### ****INDICE****

### ****📌 1. Fundamentos do SQL (o básico que todo mundo precisa saber)****

✅ O que é SQL e para que serve  
✅ Bancos de dados relacionais (MySQL, PostgreSQL, SQL Server, etc.)  
✅ Comando SELECT – consulta de dados ------------------------------------------------------------------ Pagina 3  
✅ Filtrando com WHERE (=, >, <, <>, BETWEEN, LIKE, IN)---------------------------------------------Pagina 8  
✅ Ordenação com ORDER BY------------------------------------------------------------------------------------Pagina 12  
✅ Eliminando duplicatas com DISTINCT-----------------------------------------------------------------------Pagina 16  
✅ Uso do LIMIT e OFFSET (ou TOP no SQL Server)------------------------------------------------------Pagina 20

### ****🔍 2. Funções e Manipulação de Dados****

✅ Funções matemáticas (SUM, AVG, MIN, MAX, COUNT)------------------------------------------------Pagina 24   
✅ Funções de string (UPPER, LOWER, CONCAT, SUBSTRING, TRIM)------------------------------Pagina 28  
✅ Funções de data (NOW, DATEADD, DATEDIFF, EXTRACT)------------------------------------------Pagina 31

### ****🛠 3. Agregações e Agrupamentos (indispensável para análise de dados)****

✅ Agrupamento com GROUP BY------------------------------------------------------------------------------------Pagina 33  
✅ Uso do HAVING para filtrar grupos------------------------------------------------------------------------------------Pagina 33  
✅ Diferença entre WHERE e HAVING------------------------------------------------------------------------------------Pagina 33

### ****🔗 4. Junções entre tabelas (o coração do SQL)****

✅ Tipos de junções (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN)----------Pagina 36  
✅ Autojunções (SELF JOIN)----------------------------------------------------------------------Pagina36  
✅ Uso do ON vs. USING---------------------------------------------------------------------------Pagina 44  
✅ Diferença entre JOIN e UNION----------------------------------------------------------------Pagina 44

### ****🔄 5. Subqueries (para consultas mais avançadas)****

✅ Subquery no WHERE (retorna um único valor)  
✅ Subquery no IN (retorna múltiplos valores)  
✅ Subquery no SELECT (para cálculos dinâmicos)  
✅ Subquery no FROM (também chamada de tabela derivada)

***FIM DO INTERMEDIÁRIO***

### ****📦 6. Manipulação de Dados (DML - Data Manipulation Language)****

✅ Inserção de dados (INSERT INTO)  
✅ Atualização de dados (UPDATE)  
✅ Remoção de dados (DELETE)  
✅ Diferença entre DELETE e TRUNCATE

### ****📊 7. Criação e Estrutura do Banco de Dados (DDL - Data Definition Language)****

✅ Criando tabelas (CREATE TABLE)  
✅ Alterando tabelas (ALTER TABLE)  
✅ Apagando tabelas (DROP TABLE)  
✅ Restrições (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, DEFAULT, NOT NULL)

***>>>>ESTUDAR ATÉ AQUI <<<<***

### ****🔒 8. Índices e Otimização de Consultas****

✅ O que são índices e como melhoram a performance (INDEX)  
✅ Como verificar e interpretar um plano de execução (EXPLAIN ANALYZE)  
✅ Normalização do banco de dados (1FN, 2FN, 3FN)

### ****🚀 9. Stored Procedures, Triggers e Views****

✅ Criando e executando procedures (CREATE PROCEDURE)  
✅ Criando triggers para automação (CREATE TRIGGER)  
✅ Criando views para facilitar consultas (CREATE VIEW)

### ****📌 10. Segurança e Controle de Acesso****

✅ Criando usuários e permissões (GRANT, REVOKE)  
✅ Controle de transações (COMMIT, ROLLBACK, SAVEPOINT)

### ****🔥 Bônus: SQL Avançado e Técnicas para Análise de Dados****

✅ Consultas recursivas com WITH RECURSIVE  
✅ CTEs (WITH ... AS) para tornar consultas mais legíveis  
✅ Funções de janela (ROW\_NUMBER(), RANK(), DENSE\_RANK(), PARTITION BY)  
✅ Modelagem de dados para Business Intelligence (BI)

💡 **Dica:** Se seu foco é **análise de dados**, pode pular algumas partes como triggers e stored procedures, que são mais úteis para desenvolvedores de banco de dados.

Se quiser, posso sugerir exercícios práticos para cada etapa. O que acha? 🚀

O comando SELECT é a base das consultas em SQL, usado para buscar dados em tabelas de um banco de dados. Vou te explicar o tema de forma organizada, do básico ao intermediário, com exemplos práticos.

Pagina 3

## 📌 ****1. Estrutura Básica do SELECT****

A estrutura mais simples do SELECT é:

SELECT coluna1, coluna2, ...FROM nome\_da\_tabela;

* **SELECT**: Indica quais colunas você quer visualizar.
* **FROM**: Indica a tabela de onde os dados virão.

### 🔹 ****Exemplo Básico:****

Tabela clientes:

| **id\_cliente** | **nome** | **cidade** | **idade** |
| --- | --- | --- | --- |
| 1 | João | São Paulo | 30 |
| 2 | Maria | Rio | 25 |

Consulta:

SELECT nome, cidadeFROM clientes;

**Resultado:**

| **nome** | **cidade** |
| --- | --- |
| João | São Paulo |
| Maria | Rio |

## ✨ ****2. Seleção de Todas as Colunas****

Para retornar todas as colunas, usa-se o asterisco \*:

sql

CopiarEditar

SELECT \* FROM clientes;

**Resultado:** Todas as colunas da tabela clientes.

## 🎯 ****3. Renomeando Colunas (AS - Alias)****

O **alias** (AS) serve para dar nomes temporários às colunas ou tabelas:

SELECT nome AS "Nome do Cliente", cidade AS "Cidade de Residência"FROM clientes;

**Resultado:**

| **Nome do Cliente** | **Cidade de Residência** |
| --- | --- |
| João | São Paulo |
| Maria | Rio |

## 🏷 ****4. Filtrando Dados com WHERE****

A cláusula WHERE filtra linhas com base em condições.

### 🔹 ****Exemplo:****

SELECT nome, cidadeFROM clientesWHERE cidade = 'São Paulo';

**Resultado:**

| **nome** | **cidade** |
| --- | --- |
| João | São Paulo |

### 🔹 ****Operadores Comuns em WHERE:****

* = (igual)
* <> ou != (diferente)
* <, >, <=, >= (menor, maior, etc.)
* BETWEEN (entre dois valores)
* IN (dentro de uma lista)
* LIKE (correspondência de padrão)
* IS NULL (valores nulos)

## 🔄 ****5. Ordenando Resultados (ORDER BY)****

A cláusula ORDER BY organiza os resultados:

* ASC (crescente, padrão)
* DESC (decrescente)

### 🔹 ****Exemplo:****

sql

CopiarEditar

SELECT nome, idadeFROM clientesORDER BY idade DESC;

**Resultado:**

| **nome** | **idade** |
| --- | --- |
| João | 30 |
| Maria | 25 |

## 🔢 ****6. Limitando Resultados (LIMIT)****

O LIMIT restringe o número de linhas retornadas.

sql

CopiarEditar

SELECT \* FROM clientes

LIMIT 1;

**Resultado:** Retorna apenas a primeira linha.

## 🎛 ****7. Removendo Duplicatas (DISTINCT)****

DISTINCT retorna valores únicos em uma coluna.

sql

CopiarEditar

SELECT DISTINCT cidadeFROM clientes;

**Resultado:**

| **cidade** |
| --- |
| São Paulo |
| Rio |

## 🧮 ****8. Funções de Agregação****

Essas funções realizam cálculos sobre os dados:

* COUNT() – Conta registros
* SUM() – Soma valores
* AVG() – Média
* MIN() – Mínimo
* MAX() – Máximo

### 🔹 ****Exemplo:****

sql

CopiarEditar

SELECT COUNT(\*) AS total\_clientes, AVG(idade) AS idade\_mediaFROM clientes;

**Resultado:**

| **total\_clientes** | **idade\_media** |
| --- | --- |
| 2 | 27.5 |

## 🧩 ****9. Agrupamento de Dados (GROUP BY)****

A cláusula GROUP BY agrupa registros para usar funções de agregação em grupos.

### 🔹 ****Exemplo:****

sql

CopiarEditar

SELECT cidade, COUNT(\*) AS total\_por\_cidadeFROM clientesGROUP BY cidade;

**Resultado:**

| **cidade** | **total\_por\_cidade** |
| --- | --- |
| São Paulo | 1 |
| Rio | 1 |

## 🔍 ****10. Filtrando Grupos (HAVING)****

HAVING filtra os resultados depois do GROUP BY, diferentemente do WHERE que filtra linhas antes do agrupamento.

### 🔹 ****Exemplo:****

sql

CopiarEditar

SELECT cidade, COUNT(\*) AS total\_por\_cidadeFROM clientesGROUP BY cidadeHAVING COUNT(\*) > 1;

**Resultado:** Filtra cidades com mais de 1 cliente.

## ⚡ ****11. Consultas com Cálculos e Expressões****

Você pode fazer cálculos diretamente no SELECT.

sql

CopiarEditar

SELECT nome, idade, idade \* 12 AS idade\_em\_mesesFROM clientes;

**Resultado:**

| **nome** | **idade** | **idade\_em\_meses** |
| --- | --- | --- |
| João | 30 | 360 |
| Maria | 25 | 300 |

## 🔄 ****12. Consultas Aninhadas (Subqueries)****

Subqueries permitem usar o resultado de uma consulta dentro de outra.

sql

CopiarEditar

SELECT nome, idadeFROM clientesWHERE idade > (SELECT AVG(idade) FROM clientes);

**Resultado:** Clientes com idade acima da média.

## 🎒 ****Práticas para Fixação****

Tente resolver:

1. Liste todos os clientes com idade maior que 25 anos.
2. Mostre a cidade e o número de clientes em cada uma.
3. Exiba o cliente mais velho.
4. Calcule a idade média dos clientes de São Paulo.
5. Filtre cidades que possuem mais de 2 clientes.

Pagina 8

O WHERE é uma cláusula essencial em SQL para filtrar registros com base em condições específicas. Vamos abordar o tema do **básico ao intermediário**, explicando cada operador, mostrando exemplos e propondo desafios.

## 🎯 ****1. Conceito da cláusula WHERE****

A cláusula WHERE filtra linhas antes que elas sejam retornadas em uma consulta. Ela pode ser combinada com diferentes operadores para definir condições.

### 🔹 ****Sintaxe básica:****

SELECT coluna1, coluna2, ...FROM nome\_da\_tabela

WHERE condição;

### 🔹 ****Exemplo simples:****

SELECT nome, idade

FROM clientes

WHERE idade > 30;

**Resultado:** Retorna todos os clientes com idade maior que 30 anos.

## 🔢 ****2. Operadores de Comparação****

### ✅ ****= (Igual)****

Filtra registros com valor exatamente igual ao especificado.

SELECT nome, cidade

FROM clientes

WHERE cidade = 'São Paulo';

### ❌ ****<> ou != (Diferente)****

Filtra registros cujo valor **não** corresponde ao especificado.

SELECT nome, cidade

FROM clientes

WHERE cidade <> 'Rio de Janeiro';

**Observação:** <> e != funcionam da mesma forma.

### 📈 ****> (Maior que) e < (Menor que)****

* >: Valores maiores que o especificado.
* <: Valores menores que o especificado.

SELECT nome, idade

FROM clientes

WHERE idade > 25;

Retorna clientes com idade acima de 25 anos.

### 🟢 ****>= (Maior ou igual a) e <= (Menor ou igual a)****

* >=: Valores maiores ou iguais.
* <=: Valores menores ou iguais.

SELECT nome, idade

FROM clientes

WHERE idade <= 30;

## 🏛 ****3. BETWEEN (Entre valores)****

O BETWEEN filtra registros dentro de um intervalo, incluindo os limites.

### 🔹 ****Exemplo:****

SELECT nome, idade

FROM clientes

WHERE idade BETWEEN 25 AND 35;

**Resultado:** Retorna clientes com idade entre 25 e 35 anos, inclusive.

## 🔍 ****4. LIKE (Correspondência de Padrão)****

O operador LIKE filtra dados que correspondem a um padrão usando curingas:

* %: Representa qualquer sequência de caracteres (incluindo nenhuma).
* \_: Representa exatamente um caractere.

### 🔹 ****Exemplos:****

-- Nomes que começam com 'M'

SELECT nome

FROM clientes

WHERE nome LIKE 'M%';

-- Nomes que terminam com 'a'

SELECT nome

FROM clientes

WHERE nome LIKE '%a';

-- Nomes com exatamente 5 letras

SELECT nome

FROM clientes

WHERE nome LIKE '\_\_\_\_\_';

## 🧩 ****5. IN (Valor dentro de uma lista)****

O IN filtra registros com valores que correspondem a uma lista específica.

### 🔹 ****Exemplo:****

SELECT nome, cidade

FROM clientes

WHERE cidade IN ('São Paulo', 'Rio de Janeiro', 'Belo Horizonte');

**Resultado:** Retorna clientes dessas três cidades.

## ⚡ ****6. Combinando Condições com AND e OR****

* AND: Todas as condições devem ser verdadeiras.
* OR: Pelo menos uma condição deve ser verdadeira.

### 🔹 ****Exemplos:****

-- Clientes de São Paulo com idade acima de 30

SELECT nome, cidade, idade

FROM clientes

WHERE cidade = 'São Paulo' AND idade > 30;

-- Clientes de São Paulo ou com idade abaixo de 25

SELECT nome, cidade, idade

FROM clientes

WHERE cidade = 'São Paulo' OR idade < 25;

## 🚀 ****7. Filtrando Valores Nulos (IS NULL / IS NOT NULL)****

* IS NULL: Filtra registros com valor nulo.
* IS NOT NULL: Filtra registros que **não** são nulos.

### 🔹 ****Exemplo:****

SELECT nome, telefone

FROM clientes

WHERE telefone IS NULL;

**Resultado:** Retorna clientes que não têm telefone cadastrado.

## 🏗

## ****8. Filtrando com Subqueries****

Combina WHERE com subconsultas para filtros avançados.

### 🔹 ****Exemplo:****

SELECT nome, idade

FROM clientes

WHERE idade > (SELECT AVG(idade) FROM clientes);

**Resultado:** Retorna clientes com idade acima da média.

## 📝 ****9. Práticas para Fixação****

Tente resolver esses desafios:

1. Liste clientes com idade diferente de 30 anos.
2. Mostre clientes que moram em 'São Paulo' ou 'Belo Horizonte'.
3. Encontre clientes cujo nome termina com a letra 'o'.
4. Liste clientes com idade entre 20 e 40 anos.
5. Mostre clientes que possuem telefone cadastrado.
6. Filtre clientes de cidades específicas usando IN.
7. Liste clientes cuja idade seja maior que a idade média.

Pagina 12

O ORDER BY é uma cláusula essencial em SQL que organiza os resultados de uma consulta em ordem específica. Vamos explorá-lo do **básico ao intermediário**, abordando sintaxe, tipos de ordenação, múltiplos critérios e recursos avançados.

## 🎯 ****1. Conceito e Sintaxe Básica do ORDER BY****

A cláusula ORDER BY organiza os dados retornados por uma consulta em ordem **crescente** ou **decrescente**.

### 🔹 ****Sintaxe:****

SELECT coluna1, coluna2, ...

FROM nome\_da\_tabela

ORDER BY coluna1 [ASC | DESC], coluna2 [ASC | DESC];

* ASC (Ascending): Ordem crescente (padrão).
* DESC (Descending): Ordem decrescente.

### 🔹 ****Exemplo Básico:****

Tabela clientes:

| **id\_cliente** | **nome** | **idade** | **cidade** |
| --- | --- | --- | --- |
| 1 | João | 30 | São Paulo |
| 2 | Maria | 25 | Rio |
| 3 | Pedro | 35 | Salvador |
| 4 | Ana | 28 | Recife |

**Consulta:** Ordenar clientes pela idade em ordem crescente.

SELECT nome, idade

FROM clientes

ORDER BY idade;

**Resultado:**

| **nome** | **idade** |
| --- | --- |
| Maria | 25 |
| Ana | 28 |
| João | 30 |
| Pedro | 35 |

## 📈 ****2. Ordenação Decrescente (DESC)****

Para inverter a ordem, use DESC:

SELECT nome, idadeFROM clientesORDER BY idade DESC;

**Resultado:**

| **nome** | **idade** |
| --- | --- |
| Pedro | 35 |
| João | 30 |
| Ana | 28 |
| Maria | 25 |

## 🏗 ****3. Ordenação por Múltiplas Colunas****

Você pode ordenar por mais de uma coluna. A ordenação secundária é aplicada quando há valores iguais na primeira coluna.

### 🔹 ****Exemplo:****

SELECT nome, cidade, idade

FROM clientes

ORDER BY cidade ASC, idade DESC;

**Resultado:**

| **nome** | **cidade** | **idade** |
| --- | --- | --- |
| Ana | Recife | 28 |
| Pedro | Salvador | 35 |
| João | São Paulo | 30 |
| Maria | Rio | 25 |

## 🎛 ****4. Ordenação por Colunas Derivadas****

Você pode ordenar por colunas calculadas diretamente no SELECT.

### 🔹 ****Exemplo:****

SELECT nome, idade, idade \* 12 AS idade\_em\_meses

FROM clientes

ORDER BY idade\_em\_meses DESC;

**Resultado:**

| **nome** | **idade** | **idade\_em\_meses** |
| --- | --- | --- |
| Pedro | 35 | 420 |
| João | 30 | 360 |
| Ana | 28 | 336 |
| Maria | 25 | 300 |

## 🔢 ****5. Ordenação por Índice da Coluna****

Em vez de usar o nome da coluna, você pode usar a posição dela na cláusula SELECT.  
(⚠️ Uso menos recomendado para consultas complexas, pois pode afetar a legibilidade.)

### 🔹 ****Exemplo:****

sql

CopiarEditar

SELECT nome, idade, cidadeFROM clientesORDER BY 2 DESC; -- 2 refere-se à coluna 'idade'

## 🧮 ****6. Combinando ORDER BY com Funções de Agregação****

Ao usar GROUP BY, você pode ordenar os grupos pelo resultado de funções de agregação.

### 🔹 ****Exemplo:****

Tabela vendas:

| **id\_venda** | **vendedor** | **valor\_venda** |
| --- | --- | --- |
| 1 | João | 5000 |
| 2 | Maria | 7000 |
| 3 | João | 3000 |
| 4 | Ana | 9000 |

**Consulta:** Total de vendas por vendedor, ordenado pelo total.

sql

CopiarEditar

SELECT vendedor, SUM(valor\_venda) AS total\_vendasFROM vendasGROUP BY vendedorORDER BY total\_vendas DESC;

**Resultado:**

| **vendedor** | **total\_vendas** |
| --- | --- |
| Ana | 9000 |
| Maria | 7000 |
| João | 8000 |

## 🔀 ****7. Ordenação com Valores Nulos (NULLS FIRST / NULLS LAST)****

Ao ordenar, valores nulos podem ser posicionados explicitamente no topo ou no final.

### 🔹 ****Exemplo:****

Tabela clientes com alguns valores nulos:

| **nome** | **idade** |
| --- | --- |
| João | 30 |
| Maria | NULL |
| Pedro | 35 |
| Ana | NULL |

sql

CopiarEditar

SELECT nome, idadeFROM clientesORDER BY idade ASC NULLS LAST;

**Resultado:**

| **nome** | **idade** |
| --- | --- |
| João | 30 |
| Pedro | 35 |
| Maria | NULL |
| Ana | NULL |

## ⚡ ****8. Ordenação Aleatória (ORDER BY RANDOM())****

Em alguns bancos de dados, é possível ordenar aleatoriamente, útil em sorteios, por exemplo.

### 🔹 ****Exemplo em PostgreSQL:****

sql

CopiarEditar

SELECT nomeFROM clientesORDER BY RANDOM();

## 📝 ****9. Práticas para Fixação****

Tente resolver:

1. Liste os clientes ordenados pelo nome em ordem alfabética inversa.
2. Mostre os clientes ordenados por cidade (crescente) e idade (decrescente).
3. Exiba o total de vendas por vendedor em ordem crescente.
4. Ordene os clientes pela idade, posicionando os valores nulos por último.
5. Liste produtos com preços calculados (preço \* quantidade), ordenados do mais caro para o mais barato.

Pagina 16

O DISTINCT em SQL é usado para **eliminar registros duplicados** nos resultados de uma consulta, retornando **apenas valores únicos**. Vamos explorar o uso dessa cláusula do **básico ao intermediário**, cobrindo a sintaxe, exemplos práticos e alguns conceitos avançados.

## 🎯 ****1. Conceito e Sintaxe Básica do DISTINCT****

A cláusula DISTINCT remove duplicatas do conjunto de resultados retornando **apenas linhas distintas**.

### 🔹 ****Sintaxe básica:****

SELECT DISTINCT coluna1, coluna2, ...

FROM nome\_da\_tabela;

* O DISTINCT se aplica a todas as colunas listadas no SELECT.

### 🔹 ****Exemplo básico:****

Suponha a tabela clientes:

| **id\_cliente** | **nome** | **cidade** |
| --- | --- | --- |
| 1 | João | São Paulo |
| 2 | Maria | Rio |
| 3 | João | São Paulo |
| 4 | Ana | Recife |
| 5 | Pedro | Salvador |
| 6 | Maria | Rio |

**Consulta:**

SELECT DISTINCT nome

FROM clientes;

**Resultado:**

| **nome** |
| --- |
| João |
| Maria |
| Ana |
| Pedro |

## 📚 ****2. DISTINCT em Múltiplas Colunas****

Quando usamos DISTINCT em mais de uma coluna, ele retorna combinações únicas desses valores.

### 🔹 ****Exemplo:****

SELECT DISTINCT nome, cidade

FROM clientes;

**Resultado:**

| **nome** | **cidade** |
| --- | --- |
| João | São Paulo |
| Maria | Rio |
| Ana | Recife |
| Pedro | Salvador |

💡 **Observação:** Se duas linhas tiverem o mesmo nome e a mesma cidade, apenas uma será exibida.

## 🧩 ****3. Diferença entre DISTINCT e GROUP BY****

Embora ambos possam retornar valores únicos, suas finalidades são diferentes:

* DISTINCT: Remove duplicatas diretamente.
* GROUP BY: Agrupa registros para cálculos agregados.

### 🔹 ****Exemplo com GROUP BY:****

SELECT nome, COUNT(\*) AS total

FROM clientes

GROUP BY nome;

**Resultado:**

| **nome** | **total** |
| --- | --- |
| João | 2 |
| Maria | 2 |
| Ana | 1 |
| Pedro | 1 |

💡 **Use** DISTINCT quando precisar apenas de valores únicos, **sem agregações**.

## 🎛 ****4. DISTINCT em Colunas Calculadas****

Você pode aplicar o DISTINCT em expressões e cálculos.

### 🔹 ****Exemplo:****

Tabela produtos:

| **id\_produto** | **nome\_produto** | **preco** | **quantidade** |
| --- | --- | --- | --- |
| 1 | Teclado | 100 | 2 |
| 2 | Mouse | 50 | 5 |
| 3 | Monitor | 600 | 1 |
| 4 | Teclado | 100 | 3 |

**Consulta:** Preços totais únicos (preço \* quantidade):

SELECT DISTINCT preco \* quantidade AS preco\_total

FROM produtos;

**Resultado:**

| **preco\_total** |
| --- |
| 200 |
| 250 |
| 600 |
| 300 |

## 🔢 ****5. Combinando DISTINCT com Funções de Agregação****

Em alguns casos, é possível aplicar o DISTINCT dentro de funções agregadas, como COUNT, para contar apenas valores únicos.

### 🔹 ****Exemplo:****

SELECT COUNT(DISTINCT cidade) AS total\_cidades

FROM clientes;

**Resultado:**

| **total\_cidades** |
| --- |
| 4 |

## 🧮 ****6. DISTINCT em Subconsultas****

Você pode usar o DISTINCT em subconsultas para filtrar resultados antes de combiná-los com outras consultas.

### 🔹 ****Exemplo:****

SELECT nome

FROM clientes

WHERE cidade IN (

SELECT DISTINCT cidade

FROM clientes

WHERE cidade <> 'São Paulo'

);

**Resultado:** Retorna nomes de clientes de todas as cidades, exceto São Paulo, sem duplicatas de cidades.

## 🚀 ****7. LIMITAÇÕES DO DISTINCT****

* DISTINCT se aplica a todas as colunas no SELECT, o que pode gerar confusão se houver pequenas diferenças entre registros.
* Pode afetar o desempenho em grandes conjuntos de dados, pois o banco de dados precisa verificar duplicatas em todas as colunas especificadas.

## 📝 ****8. Práticas para Fixação****

Tente resolver:

1. Liste todas as cidades únicas da tabela de clientes.
2. Mostre combinações únicas de nome e cidade.
3. Conte o número de produtos com preços distintos.
4. Liste preços totais únicos (preço \* quantidade) de produtos.
5. Exiba o número de cidades distintas em que os clientes residem.

Pagina 20

Vamos explorar **LIMIT** e **OFFSET**, bem como o equivalente **TOP** no SQL Server. A compreensão desses conceitos ajudará a otimizar a exibição e a manipulação de dados em consultas SQL, principalmente quando trabalhamos com grandes volumes de dados.

### 1. ****LIMIT e OFFSET****

Esses comandos são usados principalmente para controlar a quantidade de dados retornados em uma consulta SQL. Eles são comuns em bancos de dados como MySQL, PostgreSQL e SQLite.

#### ****LIMIT****

O **LIMIT** limita o número de registros retornados por uma consulta. Isso é útil quando você não precisa de todos os resultados de uma consulta, apenas uma quantidade específica.

**Sintaxe:**

sql

CopiarEditar

SELECT coluna1, coluna2FROM tabela

LIMIT N;

Onde:

* **N** é o número de registros que você deseja retornar.

Exemplo:

sql

CopiarEditar

SELECT nome, idadeFROM clientes

LIMIT 5;

Este comando retorna os primeiros 5 registros da tabela clientes.

#### ****OFFSET****

O **OFFSET** é usado para pular um número específico de registros antes de retornar os resultados. É útil quando você está implementando paginação de dados (por exemplo, em um sistema de busca, onde você mostra um subconjunto de resultados por vez).

**Sintaxe:**

sql

CopiarEditar

SELECT coluna1, coluna2FROM tabela

LIMIT N OFFSET M;

Onde:

* **N** é o número de registros que você deseja retornar.
* **M** é o número de registros que você deseja pular.

Exemplo:

sql

CopiarEditar

SELECT nome, idadeFROM clientes

LIMIT 5 OFFSET 10;

Este comando retorna 5 registros, começando do 11º registro (pula os primeiros 10).

#### ****Combinação de LIMIT e OFFSET****

Em muitos casos, você vai usar o **LIMIT** e o **OFFSET** juntos para implementar a paginação de dados.

**Exemplo:** Suponha que você tem 1000 registros na tabela e quer exibir 10 registros por página. Você pode usar as cláusulas **LIMIT** e **OFFSET** para controlar a página que será exibida:

* Página 1: LIMIT 10 OFFSET 0
* Página 2: LIMIT 10 OFFSET 10
* Página 3: LIMIT 10 OFFSET 20

Essa abordagem é muito comum em sistemas que exibem resultados paginados, como um site de e-commerce ou uma plataforma de consultas de dados.

### 2. ****TOP no SQL Server****

No **SQL Server**, não usamos **LIMIT** e **OFFSET**, mas sim o comando **TOP**. A função do **TOP** é similar ao **LIMIT**, mas com uma sintaxe diferente.

#### ****TOP****

O **TOP** limita o número de registros retornados pela consulta. Você pode usá-lo para retornar os primeiros **N** registros de uma consulta.

**Sintaxe:**

sql

CopiarEditar

SELECT TOP N coluna1, coluna2FROM tabela;

Onde:

* **N** é o número de registros que você deseja retornar.

Exemplo:

sql

CopiarEditar

SELECT TOP 5 nome, idadeFROM clientes;

Esse comando retorna os primeiros 5 registros da tabela clientes.

#### ****TOP com PERCENT****

Você também pode usar o **TOP** com a palavra-chave **PERCENT** para retornar uma porcentagem dos registros.

**Sintaxe:**

sql

CopiarEditar

SELECT TOP N PERCENT coluna1, coluna2FROM tabela;

Exemplo:

sql

CopiarEditar

SELECT TOP 10 PERCENT nome, idadeFROM clientes;

Este comando retorna os primeiros 10% dos registros da tabela clientes.

#### ****TOP com ORDER BY****

É comum usar o **TOP** em conjunto com o **ORDER BY** para garantir que você está pegando os primeiros registros de acordo com algum critério específico, como a maior ou menor data, ou valores mais altos.

**Exemplo:**

sql

CopiarEditar

SELECT TOP 5 nome, idadeFROM clientesORDER BY idade DESC;

Esse comando retorna os 5 primeiros registros com base na idade, em ordem decrescente.

### 3. ****Comparação entre LIMIT/OFFSET e TOP****

| **Funcionalidade** | **LIMIT/OFFSET (MySQL, PostgreSQL, SQLite)** | **TOP (SQL Server)** |
| --- | --- | --- |
| **Limitar resultados** | Usado para limitar o número de registros | Usado para limitar o número de registros |
| **Pular registros** | OFFSET é usado para pular registros antes de retornar o resultado | Não há equivalente direto ao OFFSET, mas pode ser simulado com a cláusula ROW\_NUMBER() |
| **Exemplo** | LIMIT 5 OFFSET 10 | SELECT TOP 5 |

### 4. ****Exemplo Completo (com Paginação)****

#### ****MySQL/PostgreSQL****

Se você deseja implementar a paginação para exibir resultados de 10 em 10 registros, pode usar a combinação de **LIMIT** e **OFFSET**:

**Página 1:**

sql

CopiarEditar

SELECT nome, idadeFROM clientes

LIMIT 10 OFFSET 0;

**Página 2:**

sql

CopiarEditar

SELECT nome, idadeFROM clientes

LIMIT 10 OFFSET 10;

**Página 3:**

sql

CopiarEditar

SELECT nome, idadeFROM clientes

LIMIT 10 OFFSET 20;

#### ****SQL Server****

No SQL Server, para implementar uma paginação semelhante, você pode usar **TOP** em combinação com a cláusula **ROW\_NUMBER()** para criar uma numeração sequencial das linhas e então filtrar as linhas pela numeração.

sql

CopiarEditar

WITH CTE AS (

SELECT nome, idade, ROW\_NUMBER() OVER (ORDER BY idade) AS row\_num

FROM clientes

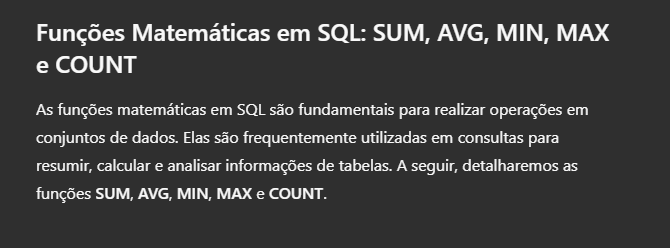
)SELECT nome, idadeFROM CTEWHERE row\_num BETWEEN 11 AND 20;

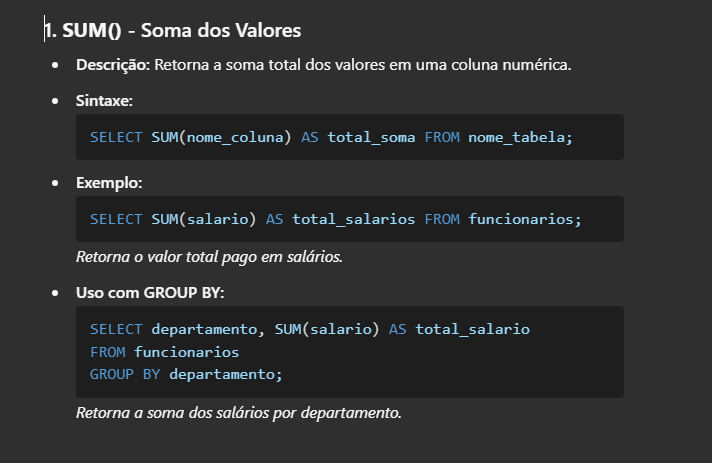
Esse exemplo retorna registros da página 2 (linhas 11 a 20), com base na coluna idade ordenada.

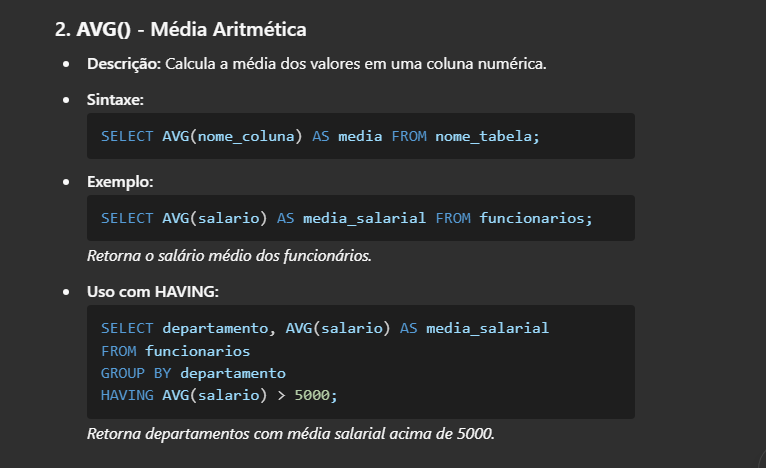
### Conclusão

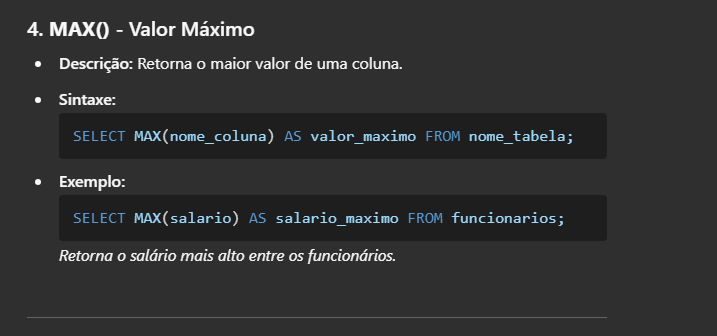
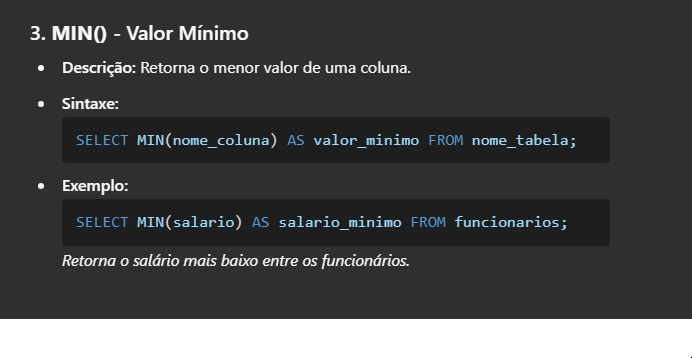
* **LIMIT** e **OFFSET** são ideais quando você precisa controlar a quantidade de dados retornados em consultas SQL, especialmente quando trabalha com sistemas de paginação.
* **TOP**, no SQL Server, serve para limitar o número de registros retornados, mas a funcionalidade de pular registros é feita de maneira diferente, com o uso de funções como **ROW\_NUMBER()**.
* Ambos os conceitos são valiosos para melhorar a performance e a experiência do usuário ao trabalhar com grandes volumes de dados.

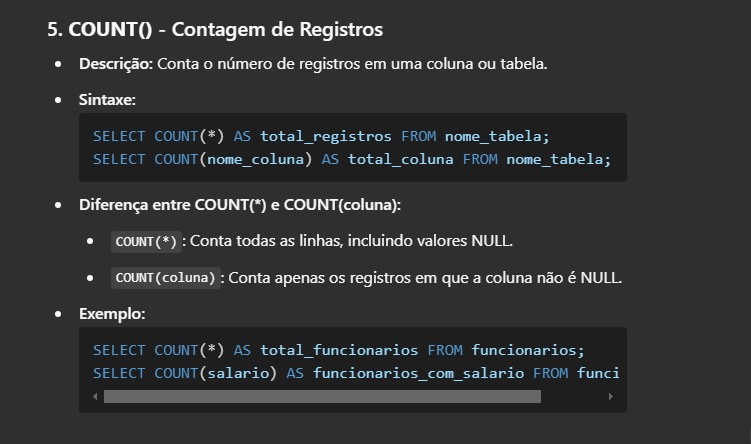
Pagina 24

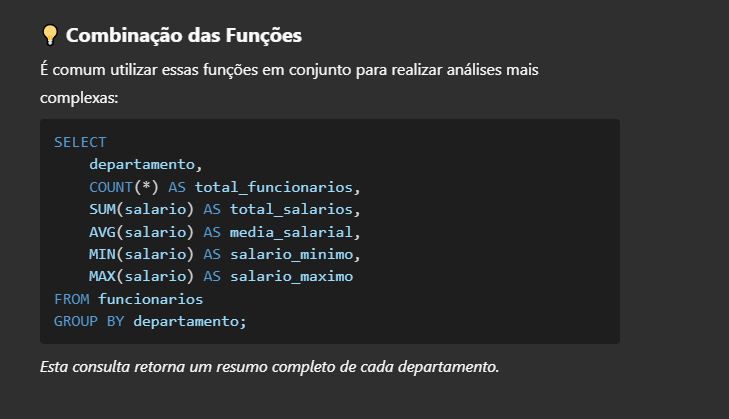




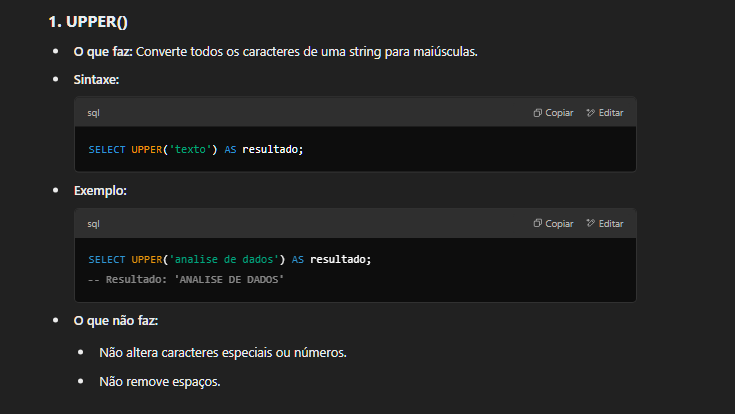


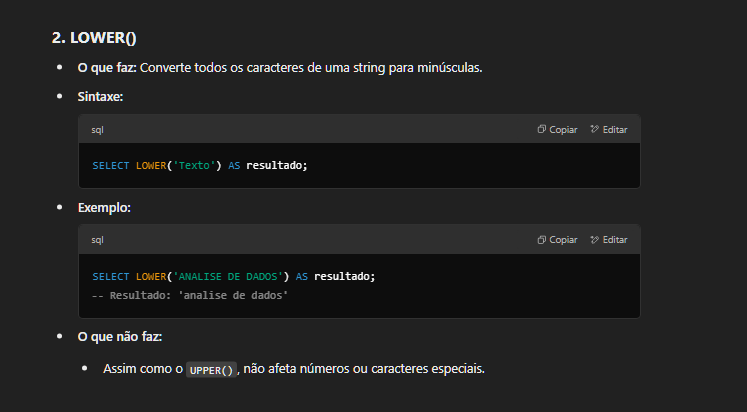


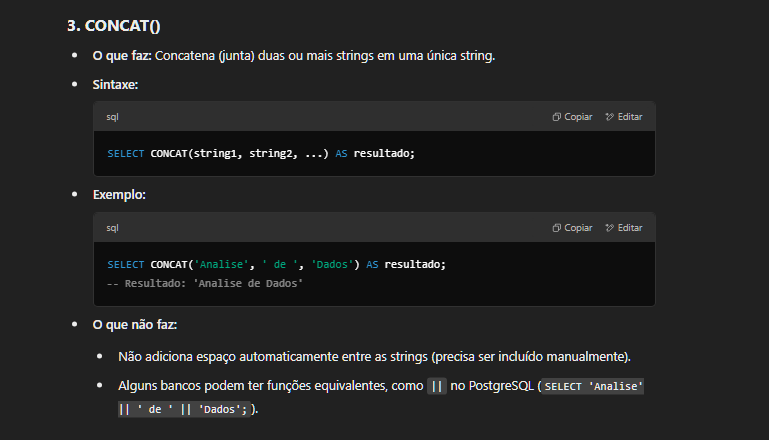




Pagina 28



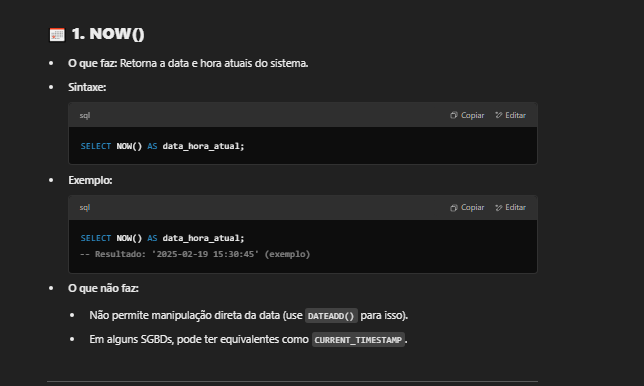




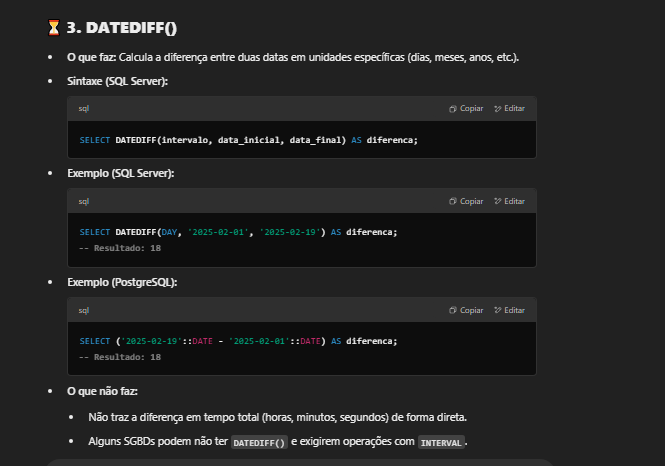


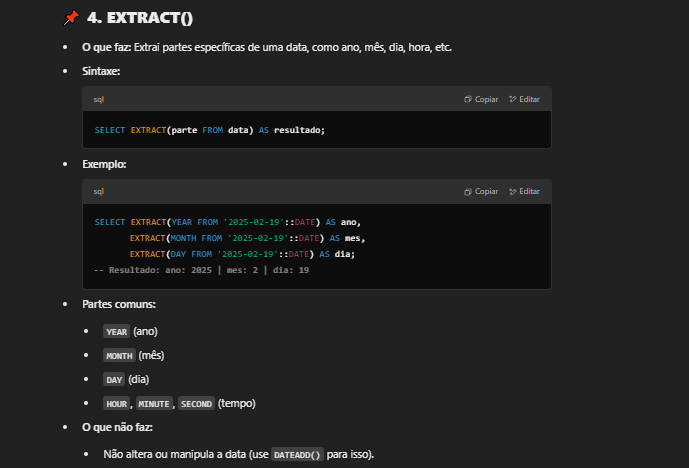


Pagina 31

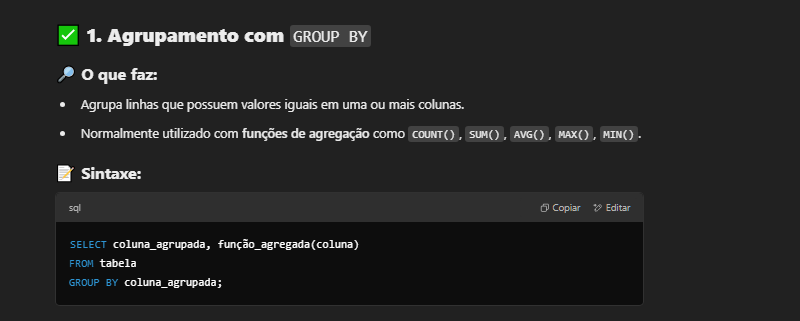


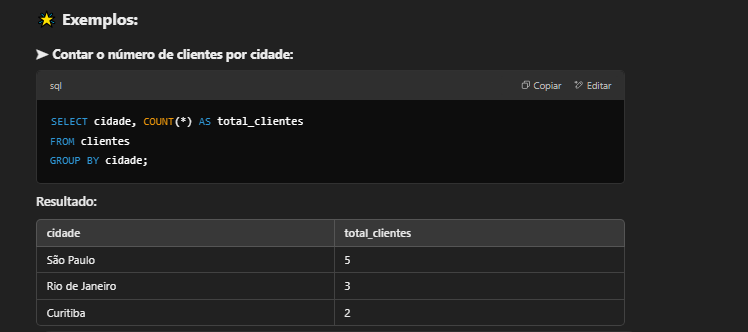


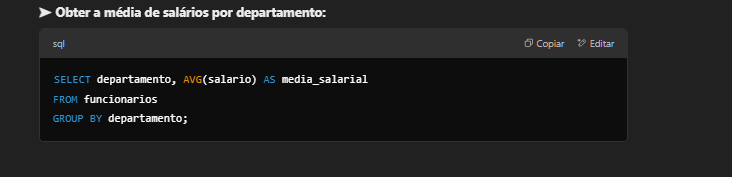


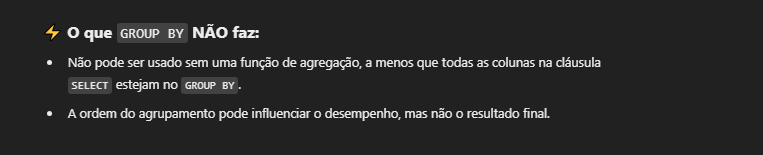


Pagina 33

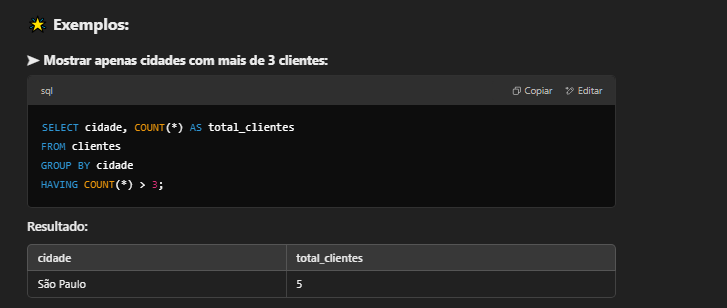


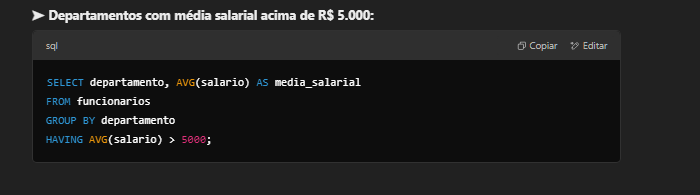


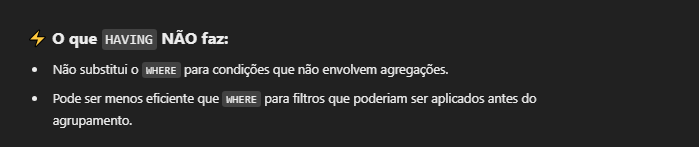




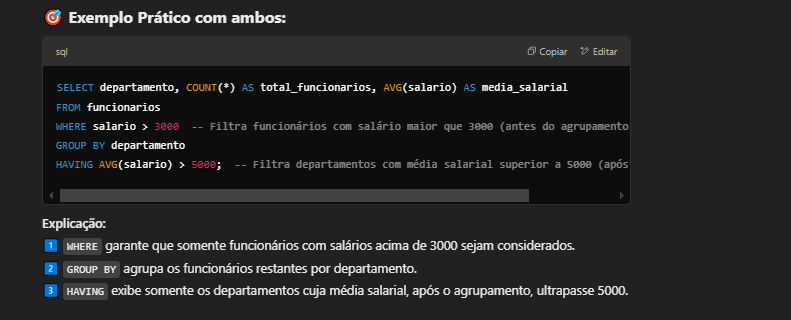


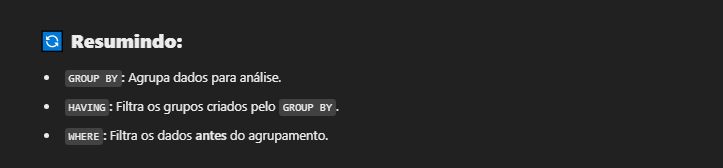








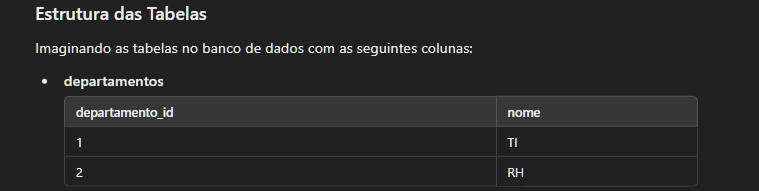




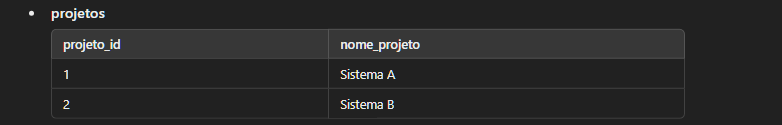
Pagina 36

***INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN. SELF JOIN***

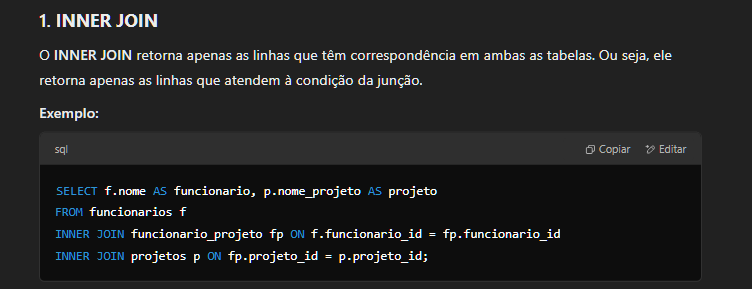
***SCHEMAS - ESQUEMAS***

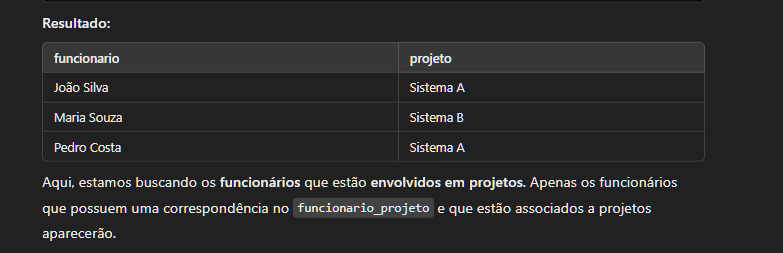


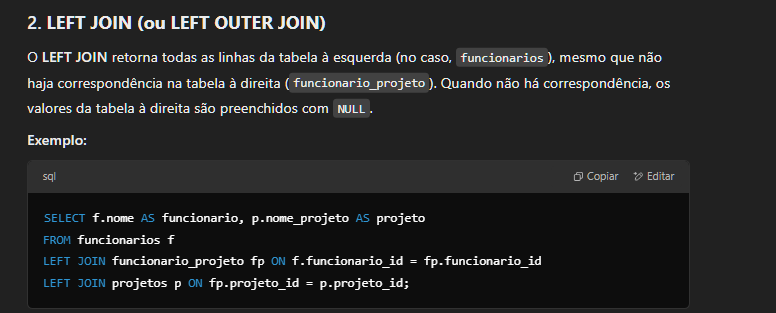


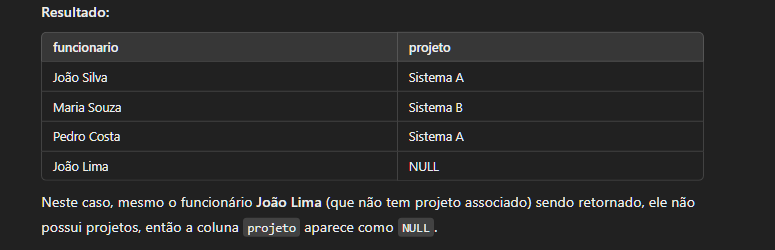


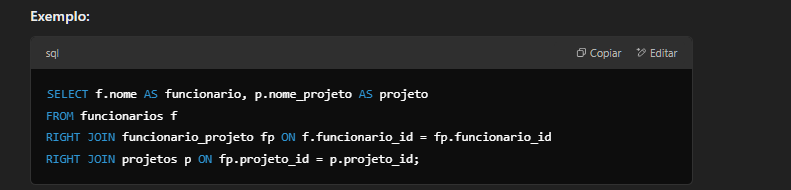
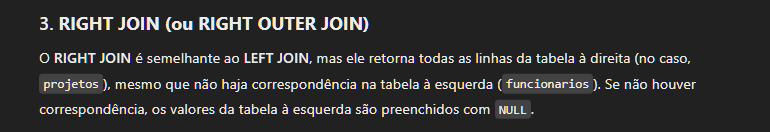


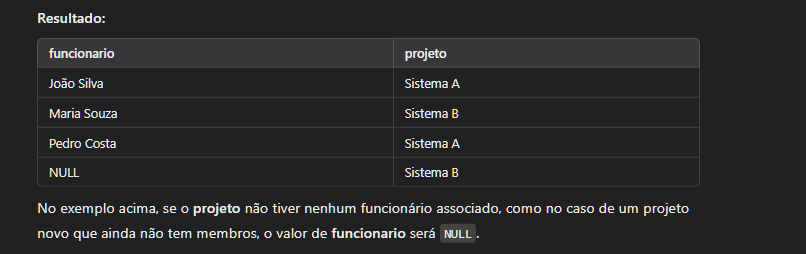


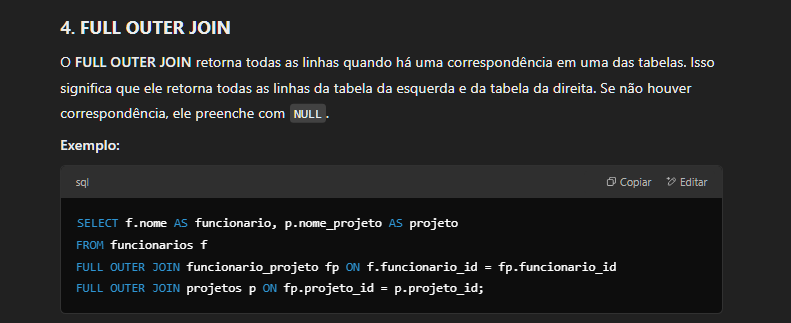


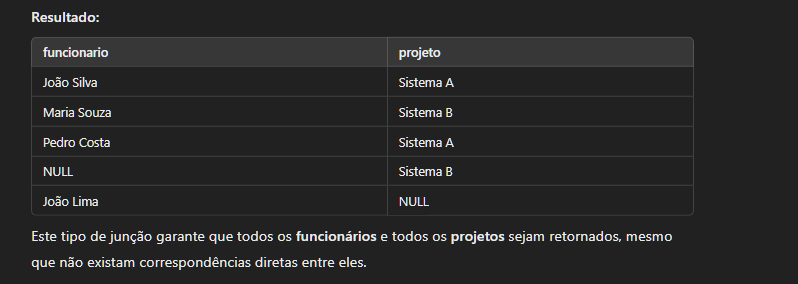


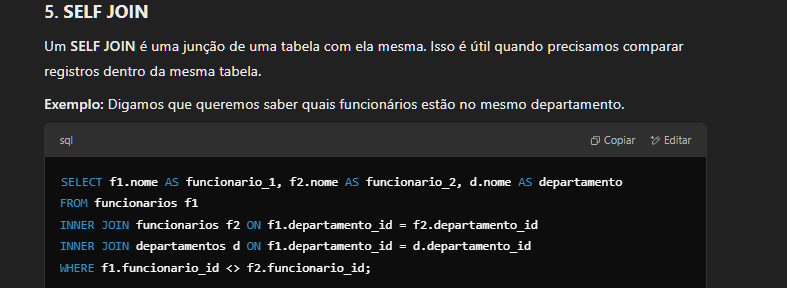


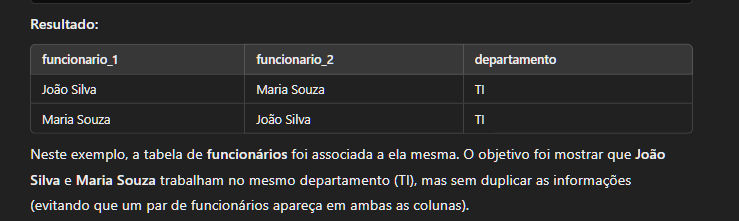


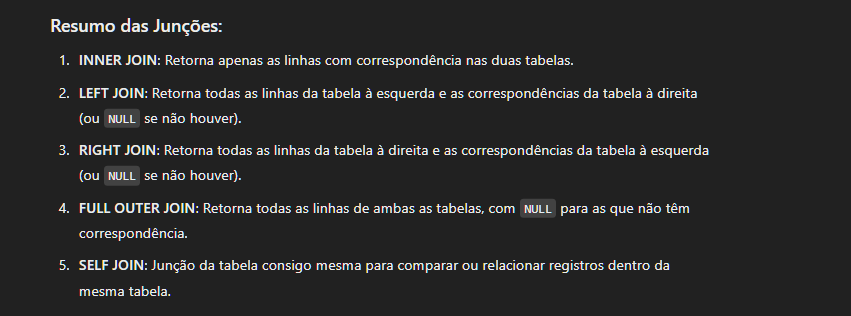


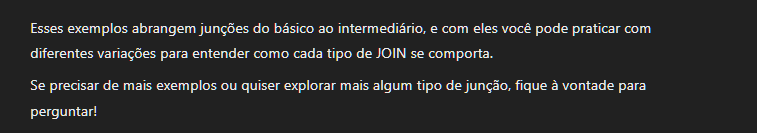


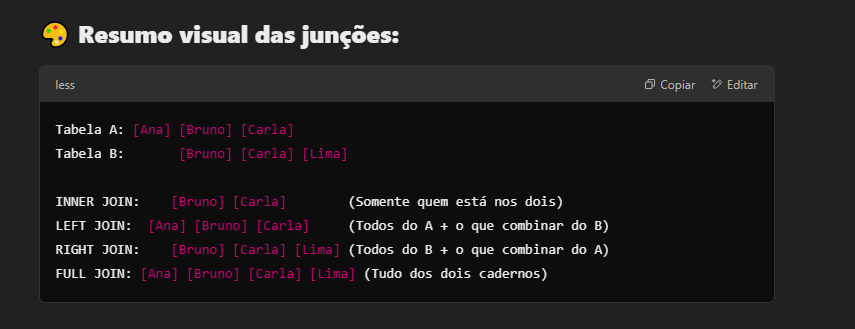


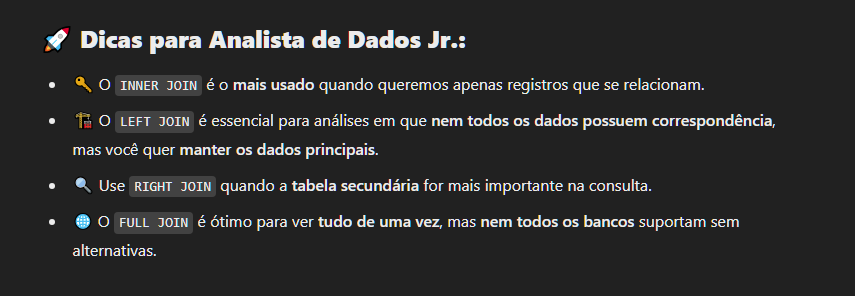


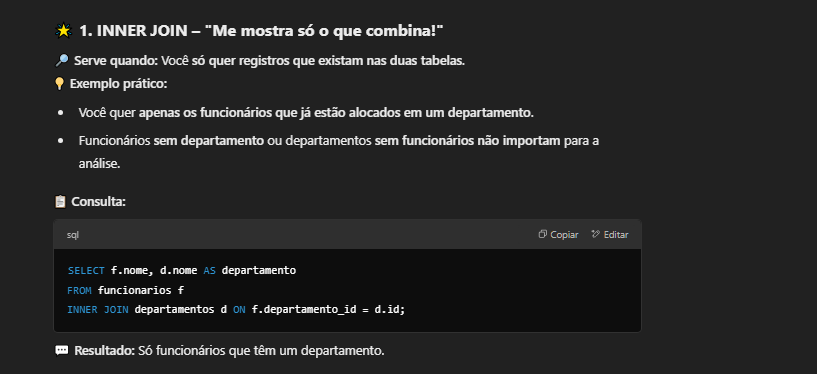


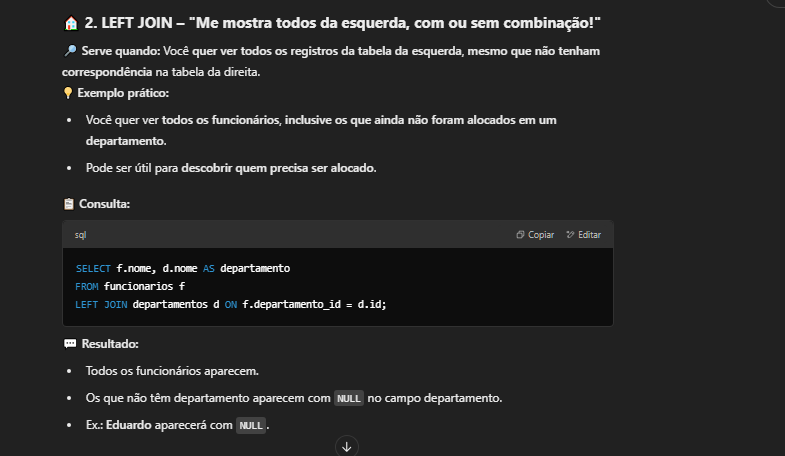


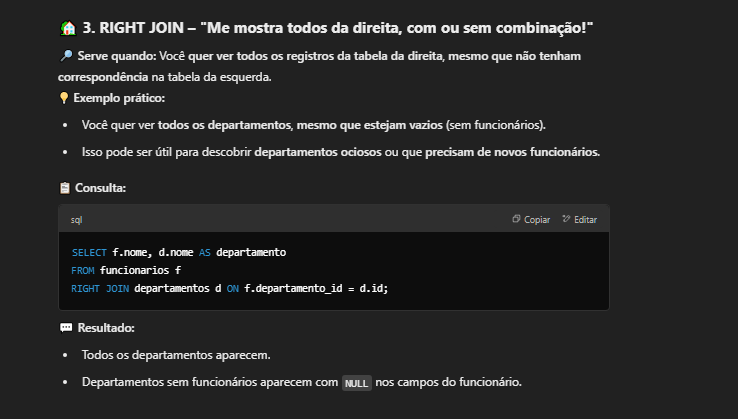


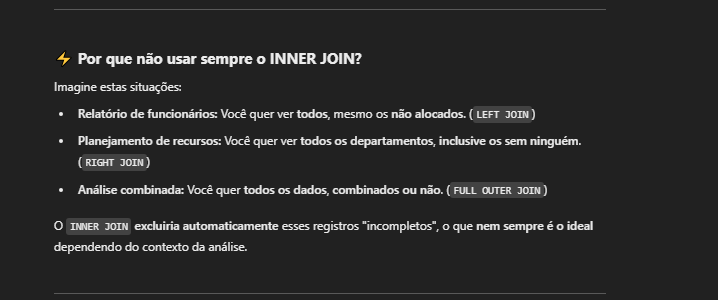


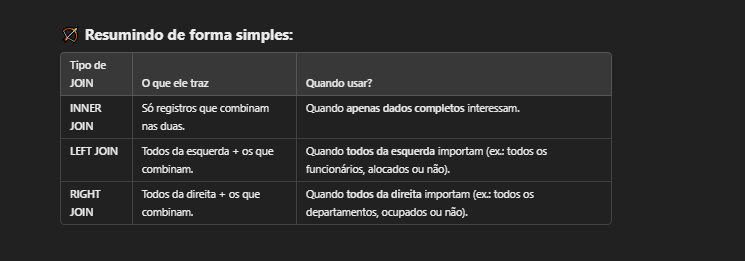




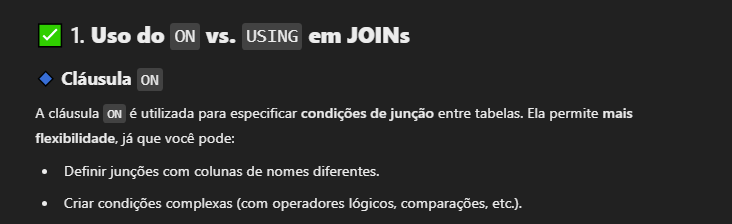


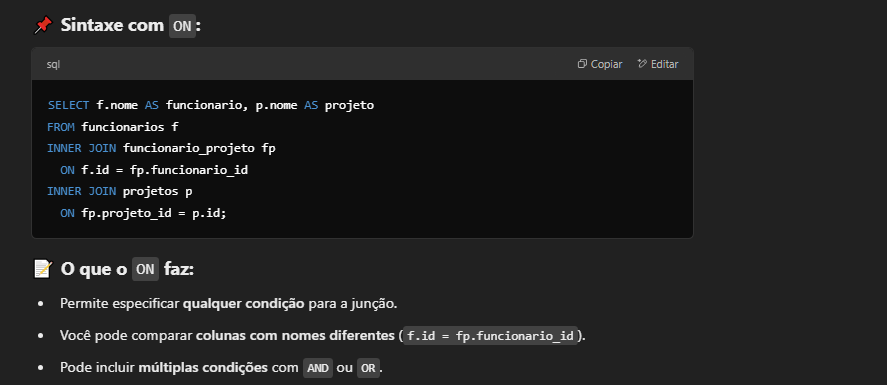


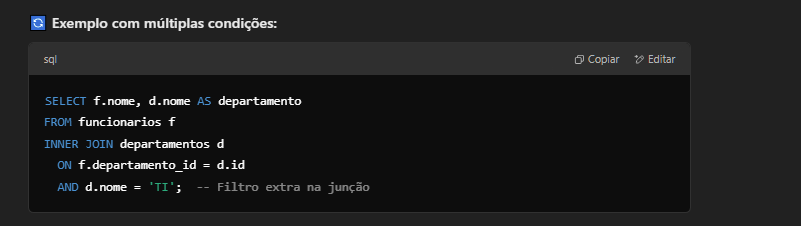


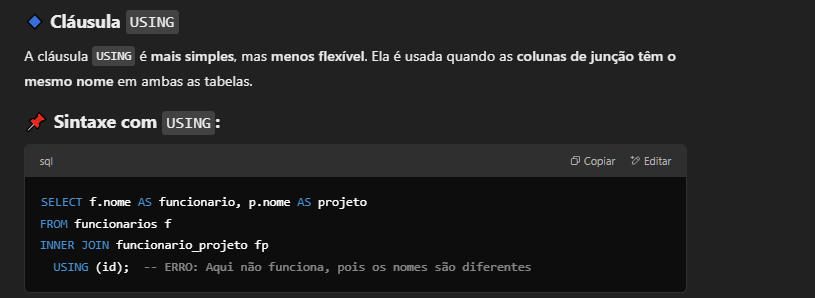


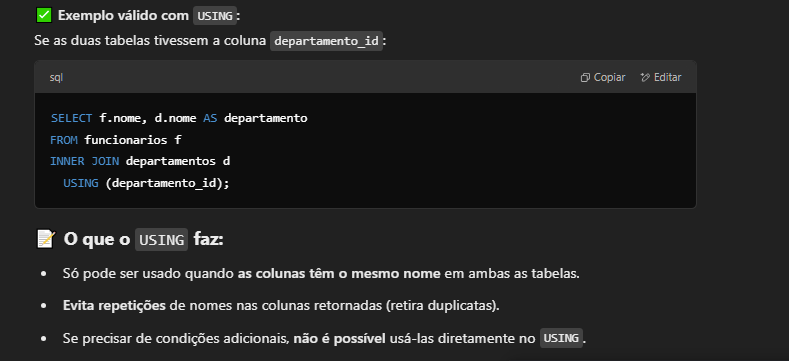
Pagina 44

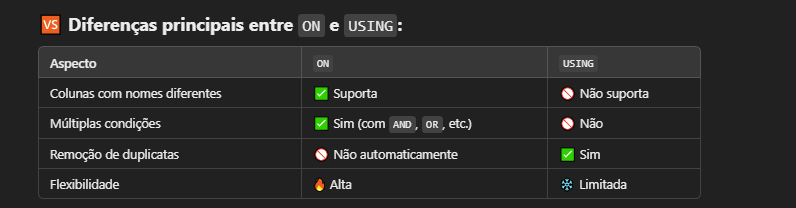


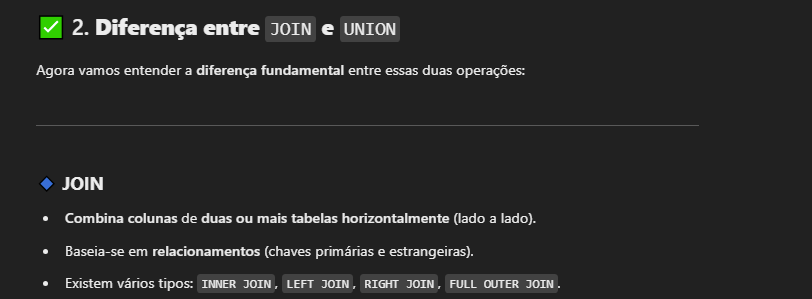


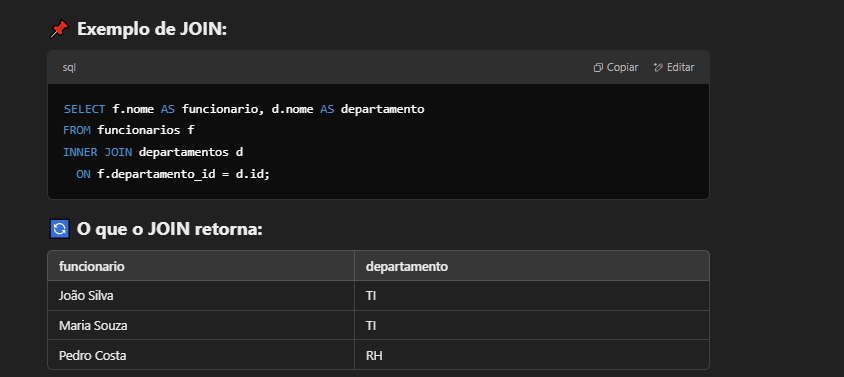


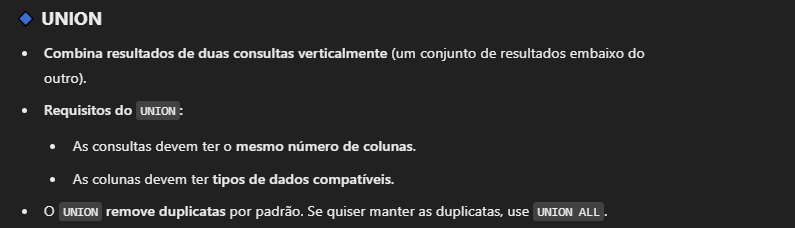


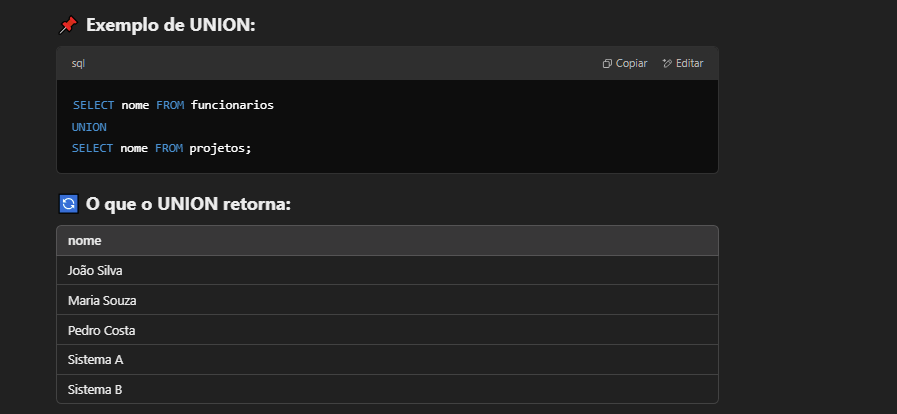


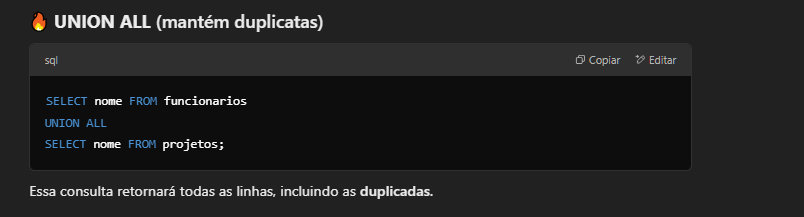


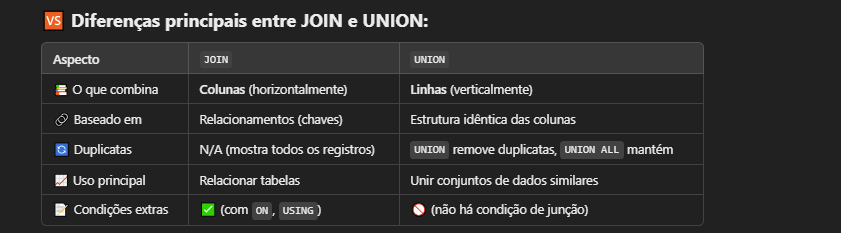


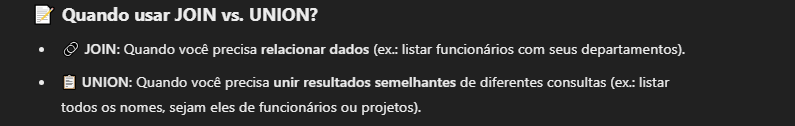


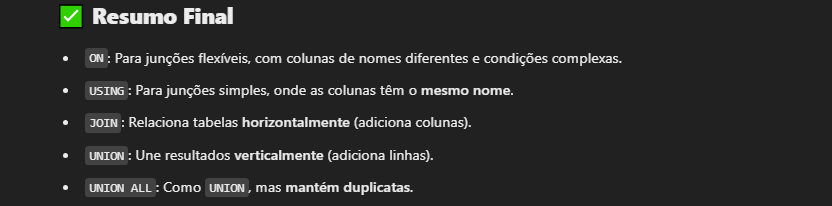












CREATE TABLE funcionarios (

id SERIAL PRIMARY KEY,

nome VARCHAR(50),

cargo VARCHAR(50),

salario DECIMAL(10,2),

departamento\_id INT

);

INSERT INTO funcionarios (nome, cargo, salario, departamento\_id) VALUES

('Ana', 'Analista', 3500, 1),

('Bruno', 'Gerente', 7000, 2),

('Carlos', 'Desenvolvedor', 4200, 1),

('Diego', 'Diretor', 10000, 3),

('Elisa', 'Analista', 3200, 1),

('Fernanda', 'Gerente', 7500, 2),

('Gabriel', 'Desenvolvedor', 4800, 1),

('Hugo', 'Analista', 3600, 3);

CREATE TABLE departamentos (

id SERIAL PRIMARY KEY,

nome VARCHAR(50),

localizacao VARCHAR(50)

);

INSERT INTO departamentos (nome, localizacao) VALUES

('TI', 'São Paulo'),

('RH', 'Rio de Janeiro'),

('Financeiro', 'São Paulo');

## ****PASSO 1: Criar a Tabela de Funcionários****

Copie e cole esse código no **pgAdmin** para criar a tabela e inserir alguns dados:

sql

CopiarEditar

CREATE TABLE funcionarios (

id SERIAL PRIMARY KEY,

nome VARCHAR(50),

cargo VARCHAR(50),

salario DECIMAL(10,2),

departamento\_id INT

);

INSERT INTO funcionarios (nome, cargo, salario, departamento\_id) VALUES

('Ana', 'Analista', 3500, 1),

('Bruno', 'Gerente', 7000, 2),

('Carlos', 'Desenvolvedor', 4200, 1),

('Diego', 'Diretor', 10000, 3),

('Elisa', 'Analista', 3200, 1),

('Fernanda', 'Gerente', 7500, 2),

('Gabriel', 'Desenvolvedor', 4800, 1),

('Hugo', 'Analista', 3600, 3);

CREATE TABLE departamentos (

id SERIAL PRIMARY KEY,

nome VARCHAR(50),

localizacao VARCHAR(50)

);

INSERT INTO departamentos (nome, localizacao) VALUES

('TI', 'São Paulo'),

('RH', 'Rio de Janeiro'),

('Financeiro', 'São Paulo');

🔹 **Criamos duas tabelas:**

* funcionarios: Armazena nome, cargo, salário e ID do departamento.
* departamentos: Armazena nome e localização do departamento.

🔹 **Inserimos dados para testar as consultas.**

## ****📌 PASSO 2: Subqueries para Praticar****

Agora, vamos treinar com exemplos de **todos os tipos de subqueries**, cada uma com explicação.

### ****1️⃣ Subquery Escalar (Retorna um único valor)****

➡️ **Objetivo**: Encontrar funcionários que ganham mais do que a média salarial da empresa.

sql

CopiarEditar

SELECT nome, salarioFROM funcionariosWHERE salario > (SELECT AVG(salario) FROM funcionarios);

🔹 **Lógica**:

1. A subquery (SELECT AVG(salario) FROM funcionarios) retorna a **média salarial**.
2. A consulta principal **compara os salários** com essa média.

✅ **Treine modificando a subquery** para calcular a média salarial **de um único departamento**.

### ****2️⃣ Subquery de Linha Única (Retorna uma linha com várias colunas)****

➡️ **Objetivo**: Encontrar o nome e o cargo do funcionário com o maior salário.

sql

CopiarEditar

SELECT nome, cargoFROM funcionariosWHERE salario = (SELECT MAX(salario) FROM funcionarios);

🔹 **Lógica**:

1. A subquery (SELECT MAX(salario) FROM funcionarios) retorna o **maior salário da empresa**.
2. A consulta principal **busca o funcionário que ganha esse valor**.

✅ **Teste buscar o funcionário com o menor salário**.

### ****3️⃣ Subquery de Múltiplas Linhas (Retorna várias linhas)****

➡️ **Objetivo**: Encontrar funcionários que trabalham em departamentos localizados em **São Paulo**.

sql

CopiarEditar

SELECT nome, cargoFROM funcionariosWHERE departamento\_id IN (SELECT id FROM departamentos WHERE localizacao = 'São Paulo');

🔹 **Lógica**:

1. A subquery (SELECT id FROM departamentos WHERE localizacao = 'São Paulo') retorna **todos os IDs dos departamentos em São Paulo**.
2. A consulta principal **busca os funcionários que pertencem a esses departamentos**.

✅ **Teste buscar funcionários de outro local**.

### ****4️⃣ Subquery Correlacionada (Executada para cada linha da consulta principal)****

➡️ **Objetivo**: Encontrar funcionários que ganham acima da média **do próprio departamento**.

sql

CopiarEditar

SELECT nome, salario, departamento\_idFROM funcionarios f1WHERE salario > (

SELECT AVG(salario)

FROM funcionarios f2

WHERE f1.departamento\_id = f2.departamento\_id

);

🔹 **Lógica**:

1. Para **cada funcionário**, a subquery calcula a **média do departamento dele**.
2. A consulta principal **filtra aqueles que ganham mais do que essa média**.

✅ **Modifique para encontrar os funcionários que ganham abaixo da média do departamento**.

### ****5️⃣ Subquery com EXISTS (Verifica se um valor existe em outra tabela)****

➡️ **Objetivo**: Encontrar departamentos que possuem pelo menos um funcionário.

sql

CopiarEditar

SELECT nome FROM departamentos dWHERE EXISTS (

SELECT 1 FROM funcionarios f WHERE f.departamento\_id = d.id

);

🔹 **Lógica**:

1. A subquery retorna **TRUE ou FALSE** para cada departamento.
2. Se existir pelo menos um funcionário no departamento, o departamento é exibido.

✅ **Teste modificar para encontrar departamentos vazios (sem funcionários)**.

### ****6️⃣ Subquery em SELECT (Usada como uma coluna na consulta principal)****

➡️ **Objetivo**: Mostrar cada funcionário junto com o nome do departamento.

sql

CopiarEditar

SELECT nome,

cargo,

salario,

(SELECT nome FROM departamentos WHERE id = funcionarios.departamento\_id) AS departamentoFROM funcionarios;

🔹 **Lógica**:

1. A subquery retorna **o nome do departamento** para cada funcionário.
2. O resultado é exibido como uma **nova coluna** na consulta principal.

✅ **Teste adicionar a localização do departamento junto com o nome**.

### ****7️⃣ Subquery com UPDATE (Alterando dados com base em outra tabela)****

➡️ **Objetivo**: Aumentar o salário dos funcionários que estão abaixo da média salarial da empresa.

sql

CopiarEditar

UPDATE funcionariosSET salario = salario \* 1.10WHERE salario < (SELECT AVG(salario) FROM funcionarios);

🔹 **Lógica**:

1. A subquery calcula a **média salarial**.
2. O UPDATE **aumenta o salário** de quem está abaixo dessa média.

✅ **Teste aplicar um aumento diferente para funcionários de um departamento específico**.

### ****8️⃣ Subquery com DELETE (Removendo dados com base em outra tabela)****

➡️ **Objetivo**: Remover funcionários que pertencem a departamentos localizados no **Rio de Janeiro**.

sql

CopiarEditar

DELETE FROM funcionariosWHERE departamento\_id IN (

SELECT id FROM departamentos WHERE localizacao = 'Rio de Janeiro'

);

🔹 **Lógica**:

1. A subquery retorna os **IDs dos departamentos no Rio**.
2. O DELETE **remove os funcionários desses departamentos**.

✅ **Teste remover funcionários de um departamento específico**.

## ****📌 PASSO 3: Próximos Passos****

Agora que você tem uma base sólida:  
✅ **Rode esses exemplos no pgAdmin** e veja os resultados.  
✅ **Altere os códigos** para entender melhor o comportamento das subqueries.  
✅ **Me envie dúvidas ou novos desafios para resolvermos juntos!** 🚀

Me avise como está indo o treino! 😃