

# C.R.U.D.

**Básico de Bancos**

# C.R.U.D. - Criar Ler Atualizar e Apagar

CRUD é um acrônimo em inglês que significa “**Create, Read, Update, Delete**” (ou, em português, “Criar, Ler, Atualizar, Excluir”).

O CRUD corresponde às ações básicas que podem ser realizadas em um banco de dados:

- **Create (criar):** criar um novo registro em uma tabela do banco de dados.
- **Read (ler):** ler os dados de um ou mais registros existentes em uma tabela do banco de dados.
- **Update (atualizar):** atualizar os dados de um registro existente em uma tabela do banco de dados.
- **Delete (excluir):** excluir um registro existente em uma tabela do banco de dados.

# Criando uma tabela no PostgreSQL

**Um exemplo completo**

# Camadas do banco de dados: esquema

Antes de descobrir como usar esquemas, você precisa saber qual é a finalidade de um esquema. Para entender isso, primeiro dê uma olhada em como o PostgreSQL está estruturado:

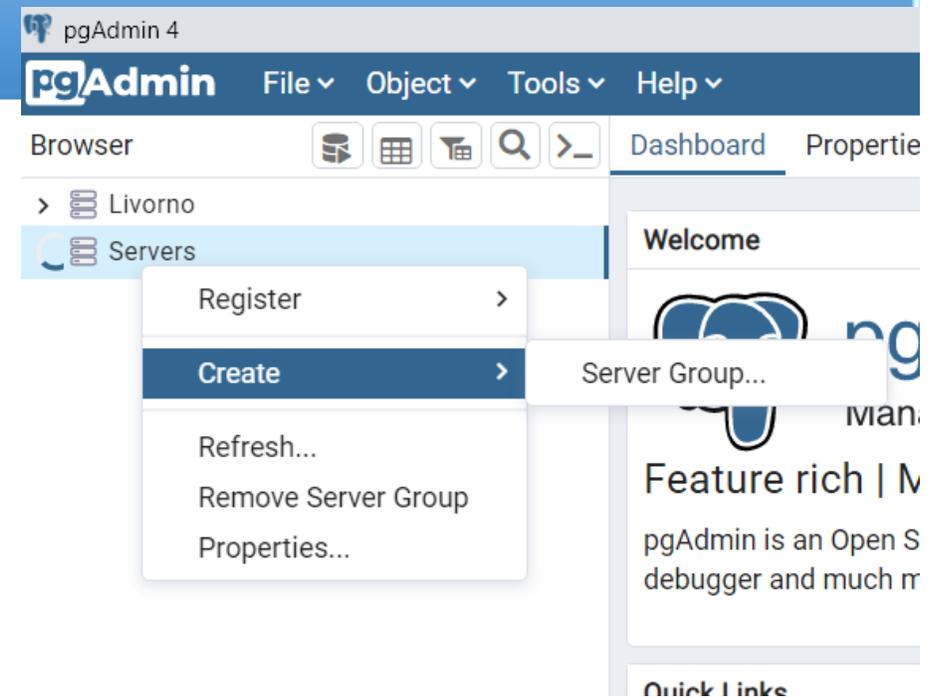
- Instance - Instância
- Database - Base de dados
- Schema - Esquema
- Table
- Row - Linha

# Para criar uma tabela no PostgreSQL, pode-se executar os seguintes passos:

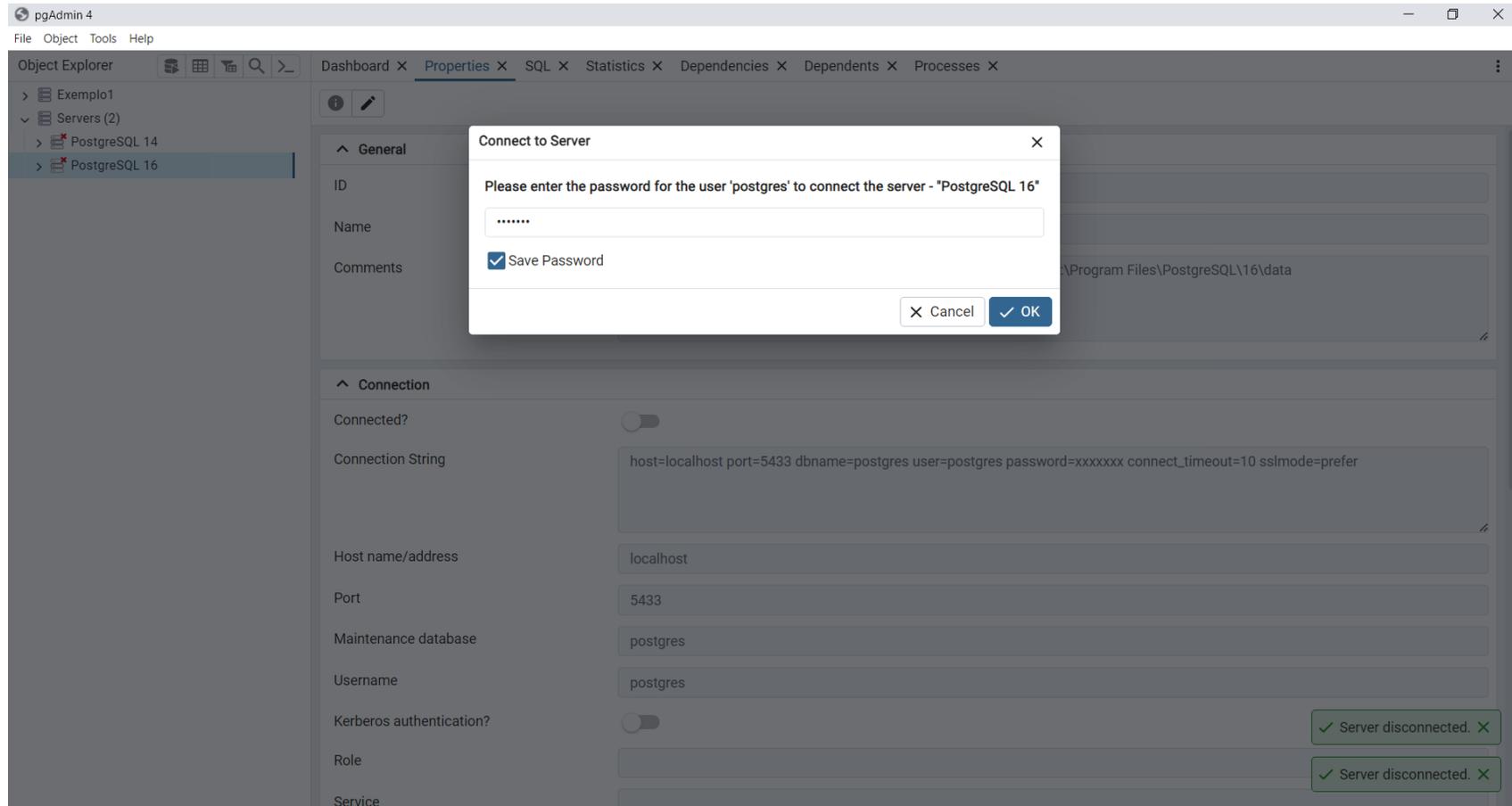
Certifique-se de que você esteja conectado ao banco de dados desejado.

Ou crie um Servidor para estar conectado

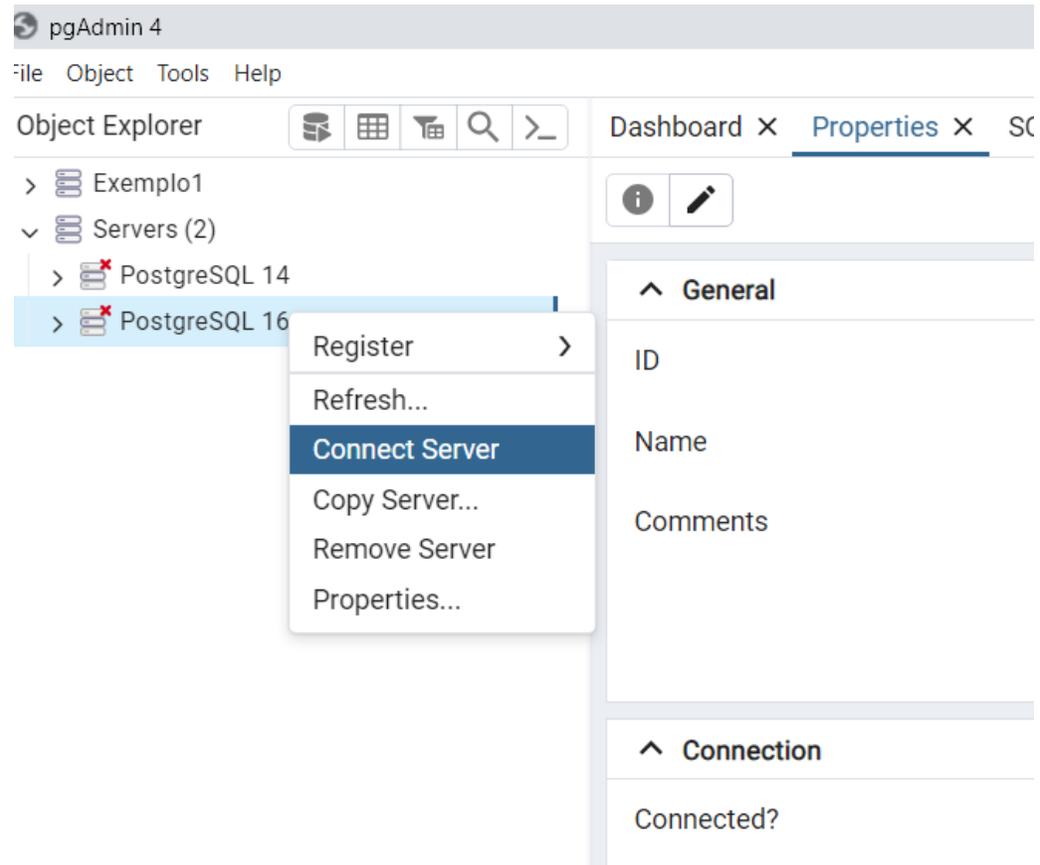
## Exemplo geral



# Instância - conecte ao servidor



# Conete a base ao servidor:



# O objetivo de um esquema

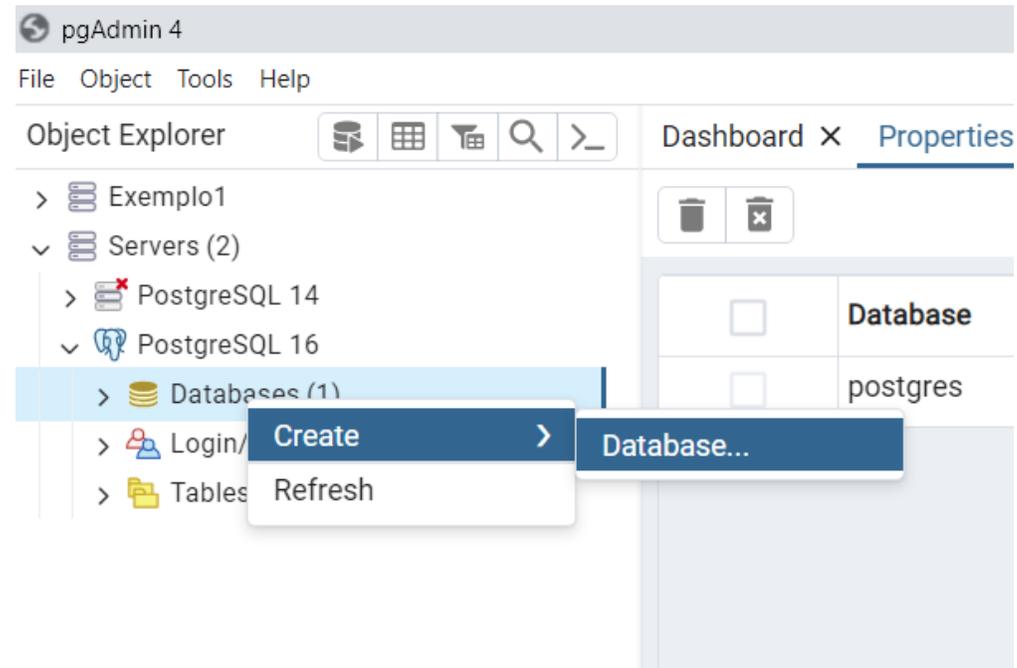
Na realidade é a isso que você se conecta: no PostgreSQL uma conexão está sempre vinculada a um banco de dados dentro de uma instância, o que acontece logo no início, logo após a autenticação do usuário.

# O objetivo de um esquema

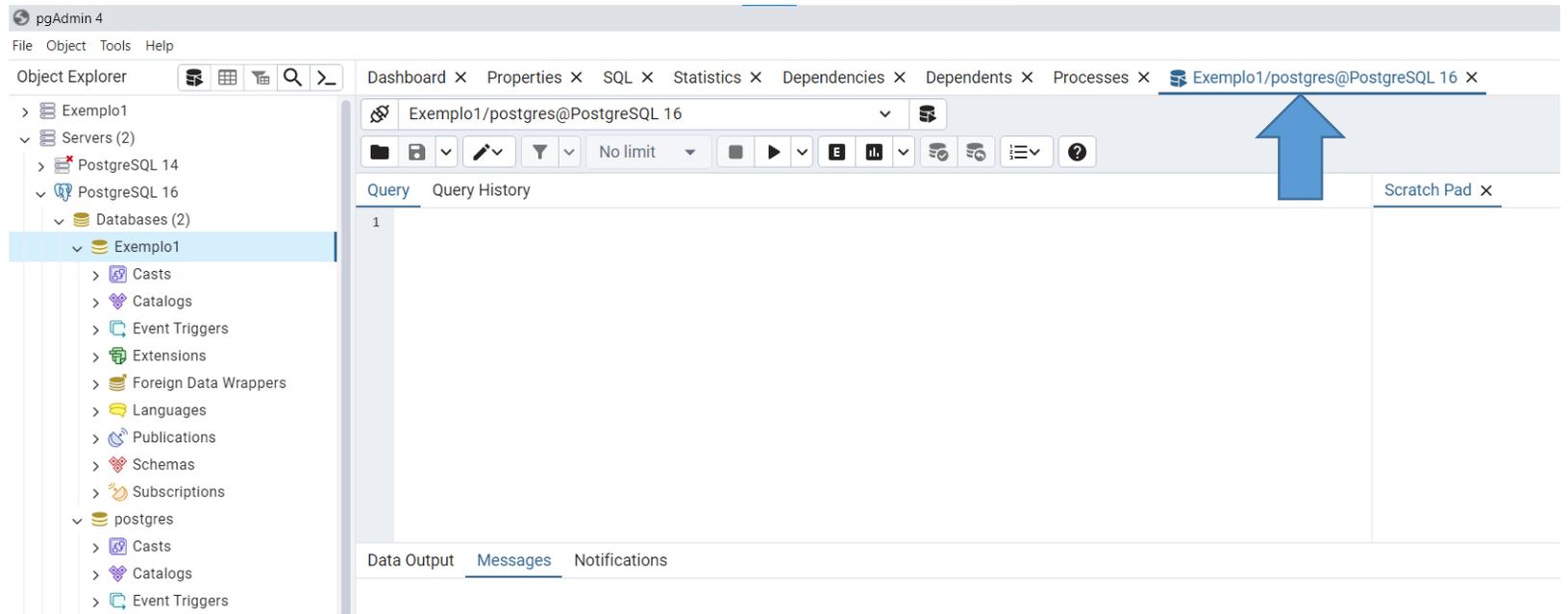
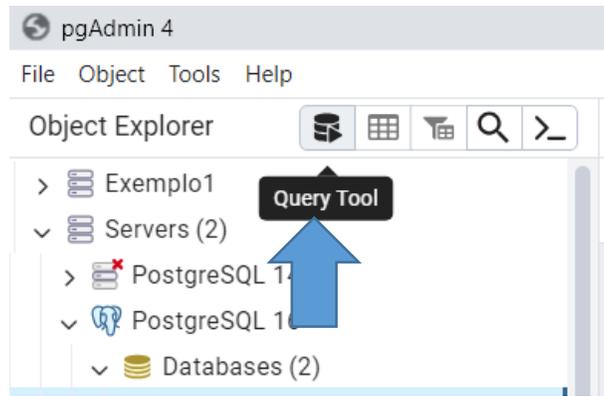
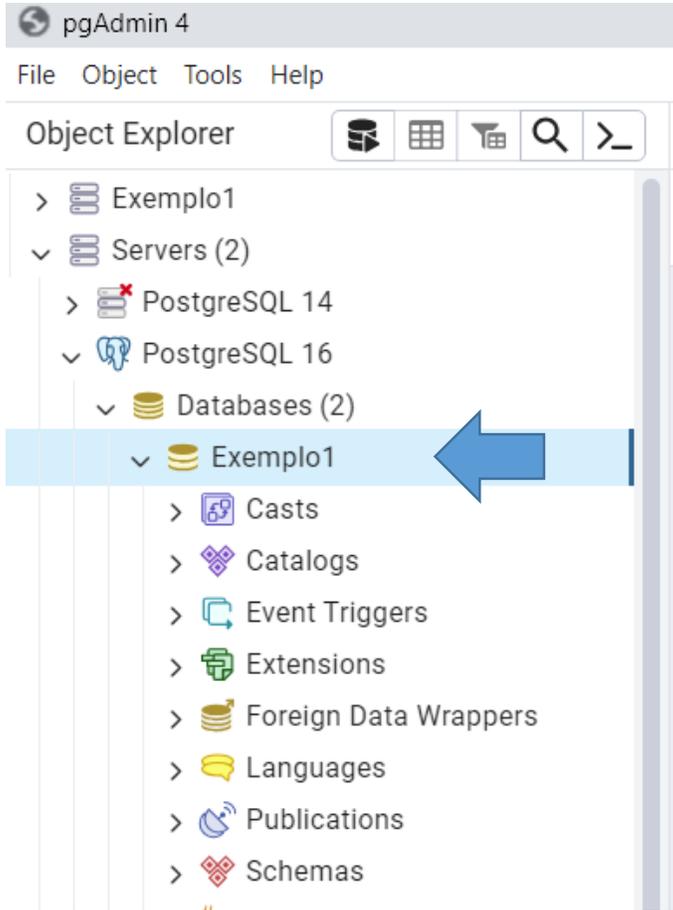
Uma “instância” é basicamente o que você inicia ao implantar o PostgreSQL.

A próxima camada é um banco de dados.

```
CREATE DATABASE "Exemplo1"  
WITH  
OWNER = postgres  
ENCODING = 'UTF8'  
LOCALE_PROVIDER = 'libc'  
CONNECTION LIMIT = -1  
IS_TEMPLATE = False;
```



# Resultado:



# Tabelas de grupo de esquemas

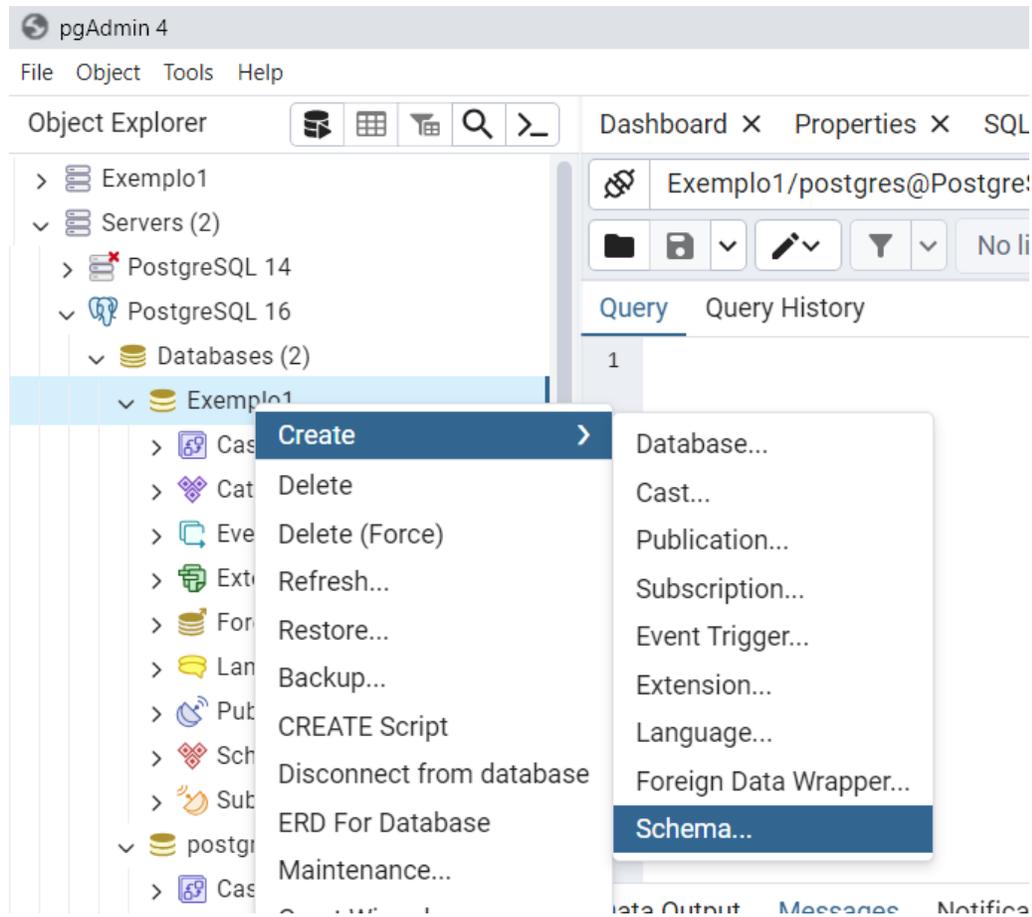
O que é importante é a próxima camada abaixo, entre bancos de dados e tabelas: Esquemas.

Basicamente, os esquemas são uma forma de agrupar tabelas.

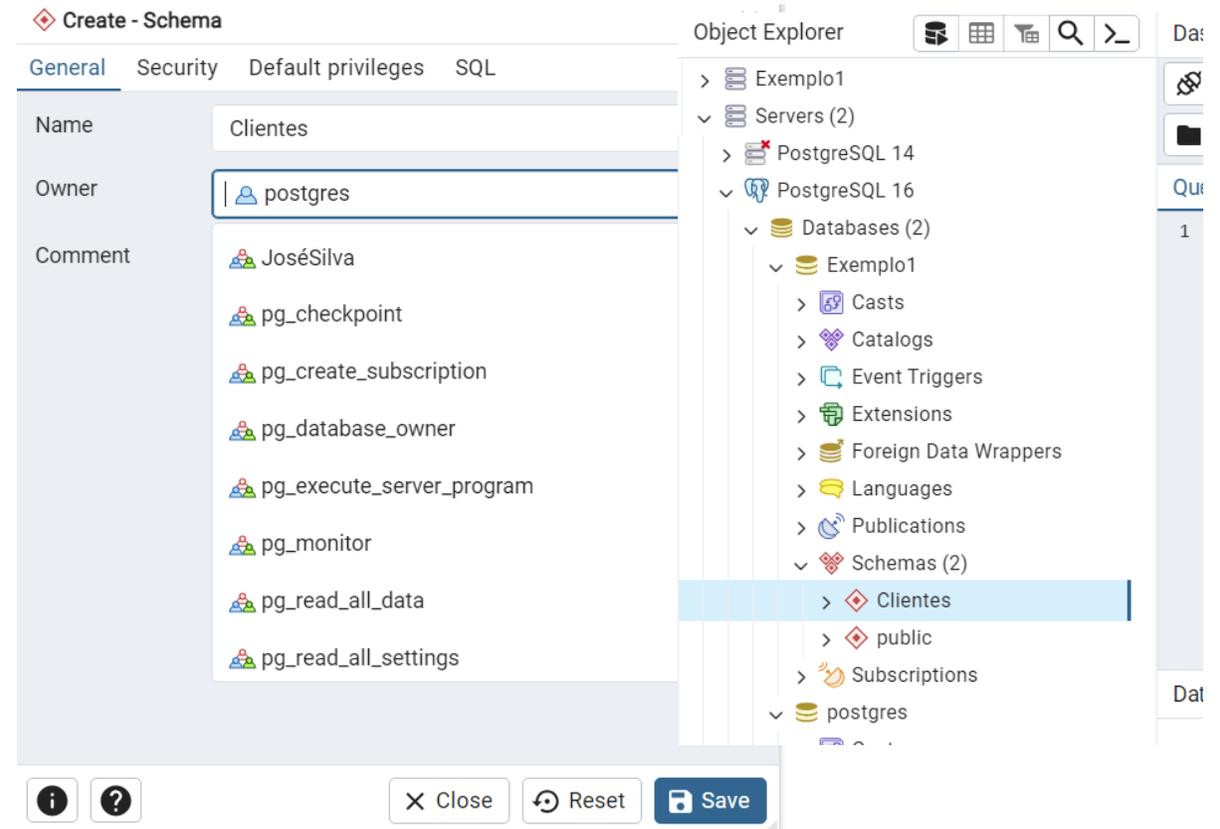
Vamos supor que haja uma estrutura de dados bastante grande: ter 500 tabelas em um só lugar é certamente mais difícil de gerenciar e entender do que ter 10 grupos contendo 50 tabelas cada.

É simplesmente como organizar imagens: você não colocaria todas elas na mesma pasta, mas sim agrupá-las por ano, local, etc. A mesma lógica pode ser aplicada às tabelas.

# Criando schemas



**CREATE SCHEMA Clientes  
AUTHORIZATION postgres;**

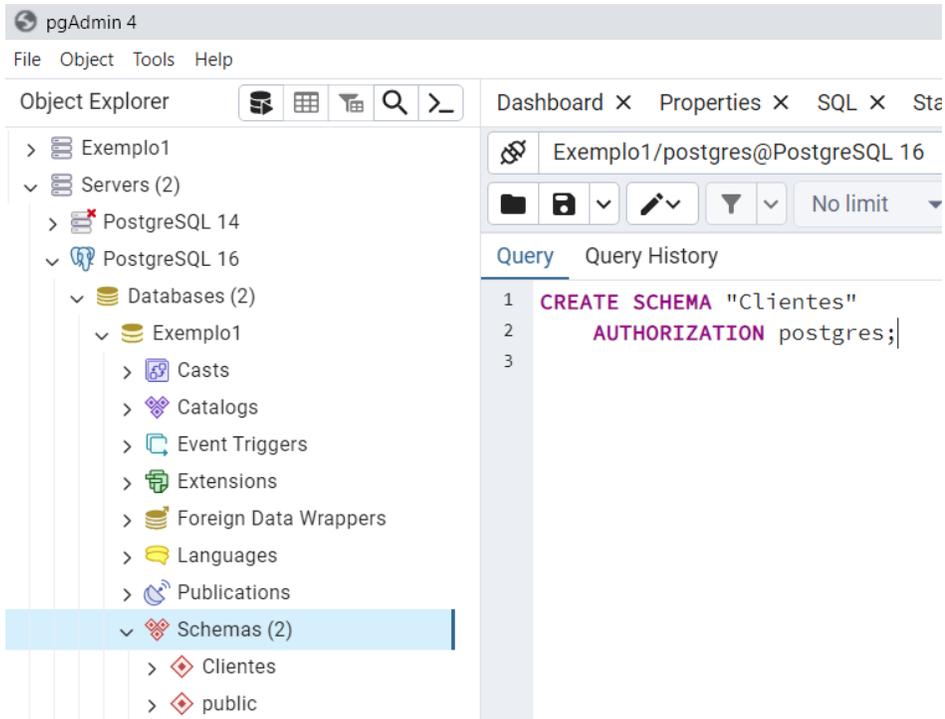


Use o comando CREATE TABLE para criar uma nova tabela.

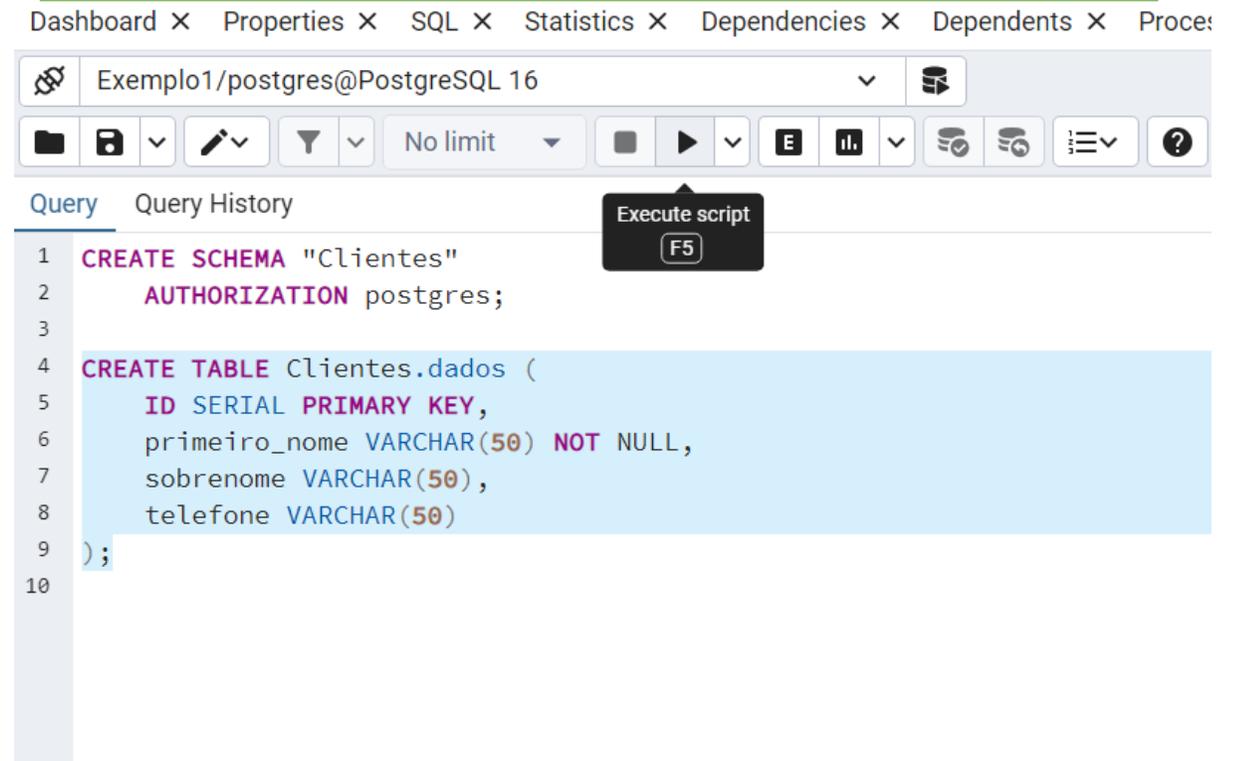
## A sintaxe básica é a seguinte:

```
CREATE TABLE nome_da_tabela (  
    nome_da_coluna1 tipo_de_dado_da_coluna1,  
    nome_da_coluna2 tipo_de_dado_da_coluna2,  
    ...  
    nome_da_colunaN tipo_de_dado_da_colunaN  
);
```

# Criando tabelas nesse esquema (schema)



```
CREATE TABLE Clientes.dados (  
    ID SERIAL PRIMARY KEY,  
    primeiro_nome VARCHAR(50) NOT NULL,  
    sobrenome VARCHAR(50),  
    telefone VARCHAR(50)  
);
```



Query Query History

```
1 CREATE SCHEMA Clientes
2     AUTHORIZATION postgres;
3
4 CREATE TABLE Clientes.dados (
5     ID SERIAL PRIMARY KEY,
6     primeiro_nome VARCHAR(50) NOT NULL,
7     sobrenome VARCHAR(50),
8     telefone VARCHAR(50)
9 );
10
```

Data Output Messages Notifications

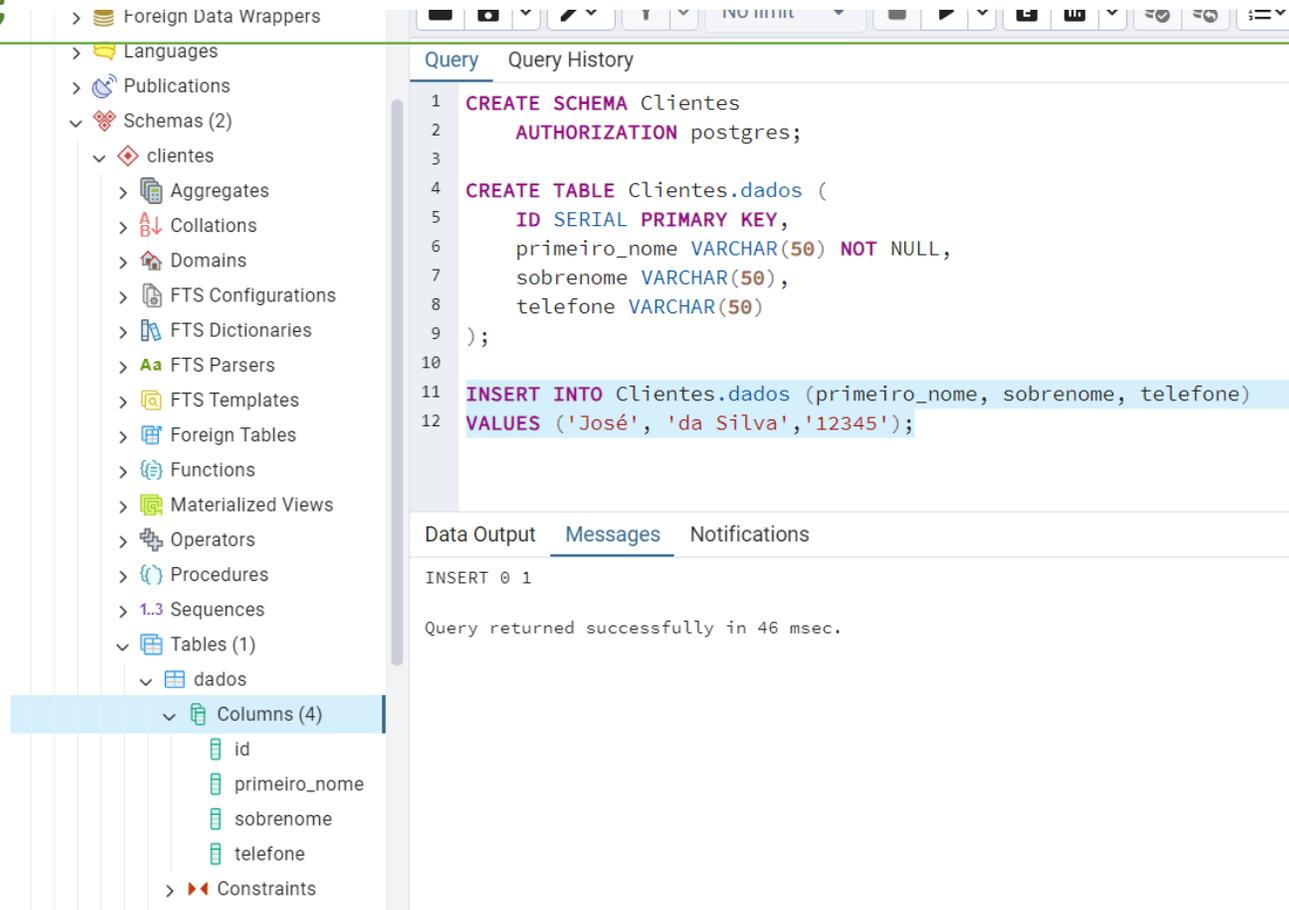
CREATE TABLE

Query returned successfully in 160 msec.

- Exemplo1
  - Casts
  - Catalogs
  - Event Triggers
  - Extensions
  - Foreign Data Wrappers
  - Languages
  - Publications
  - Schemas (2)
    - clientes
      - Aggregates
      - Collations
      - Domains
      - FTS Configurations
      - FTS Dictionaries
      - FTS Parsers
      - FTS Templates
      - Foreign Tables
      - Functions
      - Materialized Views
      - Operators
      - Procedures
      - 1.3 Sequences
      - Tables (1)
        - dados
          - Columns
          - Constraints
          - Indexes
          - RLS Policies

# Inserindo dados:

```
INSERT INTO Clientes.dados (primeiro_nome, sobrenome, telefone)
VALUES ('José', 'da Silva', '12345');
```



The screenshot shows a database management interface. On the left is a tree view of the database structure. The 'clientes' schema is expanded, showing a table named 'dados'. The 'dados' table is further expanded to show its columns: 'id', 'primeiro\_nome', 'sobrenome', and 'telefone'. On the right, the 'Query' pane contains the following SQL code:

```
1 CREATE SCHEMA Clientes
2   AUTHORIZATION postgres;
3
4 CREATE TABLE Clientes.dados (
5   ID SERIAL PRIMARY KEY,
6   primeiro_nome VARCHAR(50) NOT NULL,
7   sobrenome VARCHAR(50),
8   telefone VARCHAR(50)
9 );
10
11 INSERT INTO Clientes.dados (primeiro_nome, sobrenome, telefone)
12 VALUES ('José', 'da Silva', '12345');
```

Below the query editor, the 'Messages' pane shows the execution result:

```
INSERT 0 1
Query returned successfully in 46 msec.
```

```
INSERT INTO Clientes.dados (primeiro_nome, sobrenome, telefone)
VALUES ('Mario', 'Livio','54321'),
       ('Jonas', 'Lavado','23456');
;
```

- > FTS Configurations
- > FTS Dictionaries
- > Aa FTS Parsers
- > FTS Templates
- > Foreign Tables
- > Functions
- > Materialized Views
- > Operators
- > Procedures
- > 1.3 Sequences
- ✓ Tables (1)
  - ✓ dados
    - Columns (4)
      - id
      - primeiro\_nome
      - sobrenome
      - telefone

```
14 SELECT * FROM Clientes.dados;
15
16 INSERT INTO Clientes.dados (primeiro_nome, sobrenome, telefone)
17 VALUES ('Mario', 'Livio','54321'),
18          ('Jonas', 'Lavado','23456');
19 ;
20
```

Data Output   Messages   Notifications

INSERT 0 2

Query returned successfully in 81 msec.

# Lendo dados de uma tabela no PostgreSQL

# SELECT

Para realizar a leitura de dados em uma tabela no PostgreSQL, você pode utilizar o comando SELECT. A sintaxe básica do comando SELECT é a seguinte:

```
SELECT coluna1, coluna2, ..., colunaN
```

```
FROM nome_da_tabela
```

```
WHERE condição;
```

# Visualizando...

```
10  
11 INSERT INTO Clientes.dados (primeiro_nome, sobrenome, telefone)  
12 VALUES ('José', 'da Silva','12345');  
13  
14 SELECT * FROM Clientes.dados;  
15  
16 INSERT INTO Clientes.dados (primeiro_nome, sobrenome, telefone)  
17 VALUES ('Mario', 'Livio','54321'),  
18         ('Jonas', 'Lavado','23456');  
19 ;  
20
```

Data Output Messages Notifications

	id [PK] integer	primeiro_nome character varying (50)	sobrenome character varying (50)	telefone character varying (50)
1	1	José	da Silva	12345
2	2	Mario	Livio	54321
3	3	Jonas	Lavado	23456

# Apagando dados de uma tabela no PostgreSQL

# DELETE

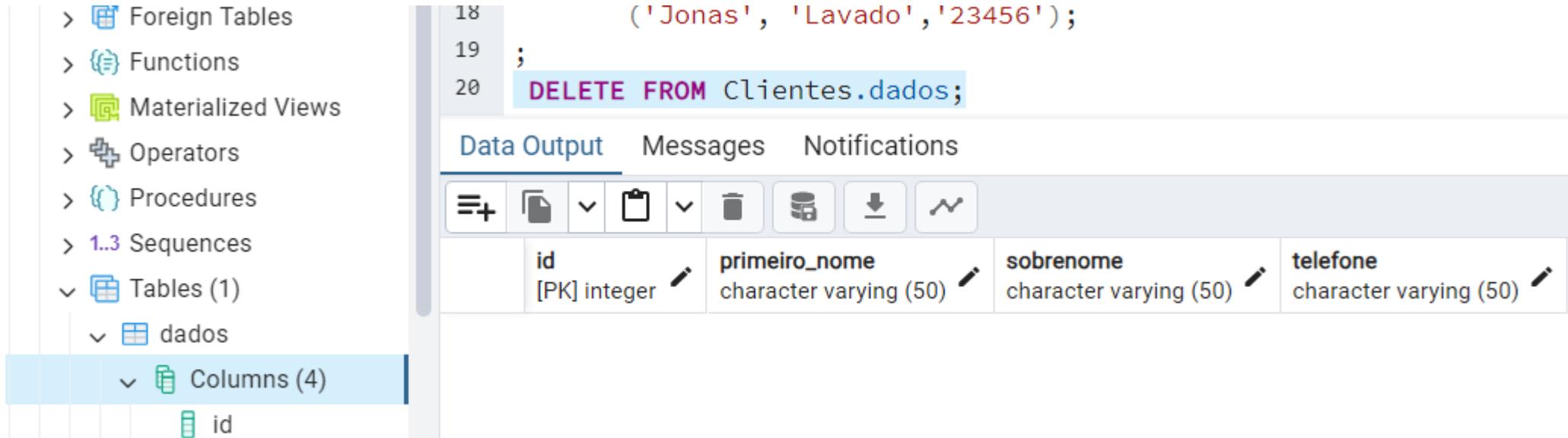
Para apagar dados em uma tabela no PostgreSQL, você pode utilizar o comando DELETE. A sintaxe básica do comando DELETE é a seguinte:

```
DELETE FROM nome_da_tabela
```

```
WHERE condição;
```

# Apagando dados

- DELETE FROM Clientes.dados;



The screenshot shows a database management interface. On the left, a tree view displays the database structure, with 'Tables (1)' expanded to show 'dados', and 'Columns (4)' expanded to show 'id'. The main area displays a SQL query with the following lines:

```
18      ('Jonas', 'Lavado', '23456');  
19 ;  
20 DELETE FROM Clientes.dados;
```

Below the query, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table schema with the following columns:

id	primeiro_nome	sobrenome	telefone
[PK] integer	character varying (50)	character varying (50)	character varying (50)

# Apagando apenas uma linha:

```
DELETE FROM Clientes.dados WHERE id=5;
```

```
22 | DELETE FROM Clientes.dados WHERE id=5;
```

Data Output Messages Notifications

	id [PK] integer	primeiro_nome character varying (50)	sobrenome character varying (50)	telefone character varying (50)
1	4	José	da Silva	12345
2	6	Jonas	Lavado	23456

Preenchendo uma tabela com  
linhas

# Seleccionando (visualizando) datos

The screenshot shows a database management interface. On the left is a tree view of database objects, including 'Tables (1)' with a sub-entry 'dados' and 'Columns (4)' with a sub-entry 'id'. The main area displays SQL code in a text editor. The code includes a table definition for 'Clientes.dados' with columns 'primeiro\_nome', 'sobrenome', and 'telefone', followed by an 'INSERT INTO' statement and a 'SELECT \* FROM' statement. Below the code editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with the results of the 'SELECT' query. The table has four columns: 'id', 'primeiro\_nome', 'sobrenome', and 'telefone'. The first row contains the values '1', 'José', 'da Silva', and '12345'.

```
7      sobrenome VARCHAR(50),  
8      telefone VARCHAR(50)  
9  );  
10  
11  INSERT INTO Clientes.dados (primeiro_nome, sobrenome, telefone)  
12  VALUES ('José', 'da Silva', '12345');  
13  
14  SELECT * FROM Clientes.dados;
```

Data Output Messages Notifications

	id [PK] integer	primeiro_nome character varying (50)	sobrenome character varying (50)	telefone character varying (50)
1	1	José	da Silva	12345

# Visualizando apenas alguns campos

```
SELECT primeiro_nome, telefone FROM Clientes.dados;
```

The screenshot shows a database query tool interface. At the top, there are two tabs: "Query" and "Query History". The "Query" tab is active, displaying the following SQL query:

```
1 SELECT primeiro_nome, telefone FROM Clientes.dados;  
2  
3  
4
```

Below the query editor, there are three tabs: "Data Output", "Messages", and "Notifications". The "Data Output" tab is active, showing a table of results. Above the table is a toolbar with icons for adding, deleting, and refreshing data.

	primeiro_nome character varying (50) 🔒	telefone character varying (50) 🔒
1	José	12345
2	Jonas	23456

# Mais seleções...

Query Query History

```
2  
3 SELECT * FROM Clientes.dados  
4 WHERE primeiro_nome= 'José' AND sobrenome = 'da Silva';  
5
```

Data Output Messages Notifications

≡+ 📄 ▼ 📋 ▼ 🗑️ 🗄️ ⬇️ 📈

	id [PK] integer 	primeiro_nome character varying (50) 	sobrenome character varying (50) 	telefone character varying (50) 
1	4	José	da Silva	12345

# Ordenando...

```
SELECT * FROM Clientes.dados  
ORDER BY primeiro_nome;
```

Query Query History

```
4 WHERE primeiro_nome= 'Jose' AND sobrenome = 'da Silva';  
5  
6 SELECT * FROM Clientes.dados  
7 ORDER BY primeiro_nome;
```

Data Output Messages Notifications

	id [PK] integer	primeiro_nome character varying (50)	sobrenome character varying (50)	telefone character varying (50)
1	6	Jonas	Louvado	23456
2	4	José	da Silva	12345

# Selecionado variáveis internas

```
SELECT current_date;
```

```
9 SELECT current_date;
```

Data Output Messages Notifications

	current_date	
	date	🔒
1	2024-05-07	

```
SELECT version();
```

```
10  
11 SELECT version();
```

Data Output Messages Notifications

	version	
	text	🔒
1	PostgreSQL 16.2, compiled by Visual C++ build 1937, 64-...	

SELECT 2 + 2;

```
13 SELECT 2 + 2;
```

Data Output Messages Notifications

	?column? integer
1	4

SELECT current\_date +2;

```
19 SELECT current_date +2;
```

Data Output Messages Notifications

	?column? date
1	2024-05-09

# Atualizando dados de uma tabela no PostgreSQL

# UPDATE

Para modificar dados em uma tabela no PostgreSQL, você pode utilizar o comando UPDATE. A sintaxe básica do comando UPDATE é a seguinte:

```
UPDATE nome_da_tabela
```

```
SET coluna1 = valor1, coluna2 = valor2, ..., colunaN = valorN
```

```
WHERE condição;
```

# Atualizando (alterando) dados.

```
UPDATE Clientes.dados SET sobrenome = 'Louvado' WHERE id=6;
```

```
24 UPDATE Clientes.dados SET sobrenome = 'Louvado' WHERE id=6;
```

Data Output Messages Notifications

	id [PK] integer	primeiro_nome character varying (50)	sobrenome character varying (50)	telefone character varying (50)
1	4	José	da Silva	12345
2	6	Jonas	Louvado	23456

# Referencia:

- <https://www.postgresql.org/files/documentation/pdf/15/postgresql-15-A4.pdf#page=45&zoom=100,96,96>

FIM