

# DOMINE AS FUNÇÕES BÁSICAS SQL



**Vault-79: Desvendando Fortalezas de  
Dados com SQL**

**Alielson Ferreira**



# PRINCIPAIS FUNÇÕES SQL

Simplificando o uso das funções básicas SQL

Bem-vindo à Vault-79, onde os segredos mais bem guardados estão esperando para serem revelados através do poder do SQL. Neste eBook, vamos explorar as profundezas do SQL enquanto você navega pelo terminal da Vault 79, em busca de informações vitais para sua jornada.



# 01

## Recuperando Dados

---

Dominar a extração eficaz de dados é crucial para análises e decisões informadas. Capacita a exploração e compreensão dos dados armazenados, fundamentais para profissionais que lidam com bancos de dados.

# SELECT: EXPLORANDO OS DADOS

No coração do SQL está a cláusula **SELECT**, permitindo que você recupere dados de tabelas. Imagine que você está procurando por registros de suprimentos na Vault 79. Vamos dar uma olhada:

```
SQL  
  
SELECT * FROM suprimentos;
```

Isso traz todos os registros da tabela "suprimentos", revelando o que está armazenado na Vault.

```
SQL  
  
| Nome                | Tipo          | Quantidade |  
|-----|-----|-----|  
| Stimpak             | Médico       | 50         |  
| RadAway             | Médico       | 20         |  
| Munição 10mm        | Armas        | 200        |  
| Munição .308        | Armas        | 100        |  
| Água purificada     | Consumíveis  | 30         |
```

## WHERE: FILTRANDO RESULTADOS

Às vezes, você precisa ser específico. Suponha que você esteja interessado apenas em suprimentos médicos. Você pode usar a cláusula **WHERE** para isso:

```
SQL

SELECT * FROM suprimentos WHERE tipo = 'médico';
```

Agora você só vê os suprimentos médicos, filtrando os resultados.

```
SQL

| Nome          | Tipo      | Quantidade |
|-----|-----|-----|
| Stimpak       | Médico    | 50         |
| RadAway       | Médico    | 20         |
```

## ORDER BY: ORDENANDO RESULTADOS

Se você quiser classificar os resultados por ordem alfabética, por exemplo, use **ORDER BY**:

**ASC**: Ordena os resultados em ordem crescente, do menor para o maior.

**DESC**: Ordena os resultados em ordem decrescente, do maior para o menor.

```
SELECT * FROM suprimentos ORDER BY nome ASC;
```

Agora você vê os suprimentos em ordem alfabética pelo nome.

Nome	Tipo	Quantidade
Água purificada	Consumíveis	30
Munição .308	Armas	100
Munição 10mm	Armas	200
RadAway	Médico	20
Stimpak	Médico	50



## DISTINCT: REMOVENDO DUPLICATAS

O comando DISTINCT é usado para selecionar valores únicos em uma coluna. No contexto da Vault, se houver várias entradas para a coluna "profissão" na tabela "habitantes", o DISTINCT garantirá que apenas uma instância de cada profissão única seja exibida, removendo duplicatas.

```
SQL  
  
SELECT DISTINCT profissão FROM habitantes;
```

```
| Profissão |  
|-----|  
| Médico   |  
| Engenheiro |  
| Segurança |  
| Agricultor |
```

# 02

## FILTRANDO E AGRUPANDO DADOS

---

Exploraremos a organização e refinamento de dados em bancos, filtrando-os para análises precisas e explorando agrupamentos para obter insights valiosos sobre tendências e padrões dentro do conjunto de dados.



# Funções de Agregação (COUNT, SUM, AVG, MAX, MIN)

Vamos explorar a tabela "habitantes" para entender como esses conceitos funcionam em conjunto.

```
SQL

SELECT
    COUNT(*) as total_habitantes,
    AVG(idade) as idade_media,
    MAX(idade) as idade_maxima,
    MIN(idade) as idade_minima
FROM
    habitantes;
```

**COUNT(\*):** Conta o número total de habitantes na Vault.

**AVG(idade):** Calcula a idade média dos habitantes.

**MAX(idade):** Determina a idade máxima entre os habitantes.

**MIN(idade):** Determina a idade mínima entre os habitantes

```
SQL

| Total_Habitantes | Idade_Media | Idade_Maxima | Idade_Minima |
|-----|-----|-----|-----|
| 50 | 32 | 40 | 20 |
```

# GROUP BY: AGRUPANDO E AGREGANDO DADOS

A cláusula **GROUP BY** é usada para agrupar linhas que têm o mesmo valor em uma ou mais colunas e aplicar funções de agregação a elas. Isso nos permite segmentar os dados com base em categorias específicas e realizar cálculos agregados dentro desses grupos.

```
SQL

SELECT profissao, COUNT(*) as total_habitantes
FROM habitantes
GROUP BY profissao;
```

Estamos agrupando os habitantes por profissão e contando quantos habitantes pertencem a cada profissão. Isso nos mostra quantos habitantes desempenham cada função, oferecendo uma visão da distribuição da força de trabalho na Vault.

```
SQL

| Profissao | Total_Habitantes |
|-----|-----|
| Médico   | 10                |
| Engenheiro | 8                 |
| Segurança | 12                |
| Agricultor | 5                 |
```

## WHERE: CONDIÇÕES AVANÇADAS

A cláusula **WHERE** é usada para filtrar os resultados de uma consulta com base em uma condição específica. Condições mais avançadas podem incluir operadores lógicos como **AND**, **OR** e **NOT**, juntamente com operadores de comparação como **>**, **<**, **=**, **!=**, entre outros.

SQL

```
SELECT * FROM habitantes WHERE idade > 30 AND profissao = 'Engenheiro';
```

O resultado exibe habitantes que possuem idade acima de 30 anos e que exercem a profissão de engenheiro.

SQL

Nome	Idade	Profissao
John	35	Engenheiro
Emily	32	Engenheiro

## HAVING: FILTRANDO AGRUPAMENTOS

A cláusula **HAVING** é usada em conjunto com a cláusula **GROUP BY** para filtrar os resultados de grupos agregados com base em uma condição específica. Enquanto a cláusula **WHERE** é usada para filtrar linhas antes da agregação, a cláusula **HAVING** é usada para filtrar grupos depois da agregação.

```
SQL

SELECT profissao, COUNT(*) as total_habitantes
FROM habitantes
GROUP BY profissao
HAVING COUNT(*) > 10;
```

Estamos contando quantos habitantes têm cada profissão e usando a cláusula **HAVING** para mostrar apenas as profissões com mais de 10 membros.

```
SQL

| Profissao | Total_Habitantes |
|-----|-----|
| Segurança | 12                |
```



# 03

## MANIPULAÇÃO DE DADOS

---

Explore a manipulação prática de dados em bancos, aprendendo a adicionar, atualizar e excluir informações. Compreender e dominar operações como INSERT, UPDATE e DELETE é essencial para a manutenção eficaz de bancos de dados

# INSERT: INSERINDO DADOS

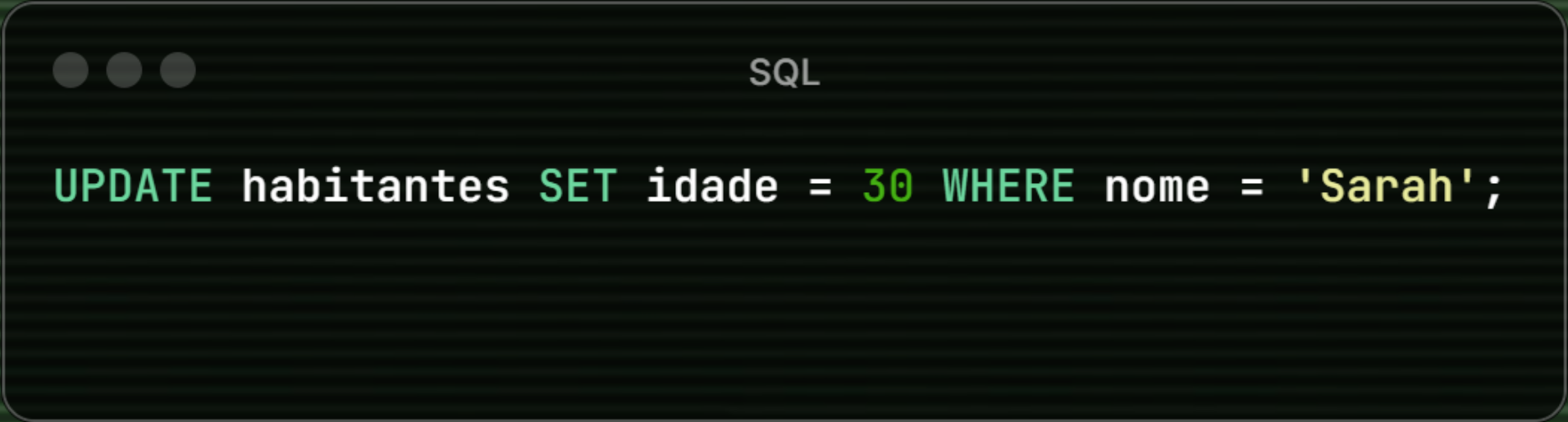
O comando **INSERT** é usado para adicionar novos registros a uma tabela.

```
SQL  
  
INSERT INTO habitantes (nome, idade, profissao) VALUES ('Sarah', 28, 'Médico');
```

Após a execução deste comando, um novo habitante chamado Sarah, com 28 anos de idade e profissão de médico, seria adicionado à tabela "habitantes".

## UPDATE: ATUALIZANDO DADOS

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

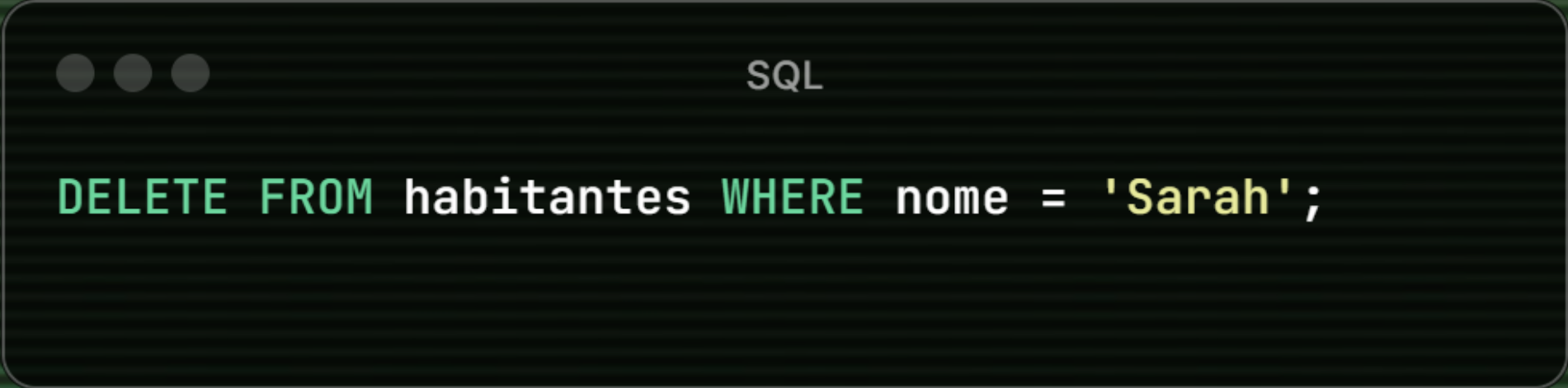
A dark-themed window with a title bar containing three window control buttons on the left and the text "SQL" on the right. The main area of the window contains a single line of SQL code.

```
UPDATE habitantes SET idade = 30 WHERE nome = 'Sarah';
```

Este comando atualizaria o registro de Sarah na tabela "habitantes", alterando sua idade para 30 anos.

## DELETE: REMOVENDO DADOS

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

A dark-themed window with a title bar containing three circles on the left and the text "SQL" on the right. The main area contains a SQL command in a monospaced font.

```
DELETE FROM habitantes WHERE nome = 'Sarah';
```

Após a execução deste comando, o registro de Sarah seria removido da tabela "habitantes".



# AGRADECIMENTOS

---

# OBRIGADO POR LER ATÉ AQUI

Esse Ebook foi gerado por IA, e diagramado por humano.  
O passo a passo se encontra no meu Github

•

Esse conteúdo foi gerado com fins didáticos de construção, não foi realizado uma validação cuidadosa humana no conteúdo e pode conter erros gerados por uma IA.



<https://github.com/alielsonfp/prompts-recipe-to-create-a-ebook>

