SQL – LINGUAGEM PARA BANCO DE DADOS

Elabora por: Ronan Adriel Zenatti

SÃO PAULO

2025

SUMÁRIO

INTRODUÇÃO E INTEGRAÇÃO

Caro(a) Aluno(a),

É com grande satisfação que lhe damos boas-vindas ao **Curso de Qualificação Profissional Básica da FAT, Fundação de Apoio a Tecnologia**, entidade que desde 1987 colabora com as instituições que atuam nas áreas da educação e da tecnologia, buscando estimular e desenvolver o conhecimento através de programas de geração, difusão e transferência de tecnologia.

A Fundação FAT possui *know-how* e histórico de atuação em projetos nas áreas de gestão, logística, tecnologia da informação, construção civil, meio ambiente, transporte e educação corporativa. Todos os projetos são desenvolvidos com base nas reais necessidades das organizações com as quais mantém relacionamento.

Estamos iniciando mais um processo de educação que, como todos os outros, precisam ser contínuos na busca de novos conhecimentos, fazer novas amizades, crescer profissionalmente e pessoalmente. Nosso maior compromisso é viabilizar experiências nas diversas áreas profissionais, proporcionar uma visão do mercado de trabalho altamente competitivo, tanto do olhar técnico quanto ético e humanístico, acreditando sempre que a relação ensino-aprendizagem deve constituir uma experiência estimulante, produtiva e prazerosa.

A partir de agora, seu objetivo principal deverá ser o de aprender a aprender e, é necessário que você realmente queira aprender bem, pois veremos que aprender não é apenas memorizar textos e livros. A meta será a de entender o conteúdo que está sendo desenvolvido durante as aulas, a aplicação à nossa realidade, relacionar

o novo com o antigo e projetar aos nossos sonhos.

Vale ressaltar que estudar nunca foi tarefa fácil. Precisaremos de muita dedicação e persistência. Mas também vale ressaltar que você nunca estará sozinho nessa caminhada, pois sempre estará acompanhado de uma equipe preparada, pronta e disposta para lhe ajudar no que for preciso e necessário para alcançar os objetivos do curso em relação à sua formação.

Vamos começar à nossa jornada de aprendizado em SQL. Este material foi cuidadosamente elaborado para acompanhá-lo(a) passo a passo, desde os fundamentos teóricos até a construção de consultas poderosas.

MÓDULO 1: FUNDAMENTOS E AMBIENTE - O PONTO DE PARTIDA NO MUNDO DOS DADOS

Neste primeiro módulo, construiremos a base sobre a qual todo o seu conhecimento em SQL será edificado. Compreenderemos o "porquê" por trás dos bancos de dados relacionais antes de mergulharmos no "como". Ao final, você terá um ambiente de trabalho totalmente funcional e entenderá as boas práticas que distinguem um profissional de dados.

Capítulo 1: O Universo dos Dados Relacionais

1.1 Boas-vindas ao Mundo dos Dados

Antes de escrevermos nossa primeira linha de código SQL, é crucial entendermos o cenário em que ele opera. Vivemos na era da informação, onde dados são gerados a cada segundo. Desde a sua série favorita em um serviço de streaming até a transação de compra em um supermercado, tudo é registrado, armazenado e, o mais importante, relacionado.

O SQL, ou *Structured Query Language* (Linguagem de Consulta Estruturada), é a linguagem universal que nos permite conversar com esses grandes volumes de dados, extraindo informações valiosas e insights de negócio. Ele não é uma linguagem de programação como Python ou Java, mas sim uma linguagem de consulta, especializada em uma única tarefa: comunicar-se com bancos de dados.

1.2 O que é um Sistema Gerenciador de Banco de Dados (SGBD)?

Permita-me aprofundar a definição. Um Sistema Gerenciador de Banco de Dados (SGBD) é um conjunto robusto de softwares que atua como uma interface inteligente entre o usuário (ou uma aplicação) e o banco de dados físico (os arquivos onde os dados são efetivamente guardados). Ele abstrai toda a complexidade de

como e onde os dados são armazenados, permitindo que nos concentremos no que realmente importa: a informação em si.

As **funções primordiais** de um SGBD moderno são:

- **Definição de Dados:** Fornecer uma linguagem (como a DDL - *Data Definition Language* do SQL) para criar e definir a estrutura do banco, as tabelas e os relacionamentos.
- **Manipulação de Dados:** Permitir a inserção, atualização, consulta e exclusão de dados por meio de uma linguagem de manipulação (como a DML - *Data Manipulation Language* do SQL).
- **Controle de Acesso e Segurança:** Gerenciar permissões de usuários, garantindo que apenas pessoas autorizadas possam ler ou modificar determinados dados.
- **Garantia de Integridade e Consistência:** Aplicar regras para assegurar que os dados permaneçam válidos e consistentes ao longo do tempo.
- **Controle de Concorrência:** Gerenciar múltiplos acessos simultâneos ao banco de dados, evitando que uma transação interfira indevidamente em outra.
- **Recuperação de Falhas:** Oferecer mecanismos de backup e restauração para proteger os dados contra falhas de hardware ou software.

1.2.1 Os Pilares da Confiabilidade: O que é ACID?

Para garantir a integridade em operações críticas (como uma transação financeira ou a finalização de uma compra), a maioria dos SGBDs relacionais opera sob os princípios **ACID**, um acrônimo que representa quatro garantias essenciais:

A - Atomicidade (*Atomicity*): Uma transação é uma unidade de trabalho "tudo ou nada". Ou todas as operações dentro dela são concluídas com sucesso, ou

nenhuma delas é. Se uma falha ocorrer no meio do caminho, o sistema reverte todas as alterações feitas, como se nada tivesse acontecido.

- ***Exemplo**:* Em uma transferência bancária, a operação de debitar da conta A e creditar na conta B deve ser atômica. Se o débito ocorrer, mas o crédito falhar, a atomicidade garante que o débito seja desfeito.

C - Consistência (*Consistency*): A transação deve levar o banco de dados de um estado válido para outro. Ela não pode violar as regras de integridade definidas (como chaves primárias ou restrições). O SGBD garante que, após a transação, os dados permaneçam consistentes.

- ***Exemplo**:* O sistema não pode permitir que uma transação resulte em um saldo de conta negativo se houver uma regra que o proíba.

I - Isolamento (*Isolation*): Transações concorrentes (executadas ao mesmo tempo) não devem interferir umas nas outras. Do ponto de vista de uma transação, é como se ela estivesse sendo executada sozinha no sistema, garantindo previsibilidade.

- ***Exemplo**:* Se duas pessoas tentam comprar o último ingresso para um show ao mesmo tempo, o isolamento garante que apenas uma delas conseguirá completar a transação com sucesso.

D - Durabilidade (*Durability*): Uma vez que uma transação é confirmada com sucesso (*commit*), suas alterações são permanentes e não serão perdidas, mesmo em caso de falha do sistema (como uma queda de energia).

- ***Exemplo**:* Após receber a confirmação de que sua compra online foi aprovada, a durabilidade garante que esse registro de compra persistirá no banco de dados.

1.2.2 Principais SGBDs do Mercado em 2025

O mercado de SGBDs é vasto e diversificado. Embora nosso foco seja o MySQL, é importante conhecer os principais atores:

- **MySQL:** (Que será utilizado neste curso) SGBD relacional de código aberto, extremamente popular para aplicações web e parte da famosa pilha de tecnologia LAMP (Linux, Apache, MySQL, PHP). Mantido pela Oracle.
- **PostgreSQL:** SGBD relacional de código aberto conhecido por sua robustez,

extensibilidade e forte aderência aos padrões SQL. É uma alternativa muito popular ao MySQL, especialmente em aplicações que exigem alta complexidade de dados.

- **Microsoft SQL Server:** SGBD relacional desenvolvido pela Microsoft, amplamente utilizado no ambiente corporativo, com forte integração ao ecossistema Windows e à plataforma de nuvem Azure.
- **Oracle Database:** Um dos SGBDs relacionais mais poderosos e completos do mercado, conhecido por sua performance e segurança em aplicações de missão crítica em grandes corporações. É um produto comercial robusto.
- **SQLite:** Um SGBD relacional extremamente leve, contido em um único arquivo, muito utilizado em aplicações embarcadas, como celulares e navegadores web.
- **SGBDs NoSQL (Não Apenas SQL):** Vale mencionar que existe uma outra categoria de bancos de dados, os NoSQL, que não seguem o modelo relacional e são usados para casos específicos (grandes volumes de dados não estruturados etc.). Exemplos incluem **MongoDB** (orientado a documentos) e **Redis** (chave-valor em memória).

1.3 O Modelo Relacional: A Espinha Dorsal do SQL

Proposto por Edgar F. Codd em 1970, o modelo relacional organiza os dados em tabelas (relações), que se assemelham a planilhas com uma estrutura rígida. A beleza deste modelo reside na sua simplicidade e base matemática.

- **Tabelas (Relações):** Uma tabela é a estrutura central de armazenamento em um banco de dados relacional. Imagine-a como um arquivo ou uma planilha dedicada a um único assunto específico. Cada tabela em um banco de dados é projetada para guardar informações sobre um tipo de entidade do mundo real, como pessoas, produtos, pedidos ou locais. A regra fundamental é: uma tabela, um assunto. **Ex**: alunos, matriculas, professores, materias.
- **Colunas (Atributos):** As colunas, também conhecidas como atributos, são as divisões verticais de uma tabela. Elas definem quais tipos de informação serão armazenados para cada registro naquela tabela. Cada coluna possui um nome único (seguindo o padrão snake_case) e um tipo de dado específico (como

texto, número, data etc.), que funciona como uma regra sobre o que pode ser inserido ali. **Ex**: primeiro_nome, data_nascimento, celular.

- **Registros (Linhas ou Tuplas):** Um registro, também chamado de linha ou tupla, representa uma única ocorrência da entidade que a tabela descreve. É um conjunto completo de dados para um item específico, preenchendo cada uma das colunas da tabela. Se a tabela é o formulário em branco, o registro é o formulário totalmente preenchido com as informações de uma única pessoa ou item.

Na imagem abaixo temos a tabela *produtos*, onde o campo *preco_unitario* utiliza o ponto como separador decimal, pois este é o padrão universal ANSI SQL para armazenamento de dados, garantindo compatibilidade e evitando erros de sintaxe entre diferentes sistemas de banco de dados; da mesma forma, o formato de data padrão em SQL é 'AAAA-MM-DD' (ano-mês-dia), cabendo à camada de aplicação (o software ou site) a responsabilidade de formatar esses valores para o padrão local brasileiro (com vírgula e no formato DD/MM/AAAA) ao exibi-los.

produtos

id_produto	nome_produto	vencimento	preco_unitario	quantidade_estoque	unidade
101	Chocolate	2025-12-31	1.50	250	UN
102	Carne	2025-10-02	15.75	120	KG
103	Chiclete	2026-12-31	0.80	400	UN

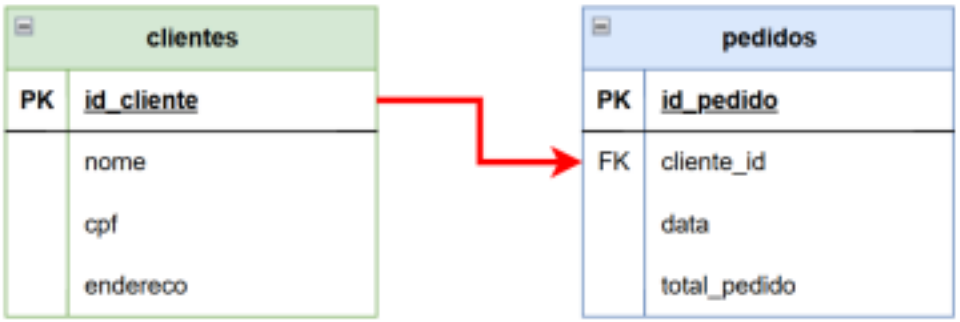
- **Chave Primária (*Primary Key*):** É uma ou mais colunas que identificam de forma única cada linha em uma tabela, geralmente chamadas de código ou ID.

Não pode haver duas linhas com a mesma chave primária. **Ex**: é como o CPF de uma pessoa no cadastro de cidadãos. É um número único que garante que não existem duas pessoas com a mesma identificação. É a identidade absoluta e inconfundível daquela pessoa naquele cadastro.

- **Chave Estrangeira (*Foreign Key*):** É uma coluna em uma tabela que estabelece um vínculo com a chave primária de outra tabela, criando assim o **relacionamento**. **Ex**: é como o campo "CPF do solicitante" em um formulário de "Solicitação de Passaporte". Esse campo não identifica o formulário, mas cria um **vínculo poderoso** com o cadastro de cidadãos. Ele garante que o passaporte só pode ser emitido para alguém que *realmente existe* no sistema.

Na imagem abaixo vemos um diagrama detalhado mostrando a tabela

clientes com a coluna *id_cliente* (chave primária) e a tabela *pedidos* com a coluna *id_pedido*



(chave primária) e *cliente_id* (chave estrangeira, que se relaciona com *id_cliente* na tabela *clientes*).

1.4 Análise de Casos de Uso Reais

Para solidificar esses conceitos, vamos analisar como o modelo relacional se aplica a negócios que você conhece:

Cenário de E-commerce:

• **Entidades:** *clientes*, *produtos*, *pedidos*, *categorias*, *itens_pedido*. •

Relacionamento:

- Um **cliente** (identificado por *id_cliente*) pode fazer múltiplos ***pedidos***.
- A tabela ***pedidos*** armazena o *cliente_id* (chave estrangeira) para saber quem fez a compra.
- Um pedido pode conter múltiplos ***produtos***. Como este é um relacionamento de muitos-para-muitos, criamos uma tabela associativa ***itens_pedido*** que conecta um *pedido_id* a um *produto_id*.

Exercício de Fixação: Pense em uma rede social que você utiliza. Quais seriam as principais entidades (tabelas) que a compõem? Liste pelo menos três tabelas, suas

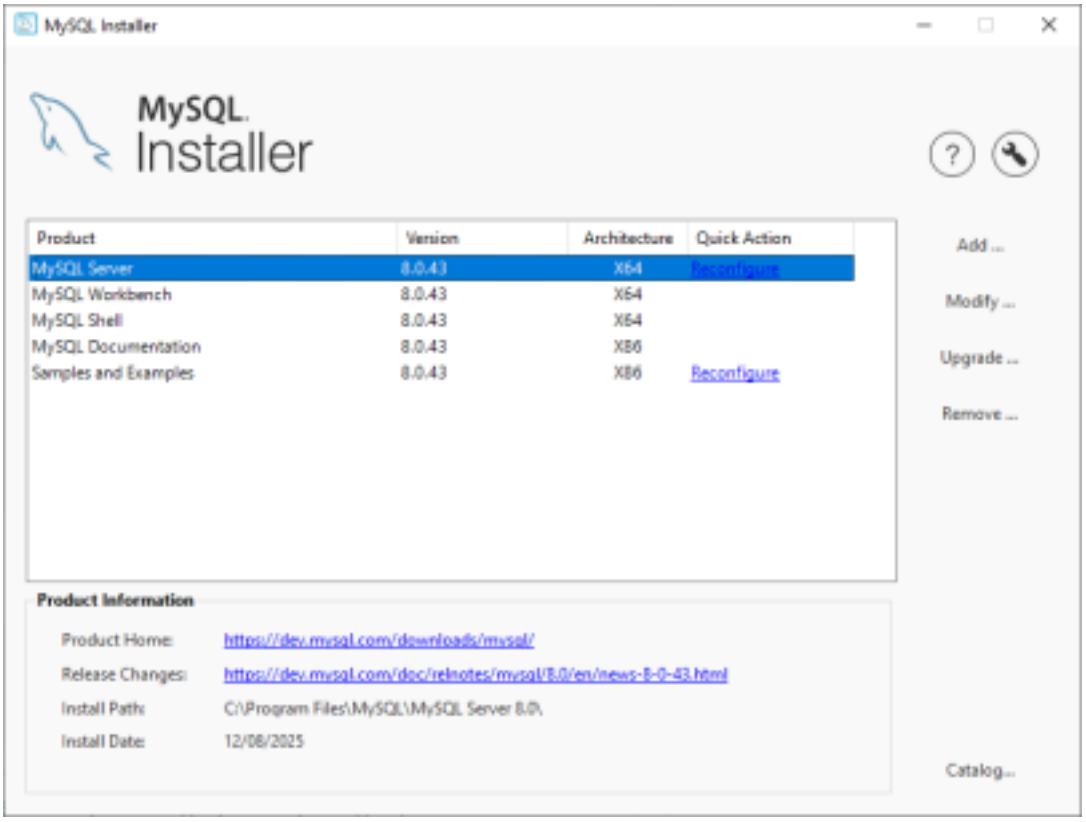
prováveis chaves primárias e algumas chaves estrangeiras que criariam os relacionamentos entre elas.

Capítulo 2: Configurando seu Computador e Boas Práticas

2.1 Instalando o MySQL Server e o MySQL Workbench

Para começarmos a praticar, precisamos de duas ferramentas essenciais: o **MySQL Server** (o SGBD em si) e o **MySQL Workbench** (a ferramenta cliente gráfica para interação). O processo de instalação é guiado e intuitivo através do "MySQL Installer", disponível no site oficial. Durante a instalação, será solicitado que você defina uma senha para o usuário administrador (root), **guarde esta senha com segurança, pois ela será necessária para todas as conexões.**

Na imagem abaixo temos o instalador do MySQL com todos os produtos que podem ser instalados.

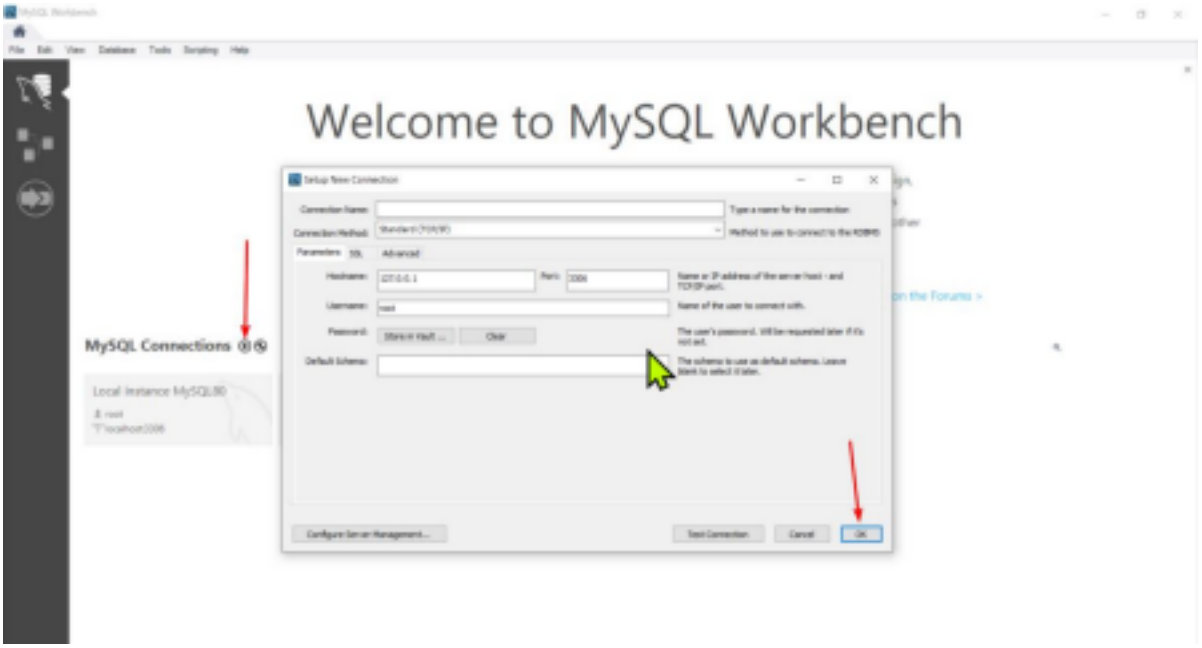


2.2 A Primeira Conexão via Interface Gráfica (Workbench)

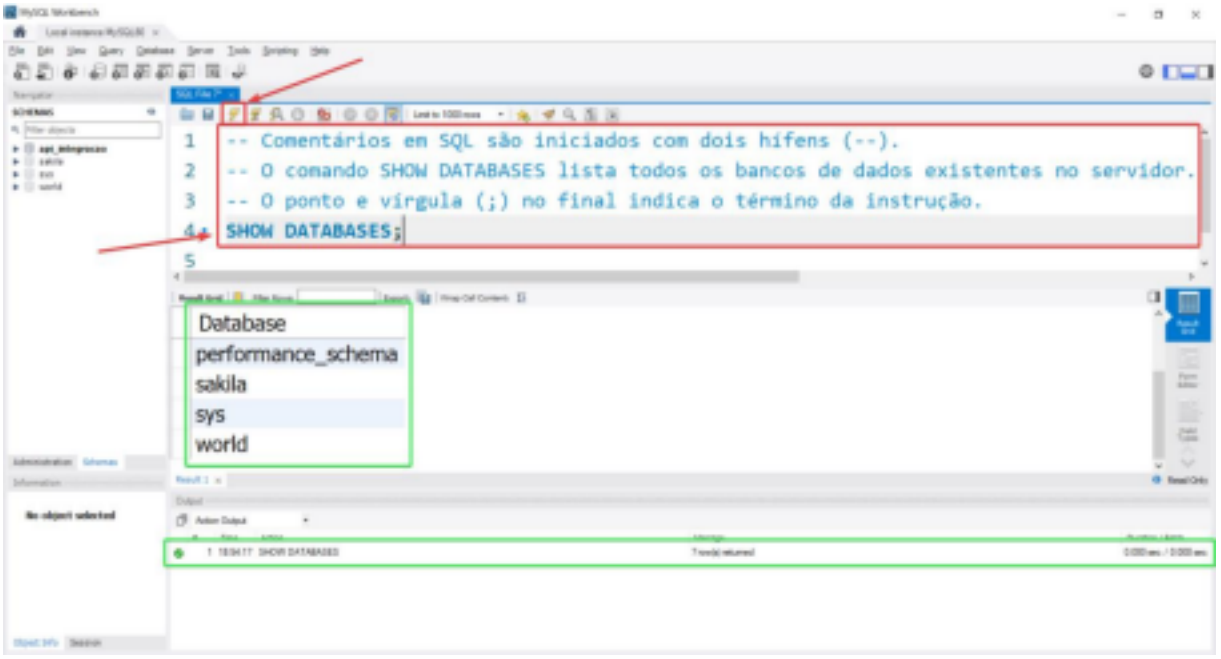
Com tudo instalado, abra o MySQL Workbench. Para se conectar a um banco de dados, um cliente (como o Workbench) precisa de informações essenciais para localizar e autenticar-se no servidor. Vamos decifrar cada um desses parâmetros:

- **Hostname (ou Host):** É o endereço do computador onde o SGBD (MySQL Server) está sendo executado. Para uma instalação local, como a nossa, usamos o endereço de loopback: 127.0.0.1 ou seu alias localhost. Ambos significam "esta própria máquina".
- **Port (Porta):** Imagine o Hostname como o endereço de um prédio. A porta é o número do apartamento. O MySQL, por padrão, "ouve" na porta 3306.
- **Username (Usuário):** O nome de usuário para se autenticar no SGBD. Na instalação inicial, criamos o superusuário root, que tem permissão total.
- **Password (Senha):** A senha que você definiu para o usuário durante a instalação do MySQL Server.

Na imagem abaixo vemos a tela inicial de conexão do MySQL WorkBench, uma conexão já deve estar configurada por padrão pelo instalador do MySQL, caso necessário crie uma conexão clicando sobre o símbolo de "**+**", na tela que irá exibir, já trará os dados de conexão padrão sendo necessário apenas adicionar a senha de conexão do usuário root.



Após conectar-se, você terá acesso ao editor de SQL. Vamos executar nosso primeiro comando para verificar se tudo está funcionando, digite o comando `SHOW DATABASES;` e clique sobre o botão de executar que tem o ícone de um raio amarelo conforme imagem abaixo.



O resultado será uma lista de bancos de dados padrão do sistema, conforme contorno verde, confirmando que sua instalação e conexão aconteceram com sucesso!

2.3 A Arte da Nomenclatura: Escrevendo um SQL Profissional

Um código limpo e padronizado não é um luxo, mas uma necessidade no desenvolvimento profissional. Ele reduz a carga cognitiva, facilita o trabalho em equipe e torna a manutenção do sistema muito mais simples.

Ao longo desta apostila, seguiremos rigorosamente as seguintes convenções que são amplamente utilizadas pela comunidade *open-source*, mas note que podem conforme o SGBD ou empresa:

- **Nomes de tabelas, colunas e variáveis:** Utilize snake_case (minúsculas com *underscore*) e SEMPRE sem acentos ou caracteres especiais.
  - **Certo:** primeiro_nome, data_nascimento, cpf

- **Comandos SQL:** Sempre em **MAIÚSCULO**.
  - **Certo:** `SELECT primeiro_nome FROM clientes;`
- **Nomes de tabelas:** Sempre no **plural**, pois a tabela representa um conjunto de dados.
  - **Certo:** client**es**, produt**os**, pedid**os**
- **Chave Primária (*Primary Key*):** A chave primária deve ser nomeada com o

prefixo **id_** e o nome da tabela no **singular**, por representar apenas 1 dados da tabela, **Ex**: id_nome_tabela_no_singular.

- **Certo:** id_cliente (na tabela clientes)

- **Chave Estrangeira (*Foreign Key*):** A chave estrangeira deve ser nomeada com o nome da tabela que ela referencia (no **singular**), seguido do sufixo **_id**. **Ex**: nome_tabela_referencia_id.

  - **Certo:** cliente_id (na tabela pedidos, referenciando clientes)

Exercício de Fixação: Utilizando o MySQL Workbench ou o terminal, execute o comando SQL para criar um banco de dados chamado meu_primeiro_banco. Em seguida, execute novamente o comando para listar todos os bancos de dados e confirme que o seu foi criado com sucesso.

Módulo 1: QUESTÕES DE AVALIAÇÃO

Questão 1 Uma startup está desenvolvendo um novo aplicativo de e-commerce. Durante uma transação de venda, o sistema precisa garantir que a criação do registro do pedido e a baixa no estoque do produto ocorram como uma única operação indivisível. Se a baixa no estoque falhar por qualquer motivo, o registro do pedido também não deve ser criado. Qual propriedade do conceito ACID está diretamente relacionada a essa garantia de "tudo ou nada"?

A) Durabilidade (Durability)

B) Isolamento (Isolation)

C) Consistência (Consistency)

D) Atomicidade (Atomicity)

Questão 2 Um analista de dados júnior está explorando a estrutura de um banco de dados pela primeira vez. Ele se depara com a tabela pedidos, que contém uma coluna cliente_id. Ele percebe que cada valor nesta coluna corresponde a um valor existente na coluna id_cliente da tabela clientes. Qual conceito do modelo relacional a coluna cliente_id representa na tabela pedidos?

A) Chave Primária (Primary Key)

B) Esquema (Schema)

C) Chave Estrangeira (Foreign Key)

D) Atributo (Attribute)

Questão 3 Uma equipe de desenvolvimento está definindo os padrões de nomenclatura para um novo projeto de banco de dados. Eles precisam nomear a tabela que armazenará as informações dos usuários e a sua chave primária. Seguindo as boas práticas apresentadas na apostila, qual das seguintes opções está correta?

A) Tabela: Usuario, Chave Primária: id

B) Tabela: usuarios, Chave Primária: id_usuario

C) Tabela: USUARIOS, Chave Primária: ID_USUARIO

D) Tabela: tb_usuarios, Chave Primária: usuarios_id

Questão 4 Um desenvolvedor acaba de instalar o MySQL Server em sua máquina local e está se conectando pela primeira vez através do MySQL Workbench. Ele precisa preencher os parâmetros de conexão. Considerando uma instalação padrão, qual combinação de Hostname e Port ele deve utilizar?

A) Hostname: mysql.server.com, Port: 3306

B) Hostname: 192.168.0.1, Port: 8080

C) Hostname: localhost, Port: 3306

D) Hostname: meu_computador, Port: 1433

Questão 5 Um SGBD moderno executa diversas funções cruciais para a gestão de dados. Uma empresa precisa garantir que apenas gerentes possam visualizar a coluna de salários na tabela de funcionários. Qual função primordial de um SGBD é responsável por gerenciar esse tipo de regra?

A) Definição de Dados

B) Controle de Acesso e Segurança

C) Recuperação de Falhas

D) Manipulação de Dados

Questão 6 No modelo relacional, os dados são organizados em uma estrutura tabular. Se a tabela alunos representa a entidade, e a coluna primeiro_nome representa uma de suas características, o que uma linha completa nessa tabela (contendo o ID, nome, data de nascimento, etc., de um aluno específico) representa?

A) Uma Relação (Relation)

B) Um Esquema (Schema)

C) Uma Tupla (ou Registro)

D) Um SGBD

Questão 7 Um desenvolvedor está trabalhando via terminal e precisa se conectar ao servidor MySQL local com o usuário root para realizar tarefas administrativas. Qual é o comando mais comum e seguro para iniciar essa conexão, que solicitará a senha de forma interativa?

A) mysql -h localhost -u root

B) mysql connect root

C) mysql -u root -p

D) mysql --user=root --password=minhasenha

Questão 8 SQL (*Structured Query Language*) é a linguagem padrão para interagir com bancos de dados relacionais. Com base na sua finalidade principal, como o SQL é classificado?

A) Uma linguagem de programação de baixo nível, como Assembly.

B) Uma linguagem de marcação, como HTML.

C) Uma linguagem de programação de propósito geral, como Python.

D) Uma linguagem de consulta, especializada em se comunicar com bancos de dados.

Questão 9 Uma empresa de desenvolvimento de software está escolhendo um

SGBD para um novo projeto de grande porte que exige alta complexidade de dados e forte aderência aos padrões SQL. Eles estão buscando uma alternativa de código aberto ao MySQL. Com base nas descrições da apostila, qual SGBD seria a escolha mais provável?

A) Microsoft SQL Server

B) SQLite

C) PostgreSQL

D) MongoDB

Questão 10 Após se conectar com sucesso ao banco de dados pelo MySQL Workbench, um DBA precisa verificar quais bancos de dados já existem no servidor antes de criar um. Qual comando SQL ele deve executar para obter essa lista?

A) LIST DATABASES;

B) SHOW DATABASES;

C) SELECT DATABASES;

D) DESCRIBE DATABASES;

MÓDULO 2: A ARQUITETURA DOS DADOS - MODELAGEM RELACIONAL

Seja bem-vindo(a) ao Módulo 2. Neste ponto, já compreendemos o que é um SGBD, o papel do MySQL e como interagir com ele. Agora, daremos um passo para trás do código para nos concentrarmos em uma fase de planejamento essencial: a **modelagem de dados**.

Pense em um arquiteto antes de construir um arranha-céu. Ele não começa a assentar tijolos aleatoriamente. Primeiro, ele desenha uma planta detalhada, considerando a fundação, a estrutura, a distribuição dos cômodos e como tudo se conecta. A modelagem de dados é a nossa versão dessa planta. Um modelo bem planejado é a diferença entre um sistema de dados sólido, que pode crescer e se adaptar, e um sistema frágil, propenso a erros e difícil de manter.

Este módulo oferece uma introdução à modelagem de dados. Aqui, vocês aprenderão os conceitos básicos para traduzir requisitos de negócio em um diagrama lógico e estruturado, pronto para ser implementado em um banco de dados

relacional. Por não ser o foco principal do curso, recomendamos que, em caso de dificuldades, você procure um material de estudo complementar para garantir que não restem lacunas em seu conhecimento.

Capítulo 3: O Modelo Conceitual -

3.1 O que é o Modelo Entidade-Relacionamento (MER)?

O Modelo Entidade-Relacionamento, é a nossa principal ferramenta na fase de modelagem conceitual. Ele é uma representação visual e abstrata dos dados, que nos permite descrever as informações de um sistema de forma independente de qualquer SGBD específico.

O objetivo do MER é criar uma linguagem comum entre os desenvolvedores, os analistas de negócio e os clientes, garantindo que todos tenham a mesma compreensão sobre quais dados serão armazenados e como eles se interligam.

3.2 Os Componentes Fundamentais do MER

Um diagrama MER é composto por três elementos principais:

Entidades: Uma entidade representa um objeto do mundo real sobre o qual desejamos guardar informações. É algo concreto ou abstrato que possui existência própria. Geralmente, entidades se tornam as tabelas do nosso banco de dados.

- ***Exemplos****:* alunos, cursos, funcionarios, produtos.

Atributos: Atributos são as propriedades ou características que descrevem uma entidade. Eles se tornarão as colunas da nossa tabela.

- ***Exemplos****:* Para a entidade alunos, os atributos poderiam ser nome, data_nascimento, cpf.

Os atributos podem ser classificados em:

- **Simples:** Atômicos, não podem ser divididos. Ex: cpf.
- **Compostos:** Podem ser divididos em partes menores. Ex: o atributo endereco pode ser composto por rua, numero, cidade.

- **Monovalorados:** Possuem apenas um valor para cada entidade. Ex: data_nascimento.
- **Multivalorados:** Podem possuir múltiplos valores para a mesma entidade. Ex: um aluno pode ter vários telefones.
- **Identificador (ou Atributo-Chave):** É o atributo que identifica unicamente cada registro de uma entidade. No diagrama, seu nome é **sublinhado**. Ex: cpf ou um id_aluno.

Relacionamentos: Um relacionamento representa a associação ou interação que existe entre duas ou mais entidades. Eles descrevem como as entidades se conectam no mundo real.

- ***Exemplos****:* Um aluno **SE-INSCREVE-EM** um curso. Um funcionario **ESTÁ ALOCADO-EM** um departamento.

3.3 A Escolha dos Tipos de Dados (Modelo Físico)

O Modelo Físico é o passo final, onde definimos os detalhes de implementação específicos do SGBD que escolhemos (no nosso caso, o MySQL). A principal decisão aqui é a escolha dos **tipos de dados** para cada coluna, visando otimizar o armazenamento e garantir a integridade.

TIPO DE DADO	USO COMUM
`INT`	Números inteiros (ex: id_cliente, quantidade).
`VARCHAR(n)`	Textos de tamanho variável até n caracteres (ex: nome).
`TEXT`	Textos longos (ex: descricao_produto).
`DECIMAL(p,s)`	Números decimais exatos (ex: preco, salario).
`DATE`	Apenas datas (AAAA-MM-DD).
`DATETIME`	Datas e horas (AAAA-MM-DD hh:mm:ss).
`TIME`	Apenas horas (hh:mm:ss)
`BOOLEAN`	Valores verdadeiros ou falsos (TRUE/FALSE ou 1/0).

3.4 Mãos à Obra: Modelando uma Biblioteca

Vamos aplicar a teoria a um cenário prático. Imagine que precisamos criar um

sistema para gerenciar uma pequena biblioteca.

1. **Identificando as Entidades:** Quais são os principais "objetos" de negócio aqui?
    - livros
    - autores
    - leitores (as pessoas que pegam livros emprestados)
    - emprestimos (o ato de um leitor pegar um livro)

2. **Definindo os Atributos:** O que precisamos saber sobre cada entidade?
    - livros: isbn (identificador), titulo, ano_publicacao.
    - autores: id_autor (identificador), primeiro_nome, ultimo_nome.
    - leitores: id_leitor (identificador), nome_completo, email.
    - emprestimos: data_emprestimo, data_devolucao_prevista.

3. **Estabelecendo os Relacionamentos:** Como eles se conectam?
    - Um autor **ESCREVE** um livro.
    - Um leitor **REALIZA** um emprestimo.
    - Um emprestimo **CONTÉM** um livro.

Capítulo 4: A Lógica dos Relacionamentos - Cardinalidade e Requisitos de Negócio

No capítulo anterior, estabelecemos as fundações do nosso modelo conceitual. Agora, vamos nos aprofundar no elemento que injeta lógica e regras de

negócio em nossa estrutura: a **cardinalidade**. Compreender e definir a cardinalidade corretamente é, talvez, a etapa mais crítica da modelagem, pois ela dita como os dados podem e não podem interagir.

4.1 A Arte de Fazer as Perguntas Certas: Como Levantar a Cardinalidade

A cardinalidade não é uma decisão do modelador de dados; ela é um **reflexo das regras de negócio**. Para descobri-la, você precisa investigar, questionar e entender o processo que está modelando. A técnica mais eficaz é um "interrogatório" de duas vias entre duas entidades que você acredita que se relacionam.

Para qualquer par de entidades (Entidade A e Entidade B), faça as seguintes perguntas a si mesmo ou, idealmente, a um especialista no assunto (seu cliente ou analista de negócio):

1. **Pergunta 1 (Sentido A → B):** "Um registro da **Entidade A** pode se relacionar com quantos registros da **Entidade B**?"
    - A resposta será sempre em termos de um mínimo e um máximo. Ex: "no mínimo 0 e no máximo Muitos" ou "exatamente 1".
2. **Pergunta 2 (Sentido B → A):** "E um registro da **Entidade B** pode se relacionar com quantos registros da **Entidade A**?"
    - Novamente, a resposta definirá um mínimo e um máximo.

A combinação das respostas máximas de ambas as perguntas (1 ou Muitos/N) definirá o tipo de relacionamento: 1:1, 1:N ou N:M.

4.2 Decifrando os Tipos de Cardinalidade com Exemplos Detalhados

Vamos aplicar nossa técnica de interrogatório a cenários de negócio mais robustos.

Relacionamento Um-para-Um (1:1)

Este relacionamento implica uma ligação exclusiva entre dois registros. É menos comum, mas vital quando ocorre.

Cenário de Negócio:

Uma empresa precisa armazenar dados biométricos de seus funcionários para controle de acesso.

Cada funcionário tem um conjunto único de dados biométricos (impressão digital), e cada conjunto de dados pertence inequivocamente a um único funcionário.

- **Entidades:** funcionarios e dados_biometricos.
- **Levantando os Requisitos (Interrogatório):**

1. **Pergunta 1 (Funcionário → Biometria):** "Um funcionario pode ter quantos conjuntos de dados_biometricos?"
    - **Resposta:** Exatamente **1**. Não mais, não menos.

2. **Pergunta 2 (Biometria → Funcionário):** "E um conjunto de dados_biometricos pode pertencer a quantos funcionarios?"
    - **Resposta:** Exatamente **1**.

Conclusão:

A combinação das respostas máximas (1 e 1) nos dá um relacionamento **1:1**.

A modelagem disso geralmente é feita colocando a chave estrangeira em uma das tabelas, com uma restrição UNIQUE para garantir a exclusividade.

Relacionamento Um-para-Muitos (1:N)

O pilar da maioria dos bancos de dados relacionais. É uma ligação entre duas tabelas onde **um registro** de uma Tabela A pode estar conectado a **vários registros** de uma Tabela B. No entanto, cada registro da Tabela B só pode estar conectado a **um único registro** da Tabela A.

Pense na relação entre uma **Mãe** e seus **Filhos**:

- Uma mãe (o lado "um") pode ter vários filhos (o lado "muitos").
- Cada filho (do lado "muitos") tem apenas uma mãe (o lado "um").

Cenário de Negócio:

Uma loja de eletrônicos vende produtos de diversas marcas (fabricantes).

Cada produto em seu catálogo é fabricado por uma única marca, mas uma mesma marca pode fabricar diversos produtos diferentes.

- **Entidades:** fabricantes e produtos.
- **Levantando os Requisitos (Interrogatório):**

1. **Pergunta 1 (Fabricante → Produto):** "Um fabricante pode ter quantos produtos em seu portfólio?"
    - **Resposta:** No mínimo 1 (se ele está no nosso sistema, deve ter ao menos um produto) e no máximo **Muitos (N)**.

2. **Pergunta 2 (Produto → Fabricante):** "E um produto pode ser fabricado por quantos fabricantes?"
    - **Resposta:** Exatamente **1**, conforme a regra de negócio.

Conclusão:

A combinação das respostas máximas (N e 1) nos dá um relacionamento **1:N**.

O lado "1" (fabricantes) "doa" sua chave primária (id_fabricante) para o lado "N" (produtos), onde ela se torna a chave estrangeira fabricante_id.

Relacionamento Muitos-para-Muitos (N:M)

Ocorre quando registros de ambos os lados podem ter múltiplas associações.

Cenário de Negócio:

Uma universidade gerencia a matrícula de alunos em suas turmas.

Um aluno pode se inscrever em várias turmas por semestre, e uma turma, por sua vez, é composta por vários alunos.

- **Entidades:** alunos e turmas.
- **Levantando os Requisitos (Interrogatório):**

1. **Pergunta 1 (Aluno → Turma):** "Um aluno pode se inscrever em

quantas turmas?"

- **Resposta:** No mínimo 1 (para ser considerado ativo) e no máximo **Muitas (M)**.

2. **Pergunta 2 (Turma → Aluno):** "E uma turma pode ter quantos alunos?"
   - **Resposta:** No mínimo alguns (para a turma existir) e no máximo **Muitos (N)**.

- **Conclusão:** A combinação (M e N) nos dá um relacionamento **N:M**. Como vimos, isso não pode ser implementado diretamente.
  - **A Solução: A Entidade Associativa:** Criamos uma entidade no meio do caminho, que representa o evento da matrícula. Vamos chamá-la de matriculas.
    - O relacionamento entre alunos e matriculas agora é **1:N** (um aluno pode ter muitas matrículas).
    - O relacionamento entre turmas e matriculas também é **1:N** (uma turma pode ter muitas matrículas).
  - A tabela matriculas terá, no mínimo, as chaves estrangeiras de ambas as tabelas (aluno_id, turma_id), que juntas formarão sua chave primária composta. Ela também pode ter atributos próprios, como data_matricula ou nota_final.

Capítulo 5: Laboratório de Modelagem - O Desafio do RH

Agora é sua vez de atuar como arquiteto(a) de dados. O desafio é modelar um banco de dados para um sistema de Recursos Humanos (RH) simples.

Requisitos e Regras de Negócio:

- A empresa precisa armazenar dados de seus **funcionários**: ID, nome completo, CPF, data de nascimento e salário.
- Os funcionários estão alocados em **departamentos**: ID do departamento e nome (ex: "Tecnologia", "Marketing").

- Um funcionário pertence a apenas **um departamento**, mas um departamento pode ter **muitos funcionários**.
- Cada funcionário ocupa um **cargo**: ID do cargo, nome do cargo (ex: "Analista de Dados", "Gerente de Projetos").
- Um funcionário tem apenas **um cargo** em um determinado momento. Um mesmo cargo pode ser ocupado por **vários funcionários**.
- A empresa quer registrar o **histórico de projetos** em que cada funcionário trabalhou. Um funcionário pode trabalhar em **vários projetos**, e um projeto pode ter **vários funcionários** trabalhando nele. Para cada participação, é importante saber a data de início e a data de fim da alocação do funcionário naquele projeto.
- Os **projetos** têm: ID do projeto, nome do projeto e data de início do projeto.

Sua Missão:

Com base nos requisitos acima, desenvolva o modelo de dados completo, passando por todas as etapas que aprendemos:

1. Identifique as entidades, atributos e relacionamentos.
2. Aplique a cardinalidade correta a cada relacionamento.
3. Elabore o MER para o modelo lógico (esquema relacional), listando as tabelas, colunas, chaves primárias e estrangeiras.

Ao final, você deve ter um diagrama MER completo e o esquema relacional documentado, prontos para a fase de criação do banco de dados que veremos no próximo módulo.

Módulo 2: QUESTÕES DE AVALIAÇÃO

Questão 1 Durante a modelagem de um sistema para uma universidade, um arquiteto de dados precisa representar o seguinte cenário: "Um aluno pode se inscrever em muitas turmas, e uma turma é composta por muitos alunos". Qual é a cardinalidade deste relacionamento e qual é a solução correta para implementá-lo no modelo lógico?

A) 1:N, implementado com uma chave estrangeira na tabela alunos.

B) 1:1, implementado com uma chave estrangeira em qualquer uma das tabelas com restrição UNIQUE.

C) N:M, implementado diretamente com duas chaves estrangeiras em cada tabela.

D) N:M, que deve ser decomposto criando uma entidade associativa matriculas entre alunos e turmas.

Questão 2 No processo de criação do Modelo Entidade-Relacionamento (MER) para um sistema de RH, o objeto "Funcionário", sobre o qual se deseja guardar informações como nome, CPF e data de nascimento, é classificado como qual componente fundamental do MER?

A) Um Relacionamento, representado por um losango.

B) Uma Entidade, representada por um retângulo.

C) Um Atributo Multivalorado, representado por uma elipse dupla.

D) Uma Cardinalidade.

Questão 3 Ao projetar a tabela funcionarios, um modelador precisa escolher o tipo de dado mais adequado para a coluna salario, que armazenará valores como 12500.75. Qual tipo de dado do MySQL é o mais indicado para garantir a precisão exata em cálculos financeiros?

A) FLOAT

B) INT

C) VARCHAR(20)

D) DECIMAL(p, s)

Questão 4 Um analista está modelando a relação entre fabricantes e produtos com a regra de negócio: "Um produto é fabricado por exatamente um fabricante, mas um

fabricante pode produzir muitos produtos". Qual é a cardinalidade correta para este

relacionamento?

A) 1:1 (Um-para-Um)

B) N:M (Muitos-para-Muitos)

C) 1:N (Um-para-Muitos)

D) Não há relacionamento entre as entidades.

Questão 5 No MER de uma biblioteca, o atributo isbn é escolhido para identificar unicamente cada livro. Como esse atributo especial, que se tornará a chave primária, é representado visualmente no diagrama?

A) É representado por um losango.

B) O nome do atributo é escrito em negrito.

C) O nome do atributo é sublinhado dentro da elipse.

D) É representado por um retângulo.

Questão 6 Ao converter um modelo conceitual (MER) para um modelo lógico, qual é a regra de mapeamento correta para as entidades e seus atributos simples?

A) Entidades tornam-se colunas e atributos tornam-se tabelas.

B) Entidades tornam-se chaves estrangeiras e atributos tornam-se chaves primárias.

C) Entidades tornam-se tabelas e atributos tornam-se colunas.

D) Entidades tornam-se bancos de dados e atributos tornam-se tabelas.

Questão 7 Um desenvolvedor precisa armazenar em uma coluna status_pedido apenas um dos seguintes valores pré-definidos: 'AGUARDANDO', 'PAGO', 'ENVIADO', 'CANCELADO'. Qual é o tipo de dado mais eficiente em armazenamento e integridade para este caso de uso?

A) VARCHAR(15)

B) TEXT

C) ENUM

D) SET

MÓDULO 3: CRIAÇÃO E CARGA DE DADOS

Seja bem-vindo(a) ao Módulo 3. Nos módulos anteriores, atuamos como arquitetos: entendemos o terreno (fundamentos de SGBDs) e desenhamos a planta detalhada da nossa construção (modelagem de dados). Agora, é o momento de colocar o capacete e as ferramentas para trabalhar. Vamos transformar nosso projeto em um edifício digital, tijolo por tijolo, viga por viga.

A linguagem SQL é dividida em subconjuntos, cada um com um propósito específico. Compreendê-los organiza nosso pensamento sobre as operações que podemos realizar.

	ANOME COMPLETO	**PROPÓSITO**	**COMANDOS PRINCIPAIS**
DQL	Data Query Language	**Consultar** e recuperar dados do banco de dados.	`SELECT`
DML	Data Manipulation Language	**Manipular** os dados: inserir, atualizar, remover.	`INSERT`, `UPDATE`, `DELETE`
DDL	Data Definition Language	**Definir** a estrutura: criar, alterar, remover objetos.	`CREATE`, `ALTER`, `DROP`
DCL	Data Control Language	**Controlar** o acesso e as permissões dos usuários.	`GRANT`, `REVOKE`
TCL	Transaction Control Language	**Gerenciar** transações para garantir a consistência.	`COMMIT`, `ROLLBACK`, `SAVEPOINT`

Neste módulo, você aprenderá a usar a **DDL (*Data Definition Language*)** para construir a estrutura física do banco de dados e a **DML (*Data Manipulation Language*)** para popular essa estrutura com os primeiros dados, dando vida ao nosso sistema. Ao final, você será capaz de converter qualquer modelo lógico em um banco de dados tangível e pronto para uso.

Capítulo 6: Criando, Editando e Deletando o Banco e as Tabelas (DDL)

A DDL é o subconjunto da linguagem SQL responsável por todos os comandos que definem ou alteram a estrutura do banco de dados e seus objetos. Nosso foco inicial será nos dois comandos mais fundamentais: `CREATE DATABASE` e `CREATE TABLE`.

6.1 CREATE DATABASE: Criando os bancos de dados

Antes de construirmos qualquer tabela, precisamos de um "terreno" para o nosso projeto. O comando `CREATE DATABASE` cria um contêiner lógico no servidor MySQL, nosso banco de dados ou *schema*, um espaço isolado onde nossas tabelas, visões e outros objetos irão residir.

A sintaxe é: `CREATE DATABASE nome_do_banco;`

Contudo, para criar um banco de dados verdadeiramente profissional e preparado para o mundo real, precisamos ir além do básico e definir como ele irá "entender" e "falar" a linguagem dos nossos dados.

6.1.1 A Linguagem do Mundo: A Escolha Crítica do *Character Set*

Ao criar um banco de dados, uma das decisões mais importantes que você tomará é a escolha do **conjunto de caracteres (*Character Set*)**.

6.1.2 O que é um Character Set?

Imagine um **dicionário** que mapeia cada caractere que conhecemos (como 'A', 'B', 'á', 'ç', '€', ' ') a um número que o computador pode armazenar. Isso é, em essência, um *character set*. Toda vez que você salva um texto, o banco de dados usa esse dicionário para converter os caracteres em números; ao ler, ele faz o processo inverso. Se o banco tentar ler os dados com o dicionário errado, o resultado será um texto "corrompido" (ex: cafÃ© em vez de café).

6.1.3 A Importância da Escolha Certa

Em aplicações modernas, seus usuários podem inserir nomes com acentos, textos em diferentes idiomas, símbolos monetários e, claro, emojis. Escolher um *character set* limitado desde o início pode causar erros difíceis de corrigir no futuro, quando o sistema já estiver em produção. A escolha correta garante que sua aplicação seja globalmente compatível.

6.1.4 Entendendo os *Bytes* e os Principais *Character Sets*

- **ascii**: O padrão original. Usa **1 *byte*** para armazenar 128 caracteres básicos da língua inglesa, sem acentos ou símbolos especiais.
- **latin1**: Uma extensão do ascii. Também usa **1 *byte***, mas mapeia os 128 números restantes para caracteres acentuados de idiomas da Europa Ocidental, como o português. Ainda é muito limitado para um cenário global.
- **UTF-8**: O padrão da internet. É um sistema de codificação de tamanho variável. Caracteres básicos como 'A' ou 'B' ocupam **1 *byte***. Caracteres acentuados como 'á' ou 'ç' ocupam **2 *bytes***. Outros símbolos e caracteres de idiomas não latinos podem ocupar **3 ou 4 *bytes***.

6.1.5 O Dilema do MySQL: utf8 vs. utf8mb4

Aqui reside um detalhe crucial e específico do MySQL que distingue um iniciante de um profissional:

utf8 (*Alias* para utf8mb3): Por razões históricas, a implementação utf8 no MySQL é **incompleta**. Ela aloca um máximo de **3 *bytes*** por caractere. Isso é suficiente para a maioria dos idiomas, mas **falha** ao tentar armazenar caracteres que exigem 4 *bytes*, como a grande maioria dos emojis ( ) e alguns símbolos matemáticos.

utf8mb4: Esta é a implementação **correta e completa** do padrão UTF-8 no MySQL. Ela aloca um máximo de **4 *bytes*** por caractere, garantindo suporte a **todos** os caracteres Unicode existentes.

Conclusão: Em qualquer aplicação moderna, **sempre utilize utf8mb4**. O utf8 original do MySQL é um legado que deve ser evitado para prevenir erros inesperados quando um usuário tentar salvar um emoji ou outro caractere especial.

6.1.6 E a Colação (*Collate*)?

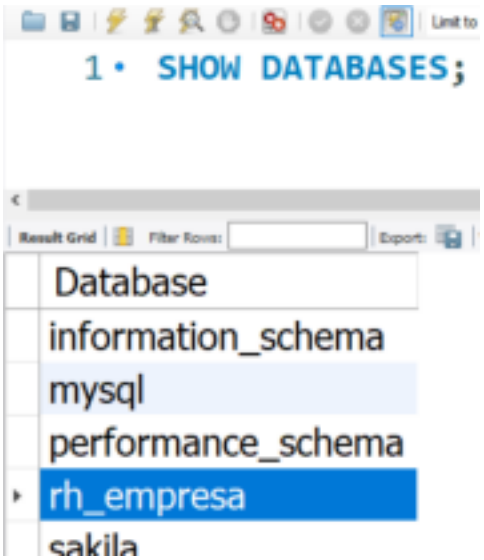
Junto com o *character set*, definimos a **colação (*Collation*)**. Se o *character set* é o dicionário, a colação é o **conjunto de regras de ordenação e comparação** para aquele dicionário. Ela define, por exemplo, se 'a' é igual a 'A' (*case-insensitive*, ou _ci) e a ordem correta de caracteres acentuados. **utf8mb4_unicode_ci** é uma excelente escolha padrão.

6.1.7 Aplicando as Boas Práticas

Agora, vamos unir todo esse conhecimento no nosso comando **CREATE DATABASE**:

```sql
-- Cria um novo banco de dados para o nosso projeto de RH
-- Utilizando o conjunto de caracteres utf8mb4, que suporta um vasto range de caracteres, incluindo emojis.
-- A collation 'utf8mb4_unicode_ci' define as regras de comparação e ordenação de texto (case-insensitive).
CREATE DATABASE rh_empresa CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Você pode utilizar o comando **SHOW DATABASES;** a qualquer momento para verificar quais banco de dados estão presentes no SGBD.



Após criar o banco, precisamos informar ao MySQL que queremos trabalhar dentro dele. Para isso, usamos o comando **`USE nome_do_banco;`**:

```
1 -- Seleciona o banco de dados 'rh_empresa' como o banco de dados ativo para os comandos seguintes.
2 USE rh_empresa;
```


6.2 Gerenciando Bancos de Dados: ALTER e DROP DATABASE

Enquanto **`CREATE DATABASE`** nos dá o ponto de partida, os comandos **`ALTER DATABASE`** e **`DROP DATABASE`** nos permitem gerenciar o ciclo de vida de um banco de dados.

`ALTER DATABASE`: A utilidade deste comando varia bastante entre os diferentes SGBDs. No MySQL, por exemplo, ele é frequentemente usado para alterar características gerais do banco de dados, como o conjunto de caracteres (*character set*) ou a "*collation*" (regras de ordenação de texto).

```
1  -- Cria um BD sem especificar CHARSET ou COLLATE.
2  CREATE DATABASE controle_de_vendas;
3
4  -- Altera o CHARSET e COLLATE do BD.
5  ALTER DATABASE controle_de_vendas
6  CHARACTER SET = utf8mb4
7  COLLATE = utf8mb4_general_ci;
```

DROP DATABASE: Este é um dos comandos mais perigosos do SQL. Ele remove permanentemente todo o banco de dados, incluindo todas as suas tabelas, views, procedures e dados. **Use-o com extremo cuidado!**

```sql
-- ATENÇÃO: Remove permanentemente 'controle_de_vendas'.
DROP DATABASE controle_de_vendas;
```

Pergunta para reflexão: Qual seria o impacto de executar um DROP DATABASE em um ambiente de produção sem um backup recente?

6.3 CREATE TABLE: As Vigas e Paredes da Estrutura

Com nosso terreno preparado e selecionado, podemos começar a erguer as estruturas. O comando **CREATE TABLE** define uma nova tabela, suas colunas, os tipos de dados de cada coluna e as restrições que se aplicam a elas. A estrutura básica do comando é:

```sql
CREATE TABLE nome_da_tabela (
    nome_coluna1 tipo_de_dado RESTRICOES,
    nome_coluna2 tipo_de_dado RESTRICOES,
    ...
    PRIMARY KEY (nome_coluna_chave)
);
```

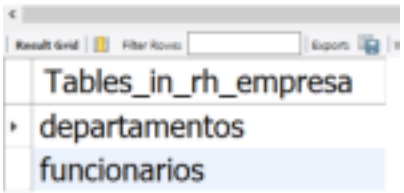
Vamos começar criando a tabela departamentos do nosso modelo de RH, que é uma tabela independente (não possui chaves estrangeiras):

```sql
-- Cria a tabela 'departamentos' para armazenar os diferentes departamentos da empresa.
CREATE TABLE departamentos (
    -- 'id_departamento' será a chave primária
    -- Com AUTO_INCREMENT o número inteiro se auto-incrementa a cada nova inserção.
    id_departamento INT AUTO_INCREMENT,

    -- 'nome' armazenará o nome do departamento, como 'Tecnologia' ou 'Marketing'.
    -- VARCHAR(100) permite textos de até 100 caracteres.
    -- NOT NULL garante que todo departamento deva, obrigatoriamente, ter um nome.
    nome VARCHAR(100) NOT NULL,

    -- Define explicitamente que 'id_departamento' é a chave primária desta tabela.
    PRIMARY KEY (id_departamento)
);
```

Você pode utilizar o comando **SHOW TABLES;** a qualquer momento para verificar

quais tabelas estão presentes no seu banco de dados ativo.

```sql
SHOW TABLES;
```



Agora estamos prontos para entender os demais conceitos que compões a criação de tabelas com SQL.

6.4 Tornando seus Scripts mais Seguros com IF EXISTS e IF NOT EXISTS

Ao escrever scripts SQL que podem ser executados múltiplas vezes, corremos o risco de encontrar erros. Por exemplo, tentar criar uma tabela que já existe resultará em um erro, interrompendo a execução do script. Para evitar isso, utilizamos as cláusulas `IF NOT EXISTS` e `IF EXISTS`.

`IF NOT EXISTS`: Geralmente utilizado junto com `CREATE`, esta cláusula verifica primeiro se o objeto (como uma tabela ou banco de dados) já existe. O comando `CREATE` só será executado se o objeto não existir.

```sql
-- Cria o BD 'meu_projeto' apenas se ele ainda não existir.
CREATE DATABASE IF NOT EXISTS meu_projeto;

USE meu_projeto;

-- Tenta criar a tabela 'usuarios'
CREATE TABLE IF NOT EXISTS usuarios (
    id_usuario INT PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(100) NOT NULL UNIQUE
);
```

A grande vantagem dessas cláusulas é a segurança: ao executar o mesmo script várias vezes, em vez de um erro que interromperia o processo, o sistema apenas exibirá um aviso (WARNING), informando que a tabela ou o banco de dados já existe e continuará para o próximo comando.

`IF EXISTS`: Que pode ser usada junto com o `DROP` por exemplo, esta cláusula garante que o comando de exclusão só seja executado se o objeto realmente existir, prevenindo erros desnecessários no log.

O uso dessas cláusulas é considerado uma boa prática na escrita de scripts de migração e automação, tornando-os mais resilientes e previsíveis.

Capítulo 7: Garantindo a Solidez - Restrições de Integridade (*Constraints*)

As restrições (*Constraints*) são regras que aplicamos às colunas de uma tabela para garantir a precisão, a confiabilidade e a integridade dos dados. Elas impedem que dados inválidos sejam inseridos, atualizados ou excluídos, agindo como guardiões da qualidade da nossa informação.

7.1 O Arsenal de Restrições

- **UNIQUE:** Assegura que todos os valores em uma coluna (ou conjunto de colunas) sejam diferentes uns dos outros. Útil para campos como CPF ou e mail, que devem ser únicos, mas não são a chave primária principal.
- **NOT NULL:** Garante que uma coluna não pode conter valores nulos. É essencial para campos obrigatórios, como nomes, e-mails ou status.
- **PRIMARY KEY (Chave Primária):** Como já vimos, identifica unicamente cada linha da tabela. Ela é, implicitamente, UNIQUE e NOT NULL.
- **FOREIGN KEY (Chave Estrangeira):** É o pilar dos bancos de dados relacionais. Ela cria um vínculo entre duas tabelas, garantindo a **integridade**

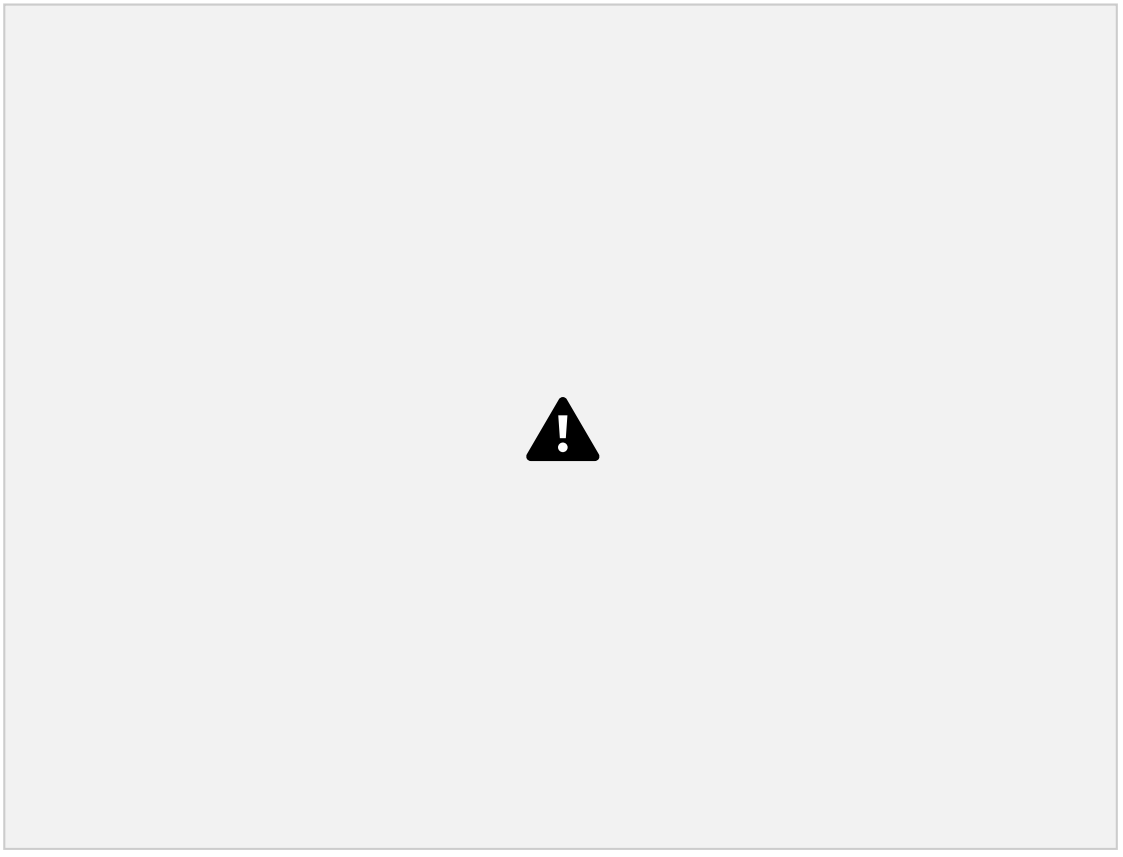
referencial. Isso significa que um valor inserido na coluna de chave estrangeira

deve, obrigatoriamente, existir na coluna de chave primária da tabela referenciada.

- **DEFAULT:** Especifica um valor padrão para uma coluna caso nenhum valor seja fornecido durante uma inserção (INSERT).
- **CHECK:** Permite especificar uma condição para que o dado seja inserido ou atualizado, porém, só é suportado no MySQL 8.0 ou superior. Ex: CHECK (salario > 0).

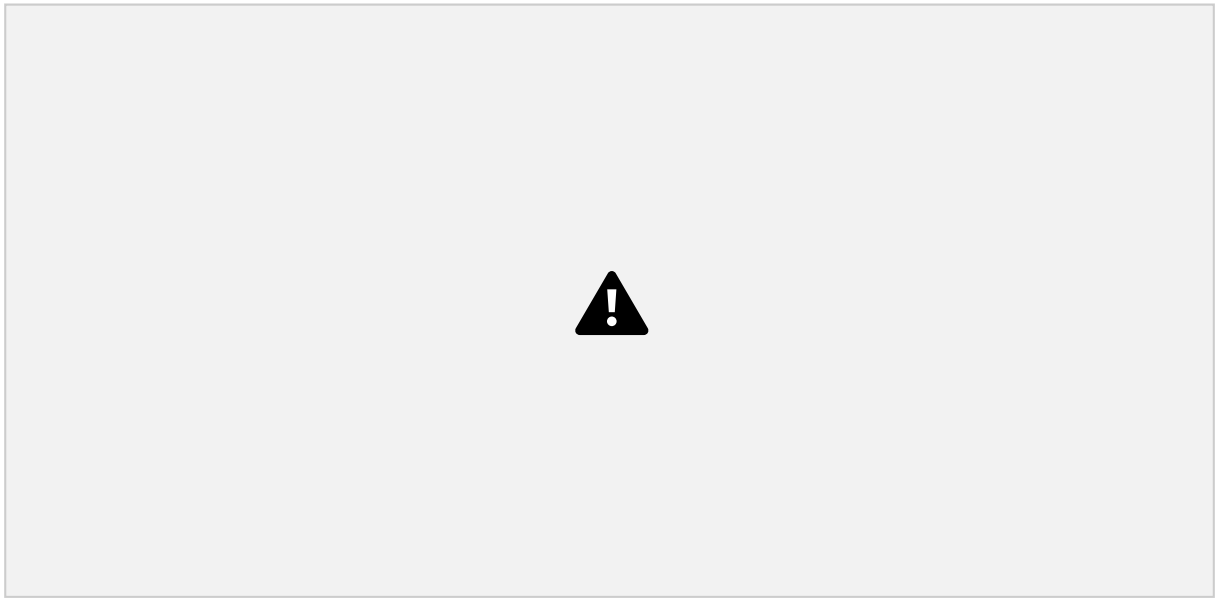
7.2 Aplicando as Restrições ao Banco de RH

Agora, vamos criar a tabela funcionarios, que se relaciona com departamentos, aplicando as restrições adequadas. Note a ordem: criamos primeiro a tabela que "doa" a chave (departamentos) e depois a que a recebe (funcionarios).



Este bloco de código cria uma estrutura sólida: é impossível cadastrar um funcionário sem nome, com CPF duplicado, com salário negativo ou em um departamento que não existe.

Você pode consultar a estrutura das tabelas já criadas utilizando o comando `DESCRIBE`:



Capítulo 8: Modificando e Removendo Estruturas

No dinâmico mundo dos dados, as estruturas que criamos raramente são estáticas. Pense nelas como a planta de um edifício: após a construção, podemos precisar adicionar uma nova sala, reformar uma parede ou até mesmo demolir uma ala inteira. Em SQL, temos as ferramentas exatas para realizar essas "reformas" em nossas tabelas.

Neste capítulo, aprenderemos os comandos essenciais para alterar a estrutura de tabelas existentes e para remover tabelas ou seus dados de forma definitiva.

8.1 O Comando ALTER TABLE: Alterando a Estrutura

O comando `ALTER TABLE` é a nossa principal ferramenta para modificar a definição de uma tabela que já existe. Com ele, podemos adicionar, remover ou alterar colunas sem a necessidade de apagar e recriar a tabela do zero.

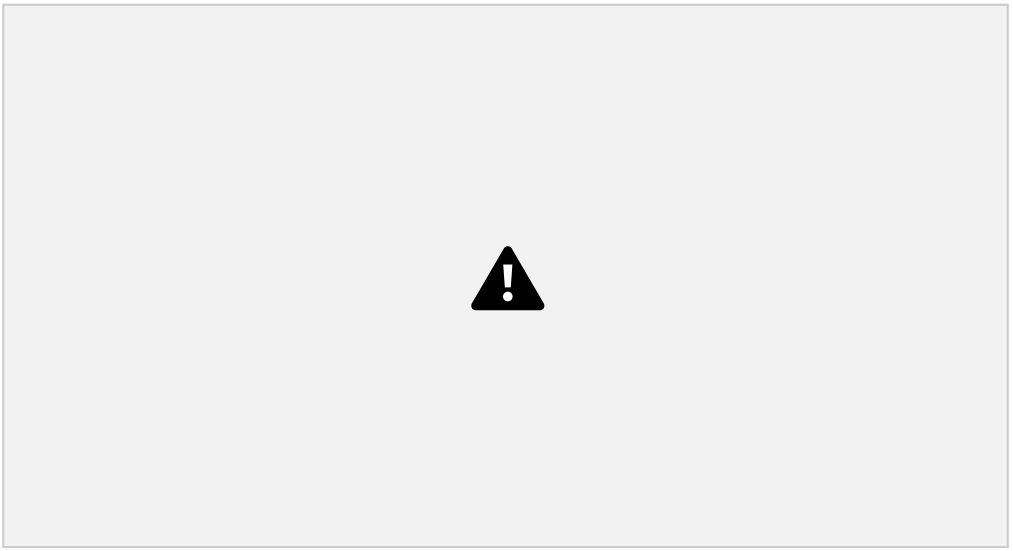
8.1.1 Adicionando uma Nova Coluna com ADD COLUMN

Imagine que, após a criação da tabela funcionarios, o RH solicita que o sistema também armazene o telefone de contato e a data de contratação de cada pessoa.

Exemplo 1: Adicionando a data de contratação

Exemplo 2: Adicionando um campo de telefone opcional

Com o comando DESCRIBE podemos ver a tabela *'funcionarios'* antes e depois da execução dos comandos, com as novas colunas *'data_contratacao'* e *'telefone'* em destaque no final.



8.1.2 Alterando uma Coluna Existente com MODIFY COLUMN ou CHANGE COLUMN

Suponha que as necessidades do negócio mudaram. A coluna *telefone* que criamos como `VARCHAR(20)` se mostrou curta para números internacionais, e o campo

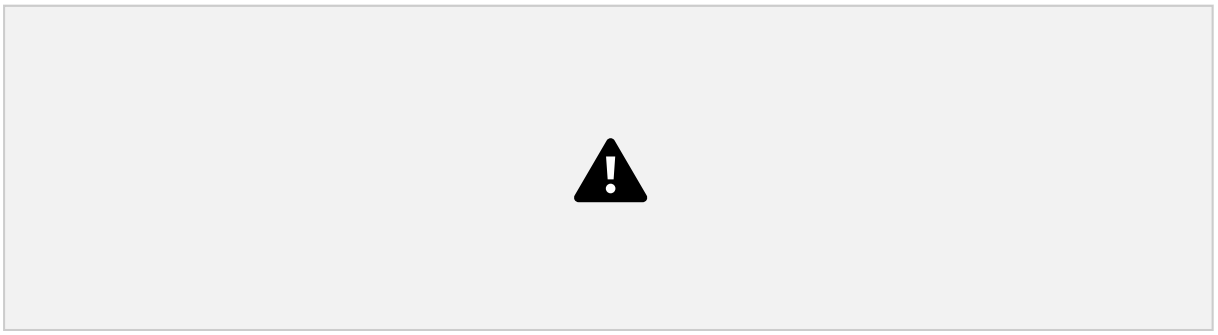
nome na tabela *departamentos* está muito genérico.

Exemplo 1: Modificando o tipo de dado

Vamos aumentar o tamanho da coluna telefone para 25 caracteres. O comando `MODIFY COLUMN` altera as propriedades de uma coluna existente.

Exemplo 2: Renomeando uma coluna com CHANGE COLUMN

Para maior clareza, vamos renomear a coluna *nome* da tabela *departamentos* para *nome_departamento*. No MySQL, o comando `CHANGE COLUMN` permite renomear e redefinir a coluna ao mesmo tempo.



Novamente, pode utilizar o comando `DESCRIBE` para verificar a alteração na tabela do banco de dados.

8.1.3 Removendo uma Coluna com DROP COLUMN

Se uma coluna se tornar obsoleta, podemos removê-la para manter nosso modelo de dados limpo. Imagine que a empresa decidiu não armazenar mais a *data_nascimento* dos funcionários por questões de privacidade.

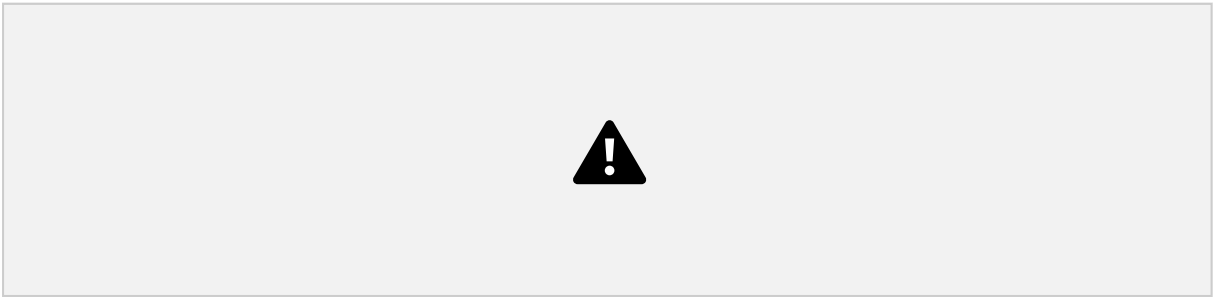
8.2 Exclusões: DROP TABLE e TRUNCATE TABLE

Existem momentos em que precisamos executar ações mais drásticas: remover todos os dados ou a tabela inteira. Para isso, temos dois comandos distintos, cada um com um propósito claro.

8.2.1 DROP TABLE: Demolindo a Estrutura Completa

Este comando é definitivo e destrutivo. Ele remove completamente a tabela do banco de dados: sua estrutura, todos os dados, índices, restrições e permissões associadas.

Exemplo: Suponha que criamos uma tabela *candidatos_temporarios* para um processo seletivo que já terminou. Ela não é mais necessária.



8.2.2 TRUNCATE TABLE: Esvaziando a Estrutura

Este comando remove **todos os registros** de uma tabela de forma extremamente rápida, mas **mantém a sua estrutura intacta** (colunas, índices, etc.). É como esvaziar completamente o edifício, tirando todos os móveis e pessoas, mas deixando a construção de pé para ser usada novamente.

É muito mais rápido que o comando `DELETE` (que veremos mais adiante) para apagar todos os dados de uma tabela.

Exemplo: Para iniciar um novo ciclo de testes, precisamos limpar a tabela *funcionarios* de todos os dados de teste inseridos, mas queremos manter a tabela pronta para receber novos dados.

Exercício de Fixação:

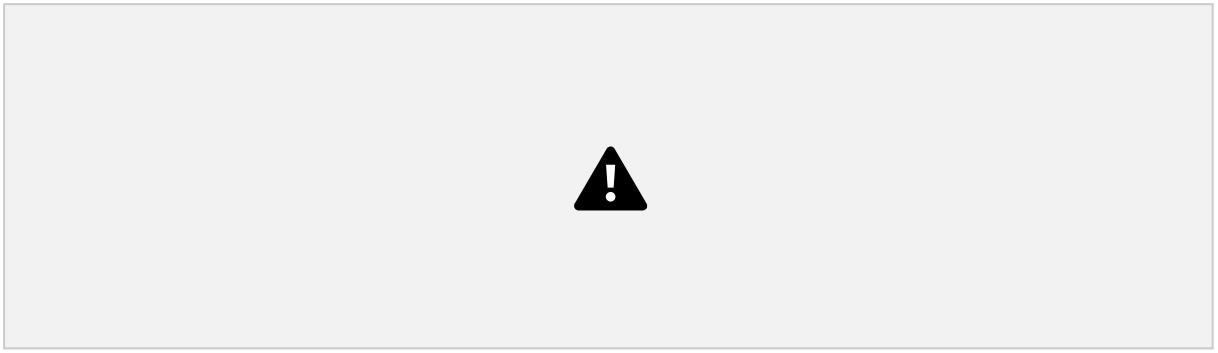
1. Crie uma tabela simples chamada *beneficios* com as colunas *id_beneficio* (`INT PRIMARY KEY`) e *nome_beneficio* (`VARCHAR(100)`).
2. O RH informou que é preciso ter uma descrição. Adicione uma coluna *descricao* do tipo `TEXT` à tabela *beneficios*.
3. A equipe decidiu que *nome_beneficio* é muito longo. Renomeie a coluna para apenas nome e mantenha sua definição como `VARCHAR(100)`.
4. Depois de um tempo, o projeto de benefícios foi cancelado. Remova permanentemente a tabela *beneficios* do banco de dados.

Capítulo 9: Inserindo Dados (DML)

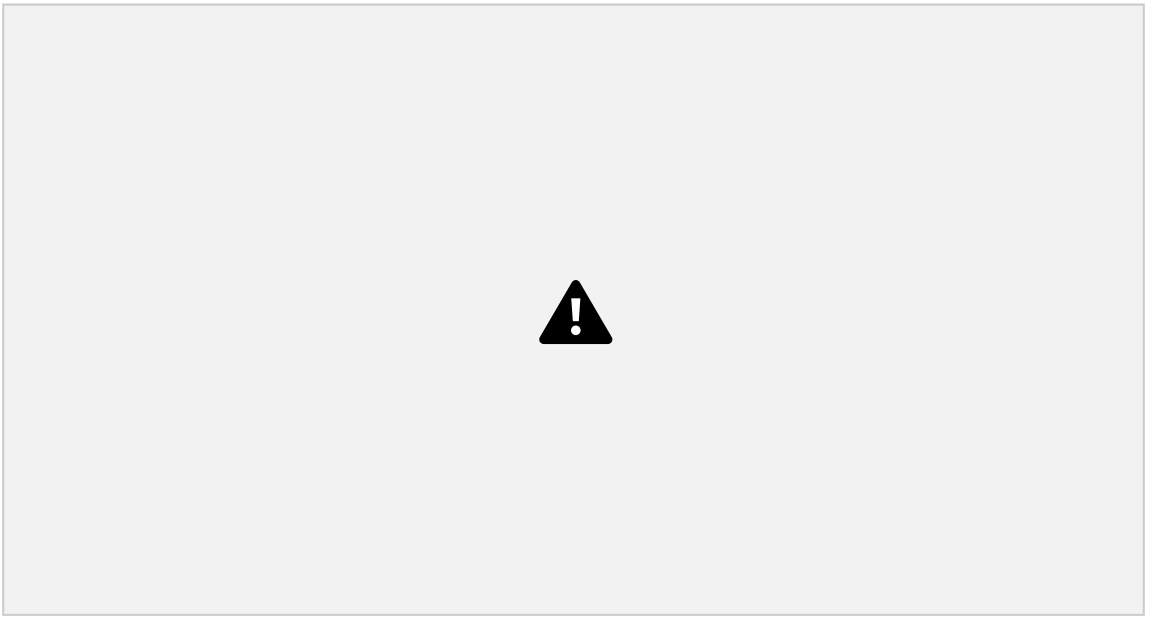
Com a estrutura pronta, é hora de populá-la. Para isso, usamos o principal comando da DML para inserção: `INSERT INTO`.

9.1 Inserindo uma Linha por Vez

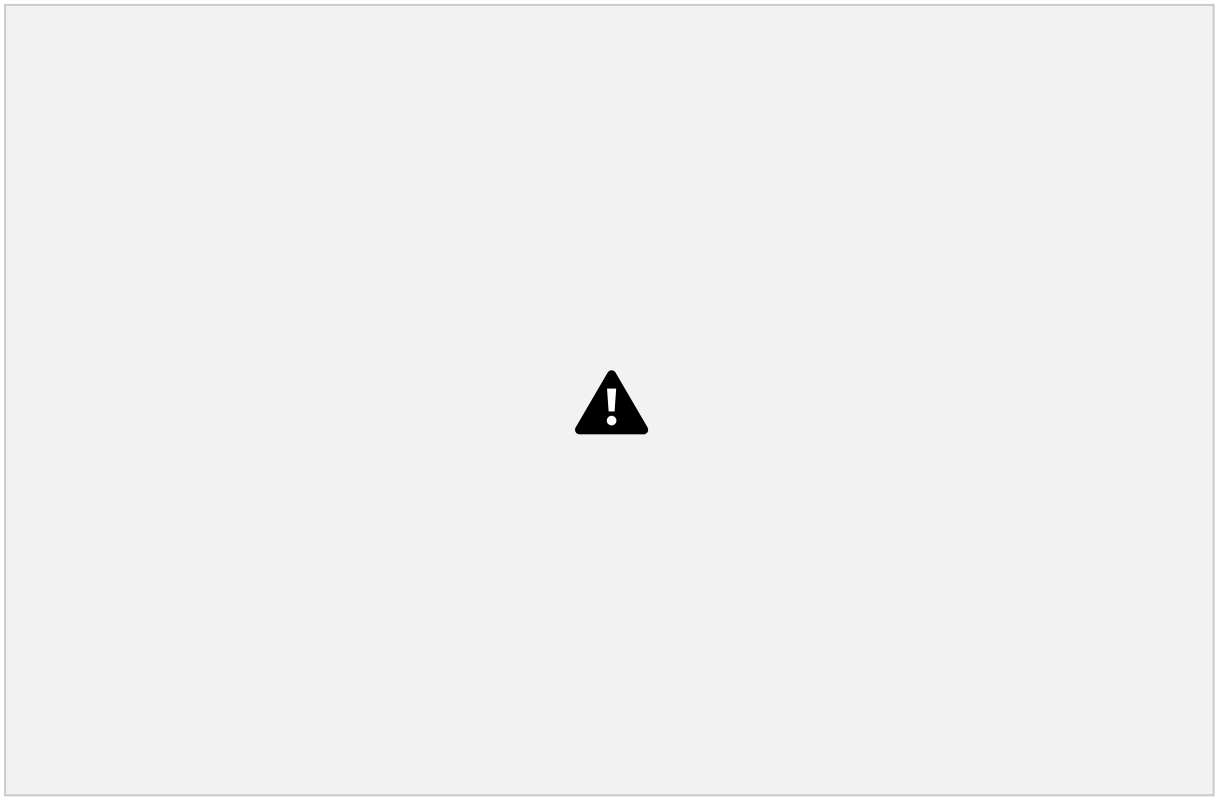
A sintaxe básica especifica a tabela, as colunas que receberão dados e os valores correspondentes. Note que não precisamos fornecer um valor para ***id_departamento***, pois ele é `AUTO_INCREMENT`.



Confira se comando funcionou com `SELECT * FROM departamentos;`.

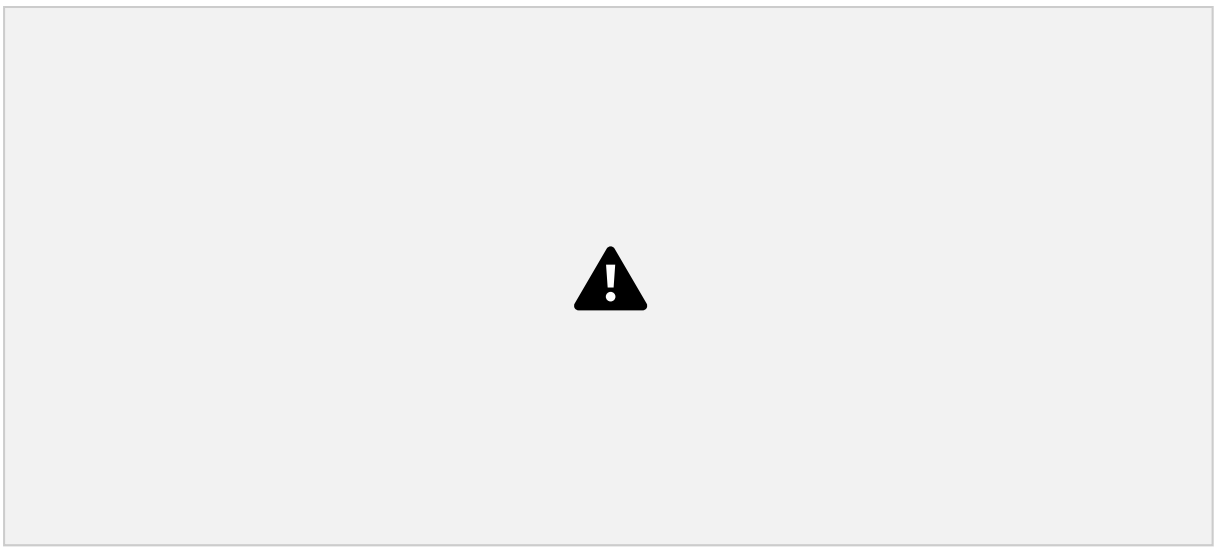


A ordem dos valores em `VALUES` corresponde à ordem das colunas foram especificadas a frente do nome da tabela.



9.2 Otimizando com Inserções Múltiplas

Para inserir vários registros de uma vez, podemos listar múltiplos conjuntos de valores, separados por vírgula. Esta forma é muito mais eficiente.



9.3 Importação de Dados em Massa via Workbench

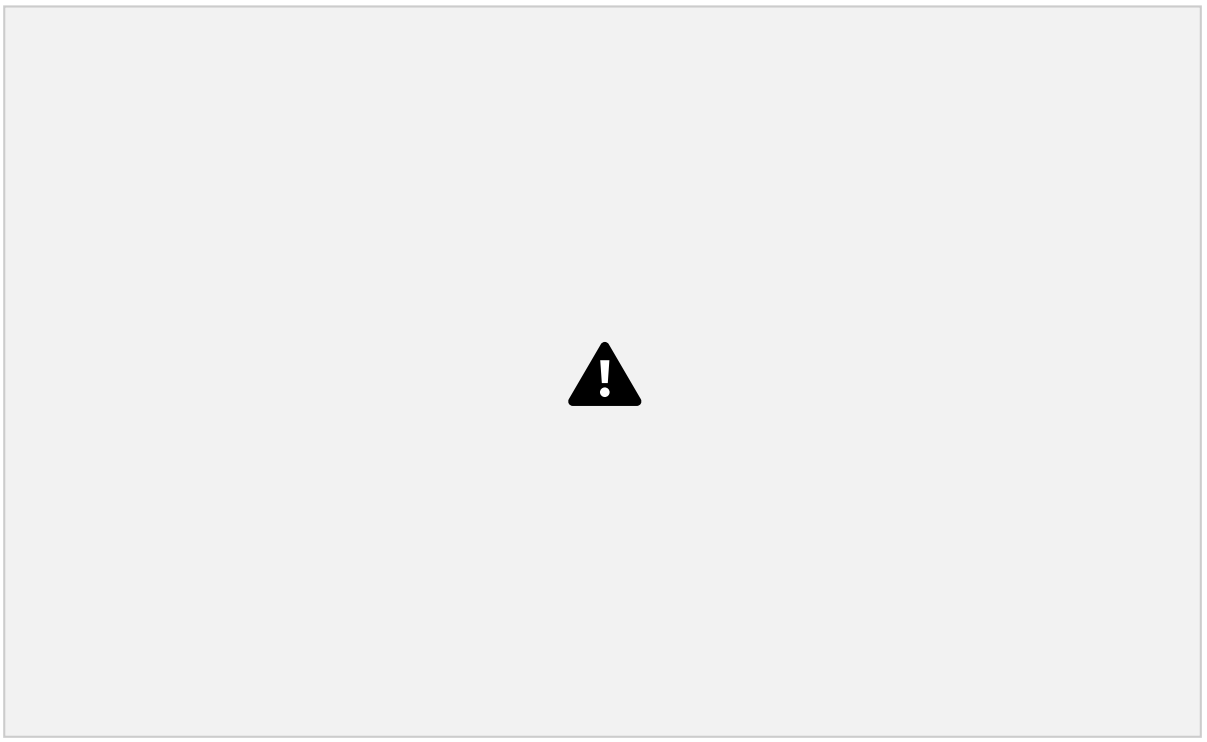
Para grandes volumes de dados, a inserção manual é inviável. O MySQL Workbench oferece um assistente de importação para arquivos CSV (*Comma Separated Values*).

Passo a Passo:

1. **Prepare o Arquivo CSV:** Crie um arquivo (ex: novos_funcionarios.csv) onde a primeira linha contém os nomes das colunas, exatamente como na tabela, e

as linhas seguintes contêm os dados.

50



Ponto de Atenção: Integridade Referencial

O arquivo CSV acima inclui funcionários para os departamentos com IDs **3** e **4**.

Lembre-se que a tabela *funcionarios* possui uma **chave estrangeira** (*FOREIGN KEY*) que a conecta com a tabela *departamentos*. Isso funciona como uma trava de segurança, garantindo que você só pode adicionar um funcionário a um departamento que **realmente existe**.

Portanto, se você tentar importar o CSV antes de criar os departamentos 3 e 4, o banco de dados irá gerar um erro de **violação de chave estrangeira**.

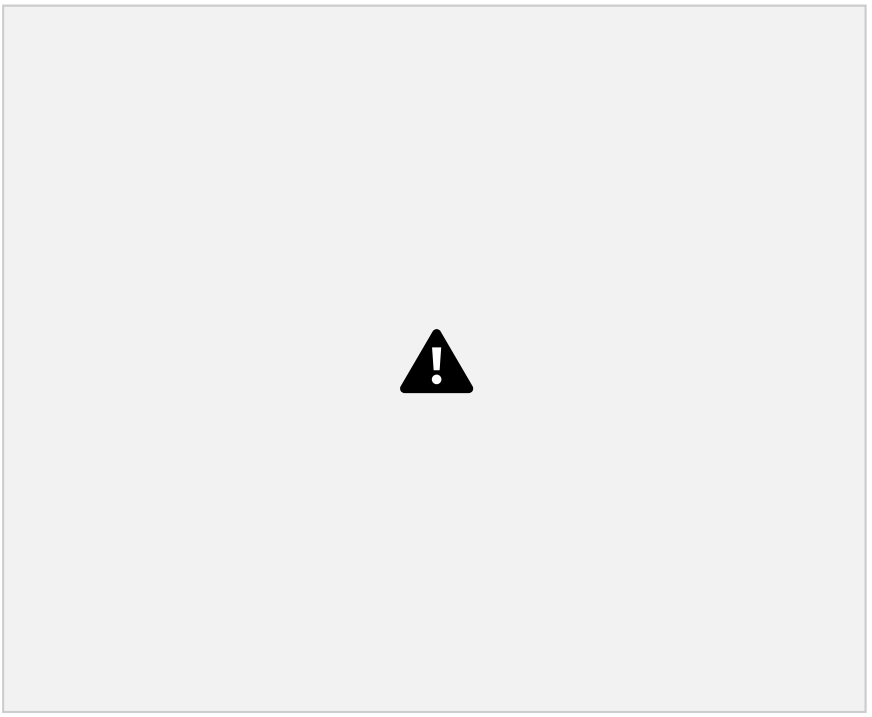
2. **Inicie o Assistente:** Na barra lateral "*Navigator*" do Workbench, encontre sua tabela (ex: funcionarios), clique com o botão direito sobre ela e selecione a opção "***Table Data Import Wizard***".

3. **Selecione o Arquivo:** Na primeira tela do assistente, aponte para o caminho do seu arquivo .csv.

4. **Configure a Importação:** Escolha se deseja importar para uma tabela existente ou criar uma nova. Selecione a tabela de destino (*funcionarios*).

5. **Mapeie as Colunas:** O assistente tentará mapear as colunas do seu CSV para as colunas da tabela. Verifique se o mapeamento está correto.

6. **Execute:** Avance pelas telas e inicie o processo de importação.



9.4 O Grande Debate das Chaves Primárias

A chave primária é o identificador único de cada linha. Sua escolha não é trivial. Vamos analisar as opções mais comuns e robustas.

`BIGINT UNSIGNED AUTO_INCREMENT`

Esta é a abordagem mais tradicional e amplamente utilizada no MySQL para chaves primárias sequenciais. Vamos quebrar o porquê desta escolha:

`UNSIGNED`: Esta palavra-chave instrui o MySQL a não armazenar o sinal (positivo ou negativo) do número, efetivamente desabilitando valores negativos.

- **Como funciona?** Um inteiro padrão usa um bit para representar o sinal. Ao remover esse bit de sinal, todo o espaço de armazenamento é usado para representar números positivos. Isso **dobra a capacidade máxima de valores positivos** do tipo de dado.
- **Por que `UNSIGNED`?** Chaves primárias sequenciais, por sua natureza, nunca são negativas. Usar **`UNSIGNED`** é uma forma de integridade de dados (garantindo que o ID será sempre >= 0) e, mais importante, nos dá o dobro de

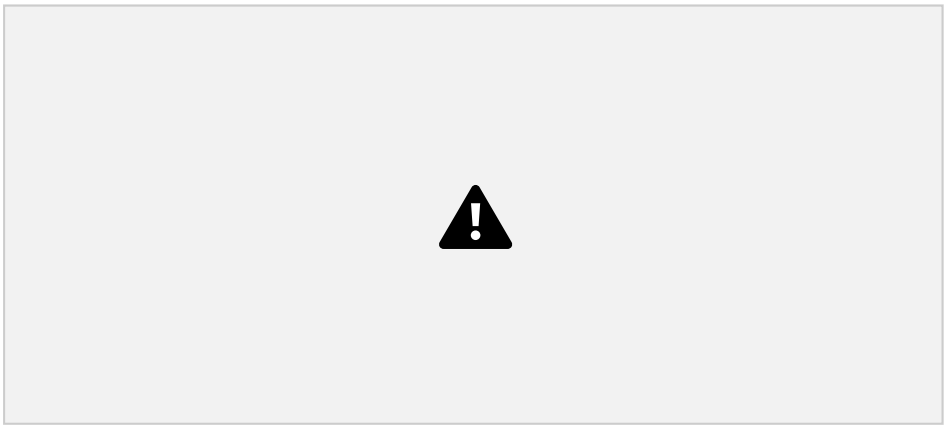
IDs disponíveis para uso futuro (de ~9.2 quintilhões para ~18.4 quintilhões no caso do **`BIGINT`**).

`BIGINT`: Enquanto **`INT`** armazena um número inteiro de até aproximadamente 2.1 bilhões (ou 4.2 bilhões se for **`UNSIGNED`**), o **`BIGINT`** armazena um número de até **9.2 quintilhões** (ou **18.4 quintilhões** se for **`UNSIGNED`**).

- **Por que BIGINT?** Em uma era de grandes quantidades de dados, onde tabelas de log, eventos ou transações podem crescer a uma velocidade espantosa, o limite de **`INT`** pode se tornar um risco real. Adotar **`BIGINT`** como padrão para chaves primárias é uma prática de **design defensivo**.

`AUTO_INCREMENT`: Como vimos, esta propriedade instrui o banco de dados a gerar automaticamente o próximo número sequencial para a chave, garantindo unicidade sem que a aplicação precise se preocupar com isso.

Exemplo de Sintaxe Atualizada:



UUID (Universally Unique Identifier)

Uma alternativa poderosa às chaves sequenciais. **`UUID`** é um padrão que gera um número de 128 bits, representado como uma string de 32 caracteres hexadecimais separados por hífens (ex: 550e8400-e29b-41d4-a716-446655440000).

Quando utilizar?

- **Sistemas Distribuídos:** Se você tem múltiplos servidores de banco de dados que precisam gerar IDs de forma independente sem o risco de colisão, **`UUID`** é

a solução. Um servidor não precisa perguntar a outro qual foi o último ID

gerado.

- **Segurança e Exposição de Dados:** IDs sequenciais (/users/1, /users/2) expõem a quantidade de registros e são facilmente enumeráveis. IDs como /users/550e8400-e29b-41d4-a716-446655440000 são impossíveis de adivinhar.
- **Inserções "Offline":** Uma aplicação pode gerar o ID do novo registro no cliente (ex: em um aplicativo móvel) antes mesmo de enviá-lo ao servidor.

Vantagens: Unicidade global, não sequencial (bom para segurança), geração independente.

Desvantagens: Ocupa mais espaço (16 bytes vs 8 do `BIGINT`), é mais lento para indexar (devido à sua natureza aleatória, que causa fragmentação no índice), e é menos legível para humanos.

No MySQL, um `UUID` pode ser gerado com a função `UUID()` e armazenado em uma coluna `CHAR(36)` ou, de forma mais otimizada, em uma `BINARY(16)`.

Capítulo 10: Laboratório de Construção - O Desafio do E commerce

Agora é o momento de consolidar todo o conhecimento adquirido neste módulo. Você será o engenheiro responsável por construir e popular um banco de dados completo a partir de um modelo lógico.

A Missão: Sua missão é pegar o modelo lógico normalizado do sistema de E commerce (que projetamos no Módulo 2) e traduzi-lo em um banco de dados MySQL 100% funcional e populado com dados iniciais.

O Modelo Lógico Fornecido:

- **clientes** (id_cliente, nome_cliente, email_cliente)

- **produtos** (id_produto, nome_produto, preco_produto)
- **pedidos** (id_pedido, data_pedido, cliente_id FK)
- **itens_pedido** (pedido_id FK, produto_id FK, quantidade)

As Tarefas:

1. **Criação do Banco:** Escreva o script CREATE DATABASE para o banco loja_online, utilizando o CHARACTER SET e COLLATE adequados.
2. **Criação das Tabelas (DDL):** Escreva os scripts CREATE TABLE para todas as quatro tabelas. Preste atenção na ordem de criação (crie as tabelas que não têm dependências primeiro). Defina corretamente todas as chaves primárias (com AUTO_INCREMENT), chaves estrangeiras, tipos de dados apropriados e restrições (NOT NULL, UNIQUE para o e-mail do cliente, CHECK para garantir que o preço e a quantidade sejam positivos).
3. **Carga de Dados (DML):** Escreva os scripts INSERT INTO para popular cada tabela com pelo menos:
    - 3 clientes
    - 5 produtos
    - 4 pedidos (distribuídos entre os clientes)
    - 10 itens de pedido (distribuídos entre os pedidos)

Entregável: Ao final, você deve ter um único arquivo .sql que, quando executado, cria o banco de dados, todas as suas tabelas e insere os dados iniciais, deixando o sistema pronto para ser consultado.

Módulo 3: QUESTÕES AVALIATIVAS

Questão 1 Um arquiteto de banco de dados está projetando a tabela logs_eventos, que pode armazenar bilhões de linhas. Ele precisa definir a chave primária id_log como um número sequencial, gerado automaticamente, garantindo que nunca será

negativo e que terá capacidade para um número extremamente grande de registros. Qual é a definição de coluna mais robusta e defensiva para este cenário?

A) id_log INT AUTO_INCREMENT

B) id_log BIGINT UNSIGNED AUTO_INCREMENT

C) id_log UUID PRIMARY KEY

D) id_log VARCHAR(255) PRIMARY KEY

Questão 2 Uma aplicação web precisa armazenar textos enviados por usuários, que podem incluir emojis ( ) e outros caracteres especiais do conjunto Unicode. Para garantir a compatibilidade total, qual CHARACTER SET deve ser especificado na criação do banco de dados?

A) latin1

B) utf8 (alias para utf8mb3)

C) utf8mb4

D) ascii

Questão 3 Ao criar a tabela funcionarios, é mandatório que cada funcionário seja associado a um departamento que já exista na tabela departamentos. Qual restrição é utilizada na coluna departamento_id da tabela funcionarios para impor essa regra de integridade referencial?

A) UNIQUE

B) CHECK (departamento_id > 0)

C) NOT NULL

D) FOREIGN KEY

Questão 4 Um DBA precisa popular a tabela produtos com 500 novos itens que estão em um arquivo CSV. Qual é a abordagem mais eficiente para realizar essa tarefa, evitando a escrita de 500 comandos INSERT individuais?

A) Utilizar a ferramenta "Table Data Import Wizard" do MySQL Workbench.

B) Escrever um único comando INSERT com 500 conjuntos de valores. C)

Criar um loop dentro de um script SQL para executar INSERT 500 vezes.

D) Inserir manualmente cada produto através da interface gráfica do Workbench.

Questão 5 Um desenvolvedor está escrevendo o script CREATE TABLE para a tabela clientes. Ele precisa garantir que o campo email não possa ter valores duplicados em toda a tabela, mas ele não será a chave primária. Qual restrição deve ser aplicada à coluna email?

A) PRIMARY KEY

B) UNIQUE

C) NOT NULL

D) DEFAULT NULL

Questão 6 Um sistema distribuído precisa gerar IDs únicos para novos registros em diferentes servidores sem que haja risco de colisão e sem a necessidade de comunicação entre eles. Além disso, a empresa não quer expor o número sequencial de registros em suas URLs. Qual tipo de chave primária é o mais indicado para este cenário?

A) BIGINT UNSIGNED AUTO_INCREMENT

B) INT com uma sequência gerenciada pela aplicação.

C) UUID (Universally Unique Identifier)

D) VARCHAR contendo a data e hora da criação.

Questão 7 No comando CREATE TABLE departamentos, a coluna id_departamento foi definida com a propriedade AUTO_INCREMENT. Qual é a principal função desta propriedade?

A) Garante que o valor da coluna será sempre maior que zero.

B) Gera automaticamente um novo número sequencial para a coluna a cada INSERT, servindo como chave primária.

C) Incrementa o valor da coluna em 1 a cada UPDATE.

D) Define o valor padrão da coluna como 1.

Questão 8 O subconjunto da linguagem SQL que é responsável por todos os

comandos que definem ou alteram a estrutura do banco de dados, como CREATE TABLE e ALTER TABLE, é conhecido como:

A) DML (Data Manipulation Language)

B) DQL (Data Query Language)

C) DCL (Data Control Language)

D) DDL (Data Definition Language)

Questão 9 Ao escrever o script de criação de um banco de dados, o desenvolvedor executa CREATE DATABASE rh_empresa CHARACTER SET utf8mb4;. Para que os comandos CREATE TABLE seguintes sejam executados dentro deste novo banco de dados, qual comando ele precisa executar imediatamente após a criação?

A) SELECT DATABASE rh_empresa;

B) CONNECT rh_empresa;

C) USE rh_empresa;

D) SET DATABASE = rh_empresa;

MÓDULO 4: A DINÂMICA DOS DADOS - CONSULTAS E MANIPULAÇÃO

Com nosso banco de dados estruturado, é hora de aprender a interagir com os dados que ele armazena. Este módulo cobre as operações fundamentais do dia a dia: consultar, atualizar e remover informações.

Capítulo 11: O Início de Tudo - Consultando Dados com SELECT (DQL)

O comando **SELECT** é, sem dúvida, o mais utilizado em SQL. É a nossa janela para os dados armazenados.

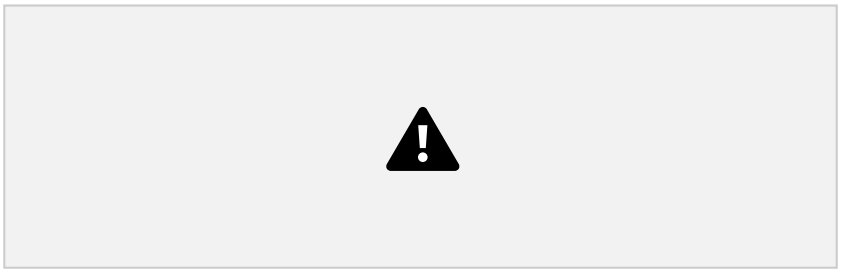
11.1 SELECT *: A Visão Completa

A forma mais básica de consulta é usar * para selecionar **todas as colunas**

de uma tabela.

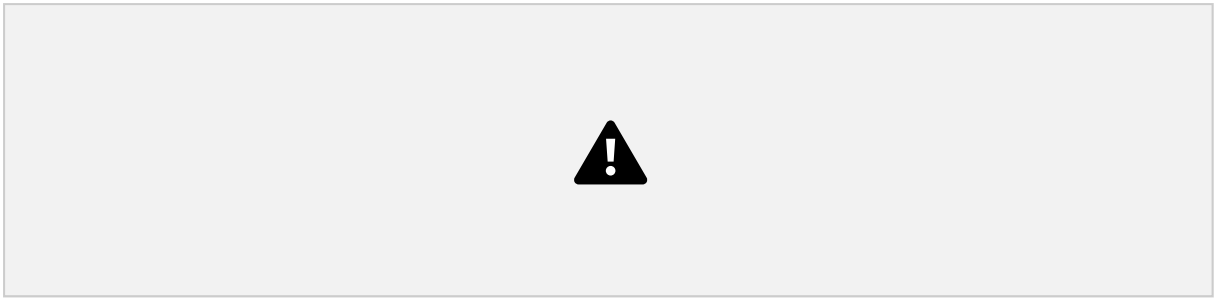
A sintaxe é: `SELECT * FROM tabela;`

Retorno:

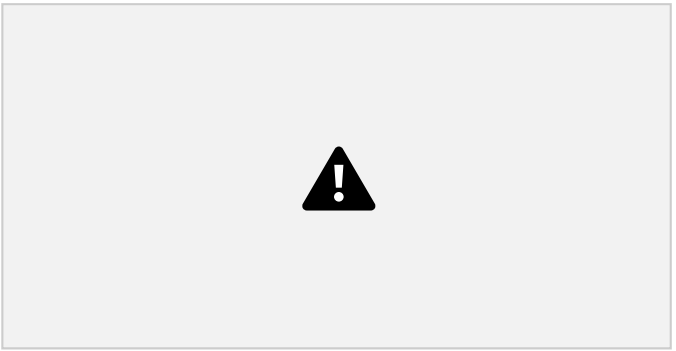


11.2 SELECT [colunas]: Sendo Específico

Em aplicações reais, raramente usamos `*`. É muito mais eficiente e claro solicitar apenas as colunas que nos interessam.



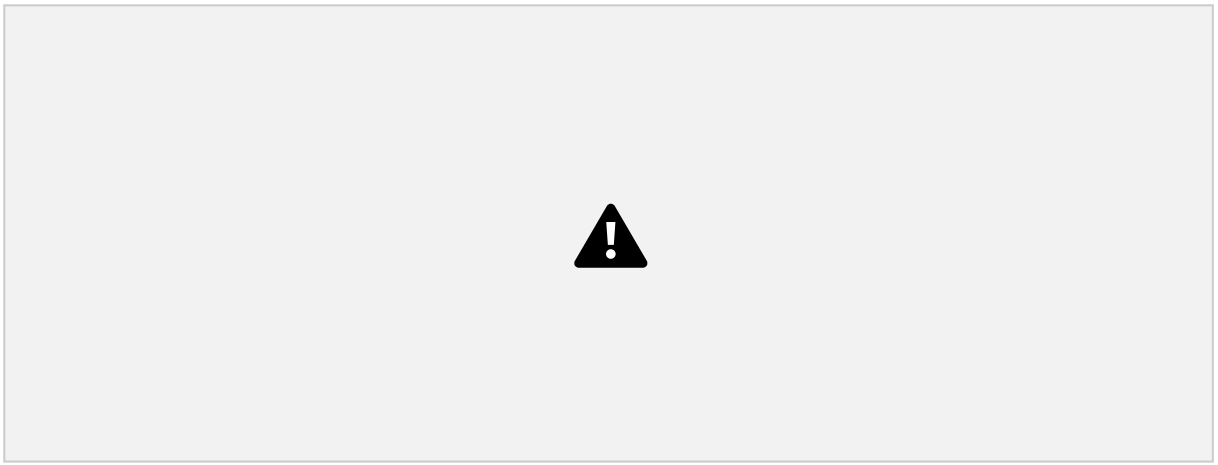
Retorno:



11.3 AS: Dando Apelidos às Colunas (Alias)

Podemos renomear as colunas no resultado da nossa consulta usando a palavra-chave `AS`. Isso é extremamente útil para gerar relatórios mais legíveis ou

para evitar ambiguidades de nomes em consultas complexas.



Resultado:

Capítulo 12: Modificando a Realidade - O Poder e o Cuidado do UPDATE

O comando `UPDATE` é uma das ferramentas mais poderosas e, ao mesmo tempo, perigosas do seu arsenal SQL. Ele permite modificar dados que já existem no banco, sendo essencial para a manutenção e evolução da informação. Usá-lo com precisão é uma marca de profissionalismo; um descuido pode levar a consequências desastrosas.

12.1 A Regra de Ouro: O Imperativo da Cláusula WHERE

Antes de explorarmos os cenários de uso, precisamos estabelecer a regra mais importante sobre o comando `UPDATE`.

ZONA DE PERIGO: O UPDATE SEM WHERE

A sintaxe do `UPDATE` é `UPDATE tabela SET coluna = valor`.

Se você executar este comando **sem uma cláusula** `WHERE` para filtrar e especificar exatamente **quais linhas** devem ser alteradas, o SGBD aplicará a modificação a **TODAS AS LINHAS DA TABELA**.

Imagine executar `UPDATE funcionarios SET salario = 5000;`.

Em um instante, todos os funcionários da empresa, do estagiário ao CEO, passarão a ter o mesmo salário. Este é um evento catastrófico, muitas vezes de difícil recuperação.

Portanto, a regra é simples: **nunca execute um UPDATE em um ambiente de produção sem ter certeza absoluta da sua cláusula WHERE.**

Uma boa prática é sempre executar um `SELECT` com a mesma cláusula `WHERE` antes, para confirmar exatamente quais registros serão afetados.

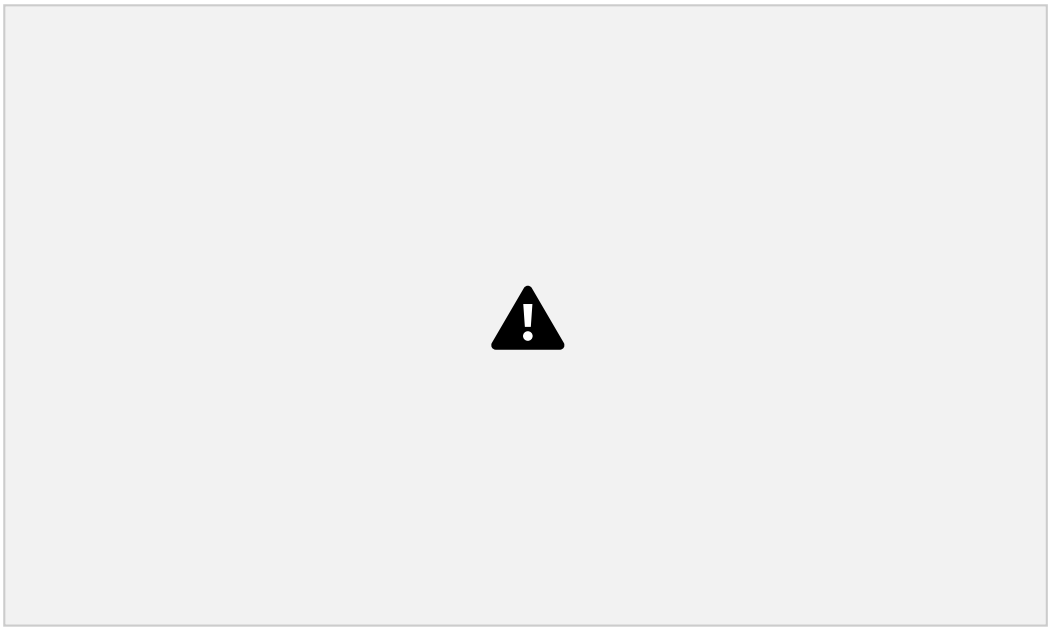
12.2 Cenários Comuns e Exemplos Práticos de UPDATE

Vamos explorar como o UPDATE é utilizado no dia a dia com nosso banco de dados da rh_empresa.

Cenário 1: Correção de Dados Simples

Este é o uso mais comum: corrigir um erro de digitação ou uma informação que se tornou obsoleta.

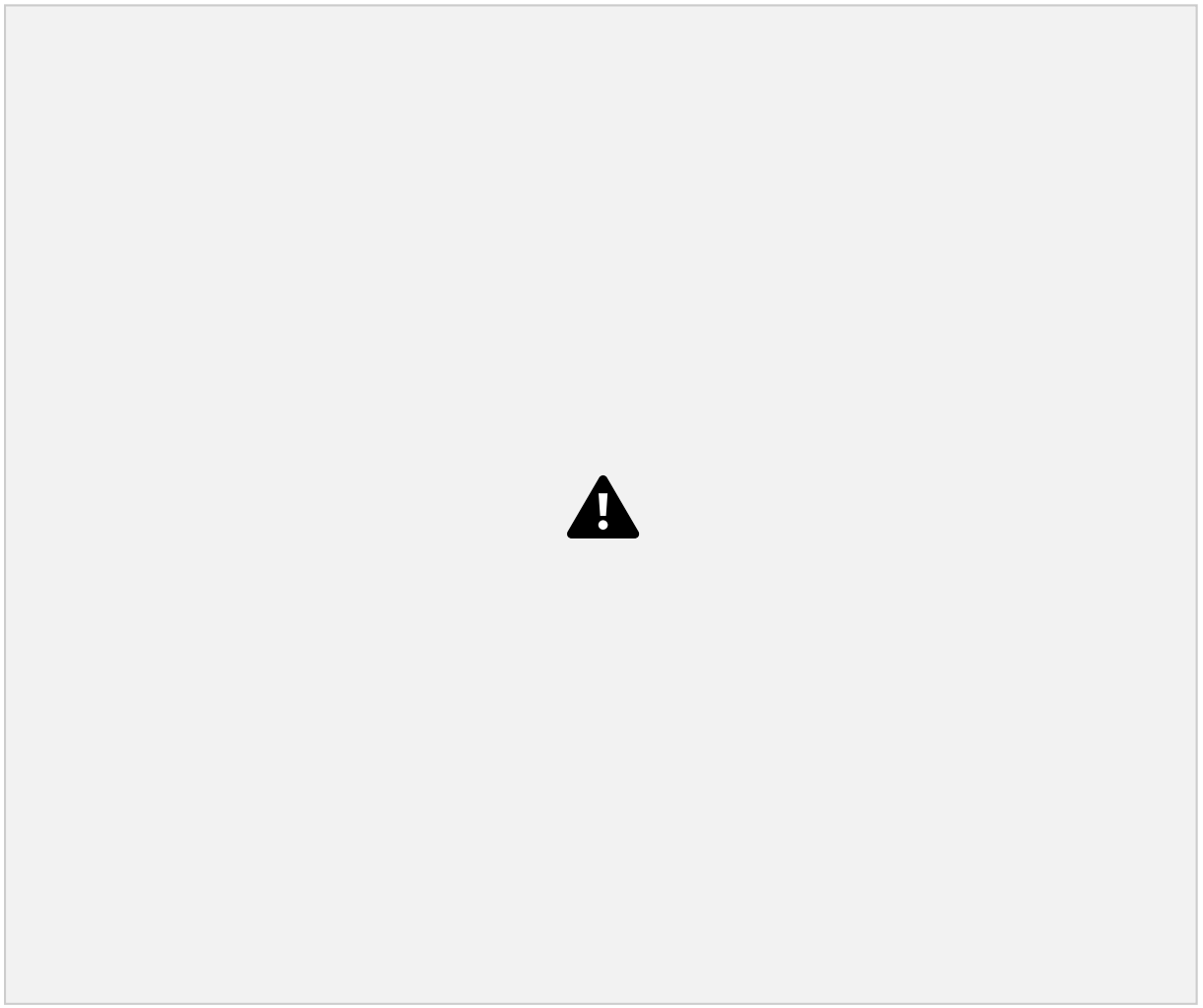
Situação: A funcionária "Ana Silva" nos informa que seu telefone, 11 11111-1111, está incorreto e o correto é 11 22222-3333.



Cenário 2: Atualizações em Massa Baseadas em Valores Existentes

Frequentemente, precisamos aplicar uma modificação a um grupo de registros, muitas vezes usando o valor atual da própria coluna em um cálculo.

Situação: A empresa decide aplicar um reajuste de 5% em todos os salários até $ 5.000.



12.2.1 O Que é o "Safe Update Mode"?

Pense no **modo de atualizações seguras** como um copiloto cuidadoso. Ele foi projetado para evitar o erro acidental mais comum e perigoso em SQL: **atualizar ou deletar múltiplas linhas por engano**.

Para se proteger, o Workbench impõe uma regra simples: se você vai executar um `UPDATE` ou `DELETE`, sua condição na cláusula `WHERE` **deve utilizar uma**

coluna que seja uma chave (como a chave primária *id_funcionario* ou uma coluna com índice `UNIQUE`, como *cpf*).

No seu comando, a condição é `WHERE salario < 5000.00`. Como a coluna *salario* não é uma chave única que identifica um registro específico, o Workbench presume que você pode estar afetando mais linhas do que o pretendido e, por segurança, bloqueia a operação.

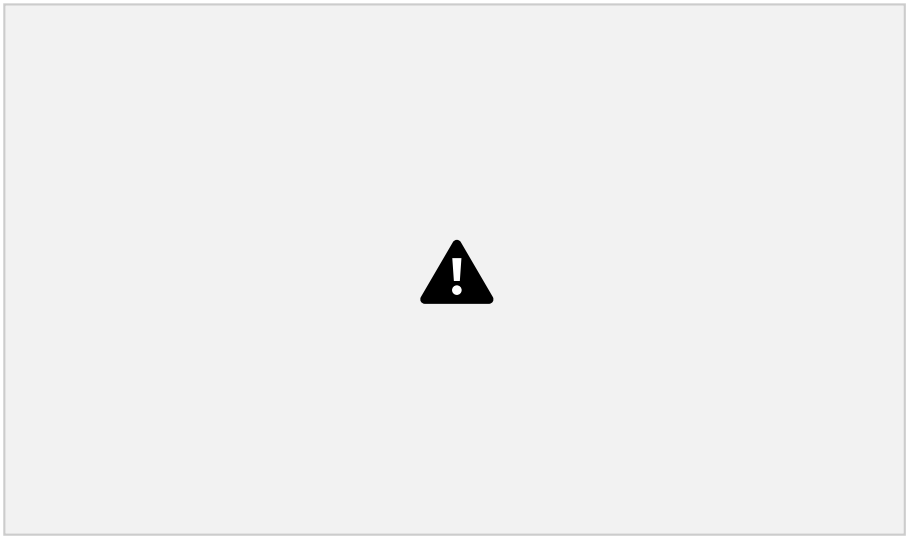
Ele está, essencialmente, dizendo: "Esta ação parece ampla e potencialmente arriscada. Prove para mim que você sabe exatamente quais linhas quer alterar, usando um identificador único."

É possível desabilitar o modo de segurança para a sua sessão atual. Isso é útil quando você tem certeza absoluta do impacto da sua consulta e precisa executar uma atualização em massa conforme foi realizado em nosso exemplo.

Cenário 3: Atualizando Múltiplas Colunas de uma Vez

É possível alterar diversas colunas em um único comando UPDATE, separando as atribuições por vírgula.

Situação: Um funcionário foi promovido. Precisamos atualizar seu salário e seu telefone.



Capítulo 13: Deleção Física vs. Lógica

"Deletar" um dado é uma decisão estratégica que vai muito além de simplesmente liberar espaço. A escolha entre remover fisicamente um registro ou apenas marcá-lo como inativo tem profundas implicações na integridade, na auditoria e na complexidade do sistema.

13.1 Deleção Física com DELETE: A Abordagem Direta e Permanente

O comando `DELETE FROM ... WHERE ...` remove permanentemente as linhas de uma tabela. A linha desaparece e, uma vez que a transação é confirmada (`COMMIT`), a recuperação só é possível através de backups.

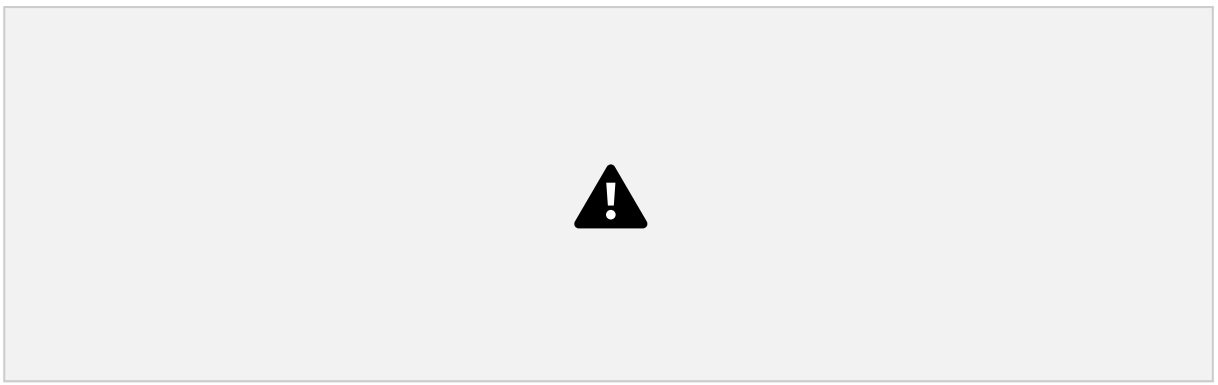
Como Funciona: O comando localiza as linhas que correspondem à cláusula WHERE e as apaga do disco.

Consequências:

- **Perda de Histórico:** A informação é perdida para sempre. Não há como saber que aquele registro um dia existiu.
- **Impacto em Chaves Estrangeiras:** Se a linha que você está deletando (ex: um cliente) é referenciada por outras tabelas (ex: pedidos), o banco de dados irá impedir a exclusão para manter a integridade referencial, a menos que a chave estrangeira tenha sido configurada com `ON DELETE CASCADE` (o que deletaria em cascata todos os pedidos daquele cliente, uma ação muito perigosa).

Quando Utilizar? É apropriado para dados que não possuem valor de negócio histórico ou que são transitórios por natureza.

Exemplo: Um sistema coleta logs temporários de atividades. Após 30 dias, esses logs são expurgados para liberar espaço. Não há necessidade de mantê-los.



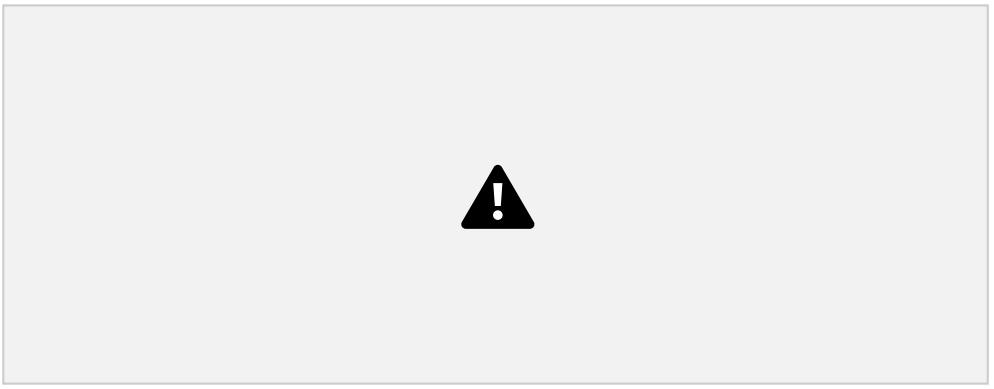
13.2 Deleção Lógica com UPDATE: A Abordagem Preservacionista e Profissional

A deleção lógica, ou *soft delete*, é a prática de **marcar um registro como inativo** em vez de removê-lo fisicamente. Esta é a abordagem preferida para a vasta maioria dos dados de negócio.

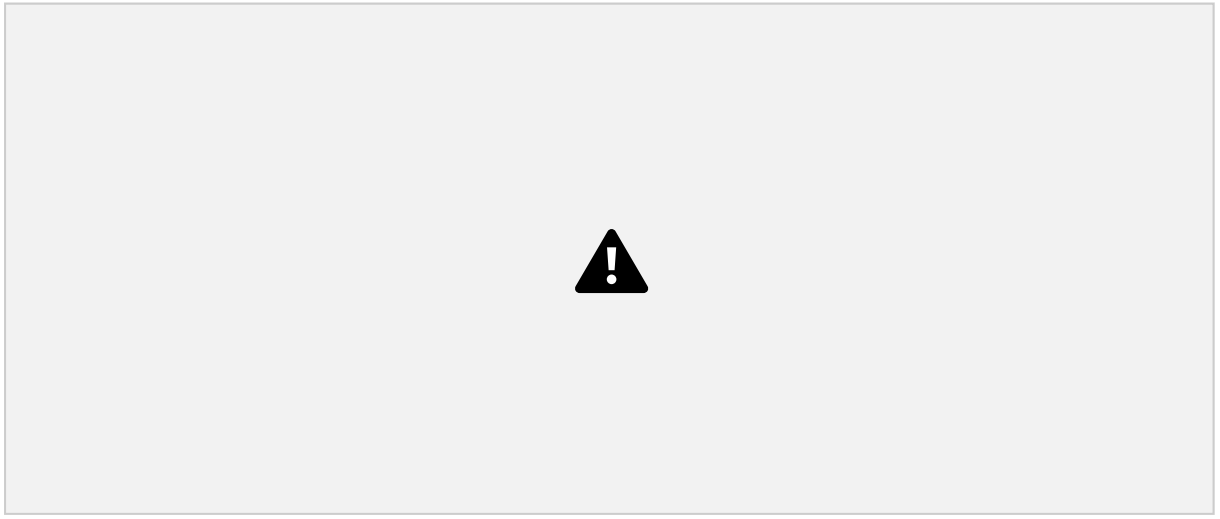
Como Funciona: Adicionamos uma coluna à tabela que controlará o estado do registro. Pode ser uma flag booleana (*esta_ativo*) ou, de forma mais robusta, uma data (*data_desativacao*).

Implementação:

- **Passo 1: Alterar a Tabela (uma única vez)**



- **Passo 2: "Deletar" o Cliente**



Consequências:

- **Histórico Preservado:** Todo o histórico do funcionário e seus dados continuam intactos no banco.
- **Integridade Referencial Garantida:** Como nenhuma linha é realmente deletada, não há risco de quebrar os relacionamentos com outras tabelas.
- **Auditoria e Análise:** É possível analisar dados de funcionários inativos, entender os motivos do desligamento etc.
- **Recuperação Simples:** Para "restaurar" o funcionário, basta executar um

`UPDATE` para definir *data_desativacao* como `NULL` novamente.

Quando Utilizar? Quase sempre, para entidades de negócio centrais como clientes, produtos, pedidos, funcionários etc.

13.3 Análise Comparativa: Qual Metodologia Escolher?

A decisão depende dos seus requisitos. A tabela abaixo resume os pontos chave:

CRITÉRIO	DELEÇÃO FÍSICA (DELETE)	DELEÇÃO LÓGICA (UPDATE)
Performance de Leitura	Melhor. A tabela tem menos linhas, tornando as buscas	Levemente inferior. A tabela cresce continuamente, e toda

CRITÉRIO	DELEÇÃO FÍSICA (DELETE)	DELEÇÃO LÓGICA (UPDATE)
	(SELECT) sobre todos os registros potencialmente mais rápidas.	consulta deve incluir WHERE data_desativacao IS NULL.
Performance de Escrita	A operação DELETE pode ser lenta e causar bloqueios (locks) intensos na tabela, especialmente com cascata.	A operação UPDATE é geralmente muito rápida e eficiente para um único registro.
Armazenamento	Eficiente. O espaço em disco é recuperado (com algumas ressalvas sobre a reorganização interna do SGBD).	Ineficiente. Os dados "deletados" continuam ocupando espaço em disco e nos backups.
Integridade Referencial	Complexo. Requer atenção especial. Pode gerar registros órfãos ou exclusões em cascata indesejadas.	Seguro e Simples. A integridade é sempre mantida, pois nenhum registro é fisicamente removido.
Histórico e Auditoria	Nulo. O histórico é completamente perdido. Impossível auditar o que foi deletado e por quem.	Excelente. Mantém um rastro completo. A coluna de data de desativação serve como um log

		de auditoria.
Recuperação de Dados	Muito Difícil. Exige a restauração de um backup, o que pode ser um processo complexo e demorado.	Trivial. Consiste em um simples comando UPDATE para reverter a marcação de inativo.
Complexidade de Código	Simples no comando (DELETE), mas complexo nas regras de negócio para lidar com as consequências.	Requer uma disciplina de equipe: toda consulta SELECT para a aplicação deve lembrar de filtrar pelos registros ativos.

13.4 Recomendação Final:

Para a vasta maioria dos dados de negócio que representam o núcleo de um sistema, a **deleção lógica é a abordagem mais segura, profissional e escalável**. Ela troca um pouco de espaço em disco e uma pequena complexidade nas consultas por imensos benefícios em segurança, auditoria e manutenção da integridade dos dados. Reserve a deleção física para dados verdadeiramente descartáveis, como logs, caches ou registros temporários.

Módulo 4: QUESTÕES DE AVALIAÇÃO

Questão 1 Um estagiário, ao tentar limpar alguns registros de teste, executa o comando UPDATE funcionarios SET salario = NULL; sem uma cláusula WHERE. Qual será o resultado catastrófico desta operação em um banco de dados de produção?

A) O comando falhará, pois salario provavelmente tem uma restrição NOT NULL.

B) Apenas o primeiro funcionário da tabela terá seu salário zerado.

C) Todos os funcionários da empresa terão o valor da coluna salario alterado para NULL.

D) O sistema irá pedir uma confirmação antes de executar a alteração em massa.

Questão 2 Uma plataforma de e-commerce precisa desativar a conta de um cliente. A empresa precisa manter todo o histórico de pedidos desse cliente para fins de relatórios fiscais e análise de dados. Qual é a abordagem profissional e recomendada para "remover" este cliente do sistema?

A) Executar DELETE FROM clientes WHERE id_cliente = X; e também deletar todos os seus pedidos em cascata.

B) Executar UPDATE clientes SET esta_ativo = 0 WHERE id_cliente = X;, utilizando uma coluna para controle de estado (deleção lógica).

C) Exportar os dados do cliente para um arquivo CSV e depois executar o comando DELETE.

D) Mover o registro do cliente para uma tabela separada chamada clientes_inativos.

Questão 3 Ao gerar um relatório de produtos, a consulta SELECT nome_produto, preco_produto FROM produtos; retorna os cabeçalhos nome_produto e preco_produto. Para tornar o relatório mais legível para a área de negócios, os cabeçalhos devem ser 'Produto' e 'Valor Unitário'. Qual palavra-chave SQL é usada para renomear as colunas no resultado da consulta?

A) RENAME

B) ALIAS

C) AS

D) LABEL

Questão 4 A linguagem SQL é dividida em subconjuntos com propósitos específicos. Os comandos INSERT, UPDATE e DELETE, que são usados para modificar os dados existentes em uma tabela, pertencem a qual subconjunto?

A) DDL (Data Definition Language)

B) DQL (Data Query Language)

C) DML (Data Manipulation Language)

D) TCL (Transaction Control Language)

Questão 5 Um analista de RH precisa corrigir o salário da funcionária com id_funcionario igual a 15, alterando-o para 5500.00. Qual é a sintaxe correta e segura para realizar esta operação?

A) UPDATE 5500.00 INTO funcionarios WHERE id_funcionario = 15; B)

UPDATE funcionarios SET salario = 5500.00 WHERE id_funcionario = 15;

C) SET salario = 5500.00 FROM funcionarios WHERE id_funcionario = 15;

D) UPDATE funcionarios SET salario = 5500.00;

Questão 6 Qual é a principal vantagem da deleção lógica (usando UPDATE) sobre a deleção física (usando DELETE) no que diz respeito à integridade referencial e auditoria?

A) A deleção lógica libera mais espaço em disco.

B) A deleção lógica é mais rápida para ser executada em grandes volumes de dados.

C) A deleção lógica preserva o histórico do registro e mantém os relacionamentos com outras tabelas intactos.

D) A deleção lógica remove permanentemente os dados, garantindo a privacidade.

Questão 7 Em um ambiente de produção, por que é considerada uma boa prática evitar o uso de SELECT * e, em vez disso, especificar explicitamente as colunas necessárias?

A) SELECT * pode causar erros de sintaxe em alguns SGBDs.

B) Reduz o tráfego de dados entre o banco de dados e a aplicação, melhorando a performance.

C) Garante que a ordem das colunas no resultado seja sempre alfabética.

D) SELECT * não permite o uso da cláusula WHERE.

Questão 8 Ao se comparar a deleção física com a lógica, qual das duas abordagens torna a recuperação de um dado "deletado" por engano uma tarefa trivial, consistindo em um simples comando UPDATE?

A) Deleção Física (DELETE)

B) Ambas as abordagens permitem recuperação trivial.

C) Nenhuma das abordagens permite recuperação.

D) Deleção Lógica (UPDATE)

Questão 9 O comando SELECT, utilizado para consultar e recuperar dados de um banco de dados, é o principal comando de qual sub-linguagem do SQL?

A) DML (Data Manipulation Language)

B) DDL (Data Definition Language)

C) DCL (Data Control Language)

D) DQL (Data Query Language)

71

Questão 10 Se um produto foi cadastrado com um nome incorreto, qual comando DML deve ser utilizado para corrigir apenas o nome daquele produto específico no banco de dados?

A) INSERT

B) UPDATE

C) DELETE

D) SELECT

72

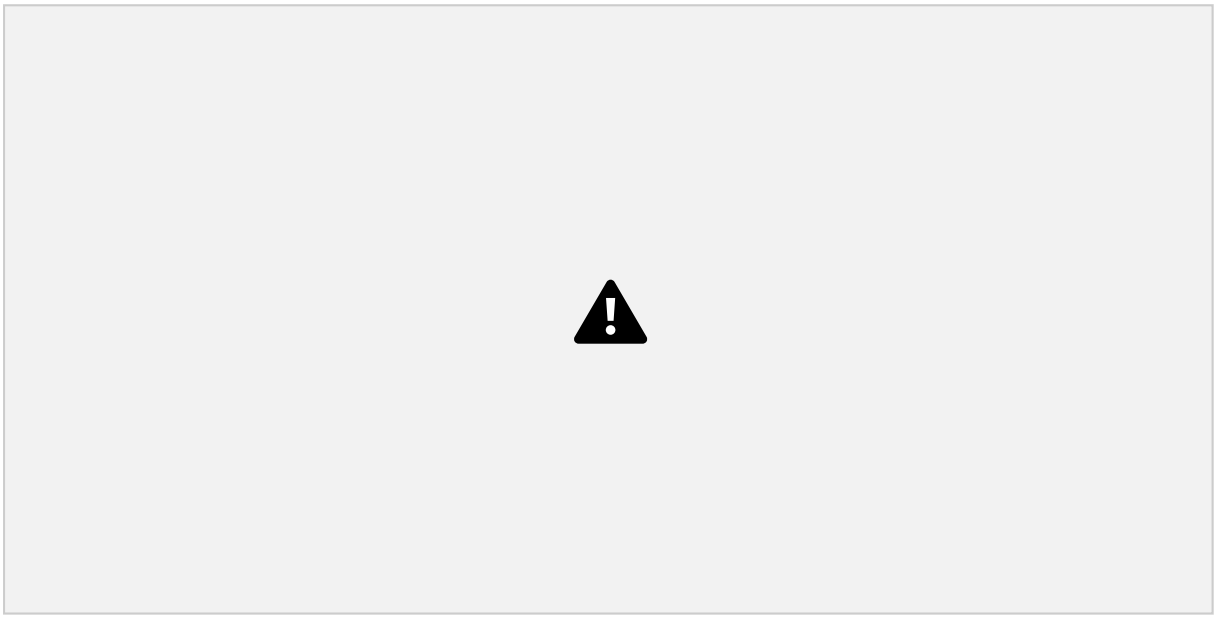
MÓDULO 5: A ARTE DA ANÁLISE - CONSULTAS AVANÇADAS

Seja bem-vindo(a) ao Módulo 5. Até agora, construímos e populamos nosso banco de dados, aprendendo a inserir, atualizar e remover registros de forma segura. Dominamos a fundação. Agora, vamos extrair o verdadeiro valor contido nesta estrutura: a **informação**.

Neste módulo, mergulharemos de cabeça na **DQL (Data Query Language)**, indo muito além do `SELECT` básico. Aprenderemos a arte de "fazer perguntas" ao banco de dados, desde as mais simples até as mais complexas. Você aprenderá a filtrar, ordenar, agregar e, o mais importante, conectar dados de múltiplas tabelas para gerar relatórios e insights que respondem a questões de negócio críticas.

Para realizar as consultas de exemplo, também utilizaremos o banco de dados *loja_online*, criado no Capítulo 11. O conteúdo já inserido será suficiente para os exercícios, mas sinta-se à vontade para adicionar mais dados (via CSV ou com ajuda de IAs) se quiser testar suas consultas em um volume maior.

Para começar, vamos analisar um modelo de solução da atividade proposta anteriormente.



Ao final, você não apenas escreverá consultas, mas traduzirá um requisito de negócio em um comando SQL preciso e eficiente.

Capítulo 14: Filtrando e Ordenando: A Busca pela Precisão

Uma consulta que retorna todos os dados de uma tabela é como receber um dicionário inteiro quando você pediu apenas a definição de uma palavra. A verdadeira habilidade está em extrair exatamente o subconjunto de dados que responde à sua pergunta. Para isso, dominaremos as cláusulas `WHERE`, `ORDER BY` e `LIMIT`.

14.1 A Cláusula WHERE: O Grande Filtro

A cláusula `WHERE` é o seu principal instrumento para filtrar dados, permitindo que você especifique as condições que os registros devem atender para serem incluídos no resultado.

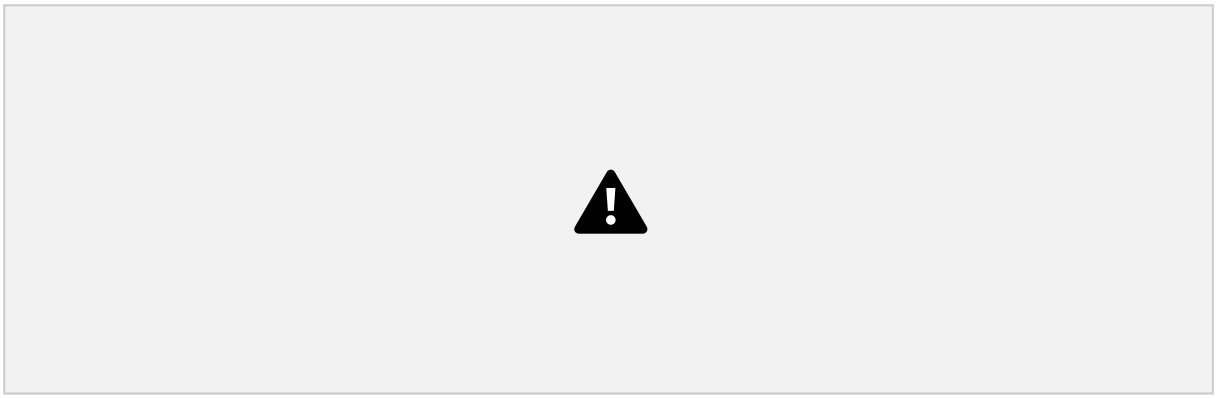
14.1.1 Operadores de Comparação

Estes são os operadores mais básicos para comparar valores.

Igualdade (`=`): `campo = 'valor';`

Contexto de Uso: Você precisa encontrar um registro específico ou um grupo de registros que compartilham um valor exato. Por exemplo, gerar um relatório de todas as vendas feitas por um funcionário específico.

Exemplo:



Diferença (≠ ou <>): `campo <> 'valor';`

Contexto de Uso: Você precisa excluir um grupo específico de registros do seu resultado. Por exemplo, listar todos os funcionários, exceto os do departamento de TI, para uma comunicação interna.

Exemplo:

Maior que (>) e Menor que (<)

Contexto de Uso: Essencial para análises baseadas em limiares. Por exemplo, identificar produtos com estoque baixo ou vendas acima de um determinado valor para bonificação.

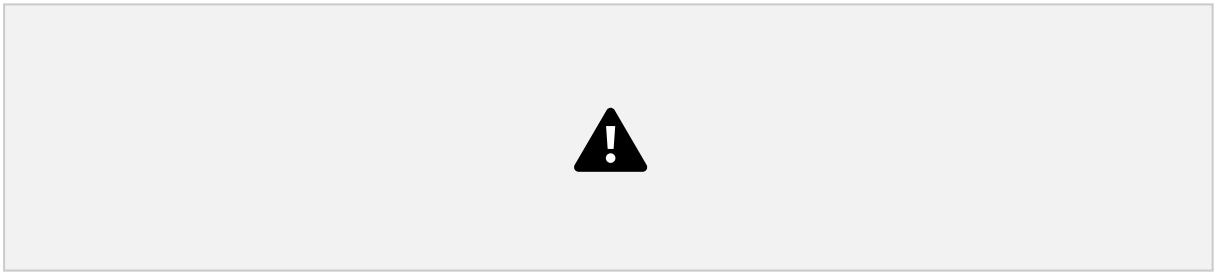
Exemplo:

Maior ou igual (≥) e Menor ou igual (≤)

Contexto de Uso: Similar ao anterior, mas inclusivo, o que é crucial para trabalhar

com faixas de valores ou datas. Por exemplo, encontrar todos os pagamentos a partir de uma determinada data, incluindo os pagamentos desta data.

Exemplo:



14.1.2 Operadores Lógicos

Estes operadores permitem combinar múltiplas condições de comparação.

AND

Contexto de Uso: Para aplicar um filtro restritivo que deve atender a múltiplos critérios simultaneamente. Por exemplo, encontrar clientes VIP que também moram em uma cidade específica. Retorna os dados **SOMENTE** se **AMBAS** as condições sejam atendidas na cláusula `WHERE`.

Exemplo:

OR

Contexto de Uso: Para aplicar um filtro abrangente que pode satisfazer qualquer um dos critérios. Por exemplo, buscar produtos que são da categoria "Periféricos" ou da marca "Multilaser".

Exemplo:

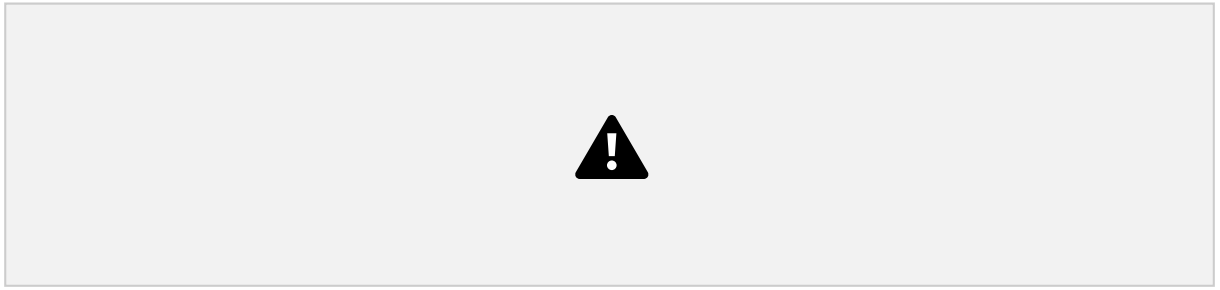
14.1.3 Operadores Especiais

Os operadores especiais são verdadeiros "canivetes suíços" da cláusula WHERE. Eles foram criados para resolver, de forma elegante e concisa, desafios de filtragem muito comuns no dia a dia. Com eles, podemos verificar se um valor está dentro de um intervalo, se pertence a uma lista de opções, se corresponde a um padrão de texto ou, de forma crucial, se um dado está ausente.

`BETWEEN`

Contexto de Uso: A forma mais clara e legível de filtrar por um intervalo de valores, incluindo os limites. Ideal para faixas de datas, preços ou idades.

Exemplo:



`IN`

Contexto de Uso: Extremamente útil para filtrar por múltiplos valores discretos, evitando o uso de vários OR. Por exemplo, selecionar todos os pedidos com status 'Enviado', 'Em trânsito' ou 'Aguardando retirada'.

Exemplo:

IN

Imagine que você precisa encontrar uma palavra em um livro, mas só se lembra de parte dela. Você não usaria o índice para procurar a palavra exata; em vez disso, você folhearia as páginas procurando por um *padrão*. O operador `LIKE` é a ferramenta que permite fazer exatamente isso em seu banco de dados.

Ele é indispensável para buscas textuais parciais, como as que acontecem em uma barra de pesquisa de um site, onde o usuário digita apenas parte do que procura. Para realizar essa "mágica", o

`LIKE` utiliza caracteres especiais chamados de **curingas** (ou *wildcards*). Vamos conhecer os dois principais:

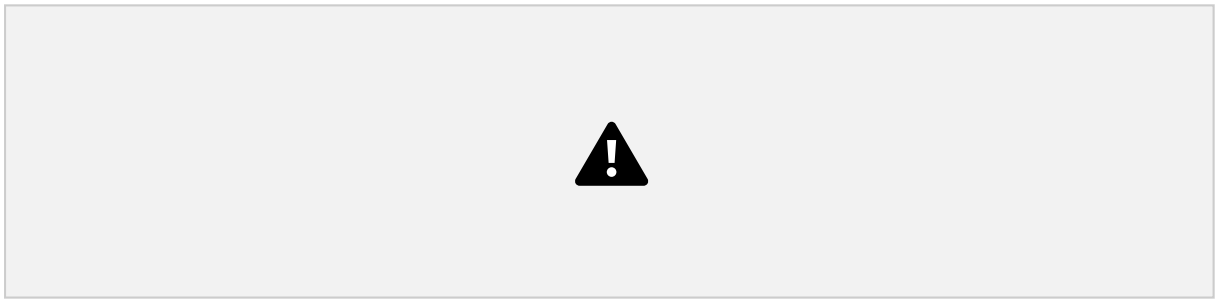
- `%` **(Porcentagem):** Representa **qualquer sequência de zero ou mais caracteres**. Pense nele como um "não importa o que venha aqui, nem a quantidade de caracteres".
- `_` **(Sublinhado):** Representa **exatamente um único caractere**. Pense nele como uma "aqui existe um caractere, mas não sei qual é".

Exemplos Práticos com o Curinga `%`

1. Encontrando um termo em qualquer posição (o mais comum)

Contexto: Você quer encontrar todos os produtos que são da linha "Gamer", não importando se a palavra "Gamer" está no início, meio ou fim do nome.

Exemplo:



Explicação: A consulta acima encontrará "Teclado Gamer", "Mousepad Gamer X" e "Cadeira Gamer Pro", porque o % antes e depois de 'Gamer' informa ao banco de dados para ignorar qualquer texto que venha antes ou depois da palavra-chave.

2. Encontrando textos que começam com um padrão

Contexto: Você precisa de uma lista de todos os clientes cujo nome começa com "Ana".

Exemplo:

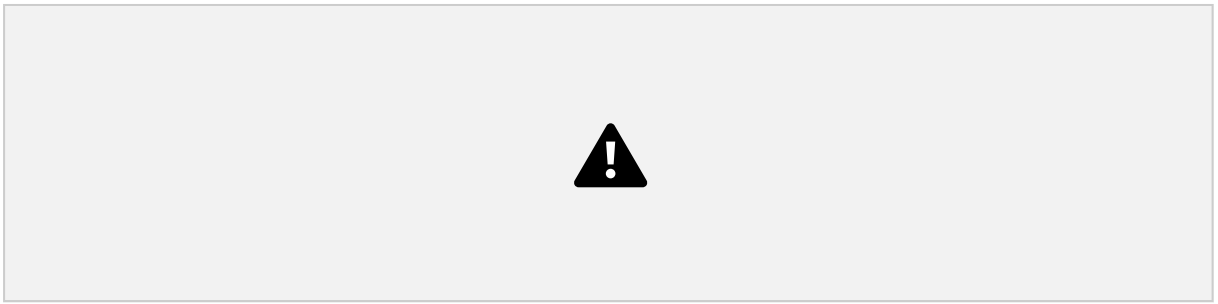
Explicação: Ao colocar o % apenas no final, a busca se torna específica: o nome deve começar exatamente com "Ana". Esta consulta retornaria "Ana Silva" e "Ana Beatriz", mas não "Mariana".

3. Encontrando textos que terminam com um padrão

Contexto: Você quer encontrar todos os arquivos de imagem cujo nome termina com a extensão ".jpg".

Exemplo:

SQL



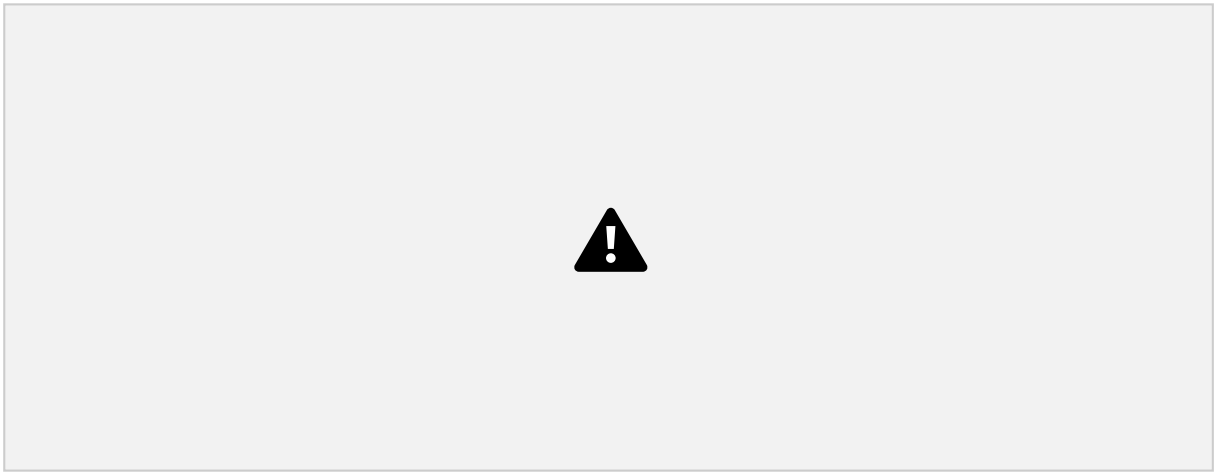
Explicação: Aqui, o oposto acontece. O % no início significa que não importa como o nome do arquivo começa, desde que termine exatamente com ".jpg".

Exemplos Práticos com o Curinga _

1. Substituindo um único caractere

Contexto: Em um cadastro, alguns nomes foram digitados com variações, como "Luiza" e "Luisa". Você quer encontrar ambos.

Exemplo:



80

Explicação: O _ ocupa o lugar de um único caractere. Ele corresponde tanto ao "z" de "Luiza" quanto ao "s" de "Luisa", tornando a busca mais flexível.

2. Verificando um padrão com tamanho fixo

Contexto: Você precisa encontrar todos os clientes com nomes de três letras que terminam com "na", como "Ana" ou "Lna".

Exemplo:

Explicação: Esta consulta especifica um padrão rígido: um caractere qualquer no início, seguido exatamente por "na". Encontraria "Ana" e "Lna", mas não "Joana" (que tem mais caracteres no início) ou "An" (que não tem um caractere no início).

IS NULL

Contexto de Uso: Fundamental para encontrar registros com dados ausentes. Por exemplo, identificar clientes que não preencheram o endereço de entrega ou produtos que ainda não têm uma descrição, muito utilizado para campos de data/hora, principalmente quando utilizando a abordagem de exclusão lógica.

Exemplo: