

UM GUIA INFINITY PARA INICIANTES CONCEITOS GERAIS DE BANCO DE DADOS

IN INFINITY SCHOOL

Infinity School

CONCEITOS GERAIS DE BANCO DE DADOS

um guia Infinity para iniciantes

Reality Editora

2024

Sumário

Apresentação

03

Introdução

04

Capítulo 1

Criação de um banco no MySQL

07

Capítulo 2

Trabalhando com tabelas no MySQL

12

Capítulo 3

Manipulação de dados das tabelas em SQL

20

Capítulo 4

Consultas em SQL

28

Conclusão

38

Apresentação

Bem-vindo ao livro **Conceitos gerais de banco de dados: um guia Infinity para iniciantes**. Neste livro, você será introduzido ao mundo do SQL e MySQL, desde os conceitos básicos até técnicas intermediárias. Seja você um iniciante absoluto ou alguém com alguma experiência em programação, este livro foi criado para ajudá-lo a compreender e dominar o uso do SQL e MySQL em seus projetos.

Ao longo deste guia, você aprenderá como criar e manipular bancos de dados, escrever consultas poderosas para extrair dados, aplicar constraints para garantir a integridade dos dados e muito mais. Cada conceito é apresentado de forma clara e acessível, com exemplos práticos para ajudá-lo a entender melhor.

Espero que este livro seja uma ferramenta valiosa em sua jornada de aprendizado de SQL e MySQL. Vamos começar essa jornada juntos!

*Inicie sua jornada de aprendizado em SQL
conosco e descubra um universo de possibilidades no campo dos bancos de dados e gerenciamento de dados.*



Introdução

MySQL é um dos sistemas de gerenciamento de banco de dados relacional (RDBMS) mais populares do mundo. Desenvolvido inicialmente por Michael Widenius e David Axmark em 1995, o MySQL ganhou destaque por sua robustez, confiabilidade e facilidade de uso. Atualmente, é amplamente utilizado em ambientes de desenvolvimento e produção, suportando desde pequenas aplicações até grandes portais web, que exigem um gerenciamento eficiente de grandes volumes de dados.

A importância do MySQL no desenvolvimento de aplicações modernas é inegável. Ele oferece uma solução eficaz para o armazenamento e a recuperação de dados, sendo uma escolha comum para o backend de sites, sistemas de e-commerce e aplicativos de dados. Empresas de todos os tamanhos confiam no MySQL para processar milhões de transações diariamente, o que demonstra sua capacidade de escalar conforme a necessidade dos negócios.

SQL, ou Linguagem de Consulta Estruturada (Structured Query Language), é a linguagem usada para gerenciar e manipular bancos de dados relacionais. Desde sua criação nos anos 1970, SQL tornou-se o padrão de fato para a interação com bancos de dados relacionais, sendo adotado por praticamente todos os RDBMS modernos, como Oracle, PostgreSQL, SQL Server e, claro, MySQL. No contexto do MySQL, o SQL é usado para realizar todas as operações de banco de dados, desde a criação de tabelas e inserção de dados até a realização de consultas complexas.

SELECT, INSERT, UPDATE

Comandos SQL como **SELECT**, **INSERT** e **DELETE** são fundamentais para o trabalho com MySQL, permitindo que desenvolvedores manipulem dados de maneira eficaz e eficiente.

Entender o SQL é essencial para qualquer pessoa que deseja trabalhar com tecnologias de banco de dados. A combinação da robustez do MySQL com a universalidade do SQL oferece uma plataforma poderosa para desenvolver aplicações resilientes e de alto desempenho. No próximo capítulo, exploraremos como instalar o MySQL e começar a usar essa ferramenta incrível.



Instalando o MySQL em diferentes sistemas operacionais

Introdução

Instalação no Windows

1. Download

- Visite o site oficial do MySQL (mysql.com) e navegue até a seção de downloads (<https://dev.mysql.com/downloads/installer/>).
- Selecione a versão do Windows e baixe o instalador.

Windows (x86, 32-bit), MSI Installer (mysql-installer-community-8.0.37.0.msi)	8.0.37	296.1M	Download
MD5: ae605e4f62aaf8bb1adef684d62a49f2 Signature			

2. Executando o instalador

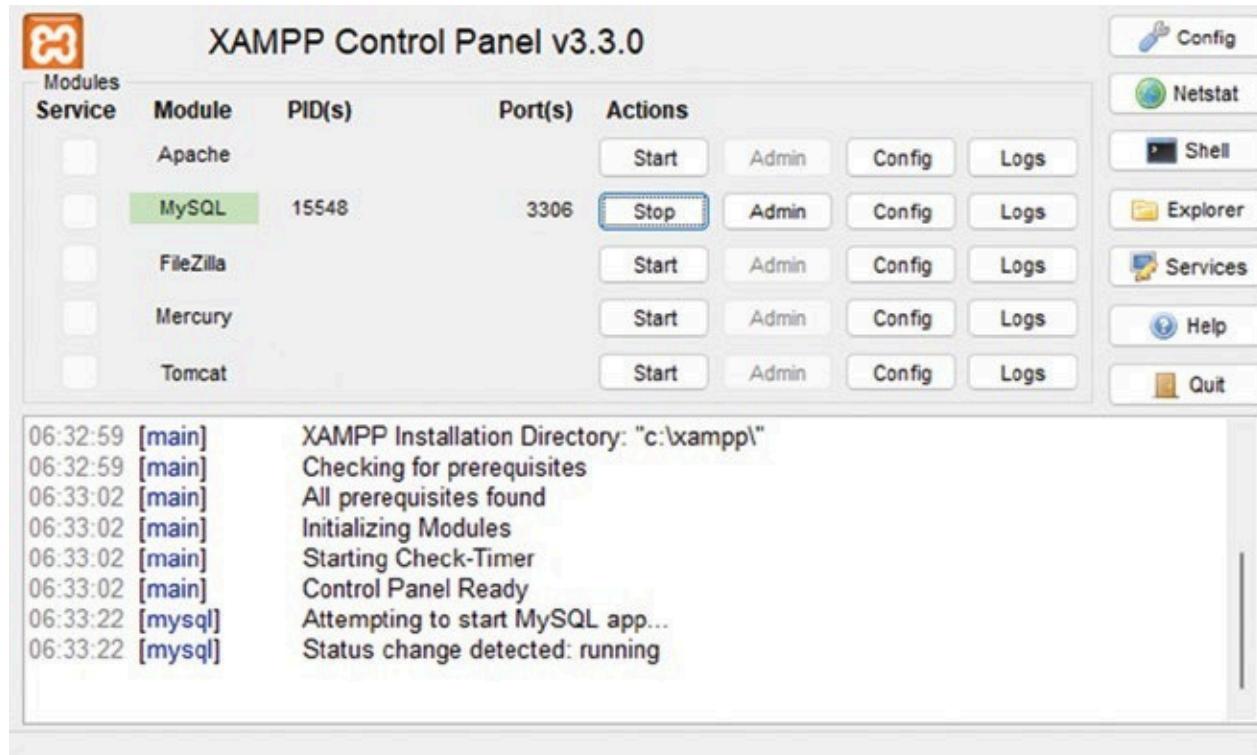
- Execute o arquivo baixado para iniciar o instalador.
- Opte pelo tipo de instalação que preferir. Recomendamos “Full” para iniciantes, pois inclui todos os componentes necessários, tanto o servidor quanto o workbench.
- Siga as instruções na tela para concluir a instalação.

3. Configuração

- Após a instalação, o assistente de configuração será iniciado.
- Configure o servidor MySQL, incluindo a definição de uma senha para o usuário root, que é o superusuário do MySQL.
- Complete o processo de configuração seguindo as instruções na tela.

4. Alternativa XAMPP

- Caso você tenha algum problema com o MySQL Server, você pode também optar por instalar o pacote de servidores XAMPP.
- O processo é simples: vá até o site (https://www.apachefriends.org/pt_br/index.html) e navegue até o download do sistema operacional que você estiver usando.
- Execute o instalador e siga as instruções para finalizar as configurações.



Clique em “Start” na opção de MySQL, e está feito: você tem um servidor de MySQL rodando na sua máquina.

Capítulo 1

Criação de um banco no MySQL

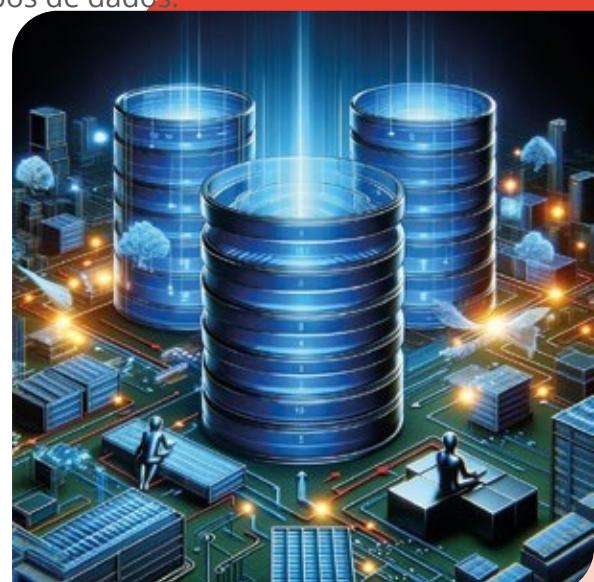
1.1 Introdução à criação de banco de dados

Antes de mergulharmos na criação de tabelas, A criação de um banco de dados é o fundo consultas e outras operações com o MySQL, mento sobre o qual todas as outras funções o primeiro passo essencial é aprender como do banco de dados são construídas. Ele serve criar um banco de dados. Um banco de dados como o principal contêiner para dados e sem é um repositório organizado de dados que ele, não seria possível realizar tarefas como permite o armazenamento, a manipulação inserir, atualizar ou buscar dados. e a recuperação de informações de maneira eficiente. No MySQL, cada banco de dados pode conter várias tabelas, cada uma representando diferentes tipos de dados.

1.2 Passo a passo para a criação de um banco de dados

1.2.1 Conecte-se ao MySQL Primeiramente, é necessário conectar-se ao servidor MySQL. Isso geralmente é feito através de uma linha de comando ou uma interface gráfica como o MySQL Workbench.

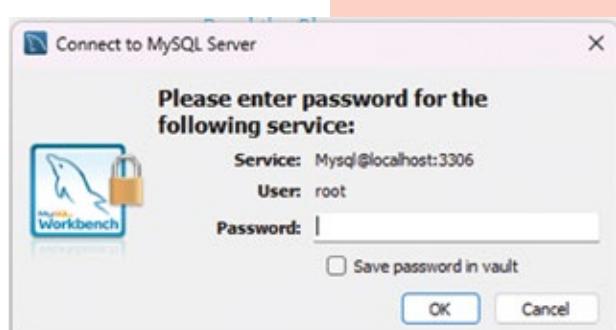
Clique no local.



MySQL Connections + ○

Local instance MySQL80

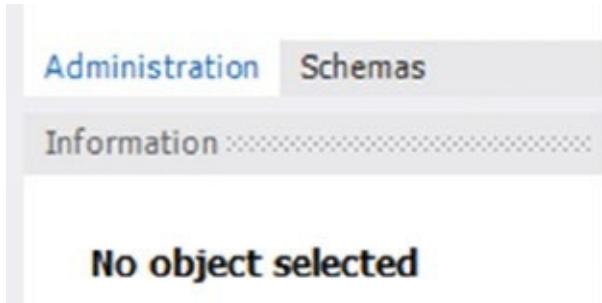
root
localhost:3306



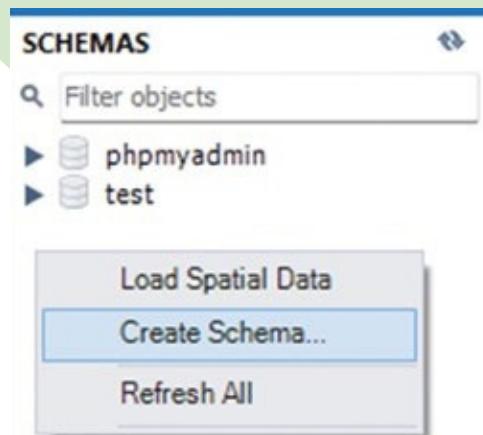
Insira sua senha, caso tenha configurado uma.

1.2.2 Crie o banco de dados Uma vez conectado, você pode criar um banco de dados utilizando o comando SQL.

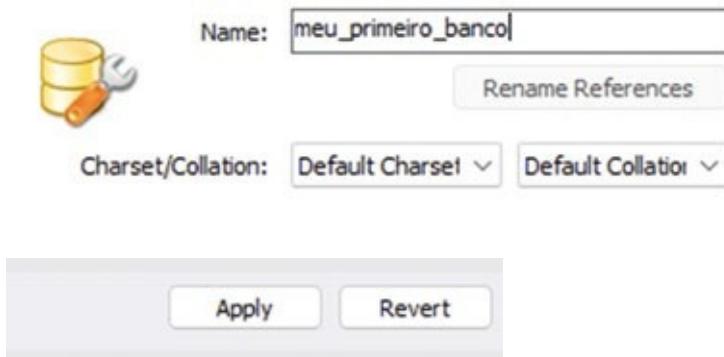
Aqui está um exemplo de como criar um banco de dados chamado "meu_primeiro_banco":



Com o botão direito em qualquer espaço vazio dos Schemas, clique em "Create Schema..."



Coloque o nome do seu banco, depois clique em "Apply".



Uma nova janela será aberta.

Clique em "Apply" nela também e depois clique em "Finish", então as janelas serão fechadas e você terá criado seu primeiro banco através da IDE do MySQL Workbench.

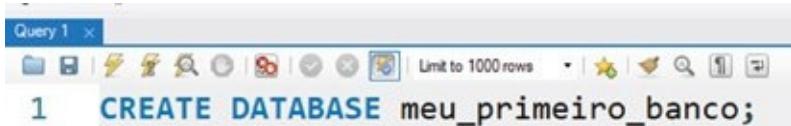
Review the SQL Script to be Applied on the Database

```
1 CREATE SCHEMA `meu_primeiro_banco` ;
2
```

Back Apply Cancel



Você também pode optar por criar seu primeiro banco através das linhas de comando usando a linguagem SQL para isso. Ao invés de fazer o passo a passo anterior, você vai abrir o cursor e digitar:



```
Query 1 ×
CREATE DATABASE meu_primeiro_banco;
```

Depois deve clicar no ícone de raio, no qual você pode clicar no primeiro raio (terceiro ícone), que executará todas as linhas de SQL que você tiver escrito no arquivo Query, ou optar por clicar no segundo raio, que por sua vez só executa a linha em que o cursor se encontra.



SCHEMAS

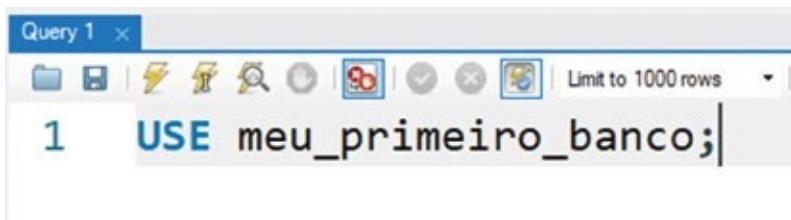
Filter objects

- ▶ **meu_primeiro_banco**
- ▶ **phpmyadmin**
- ▶ **test**

Caso seu banco não apareça na aba de Schemas, clique no botão de “refresh”, no canto superior direito.

1.2.3 Selecionar o banco de dados para uso

Após a criação do banco de dados, você precisa selecioná-lo para começar a trabalhar com ele:



```
Query 1 ×
USE meu_primeiro_banco;
```

Após clicar no raio com esse comando, sua aba de Schemas ficará assim:



SCHEMAS

Filter objects

- ▼ **meu_primeiro_banco**
 - Tables
 - Views
 - Stored Procedures
 - Functions
- ▶ **phpmyadmin**
- ▶ **test**

Infinity Quiz | Teste seus conhecimentos

1 • Qual comando SQL é usado para criar um novo banco de dados no MySQL?

- a) CREATE NEW DATABASE
- b) NEW DATABASE
- c) CREATE DATABASE
- d) MAKE DATABASE

2 • Caso o MySQL Server não esteja funcionando ou você tenha tido problema para configurá-lo, qual outro programa pode substituí-lo?

- a) Command Server
- b) XAMPP
- c) SQL Command
- d) CRAMPP

3 • Após criar um banco de dados no MySQL, qual comando você deve usar para começar a usá-lo?

- a) SELECT DATABASE
- b) USE
- c) CONNECT
- d) ACCESS

4 • Qual ferramenta gráfica é comumente usada para gerenciar bancos de dados MySQL?

- a) MySQL Command Center
- b) MySQL Workbench
- c) SQL Manager
- d) DataDes

5 • Além da linha de comando SQL, de qual outra forma se pode criar um banco de dados com MySQL Workbench?

- a) Botão “Criar banco”
- b) Navegar até a aba de Bancos e clicar em “Novo banco”
- c) Navegar até a aba Schemas, clicar com o botão direito e clicar em “Create Schema...”
- d) Não existe uma forma de criar sem ser por linha de comando





Respostas Infinity Quiz:

1 – C; 2 – B; 3 – B; 4 – B; 5 – C.

Capítulo 2

Trabalhando com tabelas no MySQL

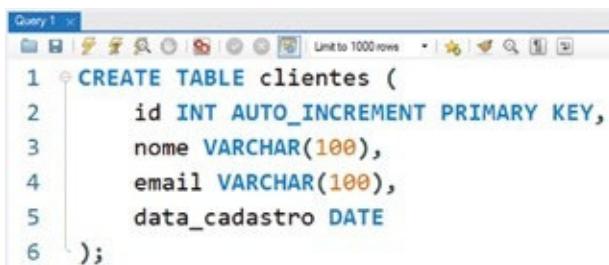
2.1 Introdução a tabelas

No MySQL, as tabelas são a estrutura fundamental na qual os dados são armazenados. Cada tabela é composta por colunas e linhas, de modo que as colunas representam os campos de dados e as linhas representam os registros. Este capítulo explorará como criar, modificar e interagir com tabelas no MySQL, proporcionando uma compreensão sólida das operações básicas necessárias para manipular dados eficazmente.

2.2 Criando tabelas

Definindo estruturas de tabela

- Antes de criar uma tabela, é crucial definir sua estrutura, incluindo o nome da tabela e a definição de cada coluna. Uma coluna é definida para uma lista de tipos de dados que você pode especificar outras propriedades, como da sua tabela: chaves primárias, valores-padrão e restrições.
- Para criar uma tabela, é necessário o comando SQL `CREATE TABLE` seguido do nome da tabela que você quer criar, seguido de parênteses com as colunas daquela tabela separadas por vírgula.
- Exemplo de criação de tabela:



```
Query 1
CREATE TABLE clientes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100),
    email VARCHAR(100),
    data_cadastro DATE
);
```

2.3 Tipos de dados

Escolher o tipo de dado correto para cada

- **INT:** Armazena números inteiros (sem parte decimal). Utilizado para dados numéricos que não requerem frações.
- **VARCHAR:** Tipo de dado de string variável. Usado para texto ou combinações de texto e números. A capacidade de armazenamento pode ser especificada – `VARCHAR(255)` para até 255 caracteres.
- **NVARCHAR:** Armazena texto Unicode. Permite armazenar caracteres de qualquer idioma, ideal para dados que requerem suporte multilíngue.

- **TEXT:** Para grandes quantidades de texto. Existem diferentes subtipos de **TEXT**, como **TINYTEXT**, **TEXT**, **MEDIUMTEXT** e **LONGTEXT**, variando em capacidade máxima de armazenamento.
- **DATE:** Armazena datas no formato YYYY-MM-DD. Útil para armazenar datas específicas sem horário.
- **DATETIME:** Armazena data e hora no formato YYYY-MM-DD HH:MM:SS. Ideal para registrar quando eventos exatos ocorreram.
- **TIMESTAMP:** Semelhante a **DATETIME**, mas comumente usado para rastrear mudanças em registros. Pode ser configurado para atualizar automaticamente quando a linha é modificada.
- **TIME:** Armazena horas no formato HH:MM:SS. Usado para diferenciar apenas o tempo, sem data.
- **YEAR:** Armazena um ano, no formato YYYY ou YY. Útil para dados que envolvem somente o ano.
- **CHAR:** Um tipo de dado de string de comprimento fixo – por exemplo, **CHAR(5)** sempre reserva espaço para 5 caracteres. Útil quando o tamanho do campo é constante, como códigos postais ou abreviações de estados.
- **BLOB:** Para armazenar dados binários, como imagens, arquivos de áudio ou outros tipos de multimídia. Existem subtipos, como **TINYBLOB**, **BLOB**, **MEDIUMBLOB** e **LONGBLOB**, diferenciados pela quantidade máxima de dados que podem armazenar.
- **FLOAT** e **DOUBLE**: Números com frações, usados para representar valores com casas decimais. **FLOAT** é para precisão simples, e **DOUBLE** é para precisão dupla.
- **DECIMAL:** Para valores decimais exatos. Útil para dados financeiros que requerem uma grande precisão, como moedas.
- **ENUM:** Permite que uma coluna seja restrita a um dos valores predefinidos. Útil para colunas que contêm um número limitado de variantes (como 'sim' ou 'não').

- **SET**:
Semelhante a **ENUM**, mas permite múltiplas escolhas ao mesmo tempo.
Pode armazenar um conjunto de valores definidos.

2.4 Autoincremento

- Em SQL, principalmente em sistemas de gerenciamento de banco de dados relacional, como MySQL, SQL Server, PostgreSQL e SQLite, o conceito de **AUTOINCREMENT** é específico para a definição de colunas que possuem valores incrementados automaticamente para cada novo registro inserido na tabela.
- No MySQL e no SQLite, o **AUTOINCREMENT** é especificado ao definir uma coluna **INTEGER** ou **INT**. Quando como chave primária (geralmente usando o tipo de dados uma nova linha for inserida na tabela, o valor dessa coluna será automaticamente incrementado em uma unidade em relação ao valor máximo atualmente existente nessa coluna.

2.5 Restrições SQL

- As constraints são regras aplicadas às colunas de uma tabela para garantir a integridade dos dados e impor restrições sobre o que pode ser armazenado nessas colunas. Elas ajudam a manter a consistência e a qualidade dos dados em um banco de dados MySQL. Existem diferentes tipos de constraints que podem ser aplicadas em colunas, incluindo restrições de chave primária, chave estrangeira, **NOT NULL**, e de verificação.

2.5.1 NOT NULL

- A restrição **NOT NULL** garante que um valor não seja nulo (ou seja, vazio) em uma coluna.
- Ela impede a inserção de valores nulos em uma coluna específica, garantindo que cada registro tenha um valor válido.
- Você pode adicionar a restrição **NOT NULL** durante a criação da tabela ou após a criação da tabela usando **ALTER TABLE**.

Uma chave primária é uma coluna (ou conjunto de colunas) que identifica de forma única cada registro em uma tabela.

- Ela garante que não haja duplicatas e que cada linha na tabela seja acessível por meio de um valor exclusivo.
- Para adicionar uma chave primária a uma coluna, você usa a cláusula **PRIMARY KEY** durante a criação da tabela ou após a criação da tabela **ALTER TABLE**.

2.5.3 Chave estrangeira (foreign key) • Uma chave estrangeira é uma coluna (ou conjunto de colunas) que estabelece uma relação entre duas tabelas.

- Ela permite que você mantenha a integridade referencial, garantindo que os valores em uma coluna correspondam aos valores em outra coluna em uma tabela relacionada.
- Para adicionar uma chave estrangeira, você usa a cláusula

FOREIGN KEY

durante a criação da tabela ou após a criação da tabela usando `ALTER TABLE`.

2.5.4 Restrição única (unique constraint)

- Uma restrição única garante que todos os valores em uma coluna (ou conjunto de colunas) sejam exclusivos, exceto valores nulos.
- Ela impede a inserção de valores duplicados em uma coluna específica.
- Você pode adicionar uma restrição única usando a cláusula

UNIQUE durante

a criação da tabela ou após a criação da tabela usando `ALTER TABLE`.

2.5.5 Restrição de verificação (check constraint)

- Uma restrição de verificação permite que você defina uma condição para os valores que podem ser inseridos em uma coluna.
- Ela garante que os valores inseridos atendam a uma condição específica.
- Você pode adicionar uma restrição de verificação usando a cláusula

CHECK

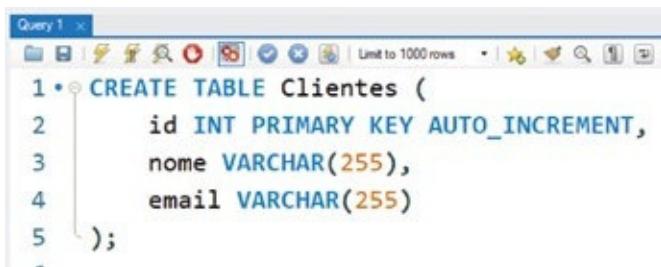
durante a criação da tabela ou após a criação da tabela usando `ALTER TABLE`.

Ao aplicar essas restrições às colunas de suas tabelas, você pode garantir a consistência e a integridade dos dados armazenados em seu banco de dados MySQL. Isso é essencial para garantir a confiabilidade e a precisão das informações mantidas pelo seu sistema.

2.6 Tabelas com chave estrangeira

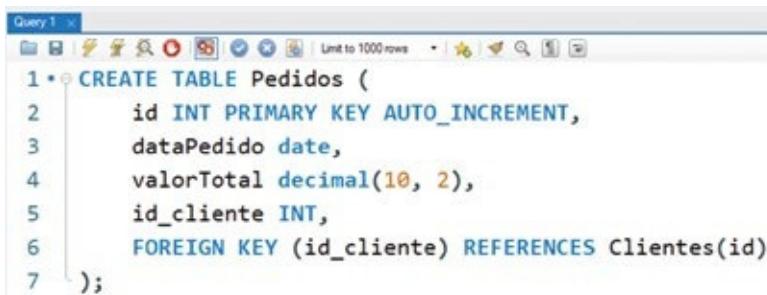
Chaves estrangeiras são essenciais para relacionar tabelas de maneira que as informações sejam consistentes e o banco de dados permaneça normalizado. Por exemplo, em um sistema que registra informações sobre clientes e seus respectivos pedidos, é útil manter essas informações em tabelas separadas, mas relacionadas através de chaves estrangeiras.

Tabela Clientes:



```
Query 1 ×
CREATE TABLE Clientes (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(255),
    email VARCHAR(255)
);
```

Tabela Pedidos:



```
Query 1 ×
CREATE TABLE Pedidos (
    id INT PRIMARY KEY AUTO_INCREMENT,
    dataPedido date,
    valorTotal decimal(10, 2),
    id_cliente INT,
    FOREIGN KEY (id_cliente) REFERENCES Clientes(id)
);
```

Esta tabela registra os pedidos feitos pelos clientes. A coluna `id_cliente` serve como chave estrangeira e referencia a `id` na tabela Clientes.

Explicação dos códigos

- Tabela Clientes:
 - `id`: chave primária da tabela, identifica unicamente cada cliente.
 - `nome` e `email`: armazenam o nome e o email do cliente, respectivamente.
- Tabela Pedidos:
 - `id`: chave primária da tabela, identifica unicamente cada pedido.
 - `id_cliente`: chave estrangeira que faz a ligação com a tabela Clientes, indicando qual cliente fez o pedido.
 - `dataPedido` e `valorTotal`: armazenam a data do pedido e o valor total, respectivamente.

Benefícios do uso de chave estrangeira

- Integridade referencial: assegura que relações entre tabelas sejam consistentes. Por exemplo, não se pode inserir um pedido para um cliente que não existe.
- Manutenção da informação: atualizações e exclusões são mais seguras e consistentes, pois o SQL pode automaticamente verificar alterações nas tabelas relacionadas para evitar dados órfãos.

2.7 Modificando tabelas

- À medida que suas aplicações e bancos de dados evoluem, você pode precisar fazer alterações nas estruturas das tabelas existentes. No MySQL, essas modificações podem incluir adicionar ou remover colunas, modificar tipos de dados de colunas existentes ou alterar outros aspectos da tabela.

- **Adicionando colunas:** para adicionar uma nova coluna a uma tabela existente, você utiliza o comando **ALTER TABLE** com a cláusula **ADD COLUMN**. Isso permite que você especifique o nome e o tipo da nova coluna, além de outras propriedades, como restrições e valores-padrão.



```
Query 1 ×
1 • ALTER TABLE clientes ADD COLUMN endereco VARCHAR(255);
2
```

Este comando adiciona uma nova coluna chamada “endereco” à tabela “clientes”.

- **Removendo colunas:** se uma coluna não é mais necessária, você pode removê-la da tabela com o comando **ALTER TABLE** combinado com a cláusula **DROP COLUMN**. Note que essa operação é irreversível e os dados na coluna serão perdidos.



```
Query 1 ×
1 • ALTER TABLE clientes DROP COLUMN endereco;
```

Este comando remove a coluna “endereco” na tabela “clientes”. • **Modificando colunas:**

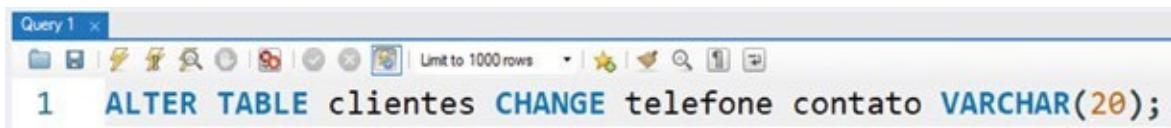
modificar uma coluna pode envolver mudanças no tipo de dado, no tamanho do campo ou em outras propriedades. O comando **ALTER TABLE** com a cláusula **MODIFY COLUMN** é usado para esses fins.



```
Query 1 ×
1 ALTER TABLE clientes MODIFY COLUMN telefone VARCHAR(20);
```

Este comando altera a coluna “telefone” na tabela “clientes” para ter um máximo de 20 caracteres.

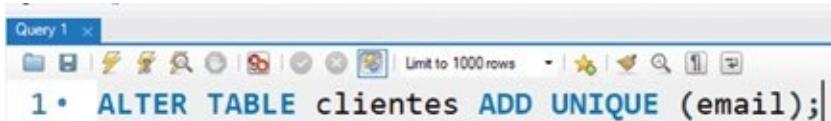
- **Renomeando colunas:** para renomear uma coluna em uma tabela **ALTER TABLE** com a cláusula **CHANGE**. No MySQL, você utiliza o comando **CHANGE**. Este comando permite não apenas renomear a coluna, mas também modificar o tipo de dados, se necessário.



```
Query 1 ×
1 ALTER TABLE clientes CHANGE telefone contato VARCHAR(20);
```

Este comando renomeia a coluna “telefone” para “contato” na tabela “clientes” e confirma que “contato” deve ser do tipo **VARCHAR(20)**.

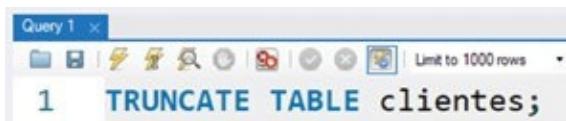
- **Adicionando restrições:** restrições são regras aplicadas às colunas de uma tabela para garantir a validade e integridade dos dados. Por exemplo, você pode querer garantir que todos os emails sejam únicos em uma tabela de clientes.



```
Query 1
1 • ALTER TABLE clientes ADD UNIQUE (email);
```

Este comando adiciona uma restrição de unicidade para a coluna “email” na tabela “clientes”.

- **Limpando uma tabela:** se você deseja remover todos os dados de uma tabela, mas manter a estrutura da tabela para uso futuro, você pode usar o comando **TRUNCATE TABLE**.



```
Query 1
1 TRUNCATE TABLE clientes;
```

Este comando remove todos os registros da tabela “clientes” de maneira rápida e eficiente. **TRUNCATE TABLE** é mais rápido que o **DELETE** porque não gera dados para operações de rollback e não ativa triggers.

- **Deletando uma tabela:** caso você queira remover completamente uma tabela, incluindo sua estrutura e dados, o comando **DROP TABLE** é utilizado.



```
Query 1
1 DROP TABLE clientes;
```

Este comando deleta a tabela “clientes” inteiramente do banco de dados. É uma operação irreversível e deve ser usada com cautela, pois todos os dados e a estrutura da tabela são permanentemente removidos.

Mãos ao código | Hora de praticar

1 • Criação de tabela de funcionários

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare uma variável para guardar um número inteiro. Na sequência, verifique se o número guardado na variável é maior do que 5. Caso verdadeiro, exiba a mensagem: “O número é maior do que 5”. Se não, informe: “O número não é maior do que 5”.

2 • Adição de coluna de data de contratação

Adicione uma nova coluna chamada “data_contratacao” (tipo **DATE**) à tabela “funcionarios”.

3 • Modificação do tipo de dado de uma coluna

| Altere o tipo de dado da coluna “salario” na tabela “funcionarios” para **DOUBLE**.

4 • Remoção de uma coluna

| Remova a coluna “cargo” da tabela “funcionarios”.

5 • Renomeação uma coluna

| Renomeie a coluna “nome” para “nome_do_funcionario” sem modificar o tipo de dados dela.

Infinity Quiz | Teste seus conhecimentos

1 • Qual comando é usado para criar uma nova tabela no MySQL?

- a) CREATE NEW TABLE
- b) NEW TABLE
- c) CREATE TABLE
- d) MAKE TABLE

2 • Qual tipo de dado você usaria para armazenar um endereço de email no MySQL?

- a) MAKE
- b) VARCHAR
- c) CHAR
- d) STRING

3 • Se você precisar mudar o nome de uma coluna na tabela “clientes” de “telefone” para “contato”, qual comando você usaria?

- a) ALTER TABLE clientes RENAME MODIFY telefone TO contato
- b) ALTER TABLE clientes CHANGE telefone contato VARCHAR(20)
- c) ALTER TABLE clientes MODIFY COLUMN telefone TO contato
- d) UPDATE TABLE clientes RENAME telefone TO contato

4 • Qual comando remove todos os dados de uma tabela, mas não a própria tabela?

- a) DELETE TABLE clientes
- b) DROP TABLE clientes
- c) TRUNCATE TABLE clientes
- d) REMOVE AL FROM clientes

5 • Qual comando descarta uma tabela existente?

- a) DELETE FROM clientes
- b) DROP TABLE clientes
- c) TRUNCATE TABLE clientes
- d) CLEAR TABLE clientes

Respostas Infinity Quiz:

1 – C; 2 – B; 3 – B; 4 – C; 5 – B.

Capítulo 3

Manipulação de dados das tabelas em SQL

3.1 Introdução

A manipulação de dados em tabelas é uma parte crucial do trabalho com bancos **INSERT**, **UPDATE** e **DELETE**,

de dados relacionais. Por meio dos comandos de os desenvolvedores e administradores de bancos de dados têm o poder de adicionar, modificar e excluir registros dentro de uma estrutura tabular. Esses comandos não apenas permitem a gestão eficiente dos dados, mas também garantem a integridade e a precisão das informações armazenadas.

3.2 Insert

O comando

INSERT é crucial em qualquer sistema de gerenciamento de banco de dados (SGBD), permitindo adicionar novos registros às tabelas. A sintaxe básica foi mencionada anteriormente, mas o comando **INSERT** pode ser expandido de várias formas para atender a diferentes necessidades.

Adicionando campos na nossa tabela “clientes”:



```
Query 1
1 INSERT INTO Clientes
2 VALUES (DEFAULT, 'Abel Jr', 'abel.jr@email.com');
```

Nessa primeira abordagem, usamos o comando SQL **INSERT INTO**, colocamos o nome da nossa tabela “clientes”, depois o comando **VALUES**, sem digitarmos em ordem os valores respectivos as colunas da tabela. Nessa tabela, por exemplo, só existem três campos: **id**, **nome** e **email**; como o campo **AUTO_INCREMENT** não precisamos passar um número para ele, pois ele será autoincrementado. Em ordem passamos o nome e o email do cliente.

Outra abordagem que podemos usar é:



```
Query 1
1 INSERT INTO Clientes (nome,email)
2 VALUES ('Abel Jr', 'abel.jr@email.com');
```

Nessa nova abordagem, adicionamos entre parênteses, logo após o nome da nossa tabela, o nome das nossas colunas. Quando se faz isso, você pode passar como valor apenas os valores para as colunas que passou, de forma que não precisamos passar o `DEFAULT` para o `id`, pois, como ele é `AUTO_INCREMENT` e nós o omitimos, já subentende-se que é o padrão.

Com essa abordagem, você pode também deixar algum valor em branco, por exemplo:

```
Query 1
1 INSERT INTO Clientes (email)
2 VALUES ('abel.jr@email.com');
```

Dessa forma, adicionamos um cliente sem nome, apenas com email e o seu `id` (que está sendo autoincrementado). Um detalhe muito importante é que essa abordagem só funcionaria se a coluna “`nome`” não tivesse a restrição `NOT NULL`.

Inserindo dados em múltiplas linhas

Você pode inserir dados em várias linhas

de uma só vez, o que é útil para otimizar as operações de inserção e reduzir o número de transações de banco de dados necessárias. Vamos supor que temos uma tabela com três campos: `nome`, `email` e `data_cadastro`.

Exemplo:

```
Query 1
1 CREATE TABLE clientes(
2     id INT PRIMARY KEY AUTO_INCREMENT,
3     nome VARCHAR(255),
4     email VARCHAR(255),
5     dataCadastro DATE
6 );
```

Vamos adicionar nela vários clientes ao mesmo tempo:

```
Query 1
1 INSERT INTO clientes (nome,email,data_cadastro)
2 VALUES ('Abel', 'abel.jr@email.com', '2024-05-07'),
3 ('João', 'joao@email.com', '2019-05-07'),
4 ('Maria', 'maria@email.com', '2015-05-07'),
5 ('Ana', 'ana@email.com', '2010-05-07');
```

Dessa forma, adicionamos em um mesmo comando 4 clientes ao mesmo tempo, separando por vírgula.

Inserindo dados de outra tabela

Você pode usar **INSERT** para adicionar dados que são selecionados de outra tabela. Isso é útil para casos como a duplicação de registros ou a migração de dados entre tabelas.

Vamos criar uma segunda tabela de clientes que foram arquivados, com a seguinte estrutura:

```
Query 1 ×
CREATE TABLE clientes_arquivados(
    id INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(255),
    email VARCHAR(255)
);
```

Agora vamos inserir dados nessa tabela, porém usando dados que já existem na nossa tabela de cliente.

Exemplo:

```
Query 1 ×
INSERT INTO clientes_arquivados (id, nome, email)
SELECT id, nome, email
FROM clientes WHERE dataCadastro < '2017-01-01';
```

Nesse exemplo, utilizamos uma combinação poderosa de comandos SQL para transferir dados de forma seletiva entre tabelas. O comando **SELECT** é essencial aqui; ele atua como um explorador que vasculha e recupera informações específicas de um banco de dados. No nosso caso, ele está configurado para buscar as colunas id, nome e email da tabela “clientes”.

A cláusula **FROM** especifica a origem exata dos dados, direcionando o **SELECT** para a tabela correta. É aqui que a magia começa, mas não termina. A verdadeira precisão vem com o comando **WHERE**, o filtro que nos permite refinar ainda mais nossa busca. Nesse comando, especificamos que apenas os registros com “data_cadastro” anterior a 1º de janeiro de 2017 devem ser considerados.

Com esses dados selecionados, o comando **INSERT INTO** entra em cena. Esse comando pega os dados filtrados e os insere na tabela “clientes_arquivados”. Essa abordagem não apenas preserva a integridade e a relevância dos dados em nossa base, mas também nos ajuda a manter um histórico organizado de clientes, separando aqueles que se cadastraram antes do início de 2017.

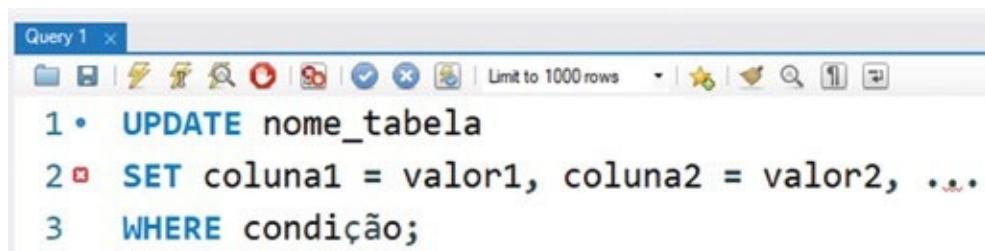
Esse método é um exemplo excelente de como comandos SQL podem trabalhar juntos harmoniosamente para realizar operações de dados complexas e úteis, mantendo a organização e a eficiência do banco de dados.

3.3 Update

Comando UPDATE: modificando dados existentes

UPDATE é usado para alterar registros existentes. Suas capacidades vão além de simples atualizações, permitindo modificações complexas baseadas em condições específicas.

Alterando campos da nossa tabela de clientes:



```

Query 1 ×
1 • UPDATE nome_tabela
2 □ SET coluna1 = valor1, coluna2 = valor2, ...
3 WHERE condição;
  
```

O comando **UPDATE** é essencial quando precisamos alterar informações já existentes em uma tabela de nosso banco de dados. Esse processo inicia-se com a própria declaração **UPDATE**, que é o sinalizador de que uma ou mais alterações estão prestes a ocorrer.

Logo após o **UPDATE**, especificamos o nome da tabela que contém os dados que queremos modificar. Isso direciona o comando para o local certo no qual a ação será executada.

Segue-se o comando **SET**, que é onde a mágica da atualização realmente acontece. Aqui, você define claramente quais colunas (campos) receberão novos valores e quais serão esses valores. Se você está atualizando mais de uma coluna, basta separar cada atribuição com uma vírgula. Por exemplo:

= 'ana@example.com'.

A cláusula **WHERE** é talvez a parte mais crítica de todo o comando **UPDATE**. Ela funciona como um filtro preciso que delimita quais registros específicos dentro da tabela devem ser atualizados. Sem essa cláusula, uma atualização indiscriminada ocorreria, afetando todos os registros da tabela, o que pode levar a alterações **WHERE** indica que apenas o registro com id igual a 1 será atualizado.

Esse conjunto de componentes forma a estrutura básica de um comando **UPDATE**, permitindo atualizações seguras e precisas dentro de um banco de dados, assegurando que apenas os dados desejados sejam modificados conforme especificado.

Um exemplo um pouco mais prático seria fazer uma alteração na nossa tabela de clientes.

Exemplo:

```
Query 1 ×
| File | New | Open | Save | Print | Find | Search | Help | Exit | Limit

1 • UPDATE clientes
2   SET nome = "Amanda"
3 WHERE id = 3;
```

Nesse exemplo, alterando a tabela que já tínhamos criado anteriormente, ele alterará apenas o nome do usuário com id 3, que seria a "Maria" para o novo nome, "Amanda"

Assim, também podemos fazer atualizações em mais de um campo ao mesmo tempo:

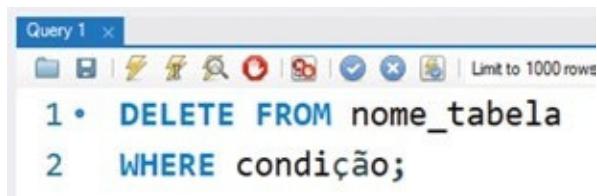
Nessa abordagem, optamos por utilizar o "nome" como critério na cláusula **WHERE**, ao invés do "id". Embora essa técnica funcione, ela não é recomendada. Utilizar o nome pode resultar na alteração do "nome" e "email" de todos os clientes chamados "João", o que pode afetar múltiplos registros indesejadamente. A prática recomendada é sempre usar o "id" na condição **WHERE**. Como é uma chave primária, o "id" é único para cada registro, garantindo que apenas uma ocorrência específica seja modificada. Essa abordagem assegura a precisão e a eficácia das atualizações no banco de dados. Exemplo:

3.4 Delete

Comando DELETE: removendo registros

O comando **DELETE** é usado para excluir registros de uma tabela, e seu uso deve ser feito com cautela para evitar a exclusão acidental de dados importantes.

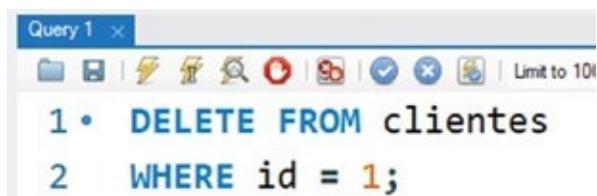
Sintaxe básica



```
Query 1 ×
  ↻ ⌂ ⚡ 🔍 ⏷ ⏵ ⏶ ⏹ ⏸ ⏹ ⏹ | Limit to 1000 rows
1 • DELETE FROM nome_tabela
2 WHERE condição;
```

O primeiro comando é o **DELETE FROM**. Essa cláusula inicia o comando e especifica que a operação a seguir será uma exclusão. O termo **FROM** é seguido pelo nome da tabela da qual os registros serão excluídos.

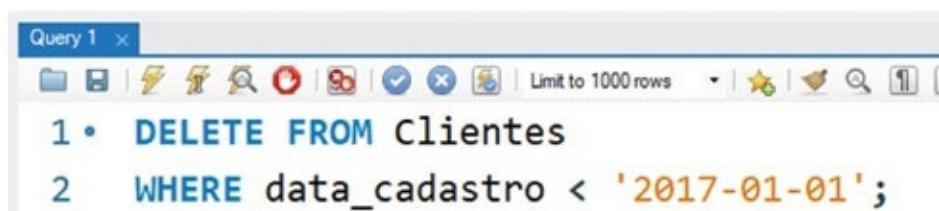
A cláusula **WHERE** é crucial porque define a condição que deve ser satisfeita para que um registro seja excluído. Essa cláusula ajuda a evitar a exclusão de registros não desejados. Se omitida, todos os registros da tabela serão deletados, o que geralmente não é desejado e pode ser muito perigoso. Agora um exemplo do uso dentro da tabela que estamos usando:



```
Query 1 ×
  ↻ ⌂ ⚡ 🔍 ⏷ ⏵ ⏶ ⏹ ⏹ ⏹ | Limit to 100 rows
1 • DELETE FROM clientes
2 WHERE id = 1;
```

Nesse caso, estamos deletando os registros do cliente com `id = 1`, porém você pode fazer condicionais mais específicas, por exemplo: deletar todos os clientes cujos cadastros sejam anteriores a 2017.

Exemplo:



```
Query 1 ×
  ↻ ⌂ ⚡ 🔍 ⏷ ⏵ ⏶ ⏹ ⏹ ⏹ | Limit to 1000 rows
1 • DELETE FROM Clientes
2 WHERE data_cadastro < '2017-01-01';
```

Mãos ao código | Hora de praticar

1 • Inserção simples

Escreva um comando SQL para inserir um novo cliente na tabela “clientes”. O novo cliente deve ter os seguintes detalhes: Nome = ‘Maria Oliveira’, Email = ‘maria.oliveira@exemplo.com’ e Data de Cadastro = ‘2023-05-01’.

2 • Inserção múltipla

Escreva um comando SQL para inserir três novos produtos na tabela “produtos”. Os produtos são: (Nome = ‘Teclado’, Preço = 120.50),
(Nome = ‘Mouse’, Preço = 80.75), (Nome = ‘Monitor’, Preço = 560.00)

3 • Atualização condicional

Escreva um comando SQL para atualizar o email de todos os clientes que têm o nome ‘Carlos Silva’ e Data de Cadastro após ‘2020-01-01’. O novo email deve ser ‘carlos.silva@novodominio.com’.

4 • Exclusão com condição

Escreva um comando SQL para deletar todos os registros da tabela “pedidos” cuja “data_pedido” é anterior a ‘2020-01-01’.

5 • Complexidade na atualização

Escreva um comando SQL para aumentar o preço de todos os produtos em 10% na tabela “produtos” cujo o preço atual seja maior que 100.00.

Infinity Quiz | Teste seus conhecimentos

1 • Qual comando SQL é usado para inserir novos dados em uma tabela?

- a) SELECT
- b) INSERT INTO
- c) UPDATE
- d) DELETE

2 • O que acontece se a cláusula WHERE for omitida em um comando DELETE?

- a) Nenhum registro será deletado.
- b) Apenas o primeiro registro será deletado.
- c) Todos os registros da tabela serão deletados.
- d) O comando será rejeitado como erro.

3 • Qual das seguintes opções é uma forma correta de atualizar o campo email de todos os clientes chamados 'Ana Santos'?

- a) UPDATE Clientes SET email = 'ana.santos@novoemail.com' WHERE nome = 'Ana Santos'
- b) INSERT INTO Clientes (email) VALUES ('ana.santos@novoemail.com') WHERE nome = 'Ana Santos'
- c) DELETE FROM Clientes SET email = 'ana.santos@novoemail.com' WHERE nome = 'Ana Santos'
- d) SELECT * FROM Clientes SET email = 'ana.santos@novoemail.com' WHERE nome = 'Ana Santos'

4 • Você quer inserir um novo cliente na tabela “clientes” com nome e email. Qual dos seguintes comandos está correto, assumindo que o campo “id” é autoincrementável?

- a) INSERT INTO Clientes (nome, email) VALUES ('João Cardoso', 'joao.cardoso@email.com')
- b) INSERT Clientes (nome, email) VALUES ('João Cardoso', 'joao.cardoso@email.com')
- c) ADD TO Clientes (nome, email) VALUES ('João Cardoso', 'joao.cardoso@email.com')
- d) INSERT INTO Clientes VALUES ('João Cardoso', 'joao.cardoso@email.com')

5 • Qual comando remove todos os dados de uma tabela, mas não a própria tabela?

- a) TRUNCATE table Clientes
- b) DELETE Clientes WHERE id = 25
- c) REMOVE FROM Clientes WHERE id = 25
- d) DELETE * FROM Clientes WHERE id = 25

Respostas Infinity Quiz:

1 – B; 2 – C; 3 – A; 4 – A; 5 – A.

Capítulo 4

Consultas em SQL

4.1 O que é uma consulta SQL?

Uma consulta SQL é uma forma de interrogação feita ao banco de dados para extrair ou manipular dados. Essas consultas são realizadas utilizando a linguagem SQL, que permite acessar, filtrar e organizar dados de maneira eficiente e precisa. Consultas SQL são fundamentais para qualquer operação de banco de dados, desde simples recuperações de dados até complexas análises e relatórios.

4.2 Estrutura básica do SELECT

Tabela original: Produtos

ID	Nome	Preço
1	Caneta	1.50
2	Lápis	0.50
3	Borracha	0.75

Consulta SQL:



The screenshot shows a MySQL Workbench interface with a query editor titled "Query 1". The editor contains the following SQL code:

```
1 •  SELECT Nome, Preço FROM Produtos;
```

The number "1" is highlighted in blue, indicating it's a comment or part of a step number. The word "SELECT" is also highlighted in blue.

Explicação:

O comando `SELECT` é utilizado para recuperar todas as entradas das colunas "nome" e "preço" da tabela "Produtos". A cláusula

os dados são extraídos. Esse tipo de consulta é útil para visualizar informações específicas de uma tabela sem modificá-las.

Resposta esperada:

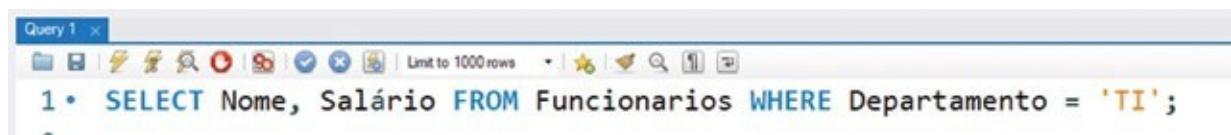
Nome	Preço
Caneta	1.50
Lápis	0.50
Borracha	0.75

4.3 Uso do WHERE

Tabela original: Funcionarios

ID	Nome	Departamento	Salário
1	Ana	Vendas	2500
2	João	TI	3200
3	Maria	Marketing	2900

Consulta SQL:



```
Query 1
1 • SELECT Nome, Salário FROM Funcionarios WHERE Departamento = 'TI';
```

Explicação:

A cláusula **WHERE** é fundamental para adicionar condições específicas às consultas SQL, permitindo filtrar dados com base em critérios definidos. Nesse caso, estamos filtrando para mostrar apenas os funcionários do departamento de TI. Isso permite uma visualização direcionada de subconjuntos de dados, facilitando a análise e o gerenciamento.

Resposta esperada:

Nome	Salário
João	3200

4.4 Uso de LIKE e %

Tabela original: Clientes

ID	Nome	Cidade
1	Ana Beatriz	São Paulo
2	Ana Carolina	Rio de Janeiro
3	João Pedro	Belo Horizonte
4	Carlos Eduardo	São Paulo
5	Pedro Alcântara	Curitiba
6	Joana Carolina	Rio de Janeiro

Consulta SQL:

```
Query 1
1 • SELECT Nome, Cidade FROM Clientes WHERE Nome LIKE 'Ana%';
```

Explicação:

O **LIKE** é combinado com o caractere % para especificar que estamos interessados em todos os registros nos quais o campo “nome” começa com “Ana”, seguido de qualquer sequência de caracteres. O % funciona como um coringa que representa zero, um ou mais caracteres, permitindo assim encontrar todos os nomes que iniciam com “Ana”.

Resposta esperada:

Nome	Cidade
Ana Beatriz	São Paulo
Ana Carolina	Rio de Janeiro

4.5 Cláusulas LIMIT e ORDER BY

Tabela original: Pedidos

ID	Produto	Quantidade	Preço_Unitário
1	Caneta	100	1.50
2	Lápis	200	0.50
3	Borracha	150	0.75

Consulta SQL:

```
Query 1
1 • SELECT * FROM Pedidos ORDER BY Preço_Unitário DESC LIMIT 2;
```

Explicação:

A cláusula `ORDER BY` organiza os dados retornados por uma ou mais colunas, nesse caso pelo “Preço_Unitário” em ordem decrescente `DESC`. A cláusula `LIMIT` restringe o número de linhas retornadas, útil para visualizar apenas uma parte dos dados ou para desempenho em grandes bancos de dados.

Resposta esperada:

ID	Produto	Quantidade	Preço_Unitário
1	Caneta	100	1.50
3	Borracha	150	0.75

4.6 GROUP BY

Tabela original: Vendas

ID	Produto	Vendedor	Quantidade
1	Caneta	Ana	50
2	Caneta	João	30
3	Lápis	Ana	70

Consulta SQL:

```
Query 1
1 • SELECT Produto, SUM(Quantidade) AS Total_Vendido FROM Vendas GROUP BY Produto;
-
```

Explicação:

A cláusula `GROUP BY` agrupa as linhas que têm os mesmos valores em colunas especificadas, permitindo a agregação de dados, como somatória (`SUM`), média (`AVG`), máximo (`MAX`) e mínimo (`MIN`). Nesse exemplo, as vendas são agrupadas por produto, e a quantidade total vendida de cada produto é calculada e apresentada.

Resposta esperada:

Produto	Total_Vendido
Caneta	80
Lápis	70

4.7 Operadores lógicos (AND, OR, NOT)

Tabela original: Funcionarios

ID	Nome	Departamento	Salário
1	Ana	Vendas	2500
2	João	TI	3200
3	Maria	Marketing	2900

Consulta SQL:

```
Query 1
SELECT Nome FROM Funcionarios WHERE Departamento = 'TI' AND Salário > 3000 OR Departamento = 'Vendas';
```

Explicação:

Essa consulta usa os operadores lógicos `AND` e `OR` para definir condições mais complexas. O

para um registro ser selecionado (pertencer ao departamento de TI e ter salário maior que 3000), enquanto o `OR` expande a seleção para incluir também todos os funcionários do departamento de Vendas. Isso demonstra a flexibilidade do SQL para combinar múltiplas condições de filtragem.

Resposta esperada:

Nome
Ana
João

4.8 INNER JOIN

Tabela original: Clientes

ID	Nome
1	Carlos Alves
2	Alícia Santos

Tabela original: Pedidos

ID	Cliente_ID	Produto
1	1	Caneta
2	2	Lápis
3	1	Borracha

Consulta SQL:

```
Query 1
1 • SELECT Clientes.Nome, Pedidos.Produto FROM Clientes
2   INNER JOIN Pedidos ON Clientes.ID = Pedidos.Cliente_ID;
```

Explicação:

O **INNER JOIN** é usado para combinar registros de duas tabelas baseando-se em uma condição de junção. Nesse caso, a consulta junta as tabelas “Clientes” e “Pedidos” pelo ID do cliente, retornando apenas as combinações em que há correspondência nos IDs das duas tabelas. Isso permite visualizar quais produtos cada cliente pediu.

Resposta esperada:

Nome	Produto
Carlos Alves	Caneta
Alícia Santos	Lápis
Carlos Alves	Borracha

4.9 LEFT JOIN e RIGHT JOIN

Consulta SQL:

```
Query 1 ×
1 • SELECT Clientes.Nome, Pedidos.Produto FROM Clientes
2   LEFT JOIN Pedidos ON Clientes.ID = Pedidos.Cliente_ID;
```

Explicação:

O **LEFT JOIN** inclui todos os registros da tabela à esquerda (Clientes) e os registros correspondentes da tabela à direita (Pedidos). Quando não há correspondência, o resultado para a tabela à direita é **NULL**. Isso é útil para identificar clientes que podem não ter feito nenhum pedido.

Resposta esperada:

Nome	Produto
Carlos Alves	Caneta
Carlos Alves	Borracha
Alicia Santos	Lápis

4.10 UNION

Consulta SQL:

```
Query 1 ×
1 • SELECT Nome FROM Clientes
2 UNION
3 SELECT Produto FROM Pedidos;
```

Explicação: **UNION** é usado para combinar os resultados de duas ou mais

consultas **SELECT**

em um único conjunto de resultados, que inclui todas as linhas de cada consulta individual, mas elimina duplicatas. Essa operação é útil para combinar listas distintas em uma única lista consolidada.

Resposta esperada:

Nome
Carlos Alves
Alicia Santos
Caneta
Lápis
Borracha



Mãos ao código | Hora de praticar

1 • Seleção simples

Escreva uma consulta SQL para selecionar todos os campos da tabela “Funcionarios” em que o departamento é “Marketing”.

2 • Uso de LIKE com padrões

Escreva uma consulta SQL para encontrar todos os clientes na tabela “Clientes” cujos nomes terminem com “Silva”.

3 • Combinação de cláusulas WHERE com AND e OR

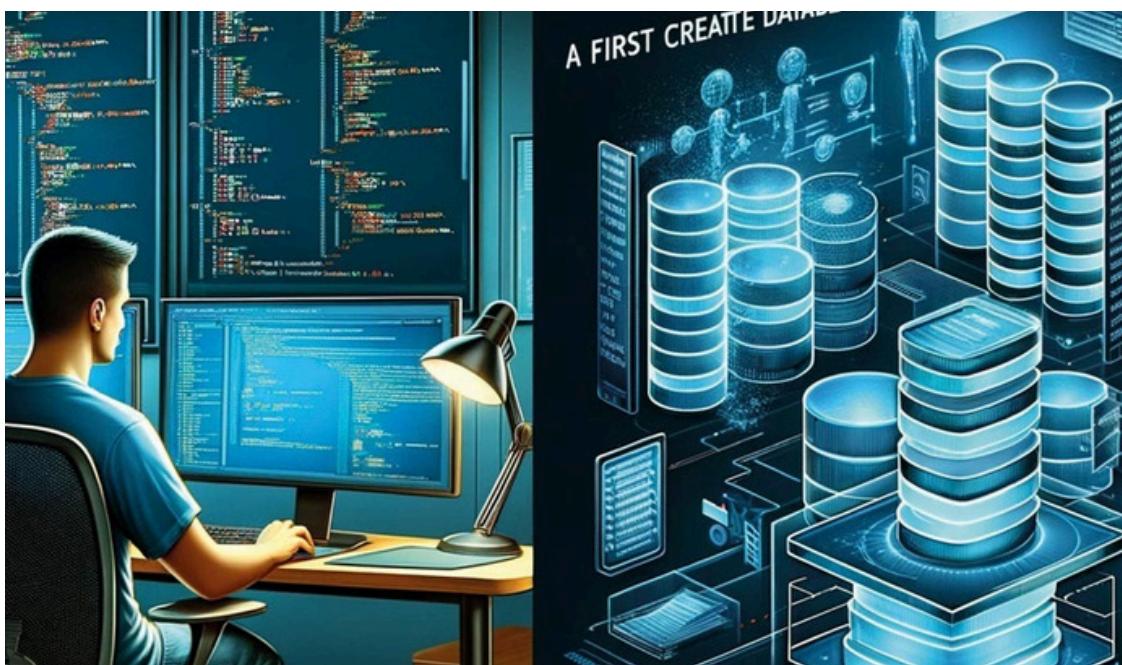
Escreva uma consulta SQL para selecionar todos os “Funcionarios” que trabalham no departamento ‘TI’ e que tenham um salário maior que 3000 ou que estejam no departamento ‘Recursos Humanos’.

4 • Ordenação e limite

Escreva uma consulta SQL para selecionar os três produtos mais caros da tabela “Produtos”. Os produtos devem ser listados em ordem decrescente de preço.

5 • Consulta com GROUP BY e função de agregação

Escreva uma consulta SQL para calcular o número total de pedidos para cada produto na tabela “Pedidos”. Agrupe os resultados pelo nome do produto.



Infinity Quiz | Teste seus conhecimentos

1 • Qual consulta SQL selecionará clientes cujo nome começa com “João”?

- a) SELECT * FROM clientes WHERE nome = 'João%'
- b) SELECT * FROM clientes WHERE nome LIKE 'João%'
- c) SELECT * FROM clientes WHERE nome IN 'João%'
- d) SELECT * FROM clientes WHERE nome == 'João%'

2 • Como você selecionaria funcionários do departamento de TI com salário superior a 5000?

- a) SELECT * FROM funcionarios WHERE departamento = 'TI' AND salario > 5000
- b) SELECT * FROM funcionarios OR departamento = 'TI' AND salario > 5000
- c) SELECT * FROM funcionarios WHERE departamento = 'TI' BUT salario > 5000
- d) SELECT * FROM funcionarios WHERE departamento = 'TI' & salario > 5000

3 • Qual comando é usado para retornar todos os registros da tabela produtos ordenados por preço em ordem crescente?

- a) SELECT * FROM produtos SORT BY preço ASC
- b) SELECT * FROM produtos ORDER BY preço DESC
- c) SELECT * FROM produtos ORDER BY preço ASC
- d) SELECT * FROM produtos ALIGN BY preço ASC

4 • Qual consulta SQL corretamente retorna os nomes dos clientes e os produtos que eles compraram, combinando as tabelas “Clientes” e “Pedidos” onde os IDs dos clientes correspondem?

- a) SELECT Clientes.Nome, Pedidos.Produto FROM Clientes INNER JOIN Pedidos ON Clientes.ID = Pedidos.Cliente_ID
- b) SELECT Clientes.Nome, Pedidos.Produto FROM Clientes JOIN Pedidos WHERE Clientes.ID = Pedidos.Cliente_ID
- c) SELECT Clientes.Nome, Pedidos.Produto FROM Clientes INNER JOIN Pedidos WHERE Clientes.ID == Pedidos.Cliente_ID
- d) SELECT Clientes.Nome, Pedidos.Produto FROM Clientes, Pedidos ON Clientes.ID = Pedidos.Cliente_ID

Respostas Infinity Quiz:

1 – B; 2 – A; 3 – C; 4 – A

Conclusão

Recapitulação dos principais conceitos aprendidos

Ao longo deste livro, exploramos profundamente os conceitos e técnicas essenciais para trabalhar com bancos de dados SQL. Desde a estruturação e criação de tabelas até a manipulação e consulta de dados, cobrimos uma variedade de operações que são cruciais no dia a dia de desenvolvedores, analistas de dados e administradores de banco de dados.

- Estrutura e criação de tabelas: aprendemos como definir tabelas com chaves primárias, atributos adequados e tipos de dados específicos, garantindo que os bancos de dados sejam organizados, eficientes e fáceis de manter.
- Manipulação de dados: vimos como inserir, atualizar e deletar registros usando comandos **INSERT**, **UPDATE** e **DELETE**. Essas operações são fundamentais para manter os dados atualizados e relevantes, permitindo que as organizações respondam dinamicamente às mudanças nas necessidades e nos dados.
- Consultas SQL: demonstramos como utilizar **SELECT** para extrair informações, o comando aplicando filtros com **WHERE**, organizando resultados com **ORDER BY**, limitando retornos com **LIMIT** e agrupando dados com **GROUP BY**. Além disso, discutimos o uso de operadores lógicos **AND**, **OR**, **NOT**) e padrões de busca com **LIKE**.
- Junções e uniões: exploramos os poderosos conceitos de **INNER JOIN**, **LEFT JOIN**, **RIGHT JOIN** e **UNION**, que permitem combinar dados de várias tabelas de maneiras complexas e informativas, essencial para relatórios detalhados e análises profundas.





Conclusão

Este livro foi concebido como um guia prático e abrangente para quem deseja dominar o SQL e os conceitos associados ao gerenciamento de bancos de dados. Cada capítulo foi estruturado para fornecer uma aprendizagem gradual, desde os fundamentos até técnicas mais avançadas, garantindo que os leitores possam construir uma base sólida de conhecimentos e habilidades.

A habilidade de manipular e consultar dados de forma eficiente é indispensável em muitas áreas, desde o desenvolvimento de software até a análise de dados e a administração de sistemas. Com este livro, esperamos que você tenha adquirido não apenas conhecimento técnico, mas também uma compreensão clara de como aplicar essas técnicas em situações reais. Encorajamos os leitores a

continuar praticando e explorando os conceitos discutidos, pois a proficiência em SQL é uma ferramenta poderosa que abre portas para oportunidades de carreira e aprimoramento profissional. À medida que avançar, mantenha-se curioso, crítico e criativo em suas consultas e manipulações de banco de dados.

Dados Internacionais de Catalogação na Publicação (CIP)
Angelica Ilacqua CRB-8/7057

Infinity School
Conceitos gerais de banco de dados / Infinity School.
-- [S. l.] : Reality Editora, 2024.
44 p.
ISBN 978-65-83135-01-8
1. SQL server (Programa de computador) I. Título

24-3011

CDD 005.3

Índices para catálogo sistemático:
1. SQL server (Programa de computador)



www.infinityschool.com.br



9 786583 135018

REALITY EDITORA