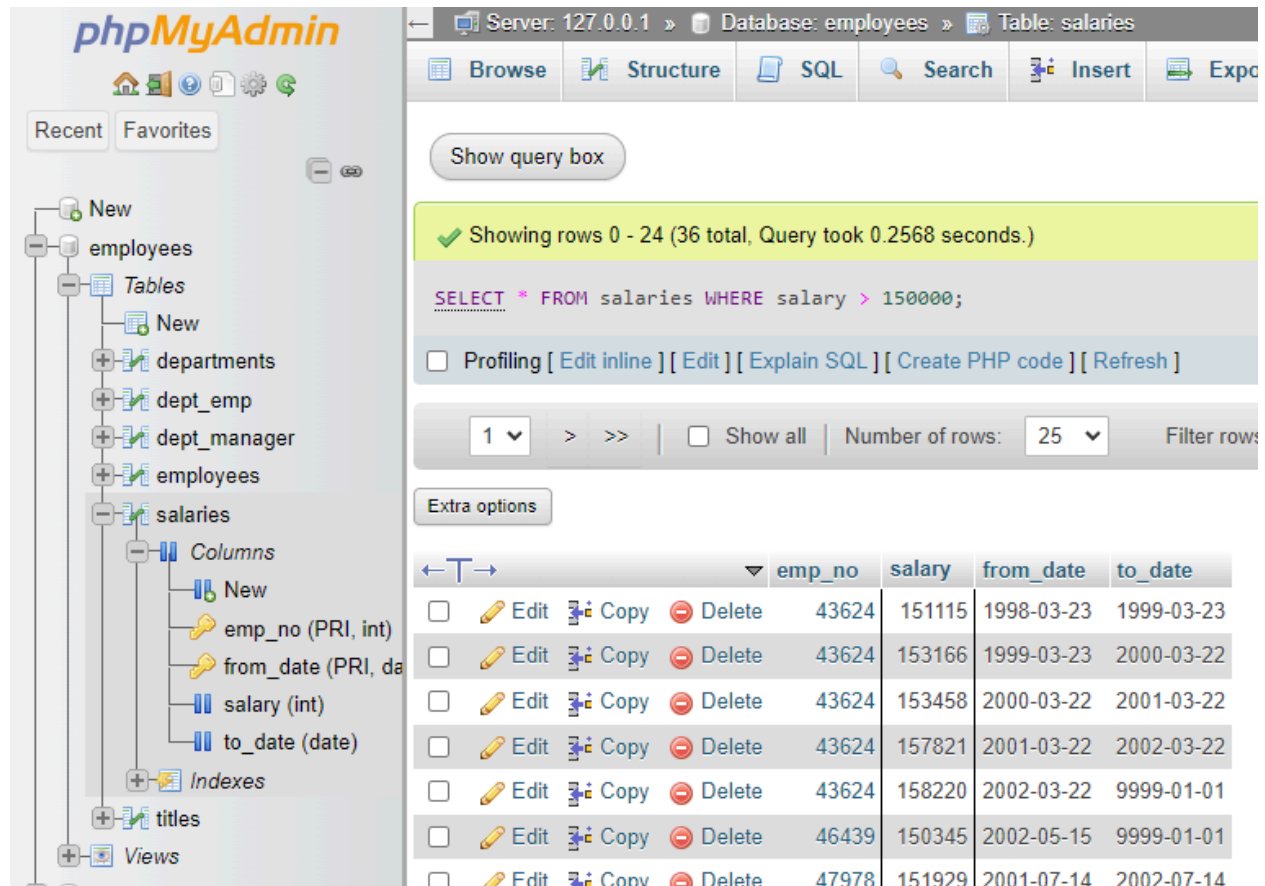
IN

Seleção entre um conjunto de **valores específicos**.

## Cláusula WHERE

Veja a combinação do **WHERE** com os operadores.

Conforme tabelas, a consulta se faz dessa forma:



The screenshot shows the phpMyAdmin interface for a database named 'employees'. The left sidebar shows the database structure with tables: departments, dept\_emp, dept\_manager, employees, salaries, titles, and views. The 'salaries' table is selected, and its columns are listed: emp\_no (PRI, int), from\_date (PRI, date), salary (int), and to\_date (date). The main panel shows a SQL query: `SELECT * FROM salaries WHERE salary > 150000;`. The query results are displayed in a table with 4 columns: emp\_no, salary, from\_date, and to\_date. The results show 7 rows of data.

emp_no	salary	from_date	to_date
43624	151115	1998-03-23	1999-03-23
43624	153166	1999-03-23	2000-03-22
43624	153458	2000-03-22	2001-03-22
43624	157821	2001-03-22	2002-03-22
43624	158220	2002-03-22	9999-01-01
46439	150345	2002-05-15	9999-01-01
47978	151929	2001-07-14	2002-07-14

## Utilizando DISTINCT

Serve para mostrar as variações de valores.

comando

`SELECT DISTINCT column FROM tableName`

```
SELECT DISTINCT gender FROM employees;
```




☐ Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]




☐ Show all | Number of rows: 25 ▾ Filter rows:

Extra options

← T →

gender

☐  Edit  Copy  Delete M

☐  Edit  Copy  Delete F

```
SELECT DISTINCT title FROM titles;
```




☐ Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]




☐ Show all | Number of rows: 25 ▾ Filter rows:




Extra options




← T →




title




☐  Edit  Copy  Delete Senior Engineer




☐  Edit  Copy  Delete Staff

☐  Edit  Copy  Delete Engineer

☐  Edit  Copy  Delete Senior Staff

☐  Edit  Copy  Delete Assistant Engineer

☐  Edit  Copy  Delete Technique Leader

☐  Edit  Copy  Delete Manager

## Operadores Lógicos

### AND

Usa **duas condições** para retornar a consulta, apenas se ambas forem **verdadeiras**.



OR

Recebe **duas condições**, retorna se pelo menos **uma** for verdadeira.

NOT

**Inverte** a lógica.

## Utilizando o AND

Utiliza um **filtro duplo**.

```
SELECT * FROM salaries WHERE salary > 155000 AND from_date > "1991-12-12" AND emp_no > 10800.
```

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [Create PHP code](#) ] [ [Refresh](#) ]

☐ Show all | Number of rows: 25 ▼ Filter rows:  Sort by key: N

Extra options

	emp_no	salary	from_date	to_date
<input type="checkbox"/> Edit  Copy  Delete	43624	157821	2001-03-22	2002-03-22
<input type="checkbox"/> Edit  Copy  Delete	43624	158220	2002-03-22	9999-01-01
<input type="checkbox"/> Edit  Copy  Delete	47978	155709	2002-07-14	9999-01-01
<input type="checkbox"/> Edit  Copy  Delete	109334	155377	2000-02-12	2001-02-11
<input type="checkbox"/> Edit  Copy  Delete	109334	155190	2002-02-11	9999-01-01
<input type="checkbox"/> Edit  Copy  Delete	253939	155513	2002-04-11	9999-01-01
<input type="checkbox"/> Edit  Copy  Delete	254466	156286	2001-08-04	9999-01-01

## Utilizando o OR

Qualquer condição, se for verdadeira, o resultado será retornado. Não há limite de condições para uma query.

Veja esta, com 3 possibilidades:

Showing rows 0 - 24 (363656 total, Query took 0.0009 seconds.) [emp\_no: 499999... - 499983...]

```
SELECT * FROM titles WHERE title = "Senior Engineer" || title = "Staff" || emp_no > 125000 ORDER BY emp_no DESC;
```

☐ Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

1 > >> | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

	emp_no	1	title	from_date	to_date
<input type="checkbox"/> Edit Copy Delete	499999		Engineer	1997-11-30	9999-01-01
<input type="checkbox"/> Edit Copy Delete	499998		Staff	1993-12-27	1998-12-27
<input type="checkbox"/> Edit Copy Delete	499998		Senior Staff	1998-12-27	9999-01-01
<input type="checkbox"/> Edit Copy Delete	499997		Senior Engineer	1992-08-29	9999-01-01
<input type="checkbox"/> Edit Copy Delete	499997		Engineer	1987-08-30	1992-08-29

## Utilizando o NOT

Inverter uma cláusula. Leia-se “**menos os que (cláusula)**”.

\*Cuidado

Atente-se a **ordem** da sentença:

```
SELECT * FROM titles WHERE NOT title = "Staff";
```

☐ Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

1 > >> | Number of rows: 25 | Filter rows: Search this table

Extra options

	emp_no	title	from_date	to_date
<input type="checkbox"/> Edit Copy Delete	10001	Senior Engineer	1986-06-26	9999-01-01
<input type="checkbox"/> Edit Copy Delete	10003	Senior Engineer	1995-12-03	9999-01-01
<input type="checkbox"/> Edit Copy Delete	10004	Engineer	1986-12-01	1995-12-01

```
SELECT * FROM titles WHERE NOT title = "Staff" AND NOT title = "Senior Engineer" AND NOT title = "Senior Staff";
```

☐ Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

1 > >> | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

				emp_no	title	from_date	to_date
<input type="checkbox"/>	Edit	Copy	Delete	10004	Engineer	1986-12-01	1995-12-01
<input type="checkbox"/>	Edit	Copy	Delete	10008	Assistant Engineer	1998-03-11	2000-07-31
<input type="checkbox"/>	Edit	Copy	Delete	10009	Assistant Engineer	1985-02-18	1990-02-18
<input type="checkbox"/>	Edit	Copy	Delete	10009	Engineer	1990-02-18	1995-02-18

## Utilizando ORDER BY

Basicamente, **ordenação de resultados**. Pode ser **ASC** ou **DESC**.

Comando

SELECT \* FROM table ORDER BY column ASC/DESC

\*Cuidado

Sempre utilizar **após WHERE** e **antes de alguma coluna**.

```
SELECT * FROM salaries ORDER BY emp_no ASC;
```

☐ Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

1

>

>>












Number of rows:

25

Filter rows:

Search this

Extra options

				emp_no	salary	from_date	to_date
<input type="checkbox"/>		Edit	 Copy  Delete	10001	60117	1986-06-26	1987-06-26
<input type="checkbox"/>		Edit	 Copy  Delete	10001	62102	1987-06-26	1988-06-25
<input type="checkbox"/>		Edit	 Copy  Delete	10001	66074	1988-06-25	1989-06-25
<input type="checkbox"/>		Edit	 Copy  Delete	10001	66596	1989-06-25	1990-06-25
<input type="checkbox"/>		Edit	 Copy  Delete	10001	66961	1990-06-25	1991-06-25

```
SELECT * FROM salaries ORDER BY emp_no DESC;
```

☐ Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

1

>

>>

Number of rows:

25

Filter rows:

Search this

Extra options

<div>← T →</div>				emp_no	1	salary	from_date	to_date
<input type="checkbox"/>		Edit	 Copy	 Delete	499999	77303	2001-11-29	9999-01-01
<input type="checkbox"/>		Edit	 Copy	 Delete	499999	74327	2000-11-29	2001-11-29
<input type="checkbox"/>		Edit	 Copy	 Delete	499999	70745	1999-11-30	2000-11-29
<input type="checkbox"/>		Edit	 Copy	 Delete	499999	67043	1998-11-30	1999-11-30
<input type="checkbox"/>		Edit	 Copy	 Delete	499999	63707	1997-11-30	1998-11-30
<input type="checkbox"/>		Edit	 Copy	 Delete	499998	55003	2001-12-25	9999-01-01

## Utilizando a LIMIT

Limite a consulta para tornar mais rápida.

## Comando

SELECT \* FROM tableName LIMIT nº

SELECT \* FROM salaries LIMIT 10;

☐ Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

Extra options

</

\*Cuidado

**LIMIT Nº sempre no fim**

```
SELECT * FROM salaries ORDER BY salary DESC LIMIT 10;
```

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [Create PHP code](#) ] [ [Refresh](#) ]

Extra options

<div>← T →</div>				emp_no	salary	from_date	to_date	
<input type="checkbox"/>		<a href="#">Edit</a>	 <a href="#">Copy</a>	 <a href="#">Delete</a>	43624	158220	2002-03-22	9999-01-01
<input type="checkbox"/>		<a href="#">Edit</a>	 <a href="#">Copy</a>	 <a href="#">Delete</a>	43624	157821	2001-03-22	2002-03-22
<input type="checkbox"/>		<a href="#">Edit</a>	 <a href="#">Copy</a>	 <a href="#">Delete</a>	254466	156286	2001-08-04	9999-01-01
<input type="checkbox"/>		<a href="#">Edit</a>	 <a href="#">Copy</a>	 <a href="#">Delete</a>	47978	155709	2002-07-14	9999-01-01
<input type="checkbox"/>		<a href="#">Edit</a>	 <a href="#">Copy</a>	 <a href="#">Delete</a>	253939	155513	2002-04-11	9999-01-01
<input type="checkbox"/>		<a href="#">Edit</a>	 <a href="#">Copy</a>	 <a href="#">Delete</a>	109334	155377	2000-02-12	2001-02-11
<input type="checkbox"/>		<a href="#">Edit</a>	 <a href="#">Copy</a>	 <a href="#">Delete</a>	109334	155190	2002-02-11	9999-01-01
<input type="checkbox"/>		<a href="#">Edit</a>	 <a href="#">Copy</a>	 <a href="#">Delete</a>	109334	154888	2001-02-11	2002-02-11
<input type="checkbox"/>		<a href="#">Edit</a>	 <a href="#">Copy</a>	 <a href="#">Delete</a>	109334	154885	1999-02-12	2000-02-12
<input type="checkbox"/>		<a href="#">Edit</a>	 <a href="#">Copy</a>	 <a href="#">Delete</a>	80823	154459	2002-02-22	9999-01-01

```
SELECT * FROM employees WHERE gender = "F" ORDER BY hire_date DESC LIMIT 15;
```

☐ Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

Extra options

						emp_no	birth_date	first_name	last_name	gender	hire_date	
<input type="checkbox"/>		Edit		Copy		Delete	499553	1954-05-06	Hideyuki	Delgrande	F	2000-01-22
<input type="checkbox"/>		Edit		Copy		Delete	222965	1959-08-07	Volkmar	Perko	F	2000-01-13
<input type="checkbox"/>		Edit		Copy		Delete	422990	1953-04-09	Jaana	Verspoor	F	2000-01-11
<input type="checkbox"/>		Edit		Copy		Delete	205048	1960-09-12	Ennio	Alblas	F	2000-01-06
<input type="checkbox"/>		Edit		Copy		Delete	226633	1958-06-10	Xuejun	Benzmuller	F	2000-01-04
<input type="checkbox"/>		Edit		Copy		Delete	72329	1953-02-09	Randi	Luit	F	2000-01-02
<input type="checkbox"/>		Edit		Copy		Delete	60134	1964-04-21	Seshu	Rathonyi	F	2000-01-02
<input type="checkbox"/>		Edit		Copy		Delete	13246	1952-06-09	Adil	Siepmann	F	1999-12-31
<input type="checkbox"/>		Edit		Copy		Delete	71159	1952-07-19	Manton	Ghemri	F	1999-12-30
<input type="checkbox"/>		Edit		Copy		Delete	242381	1952-05-14	Garnik	Kolvik	F	1999-12-30
<input type="checkbox"/>		Edit		Copy		Delete	220951	1957-10-14	Mang	Kohling	F	1999-12-28
<input type="checkbox"/>		Edit		Copy		Delete	243702	1961-10-04	Munehiro	Luke	F	1999-12-24
<input type="checkbox"/>		Edit		Copy		Delete	90996	1960-07-03	Dannz	Kamble	F	1999-12-18
<input type="checkbox"/>		Edit		Copy		Delete	292805	1963-08-13	Jacqueline	Hoogerwoord	F	1999-12-16
<input type="checkbox"/>		Edit		Copy		Delete	441722	1958-12-01	Gonzalo	Aamodt	F	1999-12-15

## Funções SQL




Blocos de códigos reaproveitáveis para extrair resultados que demandam muita programação. Basicamente simplificam as consultas.

## Sem Função

```
SELECT salary FROM salaries ORDER BY salary ASC LIMIT 1;
```

☐ Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

Extra options

<div>← T →</div>				salary
<input type="checkbox"/>	 Edit	 Copy	 Delete	38623

## Com Função

```
SELECT MIN(salary) FROM salaries;
```

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [Create PHP code](#) ] [ [Refresh](#) ]

☐ Show all | Number of rows: 25 ▼ Filter rows:

[Extra options](#)

MIN(salary)
38623

## Função MIN

Retorna o **menor valor** de uma coluna.

### Comando

```
SELECT MIN(column) FROM tableName
```

```
SELECT MIN(salary) FROM salaries;
```

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [Create PHP code](#) ] [ [Refresh](#) ]

☐ Show all | Number of rows: 25 ▼ Filter rows:

[Extra options](#)

MIN(salary)
38623

## Função MAX

Retorna o **maior valor** de uma coluna.

### Comando

```
SELECT MAX(column) FROM tablename
```



The screenshot shows a SQL IDE interface. On the left, the 'SCHEMAS' pane lists databases like 'employees', 'sakila', 'sys', 'teste', 'testworkbench', and 'world'. The 'employees' database is expanded, showing tables like 'departments', 'dept\_emp', 'dept\_manager', 'employees', 'salaries', and 'titles'. The main editor window, titled 'SQL File 3\*', contains the following SQL query:

```
1 • SELECT MAX(salary) AS maior_salario FROM salaries;
```

Below the query editor, the 'Result Grid' shows the output of the query:

	maior_salario
▶	158220

## Função COUNT

Retorna número de valores que combinam com algum critério, ou seja, utilizado com **WHERE**. A retorno será exatamente quantas **linhas** aquela coluna retorna **true**.

### Comando

SELECT **COUNT**(\* ou column) FROM tableName **WHERE** condition

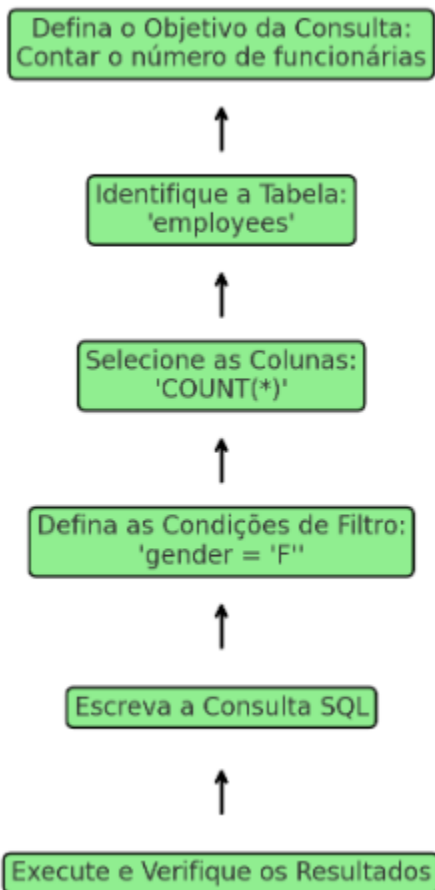
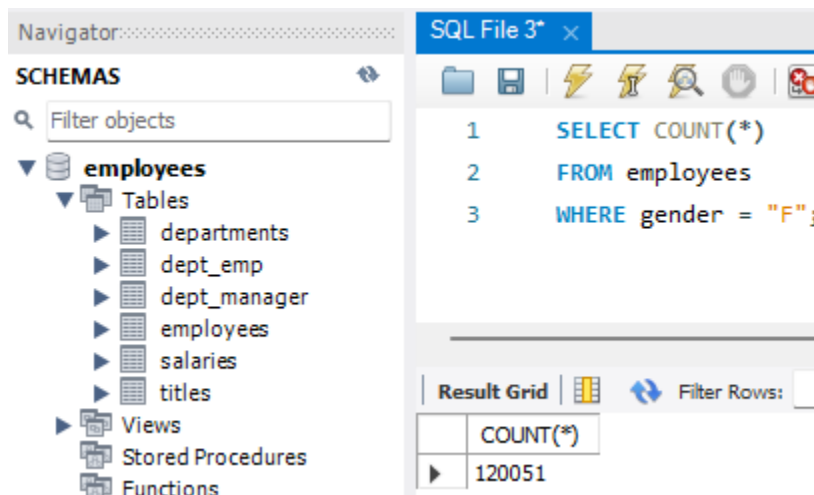
The screenshot shows the same SQL IDE interface. The main editor window contains the following SQL query:

```
1 SELECT COUNT(*) FROM salaries WHERE salary > 100000;
```

Below the query editor, the 'Result Grid' shows the output of the query:

	COUNT(*)
▶	94696

## Lógica da Consulta

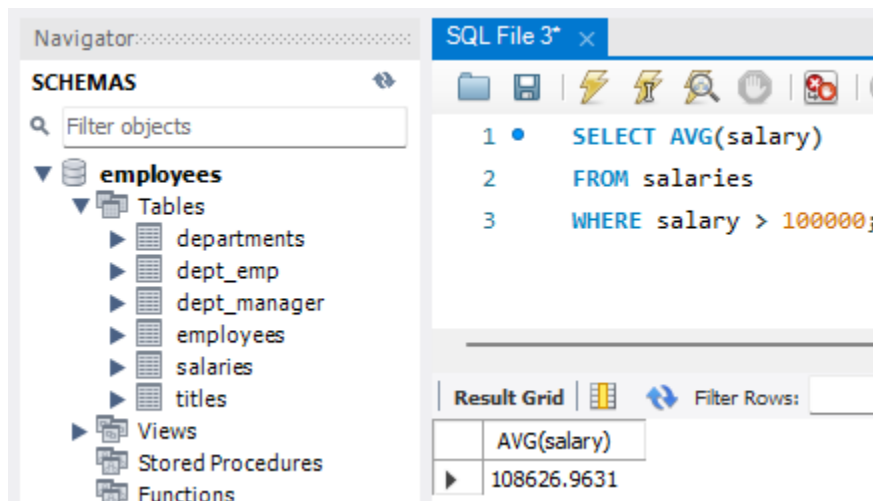
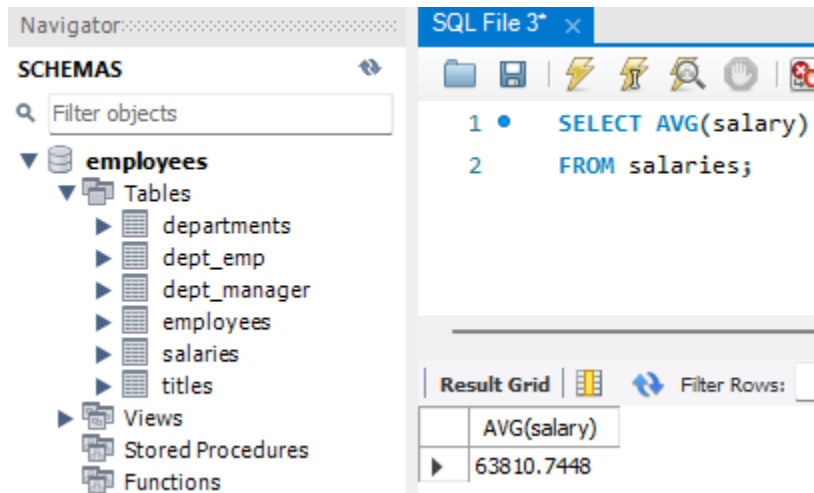


## FUNÇÃO AVG

Retorna **média** de uma coluna.

## Comando

SELECT **AVG**(columnName) FROM tableName;



## Cuidados

Utilize colunas com **números**, strings nem faz sentido.

## Função SUM

Retorna a soma dos **valores numéricos** de uma coluna.

## Comando

SELECT **SUM**(columnName) FROM tableName

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane is expanded to show the 'employees' database, with the 'salaries' table selected. The main pane displays a SQL query in 'SQL File 3\*':

```
1 • SELECT SUM(salary)
2 FROM salaries;
```

Below the query, the 'Result Grid' shows the following data:

SUM(salary)
181480757419

The screenshot shows the same SQL Server Enterprise Manager interface, but the query in 'SQL File 3\*' is now filtered:

```
1 • SELECT SUM(salary)
2 FROM salaries
3 WHERE salary > 100000;
```

The 'Result Grid' shows the following data:

SUM(salary)
10286538895

## Função LIKE

Retorna resultados semelhantes a algum **padrão**. Por lógica, vem **depois** do WHERE.

Uso do “%”

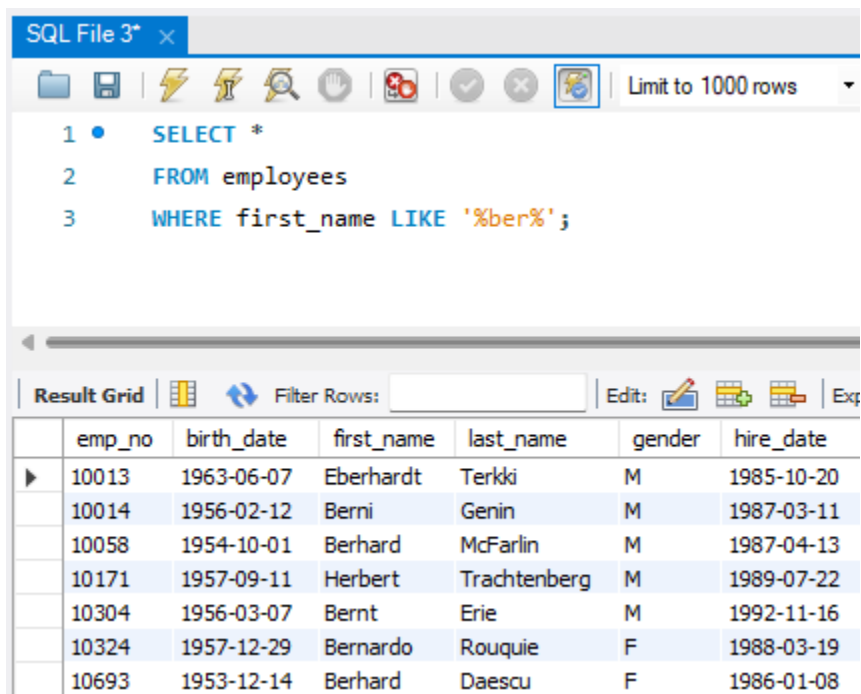
Não é obrigatório

Comando

SELECT \* FROM tableName WHERE columnName **LIKE** pattern.

## Fluxograma do Raciocínio para a Consulta

- Defina o Objetivo da Consulta:  
Contar o número de funcionários cujo nome contém "ber".
- Identifique a Tabela:  
employees.
- Selecione as Colunas:  
Todas as colunas (\*).
- Defina as Condições de Filtro:  
first\_name LIKE '%ber%'.



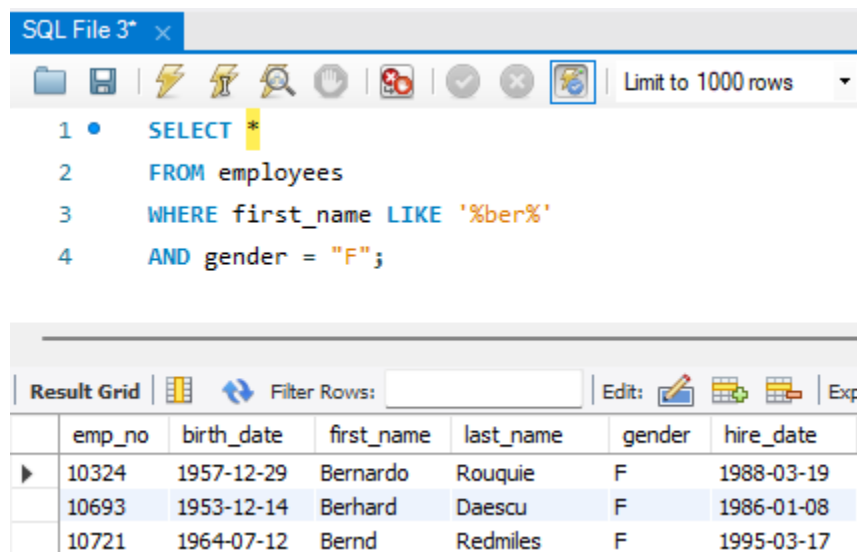
The screenshot shows an SQL IDE window titled "SQL File 3\* x". The query editor contains the following SQL statement:

```
1 • SELECT *
2 FROM employees
3 WHERE first_name LIKE '%ber%';
```

Below the query editor, the "Result Grid" tab is active, displaying the results of the query. The grid has columns for emp\_no, birth\_date, first\_name, last\_name, gender, and hire\_date. The results show 7 rows of data, all of which contain the substring "ber" in the first\_name column.

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	10013	1963-06-07	Eberhardt	Terkki	M	1985-10-20
	10014	1956-02-12	Berni	Genin	M	1987-03-11
	10058	1954-10-01	Berhard	McFarlin	M	1987-04-13
	10171	1957-09-11	Herbert	Trachtenberg	M	1989-07-22
	10304	1956-03-07	Bernt	Erie	M	1992-11-16
	10324	1957-12-29	Bernardo	Rouquie	F	1988-03-19
	10693	1953-12-14	Berhard	Daescu	F	1986-01-08

## LIKE + AND



The screenshot shows a SQL IDE window titled "SQL File 3\* x". The query editor contains the following SQL code:

```
1 • SELECT *
2 FROM employees
3 WHERE first_name LIKE '%ber%'
4 AND gender = "F";
```

Below the query editor, the "Result Grid" is displayed, showing the results of the query. The grid has columns: emp\_no, birth\_date, first\_name, last\_name, gender, and hire\_date. The results are as follows:

emp_no	birth_date	first_name	last_name	gender	hire_date
10324	1957-12-29	Bernardo	Rouquie	F	1988-03-19
10693	1953-12-14	Berhard	Daescu	F	1986-01-08
10721	1964-07-12	Bernd	Redmiles	F	1995-03-17

## Operador IN

Busca **conjunto** de valores.

### Comando

```
SELECT * FROM tableName WHERE column IN(value, value, value, ...)
```

### Fluxograma do Raciocínio

- Defina o **Objetivo da Consulta**:  
Objetivo: Selecionar todos os registros de funcionários que pertencem aos departamentos 'd004', 'd005', e 'd006'.
- **Identifique** a Tabela:  
Tabela: dept\_emp.
- **Selecione** as Colunas:  
Colunas: Todas as colunas (\*).
- **Defina** as **Condições** de Filtro:  
Condição: dept\_no IN('d004', 'd005', 'd006').

SQL File 3\* x

Limit to 10

```

1 • SELECT *
2   FROM dept_emp
3   WHERE dept_no IN('d004', 'd005', 'd006');
4

```

Result Grid | Filter Rows: | Edit:

	emp_no	dept_no	from_date	to_date
▶	10003	d004	1995-12-03	9999-01-01
	10004	d004	1986-12-01	9999-01-01
	10010	d004	1996-11-24	2000-06-26
	10018	d004	1992-07-29	9999-01-01

## IN + AND

SQL File 3\* x

Limit to 1000 rows

```

1 • SELECT *
2   FROM employees
3   WHERE last_name IN('Facello', 'Peac') AND gender = 'M';

```

Result Grid | Filter Rows: | Edit: | Export/In

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	10001	1953-09-02	Georgi	Facello	M	1986-06-26
	10327	1954-04-01	Roded	Facello	M	1987-09-18
	12751	1964-07-06	Nahum	Facello	M	1995-01-09
	14402	1963-10-02	Mooi	Peac	M	1988-11-03
	15685	1958-07-12	Kasturi	Facello	M	1992-03-13

## Operador BETWEEN

Parecido com o IN, mas utiliza **Faixa de Valores**. Geralmente, utilizado com o operador **AND** para determinar onde **termina** o intervalo.


## Comando

SELECT \* FROM tableName WHERE columnName BETWEEN 'value' AND 'value'

## Fluxograma do Raciocínio

- Defina o Objetivo da Consulta:  
Objetivo: Selecionar todos os registros de funcionários que pertencem aos departamentos entre 'd005' e 'd006'.
- Identifique a Tabela:  
Tabela: dept\_emp.
- Selecione as Colunas:  
Colunas: Todas as colunas (\*).
- Defina as Condições de Filtro:  
Condição: dept\_no BETWEEN 'd005' AND 'd009'.

```
1 • SELECT *
2 FROM dept_emp
3 WHERE dept_no BETWEEN 'd005' AND 'd009';
```

Result Grid			
Filter Rows: <input type="text"/>			
Edit: 			
emp_no	dept_no	from_date	to_date
10001	d005	1986-06-26	9999-01-01
10006	d005	1990-08-05	9999-01-01
10008	d005	1998-03-11	2000-07-31
10012	d005	1992-12-18	9999-01-01
10014	d005	1993-12-29	9999-01-01
10018	d005	1987-04-03	1992-07-29
10021	d005	1988-02-10	2002-07-15
10022	d005	1999-09-03	9999-01-01
10023	d005	1999-09-27	9999-01-01
10025	d005	1987-08-17	1997-10-15
10027	d005	1995-04-02	9999-01-01
10028	d005	1991-10-22	1998-04-06
10031	d005	1991-09-01	9999-01-01
10037	d005	1990-12-05	9999-01-01



IN x BETWEEN

IN

**Não** tem um intervalo **em sequência**.

Ex: d005, d007, d009

AND

Tem um intervalo **em sequência**

Ex:

...BETWEEN 'd005' AND 'd009'; ('d005', 'd006', 'd007', 'd008', 'd009')

## Criando ALIAS

Serve para **renomear** uma **coluna** com nome “estranho”.

Comando

```
SELECT SUM(columnName) AS newName FROM tableName
```

Sem ALIAS

```
1  SELECT SUM(salary)
2  FROM salaries;
```

Result Grid		Filter Rows:
	SUM(salary)	
▶	181480757419	

```
1 • SELECT *
2 FROM employees.departments;
```

Result Grid			Filter Rows:
	dept_no	dept_name	
▶	d009	Customer Service	
	d005	Development	
	d002	Finance	
	d003	Human Resources	
	d001	Marketing	
	d004	Production	
	d006	Quality Management	
	d008	Research	
	d007	Sales	
✱	NULL	NULL	

Com ALIAS




```
1 SELECT SUM(salary) AS soma_salario
2 FROM salaries;
```

Result Grid		Filter Rows:	Ex
	soma_salario		
▶	181480757419		

```

1 • SELECT dept_no AS num_department, dept_name AS nam_department
2 FROM employees.departments;

```




Result Grid    Filter Rows: <input type="text"/>   Export:    Wrap Cell Content: 		
	num_department	nam_department
▶	d009	Customer Service
	d005	Development
	d002	Finance
	d003	Human Resources
	d001	Marketing
	d004	Production
	d006	Quality Management
	d008	Research
	d007	Sales

## Colocando Strings

```

1 • SELECT dept_no AS "Department Number", dept_name AS "Department Name"
2 FROM employees.departments;

```

Result Grid    Filter Rows: <input type="text"/>   Export:    Wrap Cell Content: 		
	Department Number	Department Name
	d009	Customer Service
	d005	Development
	d002	Finance
	d003	Human Resources
	d001	Marketing
	d004	Production
	d006	Quality Management
	d008	Research
	d007	Sales

# Criando Constraints na Tabela

## O que são Constraints?

São **regras** que determinam **como os campos serão preenchidos**. Definir que um campo não pode ser nulo "**NOT NULL**", por exemplo. Elas podem ser criadas logo na criação das tabelas, mas podem ser adicionadas posteriormente também. Constraints são importantes para a **organização e padronização** do projeto.

## Utilizando NOT NULL

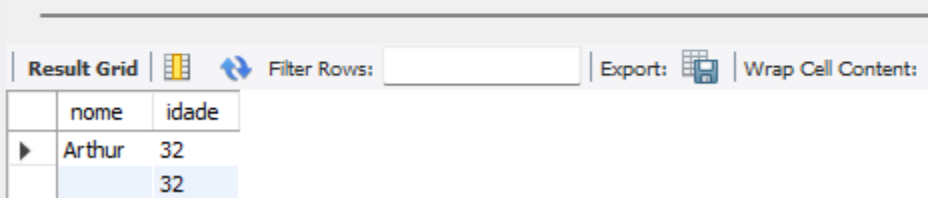
Impede que um **valor** de uma **coluna** seja nulo. Coloca-se a instrução **logo após o nome e tipo da coluna**.

```
1 CREATE TABLE pessoas(  
2     nome VARCHAR(100) NOT NULL,  
3     idade INT  
4 );
```

### \*Cuidado

Não considera **string vazia** um dado nulo!

```
1 • INSERT INTO pessoas (nome, idade) VALUES ("Arthur", 32);  
2 • SELECT * FROM pessoas;  
3 • INSERT INTO pessoas (nome, idade) VALUES ("", 32);
```



The screenshot shows a database interface with a toolbar containing 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. Below the toolbar is a table with two columns: 'nome' and 'idade'. The first row contains 'Arthur' and '32'. The second row contains an empty string and '32'.

nome	idade
Arthur	32
	32

Como funciona?

Colocando diretamente o valor **NULL**.

```

1 • INSERT INTO pessoas (nome, idade) VALUES ("Arthur", 32);
2 • SELECT * FROM pessoas;
3 • INSERT INTO pessoas (nome, idade) VALUES ("", 32);
4 • INSERT INTO pessoas (nome, idade) VALUES (NULL, 28);

```

Output			
Action Output			
#	Time	Action	Message
✓ 1	06:34:50	SHOW DATABASES	9 row(s) returned
✓ 2	06:35:14	CREATE DATABASE constraints	1 row(s) affected
✓ 3	06:35:25	USE constraints	0 row(s) affected
✓ 4	06:40:16	CREATE TABLE pessoas (nome VARCHAR(100) NOT NULL, idade INT)	0 row(s) affected
✓ 5	06:40:37	SELECT * FROM constraints pessoas LIMIT 0, 1000	0 row(s) returned
✓ 6	06:41:47	INSERT INTO pessoas (nome, idade) VALUES ("Arthur", 32)	1 row(s) affected
✓ 7	06:42:14	SELECT * FROM pessoas LIMIT 0, 1000	1 row(s) returned
✓ 8	06:44:13	INSERT INTO pessoas (nome, idade) VALUES ("", 32)	1 row(s) affected
✓ 9	06:44:51	SELECT * FROM pessoas LIMIT 0, 1000	2 row(s) returned
✗ 10	06:47:19	INSERT INTO pessoas (nome, idade) VALUES (NULL, 28)	Error Code: 1048. Column 'nome' cannot be null

## UNIQUE

É a constraint que garante que **todos os valores de uma coluna serão diferentes**. Na prática, uma coluna email, que use essa constraint, **não pode ter emails duplicados**.

```

1 ALTER TABLE pessoas ADD COLUMN email VARCHAR(255) UNIQUE;
2 • SELECT * FROM pessoas;

```

Result Grid			
Filter Rows: <input type="text"/>			
Export: <input type="button" value="Export"/>			
Wrap Cell Content: <input type="button" value="Wrap"/>			
	nome	idade	email
▶	Arthur	32	NULL
		32	NULL

```

1 • INSERT INTO pessoas VALUES ("Maria", 35, "maria@gmail.com");
2 • SELECT * FROM pessoas;

```

Result Grid			
Filter Rows: <input type="text"/>			
Export: <input type="button" value="Export"/>			
Wrap Cell Content: <input type="button" value="Wrap"/>			
	nome	idade	email
▶	Arthur	32	NULL
		32	NULL
	Maria	35	maria@gmail.com

Limit to 1000 rows

1

INSERT INTO pessoas VALUES ("Maria Eduarda", 35, "maria@gmail.com");

Output

Action Output

#	Time	Action	Message
✓ 1	06:34:50	SHOW DATABASES	9 row(s) returned
✓ 2	06:35:14	CREATE DATABASE constraints	1 row(s) affected
✓ 3	06:35:25	USE constraints	0 row(s) affected
✓ 4	06:40:16	CREATE TABLE pessoas (nome VARCHAR(100) NOT NULL, idade INT)	0 row(s) affected
✓ 5	06:40:37	SELECT * FROM constraints pessoas LIMIT 0, 1000	0 row(s) returned
✓ 6	06:41:47	INSERT INTO pessoas (nome, idade) VALUES ("Arthur", 32)	1 row(s) affected
✓ 7	06:42:14	SELECT * FROM pessoas LIMIT 0, 1000	1 row(s) returned
✓ 8	06:44:13	INSERT INTO pessoas (nome, idade) VALUES ("", 32)	1 row(s) affected
✓ 9	06:44:51	SELECT * FROM pessoas LIMIT 0, 1000	2 row(s) returned
✗ 10	06:47:19	INSERT INTO pessoas (nome, idade) VALUES (NULL, 28)	Error Code: 1048. Column 'nome' cannot be null
✓ 11	06:49:48	CREATE TABLE pessoas2 (nome VARCHAR(100) NOT NULL, idade INT NOT NULL)	0 row(s) affected
✗ 12	06:50:38	INSERT INTO pessoas2 (nome, idade) VALUES (NULL, NULL)	Error Code: 1048. Column 'nome' cannot be null
✗ 13	06:50:55	INSERT INTO pessoas2 (nome, idade) VALUES ("Alexander", NULL)	Error Code: 1048. Column 'idade' cannot be null
✗ 14	06:51:07	INSERT INTO pessoas2 (nome, idade) VALUES (NULL, 27)	Error Code: 1048. Column 'nome' cannot be null
✗ 15	06:54:54	INSERT INTO pessoas2 (nome, idade) VALUES ("Alex")	Error Code: 1136. Column count doesn't match value count at row 1
✗ 16	06:55:10	INSERT INTO pessoas2 (nome) VALUES ("Alex")	Error Code: 1364. Field 'idade' doesn't have a default value
✓ 17	06:56:34	SELECT * FROM pessoas2 LIMIT 0, 1000	0 row(s) returned
✓ 18	06:57:22	INSERT INTO pessoas2 (nome, idade) VALUES ("Alexander", 27)	1 row(s) affected
✓ 19	06:57:32	SELECT * FROM pessoas2 LIMIT 0, 1000	1 row(s) returned
✗ 20	06:58:15	INSERT INTO pessoas2 (nome) VALUES ("Nathan")	Error Code: 1364. Field 'idade' doesn't have a default value
✓ 21	06:58:41	SELECT * FROM pessoas2 LIMIT 0, 1000	1 row(s) returned
✗ 22	06:59:23	INSERT INTO pessoas2 (nome) VALUES ("Alex")	Error Code: 1364. Field 'idade' doesn't have a default value
✓ 23	07:04:47	ALTER TABLE pessoas ADD COLUMN email VARCHAR(255) UNIQUE	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
✓ 24	07:04:49	SELECT * FROM pessoas LIMIT 0, 1000	2 row(s) returned
✓ 25	07:07:10	INSERT INTO pessoas VALUES ("Maria", 35, "maria@gmail.com")	1 row(s) affected
✓ 26	07:07:12	SELECT * FROM pessoas LIMIT 0, 1000	3 row(s) returned
✗ 27	07:08:56	INSERT INTO pessoas VALUES ("Maria Eduarda", 35, "maria@gmail.com")	Error Code: 1062. Duplicate entry 'maria@gmail.com' for key 'pessoas_email'

## Adicionar PRIMARY KEY

Cria uma **chave primária** para uma coluna, apenas pode ser utilizada **uma única vez** por tabela, geralmente é o **ID**. Valor **sempre único**, ou seja, **não pode ser nulo**. é um **identificador único de registro na tabela**.

```

1 CREATE TABLE produtos (
2     id INT NOT NULL,
3     nome VARCHAR(255),
4     sku VARCHAR(255),
5     PRIMARY KEY (id)
6 );

```

```

1 • INSERT INTO produtos VALUES(1, "batedeira", "123abc");
2 • SELECT * FROM produtos;
3 • INSERT INTO produtos VALUES(1, "fogão", "123abc");

```

Output			
Action Output			
#	Time	Action	Message
✓ 1	06:34:50	SHOW DATABASES	9 row(s) returned
✓ 2	06:35:14	CREATE DATABASE constraints	1 row(s) affected
✓ 3	06:35:25	USE constraints	0 row(s) affected
✓ 4	06:40:16	CREATE TABLE pessoas (nome VARCHAR(100) NOT NULL, idade INT )	0 row(s) affected
✓ 5	06:40:37	SELECT * FROM constraints.pessoas LIMIT 0, 1000	0 row(s) returned
✓ 6	06:41:47	INSERT INTO pessoas (nome, idade) VALUES ("Arthur", 32)	1 row(s) affected
✓ 7	06:42:14	SELECT * FROM pessoas LIMIT 0, 1000	1 row(s) returned
✓ 8	06:44:13	INSERT INTO pessoas (nome, idade) VALUES ("", 32)	1 row(s) affected
✓ 9	06:44:51	SELECT * FROM pessoas LIMIT 0, 1000	2 row(s) returned
✗ 10	06:47:19	INSERT INTO pessoas (nome, idade) VALUES (NULL, 28)	Error Code: 1048. Column 'nome' cannot be null
✓ 11	06:49:48	CREATE TABLE pessoas2 (nome VARCHAR(100) NOT NULL, idade INT NOT NULL )	0 row(s) affected
✗ 12	06:50:38	INSERT INTO pessoas2 (nome, idade) VALUES (NULL, NULL)	Error Code: 1048. Column 'nome' cannot be null
✗ 13	06:50:55	INSERT INTO pessoas2 (nome, idade) VALUES ("Alexander", NULL)	Error Code: 1048. Column 'idade' cannot be null
✗ 14	06:51:07	INSERT INTO pessoas2 (nome, idade) VALUES (NULL, 27)	Error Code: 1048. Column 'nome' cannot be null
✗ 15	06:54:54	INSERT INTO pessoas2 (nome, idade) VALUES ("Alex")	Error Code: 1136. Column count doesn't match value count at row 1
✗ 16	06:55:10	INSERT INTO pessoas2 (nome) VALUES ("Alex")	Error Code: 1364. Field 'idade' doesn't have a default value
✓ 17	06:56:34	SELECT * FROM pessoas2 LIMIT 0, 1000	0 row(s) returned
✓ 18	06:57:22	INSERT INTO pessoas2 (nome, idade) VALUES ("Alexander", 27)	1 row(s) affected
✓ 19	06:57:32	SELECT * FROM pessoas2 LIMIT 0, 1000	1 row(s) returned
✗ 20	06:58:15	INSERT INTO pessoas2 (nome) VALUES ("Nathan")	Error Code: 1364. Field 'idade' doesn't have a default value
✓ 21	06:58:41	SELECT * FROM pessoas2 LIMIT 0, 1000	1 row(s) returned
✗ 22	06:59:23	INSERT INTO pessoas2 (nome) VALUES ("Alex")	Error Code: 1364. Field 'idade' doesn't have a default value
✓ 23	07:04:47	ALTER TABLE pessoas ADD COLUMN email VARCHAR(255) UNIQUE	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
✓ 24	07:04:49	SELECT * FROM pessoas LIMIT 0, 1000	2 row(s) returned
✓ 25	07:07:10	INSERT INTO pessoas VALUES ("Maria", 35, "maria@gmail.com")	1 row(s) affected
✓ 26	07:07:12	SELECT * FROM pessoas LIMIT 0, 1000	3 row(s) returned
✗ 27	07:08:56	INSERT INTO pessoas VALUES ("Maria Eduarda", 35, "maria@gmail.com")	Error Code: 1062. Duplicate entry 'maria@gmail.com' for key 'pessoas.email'
✓ 28	07:17:13	CREATE TABLE produtos (id INT NOT NULL, nome VARCHAR(255), sku VARCHAR(255), PRIMARY KEY (id) )	0 row(s) affected
✓ 29	07:18:16	SELECT * FROM constraints.produtos LIMIT 0, 1000	0 row(s) returned
✓ 30	07:20:18	INSERT INTO produtos VALUES(1, "batedeira", "123abc")	1 row(s) affected
✓ 31	07:20:21	SELECT * FROM produtos LIMIT 0, 1000	1 row(s) returned
✗ 32	07:21:16	INSERT INTO produtos VALUES(NULL, "fogão", "123abc")	Error Code: 1048. Column 'id' cannot be null
✗ 33	07:21:45	INSERT INTO produtos VALUES(1, "fogão", "123abc")	Error Code: 1062. Duplicate entry '1' for key 'produtos.PRIMARY'

Error Code: 1062. Duplicate entry '1' for key 'produtos.PRIMARY'

## AUTO INCREMENT

Esta constraint adiciona **automaticamente** a quantidade 1(um) em uma coluna, geralmente é usada em ID's. Torna o processo mais simples, sem precisar inserir manualmente com **INSERT**.

```

1 • CREATE TABLE frutas (
2     id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
3     nome VARCHAR(255)
4 );

```

```
1 INSERT INTO frutas (nome) VALUES("banana");
2 • SELECT * FROM frutas;
```

Result Grid			Filter Rows:	Edit:
	id	nome		
▶	1	banana		
*	NULL	NULL		

```
1 INSERT INTO frutas (nome) VALUES("maçã");
2 • SELECT * FROM frutas;
```

Result Grid			Filter Rows:	Edit:
	id	nome		
▶	1	banana		
	2	maçã		
*	NULL	NULL		

### Observação

SQL, MySQL guarda a informação e **nunca repete**, mesmo depois que delete e coloque outro elemento, a contagem **não é alterada**.



```

1 DELETE FROM frutas WHERE id = 4;
2 • SELECT * FROM frutas;
3 • INSERT INTO frutas (nome) VALUES ("maracujá");

```

Result Grid			Filter Rows:	Edit:
	id	nome		
	1	banana		
	2	maçã		
	3	laranja		
	5	maracujá		
*	NULL	NULL		

## FOREIGN KEY

É uma constraint que **liga** uma tabela a outra. Lembre-se que uma **FOREIGN KEY** se refere a uma **PRIMARY KEY** de uma outra tabela. Desta forma, serve para **impedir remoção de dados** que possuem **ligação** entre tabelas.

## FOREIGN KEY

Define que a coluna é uma **chave estrangeira**.

## REFERENCES

**Faz referência** à coluna **PK** da tabela1. Isso cria uma relação entre a tabela2 e tabela1, garantindo que cada valor na **coluna FK** corresponda a **um valor existente** na coluna **PK** da tabela1.

## Comando

Tabela 1 = pessoas

Tabela 2 = endereços



## \*Cuidado

### Receber ID válido

A coluna que serve para **criar relação** entre as tabelas, tem que receber um **ID** válido, ou seja, um **ID** que já exista na **primeira tabela**. Dessa forma, **cria-se uma relação**.

```
1 • INSERT INTO enderecos (street, num, pessoa_id) VALUES ("Rua da Estrela", "242B", 1);
2 • SELECT * FROM enderecos;
```

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

	id	street	num	pessoa_id
▶	4	Rua da Estrela	242B	1
*	NULL	NULL	NULL	NULL

## Não apagar **tabela** com relação

```
1 • INSERT INTO enderecos (street, num, pessoa_id) VALUES ("Rua da Estrela", "242B", 1);
2 • SELECT * FROM enderecos;
3 • SELECT * FROM pessoas;
4 • DROP TABLE pessoas;
```

Output			Action Output		Message	
#	Time	Action				
39	07:24:52	SELECT * FROM produtos LIMIT 0, 1000			2 row(s) returned	
40	07:29:43	CREATE TABLE frutas (id INT PRIMARY KEY AUTO_INCREMENT NOT NULL, nome VARCHAR(255))			0 row(s) affected	
41	07:30:12	SELECT * FROM constraints LIMIT 0, 1000			0 row(s) returned	
42	07:31:25	INSERT INTO frutas (nome) VALUES("banana")			1 row(s) affected	
43	07:31:26	SELECT * FROM frutas LIMIT 0, 1000			1 row(s) returned	
44	07:31:53	INSERT INTO frutas (nome) VALUES("maçã")			1 row(s) affected	
45	07:31:55	SELECT * FROM frutas LIMIT 0, 1000			2 row(s) returned	
46	07:33:00	INSERT INTO frutas (nome) VALUES("laranja")			1 row(s) affected	
47	07:33:01	SELECT * FROM frutas LIMIT 0, 1000			3 row(s) returned	
48	07:33:14	INSERT INTO frutas (nome) VALUES("pêssego")			1 row(s) affected	
49	07:33:15	SELECT * FROM frutas LIMIT 0, 1000			4 row(s) returned	
50	07:35:55	DELETE FROM frutas WHERE id = 4			1 row(s) affected	
51	07:35:57	SELECT * FROM frutas LIMIT 0, 1000			3 row(s) returned	
52	07:36:40	INSERT INTO frutas (nome) VALUES ("maracujá")			1 row(s) affected	
53	07:36:42	SELECT * FROM frutas LIMIT 0, 1000			4 row(s) returned	
54	08:03:07	DROP TABLE pessoas			0 row(s) affected	
55	08:04:26	CREATE TABLE pessoas (id INT PRIMARY KEY AUTO_INCREMENT NOT NULL, nome VARCHAR(255) NOT NULL, idade INT NOT NULL)			0 row(s) affected	
56	08:23:06	CREATE TABLE enderecos (id INT PRIMARY KEY AUTO_INCREMENT NOT NULL, street VARCHAR(100), num VARCHAR(10), pessoa_id INT)			0 row(s) affected	
57	08:24:12	INSERT INTO pessoas(nome, idade) VALUES ("Arthur Guilherme", 32)			1 row(s) affected	
58	08:24:15	SELECT * FROM pessoas LIMIT 0, 1000			1 row(s) returned	
59	08:26:26	INSERT INTO enderecos(street, num, peddoo_id) VALUES ("Rua da Estrela", "242B", 10)			Error Code: 1054. Unknown column 'peddoo_id' in field list	
60	08:26:28	SELECT * FROM enderecos LIMIT 0, 1000			0 row(s) returned	
61	08:26:54	INSERT INTO enderecos(street, num, pessoa_id) VALUES ("Rua da Estrela", "242B", 10)			Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (constraints `enderecos`, CONSTRAINT `enderecos_ibfk_1` FOREIGN KEY (pessoa_id) REFERENCES pessoas (id))	
62	08:27:01	SELECT * FROM enderecos LIMIT 0, 1000			0 row(s) returned	
63	08:27:18	INSERT INTO enderecos(street, num, pessoa_id) VALUES ("Rua da Estrela", "242B", 12)			Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (constraints `enderecos`, CONSTRAINT `enderecos_ibfk_1` FOREIGN KEY (pessoa_id) REFERENCES pessoas (id))	
64	08:28:20	INSERT INTO enderecos (street, num, pessoa_id) VALUES ("Rua da Morte", "6665", 8)			Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (constraints `enderecos`, CONSTRAINT `enderecos_ibfk_1` FOREIGN KEY (pessoa_id) REFERENCES pessoas (id))	
65	08:29:02	INSERT INTO enderecos (street, num, pessoa_id) VALUES ("Rua da Estrela", "242B", 1)			1 row(s) affected	
66	08:29:04	SELECT * FROM enderecos LIMIT 0, 1000			1 row(s) returned	
67	08:46:46	SELECT * FROM pessoas LIMIT 0, 1000			1 row(s) returned	
68	08:47:05	SELECT * FROM pessoas LIMIT 0, 1000			1 row(s) returned	
69	08:47:06	DROP TABLE pessoas			Error Code: 3730. Cannot drop table 'pessoas' referenced by a foreign key constraint 'enderecos_ibfk_1' on table 'enderecos'.	

Error Code: 3730. Cannot drop table 'pessoas' referenced by a foreign key constraint 'enderecos\_ibfk\_1' on table 'enderecos'.

Delete Dados Referenciados, nesse caso são dados da **tabela endereços**, e depois é possível deletar registros ou a tabela **pessoas**.

## INDEX

Para grandes DB, serve para tornar a consulta **mais rápida**, é adicionado um **INDEX(índice)** a coluna. **Não é necessário** aplicar em **todas as colunas**, apenas nas que são **mais consultadas ou lentas** na consulta.

As consultas que são melhoradas pelo INDEX são as com **WHERE**.

### Comando

```
CREATE INDEX indexName  
ON tableName(ColumnName);
```

### Consultas **SEM** índice

A consulta é feita percorrendo **todas as colunas** da tabela até encontrar o que foi solicitado.

```
1  SELECT * FROM employees  
2  WHERE first_name = "Saniya";
```

emp_no	birth_date	first_name	last_name	gender	hire_date
10008	1958-02-19	Saniya	Kalloufi	M	1994-09-15
11208	1952-05-29	Saniya	Valtorta	F	1988-08-26
12582	1961-01-23	Saniya	Herath	F	1987-02-07
12796	1954-05-02	Saniya	Stanfel	M	1992-05-24
13802	1964-01-08	Saniya	Benzmuller	F	1990-10-27
14372	1952-04-15	Saniya	Srimani	F	1988-01-23
15351	1958-02-08	Saniya	Biros	M	1988-01-23
16644	1952-05-20	Saniya	Yoshimura	M	1991-12-13
16727	1963-11-05	Saniya	Codenie	M	1987-09-24
17171	1957-12-24	Saniya	Litzkow	M	1990-07-05
18413	1957-10-05	Saniya	Nannarelli	M	1993-08-31
18864	1956-12-12	Saniya	Kaiserswe...	M	1985-02-07
19249	1953-08-26	Saniya	Iivonen	M	1988-03-27




257 row(s) returned

0.141 sec / 0.000 sec

## Consultas **COM** índice

As demais colunas são ignoradas, ou seja, apenas procura pela coluna que está sendo consultada.

```
1 • CREATE INDEX index_nome
2   ON employees(first_name);
3
4 • SELECT * FROM employees;
5
6 • SELECT * FROM employees
7   WHERE first_name = "Saniya";
```

Result Grid						
Filter Rows: <input type="text"/>						
Edit:   						
	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	10008	1958-02-19	Saniya	Kalloufi	M	1994-09-15
	11208	1952-05-29	Saniya	Valtorta	F	1988-08-26
	12582	1961-01-23	Saniya	Herath	F	1987-02-07
	12796	1954-05-02	Saniya	Stanfel	M	1992-05-24
	13802	1964-01-08	Saniya	Benzmuller	F	1990-10-27
	14372	1952-04-15	Saniya	Srimani	F	1988-01-23
	15351	1958-02-08	Saniya	Biros	M	1988-01-23
	16644	1952-05-20	Saniya	Yoshimura	M	1991-12-13
	16727	1963-11-05	Saniya	Codenie	M	1987-09-24
	17171	1957-12-24	Saniya	Litzkow	M	1990-07-05
	18413	1957-10-05	Saniya	Nannarelli	M	1993-08-31
	18864	1956-12-12	Saniya	Kaiserswe...	M	1985-02-07
	19249	1953-08-26	Saniya	Iivonen	M	1988-03-27

257 row(s) returned 0.015 sec / 0.000 sec

## Remover INDEX

Quanto num for necessário, é bom remover. Excesso de índices pode prejudicar ou atrapalhar uns aos outros.

### Comando

```
DROP INDEX columnName
ON tableName;
```

```
1 • DROP INDEX index_nome
2 ON pessoas;
```

## Unir Tabelas com JOINS

### O que é um JOIN?

Consultas que envolvem **duas ou mais tabelas** que geralmente possuem **relação entre si**, sendo possível realizar sem relação também. É uma consulta mais complexa e com mais dados.

### O que retorna?

Retorna **colunas** em seu resultado.

### Tipo de JOINS mais comuns

LEFT JOIN, RIGHT JOIN e INNER JOIN.

### Existem outros JOINS?

Sim, mas os mais utilizados são os já citados. INNER JOIN é o mais utilizado.

## INNER JOIN

Basicamente, resulta em **colunas** que fazem relação **entre tabelas**. É determinado qual coluna resgatar, através da instrução **SELECT**.

Também utiliza a instrução **ON** para determinar colunas que precisam ser **iguais**, onde determinada coluna é igual a determinada outra coluna.

## Como Funciona?

Verifique relação entre tabelas que serão usadas

**SCHEMAS**

Filter objects

- banco
  - Tables
  - Views
  - Stored Procedures
  - Functions
- constraints
- employees**
  - Tables
    - departments
    - dept\_emp
    - dept\_manager
    - employees
    - salaries
    - titles
  - Views
  - Stored Procedures
  - Functions

1 **SELECT \* FROM employees;**

Limit to 1000 rows

Result Grid Filter Rows: Edit:

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	10001	1953-09-02	Georgi	Facello	M	1986-06-26
	10002	1964-06-02	Bezael	Simmel	F	1985-11-21
	10003	1959-12-03	Parto	Bamford	M	1986-08-28
	10004	1954-05-01	Chirstian	Koblick	M	1986-12-01
	10005	1955-01-21	Kyoichi	Maliniak	M	1989-09-12
	10006	1953-04-20	Anneke	Preusig	F	1989-06-02
	10007	1957-05-23	Tzvetan	Zielinski	F	1989-02-10

Filter objects

- banco
  - Tables
  - Views
  - Stored Procedures
  - Functions
- constraints
- employees**
  - Tables
    - departments
    - dept\_emp
    - dept\_manager
    - employees
    - salaries
    - titles
  - Views
  - Stored Procedures
  - Functions

1 **SELECT \* FROM salaries;**

Result Grid Filter Rows: Edit:

	emp_no	salary	from_date	to_date
▶	10001	60117	1986-06-26	1987-06-26
	10001	62102	1987-06-26	1988-06-25
	10001	66074	1988-06-25	1989-06-25
	10001	66596	1989-06-25	1990-06-25
	10001	66961	1990-06-25	1991-06-25
	10001	71046	1991-06-25	1992-06-24
	10001	74333	1992-06-24	1993-06-24

## Selecione as Tabelas e Colunas

Determine o que será mostrado no **resultado** do JOIN.

Primeiro, selecione as tabelas e colunas que serão mostradas na consulta:

```
1      SELECT employees.first_name, employees.last_name, salaries.salary
```

### Notação

Forma de selecionar **tabela** e **Coluna** simultaneamente, assim, é possível selecionar várias colunas de várias tabelas distintas para realizar o **join**.

TableName.ColumnName

### Tabela Principal

Veja qual é a **tabela principal** da relação e determine

```
2      FROM employees
```

### Tabela JOIN

Veja a tabela que será **ligada a principal**

```
3      INNER JOIN salaries
```

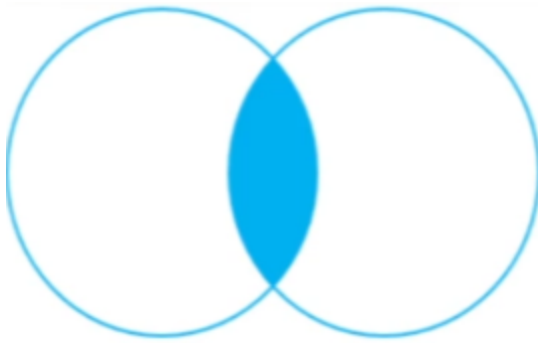
### Onde fazer registro

Onde existe o “**match**”, a mesma informação que pode ligar ambas tabelas.

```
4      ON employees.emp_no = salaries.emp_no
```

### Retorno





### INNER JOIN

```

1  SELECT employees.first_name, employees.last_name, salaries.salary
2  FROM employees
3  INNER JOIN salaries
4  ON employees.emp_no = salaries.emp_no;
5

```

Result Grid		Filter Rows:		Export:		Wrap Cell Content:		Fetch rows
	first_name	last_name	salary					
▶	Georgi	Facello	60117					
	Georgi	Facello	62102					
	Georgi	Facello	66074					
	Georgi	Facello	66596					

### Com ALIAS

```

1  SELECT employees.first_name, employees.last_name, salaries.salary AS salario
2  FROM employees
3  INNER JOIN salaries
4  ON employees.emp_no = salaries.emp_no;
5

```

Result Grid		Filter Rows:		Export:		Wrap Cell Content:		Fetch rows:
	first_name	last_name	salario					
	Charlene	Brattka	113229					
	Charlene	Brattka	112470					
	Charlene	Brattka	111623					

## Com WHERE

```
1  SELECT employees.first_name, employees.last_name, salaries.salary AS salario
2  FROM employees
3  INNER JOIN salaries
4  ON employees.emp_no = salaries.emp_no
5  WHERE salaries.salary >= 120000;
```

Result Grid			
		Filter Rows:	
		Export:	
		Wrap Cell Content:	
		Fetch rows:	
	first_name	last_name	salario
▶	Yannis	Mandell	122275
	Yannis	Mandell	125947
	Bernt	Erie	121296
	Bernt	Erie	121055
	Bernt	Erie	120872

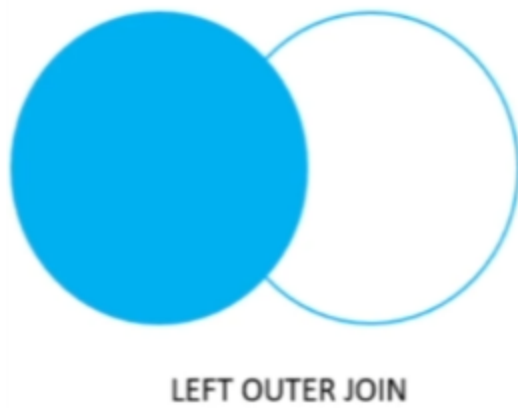
## Com ORDER BY

```
1  SELECT employees.first_name, employees.last_name, salaries.salary AS salario
2  FROM employees
3  INNER JOIN salaries
4  ON employees.emp_no = salaries.emp_no
5  WHERE salaries.salary >= 120000
6  ORDER BY salaries.salary DESC;
7
```

Result Grid			
		Filter Rows:	
		Export:	
		Wrap Cell Content:	
		Fetch rows:	
	first_name	last_name	salario
▶	Tokuyasu	Pesch	158220
	Tokuyasu	Pesch	157821
	Honesty	Mukaidono	156286
	Xiahua	Whitcomb	155709

## LEFT JOIN

Retorna os dados da **Tabela Principal** e os dados **comuns** com a **Tabela Auxiliar**. Esse JOIN também traz colunas que **não tem relações**.



#### Tabela Principal

```
1 • SELECT * FROM pessoas;
```

Result Grid			
Filter Rows:			
	id	nome	idade
▶	1	Arthur Guilherme	32
	2	José	55
*	NULL	NULL	NULL

#### Tabela Auxiliar

```
1 • SELECT * FROM enderecos;
```

Result Grid				
Filter Rows:				
	id	street	num	pessoa_id
▶	4	Rua da Estrela	242B	1
*	NULL	NULL	NULL	NULL

Selecione Tabelas e Colunas

Retorno desejado da Consola

```
1 SELECT pessoas.nome, enderecos.*
```

Notação

É possível selecionar todas as colunas de uma tabela, **com notação**.

Tabela Principal

```
2 FROM pessoas
```

Tabela JOIN

```
3 LEFT JOIN enderecos
```

Onde fazer registro

```
4 ON pessoas.id = enderecos.pessoa_id
```

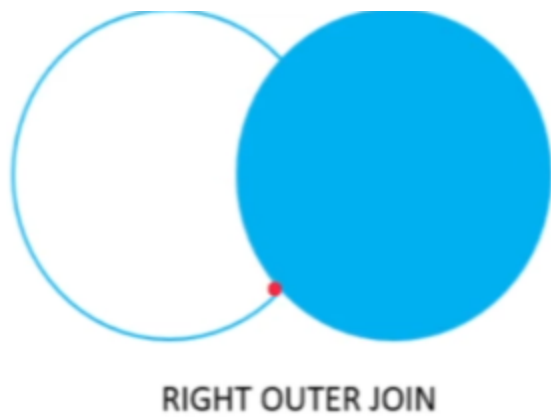
Retorno

```
1 SELECT pessoas.nome, enderecos.*
2 FROM pessoas
3 LEFT JOIN enderecos
4 ON pessoas.id = enderecos.pessoa_id;
5
```

	nome	id	street	num	pessoa_id
▶	Arthur Guilherme	4	Rua da Estrela	242B	1
	José	NULL	NULL	NULL	NULL

RIGHT JOIN

Parecido com o **LEFT JOIN** mas a **tabela auxiliar** que vai mostrar todos as colunas.



### Tabela Principal

```
1 • SELECT * FROM pessoas;
```

Result Grid			
Filter Rows:			
	id	nome	idade
▶	1	Arthur Guilherme	32
	2	José	55
	3	Otávio	16
▲	NULL	NULL	NULL

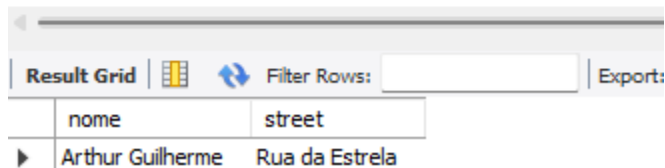
### Tabela Auxiliar

```
1 • SELECT * FROM enderecos;
```

Result Grid				
Filter Rows:				
	id	street	num	pessoa_id
▶	4	Rua da Estrela	242B	1
*	NULL	NULL	NULL	NULL

## Retorno

```
1 • SELECT pessoas.nome, enderecos.street
2 FROM pessoas
3 RIGHT JOIN enderecos
4 ON pessoas.id = enderecos.pessoa_id;
```

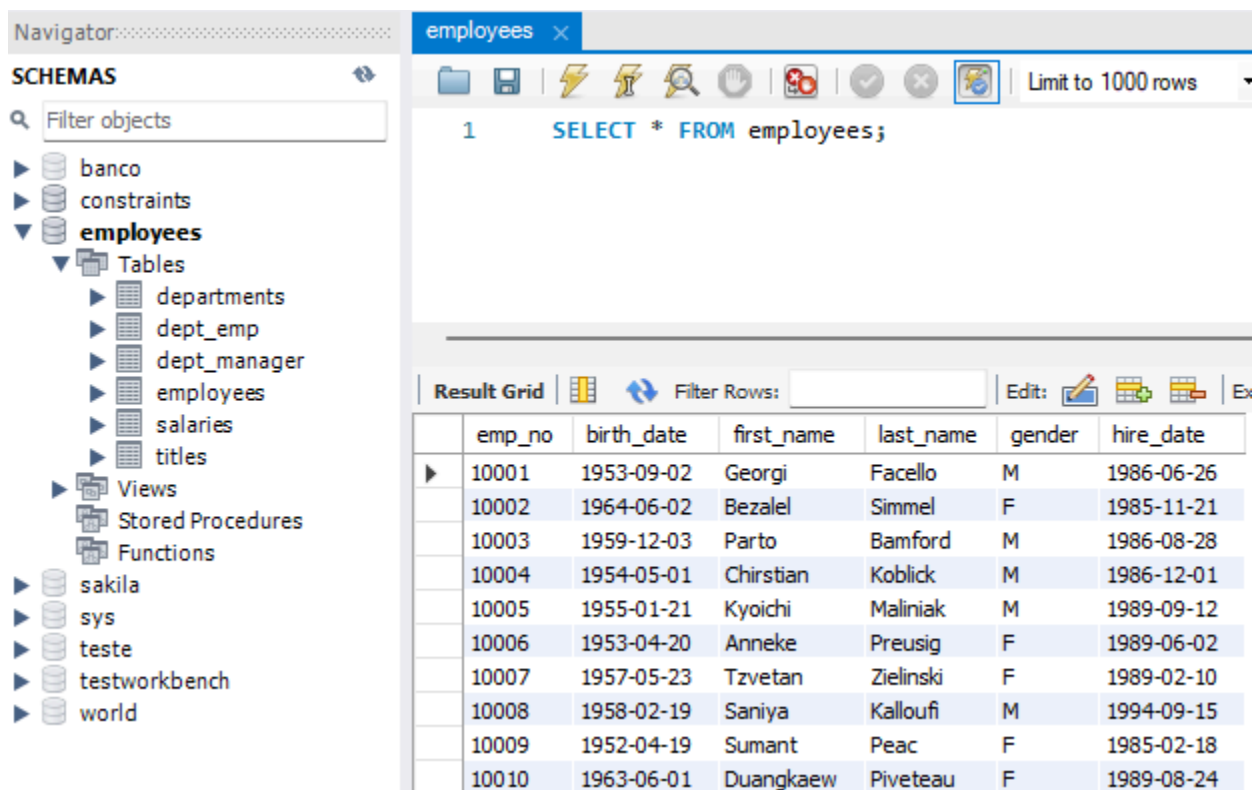


The screenshot shows a database interface with a 'Result Grid' tab. The grid has two columns: 'nome' and 'street'. Below the column headers, there is one row of data: 'Arthur Guilherme' and 'Rua da Estrela'.

nome	street
Arthur Guilherme	Rua da Estrela

## JOIN com +2 Tabelas

### Tabela 1



The screenshot shows a database interface with a 'Navigator' pane on the left and a main query editor on the right. The 'Navigator' pane shows a tree view of the database schema, including tables, views, stored procedures, and functions. The main query editor shows a query: `SELECT * FROM employees;`. Below the query, there is a 'Result Grid' tab showing the results of the query. The grid has seven columns: 'emp\_no', 'birth\_date', 'first\_name', 'last\_name', 'gender', and 'hire\_date'. Below the column headers, there are ten rows of data.

emp_no	birth_date	first_name	last_name	gender	hire_date
10001	1953-09-02	Georgi	Facello	M	1986-06-26
10002	1964-06-02	Bezalel	Simmel	F	1985-11-21
10003	1959-12-03	Parto	Bamford	M	1986-08-28
10004	1954-05-01	Chirstian	Koblick	M	1986-12-01
10005	1955-01-21	Kyoichi	Maliniak	M	1989-09-12
10006	1953-04-20	Anneke	Preusig	F	1989-06-02
10007	1957-05-23	Tzvetan	Zielinski	F	1989-02-10
10008	1958-02-19	Saniya	Kalloufi	M	1994-09-15
10009	1952-04-19	Sumant	Peac	F	1985-02-18
10010	1963-06-01	Duangkaew	Piveteau	F	1989-08-24

Tabela 2

The screenshot shows a SQL IDE interface. On the left is a 'Navigator' pane with a tree view of database schemas. The 'employees' schema is expanded, showing tables: departments, dept\_emp, dept\_manager, employees, salaries, and titles. The 'salaries' table is selected. The main editor pane shows a SQL query: `SELECT * FROM salaries;`. Below the query is a 'Result Grid' showing the data returned by the query. The grid has five columns: emp\_no, salary, from\_date, and to\_date. It contains 10 rows of data, all with emp\_no = 10001, representing the salary history of a specific employee.

	emp_no	salary	from_date	to_date
▶	10001	60117	1986-06-26	1987-06-26
	10001	62102	1987-06-26	1988-06-25
	10001	66074	1988-06-25	1989-06-25
	10001	66596	1989-06-25	1990-06-25
	10001	66961	1990-06-25	1991-06-25
	10001	71046	1991-06-25	1992-06-24
	10001	74333	1992-06-24	1993-06-24
	10001	75286	1993-06-24	1994-06-24
	10001	75994	1994-06-24	1995-06-24
	10001	76884	1995-06-24	1996-06-23

Tabela 3

The screenshot shows the SQL Developer interface. On the left, the 'Navigator' pane displays the 'employees' schema, expanded to show tables: departments, dept\_emp, dept\_manager, employees, salaries, and titles. The main window shows a query editor with the SQL statement: `1 • SELECT * FROM titles;`. Below the editor, the 'Result Grid' displays the query results.

	emp_no	title	from_date	to_date
▶	10001	Senior Engineer	1986-06-26	9999-01-01
	10002	Staff	1996-08-03	9999-01-01
	10003	Senior Engineer	1995-12-03	9999-01-01
	10004	Engineer	1986-12-01	1995-12-01
	10004	Senior Engineer	1995-12-01	9999-01-01
	10005	Senior Staff	1996-09-12	9999-01-01
	10005	Staff	1989-09-12	1996-09-12
	10006	Senior Engineer	1990-08-05	9999-01-01
	10007	Senior Staff	1996-02-11	9999-01-01
	10007	Staff	1989-02-10	1996-02-11

## Selecione Tabelas e Colunas

```
1 • SELECT employees.first_name, salaries.salary, titles.title;
```

## Tabela Principal

```
2 FROM employees;
```

## INNER JOIN +2 Tabelas

Só realizá-lo o **nº de vezes que for preciso**, que nesse caso são 3 tabelas. Logo, 2 vezes:

```
3 INNER JOIN salaries
4 ON employees.emp_no = salaries.emp_no
5 INNER JOIN titles
6 ON salaries.emp_no = titles.emp_no;
```



## Retorno

```
1 • SELECT employees.first_name, salaries.salary, titles.title
2 FROM employees
3 INNER JOIN salaries
4 ON employees.emp_no = salaries.emp_no
5 INNER JOIN titles
6 ON salaries.emp_no = titles.emp_no;
```

first_name	salary	title
Georgi	60117	Senior Engineer
Georgi	62102	Senior Engineer
Georgi	66074	Senior Engineer
Georgi	66596	Senior Engineer
Georgi	66961	Senior Engineer
Georgi	71046	Senior Engineer
Georgi	74333	Senior Engineer
Georgi	75286	Senior Engineer

## com WHERE

```
1 • SELECT employees.first_name, salaries.salary, titles.title
2 FROM employees
3 INNER JOIN salaries
4 ON employees.emp_no = salaries.emp_no
5 INNER JOIN titles
6 ON salaries.emp_no = titles.emp_no
7 WHERE salaries.salary > 125000;
```

first_name	salary	title
Yannis	125947	Senior Staff
Yannis	125947	Staff
Yucel	125811	Senior Staff
Yucel	126614	Senior Staff
Ramalingam	126034	Senior Staff
Ramalingam	126034	Staff
Arno	127479	Senior Staff
Arno	131431	Senior Staff

# Agrupamento e Subqueries

## Operador UNION

Combina o resultado de **dois ou mais SELECTs**. Agrega os valores em apenas **uma coluna**. Porém, somente valores **comuns** as tabelas selecionadas. Ou seja, retorna em uma coluna valores que são iguais e estão nas tabelas escolhidas.

O que retorna?

Retorna **uma coluna** e os respectivos valores dos SELECTs. **Não** traz resultados duplicados. Obtém os dados **únicos** das duas tabelas.

Requisito

Colunas precisam ter o **mesmo nome**.

Como Funciona?

Tabela 1

```
1 • SELECT * FROM departments;
```

Result Grid			Filter Rows:
	dept_no	dept_name	
▶	d009	Customer Service	
	d005	Development	

Tabela 2

```
1 • SELECT * FROM dept_emp;
```

Result Grid				
Filter Rows:				
	emp_no	dept_no	from_date	to_date
▶	10001	d005	1986-06-26	9999-01-01
	10002	d007	1996-08-03	9999-01-01

Coluna 1

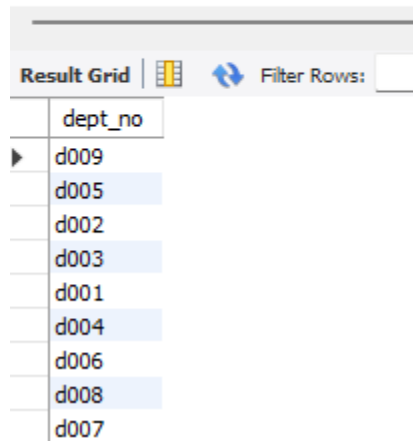
```
1 SELECT dept_no
2 FROM departments
```

Coluna 2

```
3 UNION SELECT dept_no
4 FROM dept_emp;
```

## Retorno

```
1  SELECT dept_no
2  FROM departments
3  UNION SELECT dept_no
4  FROM dept_emp;
```



dept_no
d009
d005
d002
d003
d001
d004
d006
d008
d007

## UNION ALL

Parecido com o operador **UNION** mas a diferença é que pode selecionar **mais de uma coluna** e pode trazer **resultados duplicados**. Vai retornar **todos os valores** da coluna selecionada e comum as duas tabelas.

## Retorno

```
1  SELECT dept_no
2  FROM departments
3  UNION ALL SELECT dept_no
4  FROM dept_emp;
```

Result Grid	
Filter Rows:	
dept_no	
d009	
d005	
d002	
d003	
d001	
d004	
d006	
d008	
d007	
d001	
d001	
d001	
d001	
d001	
d001	
d001	
d001	
d001	

## GROUP BY

Serve para agrupar linhas que têm valores iguais em colunas especificadas, permitindo a execução de **funções de agregação** (como COUNT, SUM, AVG, etc.) em cada grupo de linhas.

O GROUP BY é utilizado para **organizar os dados em grupos** baseados em uma ou mais colunas e, em seguida, executar operações de agregação em cada grupo.

Como funciona?

Tabela 1

```
1 • SELECT * FROM employees;
```

Result Grid						
		Filter Rows:	Edit:			
	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	10001	1953-09-02	Georgi	Facello	M	1986-06-26
	10002	1964-06-02	Bezalel	Simmel	F	1985-11-21
	10003	1959-12-03	Parto	Bamford	M	1986-08-28
	10004	1954-05-01	Chirstian	Koblick	M	1986-12-01
	10005	1955-01-21	Kyoichi	Maliniak	M	1989-09-12
	10006	1953-04-20	Anneke	Preusis	F	1989-06-02

Selecionar Coluna

```
1 • SELECT gender
```

Escolha o que fazer

```
1 • SELECT gender, COUNT(gender) AS 'Qtd for gender'
```

Defina a Tabela

```
2 FROM employees
```

Agrupe a Coluna

```
3 GROUP BY gender;
```

## Retorno

```
1 • SELECT gender, COUNT(gender) AS 'Qtd for gender'
2 FROM employees
3 GROUP BY gender;
```

	gender	Qtd for gender
▶	M	179973
	F	120051

## HAVING

Muito parecido com WHERE, mas utilizado com **aggregate functions** (SUM, AVG, GROUP BY). No WHERE, isso não funciona.

É como uma **condição** para o **agrupamento**.

Aqui basicamente está consultando da seguinte forma:

**Selecione** a coluna *hire\_date*, **CONTE** cada item da coluna (*hire\_date*) e **APRESENTE** como “Qtd por Contratação”

**DA TABELA** *employees*

**Agrupe pela** coluna *hire\_date*

**Com a condição de** **CONTAR** cada valor da coluna (*hire\_date*) **MAIOR QUE** 130

**Ordenando por** **CONTAGEM**(*hire\_date*) **Descendente**;

```
1 SELECT hire_date, COUNT(hire_date) AS "Qtd for Hired"
2 FROM employees
3 GROUP BY hire_date
4 HAVING COUNT(hire_date) > 130
5 ORDER BY COUNT(hire_date) DESC;
6
```

	hire_date	Qtd for Hired
▶	1985-06-20	132
	1985-03-21	131

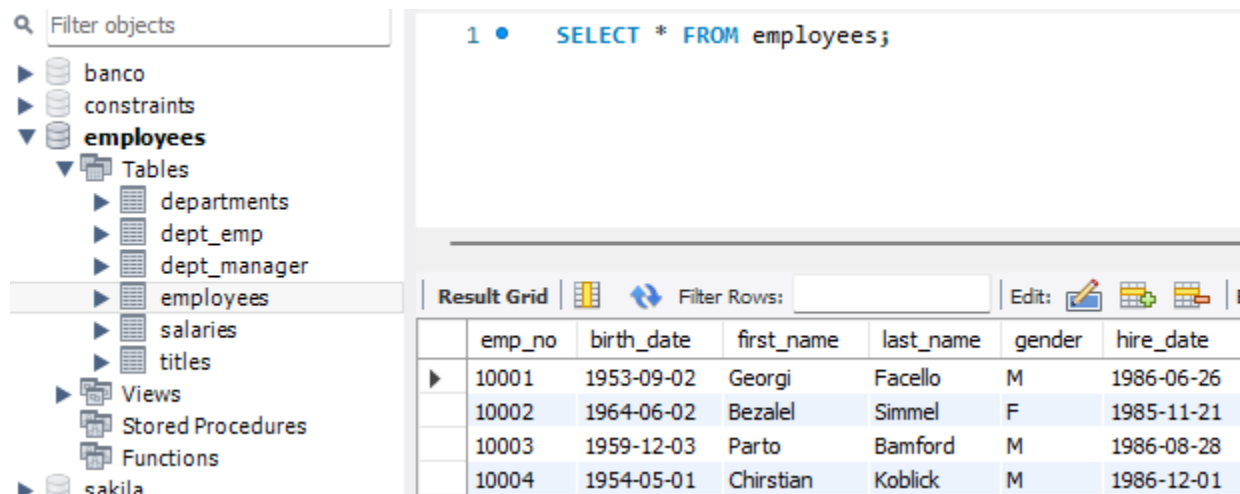
## Subquery

É uma query **dentro** de outra query. Geralmente, **dois** SELECT's. É parecida com um JOIN.

Como Fazer?

Saber o Objetivo da Consulta

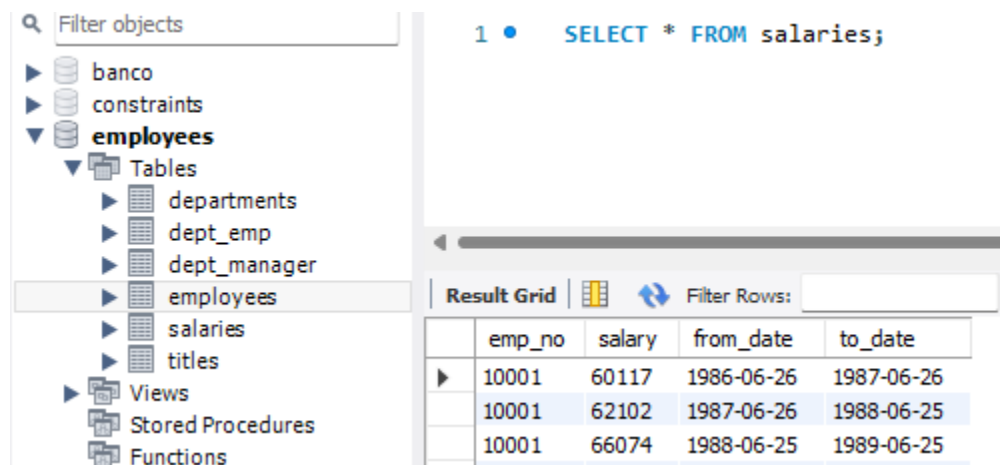
tabela 1



The screenshot shows a database management tool interface. On the left is a tree view of database objects. The 'employees' table is selected. The main area displays a SQL query: `1 • SELECT * FROM employees;`. Below the query is a 'Result Grid' showing the data from the 'employees' table.

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	10001	1953-09-02	Georgi	Facello	M	1986-06-26
	10002	1964-06-02	Bezalel	Simmel	F	1985-11-21
	10003	1959-12-03	Parto	Bamford	M	1986-08-28
	10004	1954-05-01	Chirstian	Koblick	M	1986-12-01

tabela2



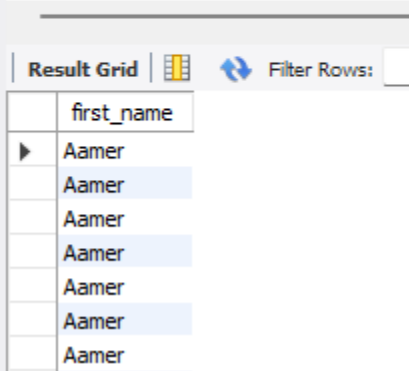
The screenshot shows the same database management tool interface. The 'employees' table is still selected in the tree view. The main area displays a SQL query: `1 • SELECT * FROM salaries;`. Below the query is a 'Result Grid' showing the data from the 'salaries' table.

	emp_no	salary	from_date	to_date
▶	10001	60117	1986-06-26	1987-06-26
	10001	62102	1987-06-26	1988-06-25
	10001	66074	1988-06-25	1989-06-25



## Primeiro SELECT

```
1 • SELECT first_name
2   FROM employees;
```



The screenshot shows a SQL query editor with the following code:

```
1 • SELECT first_name
2   FROM employees;
```

Below the code is a 'Result Grid' tab. The grid has a single column labeled 'first\_name'. It contains 8 rows, all with the value 'Aamer'. The first row is highlighted with a blue background.

first_name
Aamer
Aamer
Aamer
Aamer
Aamer
Aamer
Aamer
Aamer

## Inserir **Outro** SELECT

Seguindo como se fosse um SELECT inicial, porém **entre parênteses** e logo após o primeiro

```
1 • SELECT first_name, (
2     SELECT SUM(salary)
3     FROM salaries
4     ) AS soma_salario
5   FROM employees;
```

## \*\* (REASSISTIR) EXISTS

Serve para **verificar** se existe registro em alguma **subquery**. Retorna apenas se existir algum registro.

## \*\* (REASSISTIR) Utilizando ANY

Muito parecido com o **EXISTS**, ele retorna os dados que são TRUE da Subquery.

## Funções de Strings

## Funções Number

## Funções Date

## Relações Entre Tabelas

O que são relacionamentos entre tabelas

SQL

Linguagem de DB relacionais.

Banco de Dados Relacionais

Relações entre tabelas.

Responsabilidades das Tabelas

Cada tabela em uma DB possui uma responsabilidade. Dessa forma, não é necessário ter uma tabela com tantas colunas, para isso é necessário fazer a devida normalização e divisão de responsabilidades.

Foreign Keys

É o **link** entre essas tabelas. Forma de identificar a relação entre as tabela.

## Tipos de Relacionamentos

Existem vários: 1 para 1, 1 para muitos, muitos para muitos.

## Tipos de Relacionamentos

### One to One (Um para Um)

Quando uma tabela possui apenas **uma conexão com outra tabela e vice-versa**. Cada registro em uma tabela corresponde a um único registro na outra tabela.

### One to Many (Um para Muitos)

Quando uma tabela possui **diversos registros** correspondentes em outra tabela, mas cada registro na segunda tabela está relacionado a apenas um registro na primeira tabela.

### Many to Many (Muitos para Muitos)

Quando duas tabelas possuem **inúmeros registros correspondentes entre si**. Devido a essa complexidade, é criada uma tabela intermediária chamada "Pivot Table".

### Pivot Table (Tabela de Associação)

**Tabela intermediária** usada para gerenciar o relacionamento muitos para muitos entre duas ou mais tabelas. A Pivot Table contém chaves estrangeiras que referenciam as tabelas que estão sendo associadas.

### One to One

Nesse tipo de relacionamento, no **máximo** existe **um registro ligado a outro**. Estrutura definida por uma **FOREIGN KEY**.

## SQL

Uma **FOREIGN KEY** é uma **CONSTRAINT**.

## Exemplo

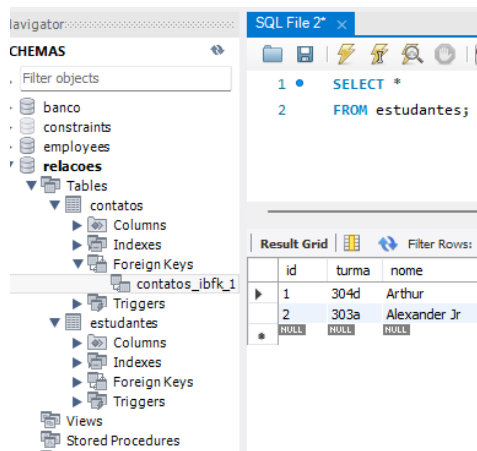
```
CREATE DATABASE relacoes;
```

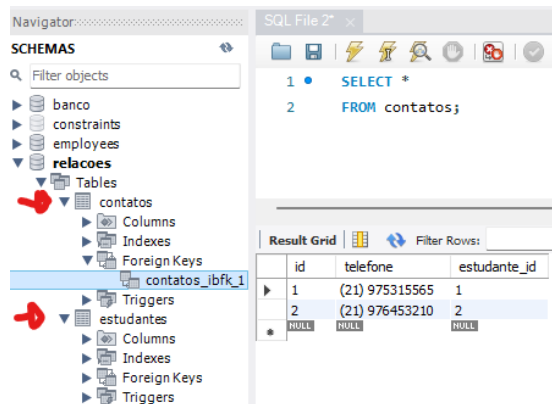
```
USE relacoes;
```

```
CREATE TABLE estudantes (  
    id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,  
    nome VARCHAR(100),  
    turma VARCHAR(5)  
);
```

```
CREATE TABLE contatos (  
    id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,  
    telefone VARCHAR(20),  
    estudante_id INT NOT NULL,  
    FOREIGN KEY (estudante_id) REFERENCES estudantes(id)  
);
```

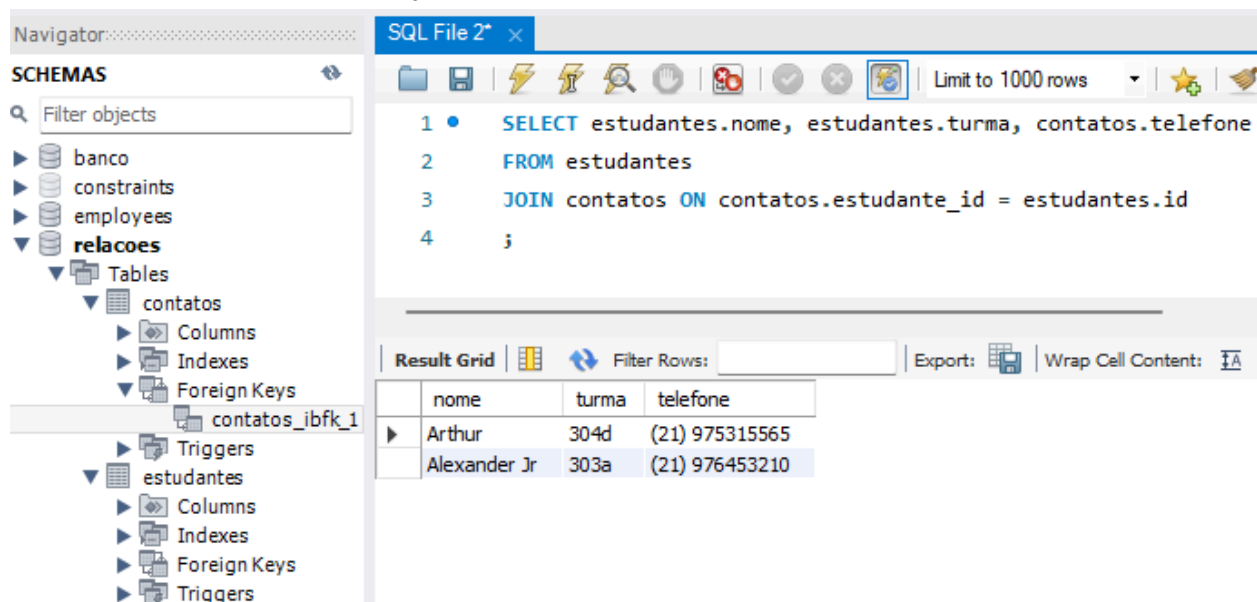
```
INSERT INTO contatos (telefone, estudante_id) VALUES ("(21) 975315565", 1);  
INSERT INTO contatos (telefone, estudante_id) VALUES ("(21) 976453210", 2);
```





## Teste

Verificar dados dos estudantes junto aos contatos:



## One to Many

Uma tabela tem **um registro** que corresponde a **vários registros** em outra tabela. A segunda tabela tem **vários registros** que correspondem a apenas **um registro** em outra tabela. Agora, para diferenciá-lo do tipo **one to one**, apenas é necessário imaginar que, conforme exemplo anterior, dois ou mais estudantes tivessem o **mesmo contato**.

## Exemplo

CREATE TABLE clientes(

```

        id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
        nome VARCHAR(100),
        data_nascimento DATE
    );

```

```

CREATE TABLE pedidos(
    id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    qtd_itens INT(10),
    total FLOAT,
    cliente_id INT NOT NULL,
    FOREIGN KEY (cliente_id) REFERENCES clientes(id)
);

```

```

INSERT INTO clientes (nome, data_nascimento) VALUES("Arthur", "1992-04-04");
INSERT INTO clientes (nome, data_nascimento) VALUES("Alexander", "1997-07-23");

```

```

INSERT INTO pedidos (qtd_itens, total, cliente_id) VALUES(5, 214.01, 1 );
INSERT INTO pedidos (qtd_itens, total, cliente_id) VALUES(8, 304.77, 2 );

```

The screenshot shows a SQL IDE interface. On the left, the 'Navigator' pane displays a tree view of the database schema. The 'relacoes' (relationships) folder is expanded, showing 'clientes' and 'pedidos' tables. The 'clientes' table is highlighted with a red bar. The 'pedidos' table is also highlighted with a red bar. The 'sakila' database is selected. On the right, the 'SQL File 2\*' pane shows a query: `SELECT * FROM clientes;`. Below the query, the 'Result Grid' pane displays the results of the query. The grid has four columns: 'id', 'nome', and 'data\_nascimento'. It contains two rows of data: one for 'Arthur' (id 1, data\_nascimento 1992-04-04) and one for 'Alexander' (id 2, data\_nascimento 1997-07-23). A third row with all NULL values is also shown.

	id	nome	data_nascimento
▶	1	Arthur	1992-04-04
	2	Alexander	1997-07-23
*	NULL	NULL	NULL

Navigator

**SCHEMAS**

Filter objects

- banco
- constraints
- employees
- relacoes**
  - Tables
    - clientes
      - Columns
      - Indexes
      - Foreign Keys
      - Triggers
    - contatos
    - estudantes
    - pedidos
      - Columns
      - Indexes
      - Foreign Keys
      - Triggers
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- teste
- testworkbench
- world

SQL File 2\* x

```

1 • SELECT *
2 FROM pedidos;

```

Result Grid

	id	qtd_itens	total	cliente_id
▶	1	5	214.01	1
	2	8	304.77	2
	3	10	214.01	1
	4	4	104.77	2
	5	8	314.01	2
	6	15	604.77	2
	7	13	414.01	2
	8	11	404.77	2
	9	20	714.01	1
	10	11	904.77	1
	11	7	314.01	1
	12	4	808.77	2
	13	7	377.66	2
	14	13	1000.99	2
*	NULL	NULL	NULL	NULL

SQL File 2\* x

```

1 -- Pedidos Arthur
2 • SELECT clientes.nome AS Nome, pedidos.id AS pedido
3 FROM clientes
4 JOIN pedidos ON pedidos.cliente_id = clientes.id
5 WHERE clientes.id = 1

```

Result Grid

	Nome	pedido
▶	Arthur	1
	Arthur	3
	Arthur	9
	Arthur	10
	Arthur	11

SQL File 2\* x

Limit to 1000 rows

```

1  -- Pedidos Alexander
2  • SELECT clientes.nome AS Nome, pedidos.id AS pedido
3  FROM clientes
4  JOIN pedidos ON pedidos.cliente_id = clientes.id
5  WHERE clientes.id = 2

```

Result Grid | Filter Rows: | Export: | Wrap Cell C

	Nome	pedido
▶	Alexander	2
	Alexander	4
	Alexander	5
	Alexander	6
	Alexander	7
	Alexander	8
	Alexander	12
	Alexander	13
	Alexander	14

## Many to Many

Duas tabelas tem **múltiplas relações** entre si. Sendo necessária a criação de uma **Pivot Table** onde apresenta apenas as **relações** entre as tabelas.

### Exemplo

```

CREATE TABLE materias (
    id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    nome VARCHAR(100)
);

```

Tabela Estudantes já criada, com um comando acima.

### PIVOT TABLE

```

CREATE TABLE estudante_materia (
    estudante_id INT NOT NULL,
    materia_id INT NOT NULL,
    FOREIGN KEY (estudante_id) REFERENCES estudantes(id),

```



```
FOREIGN KEY (materia_id) REFERENCES materias(id)
);
```

```
INSERT INTO materias (nome) VALUES ("Matemática");
INSERT INTO materias (nome) VALUES ("Português");
INSERT INTO materias (nome) VALUES ("Química");
INSERT INTO materias (nome) VALUES ("Física");
INSERT INTO materias (nome) VALUES ("Lógica");
```

The screenshot shows a database management interface. On the left, a tree view displays the database structure, including tables like `clientes`, `contatos`, `estudante_materia`, `estudantes`, `materias`, and `pedidos`. The `relacoes` folder is expanded, showing these tables. On the right, a SQL query is entered: `SELECT * FROM estudantes;`. Below the query, the results are displayed in a table grid. The table has three columns: `id`, `turma`, and `nome`. The results show six rows of student data, followed by a row with three NULL values.

	id	turma	nome
1	1	304d	Arthur
2	2	303a	Alexander Jr
3	3	202b	Maria
4	4	202b	José
5	5	402a	Pedro
6	6	301c	João
*	NULL	NULL	NULL

Navigator

SCHEMAS

Filter objects

banco

constraints

employees

relacoes

Tables

clientes

contatos

estudante\_materia

estudantes

materias

pedidos

Views

Stored Procedures

Functions

sakila

sys

teste

testworkbench

world

Administration

Schemas

Information

SQL File 2\* x

1

SELECT \* FROM estudante\_materia;

Result Grid

Filter Rows:

Ex

	estudante_id	materia_id
▶	1	1
	1	1
	1	1
	1	2
	1	3
	1	4
	2	1
	3	2
	3	1
	4	2
	4	3
	4	4
	5	1
	5	2
	5	3
	6	3
	6	4
	6	5
	6	5
	6	4
	6	4
	6	4

estudante\_materia 20 x

The screenshot shows a database management tool interface. On the left is a 'Filter objects' sidebar with a tree view containing databases like 'banco', 'constraints', 'employees', 'relacoes', 'sakila', 'sys', 'teste', 'testworkbench', and 'world'. The 'relacoes' database is expanded, showing 'Tables' (clientes, contatos, estudante\_materia, estudantes, materias, pedidos), 'Views', 'Stored Procedures', and 'Functions'. The main area displays a SQL query: `1 • SELECT * FROM materias;`. Below the query is a 'Result Grid' showing the following data:

	id	nome
▶	1	Matemática
	2	Português
	3	Química
	4	Física
	5	Lógica
*	NULL	NULL

INSERT INTO estudante\_materia (estudante\_id, materia\_id) VALUES (1, 1);

The screenshot shows a database management tool interface. On the left is a 'Filter objects' sidebar with a tree view containing databases like 'banco', 'constraints', 'employees', 'relacoes', 'sakila', 'sys', 'teste', 'testworkbench', and 'world'. The 'relacoes' database is expanded, showing 'Tables' (clientes, contatos, estudante\_materia, estudantes, materias, pedidos), 'Views', 'Stored Procedures', and 'Functions'. The main area displays a SQL query: `1 SELECT *  
2 FROM estudantes  
3 JOIN estudante_materia ON estudante_materia.estudante_id = estudantes.id  
4 AND estudante_materia.materia_id = 2`. Below the query is a 'Result Grid' showing the following data:

	id	turma	nome	estudante_id	materia_id
▶	1	304d	Arthur	1	2
	3	202b	Maria	3	2
	4	202b	José	4	2
	5	402a	Pedro	5	2

## Planejamento de Banco de Dados

### Importância do Database Design

Basicamente é um **mapa** do Database. Mostra o **relacionamento** entre entidades e ajuda antecipar problemas da **regra de negócios**.

## Análise de Requisitos

Planejamento e definição do sistema, é **como o sistema deve funcionar**. Isso dá a base para **planejar o banco de dados**.

## Normalização de um Banco de Dados

É um processo dividido em níveis que ajudam a normalizar, melhorar, otimizar o DB.

### Níveis

#### Zero

Colocar Primary Key.

#### NF1

Cada coluna guarda **um valor**.

#### NF2

Colunas que não pertencem ao **tópico principal** da **tabela principal**, devem **virar outra tabela**.

#### NF3

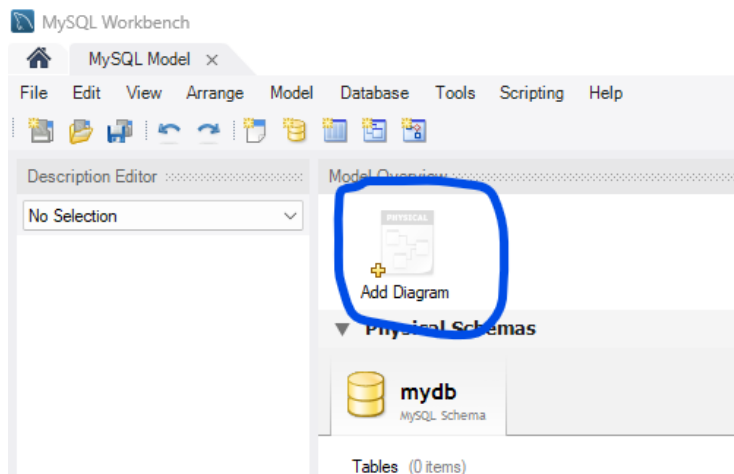
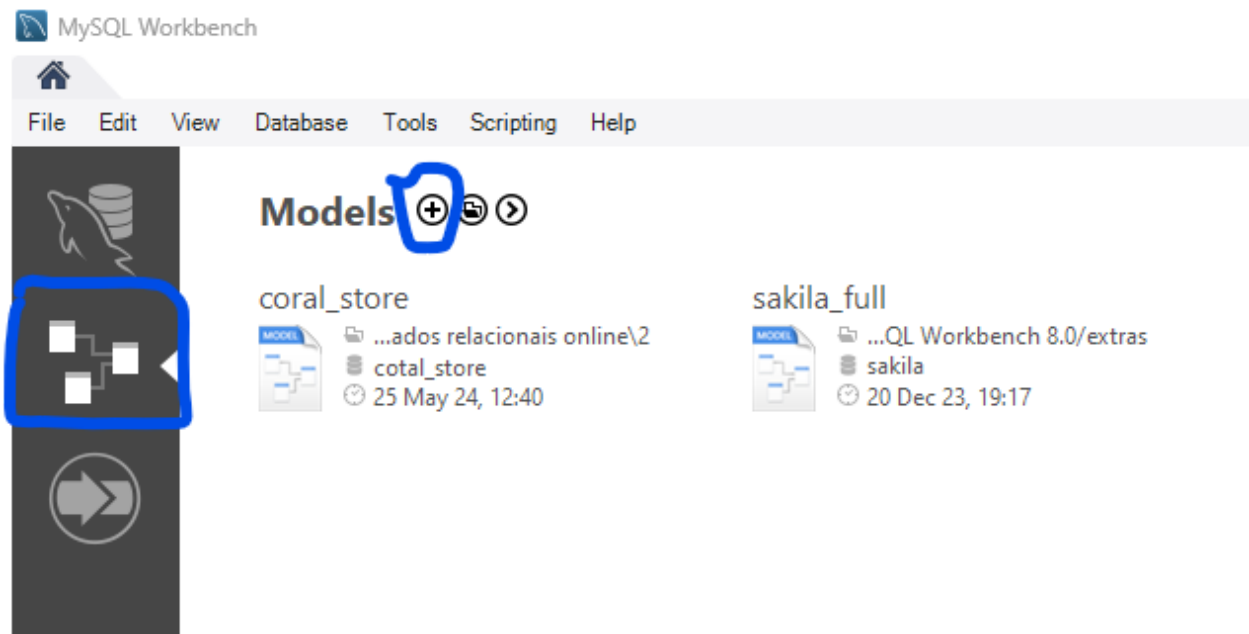
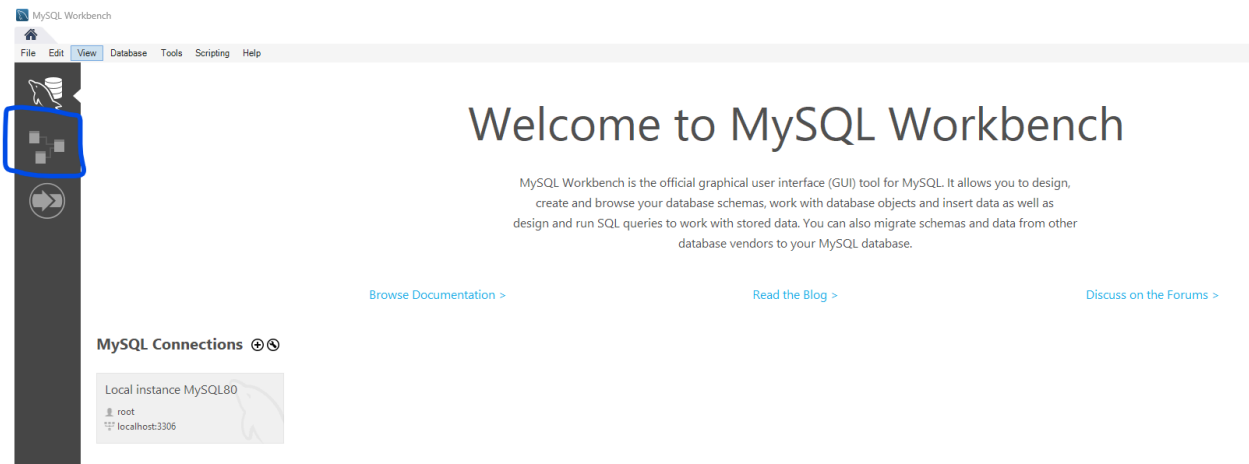
Deixa na DB apenas valores que não dependem de outros, ou seja, apenas dados independentes.

Ex: Ter uma tabela com impostos e outra com salários base. Assim, pode ser atualizados os valores de impostos sem comprometer o DB.

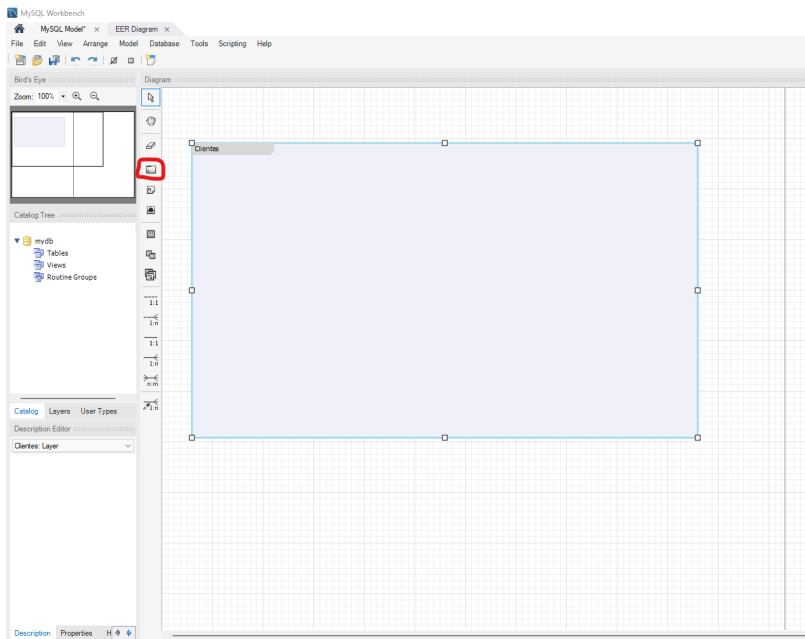
## Diagramas de Entidades Relacionais (ER)

Quadro que **define as tabelas e relações** entre si. Permite deixar o DB mais visual, **dar nomes** as tabelas e colunas. **Definir os tipos de dados** e **facilitar o entendimento** da DB de forma masi rápida.

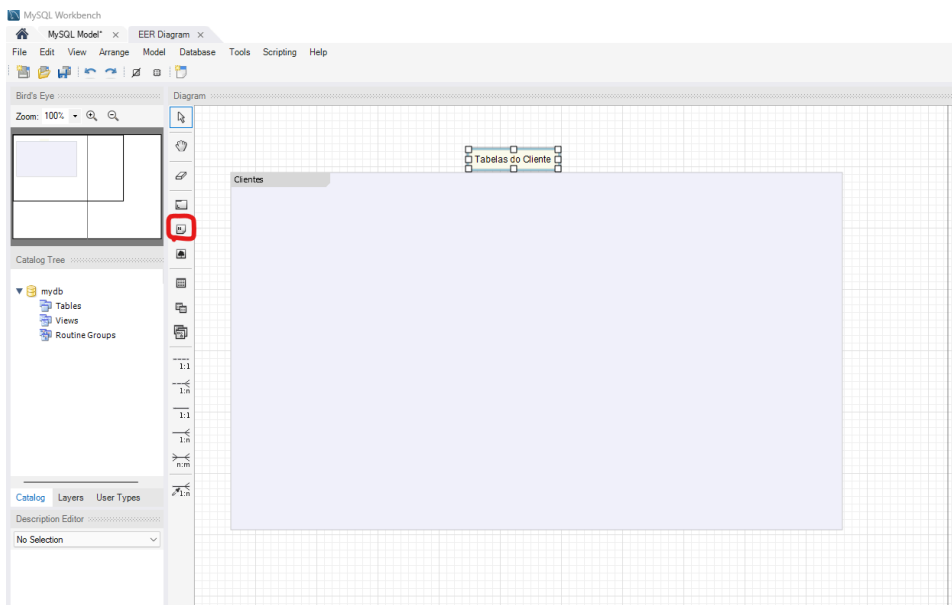
## Diagrama ER na prática no Workbench



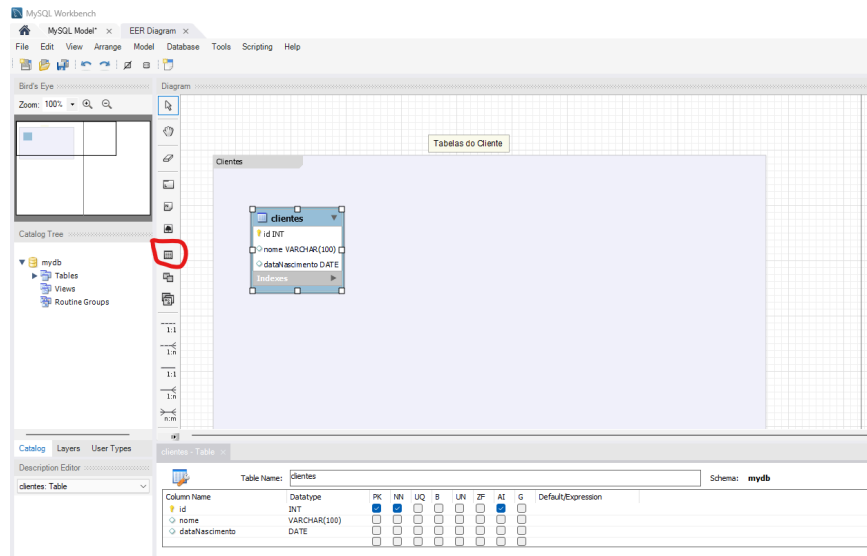
## Criar Camada



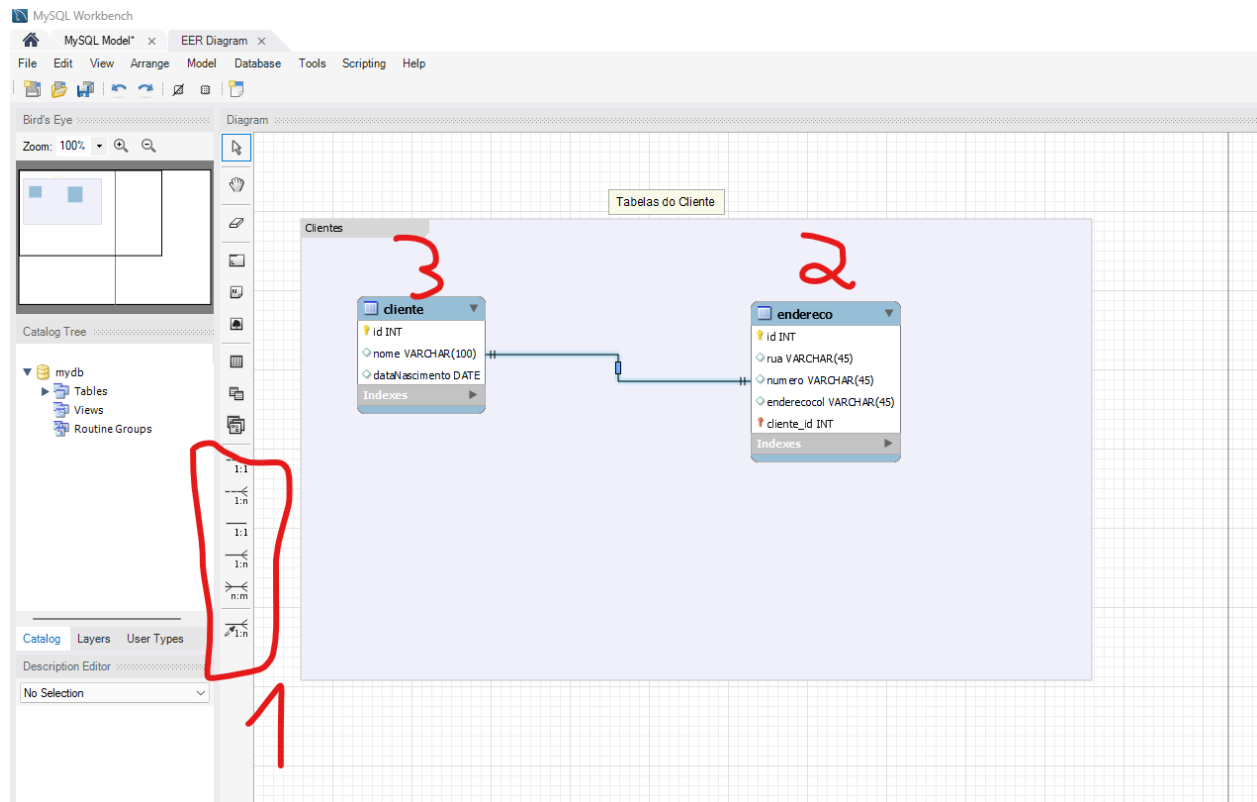
## Nota de Texto



## Criar Tabelas



## Criar Relações



# Implementação

## Análise do Problema

Tenha todos os requisitos em mãos e entenda cada um para construir o sistema.

### *Pizzaria do João*

*Nosso amigo João quer abrir uma pizzaria, porém apenas delivery. Ele precisa de um software para gerenciar os pedidos que recebe. Inicialmente ele quer que os clientes façam o pedido em um site,*

*podendo escolher:*

- Massa;
- Borda;
- Sabores ( no máximo 3 );

*Além disso, ele precisa conseguir gerenciar estes pedidos.*

*E também mudar o status de cada um:*

- Em produção;
- Entrega;
- Concluído;

*O pedido será retirado no balcão, então **não precisamos salvar dados do cliente**, apenas a pizza.*

*João quer ter a possibilidade de **remover pedidos**, para os que são **cancelados pelos clientes**.*

*João também passou a relação de algumas massas que utiliza:*

- Massa comum;
- Massa integral;
- Massa temperada;

*As bordas são:*

- Cheddar;
- Catupiry;

*E os sabores são (máximo 3):*

- 4 Queijos;
- Frango com Catupiry;
- Calabresa;
- Lombinho;
- Filé com Cheddar;
- Portuguesa;
- Margherita;

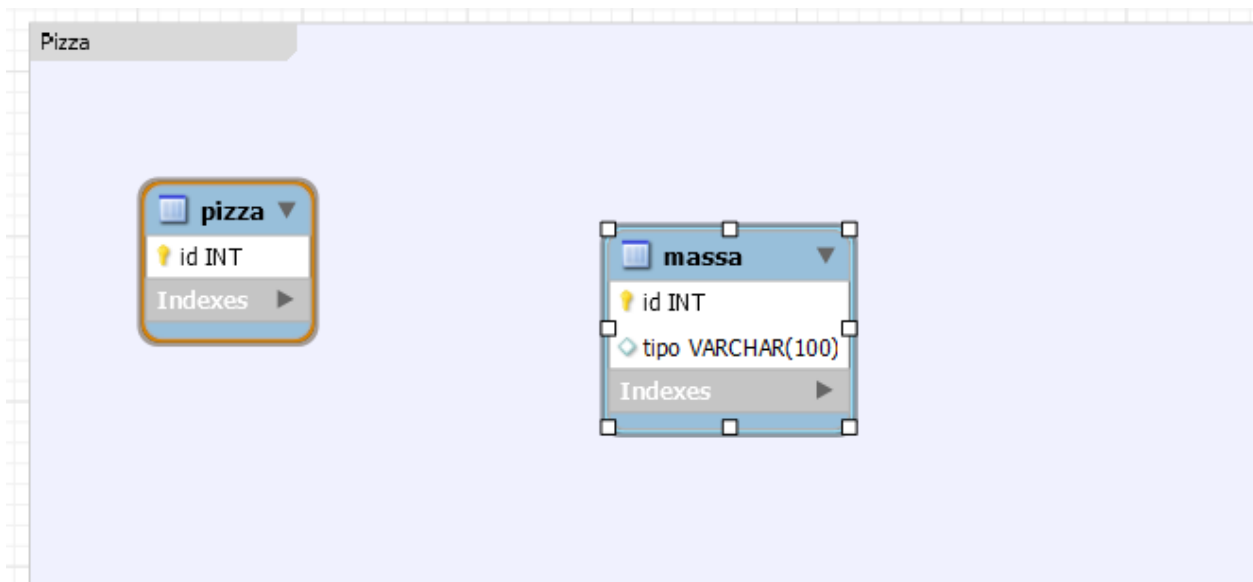


## Criando um Diagrama em ER

### Criar Camadas Necessárias



### Criar Tabelas



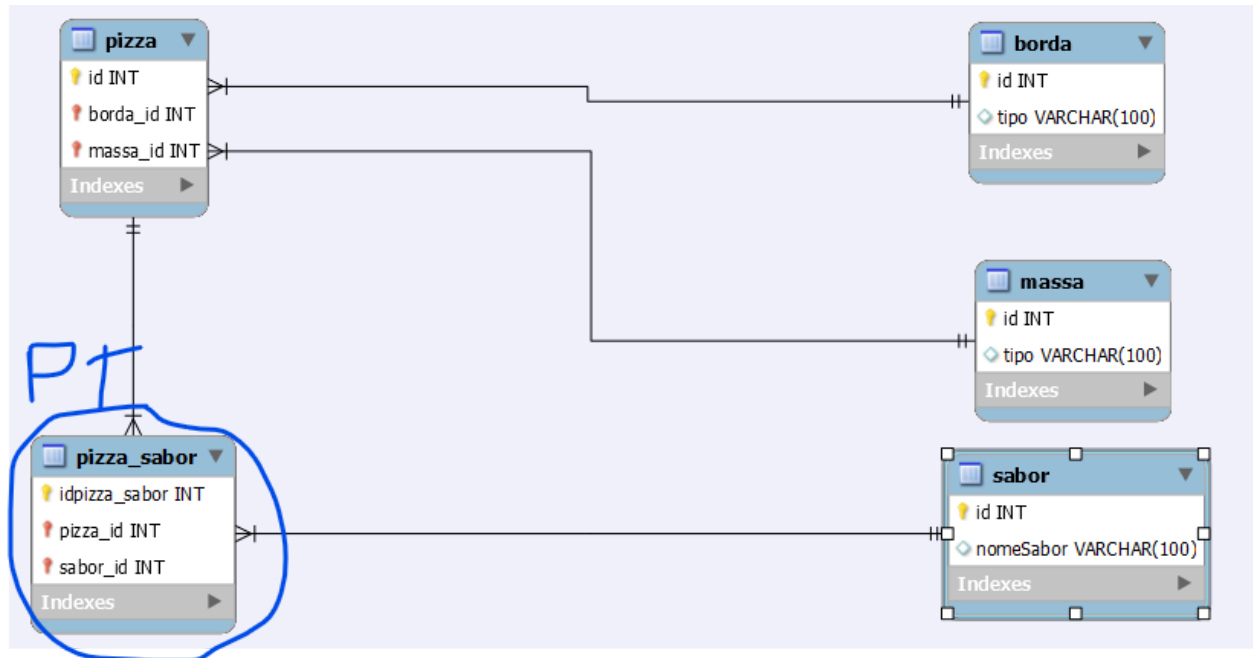
## Criar Relações

“Uma pizza tem **apenas uma massa**, mas uma massa pode estar em **uma ou várias pizzas**.”

“Uma pizza tem **apenas uma borda**, mas uma borda pode estar em **uma ou várias pizzas**.”

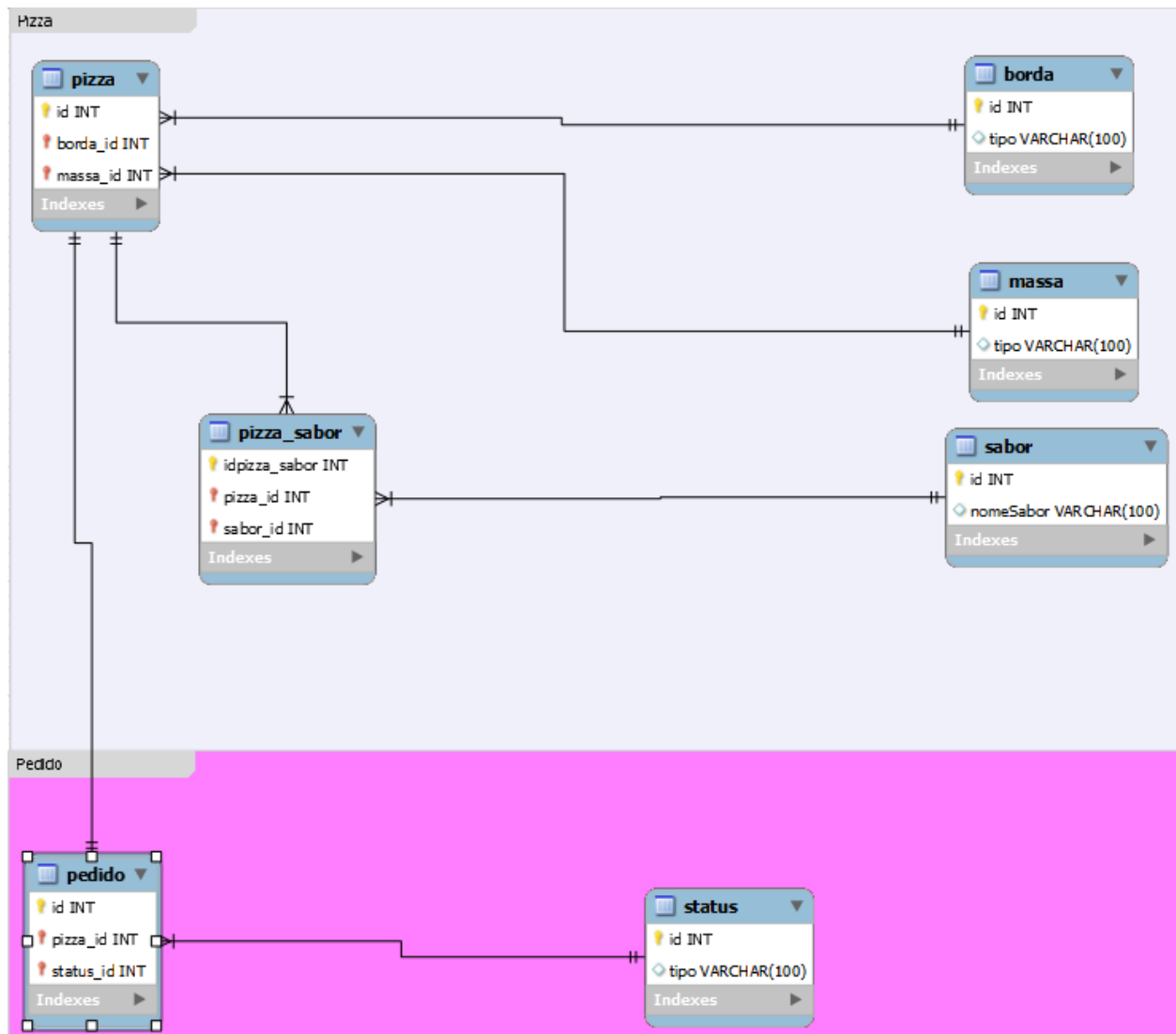
“Uma pizza tem **um ou vários sabores**, mas uma massa pode estar em **uma ou várias pizzas**.”

Por isso, cria-se uma **Pivot Table**.



“Uma pizza pode estar em apenas **um pedido**, um pedido pode ter apenas **uma pizza**.”

“Um pedido pode ter um ou vários status, mas um status pode ter estar em apenas **um pedido**.”



## Projeto: Pizzaria do João

### Criação do Bando de Dados

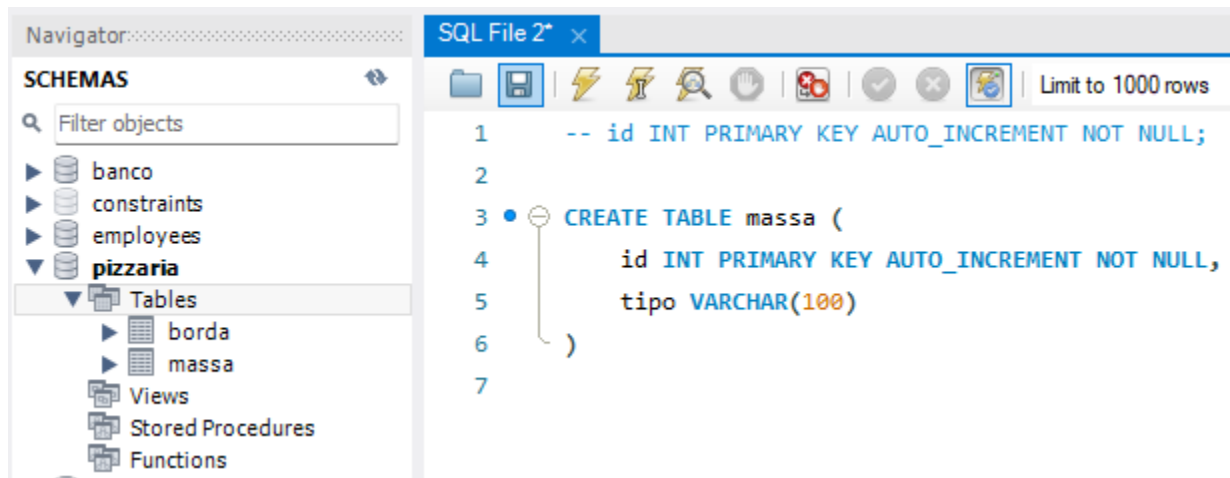
CREATE pizzaria;

USE pizzaria;



```
CREATE TABLE borda (
    id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    tipo VARCHAR(100)
)
```

```
CREATE TABLE massa (
    id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    tipo VARCHAR(100)
)
```

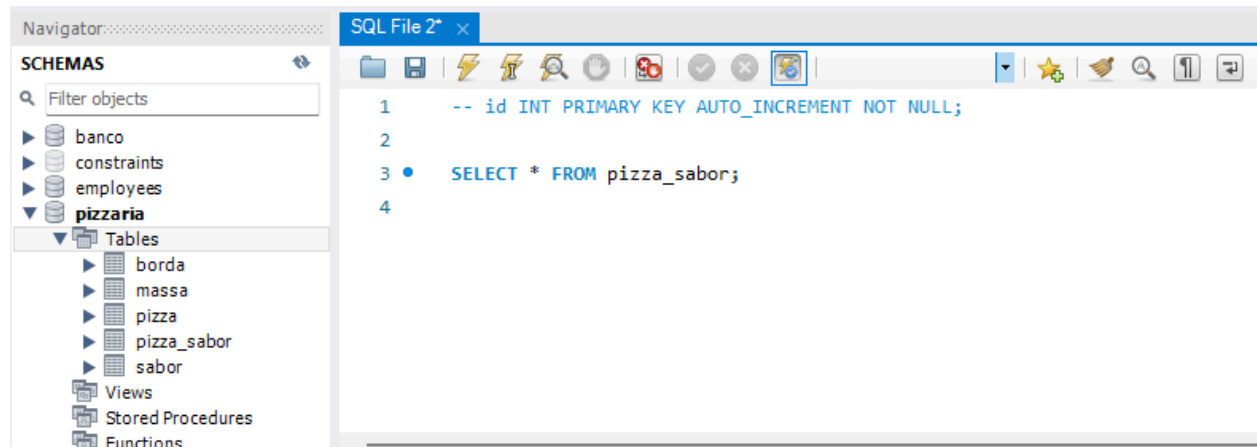


```
CREATE TABLE pizza (
    id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    borda_id INT NOT NULL,
    massa_id INT NOT NULL,
    FOREIGN KEY (borda_id) REFERENCES borda(id),
    FOREIGN KEY (massa_id) REFERENCES massa(id)
)
```

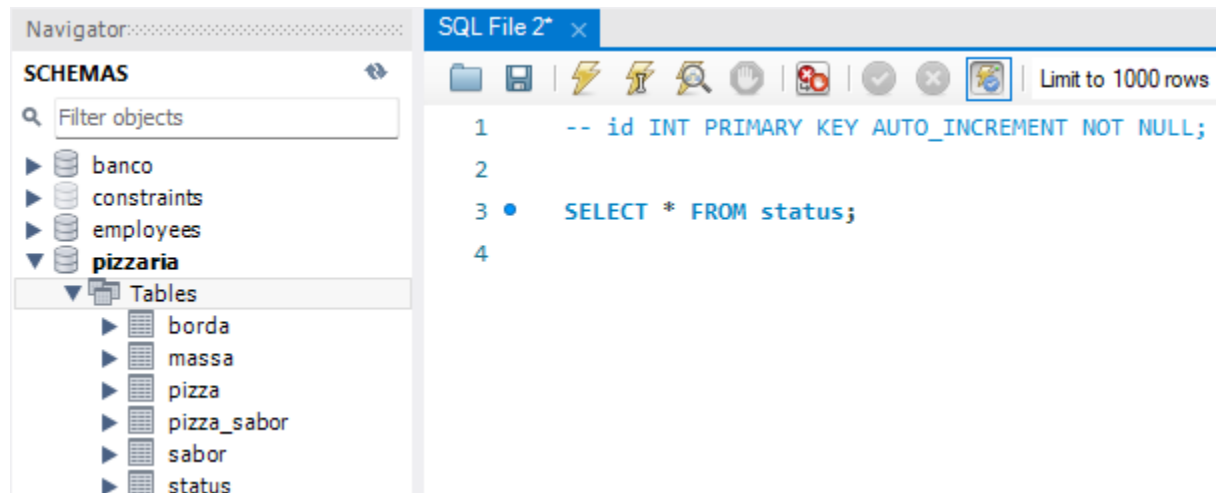
## Pivot Table

```
CREATE TABLE pizza_sabor(
    id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    pizzaria_id INT NOT NULL,
    sabor_id INT NOT NULL,
    FOREIGN KEY (pizzaria_id) REFERENCES pizza(id),
    FOREIGN KEY (sabor_id) REFERENCES sabor(id)
)
```

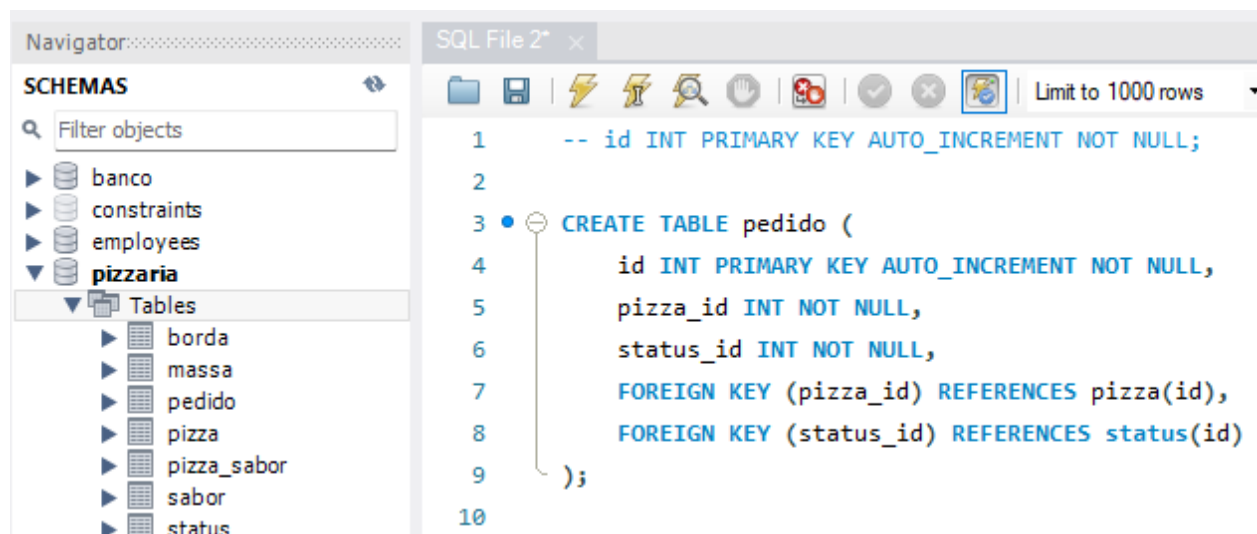
);



```
CREATE TABLE status (  
    id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,  
    tipo VARCHAR(100)  
);
```



```
CREATE TABLE pedido (  
    id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,  
    pizza_id INT NOT NULL,  
    status_id INT NOT NULL,  
    FOREIGN KEY (pizza_id) REFERENCES pizza(id),  
    FOREIGN KEY (status_id) REFERENCES status(id)  
);
```

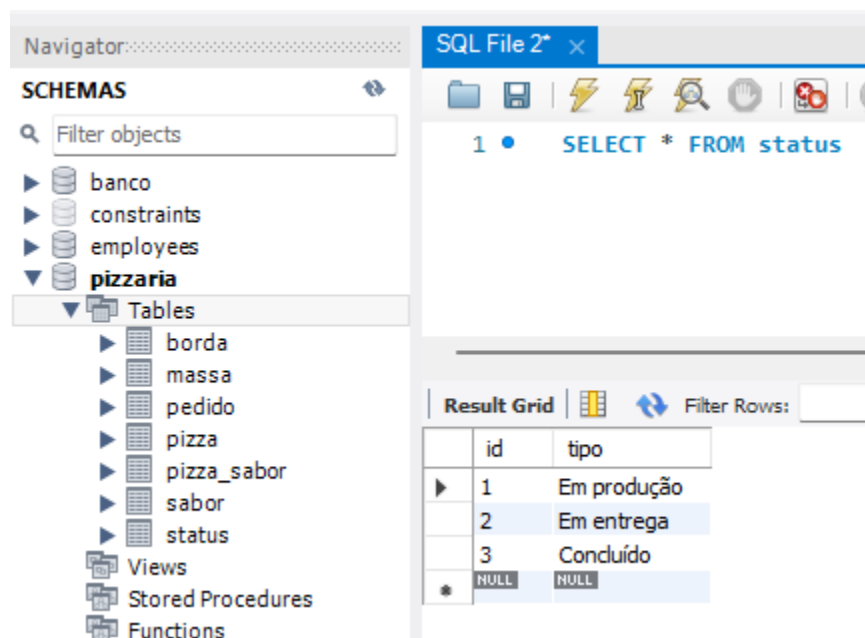


## Inserindo Dados no Sistema

```

INSERT INTO status (tipo) VALUES ("Em produção");
INSERT INTO status (tipo) VALUES ("Em entrega");
INSERT INTO status (tipo) VALUES ("Concluído");

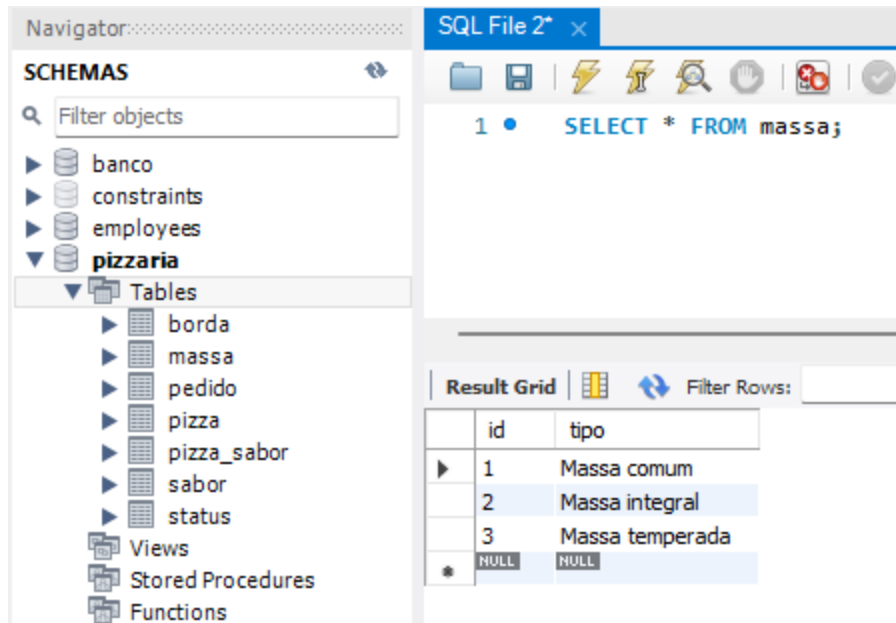
```



```

INSERT INTO massa (tipo) VALUES ("Massa comum");
INSERT INTO massa (tipo) VALUES ("Massa integral");
INSERT INTO massa (tipo) VALUES ("Massa temperada");

```



```
INSERT INTO sabor (nome) VALUES ("Frango com Catupiry");  
INSERT INTO sabor (nome) VALUES ("Calabresa");  
INSERT INTO sabor (nome) VALUES ("Lombinho");  
INSERT INTO sabor (nome) VALUES ("Portuguesa");  
INSERT INTO sabor (nome) VALUES ("Margherita");  
INSERT INTO sabor (nome) VALUES ("Filé com Cheddar");  
INSERT INTO sabor (nome) VALUES ("4 Queijos");
```

## Setup do Projeto

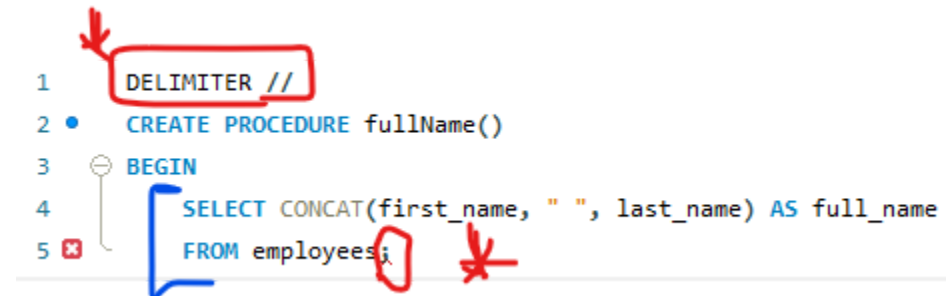
## Store Procedures na Prática

### O que são Store Procedures?

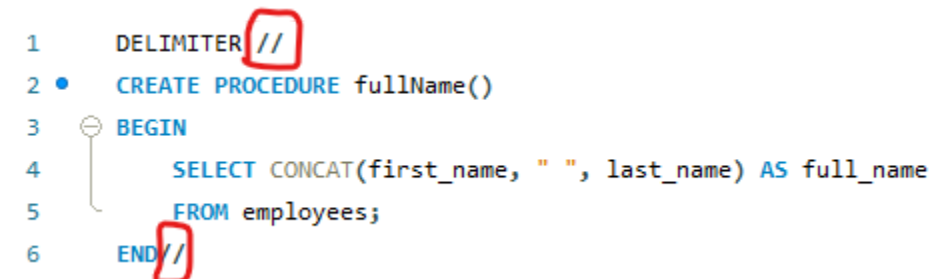
São **queries** reutilisáveis. Semelhante a **funções de linguagem** de programação. Recebe parâmetros. Utiliza um recurso chamado **DELIMITERS** que determina quando começa e termina uma procedure.

## Alterando Delimitadores

Usa-se o delimitador “;”. Cada query utiliza ele, para indicar que é o fim dela. Para utilizar procedures é necessário **modificar o delimiter padrão**. Dessa forma, **não** será executada a query com o **Delimitador Padrão**, ou seja, dentro da Procedure ela **será finalizada** mas apenas executada com a CALL posteriormente.



```
1 DELIMITER //
2 • CREATE PROCEDURE fullName()
3 BEGIN
4     SELECT CONCAT(first_name, " ", last_name) AS full_name
5     FROM employees;
```



```
1 DELIMITER //
2 • CREATE PROCEDURE fullName()
3 BEGIN
4     SELECT CONCAT(first_name, " ", last_name) AS full_name
5     FROM employees;
6 END//
```

## Mais Utilizados

\$ ou //.



Como fazer a troca?

The screenshot shows a SQL IDE interface with a Navigator pane on the left, a SQL File editor in the center, and an Output pane at the bottom.

**Navigator:** Schemas: banco (Tables, Views, Stored Procedures, Functions), constraints, employees (Tables, Views, Stored Procedures, Functions), relacoes (Tables), clientes.

**SQL File 2\*:**

```
1 SELECT *
2 FROM salaries;
```

*Handwritten note: Tradicional*

**Result Grid:**

	emp_no	salary	from_date	to_date
▶	10001	60117	1986-06-26	1987-06-26
	10001	62102	1987-06-26	1988-06-25
	10001	66074	1988-06-25	1989-06-25
	10001	66596	1989-06-25	1990-06-25

**Output:**

Action Output

#	Time	Action	Message
✓ 1	07:35:12	USE employees	0 row(s) affected
✓ 2	07:35:46	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned
✓ 3	07:39:04	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned
✗ 4	07:39:16	SELECT * FROM salaries//	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '//' at line 2
✗ 5	07:39:20	SELECT * FROM salaries//	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '//' at line 2

**SQL Editor:**

```
1 DELIMITER //
2 SELECT *
3 FROM salaries//
```

*Handwritten note: OK ✓*

**Result Grid:**

	emp_no	salary	from_date	to_date
▶	10001	60117	1986-06-26	1987-06-26
	10001	62102	1987-06-26	1988-06-25
	10001	66074	1988-06-25	1989-06-25
	10001	66596	1989-06-25	1990-06-25
	10001	66961	1990-06-25	1991-06-25

**Output:**

Action Output

#	Time	Action	Message
✓ 1	07:35:12	USE employees	0 row(s) affected
✓ 2	07:35:46	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned
✓ 3	07:39:04	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned
✗ 4	07:39:16	SELECT * FROM salaries//	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '//' at line 2
✗ 5	07:39:20	SELECT * FROM salaries//	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '//' at line 2
✓ 6	07:40:12	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned

The screenshot shows a database management tool interface. At the top, a SQL query is entered in a text area:

```
1 DELIMITER $$
2 SELECT *
3 FROM salaries$$
```

Below the query, a "Result Grid" displays the results of the query. The grid has four columns: `emp_no`, `salary`, `from_date`, and `to_date`. The results are as follows:

emp_no	salary	from_date	to_date
10001	60117	1986-06-26	1987-06-26
10001	62102	1987-06-26	1988-06-25
10001	66074	1988-06-25	1989-06-25
10001	66596	1989-06-25	1990-06-25
10001	66961	1990-06-25	1991-06-25

Below the result grid, an "Output" section shows a log of actions and messages. The log is as follows:

#	Time	Action	Message
1	07:35:12	USE employees	0 row(s) affected
2	07:35:46	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned
3	07:39:04	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned
4	07:39:16	SELECT * FROM salaries//	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '//' at line 2
5	07:39:20	SELECT * FROM salaries//	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '//' at line 2
6	07:40:12	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned
7	07:41:09	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned

## Criando uma Procedure

Depois de **alterar DELIMITER**

**BEGIN**

Inicia a Procedure.

**END**

Finaliza Procedure.

**Query**

Criar a consulta que será repetida com a Procedure.

**CALL**

Chamar a Procedure.

## Criando uma Procedure Simples

### Mudar Delimitador

```
1 DELIMITER //
```

### Criar

CREATE PROCEDURE nomeProcedure()

```
1 DELIMITER //  
2 ✖ CREATE PROCEDURE fullName()
```

### Iniciar Instrução

```
1 DELIMITER //  
2 • CREATE PROCEDURE fullName()  
3 ✖ BEGIN
```

### Escrever Query

```
1 DELIMITER //  
2 • CREATE PROCEDURE fullName()  
3 BEGIN  
4 SELECT CONCAT(first_name, " ", last_name) AS full_name  
5 ✖ FROM employees;
```

### Finalizar Instrução

```
1 DELIMITER //  
2 • CREATE PROCEDURE fullName()  
3 BEGIN  
4 SELECT CONCAT(first_name, " ", last_name) AS full_name  
5 FROM employees;  
6 END//
```

## Salvar Procedure

Ctrl  
+  
Enter

```
1 DELIMITER //
2 • CREATE PROCEDURE fullName()
3 BEGIN
4     SELECT CONCAT(first_name, " ", last_name) AS full_name
5     FROM employees;
6 END//
```

Output

Action Output

#	Time	Action	Message
✓ 1	07:35:12	USE employees	0 row(s) affected
✓ 2	07:35:46	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned
✓ 3	07:39:04	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned
✗ 4	07:39:16	SELECT * FROM salaries//	Error Code: 1064. You have an error
✗ 5	07:39:20	SELECT * FROM salaries//	Error Code: 1064. You have an error
✓ 6	07:40:12	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned
✓ 7	07:41:09	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned
✓ 8	07:56:51	CREATE PROCEDURE fullName() BE...	0 row(s) affected

## CALL Procedure

```
1 DELIMITER //
2 • CREATE PROCEDURE fullName()
3 BEGIN
4     SELECT CONCAT(first_name, " ", last_name) AS full_name
5     FROM employees;
6 END//
7
8 • CALL fullName()
```

Result Grid Filter Rows: Export: Wrap Cell Contents: Fetch rows:

full\_name  
Georgi Facello  
Bezalel Simmel  
Parto Bamford  
Christian Koblick  
Kyoichi Maliniak  
Anneke Preusig  
Tzvetan Zhelevski  
Saniya Kalloufi  
Sumant Peac  
Duangkraew Piveteau  
Mary Sluis  
Patrioio Bridgland  
Eberhardt Terkki  
Berni Genin

Result 5

Output

Action Output

#	Time	Action	Message
✓ 1	07:35:12	USE employees	0 row(s) affected
✓ 2	07:35:46	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned
✓ 3	07:39:04	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned
✗ 4	07:39:16	SELECT * FROM salaries//	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '//' at line 2
✗ 5	07:39:20	SELECT * FROM salaries//	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '//' at line 2
✓ 6	07:40:12	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned
✓ 7	07:41:09	SELECT * FROM salaries LIMIT 0, 1000	1000 row(s) returned
✓ 8	07:56:51	CREATE PROCEDURE fullName() BE...	0 row(s) affected
✗ 9	07:58:38	CREATE PROCEDURE fullName() BE...	Error Code: 1304. PROCEDURE fullName already exists
✓ 10	07:59:33	CALL fullName	300024 row(s) returned

## Resgatando todas as Procedures

É possível **chegar as procedures** criadas, utilize o comando **SHOW PROCEDURE STATUS** e veja os **detalhes**.

1 • SHOW PROCEDURE STATUS

Result Grid   Filter Rows:   Export:   Wrap Cell Content:						
Db	Name	Type	Definer	Modified	Created	Securi
employees	fullName	PROCEDURE	root@localhost	2024-07-23 07:56:51	2024-07-23 07:56:51	DEFINE
employees	selectAll	PROCEDURE	root@localhost	2024-07-23 08:07:10	2024-07-23 08:07:10	DEFINE

## Removendo Procedures