

# Dominando SQL

---



Principais Comandos para o Uso Eficiente

Isabele Silvestre

# SUMÁRIO

---

I.	INTRODUÇÃO .....	3
II.	SELECT .....	5
III.	WHERE .....	8
IV.	INSERT INTO .....	11
V.	UPDATE .....	14
VI.	DELETE .....	17
VII.	ORDER BY .....	20
VIII.	GROUP BY .....	23
IX.	LIMIT .....	28
X.	DISTINCT .....	32
XI.	FUNÇÕES DE AGREGAÇÃO .....	35
XII.	CONCLUSÃO .....	41

# I

## INTRODUÇÃO

---

# Dominando SQL

Esteja preparado para mergulhar no vasto universo dos bancos de dados e do SQL!

SQL ou Structured Query Language é uma linguagem de programação usada para gerenciar bancos de dados relacionais.

Imagine um mundo onde a informação é a chave para tudo. Do controle de inventário em uma loja online à gestão de registros de pacientes em um hospital, o SQL desempenha um papel vital na organização e manipulação de dados em uma variedade de cenários.

Seja você um iniciante na jornada do desenvolvimento de software ou um veterano experiente em busca de aprimoramento, este ebook é a sua porta de entrada para o mundo complexo e fascinante do SQL.

Ao longo das próximas páginas, vamos explorar os conceitos fundamentais do SQL e desvendar os segredos por trás da gestão eficaz de bancos de dados, com o MySQL como nossa bússola neste universo digital.



# III

## SELECT

---

Veremos como usar o comando **SELECT** para obter informações específicas de nossos bancos de dados, com exemplos práticos.



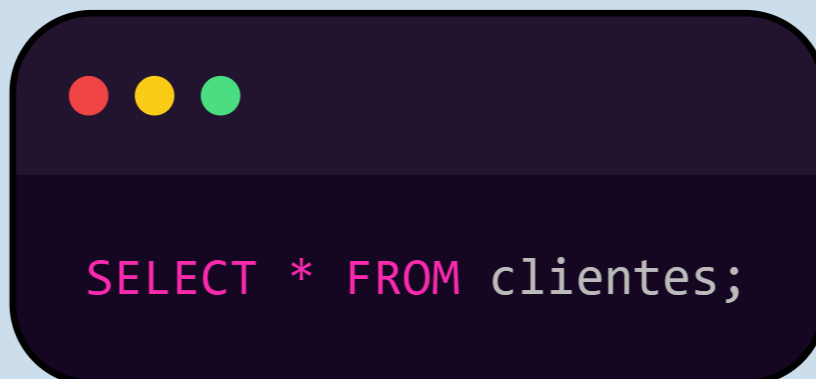
# SELECT

O comando SELECT é fundamental no SQL e é usado para recuperar dados de uma ou mais tabelas em um banco de dados.

## SELECT com asteriscos

Quando usamos SELECT \*, estamos dizendo ao SQL para retornar todas as colunas da tabela especificada. Este comando é útil quando queremos visualizar todos os dados de uma tabela sem precisar listar cada coluna manualmente.

Exemplo:



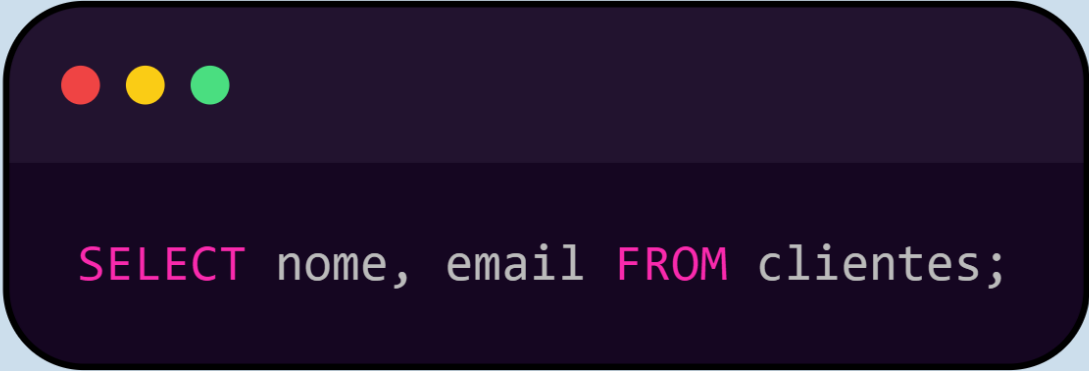
Este comando retorna todos os registros da tabela clientes.



# SELECT sem asteriscos

Quando especificamos as colunas que queremos selecionar, somos mais precisos sobre os dados que desejamos recuperar. Este método é mais eficiente e claro, especialmente em tabelas com muitas colunas.

Exemplo:

A dark-themed terminal window with three colored window control buttons (red, yellow, green) in the top-left corner. The text `SELECT nome, email FROM clientes;` is displayed in a light blue monospace font.

```
SELECT nome, email FROM clientes;
```

Este comando retorna apenas as colunas nome e e-mail dos registros na tabela clientes.



# III

## WHERE

---

Veremos como usar o comando **WHERE** para obter informações específicas de nossos bancos de dados, com exemplos práticos.



# WHERE

O comando WHERE é utilizado para filtrar os resultados retornados por uma consulta com base em condições específicas.

Exemplo:



```
SELECT * FROM produtos  
WHERE PRECO > 50;
```

Este comando retorna todos os produtos cujo preço é superior a 50.


O WHERE é crucial para selecionar apenas os dados que atendem a critérios específicos, tornando as consultas mais precisas e eficientes.



# Combinações com AND e OR

Podemos combinar várias condições usando AND e OR para criar filtros mais complexos.


Exemplo com AND:



```
SELECT * FROM clientes  
WHERE cidade = 'Sao Paulo'  
AND idade > 30;
```

Este comando retorna todos os clientes de São Paulo com mais de 30 anos.

Exemplo com OR:



```
SELECT * FROM clientes  
WHERE preco < 20  
OR estoque > 100;
```

Este comando retorna produtos que custam menos de 20 ou têm mais de 100 unidades em estoque.



# IV

## INSERT INTO

---

Veremos como usar o comando **INSERT INTO** para adicionar dados a um banco de dados, com exemplos práticos.


# INSERT INTO

O comando INSERT INTO é utilizado para adicionar novos registros a uma tabela no banco de dados. Ele é essencial para alimentar seu banco de dados com informações.

## Inserindo um único registro

Imagine que temos uma tabela chamada clientes com as seguintes colunas: id, nome, e-mail, idade.

Exemplo:

A terminal window with a dark purple background and rounded corners. At the top left, there are three colored circles: red, yellow, and green. The text inside the terminal is written in a light purple/pink monospace font.

```
INSERT INTO clientes (nome, email, idade)
VALUES ('Maria', 'maria@gmail.com', '28');
```

Clientes: Nome da tabela onde o registro será inserido.

(nome, e-mail, idade): As colunas específicas nas quais os valores serão inseridos.


VALUES ('Maria', 'maria@email.com', 28'): Os valores correspondentes a serem inseridos nas colunas.



## Inserindo múltiplos registros

Também podemos inserir vários registros de uma vez, o que é muito eficiente para adicionar grandes quantidades de dados.

Exemplo:

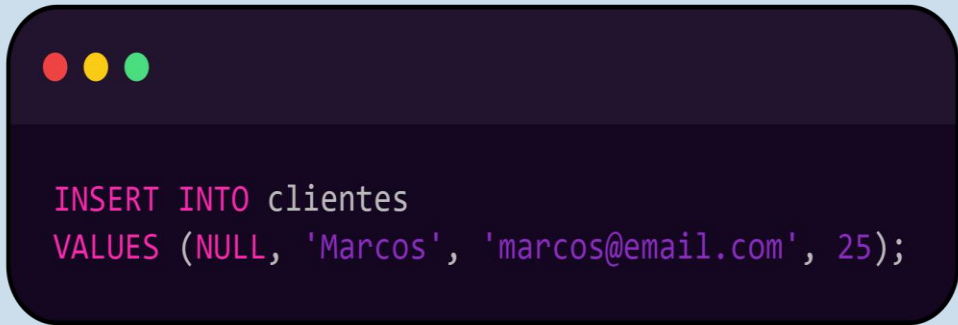


```
INSERT INTO clientes (nome, email, idade)
VALUES
('João', 'joao@email.com', 34),
('Ana', 'ana@email.com', 22),
('Carlos', 'carlos@email.com', 40);
```

## Inserindo dados em todas as colunas

Se quisermos inserir dados em todas as colunas da tabela e a tabela tem valores padrão definidos para colunas como id (chave primária autoincrementada), podemos omitir a lista de colunas.

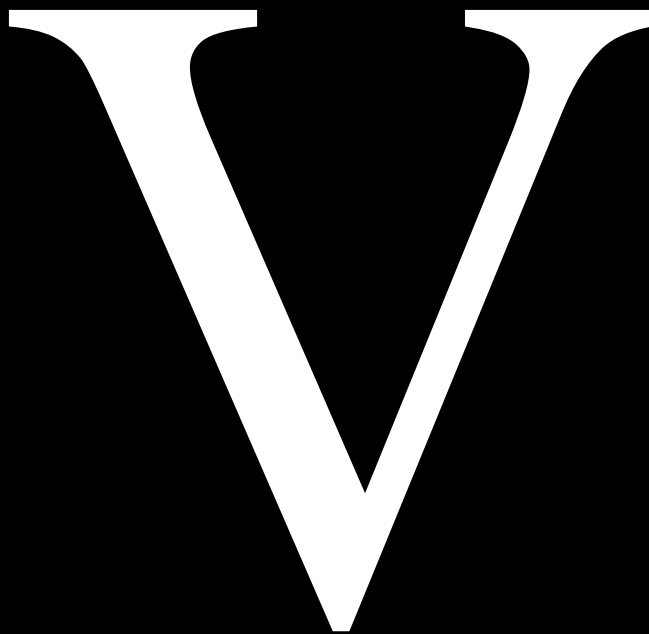
Exemplo:



```
INSERT INTO clientes
VALUES (NULL, 'Marcos', 'marcos@email.com', 25);
```

Neste exemplo, NULL é usado para o campo id porque ele será gerado automaticamente pelo banco de dados (se estiver configurado como autoincremento).





## UPDATE

---

Veremos como usar o comando **UPDATE** para fazer alterações específicas nos registros já existentes de forma simples e prática com exemplos.

# UPDATE

O comando UPDATE é usado para modificar os dados existentes em uma tabela no banco de dados. Ele nos permite fazer alterações específicas nos registros já existentes.

## Atualizando um único registro

Imagine que temos uma tabela chamada produtos com as seguintes colunas: id, nome, preço, estoque.

Suponha que queiramos atualizar o preço de um produto com o ID 1 para 60.

Exemplo:



```
UPDATE produtos  
SET preco = 60  
WHERE id = 1;
```

Produtos: Nome da tabela que será atualizada.

SET preco = 60: Define o novo valor da coluna preço.

WHERE id = 1: Especifica a condição para identificar o registro a ser atualizado, neste caso, onde o id é igual a 1.



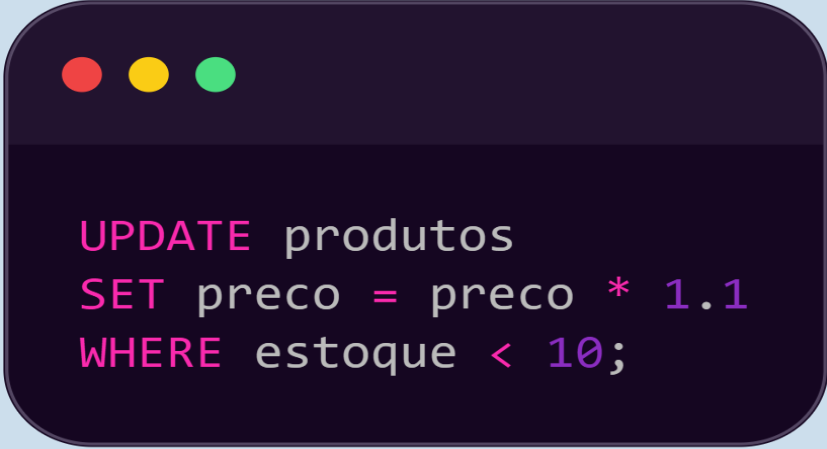


# Atualizando múltiplos registros

Também podemos atualizar vários registros de uma vez, desde que eles atendam à mesma condição.

Suponha que queiramos aumentar o preço de todos os produtos com estoque menor que 10 em 10%.

Exemplo:



```
UPDATE produtos
SET preco = preco * 1.1
WHERE estoque < 10;
```

UPDATE produtos : Indica que queremos atualizar os registros na tabela produtos.

SET preco = preco \* 1.1: Define o que será atualizado. Neste caso, estamos multiplicando o valor da coluna preço por 1.1, o que aumenta o preço em 10%.

WHERE estoque < 10: Especifica quais registros serão atualizados. Neste caso, apenas os produtos com um estoque inferior a 10 serão afetados pela atualização de preço.



# VI

## DELETE

---

Veremos como usar o comando **DELETE** para excluir registros de uma consulta de acordo com uma ou mais condições específicas, com exemplos.

# DELETE

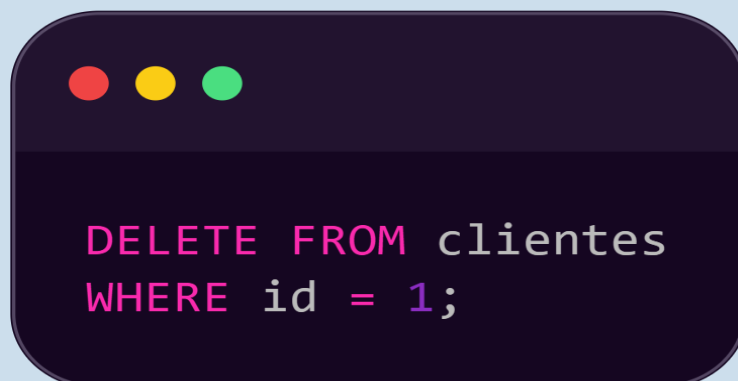
O comando DELETE é uma ferramenta essencial para a administração de banco de dados, permitindo a remoção de dados que não são mais necessários. No entanto, é importante usar com cuidado e cautela, garantindo que apenas os registros desejados sejam excluídos.

## Excluindo um único registro

Imagine que temos uma tabela chamada clientes com as seguintes colunas: id, nome, e-mail, idade.

Suponha que queiramos excluir um cliente com o ID 1.

Exemplo:

A terminal window with a dark purple background and rounded corners. At the top left, there are three colored circles: red, yellow, and green. The text inside the terminal is written in a light blue monospace font and shows a SQL command to delete a record from a table named 'clientes' where the 'id' is 1.

```
DELETE FROM clientes  
WHERE id = 1;
```

Clientes: Nome da tabela da qual os registros serão excluídos.

WHERE id = 1: Especifica a condição para identificar o registro a ser excluído, neste caso, onde o id é igual a 1.

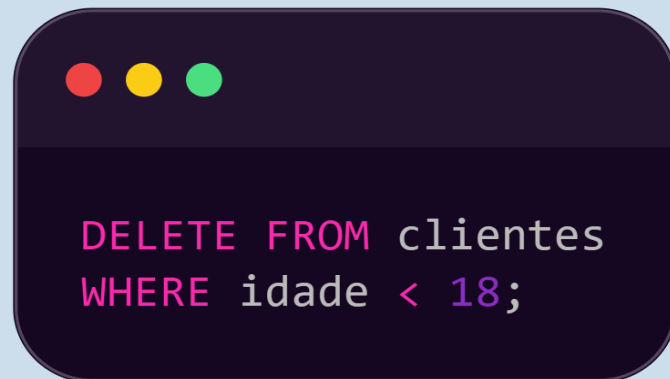


# Excluindo múltiplos registros

Também podemos excluir vários registros de uma vez, desde que eles atendam à mesma condição.

Suponha que queiramos excluir todos os clientes com idade inferior a 18 anos.

Exemplo:



```
DELETE FROM clientes
WHERE idade < 18;
```

DELETE FROM clientes: Indica que queremos excluir registros da tabela clientes.

WHERE idade < 18: Especifica a condição que os registros devem atender para serem excluídos. Neste caso, estamos excluindo clientes cuja idade é inferior a 18 anos.



# VIII

## ORDER BY

---

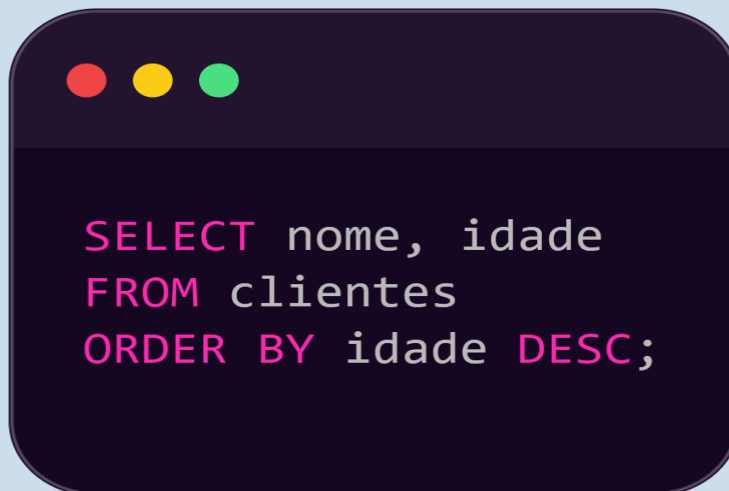
Veremos como usar o comando **ORDER BY** para ordenar os resultados de uma consulta de acordo com uma ou mais colunas específicas, com exemplos simples.

# ORDER BY

O comando ORDER BY permite ordenar os resultados da consulta de acordo com uma ou mais colunas, especificando a direção da ordenação (ASC para ascendente e DESC para descendente).

## Ordem Decrescente

Exemplo:



```
SELECT nome, idade
FROM clientes
ORDER BY idade DESC;
```

SELECT nome, idade: Indica que queremos selecionar as colunas nome e idade da tabela clientes.

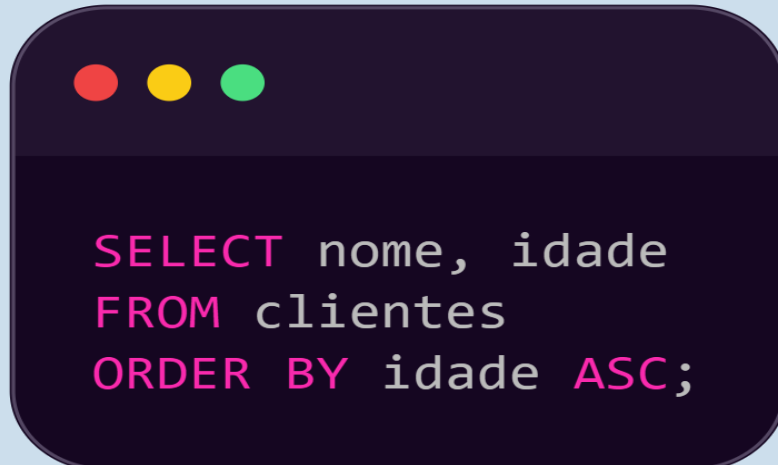
FROM clientes: Especifica a tabela da qual queremos selecionar os dados, neste caso, a tabela clientes.

ORDER BY idade DESC: Esta cláusula ordena os resultados da consulta com base na coluna idade de forma descendente (do maior para o menor). Isso significa que os clientes mais velhos aparecerão primeiro na lista de resultados.



# Ordem Crescente

Exemplo:



```
SELECT nome, idade  
FROM clientes  
ORDER BY idade ASC;
```

**SELECT nome, idade:** Indica que queremos selecionar as colunas nome e idade da tabela clientes.

**FROM clientes:** Especifica a tabela da qual queremos selecionar os dados, neste caso, a tabela clientes.

**ORDER BY idade ASC:** Esta cláusula ordena os resultados da consulta com base na coluna idade de forma ascendente (do menor para o maior). Isso significa que os clientes mais jovens aparecerão primeiro na lista de resultados.





# VIII

## GROUP BY

---

Veremos como usar a cláusula **GROUP BY** para agrupar os resultados de uma consulta de acordo com uma ou mais colunas específicas, com exemplos.

# GROUP BY

O comando GROUP BY é usado em consultas SQL para agrupar linhas de dados que possuem valores semelhantes em uma ou mais colunas e, em seguida, aplicar funções de agregação a esses grupos.

Aqui está um exemplo de como utilizá-lo:

Suponha que temos uma tabela chamada pedidos com as colunas cliente e valor, que armazena informações sobre os pedidos de clientes.

Cliente	Valor
Ana	100
João	150
Ana	200
Maria	120
João	80

Agora, vamos usar o GROUP BY para calcular o valor total gasto por cada cliente:

```
SELECT cliente, SUM(valor) AS total_gasto
FROM pedidos
GROUP BY cliente;
```



Neste exemplo:

A cláusula SELECT seleciona a coluna cliente e calcula a soma dos valores usando a função de agregação SUM(valor), renomeando o resultado como total gasto.

A cláusula FROM especifica a tabela pedidos de onde os dados serão selecionados.

A cláusula GROUP BY agrupa os registros pela coluna cliente.

Como resultado, obtemos:

Cliente	Total_gasto
Ana	300
João	230
Maria	120


Isso significa que Ana gastou um total de 300, João gastou 230 e Maria gastou 120.



# Filtragem de dados com HAVING

Podemos usar a cláusula HAVING em conjunto com o GROUP BY para filtrar grupos com base em condições específicas.

Exemplo:

A terminal window with a dark purple background and rounded corners. It has three colored window control buttons (red, yellow, green) in the top-left corner. The text inside is a SQL query in a light pink/magenta monospace font.

```
SELECT departamento, AVG(salario) AS media_salario
FROM funcionarios
GROUP BY departamento
HAVING AVG(salario) > 5000;
```

SELECT departamento, AVG(salario) AS media\_salario: Esta parte da consulta seleciona o nome do departamento da tabela "funcionarios" e calcula a média dos salários dos funcionários nesse departamento usando a função de agregação AVG(). O resultado dessa média é renomeado como "media\_salario" usando o alias AS.

FROM funcionarios: Esta parte especifica a tabela da qual queremos selecionar os dados, que é a tabela "funcionarios".

GROUP BY departamento: Esta parte agrupa os resultados pelo nome do departamento.

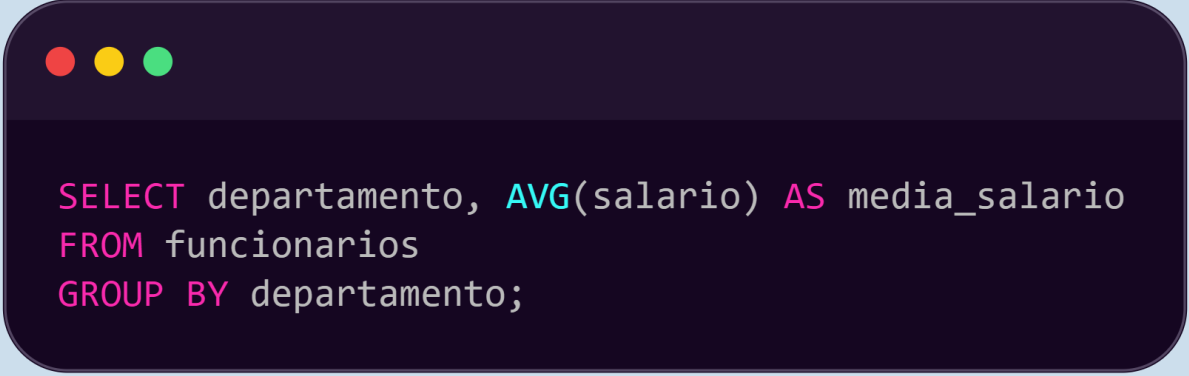
HAVING AVG(salario) > 5000: Esta parte filtra os resultados com base na média salarial calculada. Aqui, estamos filtrando os resultados para mostrar apenas os departamentos cuja média salarial seja superior a \$5000.



# Funções de Agregação

As funções de agregação mais comuns usadas com o GROUP BY incluem COUNT, SUM, AVG, MIN e MAX, entre outras. Essas funções são aplicadas a cada grupo formado pelo GROUP BY para calcular valores agregados.

Exemplo:

A terminal window with a dark purple background and rounded corners. It has three colored window control buttons (red, yellow, green) in the top-left corner. The text inside is a SQL query in a light pink font.

```
SELECT departamento, AVG(salario) AS media_salario
FROM funcionarios
GROUP BY departamento;
```

SELECT departamento, AVG(salario) AS media\_salario: Esta parte da consulta seleciona o nome do departamento da tabela "funcionarios" e calcula a média dos salários dos funcionários nesse departamento usando a função de agregação AVG(). O resultado dessa média é renomeado como "media\_salario" usando o alias AS.

FROM funcionarios: Esta parte especifica a tabela da qual queremos selecionar os dados, que é a tabela "funcionarios".

GROUP BY departamento: Esta parte agrupa os resultados pelo nome do departamento. Isso significa que o cálculo da média salarial será feito separadamente para cada departamento.

Este comando calcula a média salarial de cada departamento, agrupando os resultados pelo nome do departamento.



# IX

## LIMIT

---

Veremos como usar o comando **LIMIT** para ordenar os resultados de uma consulta de acordo com uma ou mais colunas específicas, com exemplos simples.

# LIMIT

O comando LIMIT é utilizado em consultas SQL para restringir o número de linhas retornadas pelos resultados da consulta. Isso é útil quando queremos visualizar apenas uma quantidade específica de resultados, especialmente em consultas que retornam grandes conjuntos de dados.

Segue um exemplo de como utiliza-lo:

Suponha que temos uma tabela chamada produtos com muitos registros e queremos visualizar apenas os 5 primeiros produtos:



```
SELECT *  
FROM produtos  
LIMIT 5;
```





Neste exemplo:

A cláusula `SELECT *` seleciona todas as colunas da tabela produtos.

A cláusula `FROM produtos` especifica a tabela da qual os dados serão selecionados.

A cláusula `LIMIT 5` restringe o número de linhas retornadas para 5.

Isso significa que apenas os 5 primeiros registros da tabela produtos serão retornados como resultado da consulta.

Agora, vamos considerar outro exemplo onde queremos visualizar os 10 produtos mais caros da tabela produtos, ordenados por preço em ordem decrescente:



```
SELECT *  
FROM produtos  
ORDER BY preco DESC  
LIMIT 10;
```



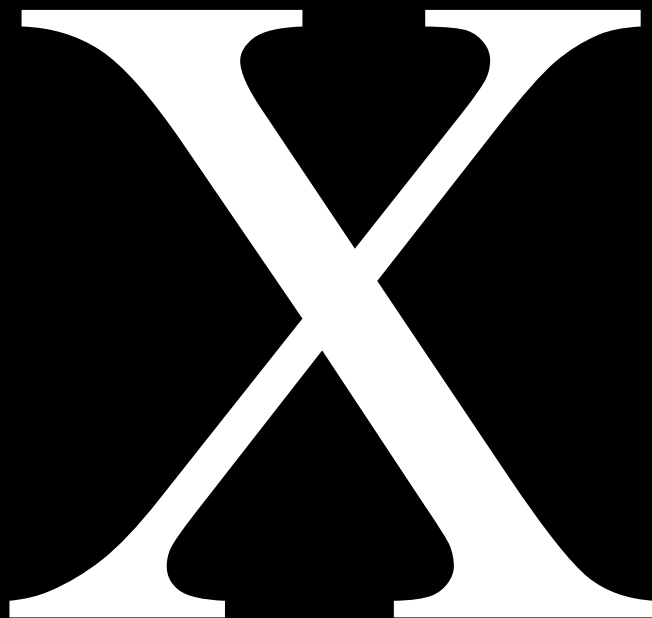
Neste exemplo:

A cláusula ORDER BY preço DESC ordena os resultados pela coluna preço em ordem decrescente.

A cláusula LIMIT 10 restringe o número de linhas retornadas para 10, o que nos permite visualizar apenas os 10 produtos mais caros.

Isso nos permite visualizar apenas os 10 produtos mais caros da tabela produtos, facilitando a análise de grandes conjuntos de dados.





## DISTINCT

---

Veremos como usar a cláusula **DISTINCT** para identificar e remover registros duplicados dos resultados de uma consulta de acordo com uma ou mais colunas específicas, com exemplos.

# DISTINCT

O comando DISTINCT é usado em consultas SQL para retornar apenas valores únicos em uma coluna ou conjunto de colunas. Isso significa que ele remove duplicatas dos resultados da consulta. Aqui está uma explicação simples:

Suponha que temos uma tabela chamada cores com a seguinte coluna:

Cor
Vermelho
Azul
Verde
Vermelho
Amarelo
Azul

Se quisermos ver apenas as cores únicas nesta tabela, usaríamos o comando DISTINCT da seguinte forma:

```
SELECT DISTINCT cor
FROM cores;
```



Isso retornaria:

Cor
Vermelho
Azul
Verde
Amarelo

O comando `DISTINCT` removeu as duplicatas, permitindo que vejam apenas as cores únicas na tabela.

Isso é útil para análise de dados quando queremos garantir que estamos lidando apenas com valores exclusivos em uma determinada coluna.



# XI

## FUNÇÕES DE AGREGAÇÃO

---

Veremos como utilizar as **FUNÇÕES DE AGREGAÇÃO** para calcular estatísticas resumidas, como média, soma, mínimo ou máximo, dos resultados de uma consulta de acordo com uma ou mais colunas específicas com exemplos simples, com exemplos simples.

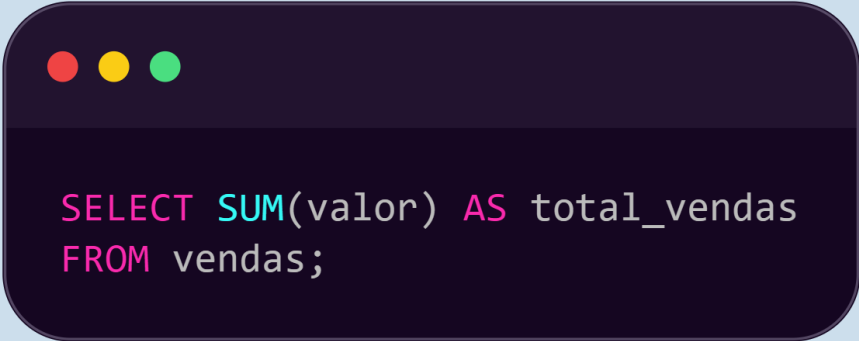
# Funções de Agregação

As funções de agregação são utilizadas para realizar cálculos em conjuntos de dados, como somar valores, contar o número de registros, calcular médias, encontrar valores mínimos e máximos, entre outros. Vou explicar as principais funções de agregação com exemplos:

## SUM()

Esta função retorna a soma dos valores de uma coluna.

Exemplo:



```
SELECT SUM(valor) AS total_vendas
FROM vendas;
```

`SELECT AVG(salario) AS media_salario`: Esta parte da instrução indica que queremos selecionar a média dos salários da coluna "salario" da tabela "funcionarios". A função `AVG()` é uma função de agregação que calcula a média dos valores em uma coluna. Estamos atribuindo um alias "media\_salario" ao resultado da média para facilitar a referência a ele posteriormente.

`FROM funcionarios`: Esta parte especifica a tabela da qual queremos selecionar os dados, que é a tabela "funcionarios".

Este comando calcula o total de vendas somando os valores da coluna "valor" na tabela "vendas".

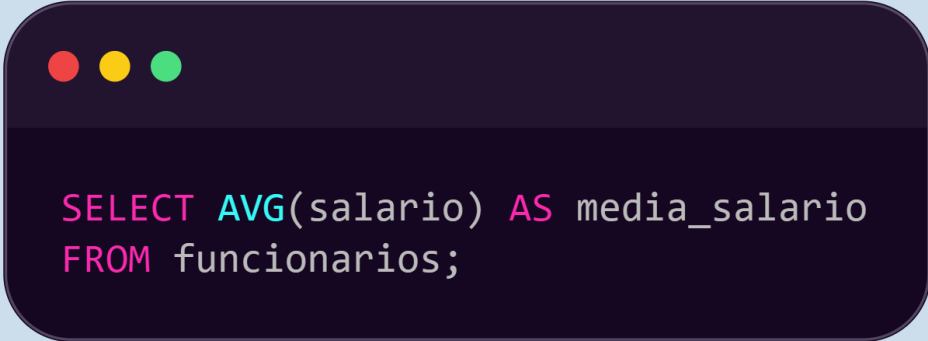




# AVG()

Esta função retorna a média dos valores de uma coluna.

Exemplo:

A terminal window with a dark purple background and rounded corners. It has three colored window control buttons (red, yellow, green) in the top-left corner. The text inside is a SQL query: 

```
SELECT AVG(salario) AS media_salario  
FROM funcionarios;
```

```
SELECT AVG(salario) AS media_salario  
FROM funcionarios;
```

SELECT AVG(salario) AS media\_salario: Esta parte da consulta seleciona a média dos salários da coluna "salario" da tabela "funcionarios". A função AVG() é uma função de agregação que calcula a média dos valores em uma coluna. O resultado da média é então renomeado como "media\_salario" usando o alias AS, facilitando a referência ao resultado.

FROM funcionarios: Esta parte especifica a tabela da qual queremos selecionar os dados, que é a tabela "funcionarios".


Este comando calcula a média salarial dos funcionários, utilizando os valores da coluna "salario" na tabela "funcionarios".



# COUNT()

Esta função retorna o número de registros em um conjunto de dados.

Exemplo:



```
SELECT COUNT(*) AS total_produtos  
FROM produtos;
```

**SELECT COUNT(\*) AS total\_produtos:** Esta parte da consulta utiliza a função de agregação COUNT() para contar o número de registros na tabela "produtos". O uso de "\*" como argumento da função COUNT() indica que queremos contar todos os registros. O resultado desse contador é então renomeado como "total\_produtos" usando o alias AS, facilitando a referência ao resultado.

**FROM produtos:** Esta parte especifica a tabela da qual queremos contar os registros, que é a tabela "produtos".

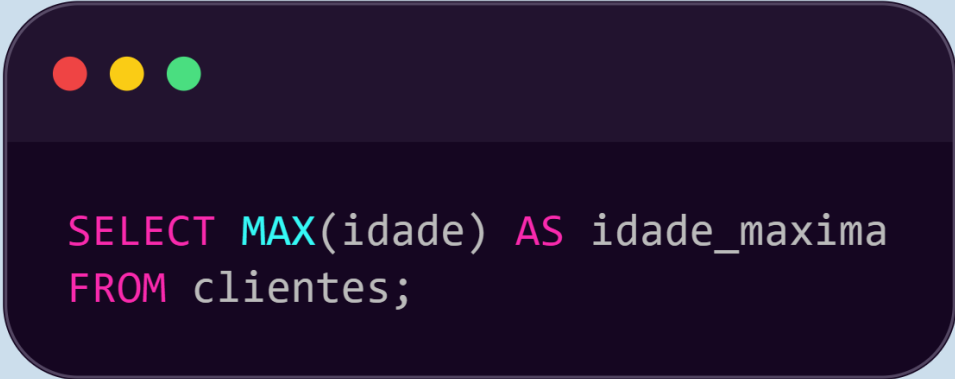
Este comando conta o número total de produtos na tabela "produtos".



# MAX()

Esta função retorna o valor máximo em uma coluna.

Exemplo:

A terminal window with a dark purple background and three colored window control buttons (red, yellow, green) in the top left corner. It displays a SQL query in a monospaced font with syntax highlighting: 'SELECT' in pink, 'MAX(idade)' in cyan, 'AS idade\_maxima' in pink, and 'FROM clientes;' in pink.

```
SELECT MAX(idade) AS idade_maxima
FROM clientes;
```

SELECT MAX(idade) AS idade\_maxima: Esta parte da consulta seleciona o valor máximo da coluna "idade" da tabela "clientes". A função MAX() é uma função de agregação que encontra o maior valor em um conjunto de dados. O resultado desse valor máximo é então renomeado como "idade\_maxima" usando o alias AS, para facilitar a referência ao resultado.

FROM clientes: Esta parte especifica a tabela da qual queremos selecionar os dados, que é a tabela "clientes".

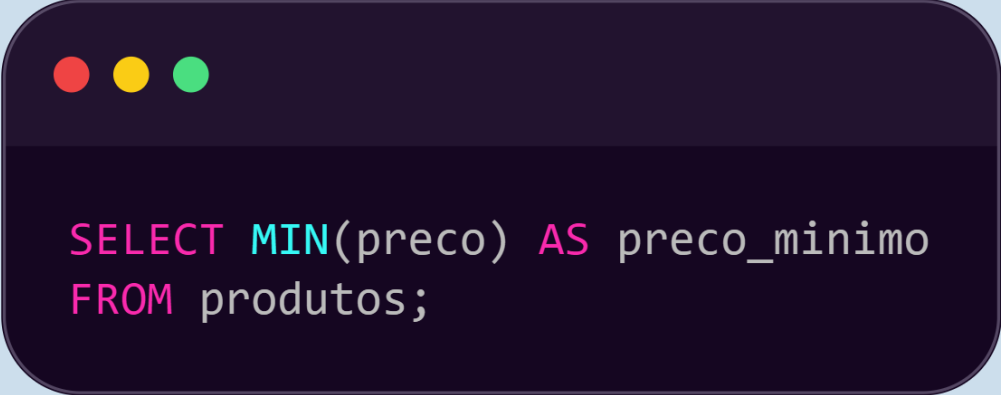
Este comando encontra a idade máxima dos clientes, utilizando os valores da coluna "idade" na tabela "clientes".



# MIN()

Esta função retorna o valor mínimo em uma coluna.

Exemplo:

A terminal window with a dark purple background and rounded corners. It has three colored window control buttons (red, yellow, green) in the top-left corner. The text inside is a SQL query: 

```
SELECT MIN(preco) AS preco_minimo  
FROM produtos;
```

```
SELECT MIN(preco) AS preco_minimo  
FROM produtos;
```

SELECT MIN(preco) AS preco\_minimo: Esta parte da consulta seleciona o valor mínimo da coluna "preco" da tabela "produtos". A função MIN() é uma função de agregação que encontra o menor valor em um conjunto de dados. O resultado desse valor mínimo é então renomeado como "preco\_minimo" usando o alias AS, facilitando a referência ao resultado.

FROM produtos: Esta parte especifica a tabela da qual queremos selecionar os dados, que é a tabela "produtos".

Este comando encontra o preço mínimo dos produtos, utilizando os valores da coluna "preco" na tabela "produtos".



# XIII

## CONCLUSÃO

---

Obrigada por dedicar seu tempo à leitura deste conteúdo!

Este ebook foi gerado por um sistema de inteligência artificial e diagramado por um humano.

É importante ressaltar que, devido à natureza automatizada da geração, podem existir possíveis imprecisões ou erros.

Este material foi criado com propósitos estritamente didáticos, visando fornecer informações úteis sobre o assunto em questão.

Espero que tenha encontrado o conteúdo útil e esclarecedor. Se houver alguma dúvida ou sugestão, não hesite em contatar.



[GitHub](https://github.com)

