Apostila do Curso

Conteúdo e Atividades

MySQL

Banco de Dados com MySQL

Lite

Banco de dados com MySql Lite

Nome: _____

Quantidade de Aulas
3 aulas

Carga horária
4.5 horas

Programas utilizados
Workbench

Sobre o curso

O curso de MySQL Lite foi desenvolvido para proporcionar ao aluno uma compreensão sobre o funcionamento, a estrutura e a manipulação de bancos de dados.

O que aprender com este curso?

O curso capacita o aluno a compreender os tipos de dados, criar, gerenciar. O estudante vai entender fundamentos como o CRUD.

Sumário

1. Introdução ao MySql

Seja muito bem-vindo(a), futuro(a) especialista em Banco de Dados! É uma grande satisfação tê-lo(a) conosco nesta jornada de aprendizado sobre MySQL. Se você chegou até aqui, é porque já compreende a importância de dominar os bancos de dados no cenário tecnológico atual. Prepare-se para desvendar os segredos do armazenamento, organização e recuperação de informações de forma eficiente, e para construir uma base sólida que impulsionará sua carreira. Vamos começar nossa aula entendendo os conceitos fundamentais que regem o mundo dos dados.

1.1. O que é um Banco de Dados?

Um banco de dados é um sistema que permite armazenar, organizar e recuperar informações de modo eficiente. Os dados são guardados em estruturas chamadas tabelas, compostas por linhas (registros) e colunas (campos). Cada linha representa uma entidade (como um cliente ou produto), e as colunas representam características dessa entidade (como nome, idade, preço etc.). O sistema responsável por gerenciar esses dados é chamado de Sistema Gerenciador de Banco de Dados (SGBD), que controla inserções, consultas, segurança, integridade e desempenho.

1.2. A História dos Bancos de Dados

Desde os primeiros tempos, a humanidade registrou informações em papéis, pergaminhos e fichários. Com o surgimento dos computadores na década de 1960, começaram os bancos de dados informatizados, mas ainda baseados em modelos rígidos como o hierárquico ou em rede, que exigiam navegação por ponteiros e pouco flexíveis.

Em 1970, Edgar Frank Codd, cientista da IBM, publicou o artigo seminal “A Relational Model of Data for Large Shared Data Banks”, propondo o modelo relacional, que trata os dados como tabelas lógicas, indiferentes à forma física de armazenamento. A partir desse modelo vieram projetos como o System R da IBM (1974–1977), que implementou a linguagem SQL (antes chamada SEQUEL) e tornou viável o modelo relacional na prática. Outro projeto pioneiro foi o Ingres, desenvolvido na Universidade da Califórnia, que deu origem a diversos bancos comerciais como Sybase e SQL Server.

Já em 1976, Peter Chen propôs o modelo Entidade-Relacionamento (ER), que trouxe uma forma clara de planejar a estrutura de dados com entidades e relacionamentos. Na década de 1980, bancos como Oracle, IBM DB e Microsoft SQL Server se consolidaram comercialmente, popularizando o uso em empresas e setores diversos.

1.3. O que um Banco de Dados pode resolver em uma Empresa?

Bancos de dados são ferramentas poderosas para resolver diversos desafios empresariais, proporcionando:

- **Organização e Estrutura:** Com tabelas bem definidas, um banco de dados evita

redundância e inconsistência nos dados. A aplicação de normalização (divisão em tabelas menores) garante que a informação seja armazenada de forma eficiente e consistente.

- **Consultas e Manipulações Eficientes:** A linguagem SQL permite ao usuário expressar o que deseja (por exemplo: listar todos os empréstimos atrasados), e o SGBD decide como buscar os dados de forma otimizada. Consultas SQL podem combinar tabelas, filtrar, agrupar e ordenar os dados de forma muito eficiente.
- **Integridade e Segurança dos Dados:** Os bancos de dados suportam restrições como unicidade, integridade referencial, campos obrigatórios e regras de negócios diretamente no esquema. Isso impede duplicação, dados inválidos ou inconsistentes. Além disso, controlam quem pode acessar ou modificar cada dado.
- **Transações Confiáveis:** Operações que envolvem múltiplas etapas (como transferir dinheiro de uma conta para outra) são tratadas como transações, que obedecem às propriedades ACID (Atomicidade, Consistência, Isolamento, Durabilidade). Garantindo que tudo aconteça corretamente ou, em caso de falha, volte ao estado anterior sem bagunça nos dados.
- **Concorrência e Desempenho:** Em ambientes com muitos usuários acessando dados ao mesmo tempo, técnicas como MVCC (Controle de Concorrência Multiversão) permitem que cada usuário veja uma cópia consistente dos dados, sem bloqueios que prejudicam a performance. Além disso, o uso de índices acelera buscas, evitando digitalizações de toda a tabela.

1.4. Por que Estudar Banco de Dados?

Desde sistemas financeiros e redes sociais até aplicações escolares, bancos de dados robustos são essenciais para sistemas confiáveis, seguros e eficientes. Compreender sua evolução histórica, o modelo relacional, SQL, integridade, transações e otimização de consultas fornece uma base sólida para projetar sistemas reais de maneira profissional. Isso transforma dados em informação útil e possibilita decisões bem fundamentadas.

1.5. O que é um SGBD?

Um SGBD, ou Sistema Gerenciador de Banco de Dados, é um conjunto de programas de computador que administra um ou vários bancos de dados. Ele age como intermediário entre as aplicações (como sistemas de cadastro, relatórios ou ERPs) e os dados armazenados, liberando o desenvolvedor da tarefa de controlar diretamente acesso, persistência e manipulação dos dados.

O SGBD gerencia operações fundamentais como criar, consultar, alterar e excluir dados em bancos de dados, garantindo que esses processos ocorram de forma organizada e segura. Ele também fornece uma interface, geralmente por meio de APIs ou drivers, que executam comandos SQL ou outros tipos de linguagens de consulta para que as aplicações possam incluir, alterar ou consultar dados sem lidar com o armazenamento físico.

1.6. Tipos de SGBDs

Existem diferentes modelos de SGBDs, cada um adequado a necessidades específicas. Os principais tipos são:

- **Relacional (RDBMS):** Organiza os dados em tabelas relacionadas entre si por chaves, sendo o tipo mais utilizado hoje. Suporta SQL e exige um esquema predefinido.

- **Não-relacional (NoSQL):** Armazena dados em formatos como documentos, chave-valor, colunas ou grafos. É flexível em esquemas e indicado para grandes volumes de dados distribuídos.

- **Orientado a objetos (SGBDO/O):** Trata os dados como objetos, mantendo identidade, métodos e heranças, ideal para aplicações com entidades complexas.

- **Objeto-relacional (SGBDOR):** Combina tabelas relacionais com recursos de orientação a objetos, permitindo tipos de dados personalizados e herança dentro do modelo relacional.

Além desses, há ainda SGBDs com modelos hierárquicos, em rede, multidimensional e em memória, conforme as necessidades de desempenho ou estrutura dos dados.

1.7. O que é SQL?

SQL (Structured Query Language ou Linguagem de Consulta Estruturada) é a linguagem padrão usada para criar, consultar, modificar e gerenciar dados em bancos de dados relacionais.

1.8. A História do SQL

A SQL surgiu no início da década de 1970 nos laboratórios da IBM, com Donald D. Chamberlin e Raymond F. Boyce. Inicialmente chamada de SEQUEL ("Structured English Query Language"), a linguagem foi renomeada para SQL porque SEQUEL já era marca registrada. O objetivo era proporcionar uma sintaxe que permitisse expressar consultas de forma clara e em inglês estruturado, manipulando dados conforme o modelo relacional de Edgar F. Codd, implementado no projeto System R da IBM. Hoje, SQL é a linguagem universal em bancos de dados relacionais, ainda que cada sistema adote variações que estendem ou adaptam o padrão.

1.9. O que é MySQL?

MySQL é um Sistema Gerenciador de Banco de Dados Relacional (RDBMS) que implementa a linguagem SQL para armazenar, buscar e manipular dados.

1.10. A História e Popularidade do MySQL

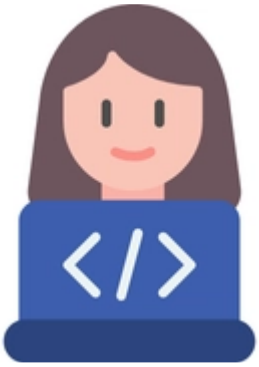
O MySQL foi criado em 1994 pela empresa sueca MySQL AB, fundada por David Axmark, Allan Larsson e Michael “Monty” Widenius. A sua primeira versão pública foi lançada em maio de 1995. O nome MySQL combina “My”, que é o nome da filha de Widenius, com “SQL” a linguagem que o sistema utiliza.

Desde o início, o MySQL foi distribuído sob a licença Open Source (GPL), permitindo seu uso gratuito, além de ser instalado em Windows, Linux, macOS e outros sistemas operacionais. Em janeiro de 2008, a MySQL AB foi adquirida pela Sun Microsystems, que por sua vez foi comprada pela Oracle Corporation em 2009. Desde então, o MySQL continua disponível como código aberto, embora existam versões comerciais com suporte adicional.

A origem do MySQL está ligada à insatisfação com o desempenho de sistemas anteriores como o mSQL e os arquivos ISAM, considerados lentos ou pouco flexíveis. Michael Widenius e seus colegas decidiram criar um sistema compatível com a API domSQL, mas com interface SQL e melhor desempenho. Com isso, permitiram que desenvolvedores migrassem facilmente para MySQL sem grandes adaptações no código.

O MySQL ganhou popularidade por sua facilidade de uso, velocidade e integração com tecnologias web, especialmente com PHP e o pacote LAMP (Linux, Apache, MySQL, PHP/Python/Perl). Isso fez do MySQL a escolha natural para muitos sites dinâmicos e sistemas de gestão de conteúdo como WordPress, Drupal, Joomla, além de grandes empresas como Facebook, Twitter e Wikipedia que utilizam MySQL em produção.

1.11. O que é MySQL Workbench?



O MySQL Workbench é uma ferramenta gráfica oficial da Oracle que combina em um único ambiente a modelagem de dados, desenvolvimento em SQL e administração de servidores MySQL.

1.12. Por que Usar MySQL Workbench?

Ele funciona como um “painel central” onde você pode:

- **Modelar visualmente seu banco de dados:** com diagramas (ERDs) criando e ajustando tabelas, chaves estrangeiras, índices, etc., tanto a partir do zero quanto a partir de um banco já existente.
- **Escrever, editar e executar comandos SQL:** com ajuda de destaque de sintaxe, auto-completar e histórico, facilitando aprendizado e produtividade.
- **Administrar o servidor e banco de dados:** acessando usuários, permissões, backups, logs e monitoramento de desempenho, tudo via interface.

O MySQL Workbench é o ambiente perfeito para aprender banco de dados MySQL de forma completa e prática. Ele facilita a visualização da estrutura, o domínio do SQL e a gestão do servidor, tornando nossa jornada de estudos muito mais eficaz e envolvente.

1.13. Passo a Passo da Instalação

Para instalar o MySQL Workbench no Windows, siga os passos abaixo:

1. **Abrir o Navegador:** Comece abrindo seu navegador de internet.
2. **Pesquisar o Site:** Na barra de pesquisa, digite "www.mysql.com/downloads" pressione Enter.
3. **Localizar o Download:** Role a página para baixo até encontrar a seção de downloads e clique em "MySQL Community (GPL) Downloads", destacado em azul.
4. **Selecionar o Instalador:** Na próxima página, clique em "MySQL Installer for Windows".
5. **Baixar a Versão Mais Recente:** Clique em "Download" na última versão disponível (ex: 8.0.43) no primeiro da lista.
6. **Ignorar Login:** Quando solicitado a fazer login, clique em "No thanks, just start my download." para iniciar o download diretamente.
7. **Executar o Instalador:** Após o download, clique duas vezes no arquivo .exe baixado para iniciar o instalador.
8. **Escolher Tipo de Instalação:** No instalador do MySQL, selecione a opção "Full" e clique em "Next".
9. **Confirmar Produtos:** Será exibida a lista de todos os produtos MySQL a serem instalados. Clique em "Execute".
10. **Avançar:** Após o download de todos os produtos, clique em "Next".
11. **Instalação dos Produtos:** O instalador realizará a instalação de cada produto. Clique em "Next" novamente.
12. **Configuração Inicial:** Clique em "Next" nas

próximas telas de configuração, sem alterar nada.

13. **Definir Senha do Root:** Neste local, será solicitada a senha do usuário "Root". Defina a senha como "123456" nos dois campos e clique em "Next".
14. **Serviço MySQL:** A tela seguinte informa sobre o serviço do MySQL no Windows. Não altere nada e clique em "Next".
15. **Configurações Finais:** Clique em "Next" nas próximas telas de configuração.
16. **Aplicar Configurações:** Clique em "Execute" para aplicar as configurações.
17. **Finalizar:** Após a aplicação das configurações, clique em "Finish".
18. **Configuração do Router**: Será exibida uma configuração de Router que não será utilizada. Clique em "Finish".
19. **Testar Conexão:** Será necessário testar a conexão com o MySQL. Coloque a senha "123456" e clique em "Check" para verificar a conexão. Se estiver tudo certo, clique em "Next".
20. **Executar Configuração:** Confirme as configurações clicando em "Execute".
21. **Finalizar Configuração:** Após a conclusão, clique em "Finish".
22. **Concluir Instalação:** Clique em "Next" e, por fim, em "Finish" para encerrar a instalação do MySQL.

Após a instalação, o MySQL Workbench será aberto automaticamente.

1.14. O que é o Usuário 'Root'?

Em MySQL, o usuário root é a conta padrão criada durante a instalação. Ele é o super-usuário do banco de dados, com "poder divino". Tem acesso completo a todos os bancos, tabelas, operações e privilégios, sem restrições. Diferente do usuário root do sistema operacional, este root existe apenas dentro do próprio MySQL.

1.15. Por que Usar o 'Root' para Conectar?

O usuário root nos oferece total autonomia para criar, gerenciar e explorar o banco de dados sem barreiras, sendo ideal para fins de estudo e desenvolvimento inicial.

1.16. Conectando ao Servidor Local MySQL

Com o MySQL Workbench aberto, clique em "local mysql" onde consta nosso servidor local MySQL que testamos durante a instalação. Antes de conectar, será solicitada a senha para acessarmos nosso MySQL local. Clique em "Password", coloque a senha que definimos durante o curso (123456) e clique em "OK" para conectar.

1.17. Entendendo SCHEMAS e Query

Ao conectar, você verá o painel "SCHEMAS" e "Query ".

- **Schema:** É sinônimo de banco de dados. São coleções de objetos (tabelas, views, rotinas, etc.) organizados em um mesmo namespace. Quando você cria um schema (banco de dados), está basicamente criando um contêiner para guardar suas tabelas, definir relacionamentos e aplicar restrições.
- **Query:** Uma Query é uma consulta SQL, um comando que você escreve para solicitar ou manipular dados no banco. O painel "Query " será o local onde você executará seus comandos SQL no MySQL.

1.18. CREATE SCHEMA vs. CREATE DATABASE

Quando criamos um banco de dados, podemos utilizar dois métodos:

- **CREATE DATABASE:** É mais comum em bancos como MySQL, SQL Server e SQLite.

- **CREATE SCHEMA**: Tem origem no padrão SQL ANSI, e é mais utilizado em bancos como PostgreSQL e Oracle, onde schema é uma estrutura dentro de um banco de dados, e não o banco em si.

Em MySQL, CREATE SCHEMA e CREATE DATABASE são sinônimos e podem ser usados de forma intercambiável.

1.19. Executando o Comando de Criação

Para criar seu primeiro banco de dados, no campo da "Query ", escreva o comando:

```sql
CREATE SCHEMA curso;
```

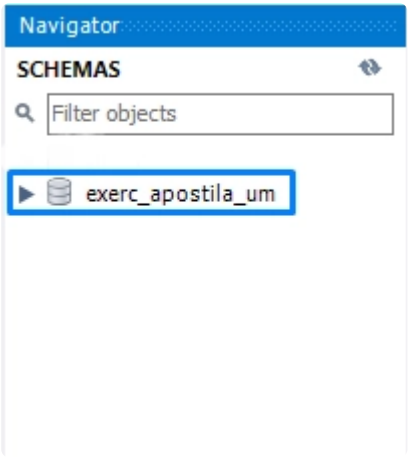
Para executar o comando, clique no primeiro símbolo de raio no menu superior da Query. Após a execução, você verá a confirmação do banco de dados "curso" criado.

1.20. Visualizando o Banco de Dados Criado

Após criar o banco de dados, para que ele apareça no painel "SCHEMAS", clique no símbolo das setas girando entre si (ícone de atualização) ao lado do painel "SCHEMAS". Agora sim, seu banco de dados curso estará visível.

1.21. Exercício prático

Com o MySQL Workbench aberto, crie um um novo banco de dados utilizando **DATABASE** ou **SCHEMA** com o nome (exerc_apostila_um) dentro da query 1.



Seu banco de dados criado deverá constar como a imagem acima.

Anotações

2. Tipos de Dados e Criação de Tabelas

Seja bem-vindo(a) à nossa segunda apostila do curso de Banco de Dados com MySQL sobre Tipos de dados e criação de Tabelas.

2.1. Introdução às Tabelas no MySQL

Imagine que você tem um caderno de anotações onde registra informações sobre seus amigos: nome, idade, telefone e endereço. Cada linha do caderno representa um amigo, e cada coluna representa uma informação específica sobre ele.

No MySQL, uma **tabela** funciona de maneira muito semelhante. Ela é a estrutura fundamental onde armazenamos dados organizados em linhas e colunas. Pense nela como uma planilha gigante, mas muito mais poderosa e eficiente para gerenciar grandes volumes de informações.

- **Cada linha** da tabela representa um **registro** único (como um amigo no seu caderno).
- **Cada coluna** da tabela representa um **tipo de informação** específica (como nome, idade, etc.).

As tabelas são a espinha dorsal de qualquer banco de dados relacional, permitindo que os dados sejam armazenados de forma lógica e acessível. Compreender o funcionamento das tabelas é o primeiro passo para dominar o MySQL e construir sistemas robustos.

2.2. Estrutura de uma Tabela: Colunas e Linhas

Para entender melhor como uma tabela funciona no MySQL, vamos detalhar seus dois componentes principais:

1. **Colunas (Campos):** Pense nas colunas como as categorias de informação que você deseja armazenar. Cada coluna é projetada para guardar um tipo específico de dado. Por exemplo, em uma tabela de "clientes", você pode ter colunas como:

- **id_cliente:** Um número único para identificar cada cliente.
- **nome:** O nome completo do cliente.
- **e-mail:** O endereço de e-mail do cliente.
- **telefone:** O número de telefone do cliente.

1. **Linhas (Registros)**: As linhas, por outro lado, são as entradas individuais de dados. Cada linha contém um conjunto completo de informações para um único item ou entidade. Usando o exemplo da tabela de "clientes":

- **Linha 1:** id_cliente: 1, nome: João Silva, email: joão@email.com, telefone: (51) 99999-9999
- **Linha 2:** id_cliente: 2, nome: Maria Oliveira, email:maria@email.com, telefone: (51) 988888-8888

Cada linha é um "registro" completo, e todas as linhas em uma tabela seguem a mesma estrutura de colunas. Essa organização padronizada é o que torna os bancos de dados relacionais tão eficientes para gerenciar informações.

2.3. Por que Usamos Tabelas?

As tabelas são a base de qualquer banco de dados relacional e são essenciais por diversas razões. Elas permitem organizar grandes quantidades de informações de forma estruturada e eficiente, o que é crucial para qualquer sistema que lide com dados. Veja os principais motivos:

- **Armazenar Dados de Maneira Organizada:** As tabelas garantem que os dados sejam guardados de forma padronizada, facilitando o acesso, a busca e a atualização das informações. Imagine tentar encontrar um dado específico em um arquivo sem nenhuma organização, seria um caos! As tabelas resolvem isso.

- **Relacionar Informações:** Uma das maiores vantagens dos bancos de dados relacionais é a capacidade de conectar diferentes tabelas. Por exemplo, em uma loja online, você pode ter uma tabela de clientes e outra de pedidos. As tabelas permitem associar um cliente específico a todos os seus pedidos, criando uma rede de informações interligadas.

- **Realizar Consultas Rápidas:** Com os dados bem organizados em tabelas, é possível realizar consultas complexas e encontrar informações específicas de forma muito rápida. Quer saber todos os clientes que moram em uma determinada cidade? Uma consulta bem feita pode trazer essa informação em segundos.

Entender o que são tabelas no MySQL é fundamental para trabalhar com bancos de dados. Elas são a base onde armazenamos e organizamos informações, permitindo que sistemas e aplicativos funcionem de maneira eficiente e escalável.

2.4. Tipos de Dados no MySQL

Quando você começa a criar suas tabelas no MySQL, cada coluna exige que você defina um **tipo de dado**. Essa definição é crucial, pois ela comunica ao banco de dados duas informações muito importantes:

- **Que tipo de informação será armazenada:** Se é um número, um texto, uma data, um valor verdadeiro/falso, etc.

- **Como o MySQL deve lidar com ela:** Isso inclui como o dado será armazenado, o espaço que ocupará, e como ele pode ser manipulado em operações e consultas.

Escolher o tipo de dado correto ajuda o banco a proteger a integridade dos dados, acelerar pesquisas e evitar desperdício de espaço. Imagine uma tabela onde você quer guardar dados sobre alunos:

- Se a coluna é para o **número de matrícula**, você usará um tipo numérico apropriado (ex: INT).

- Se é para o **nome do aluno**, deve usar um tipo de texto (ex: VARCHAR).

- Se é para a **data de aniversário**, existe um tipo específico para datas (ex: DATE).

Cada tipo de dado tem suas próprias regras e limites. Por exemplo:

Número:

- **INT** : Armazena apenas números inteiros (sem casas decimais).
- **DECIMAL** (ou **NUMERIC**): Armazena números com casas decimais exatas, útil para valores como preços ou notas, onde a precisão é fundamental.

Textos:

- **VARCHAR**: Serve para textos curtos e que mudam de tamanho, como nomes ou e-mails. O VAR significa “variável”, ou seja, o banco não reserva espaço fixo para cada registro, economizando espaço.

- **TEXT:** Usado quando se espera guardar textos longos, por exemplo, descrições completas ou artigos.

Valores Pré-definidos:

- **ENUM:** Se você sabe que sempre haverá apenas algumas opções de valor (ex: status 'ativo', 'inativo'), pode usar ENUM , que limita as entradas a uma lista definida.

Datas e Horários:

- **DATE:** Para guardar apenas a data no formato YYYY-MM-DD.
- **DATETIME:** Para guardar data e hora juntas.
- **TIMESTAMP:** Similar ao **DATETIME**, mas trata fusos horários automaticamente e é ideal para registrar o momento exato de inserções ou alterações no banco.

Escolher o tipo de dado certo faz uma grande diferença: evita erros na hora de salvar ou calcular valores, torna as buscas por informações mais rápidas e permite que o banco use menos memória. Não se preocupe em memorizar todos agora; a prática o ajudará a entender qual usar em cada situação.

2.5. O que é um Registro?

Em uma tabela do MySQL, um **registro** (também chamado de **linha** ou row, e às vezes de tupla) representa uma única entrada. Pense nele como um conjunto de informações relacionadas que pertence a uma pessoa, objeto ou evento específico.

Cada registro agrupa valores em colunas distintas. Por exemplo, em uma tabela de alunos , cada registro pode conter:

- **ID:** O identificador único do aluno.
- **nome:** O nome completo do aluno.
- **idade**: A idade do aluno.
- **turma:** A turma em que o aluno está matriculado.
- **data_de_matricula:** A data em que o aluno foi matriculado.

Quando você insere dados em uma tabela, você está adicionando um novo registro. Ao consultar, você está buscando um ou mais registros. Cada coluna desse registro espera um valor compatível com o tipo de dado definido para ela, e juntos, esses valores formam uma linha lógica na tabela.

A estrutura de cada registro é igual em toda a tabela, o que significa que todos os registros têm as mesmas colunas, embora com valores diferentes. Para entender de forma simples, imagine uma prateleira de fichas de papel, onde cada ficha possui campos preenchidos como nome, data de nascimento, telefone, etc. No contexto do MySQL, cada ficha equivale a um registro armazenado em uma linha da tabela. E assim como cada ficha tem o mesmo formato, cada registro na tabela segue a mesma estrutura definida pelas colunas.

2.6. Criando sua Primeira Tabela no MySQL Workbench

Agora que já revisamos os conceitos fundamentais, vamos colocar a mão na massa e criar nossa primeira tabela no MySQL Workbench. Esta ferramenta é o seu principal aliado para interagir com o banco de dados de forma visual e prática.

Passo a Passo:

1. **Abrindo o MySQL Workbench:** Você pode abrir o MySQL Workbench pesquisando por ele na barra de pesquisa do Windows ou através do ícone na sua área de trabalho.
2. **Conectando ao Serviço Local:** Com o Workbench aberto, você verá uma lista de conexões. Clique na sua conexão local, geralmente identificada como Local instance MySQL ou similar, dentro de MySQL Connections.
3. **Inserindo a Senha:** Ao tentar acessar seu

serviço MySQL local, será solicitada a senha que você definiu durante a instalação. Digite a senha (no nosso exemplo, 123456) e clique em "OK".

4. **Dentro do Serviço Local:** Parabéns! Você está agora dentro do serviço local do MySQL Workbench, pronto para começar a trabalhar com seus bancos de dados e tabelas.

2.7. Entendendo os Comandos SQL: Convenções de Escrita

Antes de mergulharmos na criação da tabela, é importante entender uma convenção comum ao escrever comandos SQL. Você deve ter notado que, ao longo do curso, utilizamos palavras-chave como CREATE TABLE , INSERT INTO e SELECT em letras maiúsculas.

Embora o SQL não exija que essas palavras estejam em maiúsculas (o SQL não é case-sensitive para comandos), essa prática é amplamente adotada por facilitar a leitura e organização do código, especialmente para quem está iniciando ou revisitando scripts depois de um tempo. Manter essa convenção torna o código mais legível e padronizado, o que é uma boa prática no desenvolvimento de bancos de dados.

2.8. Detalhando a Criação da Tabela 'alunos'

Vamos agora criar a tabela **alunos** e entender cada parte do comando SQL. Esta tabela armazenará informações básicas sobre os alunos de uma instituição. No MySQL Workbench, com seu Schema curso selecionado (clique com o botão direito em curso no painel **SCHEMAS** e selecione **Set as Default Schema**), digite o seguinte comando na área da **Query 1**:

```sql
CREATE TABLE alunos (
id INT AUTO_INCREMENT PRIMARY KEY,
nome VARCHAR(100) NOT NULL,
nota NUMERIC(5,2),
ativo BOOLEAN DEFAULT TRUE,
criado_em TIMESTAMP DEFAULT
CURRENT_TIMESTAMP
);
```

Agora, vamos detalhar cada linha deste comando:

2.8.1. Coluna id : INT AUTO_INCREMENT PRIMARY KEY

- **id:** É o nome da nossa coluna, que servirá como identificador único para cada aluno. Por exemplo, o ID 21 pertence ao aluno João. Isso facilita a identificação e futuras consultas.
- **INT:** Define o tipo de dado da coluna como integer , ou seja, armazenará números inteiros (sem casas decimais). É perfeito para identificadores, pois são eficientes e fáceis de usar.
- **AUTO_INCREMENT:** Esta propriedade diz ao MySQL para gerar automaticamente um número único para esta coluna sempre que um novo registro for inserido. Por padrão, ele começa com 1 e aumenta em 1 a cada novo aluno. Isso significa que você não precisa se preocupar em digitar manualmente o próximo ID.
- **PRIMARY KEY:** Define esta coluna como a **chave primária** da tabela. Uma chave primária é um identificador exclusivo para cada registro. Isso garante que nenhum outro registro possa ter o mesmo valor no campo id , assegurando que cada aluno seja único e facilmente localizável. A combinação AUTO_INCREMENT com PRIMARY KEY é muito comum e prática para criar identificadores únicos.

2.8.2. Coluna nome : VARCHAR( ) NOT NULL

- **nome:** Nome da coluna, que armazenará o nome completo do aluno.
- **VARCHAR(100):** Define o tipo de dado como VARCHAR , que é usado para armazenar palavras ou textos curtos. O número 100 entre parênteses indica o limite máximo de caracteres que podem ser armazenados (neste caso, até 100 caracteres). O VAR em VARCHAR significa "variável", o que é muito eficiente, pois o banco de dados só usa o espaço necessário para o texto inserido, economizando memória.
- **NOT NULL:** Esta é uma restrição que impede que a coluna fique vazia. Ou seja, sempre que você for cadastrar um novo aluno, é obrigatório informar um valor para o nome . Se tentar inserir um registro sem preencher esta coluna, o MySQL impedirá a operação e mostrará um erro, garantindo a integridade dos dados.

2.8.3. Coluna nota : NUMERIC(5,2)

- **nota:** Nome da coluna, para armazenar a nota do aluno.
- **NUMERIC(5,2):** Define o tipo de dado como NUMERIC , que é ideal para números com precisão fixa, como notas ou valores monetários. 5 (precisão) indica o número total de dígitos permitidos (incluindo os antes e depois da vírgula), e o 2 (escala) indica o número de casas decimais após a vírgula. Assim, valores como 123.45 ou 9.30 são válidos, mas 123.456 não seria. No MySQL, NUMERIC é sinônimo de DECIMAL , e ambos funcionam da mesma forma.

2.8.4. Coluna ativo : BOOLEAN DEFAULT TRUE

- **Ativo:** Nome da coluna, para indicar se o aluno está ativo ou não.
- **BOOLEAN:** No MySQL, BOOLEAN (ou BOOL) é um apelido para TINYINT(1). Internamente, o banco armazena 0 para falso e 1 para verdadeiro. Isso é perfeito para guardar valores simples de verdadeiro/falso.
- **DEFAULT TRUE:** Esta propriedade define um valor padrão para a coluna. Se você não informar um valor para ativo ao inserir um novo registro, o MySQL automaticamente definirá como TRUE (ou 1). Isso é útil para que todos os novos alunos sejam considerados ativos por padrão, a menos que você especifique o contrário.

2.8.5. Coluna criado_em : TIMESTAMP DEFAULT CURRENT_TIMESTAMP

- **criado_em:** Nome da coluna, para registrar a data e hora de criação do registro.

- **TIMESTAMP:** Este tipo de dado armazena uma marca de tempo (data e hora) com precisão até segundos. Ele é automaticamente convertido para UTC (Tempo Universal Coordenado) internamente e, ao ser consultado, é convertido para o fuso horário da sua sessão. É ideal para registrar quando um registro foi criado ou atualizado.
- **DEFAULT CURRENT_TIMESTAMP:** Esta propriedade garante que, se você não informar manualmente o valor para esta coluna, o MySQL automaticamente preencherá com a data e hora atuais do momento da inserção do registro. Isso é muito útil para manter um histórico de quando os dados foram adicionados.

Note que a última coluna (criado_em) não termina com vírgula, pois é a última definição antes de fechar o parêntese da tabela.

2.9. Executando o Comando de Criação da Tabela

Com o comando CREATE TABLE pronto e finalizado, precisamos executá-lo para que a tabela seja de fato criada no seu banco de dados. Antes de executar, é crucial definir em qual banco de dados a tabela será criada. Existem duas formas principais de fazer isso:

1. **Configurando o Banco de Dados Padrão (Recomendado para o Curso):**

- No painel SCHEMAS do MySQL Workbench, clique com o botão direito do mouse em cima do seu banco de dados (curso).

- Selecione a opção Set as Default Schema. Ao ativar, o nome do Schema ficará em negrito, indicando que agora ele é o banco de dados padrão para a execução das suas queries na Query 1.
- Esta é a configuração que será utilizada no curso, pois simplifica a escrita dos comandos.

1. **Alterando o Comando (Para Uso Avançado ou em Outros Contextos):**

- Você também pode especificar o banco de dados diretamente no comando SQL, usando a sintaxe USE nome_do_banco; antes do CREATE TABLE , ou prefixando o nome da tabela com o nome do banco de dados, como CREATE TABLE curso. alunos (...) . Nesse caso, mesmo que outro Schema esteja selecionado como padrão, a tabela será criada dentro de curso porque o nome do Schema está explicitamente indicado.

- Por enquanto, vamos manter o comando padrão, com o nosso banco de dados definido como padrão.

2.9.1. Executando o Comando:

Após definir o banco de dados padrão, clique no ícone de um raio (geralmente localizado acima da área da Query 1) para executar o comando. Se tudo estiver correto, você verá uma mensagem de sucesso na área e será criada com as colunas e tipos de dados solicitados.

2.10. Exercício Prático

Para consolidar o que aprendemos, vamos a um desafio prático! Imagine que uma biblioteca comunitária de uma cidade recrutou você para modelar o banco de dados usado no cadastro de seu acervo. Sua primeira tarefa é criar uma tabela de livros que contenha as seguintes informações:

- Número de identificação do livro
- Nome do livro
- Preço do livro
- Indica se a biblioteca possui exemplares (status sim/não)
- Data em que o livro foi registrado no acervo

2.10.1. Instruções:

- Crie um novo banco de dados chamado `biblioteca_comunitaria` utilizando:

```sql
CREATE DATABASE biblioteca_comunitaria;
```

> Não se esqueça do ponto e vírgula (;) para finalizar o comando.

- Execute esse comando para criar o banco de dados.
- Atualize o painel SCHEMAS no MySQL Workbench para visualizar o novo banco.
- Clique com o botão direito sobre biblioteca_comunitaria selecione **Set as Default Schema** para defini-lo como padrão.

- Dentro desse Schema padrão, crie a tabela `livros` usando os tipos de dados mais adequados para cada coluna solicitada. Use restrições como `NOT NULL`, `PRIMARY KEY`, `AUTO_INCREMENT`, `DEFAULT` quando fizerem sentido.

Olá! Seja muito bem-vindo(a) à sua terceira aula de Banco de Dados com MySQL. Estamos animados para continuar nossa jornada no mundo dos dados e aprofundar seus conhecimentos. Prepare-se para aprender conceitos fundamentais que são a base de qualquer aplicação que lida com informações!

3.1. O que é CRUD?

Nesta aula, vamos desvendar um dos pilares do gerenciamento de dados: o CRUD. Mas o que significa essa sigla? CRUD é um acrônimo formado pelas letras iniciais de quatro operações básicas e essenciais que podemos realizar com os dados dentro de um banco de dados:

- **C - Create** (Criar)
- **R - Read** (Ler)
- **U - Update** (Atualizar)
- **D - Delete** (Excluir)

Essas operações são a espinha dorsal de qualquer sistema que manipula informações. Pense em qualquer aplicativo que você usa diariamente: uma rede social, um aplicativo de banco, uma loja online. Todos eles dependem do CRUD para funcionar. Quando você cria um perfil, lê suas mensagens, atualiza suas informações ou exclui uma postagem, você está realizando operações CRUD!

3.1.1. A Importância do CRUD

Por que o CRUD é tão importante? Simplesmente porque ele representa o ciclo de vida completo dos dados em um sistema. Desde o momento em que uma informação é gerada até o momento em que ela é removida, ela passa por essas quatro fases. Seja um site, um aplicativo móvel ou um sistema empresarial complexo, sempre haverá a necessidade de:

- **Criar** novos registros (ex: cadastrar um novo usuário).
- **Ler** dados existentes (ex: exibir a lista de produtos).
- **Atualizar** informações (ex: mudar o endereço de um cliente).
- **Excluir** dados que não são mais necessários (ex: remover um item do carrinho de compras).

Compreender o CRUD não é apenas sobre memorizar comandos, mas sim sobre entender a lógica por trás de como os dados são gerenciados e manipulados em qualquer sistema. É um conhecimento fundamental que você levará para qualquer tecnologia de banco de dados que venha a aprender no futuro.

Agora que sabemos o que é CRUD e por que ele é tão vital, vamos explorar cada uma dessas operações em detalhes, com exemplos práticos no MySQL.

3.2. Os Comandos do CRUD na Prática

Vamos agora mergulhar nos comandos SQL que nos permitem executar as operações CRUD no MySQL. É importante lembrar que, embora os exemplos sejam focados no MySQL, os conceitos

são universais para a maioria dos bancos de dados relacionais.

3.2.1. C - Create (Criar)

A operação Create é o primeiro passo no ciclo de vida dos dados. É o ato de adicionar novos registros ao seu banco de dados. Pense em um formulário de cadastro de usuários em um site: quando você preenche seus dados e clica em 'Cadastrar', você está realizando uma operação de Create.

3.2.2. Comando INSERT INTO

No MySQL, o comando utilizado para criar novos registros é o INSERT INTO. Ele nos permite especificar em qual tabela queremos inserir os dados e quais valores serão atribuídos a cada coluna.

3.2.2.0.1. Sintaxe Básica:

```sql
INSERT INTO nome_da_tabela (coluna1,
coluna2, coluna3, ...)
VALUES (valor1, valor2, valor3, ...);
```

- **nome_da_tabela:** O nome da tabela onde você deseja inserir o novo registro.
- **(coluna1, coluna2, coluna3, ...):** Uma lista das colunas nas quais você deseja inserir dados. É uma boa prática sempre especificar as colunas, mesmo que você esteja inserindo dados em todas elas, para evitar erros e tornar seu código mais legível.
- **(valor1, valor2, valor3, ...):** Os valores correspondentes a cada coluna, na mesma ordem em que as colunas foram listadas. Valores de texto devem estar entre aspas simples (' '), enquanto números e booleanos (TRUE/FALSE) não precisam.

3.2.3. Exemplo Prático e Explicação Detalhada

Vamos usar o exemplo do nosso roteiro para entender melhor:

3.2.3.0.1. Comando:

```sql
INSERT INTO clientes (nome, email) VALUES
('Ana Souza', 'ana@email.com');
```

3.2.4. Explicação:

Neste comando, estamos dizendo ao MySQL:

- **INSERT INTO clientes:** "Quero inserir um novo registro na tabela chamada clientes."
- **(nome, email):** "Os dados que vou fornecer são para as colunas nome e email."
- **VALUES ('Ana Souza', 'ana@email.com'):** "Os valores para essas colunas serão 'Ana Souza' para nome e 'ana@email.com' para email."

3.2.5. Um pouco mais a fundo:

Imagine que a tabela clientes também tivesse uma coluna id (que é preenchida automaticamente) e uma coluna data_cadastro. Ao especificar apenas nome e e-mail no INSERT INTO, estamos informando ao banco que queremos preencher apenas essas duas colunas com os valores fornecidos. As outras colunas, como id e data_cadastro, serão tratadas de acordo com suas configurações (auto-incremento, valor padrão, etc.).

3.2.6. História para ilustrar:

Pense em uma biblioteca. Quando um novo livro é adquirido, ele precisa ser catalogado. O INSERT INTO é como o bibliotecário preenchendo a ficha de um novo livro: ele anota o título, o autor, o ano de publicação e onde o livro

será guardado. Cada nova ficha é um novo registro, e o INSERT INTO garante que todas as informações essenciais sejam registradas corretamente no sistema da biblioteca (o banco de dados).

Ao final desta operação, um novo registro é adicionado à sua tabela, pronto para ser consultado, atualizado ou, eventualmente, excluído.

3.2.7. R - Read (Ler)

A operação Read é, provavelmente, a mais utilizada em qualquer sistema. Ela nos permite consultar, buscar e visualizar os dados que já estão armazenados no banco de dados. Sempre que você abre um aplicativo e vê informações (suas mensagens, seus produtos favoritos, seu histórico de compras), você está realizando uma operação de Read.

3.2.8. Comando SELECT

No MySQL, o comando principal para ler dados é o SELECT. Ele é extremamente versátil e permite que você especifique exatamente quais colunas deseja ver e de quais tabelas.

3.2.9. Sintaxe Básica:

```sql
SELECT coluna1, coluna2, ...
FROM nome_da_tabela;
```

Ou para selecionar todas as colunas:

```sql
SELECT *
FROM nome_da_tabela;
```

- coluna1, coluna2, ...: As colunas específicas que você deseja visualizar. Se você quiser ver todas as colunas, use o asterisco (*).
- nome_da_tabela: A tabela de onde você deseja buscar os dados.

3.2.10. Exemplo Prático e Explicação Detalhada

Vamos analisar o exemplo do roteiro:

Comando:

```sql
SELECT * FROM alunos;
```

Explicação:

Este comando é bastante direto:

- SELECT *: "Quero selecionar todas as colunas (campos)."
- FROM alunos: "Da tabela chamada alunos."

Ou seja, você está pedindo ao MySQL: "Mostre-me todas as colunas e todos os registros da tabela alunos."

Dica Importante:

Embora SELECT * seja ótimo para testes e para ter uma visão geral dos dados, em aplicações reais, é uma boa prática selecionar apenas as colunas que você realmente precisa. Isso otimiza o desempenho do banco de dados e reduz a quantidade de dados transferidos. Por exemplo, se você quisesse ver apenas o nome e a nota dos alunos, o comando seria:

```sql
SELECT nome, nota FROM alunos;
```

3.2.11. História para ilustrar:

Voltando à nossa biblioteca, o SELECT é como o bibliotecário procurando por um livro específico ou listando todos os livros de um determinado autor. Se ele usa SELECT * FROM livros;, é como se ele pegasse o catálogo inteiro e lesse todas as informações de todos os livros. Mas se ele usa SELECT titulo, autor FROM livros

WHERE gênero = 'Ficção';, ele está sendo mais específico, buscando apenas o título e o autor dos livros de ficção, sem se preocupar com a data de publicação ou o número de prateleira naquele momento.

Após executar um comando SELECT, o banco de dados retorna um conjunto de resultados (uma ou mais linhas) que correspondem à sua consulta, permitindo que você visualize e utilize os dados conforme necessário.

3.2.12. U - Update (Atualizar)

A operação Update é utilizada quando precisamos modificar ou corrigir dados que já existem no banco de dados. Situações comuns incluem a atualização de um endereço de e-mail, a mudança de status de um pedido, ou a correção de um erro de digitação em um nome.

3.2.13. Comando UPDATE

No MySQL, o comando para atualizar registros é o UPDATE. Ele permite que você defina novos valores para uma ou mais colunas em registros específicos.

Sintaxe Básica:

```
UPDATE nome_da_tabela
SET coluna1 = novo_valor1, coluna2 =
novo_valor2, ...
WHERE condicao;
```

- nome_da_tabela: A tabela onde você deseja atualizar os registros.
- SET coluna1 = novo_valor1, ...: Define os novos valores para as colunas especificadas.
- WHERE condição: Esta é a parte mais crítica do comando UPDATE. A condição WHERE especifica quais registros serão atualizados. Se você omitir a cláusula WHERE, todos os registros da tabela serão atualizados, o que pode causar perda de dados irreversível e indesejada.

3.2.14. Exemplo Prático e Explicação Detalhada

Vamos analisar o exemplo do roteiro:

Comando:

```sql
UPDATE alunos SET ativo = FALSE WHERE id =
1;
```

Explicação:

Este comando está realizando a seguinte ação:

- UPDATE alunos: "Quero modificar um registro na tabela alunos."
- SET ativo = FALSE: "Defina o valor da coluna ativo para FALSE (falso)."
- WHERE id = 1: "Apenas para o registro onde o id é igual a 1."

Em outras palavras, estamos marcando o aluno com id = 1 como inativo no banco de dados. Isso é útil, por exemplo, se um aluno trancou a matrícula ou concluiu o curso.

3.2.15. A Importância do WHERE:

Como mencionado, a cláusula WHERE é vital. Se o comando fosse UPDATE alunos SET ativo = FALSE; (sem o WHERE), todos os alunos da tabela seriam marcados como inativos, o que raramente é o desejado. Por isso, sempre use o WHERE com um identificador único (como id) para garantir que apenas o registro correto seja alterado.

História para ilustrar:

Imagine que você tem uma lista de contatos no seu celular. Um amigo mudou de número de telefone. A operação UPDATE é como você abrindo o contato dele e editando o número antigo para o novo. Você não mudaria o número de todos os seus contatos, certo? Você especifica exatamente qual contato quer atualizar. O WHERE é o que te permite encontrar o contato

certo (pelo nome, por exemplo) antes de fazer a alteração.

Após a execução bem-sucedida de um comando UPDATE, os dados no banco de dados são modificados de acordo com as instruções, refletindo as informações mais recentes.

3.2.16. D - Delete (Excluir)

A operação Delete é a última do ciclo de vida dos dados. Ela é utilizada quando precisamos remover um registro do banco de dados. Isso pode acontecer quando um usuário exclui sua conta, um produto é descontinuado, ou uma informação se torna obsoleta.

3.2.17. Comando DELETE

No MySQL, o comando para excluir registros é o DELETE. Assim como o UPDATE, ele depende crucialmente da cláusula WHERE para especificar quais registros devem ser removidos.

Sintaxe Básica:

```sql
DELETE FROM nome_da_tabela
WHERE condicao;
```

- nome_da_tabela: A tabela de onde você deseja excluir os registros.
- WHERE condicao: A condição que especifica quais registros serão excluídos. Se você omitir a cláusula WHERE, todos os registros da tabela serão excluídos, o que é uma ação extremamente perigosa e geralmente indesejada.

Exemplo Prático e Explicação Detalhada

Vamos analisar o exemplo do roteiro:

Comando:

```sql
DELETE FROM alunos WHERE id = 1;
```

Explicação:

Este comando está realizando a seguinte ação:

- DELETE FROM alunos: "Quero apagar um ou mais registros da tabela alunos.
- WHERE id = 1: "Apenas o registro onde o id é igual a 1."

Em resumo, estamos removendo permanentemente o aluno com id = 1 do nosso banco de dados.

A Importância do WHERE (novamente!):

Assim como no UPDATE, a cláusula WHERE é fundamental no DELETE. Um comando DELETE FROM alunos; (sem o WHERE) apagaria todos os alunos da tabela de uma vez, o que seria um erro catastrófico na maioria dos casos. Sempre tenha certeza de que sua cláusula WHERE está correta antes de executar um comando DELETE.

História para ilustrar:

Pense em uma caixa de e-mails. Quando você seleciona um e-mail e clica em "Excluir", você está realizando uma operação DELETE. Você não clica em um botão que apaga todos os seus e-mails de uma vez, a menos que seja essa a sua intenção. Você especifica qual e-mail quer remover. O WHERE é o que garante que apenas o e-mail selecionado seja movido para a lixeira.

Após a execução de um comando DELETE, os registros que correspondem à condição WHERE são permanentemente removidos do banco de dados. É uma ação poderosa e deve ser usada com cuidado.

3.2.18. MySQL Workbench: Preparando o Ambiente

Para colocar em prática tudo o que aprendemos sobre CRUD, utilizaremos o MySQL Workbench, uma ferramenta visual que nos permite interagir com o banco de dados de forma intuitiva. É o ambiente onde você escreverá e executará seus comandos SQL.

Acessando o Serviço Local

Ao abrir o MySQL Workbench, você verá uma tela inicial com suas conexões. Para acessar o nosso serviço local de MySQL, siga estes passos:

1. Com o MySQL Workbench aberto, procure pela seção "MySQL Connections".
2. Clique na conexão "local mysql" (ou o nome que você configurou para sua conexão local).

Este é o seu portal para o banco de dados! Acostume-se com ele, pois será seu principal ambiente de trabalho.

Entendendo a Opção "Save password in vault"

Antes de inserir sua senha para acessar a conexão, você pode ter notado uma opção chamada "Save password in vault". Vamos entender o que ela faz:

- **Significado:** Traduzida para o português, significa "Salvar a senha no cofre".
- **Funcionalidade:** Quando você marca essa caixinha, o MySQL Workbench armazena a senha que você digitou de forma segura no seu computador. Ele utiliza um sistema de armazenamento protegido, conhecido como "vault" (cofre).
- **Benefício:** Na prática, isso significa que você não precisará digitar a senha toda vez que abrir o Workbench e quiser se conectar ao seu MySQL local. O Workbench lembrará da senha para aquela conexão específica, tornando seu acesso mais rápido e conveniente.

Importante: Embora seja conveniente, certifique-se de que seu ambiente de trabalho é seguro antes de usar essa opção, especialmente em computadores compartilhados.

Após ativar essa opção e inserir sua senha (por exemplo, 123456), clique em "OK". Parabéns! Você estará dentro do seu serviço MySQL local, pronto para começar a executar os comandos CRUD em seu banco de dados.

3.2.19. Prática Guiada: CRUD Completo

Vamos realizar uma prática guiada completa, aplicando todas as operações em nossa tabela alunos no MySQL Workbench. Esta seção simula um cenário real de gerenciamento de dados.

Nossa tabela alunos possui as colunas nome (texto), nota (decimal) e ativo (booleano), além de colunas que são preenchidas automaticamente, como id e criacao_em.

3.2.20. Criando um Novo Registro (CREATE)

Cenário: Um novo aluno, "Anderson da Silva", acabou de se matricular. Ele ainda não tem nota e seu status inicial é inativo.

Comando:

```sql
INSERT INTO alunos (nome, nota, ativo)
VALUES (\"Anderson da Silva\", 0, FALSE);
```

Explicação: Estamos inserindo um novo registro na tabela alunos. O nome é "Anderson da Silva", a nota inicial é 0, e o ativo é FALSE, indicando que ele está inativo no momento.

3.2.21. Consultando o Registro Criado (READ)

Cenário: Precisamos verificar se o registro de "Anderson da Silva" foi inserido corretamente e qual o id que o banco de dados atribuiu a ele.

Comando:

```sql
SELECT * FROM alunos;
```

Explicação: Este comando nos permite visualizar todos os registros e todas as colunas da tabela alunos. Você deverá ver o novo registro de "Anderson da Silva" com seu id (provavelmente 2, se for o segundo registro inserido) e a criacao_em preenchida automaticamente.

3.2.22. Atualizando o Registro (UPDATE)

Cenário: "Anderson da Silva" começou a frequentar as aulas e agora precisa ser ativado no sistema.

Comando:

```sql
UPDATE alunos SET ativo = TRUE WHERE id = 2;
```

Explicação: Estamos atualizando o status do aluno com id = 2 (Anderson da Silva) para ativo = TRUE. Lembre-se de usar o id correto do aluno que você criou.

3.2.23. Confirmando a Atualização (READ)

Cenário: Queremos ter certeza de que a atualização do status de "Anderson da Silva" foi efetivada.

Comando:

```sql
SELECT nome, ativo FROM alunos WHERE id = 2;
```

Explicação: Consultamos apenas as colunas nome e ativo para o aluno com id = 2. O resultado deve mostrar "Anderson da Silva" com ativo como TRUE (ou 1).

Excluindo o Registro (DELETE)

Cenário: "Anderson da Silva" precisou se transferir para outra escola e seu registro deve ser removido do nosso sistema.

Comando:

```sql
DELETE FROM alunos WHERE id = 2;
```

Explicação: Este comando remove o registro do aluno com id = 2 da tabela alunos. Cuidado: Esta ação é irreversível.

3.2.24. Confirmando a Exclusão (READ)

Cenário: Precisamos verificar se o registro de "Anderson da Silva" foi realmente excluído.

Comando

```sql
SELECT * FROM alunos;
```

Explicação: Ao executar este comando, você notará que o registro de "Anderson da Silva" não aparece mais na lista, confirmando sua exclusão.

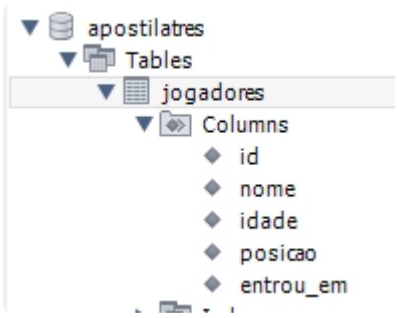
Parabéns! Você acaba de realizar um ciclo completo de CRUD na prática. Isso demonstra o poder e a importância dessas quatro operações no gerenciamento de dados.

3.3. Exercícios práticos

Como exercício prático desta apostila, vamos realizar a utilização do CRUD. Primeiro crie um banco de dados chamado "apostilatres", depois crie uma tabela jogadores com as seguintes colunas:

- id (inteiro, chave primária, auto incremento)
- nome (texto de até 100 caracteres, obrigatório)
- idade (inteiro, obrigatório)
- posicao (texto de até 100 caracteres)
- entrou_em (registro de data e hora com valor padrão automático)

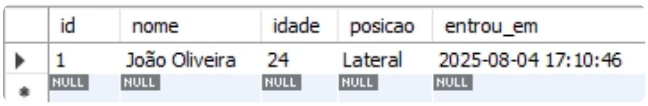
Deverá ficar da seguinte forma abaixo:



Agora que você criou a tabela jogadores, insira um novo jogador com as seguintes informações:

- Nome: João Oliveira
- Idade: 24
- Posição: Lateral
- Data e hora de entrada: deixe o banco preencher automaticamente

Após criar, realize uma consulta e visualize se consta conforme o abaixo:

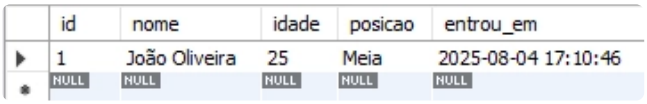


id	nome	idade	posicao	entrou_em
1	João Oliveira	24	Lateral	2025-08-04 17:10:46
NULL	NULL	NULL	NULL	NULL

Agora vamos atualizar os dados do jogador que você acabou de inserir. Atualize o jogador João Oliveira para:

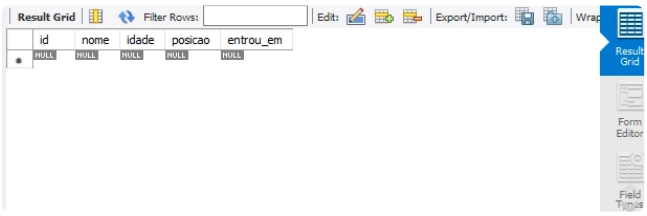
- Idade: 25
- Posição: Meia

Depois de criar consulte se consta atualizado o registro conforme abaixo:



id	nome	idade	posicao	entrou_em
1	João Oliveira	25	Meia	2025-08-04 17:10:46
NULL	NULL	NULL	NULL	NULL

Agora, exclua o jogador e depois consulte para verificar se foi excluído. Deverá ficar em branco, sem registro conforme abaixo:



3.4. Conclusão

Chegamos ao final da nossa aula sobre CRUD em MySQL! Esperamos que você tenha compreendido a importância dessas quatro operações fundamentais: Criar, Ler, Atualizar e Excluir. Elas são a base para qualquer interação com bancos de dados e, ao dominar esses conceitos, você estará muito mais preparado para desenvolver aplicações robustas e eficientes.

Lembre-se que a prática leva à perfeição. Continue experimentando os comandos no MySQL Workbench, crie suas próprias tabelas, insira dados, consulte-os de diferentes formas, atualize informações e, com cautela, pratique a exclusão de registros. Quanto mais você praticar, mais natural se tornará o uso do SQL e dos conceitos de banco de dados.

Anotações