

# Banco de Dados I

Ronierison Maciel

Agosto 2024



# Quem sou eu?



**Nome:** Ronierison Maciel / Roni

**Formação:** Mestre em Ciência da Computação

**Ocupação:** Pesquisador, Professor e Desenvolvedor de Software

**Hobbies:** Jogar cartas, ficar com a família no final de semana conversando sobre diversos temas

**Interesses:** Carros, aprimoramento na área educacional, desenvolvimento de software, data science e machine learning

**Email:** ronierison.maciel@pe.senac.br

**GitHub:** <https://github.com/ronierisonmaciel>

# Conteúdo

- 1 Introdução
- 2 Modelo de dados relacional e SQL básico
- 3 Modelagem de dados
- 4 Normalização
- 5 Estrutura de Arquivo
- 6 Indexação
- 7 Transação
- 8 Triggers
- 9 Concorrência

# Objetivos da semana

## Tópicos:

- Visão geral sobre BD e SGBD, instalação do MySQL
- Modelos de Dados, esquemas e arquiteturas
- Linguagens e interfaces de SGBDs, criação de tabelas

## Objetivo:

- Introduzir os conceitos fundamentais de Banco de Dados e realizar a configuração inicial do MySQL



# O que são Bancos de Dados (BD)?

Um Banco de Dados é uma coleção organizada de dados, tipicamente armazenados e acessíveis eletronicamente.

- **Exemplo:** Catálogo de produtos de uma loja, lista de alunos de uma escola.

CLIENTE	
CÓDIGO	NOME
1234	CARLOS
5678	JOÃO
9101	PEDRO
1213	MARIA

VENDEDOR	
CÓDIGO	NOME
11	CARMEM
12	DJANIRA
13	ZÉCA
14	MARIO

PRODUTO	
CÓDIGO	DESCRIÇÃO
123	LAPIS
456	CANETA
789	PAPEL A4
101	TESOURA
123	BORRACHA
141	LIVRO

CONTÉM		
PEDIDO	PRODUTO	QUANTIDADE
100/05	123	10
100/05	789	20
101/05	456	30
102/05	456	40
103/05	101	50
103/05	121	60
103/05	141	70
104/05	456	80

PEDIDO			
NÚMERO	DATA	VENDEDOR	CLIENTE
100/05	01/01/05	12	5678
101/05	01/02/05	11	9101
102/05	01/03/05	13	1213
103/05	01/04/05	14	1234
104/05	01/05/05	12	1213

Figure: Tabelas

# O que é um Sistema de Gerenciamento de Banco de Dados (SGBD)?

## Conteúdo:

- **Definição:** Um SGBD é um software que permite a criação, gestão, manipulação e controle de acesso a bancos de dados.
- **Funções:** Controle de concorrência, recuperação de falhas, segurança e integridade de dados.

**Exemplos:** MySQL, PostgreSQL, Oracle, SQL Server.

# Importância dos Bancos de Dados

## Por que Bancos de Dados são importantes?

- Centralização e organização dos dados.
- Suporte à tomada de decisão.
- Eficiência e escalabilidade em operações de negócio.

**Casos de Uso:** Comércio eletrônico, sistemas de gerenciamento de clientes (CRM), sistemas de controle financeiro.

## Visão geral dos SGBDs mais usados

- **MySQL**: Open source, amplamente usado em aplicações web.
- **PostgreSQL**: Avançado, com suporte a tipos de dados complexos.
- **Oracle**: Focado em grandes empresas, oferece alta performance e segurança.
- **SQL Server**: Solução da Microsoft, integrada com outras ferramentas da empresa.

**Gráfico Comparativo:** Popularidade dos SGBDs (baseado em pesquisas recentes).

# Instalação do MySQL

## Como instalar o MySQL

Passos:

- Baixar o MySQL Community Server do site oficial.
- Executar o instalador e seguir as instruções.
- Configurar a senha do root e as opções de segurança.
- Verificar a instalação via terminal.

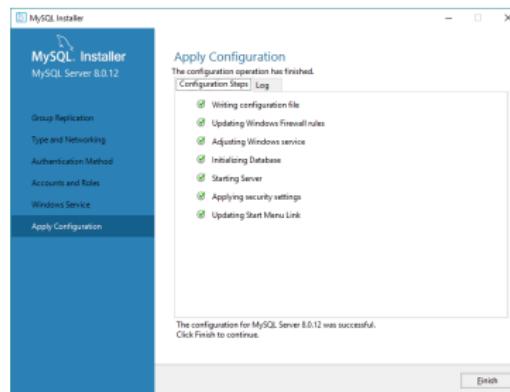


Figure: Instalação do MySQL

## O que é MySQL Workbench?

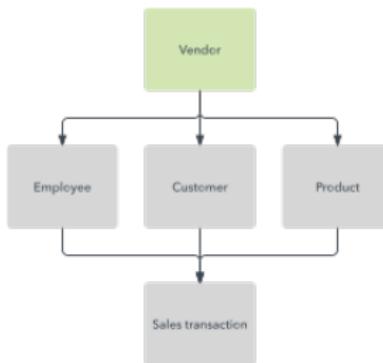
Ferramenta GUI para modelagem de dados, desenvolvimento SQL e administração de servidores.

- Conectar ao servidor MySQL.
- Criar um banco de dados.
- Explorar as funcionalidades básicas.

# Modelos de dados

## Introdução aos Modelos de Dados

- **Modelos hierárquicos:** Dados organizados em uma estrutura de árvore.
- **Modelos em rede:** Dados organizados em gráficos, permitindo múltiplas relações.
- **Modelos relacionais:** Dados organizados em tabelas, a base do MySQL.



Student ID	First name	Last name
52-743965	Charles	Peters
48-209689	Anthony	Sondrup
14-204968	Rebecca	Phillips

ProviderID	Provider name
156-983	UnitedHealth
146-823	Blue Shield
447-784	Carefirst Inc.

Student ID	ProviderID	Type of plan	Start date
52-743965	156-983	HSA	04/01/2016
48-209689	146-823	HMO	12/01/2015
14-204968	447-784	HSA	03/14/2016

## Estrutura dos Bancos de Dados

- Esquema: A estrutura lógica de um banco de dados, definindo como os dados são organizados e inter-relacionados.

## Arquiteturas:

- **Monolítica:** Todos os dados e serviços estão centralizados em um único sistema.
- **Cliente-Servidor:** Dados armazenados em servidores, acessados por clientes.
- **Distribuída:** Dados espalhados por múltiplos sistemas interconectados.

# Linguagens de SGBDs

## Linguagens e interfaces de SGBDs

### Linguagens:

- **DDL** (Data Definition Language): Criação e modificação de estruturas de dados.
- **DML** (Data Manipulation Language): Inserção, atualização e exclusão de dados.

### Interfaces:

- **CLI** (Command-Line Interface): Interação via comandos de texto.
- **GUI** (Graphical User Interface): Interação via interface gráfica, como o MySQL Workbench.

```
1 ALTER TABLE clientes ADD telefone VARCHAR(15);  
2 /* Comandos DDL */
```

```
1 SELECT nome, email FROM clientes WHERE data_cadastro > '  
2 2023-01-01';  
3 /* Comandos DML */
```

# Criação de tabelas em MySQL

## Definindo tabelas e tipos de dados

### Tipos de Dados:

- Numéricos (INT, FLOAT).
- Texto (VARCHAR, TEXT).
- Data/Hora (DATE, TIMESTAMP).

### Exemplo de criação de tabela:

```
1 CREATE TABLE alunos (
2     id INT AUTO_INCREMENT PRIMARY KEY,
3     nome VARCHAR(100),
4     data_nascimento DATE
5 );
```



# Prática da Semana 1

## Descrição:

- ① Instalar o MySQL e MySQL Workbench em seus computadores.
- ② Criar um banco de dados simples e tabelas com diferentes tipos de dados.
- ③ Inserir alguns registros e praticar comandos básicos de consulta.

**Entrega:** Enviar um relatório com capturas de tela e código SQL até a próxima aula.

# Próximos passos

## O que vamos fazer na próxima semana?

- Revisão dos conceitos básicos de SQL.
- Introdução ao modelo de dados relacional.
- Primeiras operações com SQL (SELECT, INSERT, UPDATE, DELETE).

# Objetivos da semana

## Tópicos:

- Conceitos do modelo de dados relacional e tabelas relacionais
- Operações básicas em SQL (SELECT, INSERT, UPDATE, DELETE)
- Transações e controle de transações em MySQL

## Objetivo:

- Entender o modelo de dados relacional e realizar operações básicas utilizando SQL no MySQL.

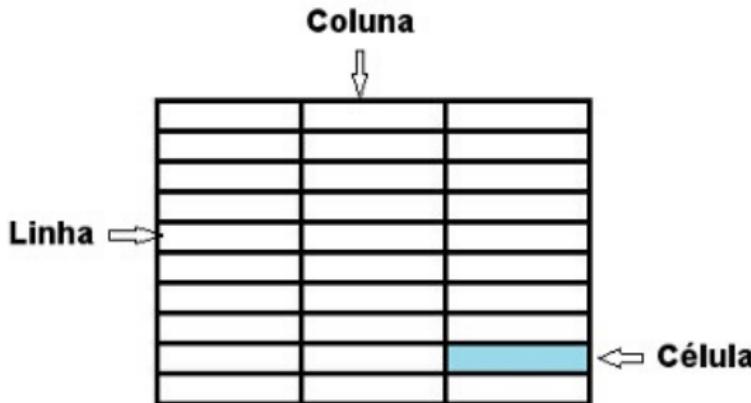
# O que é o modelo de dados relacional?

## Conceito:

- Modelo de dados que organiza informações em tabelas (também conhecidas como relações).

## Elementos-chave:

- **Tabelas**: Conjuntos de dados organizados em linhas e colunas.
- **Linhas (Tuplas)**: Cada linha representa um registro único.
- **Colunas (Atributos)**: Cada coluna representa um campo de dado dentro de um registro.



# Componentes de uma tabela relacional

## Estrutura de uma tabela relacional

- **Chave primária:** Coluna ou conjunto de colunas que identifica de forma única cada registro em uma tabela.
- **Chave estrangeira:** Coluna que cria uma ligação entre duas tabelas diferentes, referenciando a chave primária de outra tabela.
- **Índices:** Estruturas que melhoraram a velocidade de operações de consulta nas tabelas.

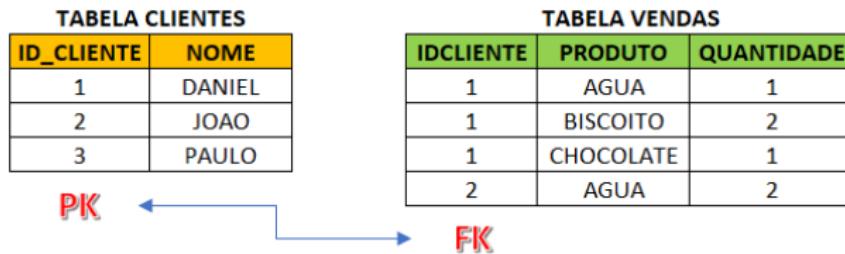


Figure: Foreign Key (FK) and Primary Key (PK)

# Definição de tabelas e relações no MySQL

## Criando tabelas e definindo relações

### Exemplo de SQL:

```
1 CREATE TABLE departamentos (
2     id INT AUTO_INCREMENT PRIMARY KEY ,
3     nome VARCHAR(100)
4 );
5
6 CREATE TABLE empregados (
7     id INT AUTO_INCREMENT PRIMARY KEY ,
8     nome VARCHAR(100) ,
9     departamento_id INT ,
10    FOREIGN KEY (departamento_id) REFERENCES departamentos(
11        id)
12 );
```

**Prática:** Criar tabelas e estabelecer relações no MySQL Workbench.



## Introdução às operações básicas em SQL

- INSERT: Inserção de novos registros.
- SELECT: Consulta de dados.
- UPDATE: Atualização de registros existentes.
- DELETE: Exclusão de registros.

# Operação INSERT

## Inserindo dados em tabelas

```
1 INSERT INTO <nome_da_tabela> (<nomes_dos_atributos>)
2 VALUES (<valores>);
```

Inserção de múltiplos registros de uma vez.

# Operação SELECT

## Consultas básicas com SELECT

```
1 SELECT <nome_da_coluna_1>, <nome_da_coluna_2>
2 FROM <nome_da_tabela>
3 WHERE <condicao>;
```

### Cláusulas:

- WHERE: Filtros para as consultas.
- ORDER BY: Ordenação dos resultados.
- GROUP BY: Agrupamento de registros.

# Operação UPDATE

## Atualizando registros com UPDATE

```
1 UPDATE <nome_da_tabela>
2 SET coluna1 = <valor_1>, <coluna_2> = <valor_2>
3 WHERE <condicao>;
```

Atualização condicional com WHERE.



# Operação DELETE

## Excluindo registros com DELETE

```
1 DELETE FROM <nome_da_tabela>
2 WHERE <condicao>;
```

Cuidado ao usar DELETE sem cláusula WHERE.

## O que são transações em bancos de dados?

- Uma **transação** é um conjunto de operações (inserir, atualizar, excluir dados) que são executadas como uma única unidade de trabalho. Se todas as operações forem bem-sucedidas, as mudanças são confirmadas. Se alguma falhar, todas as mudanças são desfeitas (rollback), garantindo a integridade do banco de dados.

## Propriedades ACID:

- **Atomicidade:** A transação é **tudo ou nada**; ou todas as operações são concluídas, ou nenhuma é.
- **Consistência:** A transação mantém o banco de dados em um estado consistente, respeitando todas as regras e restrições definidas.
- **Isolamento:** Transações são executadas de forma independente, sem interferir umas nas outras, como se fossem únicas no sistema.
- **Durabilidade:** Após a confirmação, as mudanças realizadas pela transação são permanentes, mesmo que ocorram falhas no sistema posteriormente.

# Controle de transações no MySQL

## Trabalhando com transações no MySQL

### Comandos importantes:

- START TRANSACTION: Inicia uma nova transação.
- COMMIT: Confirma as mudanças realizadas pela transação.
- ROLLBACK: Reverte as mudanças realizadas pela transação.

### Exemplo:

```
1 START TRANSACTION;
2 UPDATE empregados SET salario = salario * 1.1 WHERE
  departamento_id = 1;
3 COMMIT;
```

Essa query usa uma transação para aumentar em 10% os salários dos empregados do departamento departamento\_id = 1. A transação começa com START TRANSACTION, a atualização é feita pelo UPDATE, e o comando COMMIT torna as mudanças permanentes.



# Prática da semana 2

## Descrição:

- ① Criar tabelas relacionadas no MySQL.
- ② Inserir, atualizar e excluir registros utilizando SQL básico.
- ③ Realizar consultas complexas com filtros e agrupamentos.
- ④ Implementar transações envolvendo múltiplas operações.

# Objetivos da semana

## Tópicos:

- Ampliar o conhecimento em SQL além dos comandos básicos já aprendidos: INSERT, SELECT, UPDATE e DELETE.
- Aprender a filtrar e ordenar dados utilizando as cláusulas WHERE e ORDER BY.
- Introduzir funções agregadas como COUNT, AVG, MIN e MAX para realizar cálculos em conjuntos de dados.
- Compreender o agrupamento de dados com a cláusula GROUP BY.

# Filtrando dados com a cláusula WHERE

A cláusula WHERE é utilizada para filtrar registros de uma tabela que atendem a determinadas condições. Ela permite que você selecione apenas os dados que são relevantes para a sua consulta.

- Filtrando por um valor específico:

```
1 SELECT * FROM empregados WHERE departamento_id = 2;
```

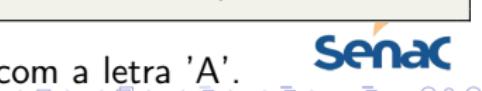
- Este comando seleciona todos os empregados que pertencem ao departamento com id igual a 2.
- Utilizando operadores de comparação:

```
1 SELECT * FROM empregados WHERE salario > 3000;
```

- Seleciona empregados com salário maior que 3000.
- Filtrando por texto com LIKE:

```
1 SELECT * FROM empregados WHERE nome LIKE 'A%';
```

- Seleciona empregados cujo nome começa com a letra 'A'.



# Exercício

Filtre os empregados que foram admitidos após a data 2022-01-01.

# Ordenando resultados com ORDER BY

A cláusula ORDER BY é usada para ordenar os resultados de uma consulta em ordem crescente ou decrescente, de acordo com uma ou mais colunas.

- Ordenação crescente (padrão):

```
1 SELECT * FROM empregados ORDER BY nome;
```

- Ordena os empregados em ordem alfabética pelo nome.

- Ordenação decrescente:

```
1 SELECT * FROM empregados ORDER BY salario DESC;
```

- Ordena os empregados do maior para o menor salário.

- Ordenação por múltiplas colunas:

```
1 SELECT * FROM empregados ORDER BY departamento_id, nome;
```

- Ordena primeiro por departamento e, dentro de cada departamento, por nome.

# Exercício

Liste os departamentos em ordem decrescente de id e, em seguida, liste os empregados ordenados por data de admissão mais recente.

# Funções agregadas e agrupamento de dados com GROUP BY

Funções agregadas realizam cálculos em um conjunto de valores e retornam um único valor. As mais comuns são:

- **COUNT**: Conta o número de registros.
- **AVG**: Calcula a média de um conjunto de valores.
- **MIN** e **MAX**: Encontram o menor e o maior valor, respectivamente.
- **SUM**: Calcula a soma de um conjunto de valores.

# Agrupamento com GROUP BY

A cláusula GROUP BY é usada em conjunto com funções agregadas para agrupar os resultados por uma ou mais colunas.

- Contando o número de empregados em cada departamento:

```
1 SELECT departamento_id, COUNT(*) AS total_employees  
2 FROM empregados  
3 GROUP BY departamento_id;
```

- Calculando o salário médio por departamento:

```
1 SELECT departamento_id, AVG(salario) AS salario_medio  
2 FROM empregados  
3 GROUP BY departamento_id;
```

- Encontrando o maior salário em cada departamento:

```
1 SELECT departamento_id, MAX(salario) AS maior_salario  
2 FROM empregados  
3 GROUP BY departamento_id;
```

- ① Liste o número total de empregados na empresa.
- ② Calcule a média salarial geral e identifique o menor e o maior salário entre todos os empregados.
- ③ Liste cada departamento com o total de salários pagos.

## O que iremos aprender na próxima semana?

- Modelagem de Dados com ER.
- Introdução ao projeto de banco de dados relacional.
- Aplicação de UML na modelagem de banco de dados.

# Objetivos da semana

## Tópicos:

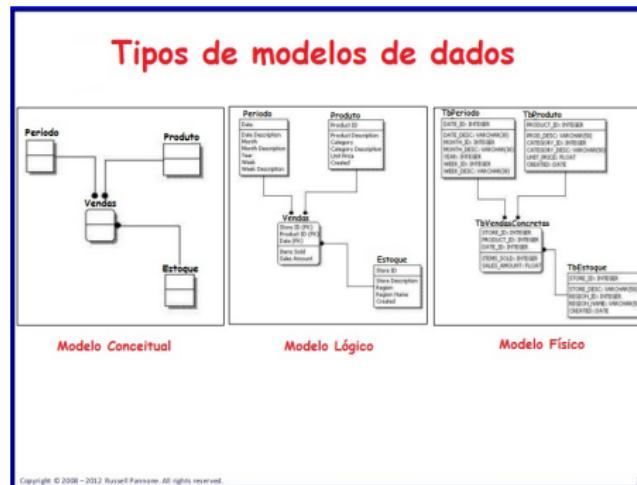
- Modelo Entidade-Relacionamento (ER)
- Projeto de Banco de Dados Relacional e UML
- **Objetivos:**

Aprender a modelar dados utilizando ER e UML, aplicando técnicas avançadas de modelagem.

# que é modelagem de dados?

## Conceito de modelagem de dados

- **Definição:** A modelagem de dados é o processo de criar um modelo visual das informações que serão armazenadas em um banco de dados.
- **Objetivo:** Estruturar os dados de forma que possam ser armazenados, acessados e gerenciados de maneira eficiente.



# Notações

Existem maneiras diferentes de apresentar um modelo Entidade Relacionamento.

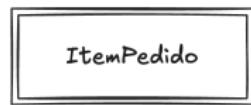
- São notações que variam na utilização de símbolos e convenções.
  - Notação de Peter Chen;
  - Notação de James Martin;
  - Notação de Peter Chen adaptada por Carlos A. Heuser;

# Notação de Peter Chen - Entidades

A notação de Peter Chen é muito difundida e utilizada e é caracterizada por sua simplicidade, representam objetos ou conceitos do mundo real que possuem existência própria no contexto do sistema.



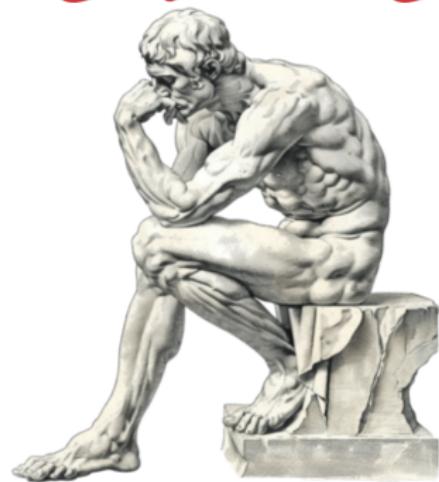
**Entidades:** Representam objetos ou conceitos do mundo real que possuem existência própria no contexto do sistema.



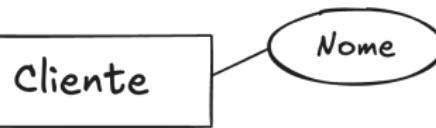
**Entidade fraca:** Não possui um identificador único próprio e dependem de uma entidade forte para sua identificação. Item de Pedido (depende do Pedido), Dependente (depende do Funcionário).

Pergunta!

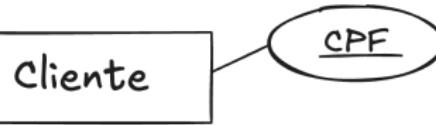
# O que é uma entidade?



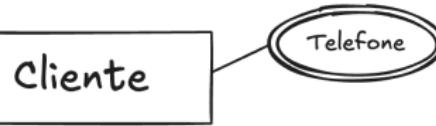
# Notação de Peter Chen - Atributos



**Simples:** Representam as características ou propriedades das entidades.

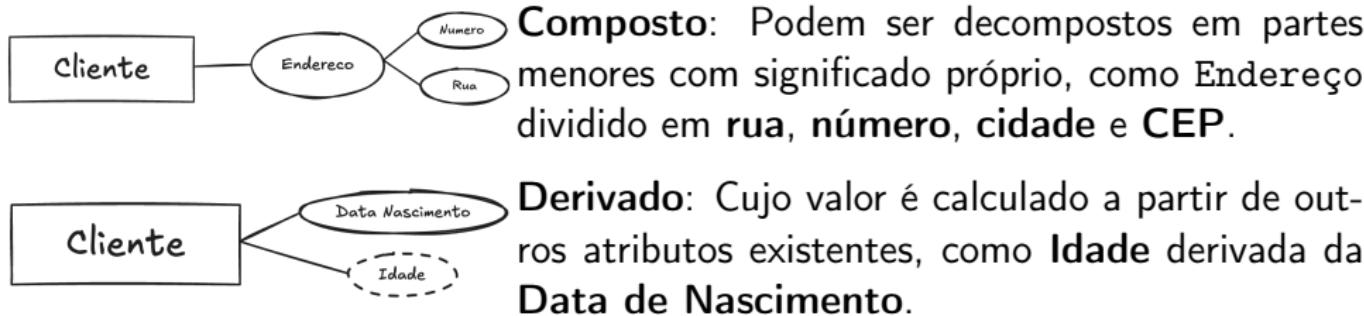


**Chave primária:** Identificam unicamente cada entidade dentro de um conjunto, funcionando como chave primária para distinguir instâncias individuais.

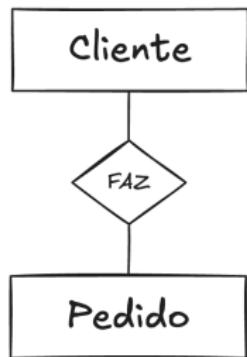


**Multivalorados:** Podem ter múltiplos valores para uma única entidade, como **telefones** em um cliente que possui vários números.

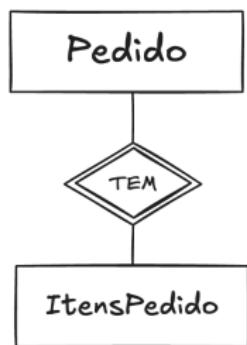
# Notação de Peter Chen - Atributos



# Notação de Peter Chen



**Relacionamento:** Associação entre duas ou mais entidades que descreve como elas interagem ou se relacionam no modelo de dados, representada por um losango no diagrama ER.

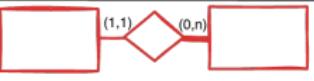


**Relacionamento identificador:** É um tipo de relacionamento onde uma entidade fraca depende de uma entidade forte para sua identificação; representado por um losango com bordas duplas, indica que a chave primária da entidade fraca inclui a chave primária da entidade forte.

- O professor Dr. Carlos Alberto Heuser, da UFRGS, adaptou a notação original de Peter Chen com o objetivo de simplificar os modelos Entidade-Relacionamento. Heuser introduziu uma notação que facilita a distinção entre atributos simples e derivados (aqueles calculados a partir de outros atributos)
  - A ferramenta brModelo foi criada pelo Heuser e utiliza essa notação.

Colocando a mão na modelagem!

- Crie um diagrama ER para um sistema de gerenciamento de uma locadora de filmes. Considere as seguintes entidades: Filme, Cliente, e Locação. Defina os atributos principais para cada entidade e as relações entre elas.

Conceito	Símbolo
Entidade	
Relacionamento	
Atributo	
Atributo identificador	
Relacionamento identificador (Entidade fraca)	
Generalização/especialização	
Entidade associativa	

# Cardinalidade



# Relações entre tabelas no modelo relacional

Cardinalidade: Quantos elementos se relacionam? Um para um, um para muitos ou muitos para muitos.

- **Relação 1:1 (um para um):**

- Significa que um registro em uma tabela A está associado a no máximo um registro em uma tabela B, e vice-versa.



- **A notação:** O "1:1" significa que para cada instância de uma entidade (como uma pessoa), há uma e somente uma instância associada na outra entidade (passaporte).

- Relação 1:N (um para muitos):

- Um registro em uma tabela A pode estar relacionado a vários registros em uma tabela B, mas cada registro em B está relacionado a apenas um registro em A.



- **A notação:** O "1" (onde N pode ser qualquer número) significa que uma instância da entidade A pode estar relacionada a muitas (ou seja, múltiplas) instâncias da entidade B.

# Relações entre tabelas no modelo relacional

- **Relação N:N (muitos para muitos):**

- Vários registros em uma tabela A podem estar relacionados a vários registros em uma tabela B. Esse tipo de relação normalmente é implementado usando uma tabela intermediária (tabela de junção) que referencia ambas as tabelas.



- **A notação:** O "N" (onde N e N podem ser quaisquer números) indica que muitas instâncias de uma entidade A podem estar relacionadas a muitas instâncias de uma entidade B.

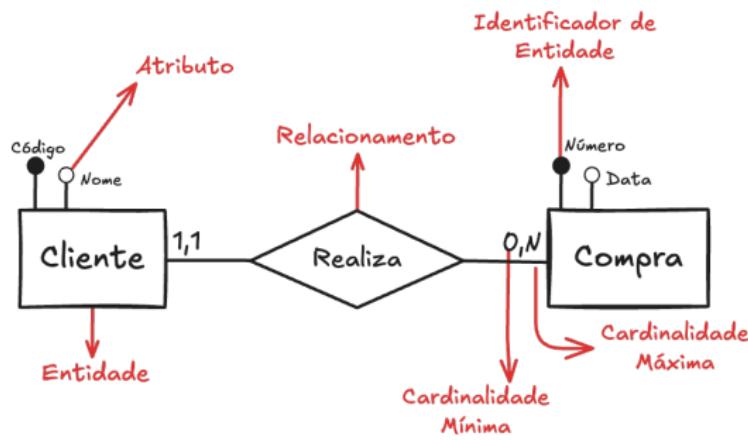
# Entidade-Relacionamento (ER)

## Conceito:

- O MER é uma representação gráfica com a estrutura lógica de um BD.

## Componentes:

- Entidades:** Objetos ou conceitos sobre os quais os dados são armazenados.
- Atributos:** Características das entidades.
- Relacionamentos:** Associações entre entidade.



# Recapitulando o diagrama ER

## Entidades:

- Representadas por retângulos. Exemplo: Clientes, Compra.

## Atributos:

- Representados por elipses. Exemplo: Nome, Código.

## Relacionamentos:

- Representados por losangos. Exemplo: Realiza, Contém, Faz.

## Chave primária:

- Um atributo ou conjunto de atributos que identifica de forma única uma entidade.

# Exemplo prático de modelagem ER

## Modelagem de um sistema de vendas

### ① Entidades identificadas:

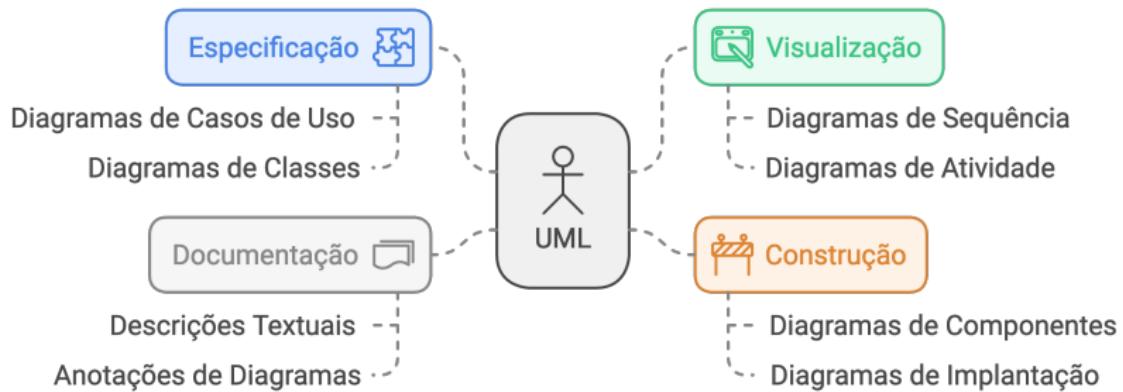
- Cliente: Atributos: ClientID, Nome, Endereço.
- Produto: Atributos: ProdutoID, NomeProduto, Preço.
- Pedido: Atributos: PedidoID, DataPedido, ClientID.

### ② Relacionamentos:

- Cliente faz Pedido.
- Pedido contém Produto.

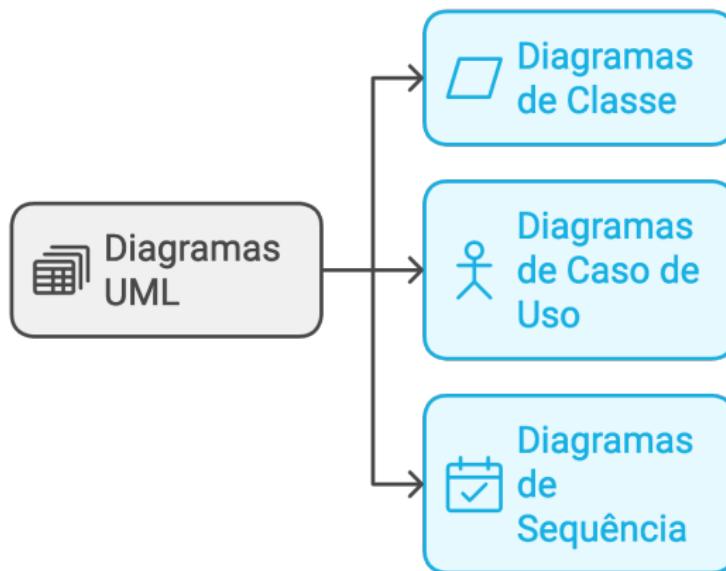
# Projeto de Banco de Dados Relacional e UML

UML é uma linguagem de modelagem padrão usada para especificar, visualizar, construir e documentar os componentes de sistemas de software.



# Projeto de Banco de Dados Relacional e UML

## Diagramas diversificados



## Orientação a Objetos:

- Suporta princípios da orientação a objetos, facilitando a representação de classes, objetos, herança, polimorfismo, etc.

## Notação Gráfica:

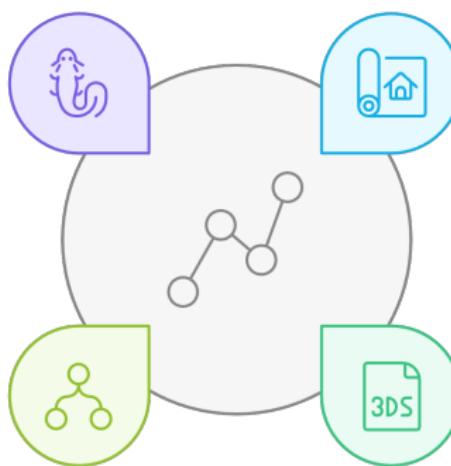
- Utiliza símbolos gráficos padronizados para representar elementos como classes, atributos, métodos, relacionamentos, etc.

## Orientação a Objetos:

### Princípios da Programação Orientada a Objetos

#### Polimorfismo

Capacidade de processar objetos de maneira diferente com base em seu tipo de dado



#### Classes

Blocos de construção fundamentais em POO definindo modelos de objetos

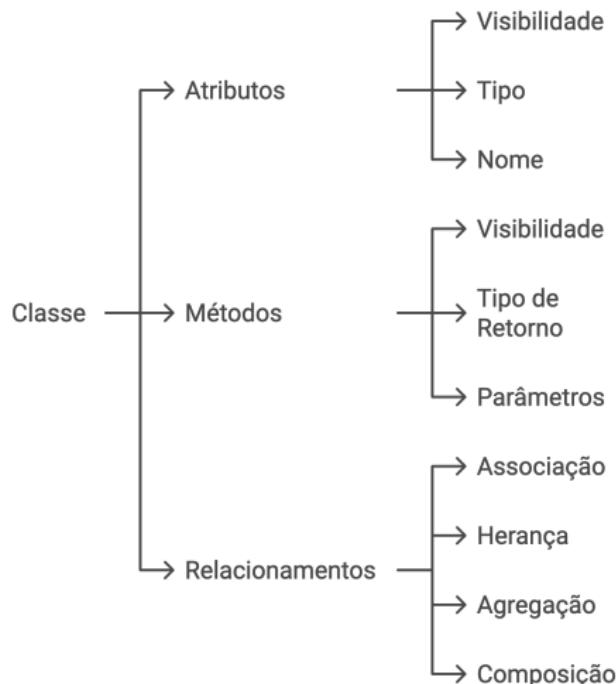
#### Herança

Mecanismo para criar novas classes a partir de classes existentes

#### Objetos

Instâncias de classes representando entidades do mundo real

# Notação típica da UML



# Notação típica da UML

- **Classes:**

- Representadas por retângulos divididos em três partes: nome da classe, atributos e métodos.

- **Atributos:**

- Listados no segundo compartimento do retângulo da classe, geralmente com visibilidade (+, -, #), tipo e nome.

- **Métodos:**

- Listados no terceiro compartimento, também com visibilidade, retorno e parâmetros.

- **Relacionamentos:**

- Linhas conectando classes, podendo indicar associações, heranças, agregações, composições, etc., com multiplicidades indicadas próximo às extremidades das linhas.

# Modelo lógico

**Objetivo:** Entender o que é o modelo lógico de banco de dados, sua importância, e como transformá-lo em uma representação formal que permita a implementação física no banco de dados.

- O que é um modelo lógico?
- É uma representação intermediária entre o modelo conceitual (ER - Entidade-Relacionamento) e o modelo físico. Ele traduz os elementos conceituais em um formato mais próximo de tabelas, chaves e colunas, mas ainda independente de um SGBD específico.
- Serve como uma ponte para garantir que as ideias do modelo conceitual sejam refletidas adequadamente na implementação final.

# Características do modelo lógico

- **Entidades como tabelas:** Cada entidade do modelo conceitual se transforma em uma tabela no modelo lógico.
- **Atributos:** Os atributos tornam-se colunas nas tabelas.
- **Relacionamentos:** Definidos por meio de chaves estrangeiras (foreign keys), que ligam as tabelas entre si.
- **Normalização:** Processo que visa reduzir redundâncias e melhorar a consistência.

# Diferença entre modelo conceitual e lógico

- **Modelo conceitual:** Abstrato, focado em representar entidades e relacionamentos de forma mais livre, muitas vezes usando diagramas ER.
- **Modelo lógico:** Mais próximo da implementação, com uma estrutura que já reflete tabelas, tipos de dados, e relações, mas sem especificar aspectos específicos de um SGBD.

# Exemplificando com um caso prático

Exemplo conceitual: Um diagrama ER com as entidades "Cliente" e "Pedido", conectadas por um relacionamento "Faz".

- Exemplo lógico: Transformando as entidades em tabelas:
  - **Tabela Cliente:** ID\_Cliente (PK), Nome, Endereço.
  - **Tabela Pedido:** ID\_Pedido (PK), Data, ID\_Cliente (FK).
  - **Relacionamento:** A FK em "Pedido" conecta cada pedido a um cliente.

## Primeira Forma Normal (1NF):

- Elimina grupos repetidos, garantindo que cada coluna contenha valores atômicos (não divisíveis).
- Suponha uma tabela "Pedido" que contenha um campo "Itens" com múltiplos produtos em um único registro. Para normalizar, criar uma nova tabela chamada "Pedido\_Item", onde cada produto do pedido seja representado em uma linha separada.

## Segunda Forma Normal (2NF):

- Garante que todos os atributos não-chave dependam unicamente da chave primária, eliminando dependências parciais.
- Em uma tabela "Pedido" com atributos "ID\_Pedido", "ID\_Cliente", e "Endereço\_Cliente", o atributo "Endereço\_Cliente" não depende da chave primária "ID\_Pedido", mas sim do "ID\_Cliente". Para normalizar, move o atributo "Endereço\_Cliente" para a tabela "Cliente"

## Terceira Forma Normal (3NF):

- Remove dependências transitivas, ou seja, nenhum atributo não-chave deve depender de outro atributo não-chave.
- Suponha uma tabela "Cliente" com os atributos "ID\_Cliente", "Nome\_Cliente", e "Cidade". Se houver também um campo "Região" que depende de "Cidade", a tabela não está em 3NF. Para normalizar, criar uma tabela separada para armazenar as informações de "Cidade" e "Região".

# Estrutura de arquivo

Como os dados são armazenados fisicamente?

## Páginas

Unidade básica de armazenamento, contém registros.



## Blocos

Agrupamento de páginas, otimiza leitura/escrita.



## Registros

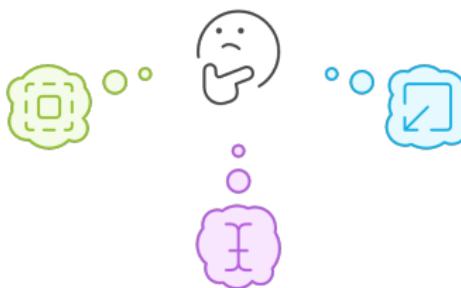
Estrutura individual de dados, armazenada em páginas.

# Estrutura de arquivo

Qual é a melhor forma de otimizar o desempenho do banco de dados MySQL?

## Aumentar o tamanho da página

Permite armazenar mais registros em uma única página, reduzindo o número de leituras e gravações no disco.



## Otimizar o tamanho do registro

Reduz o espaço ocupado por cada registro, permitindo armazenar mais registros em uma única página.

## Reducir o tamanho do bloco

Permite acessar dados mais rapidamente, mas pode aumentar o número de leituras e gravações no disco.

- Para explicar o conceito de páginas, observaremos quando os dados são inseridos em uma tabela, o MySQL agrupa essas linhas em páginas.

```
1 CREATE TABLE Alunos (
2     id INT PRIMARY KEY AUTO_INCREMENT ,
3     nome VARCHAR(50) ,
4     idade INT
5 ) ENGINE=InnoDB;
6
7 INSERT INTO Alunos (nome, idade) VALUES ('João', 20), ('Maria', 22), ('Lucas', 19), ('Ana', 21);
```

- O conceito de bloco é mais interno ao sistema operacional e ao mecanismo de armazenamento, mas pode ser compreendido como o agrupamento de páginas em uma estrutura lógica.

```
1 SHOW ENGINE INNODB STATUS;
```

# Registro

- Um registro no MySQL contém todos os dados de uma linha e ocupa espaço dentro da página.

```
1 INSERT INTO Alunos (nome, idade) VALUES ('Carlos', 23), ('Júlia', 20), ('Fernando', 25);
2
3 SELECT * FROM Alunos WHERE idade > 20;
```

- Essa consulta busca registros específicos, e cada linha recuperada ocupa espaço dentro de uma página.
- Ao adicionar mais registros, a quantidade de páginas necessárias aumenta, o que influencia a organização do armazenamento físico.

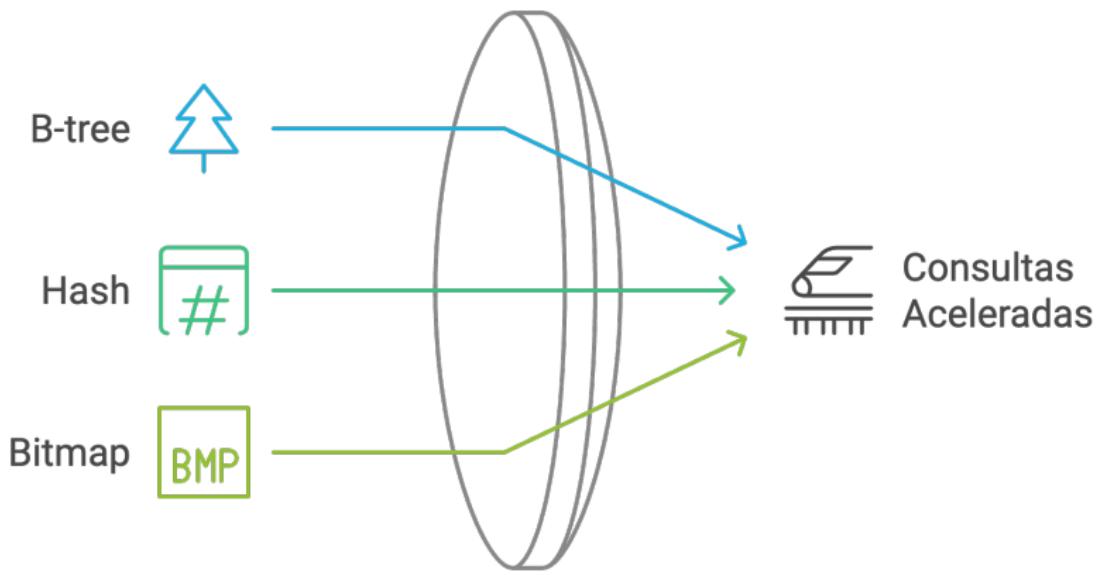
# Estrutura de armazenamento no MySQL



# Conceito de indexação

A indexação é uma técnica usada para acelerar o acesso aos dados, permitindo que o banco de dados encontre registros específicos sem precisar fazer uma leitura completa da tabela. Ao criar um índice, o banco de dados cria uma estrutura que mapeia rapidamente os registros aos seus locais de armazenamento, assim como um índice de livro permite localizar capítulos rapidamente.

## Indexação em Bancos de Dados



- Criaremos uma tabela chamada Produtos com três colunas (id, nome, preco). Vamos inserir um número significativo de registros para simular uma tabela realista.

```
1 CREATE TABLE Produtos (
2     id INT PRIMARY KEY AUTO_INCREMENT ,
3     nome VARCHAR(100) ,
4     preco DECIMAL(10, 2)
5 ) ENGINE=InnoDB;
6
7 INSERT INTO Produtos (nome, preco)
8 SELECT CONCAT('Produto', FLOOR(RAND() * 1000)), RAND() *
9     1000
10 FROM information_schema.tables
11 LIMIT 1000;
```

```
1 -- Consulta sem índice na coluna 'preco'
2 SELECT * FROM Produtos WHERE preco BETWEEN 100 AND 200;
```

- O índice B-tree (ou árvore B) é o tipo de índice padrão em muitos sistemas de banco de dados, incluindo o MySQL. Esse índice organiza os dados em uma estrutura de árvore balanceada, permitindo uma busca eficiente em grandes conjuntos de dados.

```
1 -- Adicionar um índice B-tree na coluna 'preco'  
2 CREATE INDEX idx_preco ON Produtos(preco);
```

```
1 -- Consulta com índice na coluna 'preco'  
2 SELECT * FROM Produtos WHERE preco BETWEEN 100 AND 200;
```

# Hash Index

- Criaremos uma tabela chamada Clientes com três colunas (id, nome, email). Vamos inserir um número significativo de registros para simular uma tabela realista.

```
1 CREATE TABLE Clientes (
2     id INT PRIMARY KEY AUTO_INCREMENT ,
3     nome VARCHAR(100) ,
4     email VARCHAR(100)
5 ) ENGINE=Memory;
6
7 INSERT INTO Clientes (nome, email)
8 SELECT CONCAT('Cliente', FLOOR(RAND() * 359)), CONCAT(
9     'cliente', FLOOR(RAND() * 359), '@dominio.com')
10 FROM information_schema.tables
11 LIMIT 359;
```

```
1 -- Consulta sem índice hash
2 SELECT * FROM Clientes WHERE email = 'cliente45@dominio.com'
3 ;
```

# Hash Index

- O índice Hash usa uma função de hashing para mapear valores da coluna para uma localização específica. Esse tipo de índice é muito rápido para buscas de igualdade (=), mas não é eficiente para buscas por intervalo (BETWEEN, >, <).

```
1 -- Adicionando índice hash na coluna 'email'  
2 CREATE INDEX idx_email_hash ON Clientes(email) USING HASH;
```

```
1 -- Consulta com índice hash na coluna 'email'  
2 SELECT * FROM Clientes WHERE email = 'cliente45@dominio.com'  
;
```

# Bitmap Index

- Criaremos uma tabela chamada Funcionarios com uma três (id, nome, status). Vamos inserir um número significativo de registros para simular uma tabela realista.

```
1 CREATE TABLE Funcionarios (
2     id INT PRIMARY KEY AUTO_INCREMENT ,
3     nome VARCHAR(100) ,
4     status ENUM('Ativo', 'Inativo')
5 ) ENGINE=InnoDB;
6
7 INSERT INTO Funcionarios (nome, status)
8 SELECT CONCAT('Funcionario', FLOOR(RAND() * 359)), IF(RAND()
> 0.5, 'Ativo', 'Inativo')
9 FROM information_schema.tables
10 LIMIT 359;
```

```
1 -- Consulta sem índice na coluna 'status'
2 SELECT * FROM Funcionarios WHERE status = 'Ativo';
```

# Bitmap Index

- O Bitmap Index usa um vetor de bits para representar a presença de valores distintos. Cada posição no bitmap representa um valor, com 0 ou 1 indicando se há ou não um registro correspondente. Este índice é eficiente em colunas de baixa cardinalidade (poucos valores distintos).

```
1 -- Adicionando um índice B-tree na coluna 'status'  
2 CREATE INDEX idx_status ON Funcionarios(status);
```

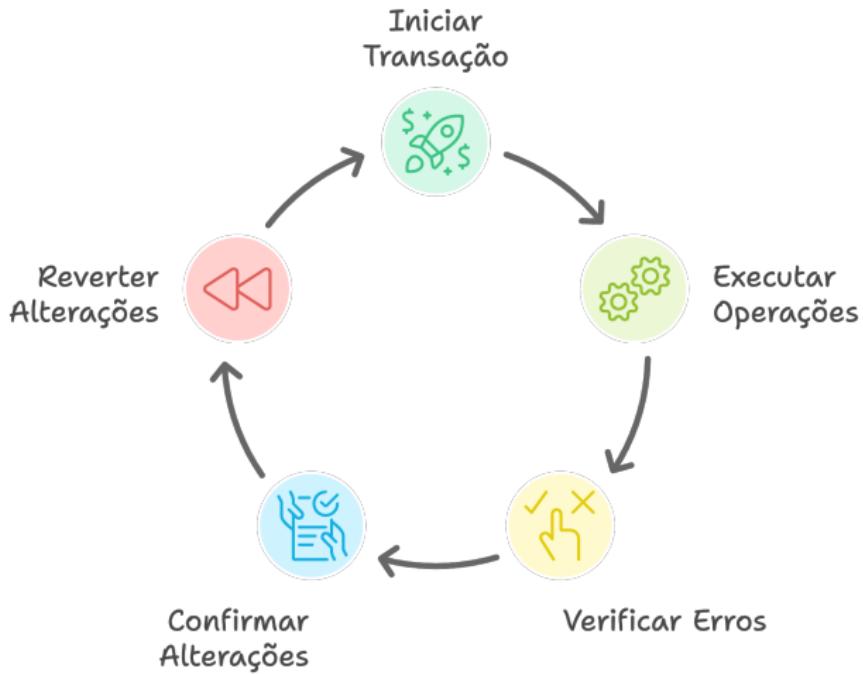
```
1 -- Consulta com índice na coluna 'status'  
2 SELECT * FROM Funcionarios WHERE status = 'Ativo';
```

# Conceito de Transação

- ① Uma transação é um conjunto de operações SQL que são executadas como uma unidade única. Ou seja, todas as operações dentro da transação devem ser completadas com sucesso para que as alterações sejam efetivadas no banco de dados. Caso ocorra um erro durante a transação, o banco de dados pode reverter (rollback) todas as alterações, garantindo que nenhuma mudança parcial permaneça.
- ② Uma transação normalmente envolve operações de inserção, atualização ou exclusão e é muito útil para garantir a integridade dos dados, especialmente em sistemas que exigem operações complexas e relacionadas.

# Ciclo de vida da transação

## Ciclo de Vida da Transação em Bancos de Dados



# Transação

- No MySQL, você pode iniciar uma transação usando comandos específicos e controlá-la com **COMMIT** e **ROLLBACK**:

```
1 START TRANSACTION; -- Inicia uma nova transação
2
3 -- Operações de transação, como INSERT, UPDATE, DELETE
4
5 COMMIT; -- Confirma as operações da transação
```

- Se algum problema ocorrer e você precisar cancelar as operações realizadas dentro da transação, pode usar **ROLLBACK**:

```
1 ROLLBACK; -- Reverte todas as operações realizadas na transação
```

## Transação de Banco de Dados



- As propriedades ACID (**Atomicidade, Consistência, Isolamento e Durabilidade**) são fundamentais para garantir que uma transação seja executada corretamente e que o banco de dados mantenha a integridade dos dados, mesmo em casos de falhas ou problemas.

# Exemplo de Atomicidade

## Situação problema:

- Imagine que você está realizando uma transferência bancária entre duas contas. A transação precisa subtrair o valor de uma conta e adicionar à outra. Se uma das operações falhar, nenhuma das duas deve ser aplicada.

```
1 START TRANSACTION;
2 UPDATE Conta SET saldo = saldo - 100 WHERE id = 1;
3 UPDATE Conta SET saldo = saldo + 100 WHERE id = 2;
4 COMMIT; -- Confirma as operações
```

- Se uma das operações falhar, você pode executar um ROLLBACK para desfazer a transação e evitar que apenas uma parte da operação seja aplicada.



# Exemplo de Consistência

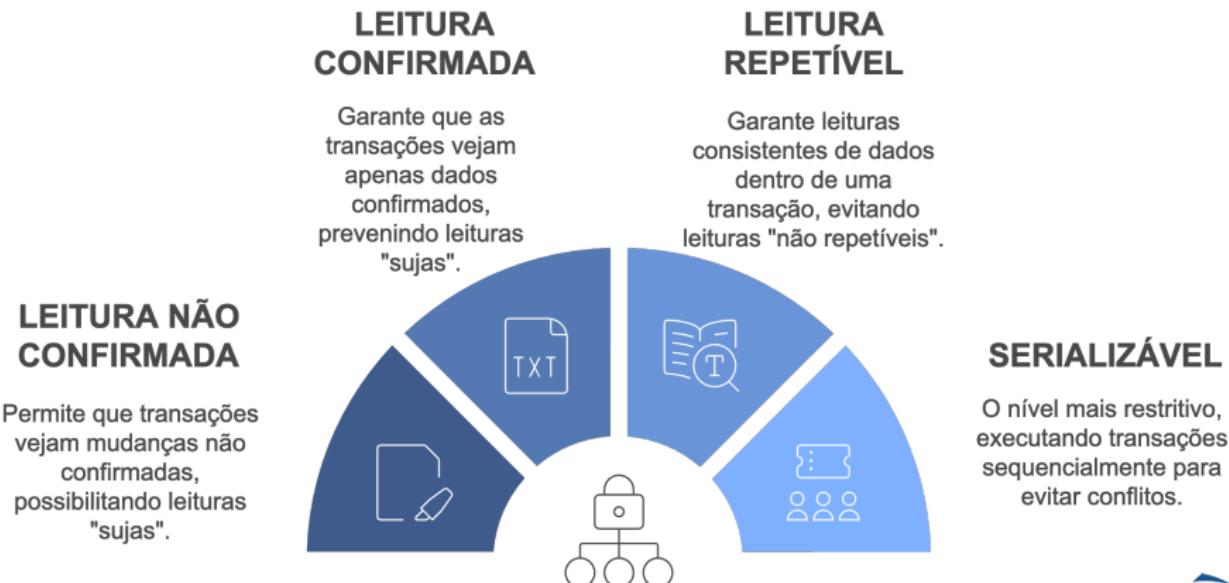
## Situação problema:

- Suponha que você tenha uma tabela de Pedidos com uma chave estrangeira para uma tabela Clientes. Se você tentar inserir um pedido com um cliente inexistente, o MySQL impedirá a operação para garantir a consistência.

```
1 START TRANSACTION;
2 INSERT INTO Pedidos (id_cliente, valor) VALUES (999, 200);
   -- Cliente ID 999 não existe
3 COMMIT;
```

- A transação falhará devido à violação da chave estrangeira, e o MySQL impedirá a inconsistência

## Níveis de Isolamento de Transação



# Exemplo de Isolamento

## Situação problema:

- Imagine duas transações simultâneas que tentam ler e modificar o mesmo dado. Dependendo do nível de isolamento, elas poderão ou não ver as alterações uma da outra antes de serem confirmadas (**COMMIT**).

```
1 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
2 START TRANSACTION;
3 -- Operações da transação
4 COMMIT;
```

- A transação falhará devido à violação da chave estrangeira, e o MySQL impedirá a inconsistência

# Durabilidade

Durabilidade assegura que, uma vez que uma transação seja confirmada, suas alterações são permanentes, mesmo em caso de falha do sistema (como queda de energia ou reinicialização do servidor). O MySQL grava as alterações no log de transações, permitindo a recuperação dos dados em caso de falhas.

- Exemplo de Durabilidade

- Após o **COMMIT**, o MySQL grava as alterações em um log permanente. Mesmo se houver uma falha de energia imediatamente após o **COMMIT**, o MySQL poderá restaurar o banco de dados para o estado mais recente e consistente.

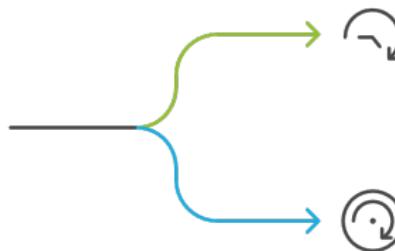
# Conceito de Triggers

- Uma trigger é um conjunto de comandos SQL que é executado automaticamente quando uma determinada ação ocorre em uma tabela. As ações que podem disparar uma trigger incluem:
  - **INSERT**: Quando um novo registro é inserido na tabela.
  - **UPDATE**: Quando um registro existente é atualizado.
  - **DELETE**: Quando um registro é excluído.
- Uma trigger pode ser configurada para ser executada:
  - Antes da ação (**BEFORE**): Executa a trigger antes da execução do comando (ideal para validar dados antes da inserção ou atualização).
  - Depois da ação (**AFTER**): Executa a trigger após a execução do comando (bom para log de auditoria e cálculos).

# Triggers



**Quando usar um trigger em um banco de dados?**



## ANTES

Validar dados antes da inserção ou atualização.

## DEPOIS

Registrar auditoria e realizar cálculos após a inserção ou atualização.

# Estrutura de uma Trigger

## Como definir uma trigger no MySQL?

### Tipo de Evento

Especifique o evento que acionará a execução (INSERT, UPDATE, DELETE).



### Momento

Especifique quando a trigger será executada (BEFORE ou AFTER).

### Tabela

Especifique a tabela à qual a trigger será aplicada.



### Instruções SQL

Defina as instruções SQL a serem executadas quando a trigger for disparada.

# Estrutura de uma Trigger

- Exemplo de sintaxe de criação de uma trigger:

```
1 CREATE TRIGGER nome_da_trigger
2 AFTER INSERT ON nome_da_tabela
3 FOR EACH ROW
4 BEGIN
5     -- Instruções SQL executadas pela trigger
6 END ;
```

# Concorrência no MySQL

O MySQL oferece ferramentas para controlar o acesso simultâneo aos dados, evitando conflitos e mantendo a integridade dos dados. Existem dois mecanismos principais de controle de concorrência:

## Qual mecanismo de bloqueio usar?

### Bloqueio Compartilhado

Permite que várias transações leiam o mesmo dado, mas impede modificações até que o bloqueio seja liberado.



### Bloqueio Exclusivo

Apenas uma transação pode modificar o dado bloqueado, evitando que outras transações leiam ou modifiquem até a liberação.

# Exemplo de bloqueio

- Vamos criar uma tabela Contas e ver como o bloqueio pode ser aplicado para gerenciar transações que alteram o saldo de uma conta.

```
1 CREATE TABLE Contas (
2     id INT PRIMARY KEY AUTO_INCREMENT ,
3     nome VARCHAR(50) ,
4     saldo DECIMAL(10, 2)
5 );
```

- Inserindo dados para teste

```
1 INSERT INTO Contas (nome, saldo) VALUES ('Alice', 500.00), (
2     'Bob', 300.00);
```

- Aplicando um bloqueio de leitura compartilhado

```
1 START TRANSACTION;
2 SELECT saldo FROM Contas WHERE nome = 'Alice' LOCK IN SHARE
   MODE;
```

# Exemplo de bloqueio

- Aplicando um bloqueio exclusivo

```
1 START TRANSACTION;  
2 SELECT saldo FROM Contas WHERE nome = 'Alice' FOR UPDATE;
```

- Para um bloqueio exclusivo, usamos **FOR UPDATE**, impedindo que outras transações leiam ou modifiquem o dado até a liberação do bloqueio.