ANOTAÇÕES DOS ESTUDOS DE BANCO DE DADOS (MYSQL)

FASE 2: Modelagem de dados

Sumário

1. Modelagem Conceitual

1.1. O que é?

A modelagem conceitual é a primeira etapa do processo de design de um banco de dados.

Seu objetivo principal é representar de forma abstrata e visual a estrutura e os relacionamentos dos dados que serão armazenados no sistema, sem considerar aspectos técnicos ou específicos de implementação.

Essa etapa utiliza ferramentas como o **Modelo Entidade-Relacionamento (MER)** para descrever:

1. **Entidades**: Representam os objetos ou conceitos principais do sistema (ex.: clientes, produtos).
2. **Atributos**: Representam as características ou propriedades das entidades (ex.: nome, endereço, preço).
3. **Relacionamentos**: Representam as associações entre as entidades (ex.: um cliente faz um pedido).

- **Por que é importante?**

1. **Clareza**: Ajuda a equipe (técnica e não técnica) a entender os requisitos de dados do sistema.
2. **Evita erros**: Identifica problemas e inconsistências antes da implementação.
3. **Planejamento**: Serve como base para as fases seguintes da modelagem (lógica e física).

1.2. Elementos da Modelagem Conceitual

1. Entidades

- São os objetos principais que queremos representar no banco de dados.
- **Exemplo**: Em uma loja, entidades podem ser:
    - **Clientes**: Pessoas que compram produtos.
    - **Produtos**: Itens vendidos na loja.
    - **Pedidos**: Compras feitas pelos clientes.

2. Atributos

- São as propriedades ou características que descrevem as entidades.
- **Exemplo**:
    - Entidade Clientes:
        - Atributos: Nome, CPF, Telefone, Email.
    - Entidade Produtos:
        - Atributos: NomeProduto, Preco, Estoque.

3. Relacionamentos

- Representam como as entidades estão associadas entre si.
- **Tipos**:
    - **1:1 (Um para Um)**: Um cliente tem apenas um cartão de fidelidade.
    - **1:N (Um para Muitos)**: Um cliente pode fazer muitos pedidos.
    - **N:N (Muitos para Muitos)**: Muitos produtos podem estar em muitos pedidos.

4. Cardinalidade

- Define o número de ocorrências que uma entidade pode ter em um relacionamento.
    - **1:1**: Cada cliente tem um único endereço.
    - **1:N**: Um pedido pode ter muitos itens.
    - **N:N**: Muitos alunos podem estar em muitos cursos.

5. Identificadores (Chaves Primárias)

- É o atributo que identifica exclusivamente uma entidade.
    - Exemplo: ClienteID para a entidade Clientes.

6. Generalização e Especialização (opcional)

- **Generalização**: Quando agrupamos entidades semelhantes em uma mais genérica.

    - Exemplo: Entidade Pessoa pode ser generalizada em Clientes e Funcionários.
- **Especialização**: Quando detalhamos uma entidade genérica em mais específicas.
    - Exemplo: Entidade Pessoa se divide em Aluno e Professor.

1.3. Ferramentas para Modelagem Conceitual

- **Modelo Entidade-Relacionamento (MER)**
    - É uma ferramenta gráfica que utiliza símbolos para representar entidades, atributos e relacionamentos.

Símbolo	Representação
Retângulo	Entidade
Elipse	Atributo
Losango	Relacionamento
Linha	Conexão

Exemplo de MER para uma Loja:

```
css

[Clientes] --- faz ---< [Pedidos] >--- contém ---< [Produtos]
```

1.4. Cardinalidade e Tipos de Relacionamentos

Os relacionamentos definem como as entidades se conectam entre si em um banco de dados.

Já a cardinalidade especifica a quantidade de ocorrências de uma entidade que podem estar associadas a outra entidade em um relacionamento.

- **Tipos de Relacionamentos**

1. **Relacionamento 1:1 (Um para Um)**
    - **Descrição**: Uma entidade A está associada a, no máximo, uma entidade B, e vice-versa.

- **Exemplo**:
  - Um **cliente** possui um único **cartão de fidelidade**.
  - Um **funcionário** tem apenas um **armário pessoal**.
- **Implementação**:
  - Ambas as tabelas podem conter uma **chave estrangeira (FK)** que referência a chave primária da outra tabela.

2. **Relacionamento 1:N (Um para Muitos)**
   - **Descrição**: Uma entidade A pode estar associada a várias entidades B, mas cada entidade B está associada a apenas uma entidade A.
   - **Exemplo**:
     - Um **cliente** pode fazer **muitos pedidos**, mas cada pedido pertence a um único cliente.
     - Um **autor** pode escrever **vários livros**, mas cada livro tem apenas um autor.
   - **Implementação**:
     - A tabela "muitos" contém uma **chave estrangeira (FK)** que referência a chave primária da tabela "um".

3. **Relacionamento N:N (Muitos para Muitos)**
   - **Descrição**: Uma entidade A pode estar associada a várias entidades B, e cada entidade B pode estar associada a várias entidades A.
   - **Exemplo**:
     - Muitos **alunos** podem estar matriculados em muitos **cursos**.
     - Muitos **médicos** podem atender a muitos **pacientes**.
   - **Implementação**:
     - É necessário criar uma **tabela intermediária** para armazenar as associações.
     - Exemplo:

        - Tabelas: Alunos, Cursos.
        - Tabela intermediária: Matriculas (contendo chaves estrangeiras AlunoID e CursoID).

- **Cardinalidade**

A **cardinalidade** é a especificação numérica que descreve quantas ocorrências de uma entidade podem estar associadas a outra entidade em um relacionamento.

Representação no Modelo Entidade-Relacionamento (MER):

- **1:1 (Um para Um)**:
  Cada registro de A está associado a um único registro de B.

- **1:N (Um para Muitos)**:
  Cada registro de A pode estar associado a muitos registros de B.

- **N:N (Muitos para Muitos)**:
  Muitos registros de A podem estar associados a muitos registros de B.

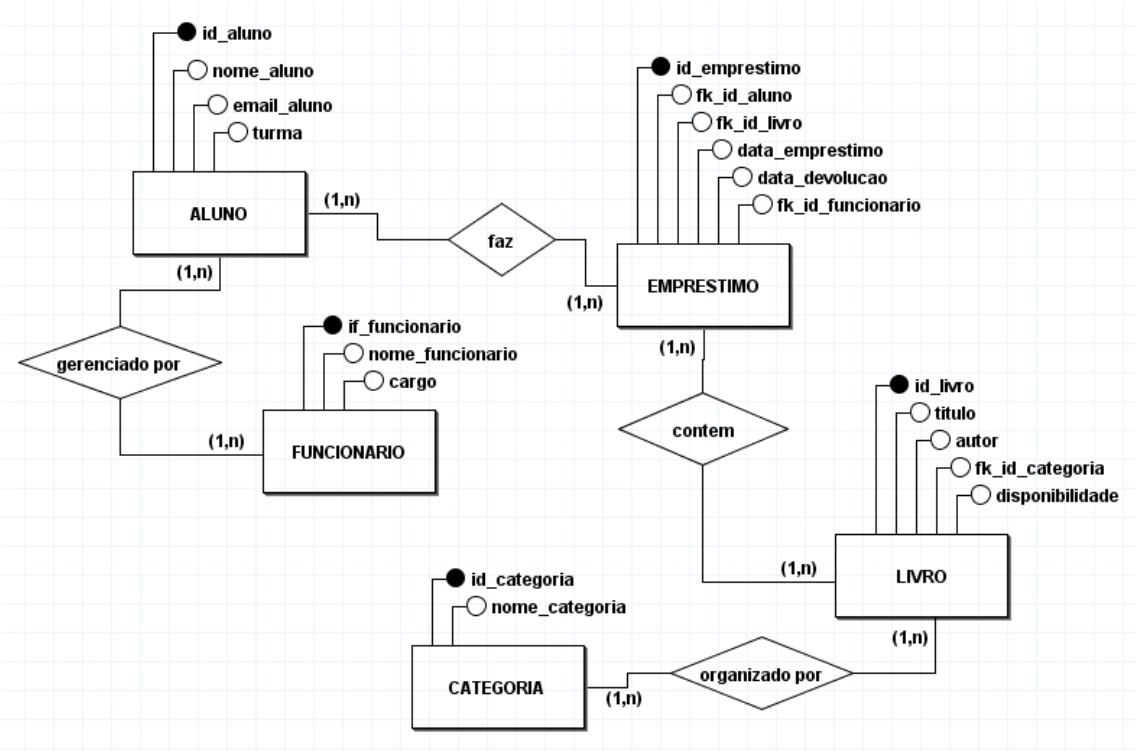
- **Cardinalidade Mínima e Máxima**
    - Além de indicar se o relacionamento é 1:1, 1:N ou N:N, é possível especificar a quantidade mínima e máxima de registros permitidos no relacionamento.
        - Cardinalidade mínima: Determina se a participação no relacionamento é obrigatória (1) ou opcional (0).
        - Cardinalidade máxima: Determina o número máximo de associações permitidas (1, N ou outro número).

1.5. Exemplo Prático de modelagem conceitual

Descrição do Sistema:
Imagine que você precisa projetar um sistema para gerenciar os livros e empréstimos de uma **biblioteca escolar**. O sistema precisa registrar:

- Os livros disponíveis.
- Os alunos que pegam livros emprestados.
- Os empréstimos e devoluções.
- Os funcionários responsáveis pela biblioteca.

Modelo Conceitua (*Diagrama*):



- **Entidades:**
  1. Aluno (quem pega livros emprestados).
  2. Livro (os livros disponíveis na biblioteca).
  3. Empréstimo (registra os livros emprestados).
  4. Funcionário (bibliotecário que gerencia os empréstimos).
  5. Categoria (classificação dos livros).

- **Relacionamentos:**

Relacionamento	Tipo	Explicação
Aluno → Empréstimo	1:N	Um aluno pode pegar vários livros emprestados.
Livro → Empréstimo	1:N	Um livro pode ser emprestado várias vezes.
Funcionário → Empréstimo	1:N	Um funcionário pode registrar vários empréstimos.
Categoria → Livro	1:N	Uma categoria pode conter vários livros.

2. Modelagem Lógica

A modelagem lógica é a etapa intermediária entre a modelagem conceitual e a modelagem física de um banco de dados. **O objetivo principal da modelagem lógica é traduzir o modelo conceitual**, que geralmente utiliza diagramas ER (Entidade-Relacionamento), para um modelo mais detalhado, **que seja compreensível pelo sistema gerenciador de banco de dados (SGBD).**

- **Objetivo da Modelagem Lógica**
  1. **Refinar as entidades** identificadas na modelagem conceitual.
  2. **Definir atributos** com mais detalhes, incluindo os tipos de dados.
  3. **Estabelecer relacionamentos e cardinalidades** em um formato adequado para a criação de tabelas.
  4. **Normalizar o banco de dados** para eliminar redundâncias e inconsistências.
  5. Adaptar o modelo para o **paradigma relacional**, onde dados são organizados em tabelas com chaves primárias e estrangeiras.

- **Passos para Criar um Modelo Lógico**
  1. **Identificar as tabelas**
     - Cada **entidade** do modelo conceitual será transformada em uma **tabela**.
     - Relacionamentos podem se transformar em tabelas separadas (no caso de N:N) ou ser representados por chaves estrangeiras.
  2. **Definir os atributos das tabelas**
     - Para cada entidade, determine os **atributos** e escolha o **tipo de dado** apropriado (inteiro, texto, data, etc.).
     - Exemplo:
       - Entidade: Cliente
       - Atributos: ClienteID, Nome, Email, DataNascimento.
  3. **Definir chaves primárias (PK)**

- Escolha um ou mais atributos que serão **unicidade** da tabela.
- Exemplo: Na tabela Cliente, o atributo ClienteID pode ser a chave primária.

4. **Estabelecer chaves estrangeiras (FK)**
    - Adicione **referências** entre tabelas para representar os relacionamentos.
    - Exemplo:
        - Relacionamento Pedido ↔ Cliente: A tabela Pedido terá um atributo ClienteID como chave estrangeira.

5. **Normalizar o modelo**
    - Aplique as **formas normais** para evitar duplicação de dados e inconsistências.
    - Exemplo: Divida uma tabela que contém Cliente e Endereço em duas tabelas separadas, se cada cliente puder ter vários endereços.

2.1. Normalização de Banco de Dados

A normalização é o processo de organizar os dados em um banco de dados para reduzir redundâncias e melhorar a integridade e a eficiência. Ela é realizada através da aplicação de formas normais (FN), que são regras que ajudam a estruturar tabelas e relacionamentos de forma lógica e eficiente.

- **Por Que Normalizar?**
    - **Reduzir redundância**: Evita armazenar as mesmas informações em múltiplos locais.
    - **Melhorar a consistência**: Minimiza o risco de inconsistências nos dados.
    - **Aprimorar a manutenção**: Facilita a atualização, inserção e remoção de dados.
    - **Otimizar o desempenho**: Evita tabelas excessivamente grandes e complexas.

2.1.1. Formas Normais (FN)

As formas normais são estágios de normalização, com cada estágio resolvendo problemas mais complexos de redundância e inconsistência.

- **1ª Forma Normal (1NF)**
  - **Objetivo**: Garantir que cada coluna de uma tabela contenha apenas valores atômicos (indivisíveis) e que não existam grupos repetitivos.
  - **Regras**:
    1. Cada coluna deve conter valores únicos para o mesmo tipo de dado.
    2. Cada célula deve conter apenas um valor (dados atômicos).
    3. Cada linha deve ser única, identificada por uma **chave primária**.

Exemplo de Desnormalização:

```
plaintext

ClienteID | Nome       | Telefones
----------|------------|----------------
1         | João Silva | 123-456, 789-012
2         | Maria Dias | 345-678
```

Problema: A coluna "Telefones" contém múltiplos valores.

Após Normalizar (1NF):

```
plaintext

ClienteID | Nome       | Telefone
----------|------------|---------
1         | João Silva | 123-456
1         | João Silva | 789-012
2         | Maria Dias | 345-678
```

- **2ª Forma Normal (2NF)**
  - **Objetivo**: Remover dependências parciais, garantindo que todos os atributos dependam da chave primária inteira, e não de apenas parte dela (em tabelas com chaves compostas).
  - **Pré-requisito**: A tabela deve estar na 1ª FN.
  - **Regras**:
    1. Identifique dependências parciais.

2. Crie tabelas separadas para eliminar essas dependências.

Exemplo de Desnormalização:

```
plaintext

PedidoID | ProdutoID | NomeProduto | Quantidade
---------|-----------|-------------|-----------
1        | 101       | Caneta      | 2
1        | 102       | Caderno     | 3
2        | 101       | Caneta      | 1
```

Problema: O atributo NomeProduto depende apenas de ProdutoID, não de toda a chave primária (PedidoID, ProdutoID).

- **Após Normalizar (2NF):**

Tabela Pedidos:

```
plaintext

PedidoID | ProdutoID | Quantidade
---------|-----------|-----------
1        | 101       | 2
1        | 102       | 3
2        | 101       | 1
```

Tabela Produtos:

```
plaintext

ProdutoID | NomeProduto
----------|------------
101       | Caneta
102       | Caderno
```

- **3ª Forma Normal (3NF)**
    - **Objetivo**: Eliminar dependências transitivas, ou seja, um atributo não-chave não pode depender de outro atributo não-chave.
    - **Pré-requisito**: A tabela deve estar na 2ª FN.
    - **Regras**:
        1. Remova atributos que dependam de outros atributos não-chave.
        2. Crie tabelas separadas para esses atributos.

Exemplo de Desnormalização:

```
plaintext

ClienteID | Nome       | CPF       | Cidade
----------|------------|-----------|-------
1         | João Silva | 123456789 | São Paulo
2         | Maria Dias | 987654321 | Recife
```

Problema: A coluna "Cidade" pode depender de "CPF" ou "ClienteID", mas pode ser melhor estruturada.

- **Após Normalizar (3NF):**

Tabela Clientes:

```
plaintext

ClienteID | Nome       | CPF
----------|------------|----------
1         | João Silva | 123456789
2         | Maria Dias | 987654321
```

Tabela Cidades:

```
plaintext

ClienteID | Cidade
----------|------------
1         | São Paulo
2         | Recife
```

- **Outras Formas Normais**
  - 4ª Forma Normal (4NF): Remove dependências multivaloradas. Uma tabela deve ter apenas uma dependência multivalorada por vez.
  - 5ª Forma Normal (5NF): Garante que nenhuma informação redundante possa ser recriada através de tabelas relacionadas.

- **Quando Normalizar e Quando Não Normalizar**
  - **Normalizar**: Quando a consistência e a integridade dos dados são cruciais.
  - **Desnormalizar**: Em sistemas com alta demanda de leitura, onde é importante priorizar a velocidade de consulta.

3. Modelagem Física

A modelagem física de dados é a etapa final do processo de modelagem, onde o projeto lógico é transformado em estruturas concretas e

específicas para um Sistema de Gerenciamento de Banco de Dados (SGBD), como MySQL, PostgreSQL, SQL Server, entre outros.

Esta etapa é responsável por definir como os dados serão fisicamente armazenados no banco, otimizando o desempenho e garantindo a integridade e segurança dos dados.

- **O Que Inclui a Modelagem Física?**

1. **Escolha do SGBD**
    - Selecionar o sistema de banco de dados que melhor atende às necessidades do projeto (por exemplo, MySQL).
    - Avaliar requisitos como volume de dados, segurança, e necessidade de escalabilidade.
2. **Criação de Tabelas**
    - Especificar os nomes das tabelas e as colunas.
    - Definir os tipos de dados para cada coluna, de acordo com o SGBD.
        - Exemplo: INT, VARCHAR, DATE no MySQL.
    - Incluir restrições (constraints):
        - **Chaves primárias**: Para identificar unicamente os registros.
        - **Chaves estrangeiras**: Para manter os relacionamentos entre tabelas.
        - **Índices**: Para otimizar consultas.
3. **Tipos de Dados**
    - Escolher tipos de dados apropriados para cada coluna para economizar espaço e garantir precisão.
        - Exemplo:
            - INT para números inteiros.
            - DECIMAL ou FLOAT para valores decimais.
            - VARCHAR para textos variáveis.
4. **Índices**
    - Criar índices para melhorar o desempenho das consultas.
    - Índices podem ser:

        - **Primários**: Criados automaticamente pela chave primária.
        - **Secundários**: Para acelerar buscas em colunas específicas.
        - **Compostos**: Baseados em múltiplas colunas.
5. **Particionamento de Tabelas**
    - Dividir grandes tabelas em partes menores para melhorar o desempenho.
    - Pode ser baseado em:
        - Intervalos de valores (por exemplo, datas).
        - Intervalos de chaves (por exemplo, IDs).
6. **Definição de Regras de Integridade**
    - Garantir consistência e validade dos dados.
    - Exemplos:
        - **NOT NULL**: Para evitar valores nulos em colunas obrigatórias.
        - **DEFAULT**: Para atribuir valores padrão.
        - **CHECK**: Para validar valores com condições.
7. **Normalização e Denormalização**
    - Revisar o modelo lógico e decidir até que ponto o banco será normalizado.
    - Em alguns casos, denormalizar (reintroduzir redundâncias) para melhorar a performance de leitura.
8. **Políticas de Armazenamento**
    - Configurar tabelaspaces e discos físicos para distribuir a carga de leitura e escrita.
    - Definir tamanhos iniciais das tabelas e sua expansão (exemplo: auto-incremento de IDs).

Diferença Entre Modelagem Lógica e Física

Aspecto	Modelagem Lógica	Modelagem Física
Foco	Representação abstrata dos dados	Implementação concreta no SGBD
Independência	Independente do SGBD	Específico para o SGBD escolhido
Elementos	Entidades, atributos, relacionamentos	Tabelas, colunas, índices, tipos de dados
Regras	Integridade referencial e restrições lógicas	Integridade física, desempenho e estrutura de armazenamento
Objetivo	Planejamento e organização	Implementação real e otimização

3.1. Tipos de Dados

Os tipos de dados determinam o tipo de valor que uma coluna em uma tabela pode armazenar.

Escolher os tipos de dados adequados é fundamental para a eficiência, precisão e integridade de um banco de dados.

3.1.1. Tipos Numéricos

- **Inteiros (Números Inteiros)**
  - Armazenam números inteiros (sem casas decimais).

TINYINT	Inteiro muito pequeno. • Intervalo: -128 a 127 (ou 0 a 255 para valores sem sinal). • Uso: Contadores pequenos, status binário (0 ou 1).
SMALLINT	Inteiro pequeno. • Intervalo: -32.768 a 32.767.
MEDIUMINT	Inteiro médio. • Intervalo: -8.388.608 a 8.388.607.
INT OU INTEGER	Inteiro padrão. • Intervalo: -2.147.483.648 a 2.147.483.647.
BIGINT	Inteiro muito grande. • Intervalo: -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. • Uso: Contadores de grandes volumes de dados (exemplo: IDs em sistemas massivos).

- **Pontos Flutuantes (Números Decimais e Reais)**
    - Armazenam números com casas decimais.

FLOAT (precisão)	Números com ponto flutuante de precisão simples. • Uso: Valores aproximados em cálculos matemáticos (exemplo: medições, coordenadas).
DOUBLE ou REAL	Números com ponto flutuante de precisão dupla. • Uso: Cálculos científicos ou financeiros de alta precisão.
DECIMAL ou NUMERIC	Números decimais com precisão exata. • Especificação: DECIMAL (M, D) (M = dígitos totais, D = dígitos após o ponto decimal). • Uso: Dados financeiros (exemplo: preços).

3.1.2. Tipos de Dados de Texto

- **Texto de Tamanho Fixo**
    - **CHAR(X)**: Cadeia de caracteres de tamanho fixo.
        - Exemplo: CHAR (3) armazena exatamente 3 caracteres.
        - Uso: Armazenar códigos fixos, como siglas ou códigos de países.

- **Texto de Tamanho Variável**
    - **VARCHAR(X)**: Cadeia de caracteres de tamanho variável (até X caracteres).
    - Uso: Nomes, descrições, e textos de comprimento variável.

- **Texto Longo**
    - **TINYTEXT**: Até 255 caracteres.
    - **TEXT**: Até 65.535 caracteres.
    - **MEDIUMTEXT**: Até 16.777.215 caracteres.
    - **LONGTEXT**: Até 4.294.967.295 caracteres.
        - Uso: Comentários, artigos, ou qualquer conteúdo extenso.

3.1.3. Tipos de Dados de Data e Hora

- **DATE**: Armazena apenas a data (formato AAAA-MM-DD).
    - Intervalo: 1000-01-01 a 9999-12-31.
    - Uso: Aniversários, datas de pedidos.

- **DATETIME**: Armazena data e hora (formato AAAA-MM-DD HH:MM:SS).
    - Uso: Marcar eventos exatos no tempo.

- **TIMESTAMP**: Semelhante ao DATETIME, mas baseado no horário UTC (Coordinated Universal Time).
    - Intervalo: 1970-01-01 00:00:01 UTC a 2038-01-19 03:14:07 UTC.
    - Uso: Registro de logs ou atualizações.

- **TIME**: Armazena apenas o horário (formato HH:MM:SS).
    - Uso: Duração de eventos.

- **YEAR**: Armazena apenas o ano (formato AAAA).
    - Intervalo: 1901 a 2155.

3.1.4. Tipos de Dados Lógicos (Booleanos)

- **BOOLEAN** ou **BOOL**: Representa valores verdadeiros ou falsos.
    - No MySQL, é uma abreviação para TINYINT(1):
        - 1 para verdadeiro.
        - 0 para falso.
    - Uso: Flags, status binário, ativações.

3.1.5. Tipos de Dados Binários

- **BINARY(M)**: Cadeia binária de tamanho fixo.
    - Uso: Armazenar hashes ou valores binários curtos.

- **VARBINARY(M)**: Cadeia binária de tamanho variável.
    - Uso: Armazenar arquivos pequenos, como imagens ou vídeos curtos em formato binário.

- **BLOB (Binary Large Object)**: Armazena dados binários de tamanhos variados.
    - **TINYBLOB**: Até 255 bytes.
    - **BLOB**: Até 65.535 bytes.
    - **MEDIUMBLOB**: Até 16.777.215 bytes.
    - **LONGBLOB**: Até 4.294.967.295 bytes.
    - Uso: Arquivos grandes, como imagens, vídeos e documentos.

3.1.6. Escolha do tipo de Dados

A escolha correta do tipo de dado impacta diretamente:

1. **Espaço de Armazenamento**:
    - Escolher tipos menores para economizar espaço.
    - Exemplo: Use TINYINT para números pequenos em vez de INT.
2. **Desempenho**:
    - Tipos adequados ajudam em consultas rápidas.
    - Exemplo: Use CHAR para tamanhos fixos em vez de VARCHAR se o comprimento for sempre o mesmo.
3. **Validação de Dados**:
    - Tipos ajudam a validar a entrada de dados.
    - Exemplo: Apenas datas válidas são aceitas em colunas do tipo DATE.

3.2. Restrições em Banco de Dados

As restrições são regras aplicadas às colunas de uma tabela para garantir a integridade, a consistência e a precisão dos dados.

Elas são essenciais para evitar erros e proteger a integridade das informações no banco de dados.

3.2.1. Comando ADD CONSTRAINT

O comando ADD CONSTRAINT é utilizado no MySQL (e em outros bancos de dados relacionais) para adicionar restrições a uma tabela. Ele é normalmente usado com a

instrução ALTER TABLE, permitindo adicionar chaves primárias, estrangeiras, restrições de unicidade e verificações em colunas já existentes.

É possível a utilização desse comando com todas as restrições abaixo e o mesmo tem a essencialidade de garantir a integridade dos dados, criando regras que evitam valores inconsistentes no banco, além de poder alterar, apagar ou adicionar novas restrições em uma tabela já criada, veja um exemplo:

- **Criando uma FOREIGN KEY utilizando o ADD CONSTRAINT:**

```
ALTER TABLE pedidos
ADD CONSTRAINT fk_pedido_cliente FOREIGN KEY (id_cliente)
REFERENCES clientes (id_cliente);
```

ADD CONSTRAINT -> Cria uma chave estrangeira chamada *fk_pedido_cliente* na tabela *pedidos*.

A coluna *id_cliente* da tabela pedidos agora precisa ter um valor que exista na tabela clientes (*id_cliente*).

- **Removendo uma CONSTRAINT já criada:**

```
ALTER TABLE clientes
DROP CONSTRAINT unique_email;
```

Remove a restrição de unicidade no campo e-mail que foi criada na tabela clientes.

3.2.2. NOT NULL

- **Descrição**: Garante que uma coluna não pode conter valores nulos (NULL).
- **Uso**: Evita que campos obrigatórios fiquem sem valor.
- **Exemplo**:

```sql
CREATE TABLE cliente (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(150)
);
```

O campo nome não pode ser nulo;
é obrigatório informar um valor ao inserir dados.

3.2.3. UNIQUE

- **Descrição**: Garante que todos os valores de uma coluna ou combinação de colunas sejam únicos.
- **Uso**: Evita duplicação de dados, como e-mails ou CPF.
- **Exemplo:**

```sql
CREATE TABLE usuario (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(150) UNIQUE
);
```

- Nenhum usuário pode ter o mesmo e-mail.

3.2.4. PRIMARY KEY

- **Descrição**: Define uma coluna (ou conjunto de colunas) como identificador exclusivo da tabela. Combina as restrições NOT NULL e UNIQUE.
- **Uso**: Garantir que cada registro em uma tabela seja único.
- **Exemplo**:

```sql
CREATE TABLE pedido (
    id INT AUTO_INCREMENT PRIMARY KEY,
    data DATE NOT NULL
);
```

O campo *id* identifica exclusivamente cada pedido.

3.2.5. FOREIGN KEY

- **Descrição**: Assegura que o valor de uma coluna corresponde a um valor em outra tabela, criando um relacionamento entre elas.
- **Uso**: Manter a integridade referencial entre tabelas.
- **Exemplo**:

```sql
CREATE TABLE cliente (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL
);

CREATE TABLE pedido (
    id INT AUTO_INCREMENT PRIMARY KEY,
    cliente_id INT,
    FOREIGN KEY (cliente_id) REFERENCES cliente(id)
);
```

A chave estrangeira *cliente_id* em pedido deve corresponder a um id existente na tabela cliente.

- **Outra maneira de criar FK:**

```sql
CREATE TABLE pedidos (
    id_pedido INT PRIMARY KEY,
    id_cliente INT,
    data_pedido DATE NOT NULL,
    CONSTRAINT fk_pedido_cliente FOREIGN KEY (id_cliente)
        REFERENCES clientes (id_cliente)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

1. *id_cliente* na tabela *pedidos* referencia *id_cliente* na tabela *clientes*.
2. *ON DELETE CASCADE*: Se um cliente for deletado, os pedidos dele também serão excluídos.
3. *ON UPDATE CASCADE*: Se *id_cliente* mudar na tabela *clientes*, ele também será atualizado na tabela pedidos.

- **Criando FK com ALTER TABLE:**

```sql
ALTER TABLE pedidos
ADD CONSTRAINT fk_pedido_cliente
FOREIGN KEY (id_cliente)
REFERENCES clientes (id_cliente)
ON DELETE SET NULL
ON UPDATE CASCADE;
```

1. Se o cliente for deletado, *id_cliente* em *pedidos* será definido como NULL (*ON DELETE SET NULL*).
2. Se *id_cliente* mudar na tabela *clientes*, ele também será atualizado em pedidos (*ON UPDATE CASCADE*).

- **Deletando uma FK (*DROP CONSTRAINT*)**

Caso precise **remover uma FOREIGN KEY**, use:

```sql
ALTER TABLE pedidos
DROP FOREIGN KEY fk_pedido_cliente;
```

- Deve-se observar o seguinte:
  - O nome da chave estrangeira deve ser exatamente o usado na criação (*fk_pedido_cliente*).

3.2.6. CHECK

- **Descrição**: Define uma condição que os valores inseridos em uma coluna devem atender.
- **Uso**: Garantir que os valores estejam dentro de um intervalo ou satisfaçam uma condição lógica.
- **Exemplo**:

```sql
CREATE TABLE funcionario (
    id INT AUTO_INCREMENT PRIMARY KEY,
    salario DECIMAL(10, 2) CHECK (salario > 0)
);
```

O salário deve ser maior que zero.

3.2.7. DEFAULT

- **Descrição**: Especifica um valor padrão para uma coluna, caso nenhum valor seja informado ao inserir dados.
- **Uso**: Preencher automaticamente campos comuns quando o valor não for fornecido.
- **Exemplo**:

```
CREATE TABLE produto (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    estoque INT DEFAULT 0
);
```

O estoque será 0 se nenhum valor for informado.

3.2.8. AUTO_INCREMENT

- **Descrição**: Atribui automaticamente um número único e crescente a uma coluna, geralmente usada para chaves primárias.
- **Uso**: Gerar identificadores exclusivos automaticamente.
- **Exemplo**:

```
CREATE TABLE categoria (
    id INT AUTO_INCREMENT PRIMARY KEY,
    descricao VARCHAR(100)
);
```

A coluna *id* será preenchida automaticamente com números sequenciais.

3.2.9. ON DELETE e ON UPDATE

- **Descrição**: Controla o que acontece com os registros filhos quando um registro pai é excluído ou atualizado.
  Essas opções são usadas com a restrição FOREIGN KEY para definir o comportamento ao excluir ou atualizar os registros relacionados.
- **Modos disponíveis**:
  - CASCADE: Propaga a ação para os registros filhos (excluir ou atualizar).
  - SET NULL: Define o valor da chave estrangeira como NULL.
  - SET DEFAULT: Define o valor como o padrão especificado.

- RESTRICT: Impede a ação se existirem registros filhos relacionados.
- NO ACTION: Similar ao RESTRICT, mas a verificação é feita apenas após a execução do comando.

- **Exemplo:**

```
CREATE TABLE cliente (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL
);

CREATE TABLE pedido (
    id INT AUTO_INCREMENT PRIMARY KEY,
    cliente_id INT,
    FOREIGN KEY (cliente_id) REFERENCES cliente(id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

Se um cliente for excluído, todos os pedidos relacionados também serão removidos.

3.2.10. ENUM

- **Descrição**: Define uma lista pré-determinada de valores para uma coluna.
- **Uso**: Restringir os valores possíveis para campos como status ou categorias.
- **Exemplo**:

```
CREATE TABLE produto (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100),
    categoria ENUM('Eletrônicos', 'Roupas', 'Alimentos') NOT NULL
);
```

A coluna *categoria* só aceita os valores *'Eletrônicos', 'Roupas' ou 'Alimentos'*.

3.2.11. SET

- **Descrição**: Semelhante ao ENUM, mas permite que uma coluna contenha múltiplos valores pré-definidos, separados por vírgulas.
- **Uso**: Para armazenar múltiplas opções selecionadas.

- **Exemplo**:

```
CREATE TABLE cliente_preferencias (
    id INT AUTO_INCREMENT PRIMARY KEY,
    preferencia SET('Promoções', 'Novidades', 'Eventos') NOT NULL
);
```

Um cliente pode escolher uma ou mais preferências.

3.2.12. UNIQUE (composto)

- **Descrição**: Define que uma combinação de valores em várias colunas deve ser única.
- **Uso**: Evitar duplicação em tabelas que dependem de múltiplas chaves.
- **Exemplo**:

```
CREATE TABLE matricula (
    aluno_id INT,
    curso_id INT,
    PRIMARY KEY (aluno_id, curso_id),
    UNIQUE (aluno_id, curso_id)
);
```

Garante que um aluno não possa se matricular no mesmo curso mais de uma vez.

3.2.13. GENERATED AS / EXPRESSÕES COMPUTADAS

- **Descrição**: Permite criar colunas com valores calculados automaticamente com base em outras colunas.
- **Uso**: Automatizar cálculos ou formatações.
- **Exemplo**:

```
CREATE TABLE vendas (
    id INT AUTO_INCREMENT PRIMARY KEY,
    preco_unitario DECIMAL(10, 2),
    quantidade INT,
    total DECIMAL(10, 2) GENERATED ALWAYS AS (preco_unitario * quantidade) STORED
);
```

A coluna *total* é calculada automaticamente com base no preço e na quantidade.

3.2.14. CHECK (Composto)

- **Descrição**: Pode combinar múltiplas condições para validar dados em uma ou mais colunas.
- **Exemplo**:

```sql
CREATE TABLE funcionario (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    idade INT CHECK (idade >= 18 AND idade <= 65),
    salario DECIMAL(10, 2) CHECK (salario > 0)
);
```

Combina restrições para idade e salário em uma mesma tabela.

Resumo das Restrições

Restrição	Descrição
NOT NULL	Garante que a coluna não pode conter valores nulos.
UNIQUE	Garante que os valores sejam únicos.
PRIMARY KEY	Combina `NOT NULL` e `UNIQUE` para identificar registros de forma única.
FOREIGN KEY	Relaciona tabelas, garantindo integridade referencial.
CHECK	Define condições lógicas para valores aceitáveis.
DEFAULT	Define valores padrão para uma coluna.
AUTO_INCREMENT	Incrementa automaticamente valores inteiros.

Tipo de Restrição	Descrição	Exemplo
Integridade de Dados	`NOT NULL`, `CHECK`, `DEFAULT`	Validação de valores
Unicidade	`PRIMARY KEY`, `UNIQUE`	Evitar duplicação
Referencial	`FOREIGN KEY`	Manter relacionamentos
Desempenho	`INDEX`, `FULLTEXT`, `SPATIAL`	Otimização de consultas
Automação	`AUTO_INCREMENT`, `GENERATED`	Geração automática de valores
Geoespacial	`SPATIAL`, `POINT`	Trabalhar com dados geográficos

4. Exercício prático

- **Cenário**
  - Você é responsável por criar um banco de dados para gerenciar uma pequena loja online. A loja vende produtos e mantém registros de clientes e seus pedidos.

I. Crie o banco de dados

II. Crie as seguintes tabelas

a. Clientes;

Coluna	Tipo	Restrições
ClienteID	INT	PRIMARY KEY, AUTO_INCREMENT
Nome	VARCHAR(100)	NOT NULL
Email	VARCHAR(100)	UNIQUE
Telefone	VARCHAR(15)	NULL

b. Produtos;

Coluna	Tipo	Restrições
ProdutoID	INT	PRIMARY KEY, AUTO_INCREMENT
NomeProduto	VARCHAR(100)	NOT NULL
Preco	DECIMAL(10,2)	NOT NULL
Estoque	INT	NOT NULL

c. Pedidos;

Coluna	Tipo	Restrições
PedidoID	INT	PRIMARY KEY, AUTO_INCREMENT
ClienteID	INT	FOREIGN KEY REFERENCES Clientes(ClienteID)
DataPedido	DATE	NOT NULL

d. ItensPedido;

Coluna	Tipo	Restrições
ItemID	INT	PRIMARY KEY, AUTO_INCREMENT
PedidoID	INT	FOREIGN KEY REFERENCES Pedidos(PedidoID)
ProdutoID	INT	FOREIGN KEY REFERENCES Produtos(ProdutoID)
Quantidade	INT	NOT NULL

III. Insira dados nas tabelas;

Clientes

sql Copiar código

```sql
INSERT INTO Clientes (Nome, Email, Telefone) VALUES
('João Silva', 'joao@gmail.com', '123456789'),
('Maria Oliveira', 'maria@gmail.com', '987654321'),
('Carlos Souza', 'carlos@gmail.com', '456123789');
```

Produtos

sql Copiar código

```sql
INSERT INTO Produtos (NomeProduto, Preco, Estoque) VALUES
('Notebook', 3500.00, 10),
('Mouse', 50.00, 100),
('Teclado', 150.00, 50),
('Monitor', 800.00, 30);
```

Pedidos

sql Copiar código

```sql
INSERT INTO Pedidos (ClienteID, DataPedido) VALUES
(1, '2024-12-01'),
(2, '2024-12-02');
```

ItensPedido

sql Copiar código

```sql
INSERT INTO ItensPedido (PedidoID, ProdutoID, Quantidade) VALUES
(1, 1, 1), -- João comprou 1 Notebook
(1, 2, 2), -- João comprou 2 Mouses
(2, 3, 1); -- Maria comprou 1 Teclado
```

IV. Faça Consultas práticas:

a. Liste todos os clientes cadastrados;

b. Liste todos os produtos com estoque acima de 20 unidade;

c. Encontre os pedidos realizados no dia 2024-12-01;

d. Liste todos os produtos comprados por João Silva. Inclua o nome do produto, quantidade e preço;

e. Atualize o estoque do produto “Mouse” subtraindo a quantidade comprada (2 unidades);

f. Exclua o cliente com ID 3 (*Carlos Souza*);