

**Banco de dados**  
SQL

**Luiza**  
<CODE>

# Ementa:

1. A diferença entre SGBD e banco de dados
2. O que é banco de dados relacional
3. Instalando e configurando o PostgreSQL (DBeaver)
4. Modelagem Banco de Dados Relacional
5. Relacionamento entre tabelas
6. SQL Nativo (DDL, DML, DQL) - Sintaxe
7. Manipulação de dados SQL (DQL) - Sintaxe
8. SQL no PostgreSQL - Prática



# 1. A diferença entre SGBD e banco de dados

## 1.1 Banco de dados

**Banco de dados** ou base de dados pode simplesmente ser definida como um local – físico ou virtual – onde se **armazenam** dados de forma **organizada** e **indexada**. Percebam que não estamos nem na parte de informática e sim na parte física e analógica.

Com isso em mente, podemos citar como exemplo de banco de dados, o bloco de notas, o documento no word, uma planilha, um caderno e até mesmo um baú. A internet também pode ser considerada um banco de dados.

Um banco de dados tem as seguintes propriedades implícitas:

- representa algum aspecto do mundo real;
- é uma coleção de dados logicamente coerente;
- é projetado, construído e *populado* para um propósito específico.

# 1. A diferença entre SGBD e banco de dados

## 1.2 Sistema de Gerenciamento de Banco de Dados (SGBD)

Um **SGBD** é um software responsável por tornar o banco de dados gerenciável permitir que ele seja manipulado, ele ainda não é o software que vai ao usuário final e sim o que o DBA ou Database Admin irá utilizar. Sua principal função é facilitar a interface com o banco e do DBA. Um SGBD é um **software de propósito geral** que facilita o processo de definir, construir, manipular e compartilhar bancos de dados entre vários usuários e aplicações.

Segue uma lista de **SGBDs** mais utilizados:

- Oracle
- MySQL
- SQL Server
- PostgreSQL
- MongoDB

## 2. O que é banco de dados relacional

### 2.1 Banco de Dados Relacional (SQL)

Um banco de dados relacional é um banco de dados que modela os dados de uma forma que eles sejam percebidos pelo usuário como tabelas, ou mais formalmente relações.

#### Exemplo de um BD Relacional

Empregado

NumEmp	NomeEmp	Salário	Dept
032	J Silva	380	21
074	M Reis	400	25
089	C Melo	520	28
092	R Silva	480	25
112	R Pinto	390	21
121	V Simão	905	28
130	J Neves	640	28

Departamento

NumDept	NomeDept	Ramal
21	Pessoal	142
25	Financeiro	143
28	Técnico	144

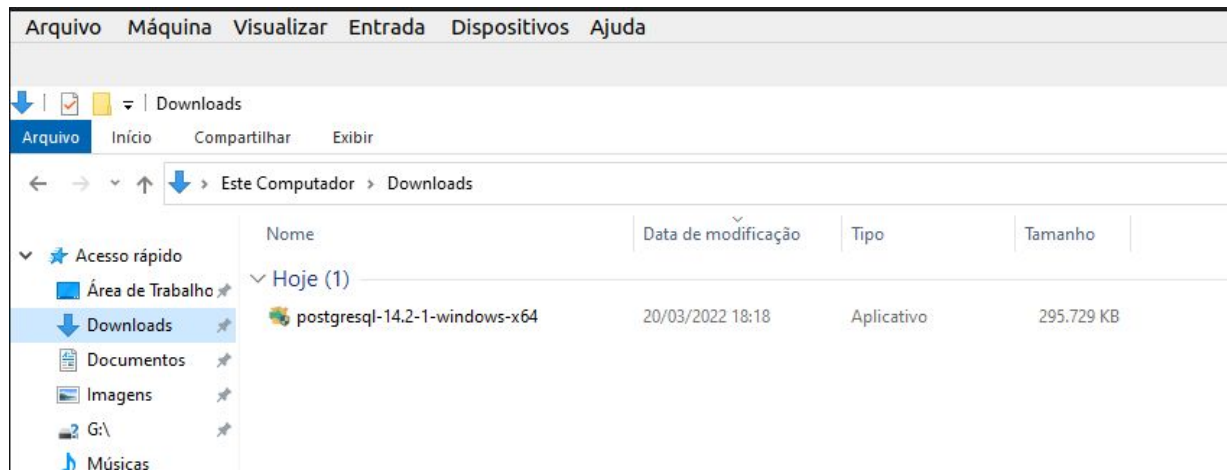
## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.1 Instalação do PostgreSQL no Windows

Realize o download do instalador no link

<https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>

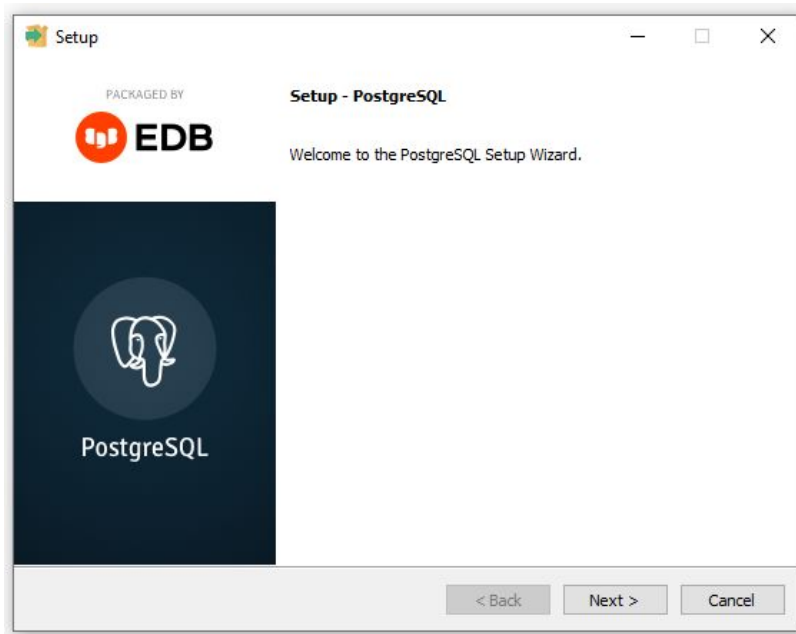
Após a finalização do download, execute o instalador:



## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.1 Instalação do PostgreSQL no Windows

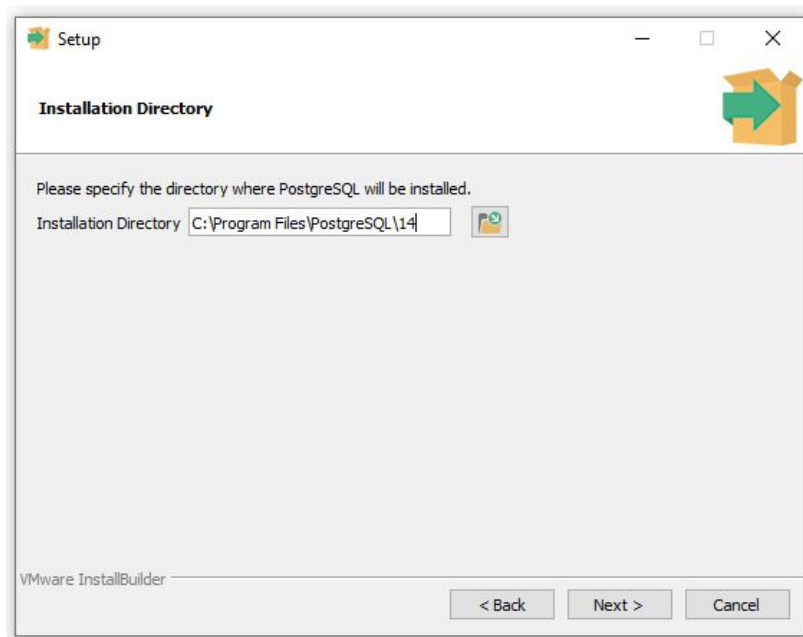
Siga para o próximo passo através da opção “Next”:



## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.1 Instalação do PostgreSQL no Windows

Siga para o próximo passo através da opção “Next”:

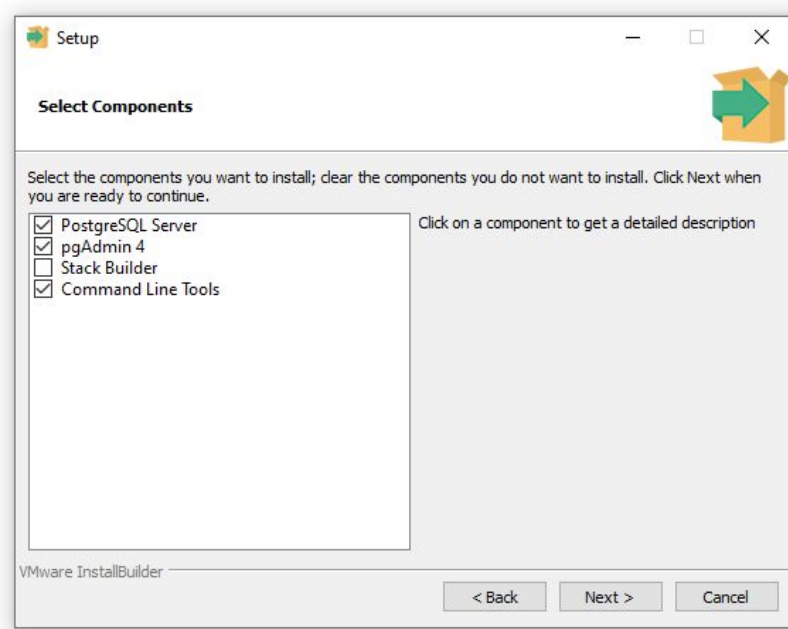




## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.1 Instalação do PostgreSQL no Windows

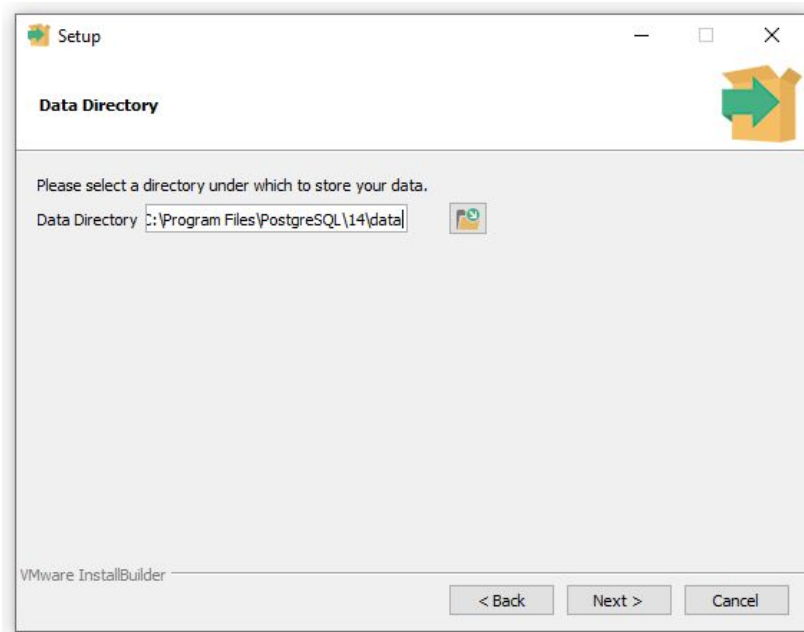
Desmarque o componente “Stack Builder” que não será necessário. Siga para o próximo passo através da opção “Next”:



## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.1 Instalação do PostgreSQL no Windows

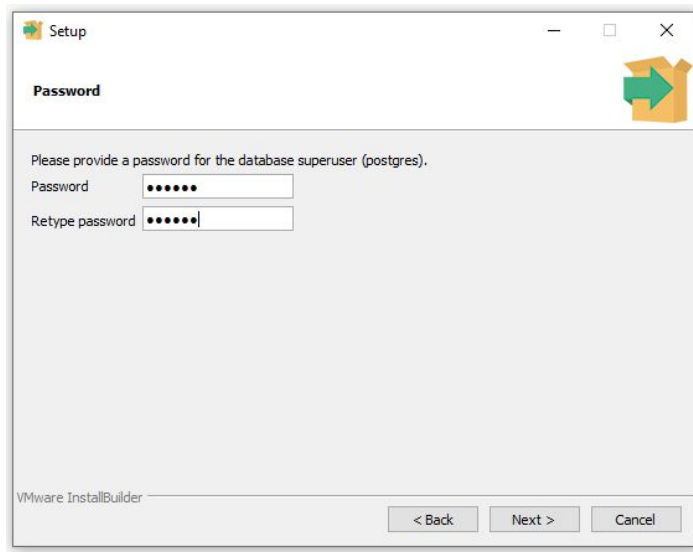
Nesta etapa podemos indicar o diretório(pasta) onde será armazenado os dados. No exemplo abaixo iremos manter o caminho sugerido pelo instalador. Siga para o próximo passo através da opção “Next”:



## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.1 Instalação do PostgreSQL no Windows

Aqui definimos uma senha de administrador do banco de dados. É extremamente importante manter essa senha de uma forma segura. Siga para o próximo passo através da opção “Next”:

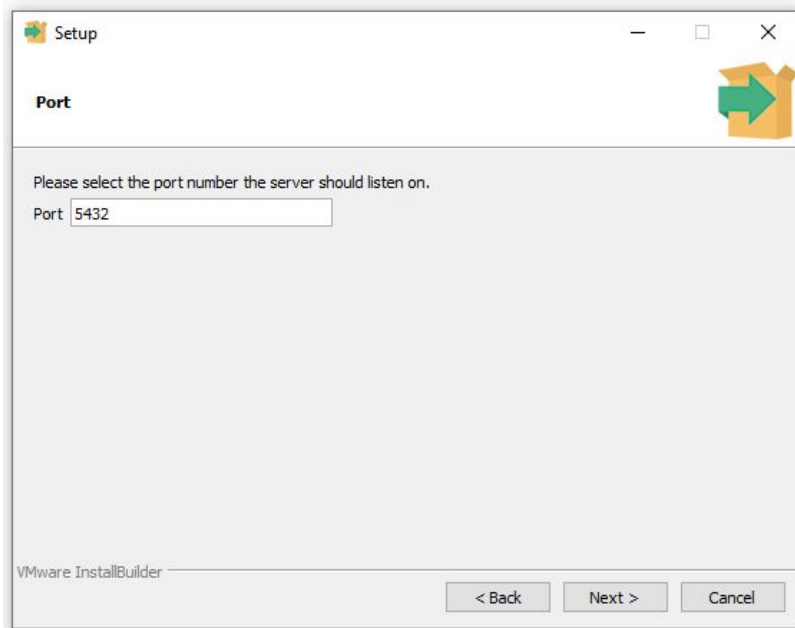


## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.1 Instalação do PostgreSQL no Windows

Neste momento é configurado a porta que o servidor irá utilizar. Iremos manter a configuração padrão.

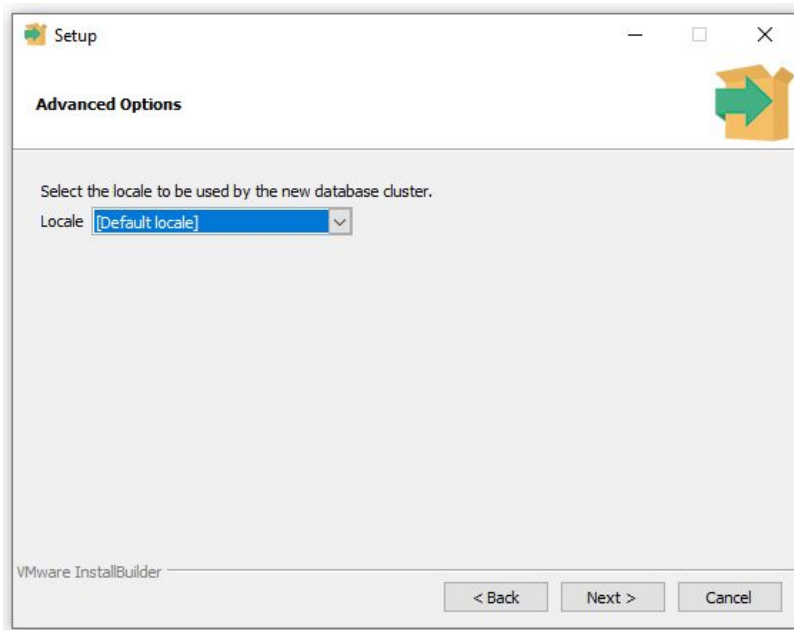
Siga para o próximo passo através da opção “Next”:



## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.1 Instalação do PostgreSQL no Windows

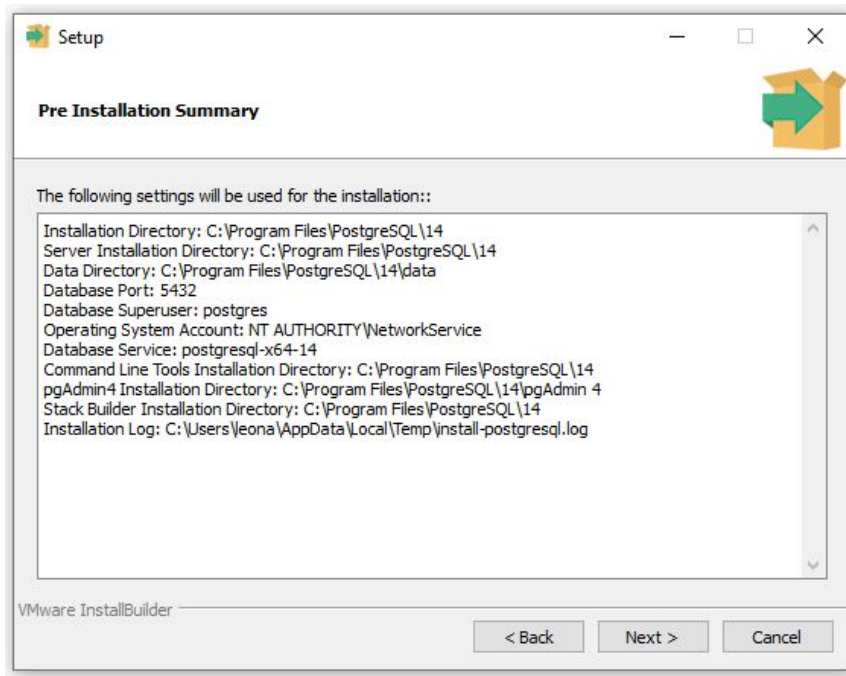
Iremos utilizar a configuração de local default.  
Siga para o próximo passo através da opção “Next”:



## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.1 Instalação do PostgreSQL no Windows

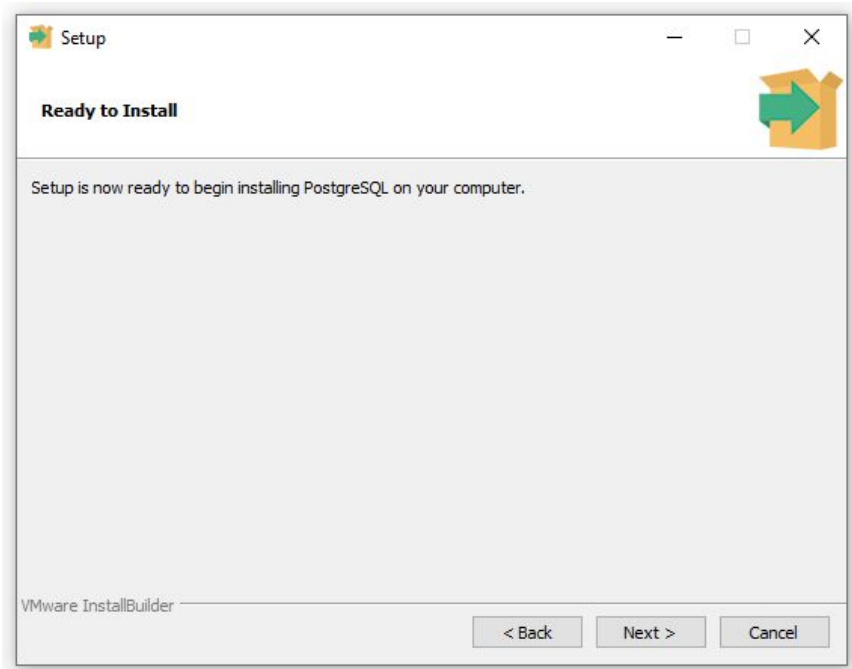
Siga para o próximo passo através da opção “Next”:



## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.1 Instalação do PostgreSQL no Windows

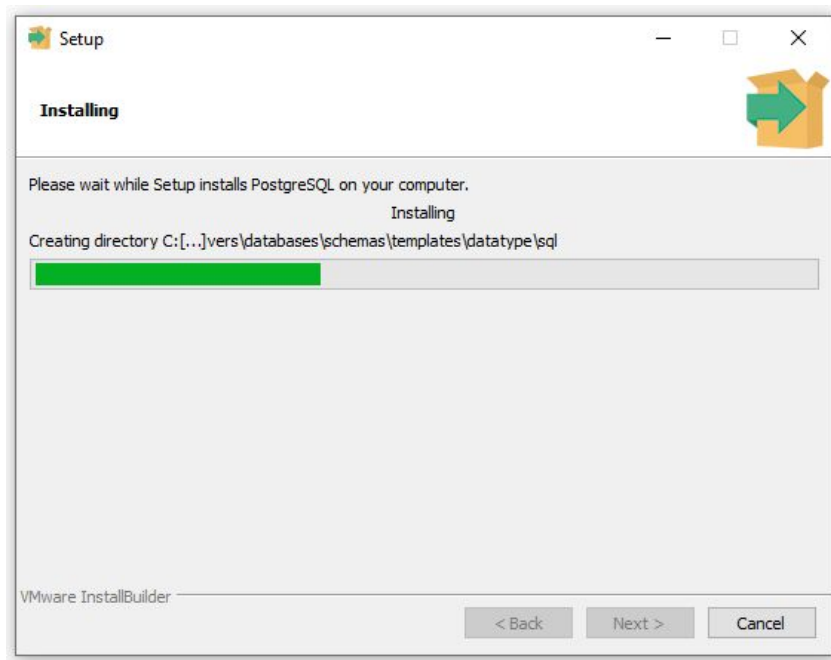
Siga para o próximo passo através da opção “Next”:



## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.1 Instalação do PostgreSQL no Windows

Aguarde a instalação finalizar.

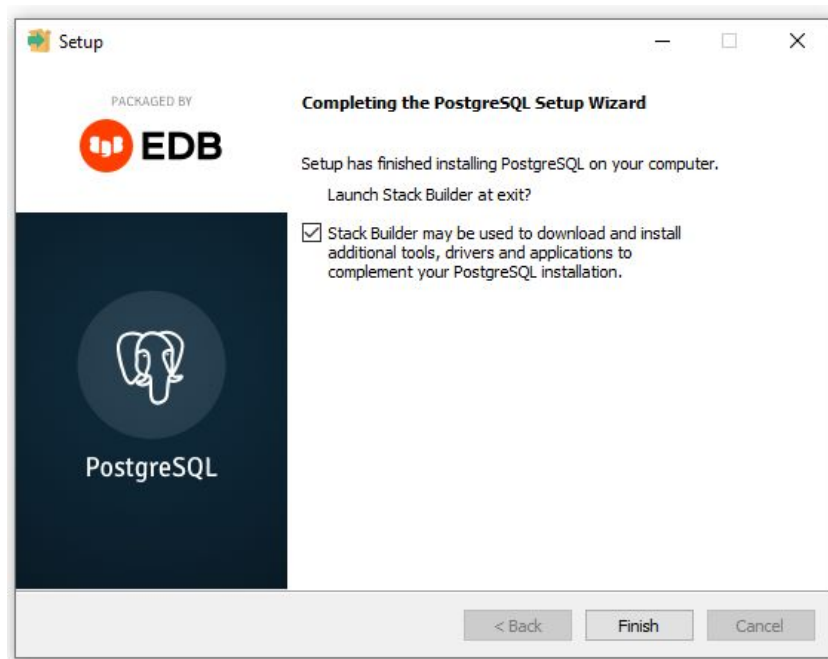




## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.1 Instalação do PostgreSQL no Windows

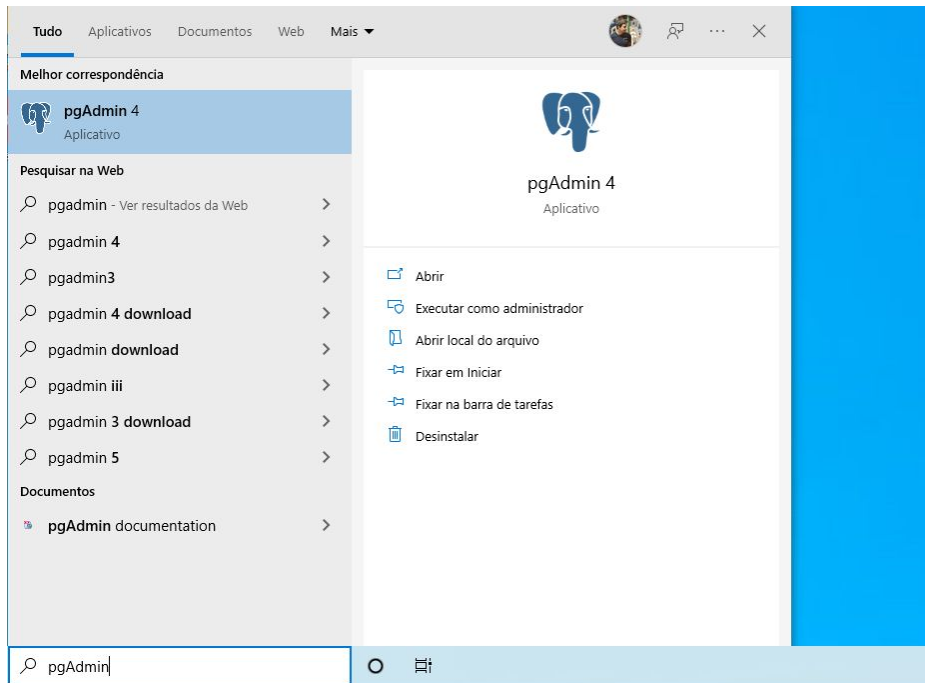
Finalize a instalação.



## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.2 Criação do banco de dados

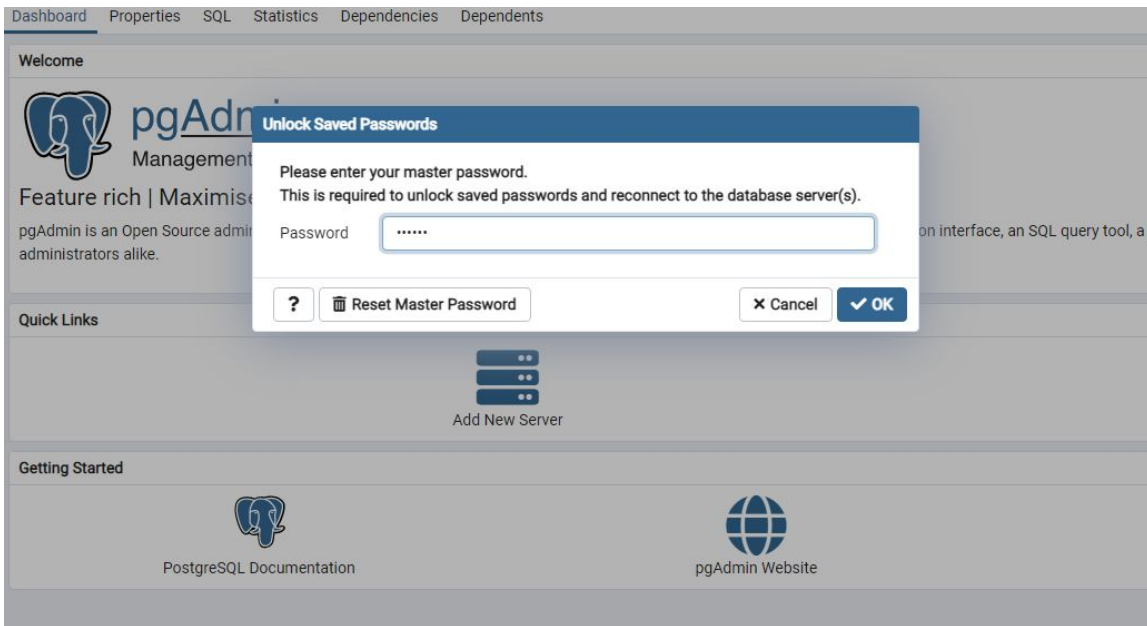
Iremos utilizar o pgAdmin 4 (que também foi instalado no pacote) para realizar a criação do banco de dados:



# 3. Instalando e configurando o PostgreSQL e DBeaver

## 3.2 Criação do banco de dados

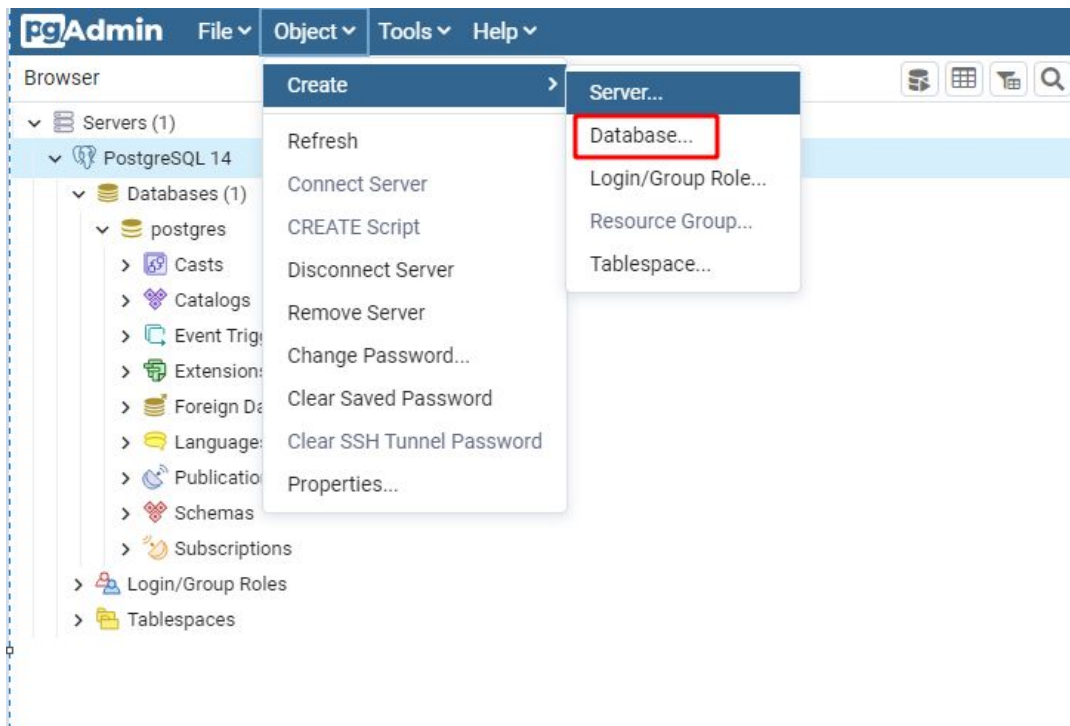
Insira a senha de administrador do banco de dados que foi configurada na instalação:



## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.2 Criação do banco de dados

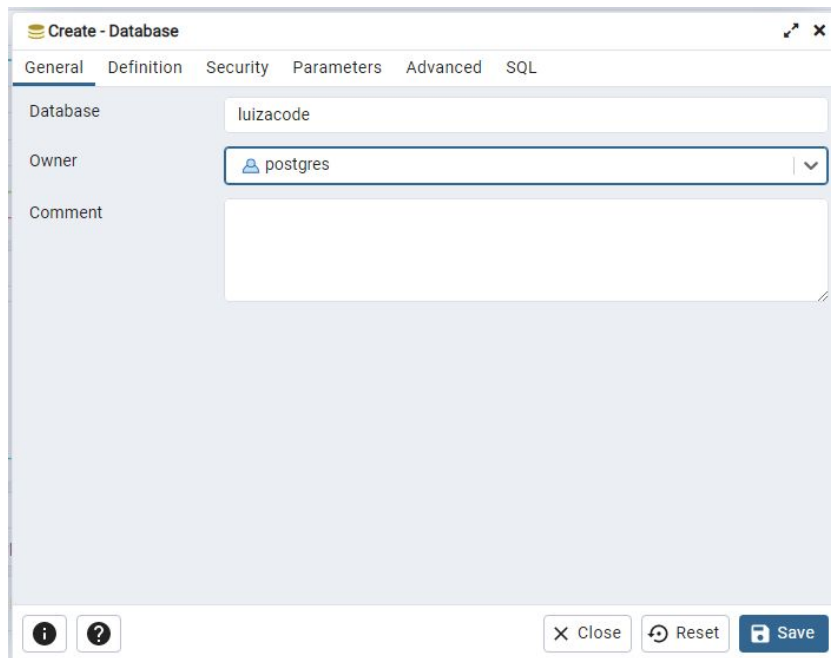
Selecione a opção Object -> Create -> Database...



## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.2 Criação do banco de dados

Defina o nome do banco de dados como **luizacode** e clique em “Save”.



The screenshot shows the 'Create - Database' dialog box in DBeaver. The 'General' tab is selected. The 'Database' field contains the text 'luizacode'. The 'Owner' field is a dropdown menu showing 'postgres' with a user icon. The 'Comment' field is an empty text area. At the bottom, there are buttons for 'Close', 'Reset', and 'Save', along with information and help icons.

## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.3 Instalação da IDE DBeaver

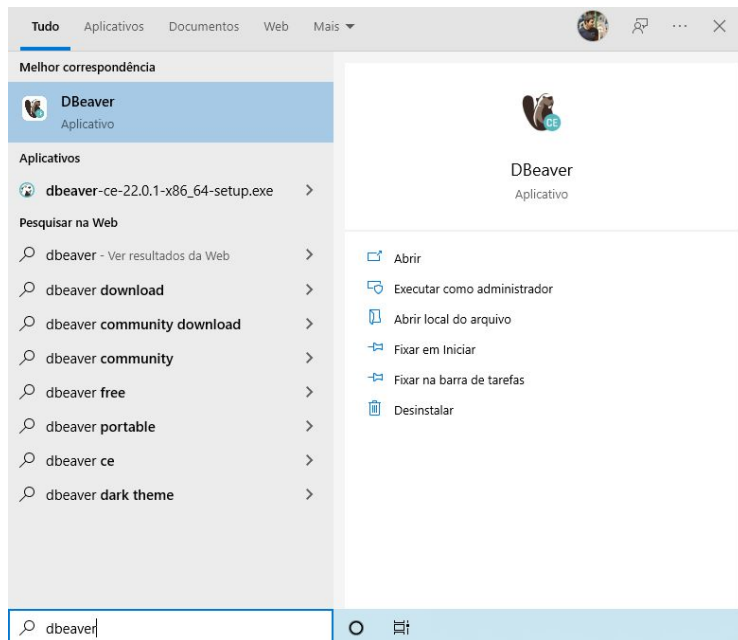
O DBeaver é um programa multiplataforma, que tem por objetivo conectar e manipular vários tipos de banco de dados. Realize o download através do link <https://dbeaver.io/download>.



## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.3 Instalação da IDE DBeaver

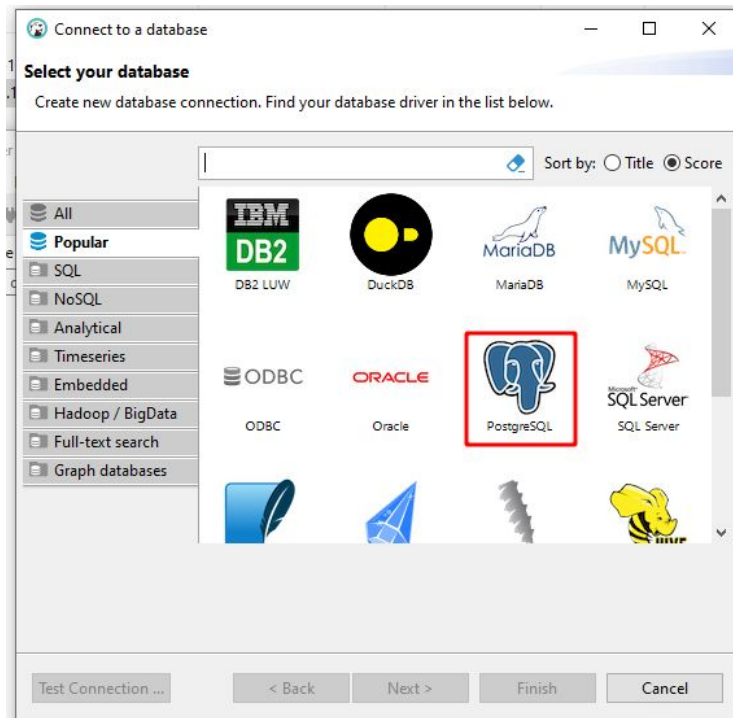
Inicie o DBeaver:



## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.3 Instalação da IDE DBeaver

Selecione a opção PostgreSQL:

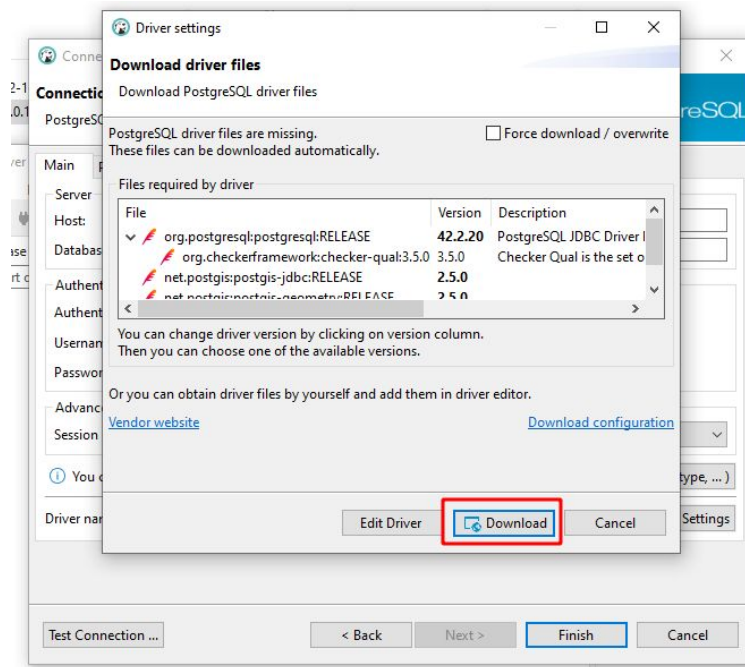




## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.3 Instalação da IDE DBeaver

A próxima etapa é realizar o download dos drivers necessários para realizar a conexão com o banco de dados. O DBeaver automaticamente sugere quais devem ser baixados:



## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.3 Instalação da IDE DBeaver

Agora inserimos as informações de conexão com o banco de dados. Aqui novamente utilizamos a senha definida na instalação.

The screenshot shows the 'Connect to a database' dialog box in DBeaver, specifically the 'PostgreSQL connection settings' tab. The 'Main' tab is selected, and the 'PostgreSQL' driver is chosen. The following fields are highlighted with red boxes:

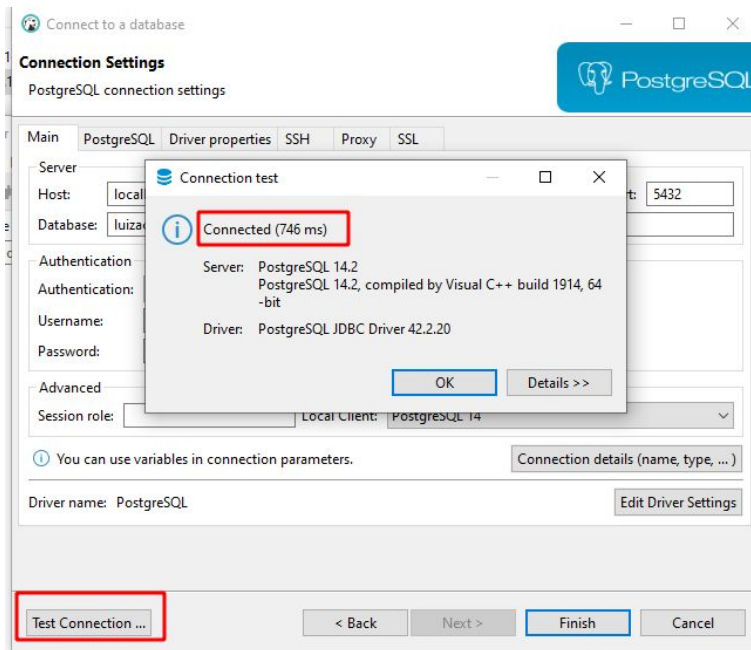
- Host: localhost
- Port: 5432
- Database: luizacode
- Authentication: Database Native (dropdown)
- Username: postgres
- Password: masked (masked with dots)

The 'Save password locally' checkbox is checked. The 'Advanced' section shows 'Session role' as an empty field and 'Local Client' as 'PostgreSQL 14'. A note at the bottom states: 'You can use variables in connection parameters.' with a link to 'Connection details (name, type, ...)'. The 'Driver name' is 'PostgreSQL', and there is an 'Edit Driver Settings' button. At the bottom, the 'Test Connection ...' button is highlighted with a blue box, along with '< Back', 'Next >', 'Finish', and 'Cancel' buttons.

## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.3 Instalação da IDE DBeaver

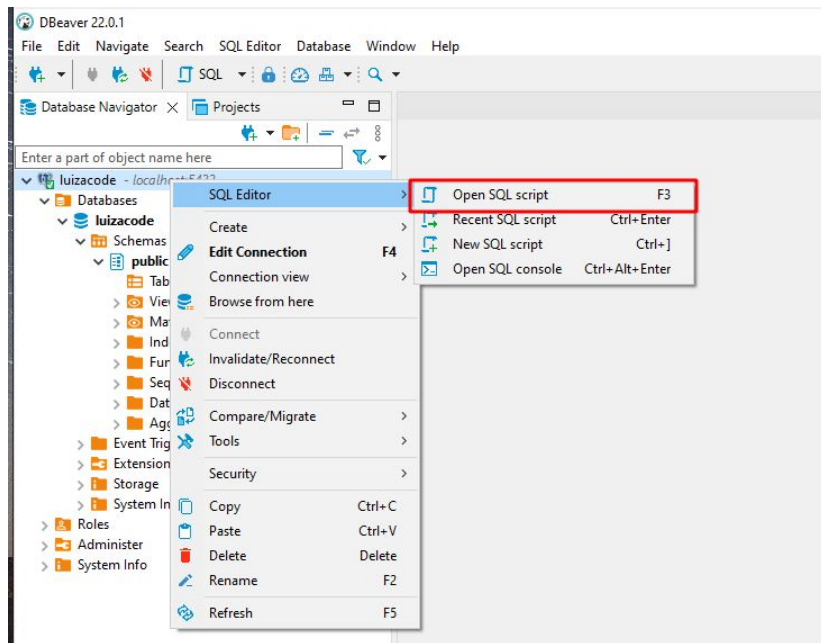
É aconselhável testar a conexão para confirmar se todos os dados foram inseridos corretamente.



## 3. Instalando e configurando o PostgreSQL e DBeaver

### 3.3 Instalação da IDE DBeaver

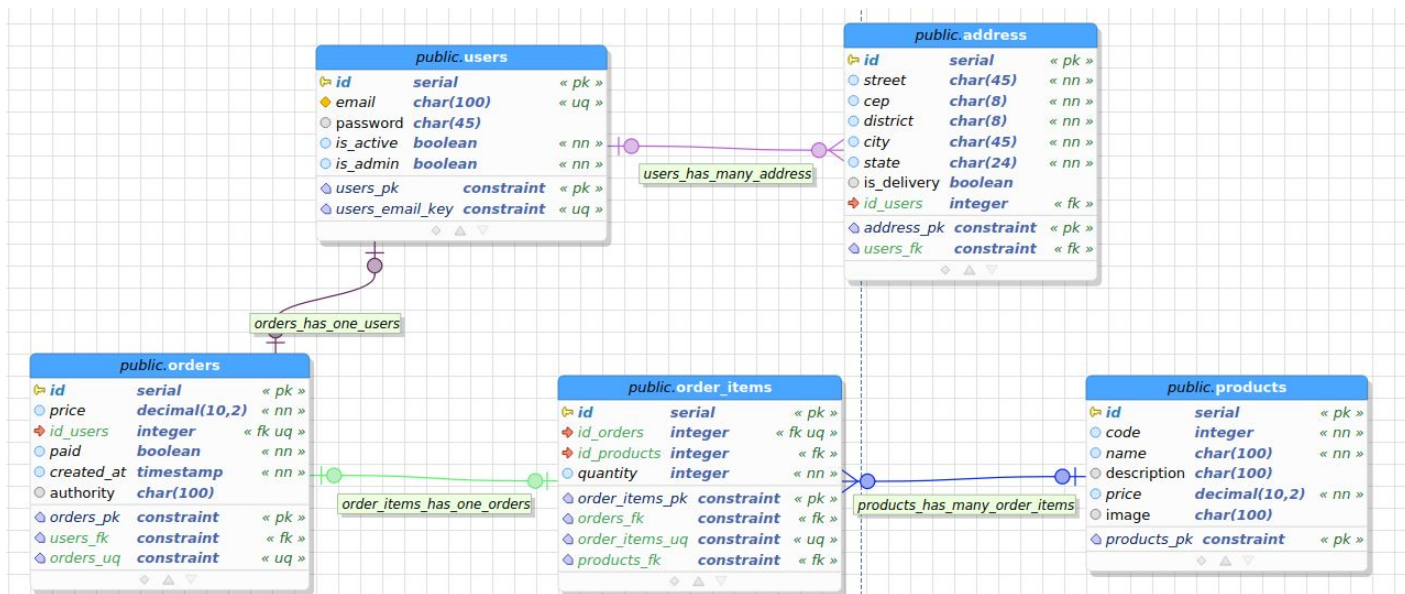
Agora que conseguimos conectar, iremos abrir um editor onde será possível realizar as operações de criação e manipulação do banco de dados.



## 4. Modelagem do banco de dados

### 4.1 Diagrama de Entidade-Relacionamento

São diagramas que representam a estrutura do banco de dados, são utilizados para projetar os esquemas das tabelas e suas ligações.



## 4. Modelagem do banco de dados

### 4.1.2 Diagrama de Entidade-Relacionamento

No PostgreSQL é usado o pgModeler para a modelagem dos dados:

<https://pgmodeler.io/>

## 4. Modelagem do banco de dados

### 4.2 Tipos de dados

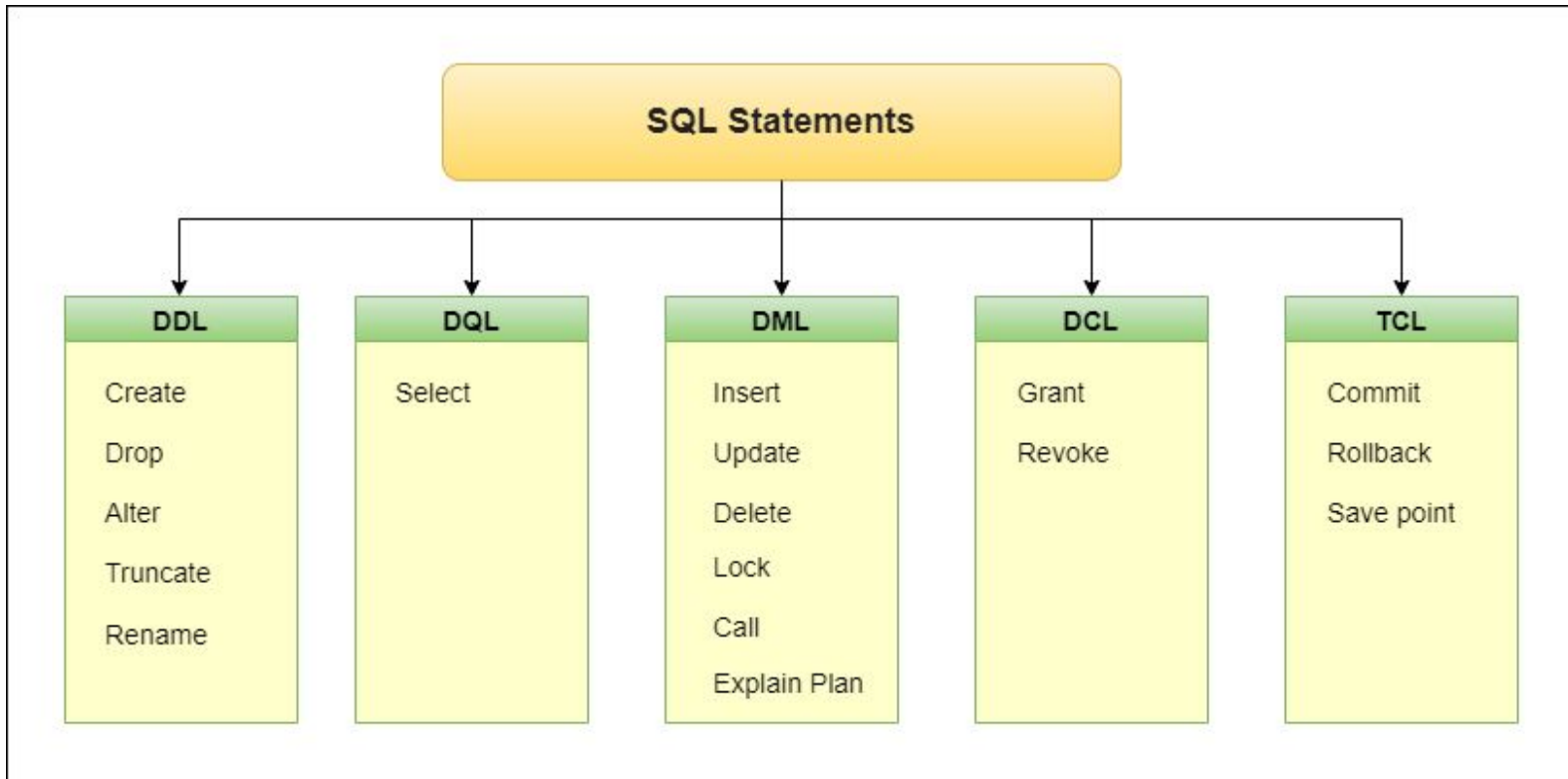
O PostgreSQL tem um rico conjunto de tipos de dados nativos disponíveis para os usuários. Além deles, é possível ainda adicionar novos tipos ao PostgreSQL usando o comando **CREATE TYPE**.

Iremos apresentar os principais:

Nome	Alias	Descrição
bigint	int8	inteiro de oito bytes
bigserial	serial8	inteiro de oito bytes de autoincremento
boolean	bool	lógica booleana (verdadeiro/falso)
character	char [ (n) ]	cadeia de caracteres de comprimento fixo
character varying	varchar [ (n) ]	cadeia de caracteres de comprimento variável
date		data do calendário (ano, mês, dia)
integer	int, int4	inteiro de quatro bytes assinado
numeric	decimal	numérico exato de precisão

## 4. Modelagem do banco de dados

### 4.3 Estruturas SQL:

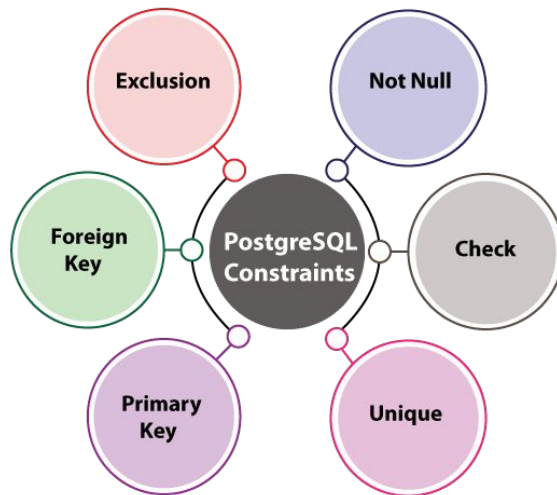




## 4. Modelagem do banco de dados

### 4.4 Constraints:

**Constraints** – Constraints são objetos usados com a finalidade de estabelecer regras referentes à integridade e à consistência nas colunas das tabelas pertencentes a um sistema de banco de dados. Isso é importante porque para planejar e criar tabelas devemos garantir a integridade dos dados presentes nas colunas e identificar os valores válidos para tais dados.



## 4. Modelagem do banco de dados

### 4.4.1 Primary Key:

**PRIMARY KEY** é uma restrição que identifica **exclusivamente** cada registro em uma tabela.

As chaves primárias devem conter valores **únicos** e não podem conter valores **NULL**.

Uma tabela pode ter apenas **UMA** chave primária; e na tabela, essa chave primária pode consistir em uma ou várias colunas (campos).

```
CREATE TABLE TABLE1 (  
    column_1 data_type PRIMARY KEY,  
    column_2 data_type,  
    ...  
);
```

## 4. Modelagem do banco de dados

### 4.4.2 Default

A constraint **DEFAULT** é utilizada para inserirmos um valor padrão em uma coluna. Esse valor padrão é inserido **automaticamente** nos registros, se nenhum valor for especificado para a coluna em questão.

```
CREATE TABLE TABLE1 (  
    column_1 data_type PRIMARY KEY,  
    column_2 data_type DEFAULT True,  
    ...  
);
```

## 4. Modelagem do banco de dados

### 4.4.3 Not null

A restrição **NOT NULL** garante que uma coluna não admita valores **NULL**. Isto significa que será abortada uma operação de INSERT ou UPDATE que coloque um valor **NULL** nessa coluna. A sua utilização é útil sempre que as regras de negócio obriguem ao preenchimento de um campo.

```
CREATE TABLE TABLE1 (  
    column_1 data_type PRIMARY KEY,  
    column_2 data_type DEFAULT NOT NULL,  
    ...  
);
```

## 4. Modelagem do banco de dados

### 4.4.4 Unique

A constraint **UNIQUE** impede que valores duplicados sejam inseridos na coluna. É implementada criando-se um índice unívoco em uma coluna.

```
CREATE TABLE TABLE1 (  
    column_1 data_type PRIMARY KEY,  
    column_2 data_type DEFAULT NOT NULL,  
    column_3 data_type,  
    CONSTRAINT column_3_uq UNIQUE (column_3)  
);
```

## 4. Modelagem do banco de dados

### 4.4.5 Foreign Key

A **FOREIGN KEY** é usada para evitar ações que destruam links entre tabelas.

A **FOREIGN KEY** é um campo (ou coleção de campos) em uma tabela, que se refere a **PRIMARY KEY** em outra tabela.

A tabela com a chave estrangeira é chamada de tabela filha e a tabela com a chave primária é chamada de tabela referenciada ou pai.

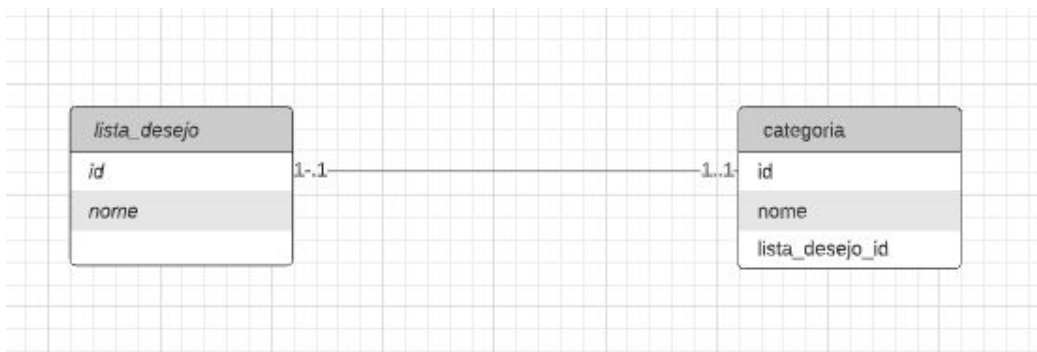
```
CREATE TABLE address(  
    id serial,  
    street char(45),  
    cep char(8),  
    district char(45),  
    city char(45),  
    state char(24),  
    is_delivery boolean default true,  
    constraint address_pkey primary key(id),  
    constraint fk_user foreign key(id) references users(id)  
);
```

## 5. Relacionamento entre tabelas

### 5.1 Relacionamento um para um

São relacionamentos em que **uma ocorrência** de uma entidade em **A** está associada no máximo a uma ocorrência em uma entidade **B** e uma ocorrência na entidade **B** está associada no máximo a uma ocorrência na entidade **A**.

No exemplo abaixo podemos afirmar que uma lista de desejo pode somente conter uma categoria vinculada a ela.



## 5. Relacionamento entre tabelas

### 5.2 Relacionamento um para muitos

Um relacionamento 1:m ocorre com frequência em situações de negócio. Às vezes ocorre em forma de árvore ou em **forma hierárquica**.

No exemplo abaixo, temos a seguinte representação: cada lista de desejo cadastrada possui **vários produtos** ligados a ela. O campo **lista\_desejo\_id** foi escolhido como chave estrangeira, que define de fato a ligação entre as duas tabelas. A representação ficaria assim:





## 6. SQL Nativo

### 6.1 Create table

A instrução **CREATE TABLE** é usada para criar uma nova tabela em um banco de dados.

Os parâmetros de **coluna** especificam os nomes das colunas da tabela.

O parâmetro **tipo** especifica o tipo de dados que a coluna pode conter (por exemplo, varchar, integer, date, etc.)

```
CREATE TABLE TABLE (  
    column_1 data_type,  
    column_2 data_type,  
    ...  
);
```

## 6. SQL Nativo

### 6.2 Drop table

A instrução **DROP TABLE** é usada para excluir uma tabela em um banco de dados.

A seguir a sintaxe da instrução:

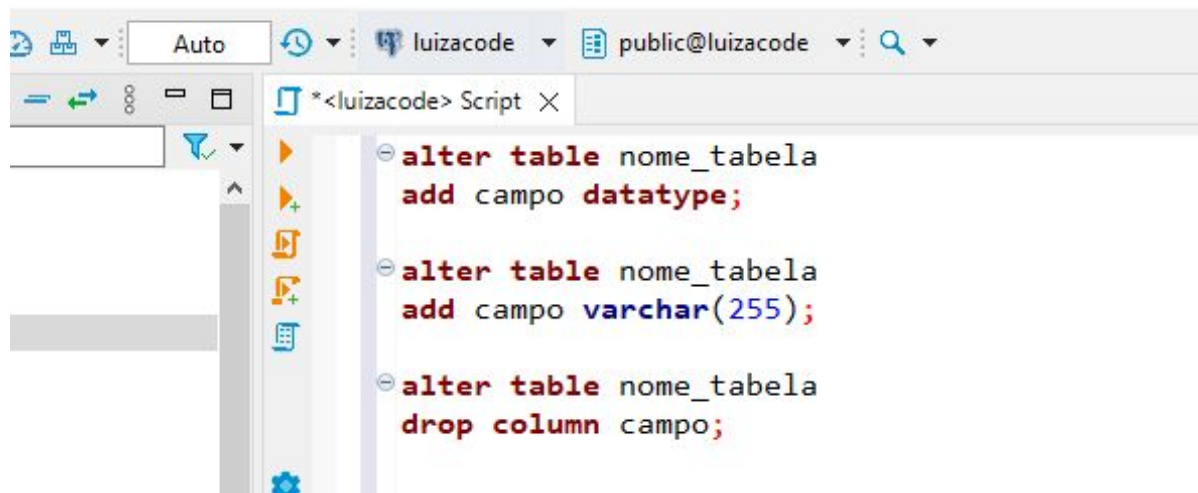
```
DROP TABLE TABLE;
```

## 6. SQL Nativo

### 6.3 Alter table

A instrução **ALTER TABLE** é usada para adicionar, excluir ou modificar colunas em uma tabela existente.

A seguir a sintaxe da instrução de adição, exclusão ou alteração de campos:



The screenshot shows a SQL editor interface with a toolbar at the top containing icons for undo, redo, save, and search. Below the toolbar is a tab labeled '\*<luizacode> Script X'. The main area displays three SQL statements, each preceded by a minus sign icon in a light blue circle:

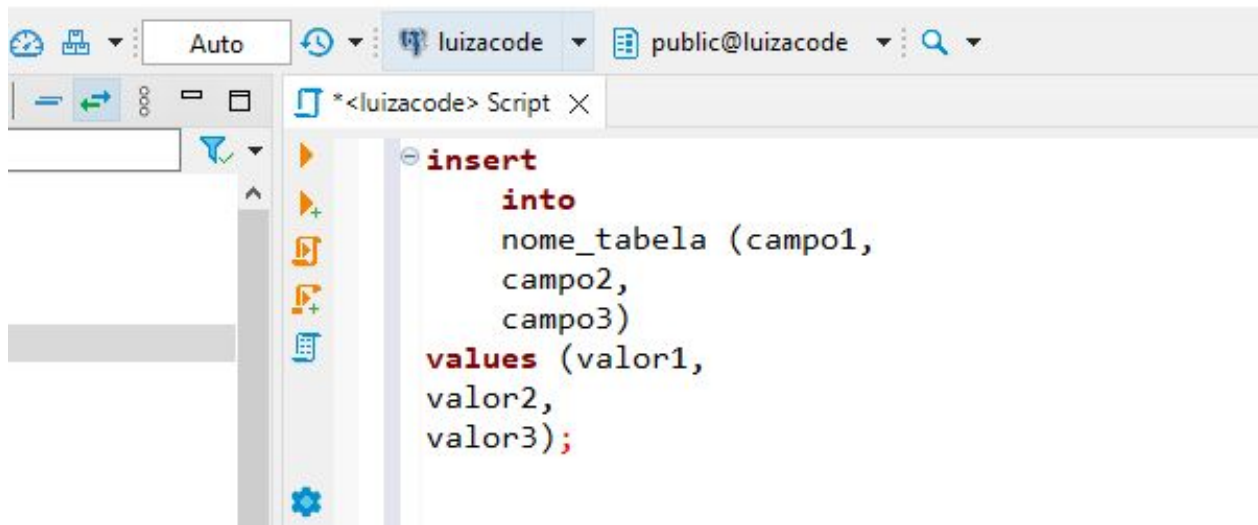
```
alter table nome_tabela  
add campo datatype;  
  
alter table nome_tabela  
add campo varchar(255);  
  
alter table nome_tabela  
drop column campo;
```

## 6. SQL Nativo

### 6.4 Insert into

A instrução **INSERT INTO** é usada para inserir novos registros em uma tabela.

Devemos especificar os **nomes** das colunas e os **valores** a serem inseridos:

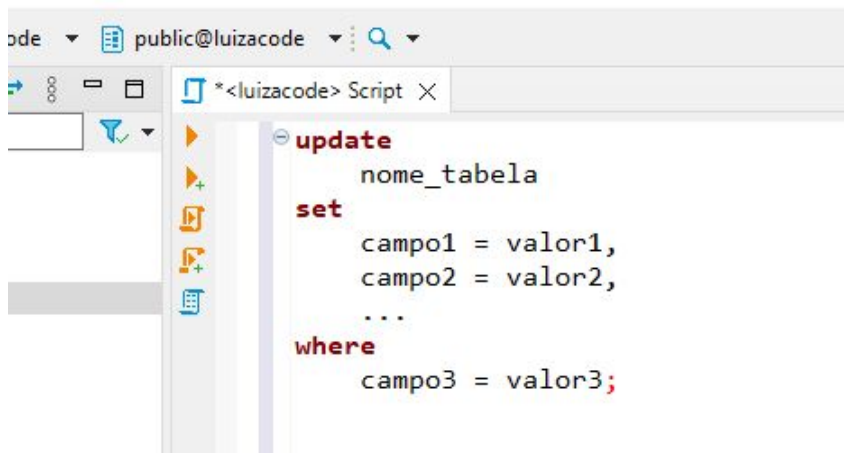


## 6. SQL Nativo

### 6.5 Update

A instrução **UPDATE** é usada para modificar os registros existentes em uma tabela.

É extremamente importante ter cuidado ao atualizar registros em uma tabela! Observe sempre a cláusula **WHERE** declaração. Ela especifica quais registros devem ser atualizados. Se você omitir essa cláusula, todos os registros da tabela serão atualizados!



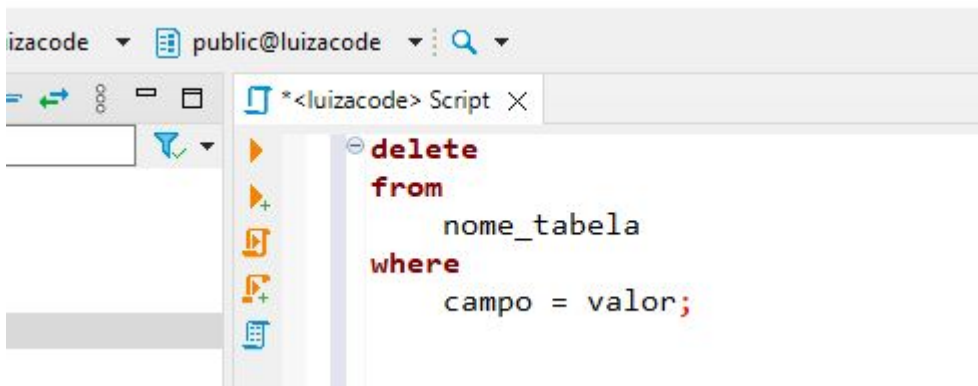
```
update
  nome_tabela
set
  campo1 = valor1,
  campo2 = valor2,
  ...
where
  campo3 = valor3;
```

## 6. SQL Nativo

### 6.6 Delete

A instrução **DELETE** é usada para excluir registros existentes em uma tabela.

Para esse instrução também é muito importante ter cuidado com a cláusula **WHERE**. Caso contrário registros que não deveriam ser apagados podem ser impactados!

A screenshot of a SQL editor interface. The top bar shows 'luizacode' and 'public@luizacode'. The main editor area displays a SQL script with the following text: 

```
delete
from
    nome_tabela
where
    campo = valor;
```


 The script is highlighted with a light blue background. The editor has a sidebar on the left with various icons for file management and execution. The title bar of the editor window says '\*<luizacode> Script X'.

## 7. Manipulação de dados SQL

### 7.1 Select

A cláusula **SELECT** é usada para selecionar dados de um banco de dados.

Os dados retornados são armazenados em uma tabela de resultados, chamada de conjunto de resultados.

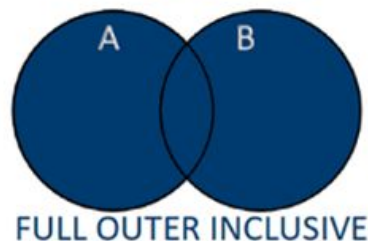
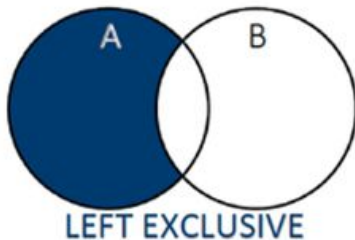
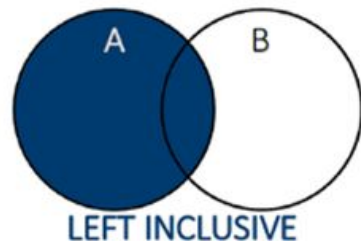


```
SELECT coluna1, coluna2, ...  
FROM nome_tabela;
```

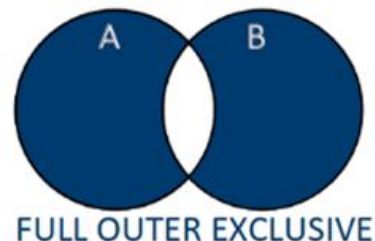
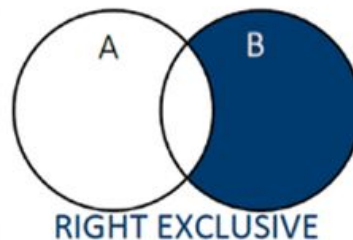
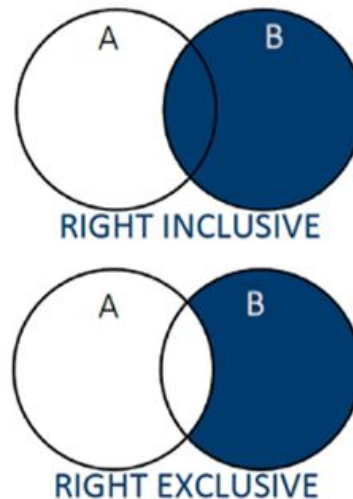
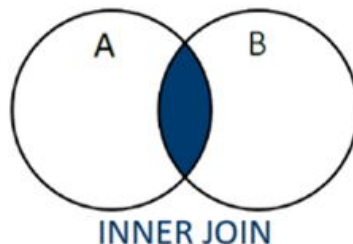
Aqui, coluna1, coluna2, ... são os nomes dos campos da tabela da qual você deseja selecionar os dados. Se você deseja selecionar todos os campos disponíveis na tabela, use a seguinte sintaxe:

# 7. Manipulação de dados SQL

## 7.2.1 SQL JOINS



SQL JOINS	
<b>LEFT INCLUSIVE</b> SELECT [Select List] FROM TableA A LEFT OUTER JOIN TableB B ON A.Key = B.Key	<b>RIGHT INCLUSIVE</b> SELECT [Select List] FROM TableA A RIGHT OUTER JOIN TableB B ON A.Key = B.Key
<b>LEFT EXCLUSIVE</b> SELECT [Select List] FROM TableA A LEFT OUTER JOIN TableB B ON A.Key = B.Key WHERE B.Key IS NULL	<b>RIGHT EXCLUSIVE</b> SELECT [Select List] FROM TableA A LEFT OUTER JOIN TableB B ON A.Key = B.Key WHERE A.Key IS NULL
<b>FULL OUTER INCLUSIVE</b> SELECT [Select List] FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key	<b>FULL OUTER EXCLUSIVE</b> SELECT [Select List] FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key WHERE A.Key IS NULL OR B.Key IS NULL
<b>INNER JOIN</b> SELECT [Select List] FROM TableA A INNER JOIN TableB B ON A.Key = B.Key	





## 7. Manipulação de dados SQL

### 7.2.2 Join

A cláusula **JOIN** é usada para combinar linhas de duas ou mais tabelas, com base em uma coluna relacionada entre elas.

Observe que a coluna **id** na tabela **lista\_desejo** se refere ao "**lista\_desejo\_id**" na tabela "**produtos**". A relação entre as duas tabelas acima é a coluna "**lista\_desejo\_id**".

Em seguida, podemos criar a seguinte instrução SQL (que contém um INNER JOIN), que seleciona os registros que possuem valores correspondentes em **ambas as tabelas**:


```
SELECT nome_tabela1.coluna1, nome_tabela2.coluna2  
FROM nome_tabela1  
INNER JOIN nome_tabela2 ON nome_tabela1.coluna1 = nome_tabela2.coluna2;
```

## 7. Manipulação de dados SQL

### 7.3 Where

A cláusula **WHERE** é usada para filtrar registros.

Ele é usado para extrair apenas os registros que atendem a uma condição especificada.

A small icon of a code editor with a list of symbols on the left side.


```
SELECT coluna1, coluna2, ...  
FROM nome_tabela  
WHERE condicao;
```

Lembrando que a cláusula **WHERE** não é usada apenas em declarações **SELECT**, ela também é usada em **UPDATE**, **DELETE**, etc.!

## 7. Manipulação de dados SQL

### 7.4 Order by

A palavra-chave **ORDER BY** é usada para classificar o conjunto de resultados em ordem crescente ou decrescente.

A vertical sidebar on the left of the code block contains two icons: a yellow one with a plus sign and a blue one with a list icon.

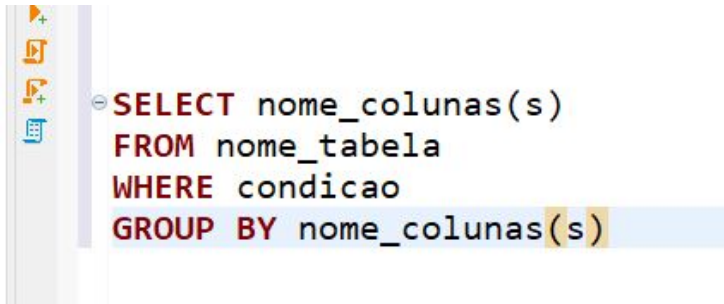
```
SELECT coluna1, coluna2, ...  
FROM nome_tabela  
ORDER BY coluna1, coluna2, ... ASC|DESC;
```

Ela classifica os registros em ordem crescente por padrão. Para classificar os registros em ordem decrescente, use a palavra-chave **DESC**.

## 7. Manipulação de dados SQL

### 7.5 Group by

A instrução **GROUP BY** agrupa as linhas que têm os mesmos valores em linhas de resumo.



```
SELECT nome_colunas(s)
FROM nome_tabela
WHERE condicao
GROUP BY nome_colunas(s)
```

Ela é frequentemente usada com funções agregadas (**COUNT**, **MAX**, **MIN**, **SUM**, **AVG**) para agrupar o conjunto de resultados por uma ou mais colunas.

## 7. Manipulação de dados SQL

### 7.6 COUNT, AVG e SUM

A função **COUNT** retorna o número de linhas que corresponde a um critério especificado.

```
SELECT COUNT(nome_coluna)
FROM nome_tabela
WHERE condicao;
```

A função **AVG** retorna o valor médio de uma coluna numérica.

```
SELECT AVG(nome_coluna)
FROM nome_tabela
WHERE condicao;
```

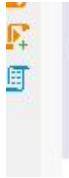
A função **SUM** retorna a soma total de uma coluna numérica.

```
SELECT SUM(nome_coluna)
FROM nome_tabela
WHERE condicao;
```

## 7. Manipulação de dados SQL


### 7.7 MIN E MAX

A função **MIN** retorna o menor valor da coluna selecionada.



```
SELECT MIN(nome_coluna)
FROM nome_tabela
WHERE condicao;
```

A função **MAX** retorna o maior valor da coluna selecionada.



```
SELECT MAX(nome_coluna)
FROM nome_tabela
WHERE condicao;
```

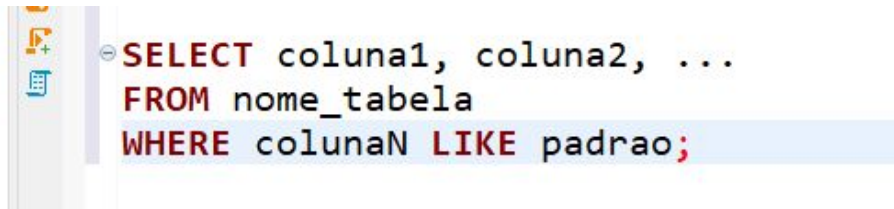
## 7. Manipulação de dados SQL

### 7.8 LIKE

O operador **LIKE** é usado em uma cláusula **WHERE** para procurar um padrão especificado em uma coluna.

Existem dois curingas frequentemente usados em conjunto com o operador:

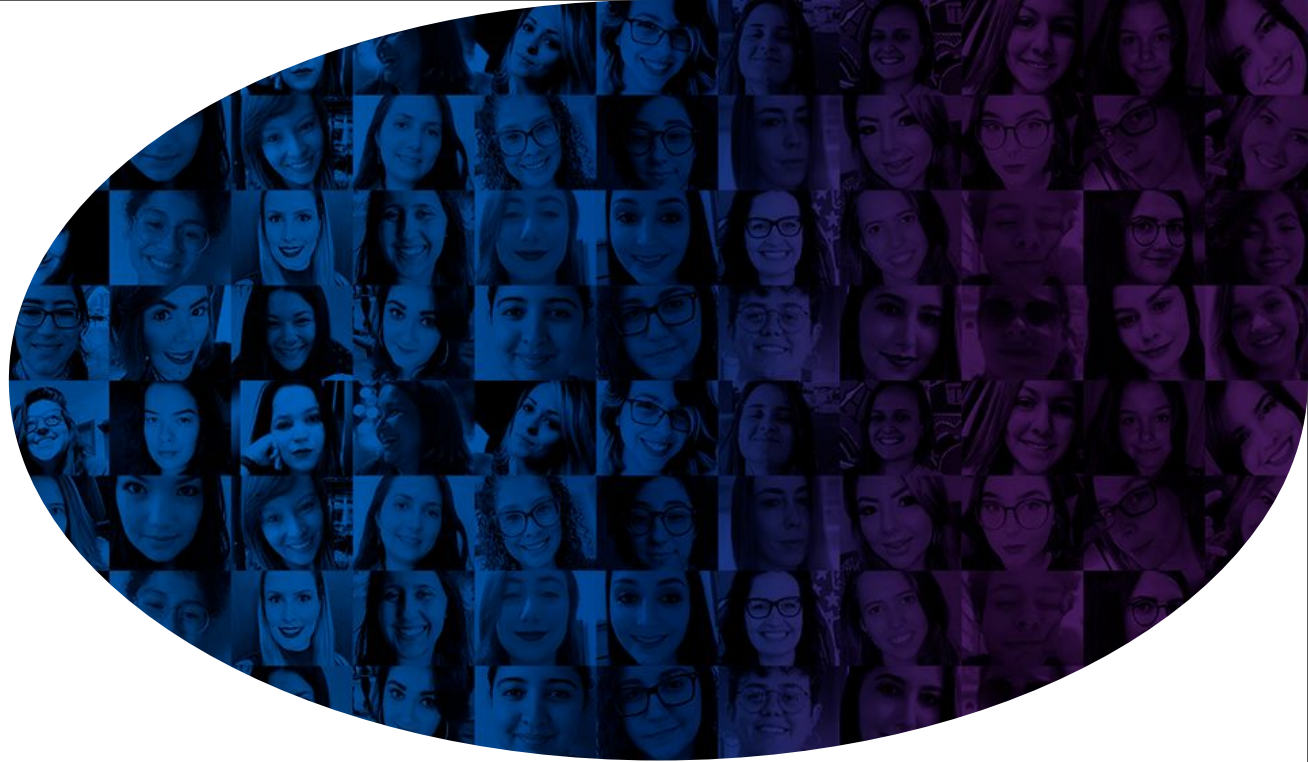
- O sinal de porcentagem (%) representa zero, um ou vários caracteres
- O sinal de sublinhado (\_) representa um único caractere

A screenshot of a code editor showing a SQL query. The query is: `SELECT coluna1, coluna2, ...  
FROM nome_tabela  
WHERE colunaN LIKE padrao;` The text is color-coded: `SELECT` is red, `FROM` is blue, `WHERE` is red, and `LIKE` is red. The rest of the text is black. The code is highlighted with a light blue background.

```
SELECT coluna1, coluna2, ...  
FROM nome_tabela  
WHERE colunaN LIKE padrao;
```

Podemos utilizar como exemplo, um padrão de busca por uma coluna que tenha o valor começando com “Luiza”:

**colunaN** **LIKE** 'Luiza%'



# Perguntas?



# Magalu



## #VemSerFeliz