Banco de Dados I

Introdução:

Banco de Dados (BD): é uma coleção de dados inter-relacionados envolvendo algum aspecto do mundo real. As informações contidas nele representa um estado ou instante de determinada aplicação, e cada mudança que ocorre nele, é um reflexo de um evento ou sequência que ocorre no ambiente.

Temos também o **Sistema de Banco de Dados (SBD)**, que nada mais é que um sistema de informação que lida com os BDs, registrando, manipulando e mantendo os dados. Vale mencionar que um SBD é composto pelos seguintes 4 componentes: *Hardware, Software, Usuários e Dados.*

Problemas no armazenamento em arquivos tradicionais:

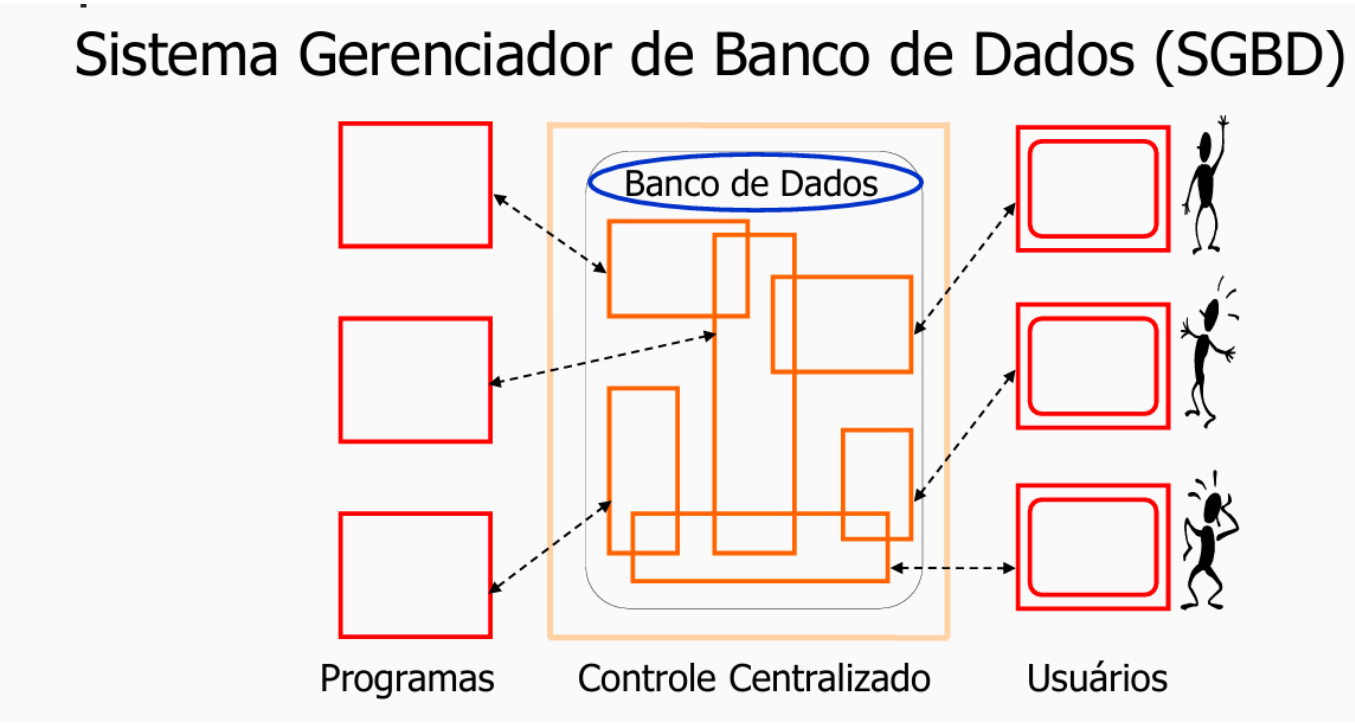
Utilizar arquivos para guardar nossos dados podem nos levar à alguns problemas indesejáveis, entre eles:

- **Redundância e inconsistência**, pois dados duplicados em múltiplos arquivos levam a desperdício de espaço e inconsistências (ex: um cliente com endereços diferentes em sistemas distintos).
- **Dificuldade de acesso**, já que consultas complexas exigem programação manual (ex: buscar todos os pedidos de um cliente em arquivos separados requer código personalizado).
- **Problemas de integrabilidade e durabilidade**, como regras de negócio (ex: "salário > 0") devem ser validadas no código da aplicação, aumentando riscos de erros. E no quesito durabilidade, caso uma máquina falhe durante a atualização de um arquivo, os dados podem ser corrompidos, além de que manter cópias idênticas dos arquivos em múltiplas máquinas exige sincronização manual, o que é propenso a erros.

De um modo geral, só valerá a pena utilizar arquivos quando os BDs e as aplicações forem simples, os requisitos de eficiência em tempo real forem altos e quando os requisitos de acesso forem monousuários.

Portanto, a vantagem dos BDs em relação ao uso de arquivos se traduz na redução da redundância, integridade garantida, independência dos dados, backup facilitado e controle centralizado via SGBD (que será explicado a seguir).

Sistema de Gerenciamento de Banco de Dados (SGBD): software projetado para armazenar, recuperar, gerenciar e proteger dados de forma eficiente. Ele atua como uma "ponte" entre os bancos de dados e os usuários ou aplicações, gerenciado dados em um único ambiente, evitando redundâncias e inconsistências.
Além disso, ele utiliza linguagem de consulta (seja ela SQL ou NoSQL), permitindo operações como inserir, atualizar, deletar e consultar dados usando comandos padronizados (ex: **SELECT * FROM clientes**). Outro positivo é a questão do compartilhamento, que permite que vários programas e usuários acessem o BD ao msm tempo.
Basicamente, o SBGD seria o "cérebro" do BD.
Lembrar a seguinte definição *SGBD + BD = Sistema de Banco de Dados (SBD)*



Modelo Relacional:

O **modelo relacional** é uma estrutura lógica para organizar dados em **tabelas (relações)**, onde:

- Cada tabela representa uma entidade (ex: `Clientes`, `Pedidos`).
- As linhas (**tuplas**) são registros individuais.
- As colunas (**atributos**) definem as propriedades dos dados (ex: `nome`, `idade`).
  Fun Fact: Foi desenvolvido por Edgar F. Codd em 1970, e é a base dos bancos de dados relacionais (SQL).

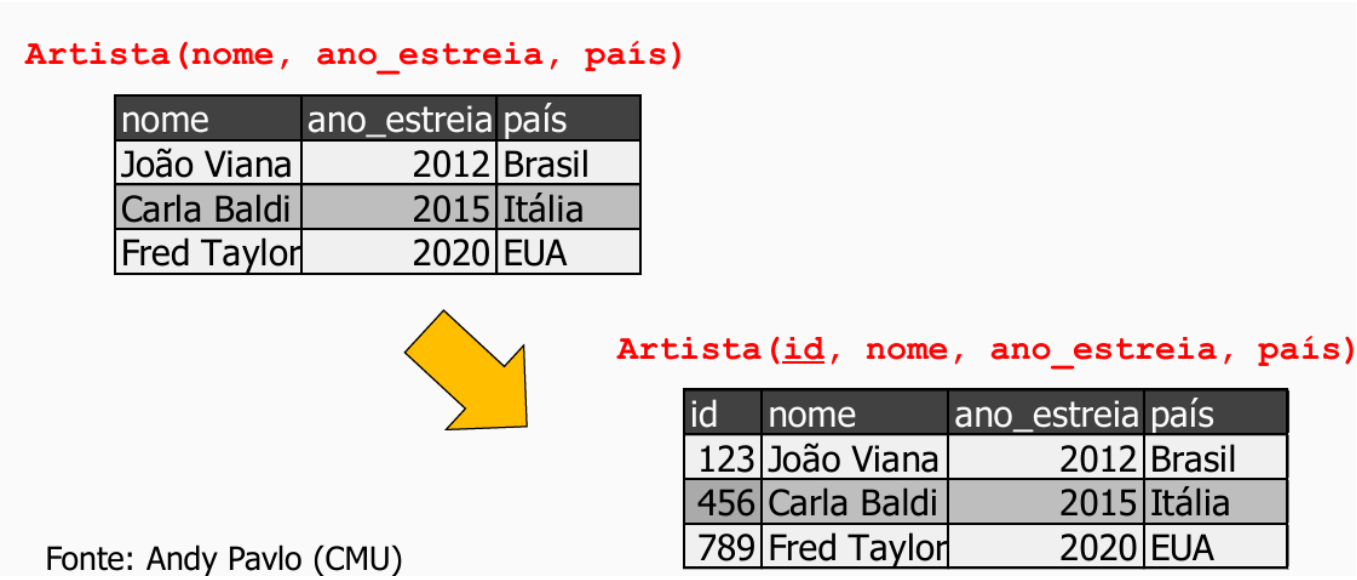
Chave Primária:

É um **atributo (ou conjunto de atributos)** que identifica **unicamente** cada registro em uma tabela.

Características:

- **Única**: Não pode haver valores repetidos.
- **Não nula**: Não aceita `NULL`.
- **Imutável**: Idealmente, não deve ser alterada.

Nesse exemplo a seguir, a chave primária seria o id



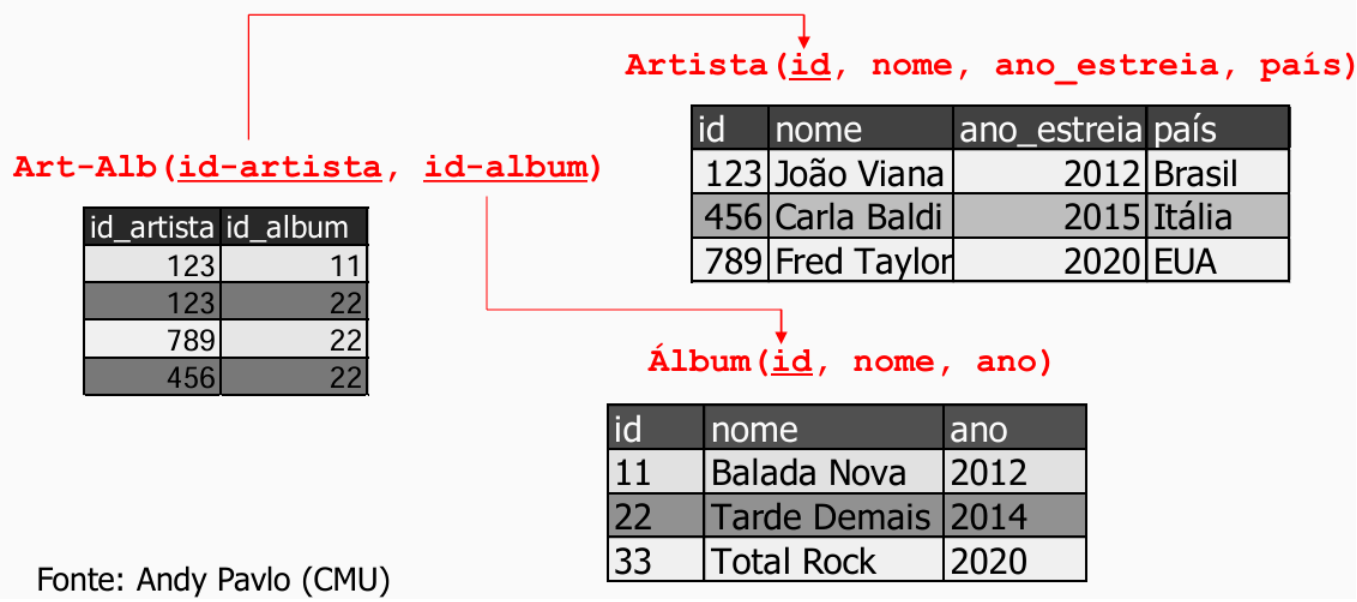
Chave Estrangeira:

É um **atributo que referencia a chave primária de outra tabela**, estabelecendo um **relacionamento** entre elas.

Características:

- **Referência**: Deve corresponder a um valor existente na chave primária da tabela referenciada.
- **Pode ser nula**: Depende das regras do modelo (exceto se marcada como `NOT NULL`).
- **Garante integridade referencial**: Evita registros "órfãos".

Nesse exemplo a seguir, as chaves estrangeiras são id-artista que referencia Artista(id) e id-album que referencia Álbum(id)



id	nome	ano_estreia	país
123	João Viana	2012	Brasil
456	Carla Baldi	2015	Itália
789	Fred Taylor	2020	EUA

id_artista	id_album
123	11
123	22
789	22
456	22

id	nome	ano
11	Balada Nova	2012
22	Tarde Demais	2014
33	Total Rock	2020

Após analisado tudo isso, podemos falar um pouco sobre os **Tipos** (ou tipos de dados), que definem o formato e as restrições dos valores que podem ser armazenados em uma coluna, garantido assim consistência, validação automática e otimização de armazenamento.
No exemplo acima, id e ano são tipos inteiros, enquanto e país são Strings.

Alguns Atores Importantes nos BDs

Administrador de Banco de Dados (DBA): O "médico" do banco de dados, responsável por **gerenciar, otimizar e garantir a saúde** dos SGBDs (como Oracle, MySQL, SQL Server).
Projetista de Banco de Dados: O "arquiteto" que desenha a estrutura do banco de dados.
Desenvolvedores de Aplicações: Os "construtores" que integram o banco de dados às aplicações (web, mobile, desktop).
Usuários Finais: Os "consumidores" dos dados, que utilizam aplicações que dependem do banco.
Um exemplo que pode ajudar a entender a relação entre esses papéis:
Projetista cria o modelo -> DBA implementa e ajusta nos SGBD -> Desenvolvedor usa o banco na aplicação -> Usuário Final interage com a aplicação.

Conceitos e Arquitetura

O que é um Esquema de Banco de Dados?

O **esquema** é a **"planta baixa"** do banco de dados. Ele descreve:

- **Estrutura**: Quais tabelas existem, seus nomes e relacionamentos.
- **Atributos**: Colunas de cada tabela e seus tipos (ex: `INT`, `VARCHAR`).
- **Restrições**: Chaves primárias, estrangeiras e regras de integridade (ex: `NOT NULL`).
  É como um **contrato** que define como os dados devem ser organizados e armazenados.

Quando o Esquema é Definido?

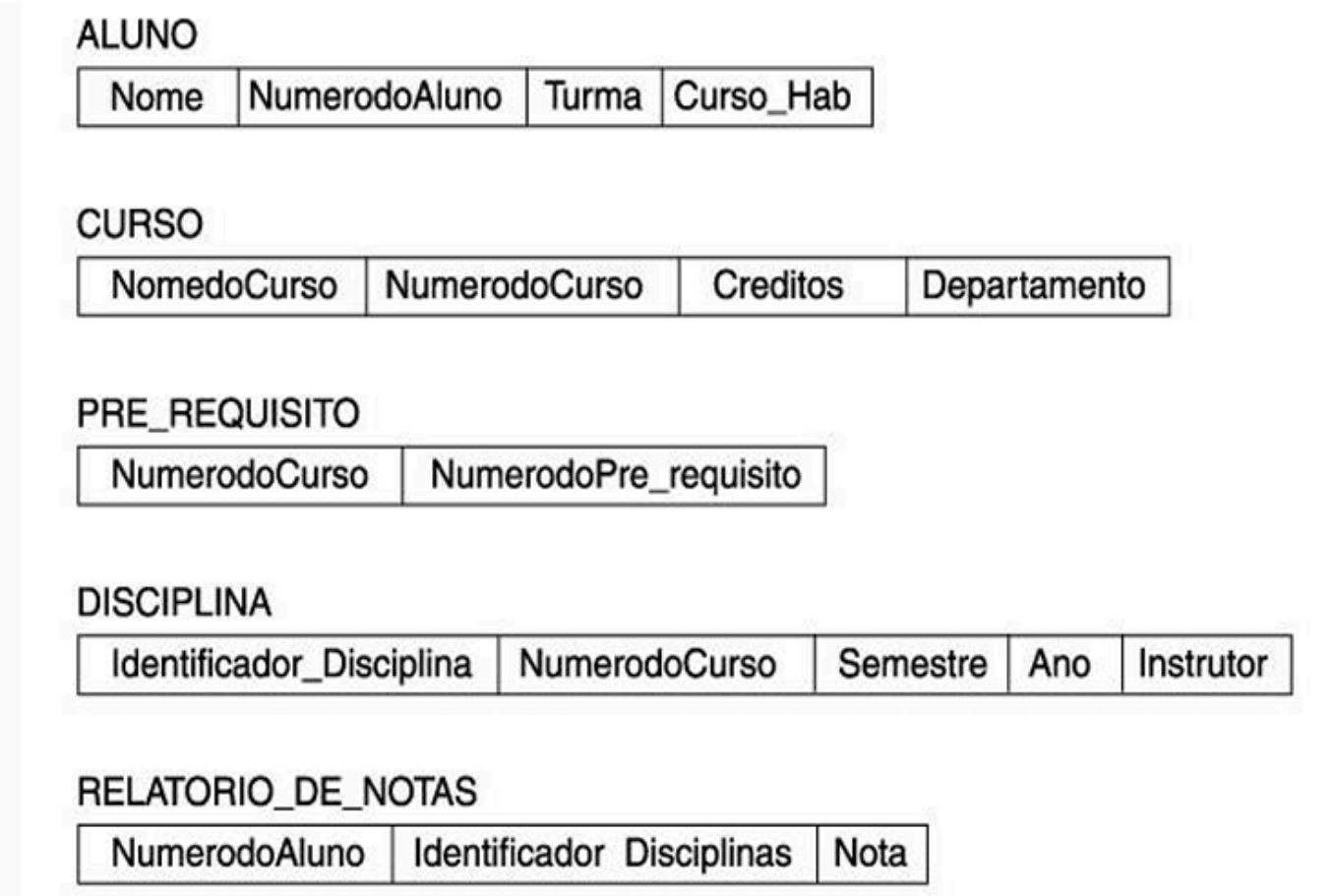
Durante a fase de **projeto do banco de dados**, geralmente seguindo estas etapas:

1. **Modelo Conceitual**: Diagrama Entidade-Relacionamento (DER, que será abordado com mais detalhes nos próximos capítulos, fique no aguardo!) para identificar entidades (ex: `Cliente`, `Pedido`).
2. **Modelo Lógico**: Transformação em tabelas com colunas e tipos (ex: `Clientes(id INT, nome VARCHAR(100))`).
3. **Modelo Físico**: Implementação no SGBD (ex: SQL para criar tabelas).

Por que o Esquema Não Deve Mudar Frequentemente?

- **Consistência**: Alterações podem quebrar aplicações que dependem da estrutura existente.
- **Custo**: Modificar tabelas com milhões de registros é complexo e lento.
- **Integridade**: Mudanças mal planejadas podem corromper dados ou relacionamentos. *Exemplo*: Adicionar uma coluna `CPF` à tabela `Clientes` exige atualizar todos os registros existentes e ajustar queries. (query nada mais é que consulta)

A seguir um exemplo de um esquema de um BD de uma universidade, observe que o diagrama apresenta a estrutura de cada tipo de registro, mas não suas instâncias (tuplas) reais:



Cada item no esquema (como ALUNO ou DISCIPLINA), são chamados de construtor do esquema. Além disso, um diagrama esquemático mostrará somente alguns aspectos do banco, como: nomes das relações, nomes dos atributos e alguns tipos de restrições.

Um dos papéis mais importantes do SGBD nisso tudo é que ele garante que cada estado do BD seja um **estado válido** -> estado que satisfaz a estrutura e as restrições definidas no esquema.

Estrutura básica de um SGBD

Em um SGBD a arquitetura é dividida em dois módulos.

Módulo Servidor

É o **"cérebro"** do SGBD, responsável por gerenciar diretamente os dados e operações críticas.

Funções Principais:

- **Armazenamento físico**: Gerencia arquivos de dados em disco.
- **Processamento de consultas**: Executa operações SQL (`SELECT` , `INSERT` , etc.).
- **Controle de transações**: Garante ACID (atomicidade, consistência, isolamento, durabilidade).
- **Segurança**: Autenticação, autorização e criptografia.
- **Gerenciamento de concorrência**: Controla acesso simultâneo (ex: locks).

Módulo Cliente

É a **interface** que permite aos usuários ou aplicações interagirem com o servidor, normalmente executado em uma estação de trabalho ou em um computador pessoal.

Funções Principais:

- **Envio de comandos**: Recebe SQL da aplicação e envia ao servidor.
- **Exibição de resultados**: Mostra dados retornados pelo servidor (ex: tabelas, mensagens de erro).
- **Ferramentas auxiliares**: Interfaces gráficas (como DBeaver, pgAdmin) ou bibliotecas (JDBC, ODBC).

Arquitetura de Três Esquemas

A **Arquitetura de Três Esquemas** é um modelo conceitual que organiza um SBD em três níveis de abstração.

Nível Interno (Físico)

Descreve **como os dados são armazenados fisicamente** no disco (arquivos, índices, estruturas de acesso). Responsável por eficiência de armazenamento e recuperação.

Nível Conceitual (Lógico)

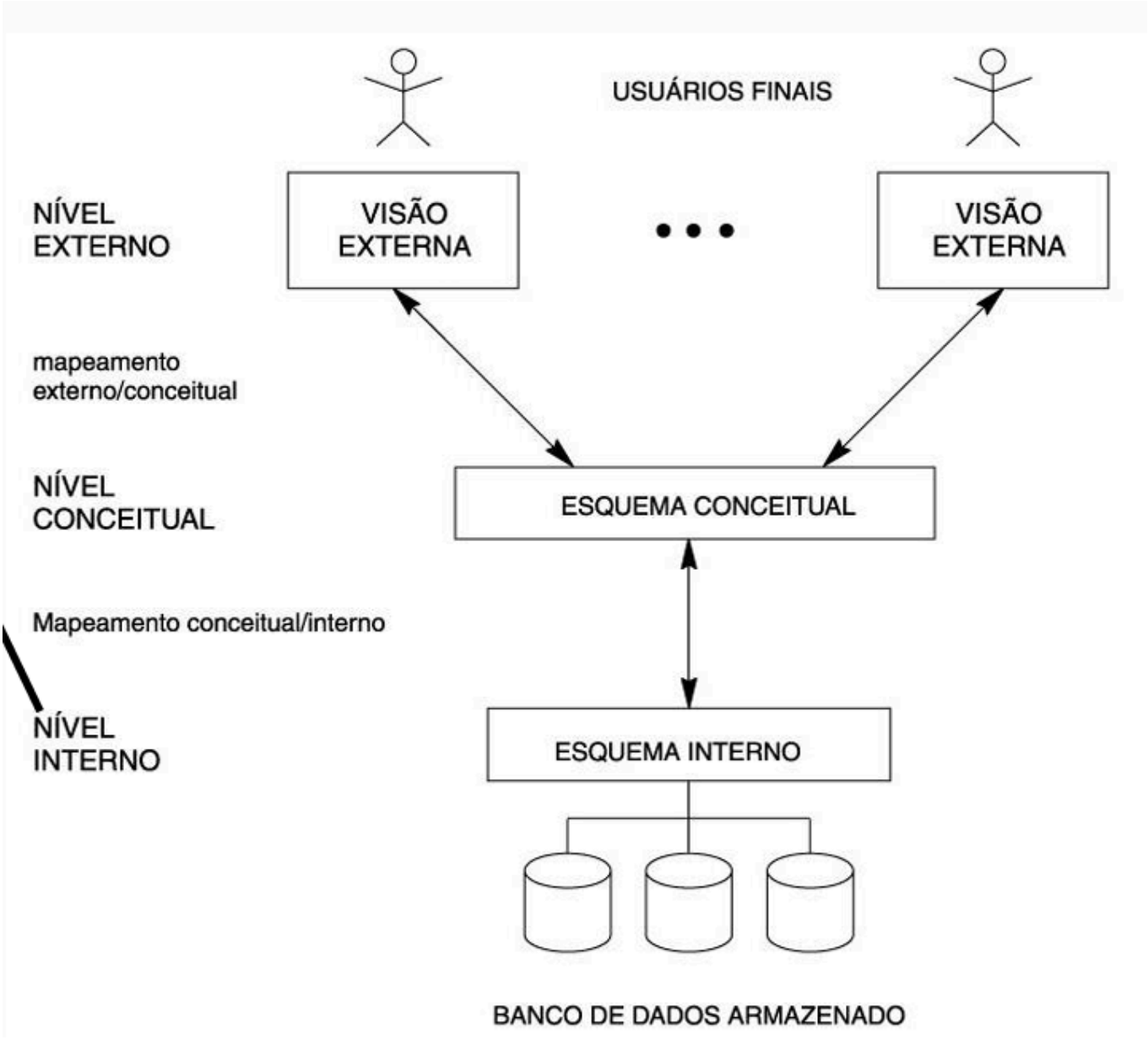
Representa a **estrutura lógica do banco de dados**, independente de implementação física ou aplicações. Responsável por integridade dos dados e regras de negócio.
Exemplo:
- Tabelas: `Clientes(id, nome, email)`, `Pedidos(id, cliente_id, valor)`.
- Relacionamentos: Chaves primárias e estrangeiras.

Nível Externo (Visões)

Define **como os usuários ou aplicações veem os dados**. Pode ser uma subsetorização ou transformação do nível conceitual. Responsável por personalização e segurança (ex: restringir acesso a colunas sensíveis).
Exemplo:
- Visão `Vendas2023`: Mostra apenas pedidos do ano 2023.
- Visão `RelatorioGerencial`: Agrega dados de múltiplas tabelas.

Resumindo tudo isso:

- **Nível Interno:** "Como os dados são guardados no disco?"
- **Nível Conceitual:** "Qual a estrutura lógica do banco?"
- **Nível Externo:** "O que cada usuário pode ver?"
  Tendo explicado tudo isso, fica um pouco mais de boa de se entender a seguinte ilustração:



Objetivos da Arquitetura

Separação entre Programas e Dados

Se a estrutura física mudar (ex: migrar de HDD para SSD), os aplicativos não precisam ser alterados. (Independência de dados lógica e física)

-Como exatamente???

- Aplicativos acessam apenas o **nível externo** ou **conceitual**.
- O SGBD traduz essas requisições para o nível físico.

Suporte a Múltiplas Visões

Diferentes usuários precisam de acessos distintos (ex: RH vs. Financeiro).

Como :0 ?

- Cada visão (nível externo) filtra ou transforma os dados do esquema conceitual.

Catálogo (Meta-dados)

- **O que é:** Um "banco de dados sobre o banco de dados" que armazena:
  - Definições de tabelas, colunas, tipos, permissões.
  - Mapeamento entre os níveis (ex: como uma visão externa se relaciona com tabelas físicas).

Resumo ultra mega curto de Linguagens em SGBDs

DDL (Data Definition Language)
- **O que faz:** Cria/altera estrutura do BD (tabelas, chaves).
- **Exemplo SQL:** `CREATE TABLE`, `ALTER TABLE`.
DML (Data Manipulation Language)
- **O que faz:** Manipula dados (CRUD: Insert, Select, Update, Delete).
- **Exemplo SQL:** `INSERT`, `SELECT`.
VDL (View Definition Language)
- **O que faz:** Define visões (telas personalizadas do BD).
- **Exemplo SQL:** `CREATE VIEW`.
SDL (Storage Definition Language)
- **O que faz:** Controla armazenamento físico (índices, partições).
- **Exemplo SQL:** `CREATE INDEX`.
SQL Unifica Tudo:

- Uma única linguagem (SQL) substitui VDL, DDL, DML e SDL na prática.

Modelo Entidade-Relacionamento (MER)

Quando estamos projetando um BD, a nossa primeira fase será a coleta e análise de requisitos, através de entrevistas com futuros usuários. Para isso a principal ferramenta a ser utilizada será o Modelo Entidade-Relacionamento (MER). O MER basicamente servirá para facilitar a comunicação entre usuários e projetistas, e sua grande vantagem é a abstração de detalhes técnicos. A representação do MER em si se dará por meio do Diagrama Entidade-Relacionamento (DER), em outras palavras

MER: Conjunto de conceitos teóricos (entidades, relacionamentos).
DER: Diagrama físico que materializa o MER. (através de losangos, retângulos, etc.)

A concepção do MER e DER ocorrerá na primeira e provavelmente a mais crítica fase do projeto de um BD, que é a fase da **Modelagem Conceitual**. Ela que irá definir o que o sistema deve armazenar, sem se preocupar com implementações técnicas (como SGBDs ou linguagens de programação).

No MER, uma **entidade** (que o objeto mais elementar que uma MER representa) é qualquer elemento do mundo real que pode ser distinguido dos demais e sobre o qual desejamos armazenar informações. Esses elementos podem ser:

- **Objetos concretos:** Como `Cliente`, `Produto` ou `Funcionário`.
- **Eventos ou conceitos:** Como `Venda`, `Consulta Médica` ou `Matrícula`.

Cada entidade deve ter uma existência independente e ser identificável de forma única dentro do sistema.

Atributos

Os atributos são as propriedades que qualificam, identificam ou classificam uma entidade. Eles podem ser classificados em:

1. **Atributos Simples**
   - Não podem ser subdivididos em componentes menores.
   - Exemplos: `data_nascimento`, `altura`, `número_de_matrícula`.
2. **Atributos Compostos**
   - Podem ser decompostos em partes menores, cada uma com significado próprio.
   - Exemplo: O atributo `endereço` pode ser dividido em `logradouro`, `número`, `CEP`, `cidade` e `estado`.
3. **Atributos Uni-valorados**
   - Atributos que possuem apenas um valor.
4. **Atributos Multivalorados**
   - Podem assumir vários valores para uma mesma entidade.
   - Exemplo: `telefones` (um cliente pode ter múltiplos números).
5. **Atributos Derivados**
   - Seus valores são calculados a partir de outros atributos.
   - Exemplo: `idade` (calculada a partir de `data_nascimento`).

Valores Nulos em Atributos

- **Definição:** Representam a ausência de valor em um atributo.
- **Casos de Uso:**
  - **Atributo não aplicável:** Quando a informação não se aplica a uma entidade específica (ex: campo "Apartamento" para quem mora em casa).
  - **Valor desconhecido:** Quando a informação existe mas não está disponível (ex: data de nascimento não informada).
- **Impacto:** Requer tratamento especial em consultas SQL (ex: `IS NULL`, `IS NOT NULL`).

Tipos de Entidade e Conjuntos de Entidades

- **Tipo de Entidade:**
  - **Definição:** Modelo abstrato que descreve uma categoria de entidades com os mesmos atributos.
  - **Exemplo:** Tipo "Funcionário" com atributos: nome, matrícula, cargo.
- **Conjunto de Entidades:**
  - **Definição:** Coleção de instâncias (entidades-membro) de um mesmo tipo.
  - **Característica:** Todas compartilham a mesma estrutura de atributos, mas com valores distintos.
  - **Exemplo:** Conjunto de todos os funcionários de uma empresa.

Atributo-Chave

- **Definição:** Atributo (ou conjunto) que identifica unicamente cada entidade em um conjunto.
- **Requisitos:**
  - **Unicidade:** Cada entidade deve ter um valor distinto para o atributo-chave.
  - **Não nulo:** Não pode ter valores nulos.
- **Tipos:**
  - **Simples:** Um único atributo como chave (ex: CPF para "Pessoa").
  - **Composta:** Combinação de atributos (ex: nome + data_nascimento + cidade).
- **Exemplo Prático:**
  - Entidade "Aluno" com atributo-chave "matrícula".
  - Entidade "Pedido" com chave composta: "número_pedido" + "loja_id".

Relação entre os Conceitos

1. Um **tipo de entidade** define a estrutura (atributos) para um **conjunto de entidades**.
2. Cada entidade no conjunto deve ter um **atributo-chave** único.
3. Atributos podem ter **valores nulos** quando não aplicáveis ou desconhecidos.

Relacionamentos e Tipos de relacionamentos

Conceitos Básicos

- **Relacionamento:** Associação entre entidades que representa uma interação significativa (ex: "EMPREGADO trabalha para DEPARTAMENTO").
- **Tipo de Relacionamento:** Padrão que define um conjunto de associações similares.

Grau do Relacionamento

Indica quantas entidades participam da associação:

- **Binário (grau 2):** Envolve duas entidades (ex: "CLIENTE compra PRODUTO").
- **Ternário (grau 3):** Envolve três entidades (ex: "FORNECEDOR fornece PEÇA para PROJETO").
- **N-ário (grau N):** Envolve N entidades (menos comum).

Papéis

- **Definição:** Função que uma entidade desempenha em um relacionamento.
- **Exemplo:** No relacionamento "EMPREGADO supervisiona EMPREGADO":
  - Papéis: "supervisor" (lado 1) e "supervisionado" (lado N).
- **OBS:** Quando as entidades são diferentes, os nomes das entidades já servem como papéis.

4. Relacionamentos Recursivos

- **Definição:** Quando a mesma entidade participa mais de uma vez no mesmo relacionamento, em papéis diferentes.
- **Exemplo:**
  - "PESSOA é amiga de PESSOA" (autorrelacionamento).
  - "ITEM é componente de ITEM" (hierarquia de peças).

Cardinalidades

Define quantas instâncias de uma entidade podem se associar a outra:

- **1:1 (Um-para-Um):** Ex: "CPF pertence a UMA PESSOA".
- **1:N (Um-para-Muitos):** Ex: "DEPARTAMENTO tem MUITOS EMPREGADOS".
- **N:M (Muitos-para-Muitos):** Ex: "ALUNO cursa MUITAS DISCIPLINAS" e "DISCIPLINA tem MUITOS ALUNOS".

Atributos de Relacionamento

- **Definição:** Dados que descrevem o relacionamento em si, não as entidades.
- **Exemplo:**
  - Em "EMPREGADO trabalha-em PROJETO", o atributo "horas_semanais" pertence ao relacionamento.

Tipos de Restrições

- **Cardinalidade:** Define quantas relações são permitidas (1:1, 1:N, N:M).

- **Participação:** Determina se a relação é obrigatória ou opcional.

Participação

- **Total (Obrigatória):**
  - A entidade **precisa** estar relacionada.
  - Ex: Todo `EMPREGADO` deve ter um `DEPARTAMENTO`.
- **Parcial (Opcional):**
  - A entidade **pode existir** sem relação.
  - Ex: Um `DEPARTAMENTO` pode existir sem `EMPREGADOS`.

Exemplo:

- **Relação:** `ALUNO` se matricula em `CURSO`.
  - **Participação Total:** Aluno **deve** ter um curso.
  - **Participação Parcial:** Curso **pode** existir sem alunos

Atributos em Relacionamentos

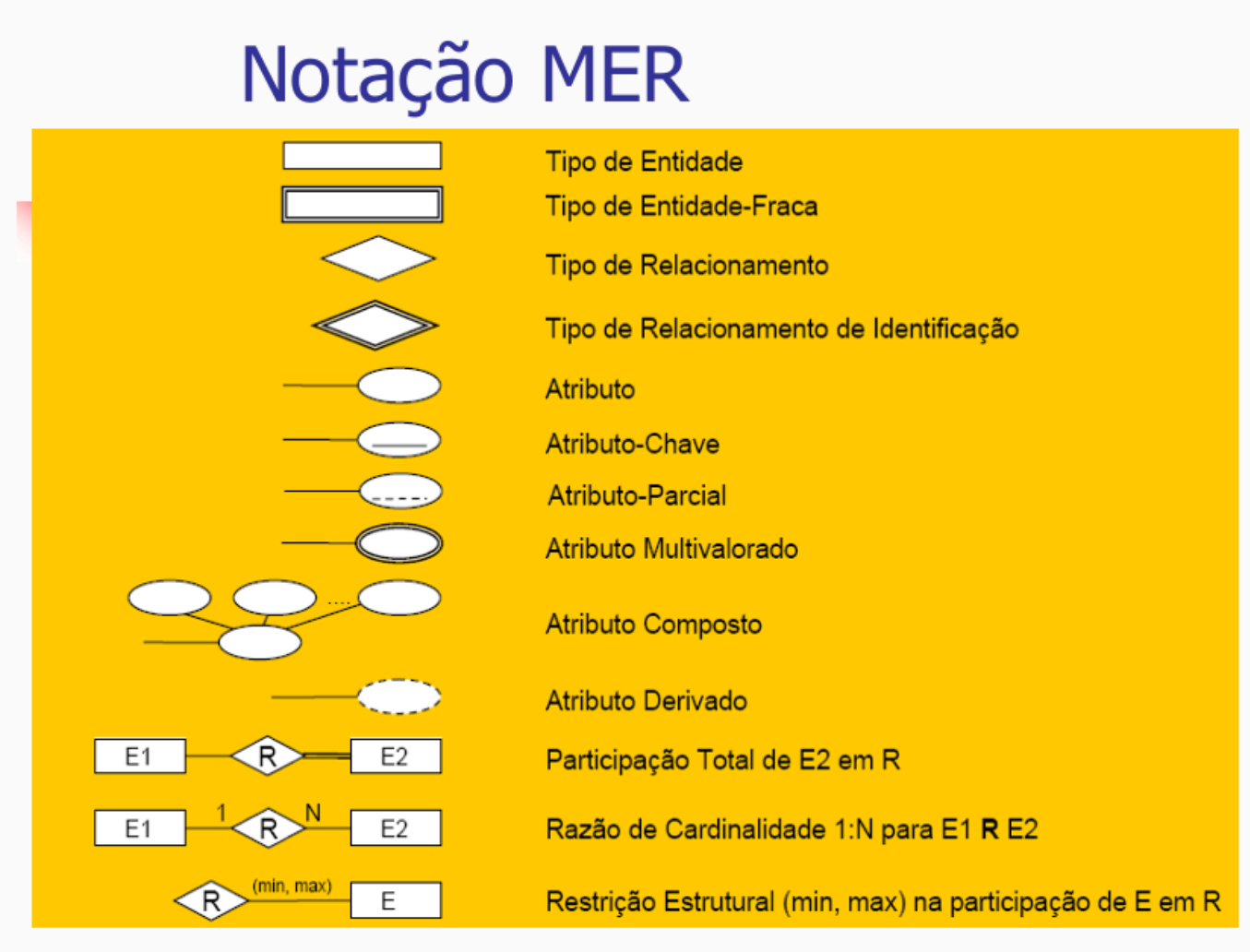
- **Definição:** Propriedades que descrevem a associação em si, não as entidades.
- **Exemplos:**
  - `horas` no relacionamento **TRABALHA-EM** (entre Empregado e Projeto).
  - `data_inicio` no relacionamento **GERENCIA** (entre Gerente e Departamento).
- **Implementação:** Em SQL, viram colunas na tabela de junção (para N:M) ou na entidade "fraca".

Entidades Fracas

- **Características:**
  - Não possuem chave primária própria.
  - Dependem de uma **entidade forte** para identificação (relacionamento de identificação).
  - Sempre têm **participação total** no relacionamento de identificação.
- **Identificação:**
  - **Chave parcial:** Atributo que distingue entidades fracas associadas à mesma entidade forte.
  - **Chave completa:** Combinação da chave da entidade forte + chave parcial
- **Exemplo:**
  - Entidade `DEPENDENTE` (fraca) vinculada a `FUNCIONÁRIO` (forte).
  - Chave parcial: `nome_dependente`.

- Chave completa: `matricula_funcionario` + `nome_dependente`

A seguir algumas notações importantes



Modelo de Entidade-Relacionamento Estendido (EER)

O **EER** estende o **MER** com conceitos avançados para representar estruturas mais complexas, como hierarquias de classes e restrições adicionais.

Generalização e Especialização

Especialização (Top-Down)

- **Definição:** Processo de dividir uma **superclasse** em **subclasses** mais específicas.
- **Exemplo:**
  - Superclasse: `Empregado`
  - Subclasses: `Secretário`, `Engenheiro`, `Técnico`

- **Características:**
  - Subclasses herdam **atributos** e **relacionamentos** da superclasse.
  - Podem ter **atributos próprios** (ex: `Engenheiro` tem `crea`).
  - Representação em DER: Símbolo de **herança** (triângulo).

Generalização (Bottom-Up)

- **Definição:** Processo de unir **subclasses** em uma **superclasse** comum.
- **Exemplo:**
  - Subclasses: `Carro`, `Moto`, `Caminhão` → Superclasse: `Veículo`
- **Objetivo:** Eliminar redundâncias (atributos comuns como `placa`, `ano`).

Subclasses e Superclasses

Termo	Definição	Exemplo
Superclasse	Entidade "pai" que generaliza outras.	`Pessoa`
Subclasse	Entidade "filha" que herda da superclasse.	`Aluno`, `Professor`

Herança

- **Atributos:** Subclasses herdam todos os atributos da superclasse.
  - Ex: `Aluno` herda `nome` e `CPF` de `Pessoa`.
- **Relacionamentos:** Subclasses participam dos relacionamentos da superclasse.
  - Ex: Se `Pessoa` tem relacionamento com `Endereço`, `Aluno` também tem.

Restrições de Disjunção

Definem se uma entidade pode pertencer a **uma ou múltiplas subclasses** simultaneamente.

Tipo	Símbolo	Descrição	Exemplo
Disjunta (Mut. Exclusiva)	`d`	Cada entidade pertence a **no máximo uma** subclasse.	`Veículo` é `Carro` **ou** `Moto`.
Sobreposta (Overlap)	`o`	Uma entidade pode pertencer a **várias subclasses** ao mesmo tempo.	`Funcionário` pode ser `Gerente` **e** `Engenheiro`.

Restrições de Integralidade

Definem se **todas** as entidades da superclasse **devem** pertencer a alguma subclasse.

Combinações de Restrições

As restrições de disjunção e integralidade são independentes, gerando 4 combinações possíveis:

Caso	Disjunção	Integralidade	Exemplo
Disjunção Total	`d`	Total	`Pessoa` deve ser `Aluno` **ou** `Professor` (nunca ambos).
Disjunção Parcial	`d`	Parcial	`Veículo` pode ser `Carro` **ou** `Moto` (ou nenhum).
Sobreposição Total	`o`	Total	`Funcionário` deve ser `Gerente` **e/ou** `Engenheiro`.
Sobreposição Parcial	`o`	Parcial	`Pessoa` pode ser `Atleta` **e** `Estudante` (ou nenhum).

Exemplo Prático

Cenário: Sistema de RH de uma universidade.

- **Disjunção:**
  - `PESSOA` é **disjunta** entre `ALUNO` e `PROFESSOR` (não pode ser ambos).
  - `FUNCIONARIO` é **sobreposta** com `PESSOA` (pode ser `PROFESSOR` **e** `FUNCIONARIO`).
- **Integralidade:**
  - `PESSOA` → `ALUNO` / `PROFESSOR`: **Parcial** (pode ser apenas `PESSOA`).
  - `FUNCIONARIO` → `TECNICO` / `ADMINISTRATIVO`: **Total** (todo funcionário tem um cargo).

Por que são importantes?

- **Precisão:** Refletem regras de negócio complexas (ex: um `Veículo` não pode ser `Carro` e `Moto`).
- **Consistência:** Evitam dados inválidos (ex: `Pessoa` sem classificação obrigatória).
- **Clareza:** Documentam constraints diretamente no diagrama EER.
  Resumo Final:
- **Disjunção:** "Pode ser de mais de um tipo?" (`d` = não, `o` = sim).
- **Integralidade:** "Precisa ser de algum tipo?" (`||` = sim, `|` = não).

Hierarquias de Especialização no Modelo EER

1. Herança Simples

- **Definição:** Cada subclasse tem **apenas uma** superclasse direta.
- **Características:**
  - Forma uma estrutura hierárquica em árvore.
  - Mais simples de implementar em bancos de dados relacionais.
- **Exemplo:**
  - Superclasse: `Veículo`
  - Subclasses: `Carro`, `Moto` (cada uma herda apenas de `Veículo`)

2. Herança Múltipla

- **Definição:** Uma subclasse pode ter **várias superclasses** diretas.
- **Características:**
  - Permite modelagem mais flexível, mas mais complexa.
  - Pode causar conflitos de atributos (quando superclasses têm atributos com mesmo nome).
- **Exemplo:**
  - Superclasses: `Funcionário`, `Estudante`
  - Subclasse: `Estagiário` (herda de ambas

  Uma **subclasse compartilhada** é uma entidade que herda características de **duas ou mais superclasses** simultaneamente, representando um caso de **herança múltipla**. Ela combina atributos e relacionamentos de todas as suas superclasses.

Características Principais

1. **Herança Múltipla:**
   - Herda atributos e relacionamentos de todas as superclasses.
   - Exemplo: `Gerente_Engenharia` herda de `Gerente` (atributo: `bonus`) e `Engenheiro` (atributo: `crea`).
2. **Restrição de Identificação:**
   - Depende das superclasses para existir (participação total).
   - Não pode ser instanciada sem vinculação a pelo menos uma superclasse.

Cardinalidade em Relacionamentos Ternários

1. Conceito Fundamental

Em relacionamentos ternários (envolvendo 3 entidades), a cardinalidade indica quantas instâncias de uma entidade podem estar associadas a um **par fixo** das outras duas entidades.

2. Regra para Determinar Cardinalidades

Siga estes passos:

1. **Fixe duas entidades** do relacionamento.
2. **Analise a terceira entidade**:
   - Se o número de instâncias associadas for **variável ou ilimitado** → Cardinalidade **N**.
   - Se for **fixo** (ex: sempre 1) → Cardinalidade **1**.

3. Exemplo Prático: Aluno-Monitora-Disciplina-Professor

- **Entidades:** `Aluno`, `Disciplina`, `Professor`.
- **Relacionamento:** `Monitora` (um aluno monitora uma disciplina para um professor).

Análise de Cardinalidade:

1. Fixe `Aluno` e `Disciplina`:
   - Quantos `Professores` podem estar associados?
     - Se um aluno monitora uma disciplina para **vários professores** → Cardinalidade **N**.
     - Se for para **apenas um professor** → Cardinalidade **1**.
2. Fixe `Aluno` e `Professor`:
   - Quantas `Disciplinas` podem ser monitoradas?
     - **N** (se o aluno pode monitorar múltiplas disciplinas para o mesmo professor).
3. Fixe `Disciplina` e `Professor`:
   - Quantos `Alunos` podem monitorar?
     - **N** (vários alunos podem monitorar a mesma disciplina-professor).