GUIA DE COMANDOS SQL

MATERIAL DE APOIO

Sumário

Introdução ao SQL e sua Importância

O SQL (Structured Query Language) é uma linguagem de programação essencial para gerenciar e manipular bancos de dados relacionais. Dominar o SQL é uma habilidade altamente valorizada no mercado de trabalho, especialmente em áreas como análise de dados, ciência de dados e desenvolvimento de software. Segundo levantamentos recentes, o SQL está entre as três habilidades mais demandadas em vagas de tecnologia, com mais de 150 mil oportunidades abertas anualmente apenas no Brasil. Além disso, bancos de dados como MySQL, PostgreSQL e SQL Server são amplamente utilizados em empresas de todos os portes, desde startups até grandes corporações. Neste material, vamos explorar os 50 comandos mais utilizados em SQL, com exemplos práticos baseados em um cenário de uma rede de supermercados, onde você aprenderá a consultar, filtrar, ordenar e manipular dados de forma eficiente.

SELECT

O comando **SELECT** é o mais básico e fundamental em SQL. Ele é usado para recuperar dados de uma ou mais tabelas em um banco de dados. No contexto de uma rede de supermercados, você pode usá-lo para listar produtos, clientes, vendas ou qualquer outra informação armazenada.

Exemplo de uso: Suponha que você queira listar todos os produtos cadastrados no banco de dados.

```
-- Seleciona todas as colunas da tabela "produtos"

SELECT * FROM produtos;

-- Seleciona as colunas preco e produto da tabela "produtos"

SELECT preco,produto FROM produtos;
```

Explicação do código:

- SELECT *: Seleciona todas as colunas.
- SELECT preco,produto: Seleciona somente as colunas preco e produto.
- FROM produtos: Especifica a tabela de onde os dados serão retirados.

WHERE

O comando **WHERE** é usado para filtrar registros com base em uma condição específica. Ele é essencial para buscar dados que atendam a critérios definidos.

Exemplo de uso: Queremos listar apenas os produtos com preço superior a R$ 10,00.

```
-- Seleciona produtos com preço maior que 10
```

```
SELECT * FROM produtos
WHERE preco > 10;
```

Explicação do código:

- WHERE preco > 10: Filtra os registros onde o valor da coluna "preco" é maior que 10.

ORDER BY

O comando **ORDER BY** é usado para ordenar os resultados de uma consulta com base em uma ou mais colunas, em ordem crescente (ASC) ou decrescente (DESC).

Exemplo de uso: Queremos listar os produtos ordenados pelo preço, do mais barato para o mais caro.

```
-- Ordena os produtos pelo preço em ordem crescente
SELECT * FROM produtos
ORDER BY preco ASC;
```

Explicação do código:

- ORDER BY preco ASC: Ordena os resultados pela coluna "preco" em ordem ascendente.

INSERT

O comando **INSERT INTO** é usado para adicionar novos registros a uma tabela.

Exemplo de uso: Vamos adicionar um novo produto ao banco de dados.

```
-- Insere um novo produto na tabela "produtos"
INSERT INTO produtos (nome, preco, categoria, estoque)
VALUES ('Arroz 5kg', 25.90, 'Alimentos', 100);
```

Explicação do código:

- INSERT INTO produtos: Especifica a tabela onde o registro será inserido.
- (nome, preco, categoria, estoque): Define as colunas que receberão valores.
- VALUES: Insere os valores correspondentes.

UPDATE

O comando **UPDATE** é usado para modificar registros existentes em uma tabela.

Exemplo de uso: Vamos atualizar o preço de um produto específico.

```
-- Atualiza o preço do produto com ID 1
UPDATE produtos
SET preco = 30.00
WHERE id = 1;
```

Explicação do código:

- UPDATE produtos: Especifica a tabela a ser atualizada.
- SET preco = 30.00: Define o novo valor para a coluna "preco".
- WHERE id = 1: Filtra o registro que será atualizado.

DELETE

O comando **DELETE** é usado para remover registros de uma tabela.

Exemplo de uso: Vamos excluir um produto que não está mais disponível.

```
-- Remove o produto com ID 10
DELETE FROM produtos
WHERE id = 10;
```

Explicação do código:

- DELETE FROM produtos: Especifica a tabela de onde o registro será removido.
- WHERE id = 10: Filtra o registro a ser excluído.

JOIN

O comando **JOIN** é usado para combinar dados de duas ou mais tabelas com base em uma condição relacionada.

Exemplo de uso: Queremos listar os produtos vendidos com detalhes do fornecedor.

```
-- Combina dados das tabelas "produtos" e "fornecedores"
SELECT produtos.nome, fornecedores.nome AS fornecedor
FROM produtos
```

JOIN fornecedores ON produtos.fornecedor_id = fornecedores.id;

Explicação do código:

- JOIN fornecedores: Combina a tabela "produtos" com a tabela "fornecedores".
- ON produtos.fornecedor_id = fornecedores.id: Define a condição de relacionamento.

GROUP BY

O comando **GROUP BY** é usado para agrupar registros com base em valores de uma ou mais colunas, geralmente em conjunto com funções de agregação como COUNT, SUM, AVG, etc.

Exemplo de uso: Queremos saber quantos produtos existem em cada categoria.

-- Agrupa produtos por categoria e conta quantos existem

SELECT categoria, COUNT(*) AS total_produtos

FROM produtos

GROUP BY categoria;

Explicação do código:

- GROUP BY categoria: Agrupa os registros pela coluna "categoria".
- COUNT(*) AS total_produtos: Conta o número de produtos em cada grupo.

HAVING

O comando **HAVING** é usado para filtrar grupos de registros após o uso do **GROUP BY**.

Exemplo de uso: Queremos listar apenas as categorias com mais de 10 produtos.

-- Filtra categorias com mais de 10 produtos

SELECT categoria, COUNT(*) AS total_produtos

FROM produtos

GROUP BY categoria

HAVING COUNT(*) > 10;

Explicação do código:

- HAVING COUNT(*) > 10: Filtra os grupos com mais de 10 produtos.

COUNT

A função **COUNT** é usada para contar o número de registros que atendem a uma condição.

Exemplo de uso: Queremos saber quantos produtos estão cadastrados no banco de dados.

-- Conta o número total de produtos

SELECT COUNT(*) AS total_produtos

FROM produtos;

Explicação do código:

- COUNT(*): Conta todos os registros da tabela.

SUM

A função **SUM** é usada para calcular a soma dos valores de uma coluna numérica.

Exemplo de uso: Queremos calcular o valor total em estoque de todos os produtos.

-- Soma o valor total do estoque

SELECT SUM(estoque * preco) AS valor_total_estoque

FROM produtos;

Explicação do código:

- SUM(estoque * preco): Multiplica o estoque pelo preço de cada produto e soma os resultados.

AVG

A função **AVG** é usada para calcular a média dos valores de uma coluna numérica.

Exemplo de uso: Queremos saber o preço médio dos produtos.

-- Calcula o preço médio dos produtos

SELECT AVG(preco) AS preco_medio

FROM produtos;

Explicação do código:

- AVG(preco): Calcula a média dos valores na coluna "preco".

MIN

A função **MIN** é usada para encontrar o menor valor em uma coluna.

Exemplo de uso: Queremos saber o preço do produto mais barato.

```
-- Encontra o preço mínimo
SELECT MIN(preco) AS preco_minimo
FROM produtos;
```

Explicação do código:

- MIN(preco): Retorna o menor valor na coluna "preco".

MAX

A função **MAX** é usada para encontrar o maior valor em uma coluna.

Exemplo de uso: Queremos saber o preço do produto mais caro.

```
-- Encontra o preço máximo
SELECT MAX(preco) AS preco_maximo
FROM produtos;
```

Explicação do código:

- MAX(preco): Retorna o maior valor na coluna "preco".

DISTINCT

O comando **DISTINCT** é usado para retornar valores únicos em uma coluna, eliminando duplicatas.

Exemplo de uso: Queremos listar todas as categorias de produtos sem repetição.

```
-- Lista categorias únicas
SELECT DISTINCT categoria
FROM produtos;
```

Explicação do código:

- DISTINCT categoria: Retorna apenas valores únicos da coluna "categoria".

LIKE

O operador **LIKE** é usado para buscar padrões em textos, utilizando curingas como % (qualquer sequência de caracteres) e _ (um único caractere).

Exemplo de uso: Queremos encontrar todos os produtos que começam com "Arroz".

```
-- Busca produtos que começam com "Arroz"
SELECT * FROM produtos
WHERE nome LIKE 'Arroz%';
```

Explicação do código:

- LIKE 'Arroz%': Busca valores que começam com "Arroz".

BETWEEN

O operador **BETWEEN** é usado para filtrar valores dentro de um intervalo específico.

Exemplo de uso: Queremos listar os produtos com preço entre R$ 10,00 e R$ 50,00.

```
-- Filtra produtos com preço entre 10 e 50
SELECT * FROM produtos
WHERE preco BETWEEN 10 AND 50;
```

Explicação do código:

- BETWEEN 10 AND 50: Filtra valores entre 10 e 50 (inclusive).

IN

O operador **IN** é usado para filtrar registros cujo valor corresponde a qualquer um dos valores em uma lista.

Exemplo de uso: Queremos listar os produtos das categorias "Bebidas" e "Limpeza".

```
-- Filtra produtos por categorias específicas
SELECT * FROM produtos
WHERE categoria IN ('Bebidas', 'Limpeza');
```

Explicação do código:

- IN ('Bebidas', 'Limpeza'): Filtra registros onde a categoria é "Bebidas" ou "Limpeza".

IS NULL

O operador **IS NULL** é usado para verificar se um valor é nulo.

Exemplo de uso: Queremos listar os produtos que não têm fornecedor cadastrado.

-- Filtra produtos sem fornecedor

SELECT * FROM produtos

WHERE fornecedor_id IS NULL;

Explicação do código:

- IS NULL: Filtra registros onde a coluna "fornecedor_id" é nula.

ALTER TABLE

O comando **ALTER TABLE** é usado para modificar a estrutura de uma tabela, como adicionar, remover ou modificar colunas.

Exemplo de uso: Vamos adicionar uma nova coluna "desconto" à tabela "produtos".

-- Adiciona uma nova coluna "desconto"

ALTER TABLE produtos

ADD desconto DECIMAL(5, 2);

Explicação do código:

- ALTER TABLE produtos: Especifica a tabela a ser modificada.
- ADD desconto DECIMAL(5, 2): Adiciona uma nova coluna chamada "desconto" com tipo decimal.

1. Renomeie a coluna "categoria" para "tipo_produto".

Claro! Vamos continuar com os comandos de 21 a 30, mantendo a estrutura didática e o cenário da rede de supermercados. Aqui estão:

CREATE TABLE

O comando **CREATE TABLE** é usado para criar uma nova tabela no banco de dados.

Exemplo de uso: Vamos criar uma tabela chamada "promocoes" para armazenar informações sobre promoções.

```
-- Cria a tabela "promocoes"
CREATE TABLE promocoes (
  id INT PRIMARY KEY AUTO_INCREMENT,
  produto_id INT,
  desconto DECIMAL(5, 2),
  data_inicio DATE,
  data_fim DATE,
  FOREIGN KEY (produto_id) REFERENCES produtos(id)
);
```

Explicação do código:

- CREATE TABLE promocoes: Cria uma nova tabela chamada "promocoes".
- id INT PRIMARY KEY AUTO_INCREMENT: Define uma coluna de ID com incremento automático.
- FOREIGN KEY (produto_id) REFERENCES produtos(id): Cria uma chave estrangeira que referencia a tabela "produtos".

DROP TABLE

O comando **DROP TABLE** é usado para excluir uma tabela do banco de dados.

Exemplo de uso: Vamos excluir a tabela "promocoes" que criamos anteriormente.

```
-- Exclui a tabela "promocoes"
DROP TABLE promocoes;
```

Explicação do código:

- DROP TABLE promocoes: Remove a tabela "promocoes" do banco de dados.

TRUNCATE TABLE

O comando **TRUNCATE TABLE** é usado para remover todos os registros de uma tabela, mas mantendo a estrutura da tabela.

Exemplo de uso: Vamos limpar todos os dados da tabela "vendas".

-- Remove todos os registros da tabela "vendas"

TRUNCATE TABLE vendas;

Explicação do código:

- TRUNCATE TABLE vendas: Remove todos os registros, mas mantém a tabela intacta.

CREATE INDEX

O comando **CREATE INDEX** é usado para criar um índice em uma ou mais colunas de uma tabela, melhorando a performance de consultas.

Exemplo de uso: Vamos criar um índice na coluna "nome" da tabela "produtos".

-- Cria um índice na coluna "nome"

CREATE INDEX idx_nome ON produtos(nome);

Explicação do código:

- CREATE INDEX idx_nome: Cria um índice chamado "idx_nome".
- ON produtos(nome): Especifica a coluna "nome" da tabela "produtos".

DROP INDEX

O comando **DROP INDEX** é usado para remover um índice de uma tabela.

Exemplo de uso: Vamos remover o índice "idx_nome" que criamos anteriormente.

-- Remove o índice "idx_nome"

DROP INDEX idx_nome ON produtos;

Explicação do código:

- DROP INDEX idx_nome: Remove o índice chamado "idx_nome".
- ON produtos: Especifica a tabela onde o índice está.

UNION

O comando **UNION** é usado para combinar os resultados de duas ou mais consultas em um único conjunto de resultados, removendo duplicatas.

Exemplo de uso: Queremos listar os nomes de todos os clientes e fornecedores em uma única lista.

-- Combina nomes de clientes e fornecedores

SELECT nome FROM clientes

UNION

SELECT nome FROM fornecedores;

Explicação do código:

- UNION: Combina os resultados das duas consultas, removendo duplicatas.

UNION ALL

O comando **UNION ALL** é semelhante ao **UNION**, mas inclui duplicatas nos resultados.

Exemplo de uso: Queremos listar os nomes de todos os clientes e fornecedores, incluindo duplicatas.

-- Combina nomes de clientes e fornecedores, incluindo duplicatas

SELECT nome FROM clientes

UNION ALL

SELECT nome FROM fornecedores;

Explicação do código:

- UNION ALL: Combina os resultados das duas consultas, mantendo duplicatas.

CASE

O comando **CASE** é usado para criar condições dentro de uma consulta, semelhante a um "if-else" em programação.

Exemplo de uso: Queremos classificar os produtos como "Barato", "Médio" ou "Caro" com base no preço.

-- Classifica os produtos por faixa de preço

SELECT nome, preco,

```
  CASE
    WHEN preco < 10 THEN 'Barato'
    WHEN preco BETWEEN 10 AND 50 THEN 'Médio'
    ELSE 'Caro'
  END AS classificacao
FROM produtos;
```

Explicação do código:

- CASE: Define as condições.
- END AS classificacao: Retorna o resultado da condição como uma nova coluna.

LIMIT

O comando **LIMIT** é usado para limitar o número de registros retornados por uma consulta.

Exemplo de uso: Queremos listar apenas os 5 primeiros produtos.

```
-- Limita o resultado a 5 registros
SELECT * FROM produtos
LIMIT 5;
```

Explicação do código:

- LIMIT 5: Retorna apenas os primeiros 5 registros.

OFFSET

O comando **OFFSET** é usado em conjunto com **LIMIT** para pular um número específico de registros antes de retornar os resultados.

Exemplo de uso: Queremos listar os produtos 6 a 10.

```
-- Pula os 5 primeiros registros e retorna os próximos 5
SELECT * FROM produtos
LIMIT 5 OFFSET 5;
```

Explicação do código:

- OFFSET 5: Pula os primeiros 5 registros.
- LIMIT 5: Retorna os próximos 5 registros.

SUBQUERY

Uma **SUBQUERY** é uma consulta dentro de outra consulta. Ela pode ser usada em várias partes de uma consulta principal, como no **SELECT**, **FROM**, **WHERE**, etc.

Exemplo de uso: Queremos listar os produtos que têm preço acima da média.

```
-- Lista produtos com preço acima da média

SELECT nome, preco

FROM produtos

WHERE preco > (SELECT AVG(preco) FROM produtos);
```

Explicação do código:

- (SELECT AVG(preco) FROM produtos): Subquery que calcula o preço médio dos produtos.
- WHERE preco > (...): Filtra os produtos com preço acima da média.

EXISTS

O operador **EXISTS** é usado para verificar se uma subquery retorna algum resultado. Ele retorna **TRUE** se a subquery retornar pelo menos um registro.

Exemplo de uso: Queremos listar os clientes que fizeram pelo menos uma compra.

```
-- Lista clientes que fizeram compras

SELECT nome

FROM clientes c

WHERE EXISTS (SELECT 1 FROM vendas v WHERE v.cliente_id = c.id);
```

Explicação do código:

- EXISTS (SELECT 1 FROM vendas v WHERE v.cliente_id = c.id): Verifica se há vendas associadas ao cliente.

NOT EXISTS

O operador **NOT EXISTS** é o oposto do **EXISTS**. Ele retorna **TRUE** se a subquery não retornar nenhum resultado.

Exemplo de uso: Queremos listar os produtos que nunca foram vendidos.

```
-- Lista produtos que nunca foram vendidos

SELECT nome

FROM produtos p

WHERE NOT EXISTS (SELECT 1 FROM vendas v WHERE v.produto_id = p.id);
```

Explicação do código:

- NOT EXISTS (SELECT 1 FROM vendas v WHERE v.produto_id = p.id): Verifica se não há vendas associadas ao produto.

VIEW

Uma **VIEW** é uma tabela virtual criada a partir de uma consulta SQL. Ela não armazena dados, mas permite visualizar os resultados de uma consulta como se fosse uma tabela.

Exemplo de uso: Vamos criar uma view para listar os produtos com estoque baixo.

```
-- Cria uma view para produtos com estoque baixo

CREATE VIEW produtos_estoque_baixo AS

SELECT nome, estoque

FROM produtos

WHERE estoque < 50;
```

Explicação do código:

- CREATE VIEW produtos_estoque_baixo AS: Cria uma view chamada "produtos_estoque_baixo".
- SELECT nome, estoque FROM produtos WHERE estoque < 50: Define a consulta que a view executará.

DROP VIEW

O comando **DROP VIEW** é usado para excluir uma view do banco de dados.

Exemplo de uso: Vamos excluir a view "produtos_estoque_baixo" que criamos anteriormente.

```
-- Exclui a view "produtos_estoque_baixo"

DROP VIEW produtos_estoque_baixo;
```

Explicação do código:

- DROP VIEW produtos_estoque_baixo: Remove a view do banco de dados.

WITH (CTE - Common Table Expressions)

O comando **WITH** é usado para criar uma CTE (Common Table Expression), que é uma tabela temporária que pode ser usada em uma consulta principal.

Exemplo de uso: Queremos listar os produtos com estoque abaixo da média usando uma CTE.

```
-- Usa uma CTE para listar produtos com estoque abaixo da média

WITH estoque_medio AS (
  SELECT AVG(estoque) AS media_estoque
  FROM produtos
)

SELECT nome, estoque

FROM produtos, estoque_medio

WHERE estoque < media_estoque;
```

Explicação do código:

- WITH estoque_medio AS (...): Cria uma CTE chamada "estoque_medio".
- SELECT nome, estoque FROM produtos, estoque_medio WHERE estoque < media_estoque: Usa a CTE na consulta principal.

ROW_NUMBER()

A função **ROW_NUMBER()** é usada para atribuir um número único a cada linha de um conjunto de resultados, com base em uma ordenação.

Exemplo de uso: Queremos numerar os produtos por ordem de preço.

```
-- Numera os produtos por ordem de preço
```

```
SELECT nome, preco,
  ROW_NUMBER() OVER (ORDER BY preco) AS numero_linha
FROM produtos;
```

Explicação do código:

- ROW_NUMBER() OVER (ORDER BY preco): Atribui um número único a cada linha, ordenada pelo preço.

RANK()

A função **RANK()** é semelhante ao **ROW_NUMBER()**, mas atribui o mesmo número a linhas com valores iguais, pulando os números subsequentes.

Exemplo de uso: Queremos classificar os produtos por preço, com empates.

```
-- Classifica os produtos por preço, com empates
SELECT nome, preco,
  RANK() OVER (ORDER BY preco) AS classificacao
FROM produtos;
```

Explicação do código:

- RANK() OVER (ORDER BY preco): Atribui uma classificação, considerando empates.

DENSE_RANK()

A função **DENSE_RANK()** é semelhante ao **RANK()**, mas não pula números subsequentes em caso de empates.

Exemplo de uso: Queremos classificar os produtos por preço, sem pular números em caso de empates.

```
-- Classifica os produtos por preço, sem pular números
SELECT nome, preco,
  DENSE_RANK() OVER (ORDER BY preco) AS classificacao
FROM produtos;
```

Explicação do código:

- DENSE_RANK() OVER (ORDER BY preco): Atribui uma classificação sem pular números.

PARTITION BY

A cláusula **PARTITION BY** é usada em conjunto com funções de janela (como **ROW_NUMBER()**, **RANK()**, etc.) para dividir os resultados em grupos.

Exemplo de uso: Queremos numerar os produtos dentro de cada categoria.

```
-- Numera os produtos dentro de cada categoria
SELECT nome, categoria, preco,
  ROW_NUMBER() OVER (PARTITION BY categoria ORDER BY preco) AS numero_linha
FROM produtos;
```

Explicação do código:

- PARTITION BY categoria: Divide os resultados por categoria.
- ROW_NUMBER() OVER (...): Numera os produtos dentro de cada categoria.

LEAD()

A função **LEAD()** é usada para acessar o valor de uma linha subsequente em um conjunto de resultados, com base em uma ordenação.

Exemplo de uso: Queremos comparar o preço de cada produto com o preço do próximo produto na lista.

```
-- Compara o preço de cada produto com o próximo
SELECT nome, preco,
  LEAD(preco) OVER (ORDER BY preco) AS proximo_preco
FROM produtos;
```

Explicação do código:

- LEAD(preco) OVER (ORDER BY preco): Acessa o preço da próxima linha, ordenada por preço.

LAG()

A função **LAG()** é usada para acessar o valor de uma linha anterior em um conjunto de resultados, com base em uma ordenação.

Exemplo de uso: Queremos comparar o preço de cada produto com o preço do produto anterior na lista.

```
-- Compara o preço de cada produto com o anterior
SELECT nome, preco,
  LAG(preco) OVER (ORDER BY preco) AS preco_anterior
FROM produtos;
```

Explicação do código:

- LAG(preco) OVER (ORDER BY preco): Acessa o preço da linha anterior, ordenada por preço.

COALESCE()

A função **COALESCE()** é usada para retornar o primeiro valor não nulo em uma lista de valores.

Exemplo de uso: Queremos exibir o preço de um produto, mas se o preço for nulo, exibir "Preço não disponível".

```
-- Retorna o preço ou uma mensagem se for nulo
SELECT nome,
  COALESCE(CAST(preco AS VARCHAR), 'Preço não disponível') AS preco_formatado
FROM produtos;
```

Explicação do código:

- COALESCE(CAST(preco AS VARCHAR), 'Preço não disponível'): Retorna o preço ou a mensagem se o preço for nulo.

NULLIF()

A função **NULLIF()** é usada para retornar **NULL** se dois valores forem iguais; caso contrário, retorna o primeiro valor.

Exemplo de uso: Queremos retornar **NULL** se o preço de um produto for igual a zero.

```
-- Retorna NULL se o preço for zero
SELECT nome,
  NULLIF(preco, 0) AS preco_ajustado
FROM produtos;
```

Explicação do código:

- NULLIF(preco, 0): Retorna **NULL** se o preço for zero; caso contrário, retorna o preço.

CAST()

A função **CAST()** é usada para converter um valor de um tipo de dado para outro.

Exemplo de uso: Queremos converter o preço de um produto para uma string.

```
-- Converte o preço para uma string
SELECT nome,
  CAST(preco AS VARCHAR) AS preco_string
FROM produtos;
```

Explicação do código:

- CAST(preco AS VARCHAR): Converte o preço (um número) para uma string.

CONVERT()

A função **CONVERT()** é semelhante ao **CAST()**, mas permite mais flexibilidade em alguns bancos de dados, como o SQL Server.

Exemplo de uso: Queremos converter a data de uma venda para o formato "YYYY-MM-DD".

```
-- Converte a data para o formato "YYYY-MM-DD"
SELECT id,
  CONVERT(VARCHAR, data_venda, 23) AS data_formatada
FROM vendas;
```

Explicação do código:

- CONVERT(VARCHAR, data_venda, 23): Converte a data para o formato "YYYY-MM-DD".

CONCAT()

A função **CONCAT()** é usada para concatenar (juntar) duas ou mais strings.

Exemplo de uso: Queremos criar uma descrição completa para cada produto, juntando o nome e a categoria.

```
-- Concatena o nome e a categoria do produto

SELECT CONCAT(nome, ' - ', categoria) AS descricao_completa

FROM produtos;
```

Explicação do código:

- CONCAT(nome, ' - ', categoria): Junta o nome, um traço e a categoria.

SUBSTRING()

A função **SUBSTRING()** é usada para extrair uma parte de uma string.

Exemplo de uso: Queremos extrair os primeiros 3 caracteres do nome de cada produto.

```
-- Extrai os primeiros 3 caracteres do nome

SELECT SUBSTRING(nome, 1, 3) AS iniciais

FROM produtos;
```

Explicação do código:

- SUBSTRING(nome, 1, 3): Extrai os caracteres da posição 1 até a posição 3.

REPLACE()

A função **REPLACE()** é usada para substituir todas as ocorrências de uma substring por outra substring.

Exemplo de uso: Queremos substituir todas as ocorrências de "kg" por "quilograma" no nome dos produtos.

```
-- Substitui "kg" por "quilograma"

SELECT REPLACE(nome, 'kg', 'quilograma') AS nome_ajustado

FROM produtos;
```

Explicação do código:

- REPLACE(nome, 'kg', 'quilograma'): Substitui "kg" por "quilograma".

TRIM()

A função **TRIM()** é usada para remover espaços em branco no início e no final de uma string.

Exemplo de uso: Queremos remover espaços em branco no nome dos produtos.

```
-- Remove espaços em branco no início e no final

SELECT TRIM(nome) AS nome_ajustado

FROM produtos;
```

Explicação do código:

- TRIM(nome): Remove espaços em branco no início e no final do nome.