53 45 52 45 49 20 46 49 45 4c 20
41 4f 53 20 50 52 45 43 45 49 54
4f 53 20 44 41 20 48 4f 4e 52 41
20 45 20 44 41 20 43 49 c3 8a 4e
43 49 41 2c 20 50 52 4f 4d 4f 56
45 4e 44 4f 20 4f 20 55 53 4f 20
45 20 4f 20 44 45 53 45 4e 56 4f
4c 56 49 4d 45 4e 54 4f 20 44 41
20 49 4e 46 4f 52 4d c3 81 54 49
43 41 20 45 4d 20 42 45 4e 45 46
c3 8d 43 49 4f 20 44 4f 20 43 49
44 41 44 c3 83 4f 20 45 20 44 41
20 53 4f 43 49 45 44 41 44 45 2e

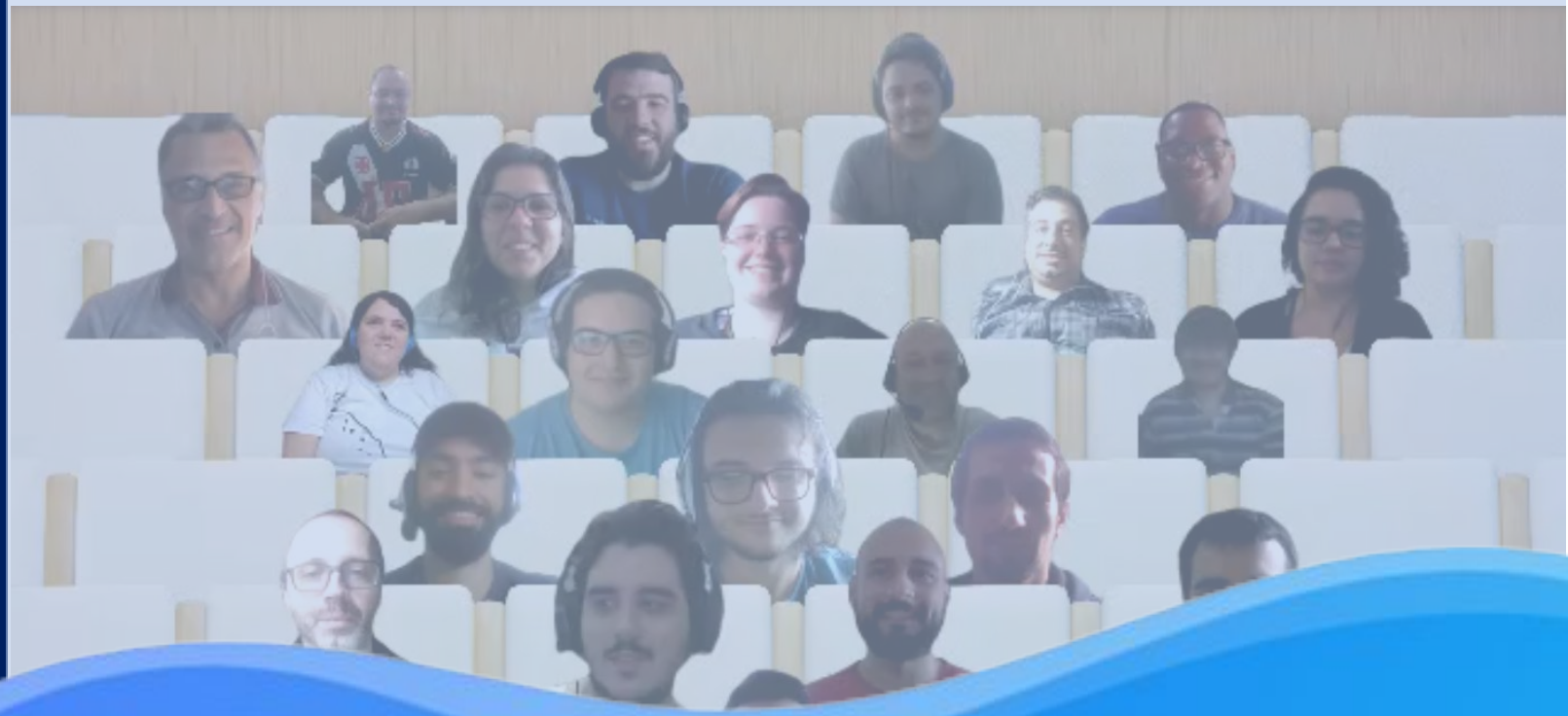
RESIDÊNCIA DE SOFTWARE

CAPACITAR
TREINAR
EMPREGAR
TRANSFORMAR

Firjan SENAI

serratec
Parque Tecnológico Região Serrana

Banco de Dados



Aula 4
Modelo Físico

Roni Schanuel
23-08-2022

Projeto Físico

O projeto físico lida com o banco de dados real com base nos requisitos reunidos durante a modelagem lógica do banco de dados. Durante a modelagem física, os objetos são definidos em um nível denominado nível de esquema. Um esquema é considerado um grupo de objetos que estão relacionados entre si em um banco de dados. Tabelas e colunas são feitas de acordo com as informações fornecidas durante a modelagem lógica. A modelagem física depende do programa de banco de dados que já está sendo usado na empresa.

Um modelo físico pode ser constituído de código SQL para criação de objetos no banco.

SQL(Standard Query Language)

É uma linguagem de pesquisa declarativa padrão para banco de dados relacionais .

Criada no início dos anos 70 com o objetivo de demonstrar que era viável implementar o modelo relacional
Proposto por E. F. Codd, membro do laboratório de pesquisa da IBM em San Jose, CA

Padrão utilizado pelos sistemas de banco de dados relacionais

Pode ser dividida nas seguintes categorias:
DQL - Data Query Language
DDL - Data Definition Language
DML - Data Manipulation Language

SQL
Structured Query language
DCL - Data Control Language
DTL - Data Transaction Language

SQL(Standard Query Language)

Resumindo o que vamos ver

- **DDL**
  CREATE TABLE
  CREATE INDEX
  CREATE VIEW
  ALTER TABLE
  ALTER INDEX
  DROP INDEX
  DROP VIEW
- **DML**
  INSERT
  UPDATE
  DELETE
- **DQL**
  SELECT
  - **Cláusulas**
    FROM, WHERE
    GROUP BY, HAVING
    ORDER BY, DISTINCT
    UNION
  - **Operadores lógicos e relacionais**
    AND, OR, NOT
    >,> =,<,< =,= = > <>
    BETWEEN, IN, LIKE
  - **Funções de agregação**
    MAX, MIN
    AVG, SUM, COUNT
  - **Junções**
    INNER JOIN
    LEFT JOIN
    RIGHT JOIN
    FULL OUTER JOIN
- **DCL**
  GRANT
  REVOKE
- **Backup e Restauração**

PostgreSQL

O PostgreSQL é um SGDB que foi desenvolvido na Universidade da Califórnia em Berkeley Computer Science Department. Foi pioneiro em muitos dos conceitos que só se tornaram disponíveis em alguns sistemas de banco de dados comerciais mais tarde. O PostgreSQL é um descendente de código aberto do código original desenvolvido em Berkeley e suporta uma grande parte do padrão SQL e oferece muitas características modernas.

O PostgresSQL possui características de um banco de dados transacional, suporta views, procedimentos, triggers e outros.

Banco de Dados

Um banco de dados é uma coleção organizada de informações ou dados estruturados armazenados em um servidor.

Para a criação de banco de dados, tabelas e atributos em um SGBD, podemos utilizar a linguagem SQL que é composta de comandos de manipulação, definição e controle de dados ou a interface gráfica de um gerenciador como PgAdmin ou Dbeaver. Para usar a linha de comando utilizaremos os comando SQL de DDL(Data Definition Language), que disponibiliza um conjunto de comandos para criação (CREATE), alteração (ALTER) e remoção (DROP) de tabelas e outras estruturas.

CREATE E DROP DATABASE

A maioria dos SGBDs disponibiliza ferramentas que permitem a criação de Banco de Dados, mas é possível criar o próprio Banco de Dados a partir de um comando SQL.

A sintaxe do comando é:

CREATE DATABASE nome_do_banco_de_dados
CREATE DATABASE AULABD

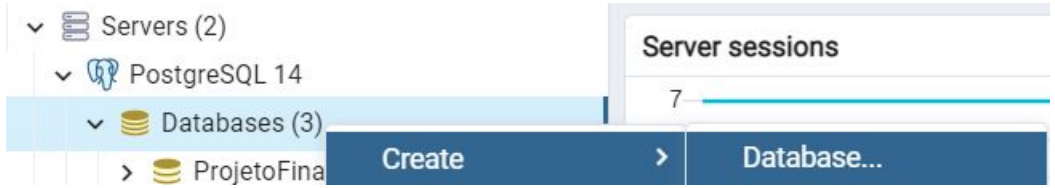
O comando DROP DATABASE permite remover um determinado Banco de Dados, apagando todas as tabelas e estruturas associadas e, consequentemente, todos os dados existentes nelas.

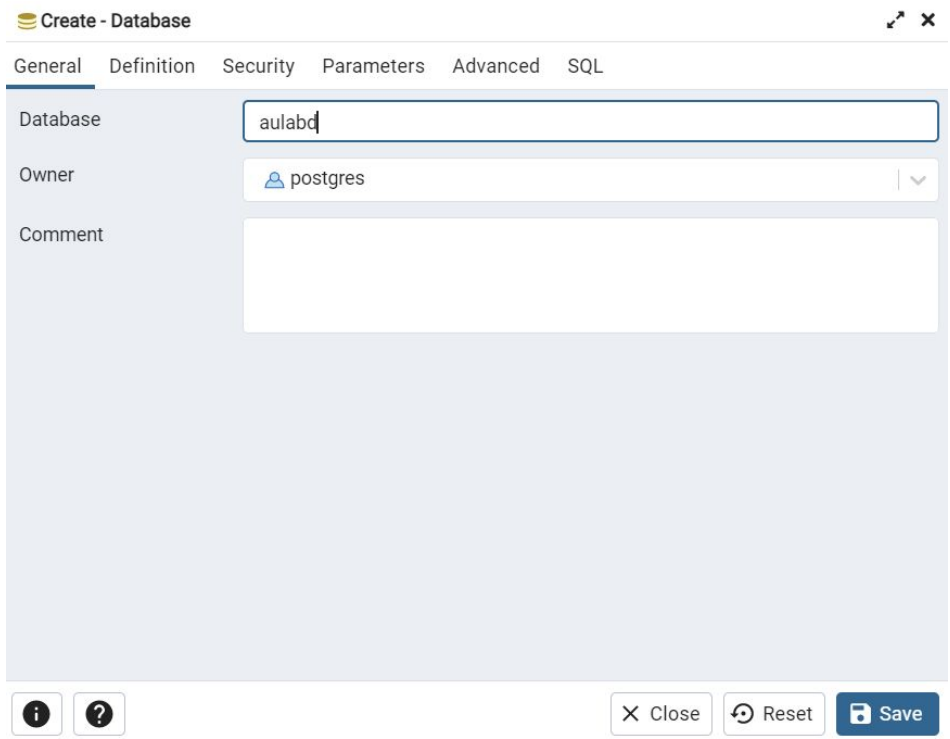
A sintaxe do comando é:

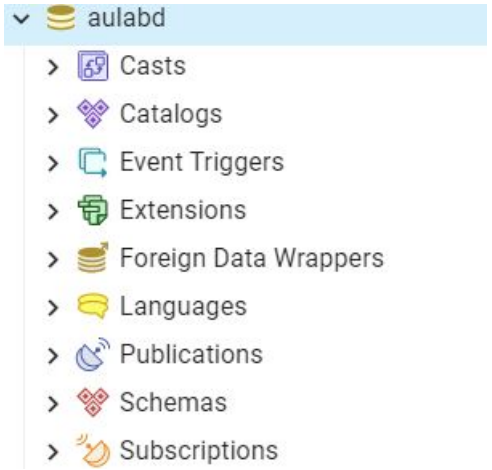
DROP DATABASE nome_do_banco_de_dados
DROP DATABASE AULABD

Criação do banco de dados no Postgres usando o PgAdmin

Vamos abrir o gerenciador do banco de dados utilizando o DBeaver ou PgAdmin.







Tabela

É um modelo de dados para um SGBD, que se baseia no princípio que todos os dados estão guardados em tabelas que são as entidades do MER. O nome de tabelas devem ser únicos no banco de dados.

Tabela Aluno

Num_Matrícula	Nome_Aluno	Data_Nascimento
1	José	1993-12-01
2	Ana	1978-10-16
4	Maria	1995-07-15
5	Joaquim	1998-03-13

SQL(Standard Query Language)

DDL - Data Definition Language

Tabelas

CREATE TABLE

Utilizado para criar novas tabelas no banco de dados. É necessário definir o nome da tabela, assim como o nome e o tipo de dados de cada coluna.

```
CREATE TABLE nome_tabela (coluna_1 datatype, coluna_2 datatype, coluna_3 datatype);
```

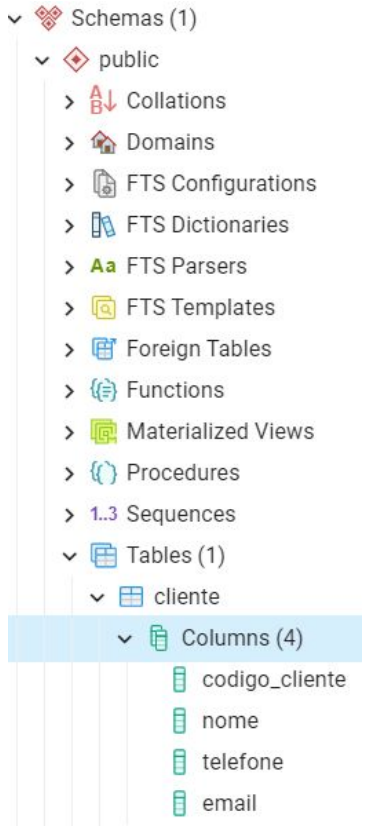
ALTER TABLE

Permite que a estrutura da tabela seja modificada, adicionando ou excluindo novas colunas ou modificando o tipo de dados de alguma coluna, por exemplo.

```
ALTER TABLE nome_tabela ADD nome_coluna datatype;
ALTER TABLE nome_tabela DROP COLUMN nome_coluna;
ALTER TABLE nome_tabela MODIFY COLUMN nome_coluna datatype;
```

Exemplo criação de tabela, remoção, adição e modificação de colunas

```
CREATE TABLE cliente (codigo_cliente int primary key, nome varchar(40), telefone varchar(11), email varchar(50))
```




```
ALTER TABLE cliente DROP COLUMN email;
ALTER TABLE cliente ADD COLUMN celular varchar(11);
ALTER TABLE cliente ALTER COLUMN nome TYPE varchar(60);
```

```
CREATE TABLE produto (codigo_produto int PRIMARY KEY,
nome VARCHAR(40) NOT NULL,
descricao TEXT,
preco NUMERIC CHECK (preco > 0) NOT NULL,
quantidade_estoque SMALLINT DEFAULT 0);
```

Exemplo criação de tabela

Constraints

Usadas para criar regras para os atributos nas tabelas do banco de dados

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Uniquely identifies a row/record in another table
- CHECK - Ensures that all values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column when no value is specified
- INDEX - Used to create and retrieve data from the database very quickly

```
CREATE TABLE produto (codigo_produto int PRIMARY KEY,
nome VARCHAR(40) NOT NULL,
descricao TEXT,
preco NUMERIC CHECK (preco > 0) NOT NULL,
quantidade_estoque SMALLINT DEFAULT 0);
```

Usando o tipo **TEXT** não precisamos definir um limite.
Podemos usar o comando **CHECK** para criar restrições na criação dos campos.

Exemplo criação de tabela chave estrangeira

```
CREATE TABLE departamento(codigo_departamento INT PRIMARY KEY,
                          nome varchar(30));


CREATE TABLE funcionario(codigo_funcionario INT PRIMARY KEY,
                         nome varchar(40),
                         telefone varchar(11),
                         email varchar (50),
                         codigo_departamento INT,
                         FOREIGN KEY (codigo_departamento) REFERENCES departamento(codigo_departamento));
```

DROP TABLE

Permite a exclusão de tabelas. Se existirem registros na tabela, os mesmos também serão excluídos. Deve-se tomar atenção no que diz respeito às violação das restrições de integridade (chaves estrangeiras, por exemplo)

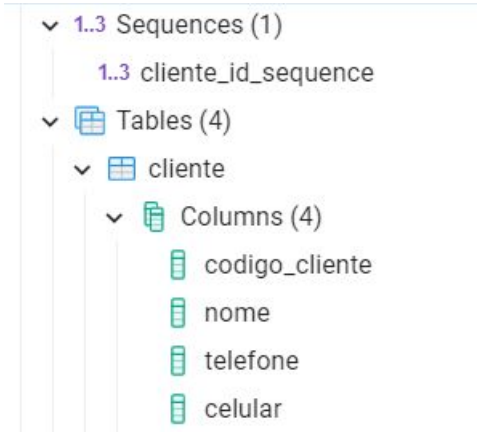
Para remover uma tabela usamos o comando **DROP TABLE <nome da tabela>**

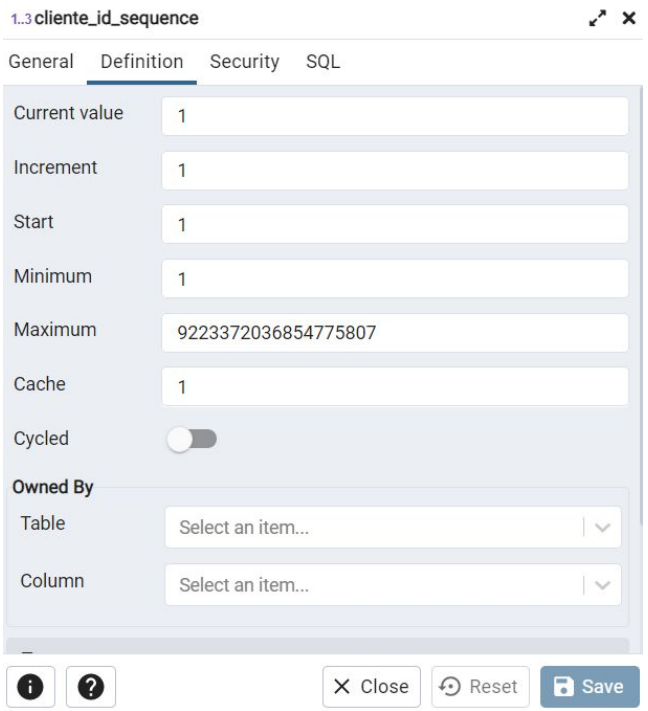
```
DROP TABLE cliente;

DROP TABLE departamento;

DROP TABLE produto;
```

Sequence e Auto Increment

Auto Increment - Valor numérico sequencial que não se repete.
Sequence - Temos um flexibilidade maior definido intervalos, valores máximos e mínimos.

```
CREATE SEQUENCE cliente_id_sequence;
CREATE TABLE IF NOT EXISTS public.cliente(
    codigo_cliente BIGINT PRIMARY KEY DEFAULT NEXTVAL ('cliente_id_sequence'),
    nome varchar(60),
    telefone varchar(11),
    celular varchar(11)
);
```





Sequence Serial

Usando o comando serial na criação da tabela o sequence é criado automaticamente.

```
create table categoria(codigo_categoria serial primary key,
                       descricao varchar(30))
```

DML - Data Manipulation Language

INSERT

Utilizado na inserção de novos registros em uma tabela.

```
INSERT INTO nome_tabela
(coluna_1,coluna_2,coluna_3) VALUES (valor_1,
'valor_2', valor_3);
```

```
INSERT INTO cliente (nome,telefone,celular) VALUES ('Jorge','22457898','999854521');
INSERT INTO cliente (nome,telefone,celular) VALUES ('Ana','22437191','999235217');
INSERT INTO cliente (nome,telefone,celular) VALUES ('Carla','22457898','9988542214');
```

```
insert into categoria (descricao) VALUES('Eletrônicos'),
                                        ('Alimentos'),
                                        ('Perfumaria'),
                                        ('Farmácia');
```

```sql
CREATE TABLE produto (codigo_produto serial PRIMARY KEY,
nome varchar(40) NOT NULL, descricao TEXT,
preco NUMERIC CHECK (preco > 0) NOT NULL,
quantidade_estoque SMALLINT DEFAULT 0,
codigo_categoria int,
FOREIGN KEY (codigo_categoria) REFERENCES categoria(codigo_categoria));
```

DML - Data Manipulation Language

```sql
INSERT INTO PRODUTO (nome, descricao, preco, quantidade_estoque,codigo_categoria) VALUES('Arroz','Tio João',7.5,40,2);
INSERT INTO PRODUTO (nome, descricao, preco, quantidade_estoque,codigo_categoria) VALUES('Feijão','Carreteiro',8.5,30,2);
INSERT INTO PRODUTO (nome, descricao, preco, quantidade_estoque,codigo_categoria) VALUES('Feijão','Granfino',8.5,30,2);
INSERT INTO PRODUTO (nome, descricao, preco, quantidade_estoque,codigo_categoria) VALUES('Macarrão','Adria',6.65,10,2);
INSERT INTO PRODUTO (nome, descricao, preco, quantidade_estoque,codigo_categoria) VALUES('Farinha de Trigo','Boa Sorte',2.5,8,2);
INSERT INTO PRODUTO (nome, descricao, preco, quantidade_estoque,codigo_categoria) VALUES('Sal','Cisne',2.5,100,2);
INSERT INTO PRODUTO (nome, descricao, preco, quantidade_estoque,codigo_categoria) VALUES('Atum','Gomes da Costa',6.8,42,2);
INSERT INTO PRODUTO (nome, descricao, preco, quantidade_estoque,codigo_categoria) VALUES('Leite Condensado','Nestle',5.9,40,2);
INSERT INTO PRODUTO (nome, descricao, preco, quantidade_estoque,codigo_categoria) VALUES('Creme de Leite','Pirancajuba',2.8,15,2);
INSERT INTO PRODUTO (nome, descricao, preco, quantidade_estoque,codigo_categoria) VALUES('Arroz','Carreteiro',7.5,4,2);
INSERT INTO PRODUTO (nome, descricao, preco, quantidade_estoque,codigo_categoria) VALUES('Neosoro','EMS',17.5,4,1);
```

DML - Data Manipulation Language

UPDATE

Permite a alteração de registros em uma tabela.

UPDATE nome_tabela SET alguma_coluna =
algum_valor WHERE alguma_coluna = algum_valor;

```
UPDATE cliente SET nome = 'Jorge Luis', telefone='22490001'
WHERE codigo_cliente = 1;

UPDATE categoria SET nome = 'Cosméticos'
WHERE codigo_categoria = 3;
```

DML - Data Manipulation Language

DELETE

Permite a exclusão de registros em uma tabela.

DELETE FROM nome_tabela WHERE alguma_coluna = algum_valor

```
DELETE FROM cliente WHERE codigo_cliente = 3;
```

DQL - Data Query Language

SELECT

Compreende o comando responsável por realizar consultas ao banco de dados. Pode ser combinado com diferentes cláusulas e operadores para a construção de consultas tanto simples quanto mais complexas.

SINTAXE:
SELECT coluna(s) FROM tabela condições

```
SELECT * FROM cliente;

SELECT nome FROM cliente;

SELECT codigo_cliente, nome FROM cliente;
```

DQL - Data Query Language

SELECT

```
SELECT * FROM cliente WHERE codigo_cliente =2;

SELECT * FROM CLIENTE WHERE codigo_cliente >= 1;
```

ORDER BY

```
SELECT * FROM cliente ORDER BY nome;

SELECT * FROM cliente ORDER BY nome DESC;

SELECT * FROM cliente ORDER BY nome, codigo_cliente DESC;
```

Exibindo linhas com valores nulos no início ou final da consulta.

```
INSERT INTO  cliente (telefone, celular) VALUES ('22310900','999023243');

SELECT * FROM cliente ORDER BY nome DESC NULLS FIRST;

SELECT * FROM cliente ORDER BY nome DESC NULLS LAST;
```

DQL - Data Query Language

LIMIT - Limita a quantidade de linhas no resultado da consulta.

Neste exemplo limitamos a consulta ao total de dez registros

```
SELECT * FROM produto LIMIT 10;
```

Neste exemplo limitamos a consulta aos três produtos mais baratos

```
SELECT * FROM produto ORDER BY preco LIMIT 3;
```

Neste exemplo limitamos a consulta aos três produtos mais caros

```
SELECT * FROM produto ORDER BY preco DESC LIMIT 3;
```

OFFSET - Quantidade de linhas que devem ser puladas antes de iniciar a contagem do LIMIT.

DQL - Data Query Language

Neste exemplo limitamos a consulta aos três produtos mais caros

```
SELECT * FROM produto ORDER BY preco DESC LIMIT 3;
```

codigo_produto [PK] integer	nome character varying (40)	descricao text	preco numeric	quantidade_estoque smallint
2	Feijão	Carreteiro	8.5	30
3	Feijão	Granfino	8.5	30
1	Arroz	Tio João	7.5	40

OFFSET - Quantidade de linhas que devem ser puladas antes de iniciar a contagem do LIMIT.

Neste exemplo limitamos a consulta aos três produtos mais caros ignorando os dois primeiros da consulta do exemplo anterior.

codigo_produto [PK] integer	nome character varying (40)	descricao text	preco numeric	quantidade_estoque smallint
1	Arroz	Tio João	7.5	40
10	Arroz	Carreteiro	7.5	4
7	Atum	Gomes da Costa	6.8	42

DQL - Data Query Language

DISTINCT - Utilizada para selecionar dados sem que haja repetição.

SELECT DISTINCT nome_coluna
FROM nome_tabela;

```
select distinct nome from produto order by nome;
```

Operadores Lógicos e Relacionais

- **AND: E** lógico.
  Avalia as condições e retorna as colunas se todas as condições forem verdadeiras.
- **OR: OU** lógico.
  Avalia as condições e retorna as colunas se alguma das condições for verdadeira.
- **NOT:** Negação lógica.
  Retorna as colunas o valor contrário da expressão seja verdadeiro.

```
select * from produto WHERE nome ='Feijão' AND preco > 7;

select * from produto WHERE NOT preco > 8;
```

Operadores Lógicos e Relacionais

Operador	Função
<	Menor
>	Maior
<=	Menor ou igual
>=	Maior ou igual
==	Igual
<>	Diferente

Operador	Função
BETWEEN	`WHERE nome_coluna BETWEEN valor_1 AND valor_2;`
LIKE	`WHERE nome_coluna LIKE padrão;`
IN	`WHERE nome_coluna IN (valor_1,valor_2,..., valor_n);`

Between - Serve para comparar intervalos de dados. No exemplo abaixo comparamos preços que estão entre e no outro exemplo os que não estão entre a faixa.

```
select * from produto WHERE preco BETWEEN 1.0 AND 5.0;

select * from produto WHERE NOT preco BETWEEN 1.0 AND 5.0;
```

Operadores Lógicos e Relacionais

Between

No exemplo abaixo selecionamos os produtos que estão com preço entre 1.0 e 3.0 e também os que estão entre 7.0 e 8.0.

```
1  select * from produto WHERE preco BETWEEN 1.0 AND 3.0 OR
2                              preco BETWEEN 7.0 AND 8.0;
3
4
```

Data Output | Explain | Messages | Notifications

	codigo_produto [PK] integer	nome character varying (40)	descricao text	preco numeric	quantidade_estoque smallint
1	1	Arroz	Tio João	7.5	40
2	5	Farinha de Trigo	Boa Sorte	2.5	8
3	6	Sal	Cisne	2.5	100
4	9	Creme de Leite	Pirancajuba	2.8	15
5	10	Arroz	Carreteiro	7.5	4

Produtos entre 1.0 e 3.0 e quantidade em estoque inferior a 20.

```
select * from produto WHERE preco BETWEEN 1.0 AND 3.0 AND
                            quantidade_estoque < 20
```

Output | Explain | Messages | Notifications

codigo_produto [PK] integer	nome character varying (40)	descricao text	preco numeric	quantidade_estoque smallint
5	Farinha de Trigo	Boa Sorte	2.5	8
9	Creme de Leite	Pirancajuba	2.8	15

Operadores Lógicos e Relacionais

IN

Serve para retorna resultados a partir de uma lista de valores.

```
SELECT * FROM produto WHERE preco IN (2.5, 2.8);

SELECT * FROM produto WHERE nome IN ('Arroz','Feijão');

SELECT * FROM produto WHERE NOT preco  IN (2.5, 2.8);
```

LIKE

Serve para retornar partes de um campo. O like é case sensitive, é utilizado o caracter % como coringa para as buscas.

```
SELECT * FROM produto WHERE nome LIKE 'A%';

SELECT * FROM produto WHERE nome LIKE '%e';

SELECT * FROM produto WHERE nome LIKE '%a%';
```

ILIKE

Faz a mesma coisa do LIKE não sendo um comando case sensitive.

```
SELECT * FROM produto WHERE nome ILIKE 'a%';
```

Operadores Lógicos e Relacionais

ILIKE

Faz a mesma coisa do LIKE não sendo um comando case sensitive.

```
SELECT * FROM produto WHERE nome ILIKE 'a%';
```

Podemos também utilizar o coringa **underline** que procura por uma ou mais ocorrências de um caractere qualquer.

```
SELECT * FROM produto WHERE nome ILIKE '_a_';
```

Output | Explain | Messages | Notifications

codigo_produto [PK] integer	nome character varying (40)	descricao text	preco numeric	quantidade_estoque smallint
6	Sal	Cisne	2.5	100

```
SELECT * FROM produto WHERE nome ILIKE '%_a_%';
```

Output | Explain | Messages | Notifications

codigo_produto [PK] integer	nome character varying (40)	descricao text	preco numeric	quantidade_estoque smallint
4	Macarrão	Adria	6.65	10
5	Farinha de Trigo	Boa Sorte	2.5	8
6	Sal	Cisne	2.5	100
8	Leite Condensado	Nestle	5.9	40

```
SELECT * FROM produto WHERE nome ILIKE 'a____';
```

Output | Explain | Messages | Notifications

codigo_produto [PK] integer	nome character varying (40)	descricao text	preco numeric	quantidade_estoque smallint
1	Arroz	Tio João	7.5	40
10	Arroz	Carreteiro	7.5	4