Guia SQL

O Manual Supremo para Iniciantes

Rodrigo Santana Pereira

Índice

Introdução

Bem-vindo ao seu guia definitivo sobre SQL! Se você é novo no mundo da programação e está ansioso para explorar o fascinante universo dos bancos de dados, está no lugar certo. SQL, ou Structured Query Language, é a linguagem padrão para gerenciar e manipular bancos de dados relacionais. Ao longo deste ebook, você aprenderá desde os conceitos básicos até comandos mais avançados, utilizando o MySQL, um dos sistemas de gerenciamento de banco de dados mais populares. Vamos começar essa jornada!

IMPORTANTE: Esse Ebook foi gerado por IA, e diagramado por humano. O seu conteúdo foi gerado com fins didáticos de construção, e não foi realizado uma validação cuidadosa humana no conteúdo. Desta forma, esse Ebook pode conter erros gerados por uma IA.

01

Tipos de Bancos de Dados

Tipos de Bancos de Dados

Os bancos de dados são fundamentais para armazenar, organizar e recuperar informações. Existem vários tipos de bancos de dados, cada um com características e usos específicos.

1. Bancos de Dados Relacionais (RDBMS):
    1. Estruturados em tabelas com linhas e colunas;
    2. Relacionam dados entre si através de chaves;
    3. Exemplos: MySQL, PostgreSQL, Oracle.

2. Bancos de Dados NoSQL:
    1. Não usam tabelas tradicionais;
    2. Flexíveis e escaláveis;
    3. Ideais para grandes volumes de dados não estruturados;
    4. Exemplos: MongoDB, Cassandra.

3. Bancos de Dados em Nuvem:
    1. Hospedados na nuvem;
    2. Oferecem escalabilidade e acessibilidade;
    3. Exemplos: Amazon RDS, Google Cloud SQL.

4. Bancos de Dados em Memória:
    1. Armazenam dados na memória RAM para acesso rápido;
    2. Usados para aplicações que requerem alta performance;
    3. Exemplos: Redis, Memcached.
    4.

Sugestão: Para iniciantes, os bancos de dados relacionais são o ponto de partida ideal. Vamos explorá-los em detalhes a seguir.

02

Bancos de Dados Relacionais

Bancos de Dados Relacionais

Os bancos de dados relacionais organizam dados em tabelas, que são como planilhas de Excel. Cada tabela contém linhas (registros) e colunas (campos), facilitando a manipulação e a consulta de dados.

Exemplo: Imagine um banco de dados para uma biblioteca. Ele pode ter tabelas como "Livros", "Autores" e "Empréstimos".

CREATE TABLE

```sql
CREATE TABLE Livros (
    LivroID INT AUTO_INCREMENT,
    Titulo VARCHAR(255) NOT NULL,
    AutorID INT,
    PRIMARY KEY (LivroID)
);

CREATE TABLE Autores (
    AutorID INT AUTO_INCREMENT,
    Nome VARCHAR(255) NOT NULL,
    PRIMARY KEY (AutorID)
);

CREATE TABLE Emprestimos (
    EmprestimoID INT AUTO_INCREMENT,
    LivroID INT,
    DataEmprestimo DATE,
    PRIMARY KEY (EmprestimoID)
);
```

03

Relacionamento entre Tabelas

Relacionamento entre Tabelas

Tabelas em bancos de dados relacionais frequentemente têm relacionamentos que ajudam a conectar dados de forma lógica. Os tipos mais comuns de relacionamentos são:

1.Um para Um:

1. Cada linha de uma tabela corresponde a uma linha em outra tabela.
2. Exemplo: Cada pessoa tem um único passaporte.

2.Um para Muitos:

1. Uma linha em uma tabela corresponde a várias linhas em outra tabela.
2. Exemplo: Um autor pode ter vários livros.

3.Muitos para Muitos:

1. Várias linhas em uma tabela correspondem a várias linhas em outra tabela.
2. Exemplo: Livros e categorias, onde um livro pode pertencer a várias categorias e vice-versa.

Exemplo: No nosso banco de dados da biblioteca, o relacionamento entre "Autores" e "Livros" é de um para muitos.

ADD FOREIGN KEY

```
ALTER TABLE Livros
ADD FOREIGN KEY (AutorID) REFERENCES Autores(AutorID);
```

04

Chaves

Chaves

As chaves são fundamentais para definir e manter os relacionamentos entre tabelas. As chaves podem ser:

1.Chave Primária (Primary Key):

- Identifica exclusivamente cada linha em uma tabela;
- Cada tabela deve ter uma chave primária;
- Exemplo: LivroID na tabela "Livros".

2.Chave Estrangeira (Foreign Key):

- Relaciona uma tabela com outra;
- Garante que os valores de uma coluna correspondam aos valores de uma coluna em outra tabela;
- Exemplo: AutorID em "Livros", referenciando AutorID em "Autores".

05

O Que é SQL?

O que é SQL?

SQL, ou Structured Query Language, é a linguagem usada para se comunicar com bancos de dados relacionais. Com SQL, você pode:

- Consultar dados (SELECT);
- Inserir novos dados (INSERT);
- Atualizar dados existentes (UPDATE);
- Deletar dados (DELETE).

Exemplo: Vamos selecionar todos os livros da nossa tabela "Livros".

Exemplo SQL

```sql
SELECT * FROM Livros;
```

06

Sublinguagens SQL

Chaves

SQL é dividido em várias sublinguagens, cada uma com um propósito específico:

1. DML (Data Manipulation Language):

    - Manipula os dados no banco dedados.
    - Comandos principais: SELECT, INSERT, UPDATE, DELETE.

2. DDL (Data Definition Language):

    - Define e modifica a estrutura do banco de dados.
    - Comandos principais: CREATE, ALTER, DROP.

3. DCL (Data Control Language):

    - Controla o acesso ao banco de dados.
    - Comandos principais: GRANT, REVOKE.

4. TCL (Transaction Control Language):

    - Gerencia transações no banco de dados.
    - Comandos principais: COMMIT, ROLLBACK, SAVEPOINT.

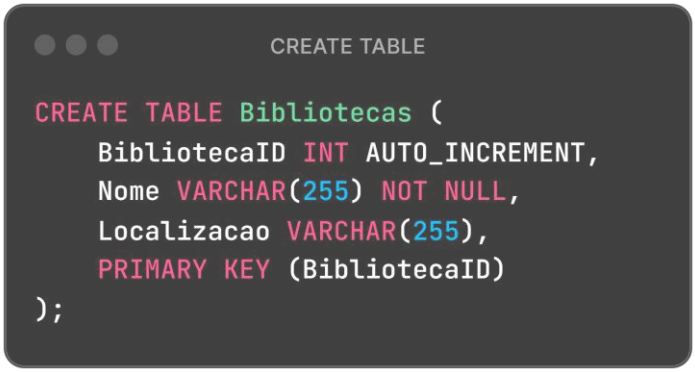
07

Comandos DDL

Comandos DDL

Os comandos DDL são usados para criar e modificar a estrutura de banco de dados. Aqui estão os mais comuns:

1. **CREATE:** Cria novos objetos no banco de dados, como tabelas e índices.



2. **ALTER:** Modifica a estrutura de um objeto existente.

3. **DROP:** Remove objetos do banco de dados.

Comandos DDL

4. **TRUNCATE:** Remove todos os registros de uma tabela, mas mantém a estrutura da tabela.

08

Comandos DML

Comandos DDL

Os comandos DML são usados para manipular dados dentro das tabelas. Aqui estão os principais:

1. **SELECT:** Recupera dados das tabelas.

```sql
SELECT Titulo, AnoPublicacao FROM Livros WHERE AutorID = 1;
```

2. **INSERT:** Adiciona novos registros a uma tabela.

```sql
INSERT INTO Autores (Nome) VALUES ('Carlos Silva');
```

Comandos DDL

3. **UPDATE:** Modifica dados existentes.

```
UPDATE
UPDATE Livros SET AnoPublicacao = 2021 WHERE LivroID = 1;
```

4. **DELETE:** Remove dados de uma tabela.

```
DELETE
DELETE FROM Livros WHERE LivroID = 2;
```

09

SELECT

SELECT

A instrução SELECT é o coração do SQL. Ela permite que você extraia dados das tabelas do banco de dados. Vamos explorar as várias maneiras de usá-la.

WHERE e Operadores de Comparação

A cláusula WHERE filtra os registros com base em condições específicas. Aqui estão os operadores de comparação mais comuns:

Operador	Descrição
`=`	Igual
`<>`	Diferente
`>`	Maior que
`<`	Menor que
`>=`	Maior ou igual
`<=`	Menor ou igual
`BETWEEN`	Dentro de um intervalo (inclusivo)
`IN`	Dentro de um conjunto de valores

Além dos operadores de comparação, os operadores de negação e condicionais são fundamentais para construir consultas complexas.

Operador	Descrição
`NOT`	Negação. Inverte o valor de uma condição
`AND`	Condicional E. Verdadeiro se ambas as condições forem verdadeiras
`OR`	Condicional OU. Verdadeiro se pelo menos uma das condições for verdadeira

SELECT

DISTINCT

A palavra-chave DISTINCT remove duplicatas dos resultados. Utilize DISTINCT quando precisar de uma lista única de valores de uma coluna.

Exemplo: Selecionar os diferentes anos de publicação dos livros.

SELECT DISTINCT

```
SELECT DISTINCT AnoPublicacao FROM Livros;
```

SELECT

COUNT

A função COUNT retorna o número de registros que atendem a um critério especificado. COUNT é ideal para obter contagens rápidas de registros.

GROUP BY

A cláusula GROUP BY agrupa registros por uma ou mais colunas. GROUP BY é essencial para resumir dados de forma organizada.

Exemplo: Agrupar os livros por ano de publicação e contar quantos livros foram publicados em cada ano

SELECT COUNT/GROUP BY

```
SELECT AnoPublicacao, COUNT(*) AS TotalLivros FROM Livros GROUP BY AnoPublicacao;
```

SELECT

LIKE

O operador LIKE busca por padrões em colunas de texto. Este pode ser utilizado para buscas flexíveis em colunas de texto. É possível utilizar caracteres especiais nas buscas com Like, sendo eles:

Caractere	Descrição
`%`	Corresponde a qualquer sequência de caracteres (incluindo nenhuma)
`_`	Corresponde a qualquer caractere único

Exemplo: Selecionar todos os livros cujo título começa com "SQL".

SELECT LIKE

```sql
SELECT Titulo FROM Livros WHERE Titulo LIKE 'SQL%';
```

SELECT

Subconsultas

Subconsultas são consultas aninhadas dentro de outra consulta. Estas são poderosas para criar consultas mais complexas e aninhadas.

Exemplo: Selecionar livros com ano de publicação superior à média.

SUBCONSULTAS

```
SELECT Titulo, AnoPublicacao FROM Livros
WHERE AnoPublicacao > (SELECT AVG(AnoPublicacao) FROM Livros);
```

SELECT

JOIN

JOIN combina registros de duas ou mais tabelas com base em uma coluna relacionada.Tipos de JOIN:

- **INNER JOIN:** Retorna registros que têm correspondências em ambas as tabelas.
- **LEFT JOIN:** Retorna todos os registros da tabela da esquerda e os correspondentes da tabela da direita.
- **RIGHT JOIN:** Retorna todos os registros da tabela da direita e os correspondentes da tabela da esquerda.
- **FULL JOIN:** Retorna todos os registros quando há uma correspondência em uma das tabelas.Exemplo: Selecionar todos os livros com seus autores.

JOIN

```sql
SELECT Livros.Titulo, Autores.Nome
FROM Livros
INNER JOIN Autores ON Livros.AutorID = Autores.AutorID;
```

10

Funções Escalares

Funções Escalares

Funções escalares operam em um único valor e retornam um valor único. Elas são úteis para operações como manipulação de texto, cálculos e formatação de datas.Algumas funções escalares comuns incluem:

- **UPPER:** Converte texto para maiúsculas.
- **LOWER:** Converte texto para minúsculas.
- **LEN:** Retorna o comprimento de uma string.
- **ROUND:** Arredonda um número para um número especificado de casas decimais.
- **GETDATE:** Retorna a data e hora atuais.
- **NOW():** Retorna a data e hora atuais.
- **CURDATE():** Retorna a data atual.
- **CURTIME():** Retorna a hora atual.
- **DATE_FORMAT():** Formata uma data de acordo com o formato especificado
- **DATEDIFF():** Calcula a diferença entre duas datas.
- **DAY():** Extrai o dia do mês de uma data.
- **MONTH():** Extrai o mês de uma data.
- **YEAR():** Extrai o ano de uma data.
- **USER():** Retorna o nome do usuário e o host atual.
- **CURRENT_USER():** Retorna o nome do usuário atualmente autenticado no servidor.
- **SESSION_USER():** Retorna o nome do usuário da sessão atual.

Funções Escalares

Exemplo:

Funções Escalares

```
SELECT
    SELECT UPPER(Titulo) AS TituloMaiusculo FROM Livros,
    NOW() AS DataHoraAtual,
    CURDATE() AS DataAtual,
    CURTIME() AS HoraAtual,
    DATE_FORMAT(NOW(), '%d/%m/%Y %H:%i:%s') AS DataHoraFormatada,
    DATE_ADD(CURDATE(), INTERVAL 7 DAY) AS DataProximaSemana,
    DATEDIFF('2024-12-31', CURDATE()) AS DiasRestantes,
    DAY(CURDATE()) AS Dia,
    MONTH(CURDATE()) AS Mes,
    YEAR(CURDATE()) AS Ano,
    USER() AS UsuarioAtual,
    CURRENT_USER() AS UsuarioAutenticado,
    SESSION_USER() AS UsuarioSessao,
    SYSTEM_USER() AS UsuarioSistema;
```

11

Funções Definidas pelo Usuário (Functions)

Functions

Funções definidas pelo usuário permitem criar operações personalizadas que retornam um valor escalar ou uma tabela. Elas são úteis para encapsular lógica repetitiva e reutilizável.

Tipos de função:

- DETERMINISTIC: Use quando a função precisa ser previsível e retornar o mesmo resultado para as mesmas entradas, ideal para cálculos matemáticos ou funções baseadas em regras fixas.
- NON-DETERMINISTIC: Use quando o resultado pode variar com o tempo ou outras condições, como funções de data/hora ou funções que dependem de variáveis externas.
- CONTAINS SQL: Use para funções que contêm comandos SQL mas não alteram dados, úteis para cálculos baseados em leituras de dados.
- READS SQL DATA: Similar ao CONTAINS SQL, indicado especificamente para funções que leem dados sem alterá-los.
- MODIFIES SQL DATA: Use com cautela para funções que precisam modificar dados, como funções que atualizam registros ou inserem novos dados.

Functions

Exemplo:

FUNCTION

```
-- Função determinística para calcular desconto
CREATE FUNCTION CalcularDesconto(Preco DECIMAL(10,2), Percentual DECIMAL(5,2))
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    RETURN Preco - (Preco * Percentual / 100);
END;

-- Função não determinística para retornar saudação
CREATE FUNCTION Saudacao()
RETURNS VARCHAR(50)
BEGIN
    DECLARE SaudacaoMsg VARCHAR(50);
    IF HOUR(CURTIME()) < 12 THEN
        SET SaudacaoMsg = 'Bom Dia!';
    ELSEIF HOUR(CURTIME()) < 18 THEN
        SET SaudacaoMsg = 'Boa Tarde!';
    ELSE
        SET SaudacaoMsg = 'Boa Noite!';
    END IF;
    RETURN SaudacaoMsg;
END;

-- Função que lê dados sem modificar
CREATE FUNCTION ObterNomeAutor(AutorID INT)
RETURNS VARCHAR(255)
CONTAINS SQL
BEGIN
    DECLARE NomeAutor VARCHAR(255);
    SELECT Nome INTO NomeAutor FROM Autores WHERE AutorID = AutorID;
    RETURN NomeAutor;
END;

-- Função que modifica dados
CREATE FUNCTION AtualizarEstoqueVenda(LivroID INT, QuantidadeVendida INT)
RETURNS BOOLEAN
MODIFIES SQL DATA
BEGIN
    UPDATE Estoque
    SET Quantidade = Quantidade - QuantidadeVendida
    WHERE LivroID = LivroID;
    RETURN TRUE;
END;
```

12

Visões (Views)

Views

Views são consultas armazenadas que podem ser tratadas como tabelas virtuais. Elas facilitam o acesso a dados complexos e a agregação de dados de várias tabelas.

Exemplo: Criar uma view que mostra os títulos dos livros com seus respectivos autores.

VIEW

```
CREATE VIEW LivrosAutores AS
SELECT Livros.Titulo, Autores.Nome AS Autor
FROM Livros
INNER JOIN Autores ON Livros.AutorID = Autores.AutorID;
```

13

Procedimentos Armazenados (Stored Procedures)

Stored Procedures

Stored Procedures são conjuntos de comandos SQL armazenados no banco de dados que podem ser executados sob demanda. Elas ajudam a automatizar tarefas repetitivas e encapsular lógica de negócios.

Exemplo: Criar um procedimento armazenado para adicionar um novo livro.

PROCEDURE

```
CREATE PROCEDURE AdicionarLivro (IN Titulo VARCHAR(255), IN AutorID INT, IN AnoPublicacao
YEAR)
BEGIN
INSERT INTO Livros (Titulo, AutorID, AnoPublicacao)
VALUES (Titulo, AutorID, AnoPublicacao);
END;
```

14

Variáveis no SQL

Variáveis no SQL

As variáveis em SQL são usadas para armazenar temporariamente valores que podem ser manipulados e utilizados em consultas e procedimentos. No MySQL, você pode declarar e utilizar variáveis em blocos de código para tornar suas operações mais flexíveis e dinâmicas.

Declarando Variáveis

Para declarar uma variável no MySQL, você pode usar a instrução DECLARE dentro de um bloco de código BEGIN ... END (tipicamente em procedimentos armazenados ou funções).

Sintaxe: DECLARE nome_variavel TIPO [DEFAULT valor_inicial];

- **nome_variavel:** Nome da variável.
- **TIPO:** Tipo de dado da variável (INT, VARCHAR, DATE, etc.).
- **DEFAULT valor_inicial**: Opcional. Define um valor inicial para a variável.

Variáveis no SQL

Atribuindo Valores às Variáveis

Você pode atribuir valores às variáveis usando a instrução SET ou a instrução SELECT ... INTO.

VARIÁVEL

```sql
SELECT COUNT(*) INTO contador FROM Livros;
SELECT 'Bom Dia!' INTO saudacao;
```

VARIÁVEL

```sql
DELIMITER //

CREATE PROCEDURE SaudacaoPersonalizada()
BEGIN
    DECLARE saudacao VARCHAR(50);
    DECLARE hora_atual INT;

    SET hora_atual = HOUR(CURTIME());

    IF hora_atual < 12 THEN
        SET saudacao = 'Bom Dia!';
    ELSEIF hora_atual < 18 THEN
        SET saudacao = 'Boa Tarde!';
    ELSE
        SET saudacao = 'Boa Noite!';
    END IF;

    SELECT saudacao AS Mensagem;
END //

DELIMITER ;

CALL SaudacaoPersonalizada();
```

15

Gatilhos (Triggers)

Triggers

Triggers são procedimentos armazenados que são automaticamente executados ou "disparados" em resposta a certos eventos em uma tabela, como inserções, atualizações ou exclusões.

Exemplo: Criar um gatilho que atualiza o total de exemplares disponíveis quando um novo livro é inserido.

TRIGGER

```
CREATE TRIGGER AtualizarEstoque
AFTER INSERT ON Livros
FOR EACH ROW
BEGIN
    UPDATE Estoque
    SET Quantidade = Quantidade + 1
    WHERE LivroID = NEW.LivroID;
END;
```

16

Comandos DCL

Comandos DCL

Os comandos DCL (Data Control Language) são usados para controlar o acesso aos dados no banco de dados. Eles permitem que você gerencie permissões e segurança, garantindo que apenas usuários autorizados possam executar determinadas operações.

Principais Comandos DCL

- GRANT: Concede permissões a um usuário;
- REVOKE: Revoga permissões de um usuário.

Permissão	Descrição
SELECT	Permite ler dados de uma tabela
INSERT	Permite inserir novos dados em uma tabela
UPDATE	Permite atualizar dados existentes
DELETE	Permite excluir dados de uma tabela
CREATE	Permite criar novos objetos no banco
DROP	Permite excluir objetos do banco
GRANT	Permite conceder permissões a outros usuários
REVOKE	Permite revogar permissões de outros usuários
ALL	Concede todas as permissões possíveis

Comandos DCL

Exemplo:

PERMISSÕES

```sql
-- Conceder permissão de SELECT
GRANT SELECT ON Livros TO 'joao'@'localhost';

-- Revogar permissão de SELECT
REVOKE SELECT ON Livros FROM 'joao'@'localhost';

-- Concedendo Autorização Total para uma Procedure
GRANT EXECUTE ON PROCEDURE AtualizarEstoque TO 'maria'@'localhost';
```

17

Comandos TCL

Comandos TCL

Os comandos TCL (Transaction Control Language) são usados para gerenciar transações no banco de dados, garantindo a integridade e consistência dos dados.

Principais Comandos TCL

- BEGIN (ou START TRANSACTION): Inicia uma transação.
- COMMIT: Confirma todas as operações realizadas durante a transação.
- ROLLBACK: Desfaz todas as operações realizadas durante a transação.
- SAVEPOINT: Cria um ponto de salvamento dentro de uma transação.
- RELEASE SAVEPOINT: Remove um ponto de salvamento previamente definido.
- ROLLBACK TO SAVEPOINT: Reverte a transação para um ponto de salvamento específico.

Comandos TCL

Exemplo:

COMANDOS TCL

```sql
START TRANSACTION;

-- Inserir um novo livro
INSERT INTO Livros (Titulo, AutorID, AnoPublicacao) VALUES ('Novo Livro', 1, 2024);
SAVEPOINT PontoA;

-- Atualizar o título de um livro
UPDATE Livros SET Titulo = 'Livro Atualizado' WHERE LivroID = 1;

-- Reverter para o ponto de salvamento PontoA
ROLLBACK TO SAVEPOINT PontoA;

-- Confirmar a transação
COMMIT;
```

18

Conclusão

Conclusão

Parabéns por chegar até aqui! Você agora possui uma sólida compreensão dos conceitos e comandos fundamentais do SQL no contexto do MySQL. Desde a criação e manipulação de bancos de dados até a execução de consultas complexas e o uso de procedimentos armazenados e triggers, você está preparado para aplicar esses conhecimentos em projetos reais. Continue praticando e explorando o mundo do SQL – sua jornada de aprendizado está apenas começando!