

RESIDÊNCIA DE SOFTWARE



# Projeto de Banco de Dados

Prof. M.Sc. Maria Luiza Mondelli  
[malumondelli@gmail.com](mailto:malumondelli@gmail.com)

# In a nutshell

**Maria Luiza Mondelli (ou Malu)**

[malumondelli@gmail.com](mailto:malumondelli@gmail.com)

Técnica em Informática

Centro de Educação Profissional em Tecnologia da Informação

Tecnóloga em TIC

Faculdade de Educação Tecnológica do Estado do Rio de Janeiro

Mestre em Modelagem Computacional

Laboratório Nacional de Computação Científica

Aluna de Doutorado em Modelagem Computacional

Laboratório Nacional de Computação Científica



Banco de Dados  
Workflows Científicos  
Bioinformática  
Biodiversidade



# Plano de aula

Aula	Descrição
1	Introdução e Modelos de BD
2	Modelo Entidade Relacionamento
3	Formas Normais
4	SQL - DDL, DML
5	SQL - DQL
6	SQL - DQL e DCL

# Avaliações

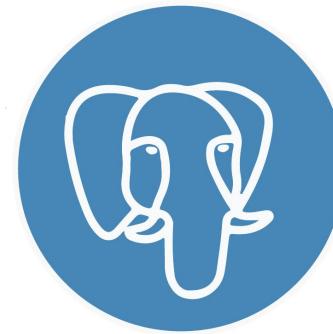
## Planejamento

- 1<sup>a</sup> avaliação (10 questões)  
*terça, 21/07*  30 pontos
- 2<sup>a</sup> avaliação (5 questões)  
*quinta, 23/07* 
- Projeto final  60 pontos  
*sexta, 24/07 e segunda, 27/07*
- Capacidades sociais,  
organizativas e metodológicas  10 pontos

# Para instalar



[DBeaver](#)



[PostgreSQL](#)



[SQLite](#)

(O DBeaver já costuma ter)

*Se não for possível instalar, [aqui tem um plano B.](#)*



# Introdução

## O que são dados?



'**Everything our senses perceive is data**, though its storage in our cranial wet stuff leaves something to be desired. Writing it down is a bit more reliable, especially when we write it down on a computer.'

Cassie Kozyrkov | Head of Decision Intelligence, Google

"**Data is the new oil**. It's valuable, but if unrefined it cannot really be used. It has to be changed into gas, plastic, chemicals, etc to create a valuable entity that drives profitable activity; so must data be broken down, analyzed for it to have value."

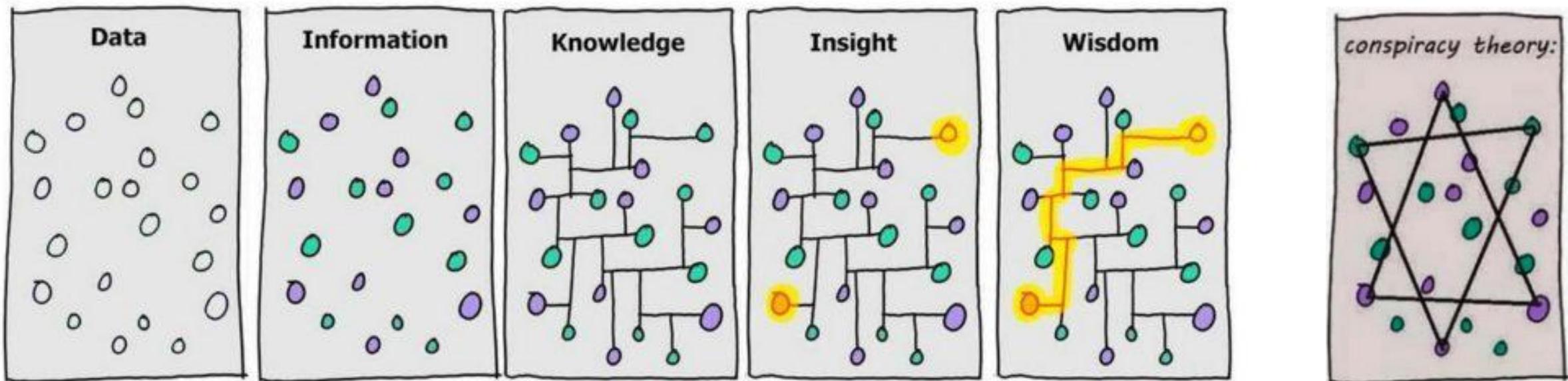
Clive Humby | UK mathematician



# Introdução

Ou ainda: fatos conhecidos que podem ser registrados e têm significado implícito.

Sistemas de Bancos de Dados | Elmasri & Navathe



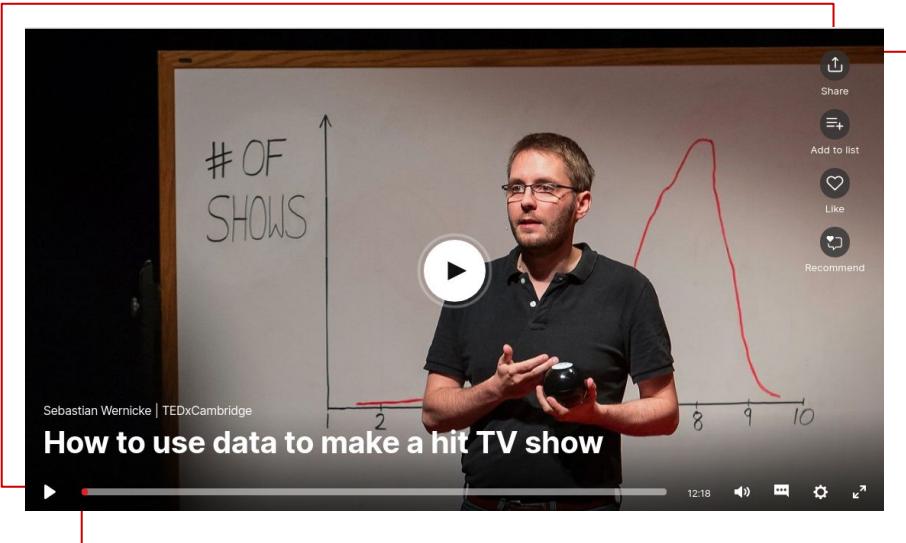
# Introdução

E por que dados são importantes?



# Para assistir

uma outra hora ;)



**TED**  
Ideas worth spreading



# Introdução

## Algumas definições

### Bancos de dados

Coleção de dados relacionados

- Representa alguns aspectos do mundo real, sendo chamado, às vezes, de **mini-mundo** ou de universo de discurso (UoD)
- É uma coleção lógica e coerente de dados com algum significado inerente
- É projetado, construído e povoado por dados, atendendo a uma **proposta específica**
- Possui algumas fontes das quais os dados são derivados
- Pode ser de **qualquer tamanho** e tem **complexidade variável**

**Definir** um banco de dados implica especificar os tipos de dados, as estruturas e as restrições para os dados a serem armazenados.



# Introdução

Dito isso...

Uma planilha pode ser considerada  
um banco de dados?



# Introdução

## Algumas definições

### Por que usar bancos de dados?

- Organização e tratamento estruturado dos dados
- Padronização e eficiência do acesso, útil para o desenvolvimento de aplicações
- Independência de dados em relação às aplicações
- Facilidade de recuperação de informações
- Consistência de acordo com as regras de negócio definidas
- Controle de redundância
- Restrições de integridade
- Backup e restauração
- Controle de concorrência



# Introdução

## Algumas definições

### Sistema Gerenciador de Banco de Dados (SGBD)

'is a collection of interrelated data and a set of programs to access those data. [...] [The goal] is to provide a way to store and retrieve database information that is both convenient and efficient.' **Database System Concepts** | Silberschatz et al.

- Sistema de software de propósito geral que **facilita** os processos entre vários usuários e aplicações  
Definição, construção, manipulação e compartilhamento de bancos de dados
- **Construir** um banco de dados é o processo de **armazenar** os dados em alguma mídia apropriada controlada pelo SGBD
- A **manipulação** permite pesquisas em banco de dados para recuperar um dado específico, atualização do banco para refletir as mudanças no minimundo e geração de relatórios dos dados
- O **compartilhamento** permite aos múltiplos usuários e programas acessar, de forma **concorrente**, o banco de dados



# Introdução

## Rápido histórico

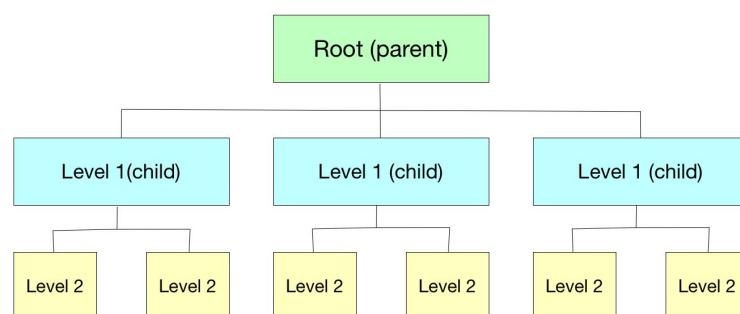
### Antes de 1960

- Registros eram feitos em papéis (listas, arquivos, jornais)
- Acesso difícil e trabalhoso
- Problemas de segurança, registros fora do lugar, incêndios

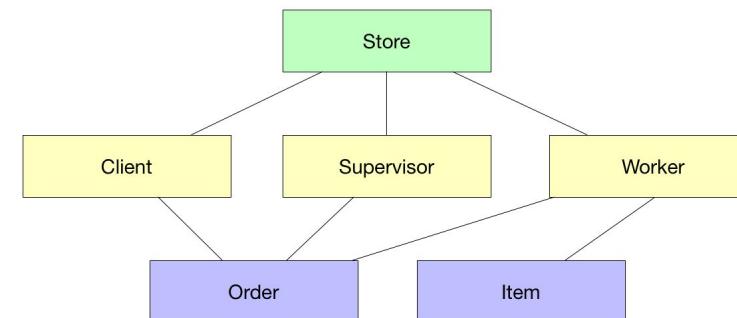
### 1960s

- O uso de computadores se tornou uma opção viável para organizações privadas
- Dados passam a ser armazenados em arquivos
- Surgimento dos modelos hierárquico (IMS) e rede (CODASYL)

The Hierarchical Database Model



The Network Database Model



# Introdução

## Rápido histórico

1970

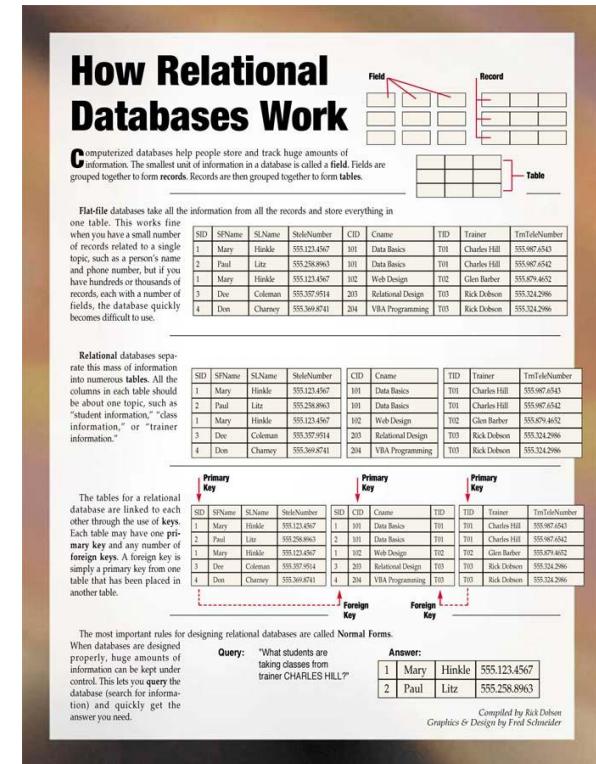
- Surgimento do modelo relacional (E.F. Codd, pesquisador da IBM)
- Separação dos níveis lógico e físico do esquema do banco de dados
- Baseado em álgebra relacional
- Se tornou o princípio padrão para sistemas de banco de dados
- Representado por uma coleção de tabelas (entidade/relação) e um conjunto de linhas (tuplas) e colunas (atributos)

Entre 1974 e 1977 Surgimento de dois sistemas de banco de dados

- INGRES, Universidade da Califórnia, Berkeley
- System R, IBM, San Jose

1976 Criação do modelo entidade-relacionamento (Dr. Peter Chen)

- Foco na aplicação dos dados ao invés da estrutura lógica das tabelas



# Introdução

## Rápido histórico

1970

- Surgimento do modelo relacional (E.F. Codd, pesquisador da IBM)
- Separação dos níveis lógico e físico do esquema do banco de dados
- Baseado em álgebra relacional
- Se tornou o princípio padrão para sistemas de banco de dados
- Representado por uma coleção de tabelas (entidade/relação) e um conjunto de linhas (tuplas) e colunas (atributos)

Entre 1974 e 1977 Surgimento de dois sistemas de banco de dados

- INGRES, Universidade da Califórnia, Berkeley
- System R, IBM, San Jose



1976 Criação do modelo entidade-relacionamento (Dr. Peter Chen)

- Foco na aplicação dos dados ao invés da estrutura lógica das tabelas



# Introdução

## Rápido histórico

1970

- Surgimento do modelo relacional (E.F. Codd, pesquisador da IBM)
- Separação dos níveis lógico e físico do esquema do banco de dados
- Baseado em álgebra relacional
- Se tornou o princípio padrão para sistemas de banco de dados
- Representado por uma coleção de tabelas (entidade/relação) e um conjunto de linhas (tuplas) e colunas (atributos)

Entre 1974 e 1977 Surgimento de dois sistemas de banco de dados

- INGRES, Universidade da Califórnia, Berkeley
- System R, IBM, San Jose

1976 Criação do modelo entidade-relacionamento (Dr. Peter Chen)

- Foco na aplicação dos dados ao invés da estrutura lógica das tabelas



# Introdução

## Rápido histórico

1980

- A linguagem SQL (Structured Query Language) se tornou o padrão para consultas em bancos de dados relacionais (ISO e ANSI)
- Surgimento de novos SGBDs (DB2, ORACLE, PostgreSQL, Clipper)

1985 Surgimento de bancos de dados orientados a objetos

- Objeto: classes, atributos, métodos, herança

1990

- Esforço para a criação de bancos de dados que estendem a teoria relacional com formas mais naturais de armazenar objetos

1995 Surgimento das primeiras aplicações na Internet com acesso à bancos de dados

- Demanda maior por arquitetura cliente/servidor (MySQL)



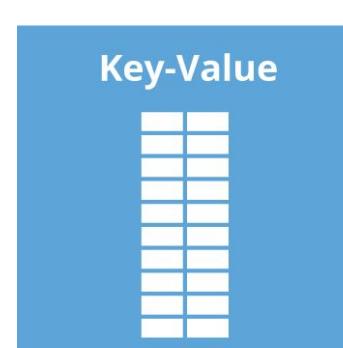
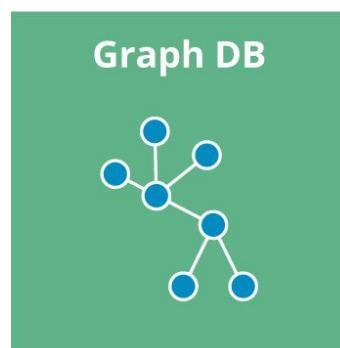
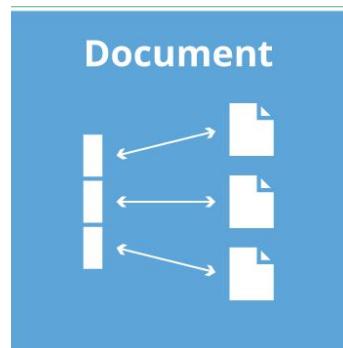
# Introdução

## Rápido histórico

### 2000 até agora

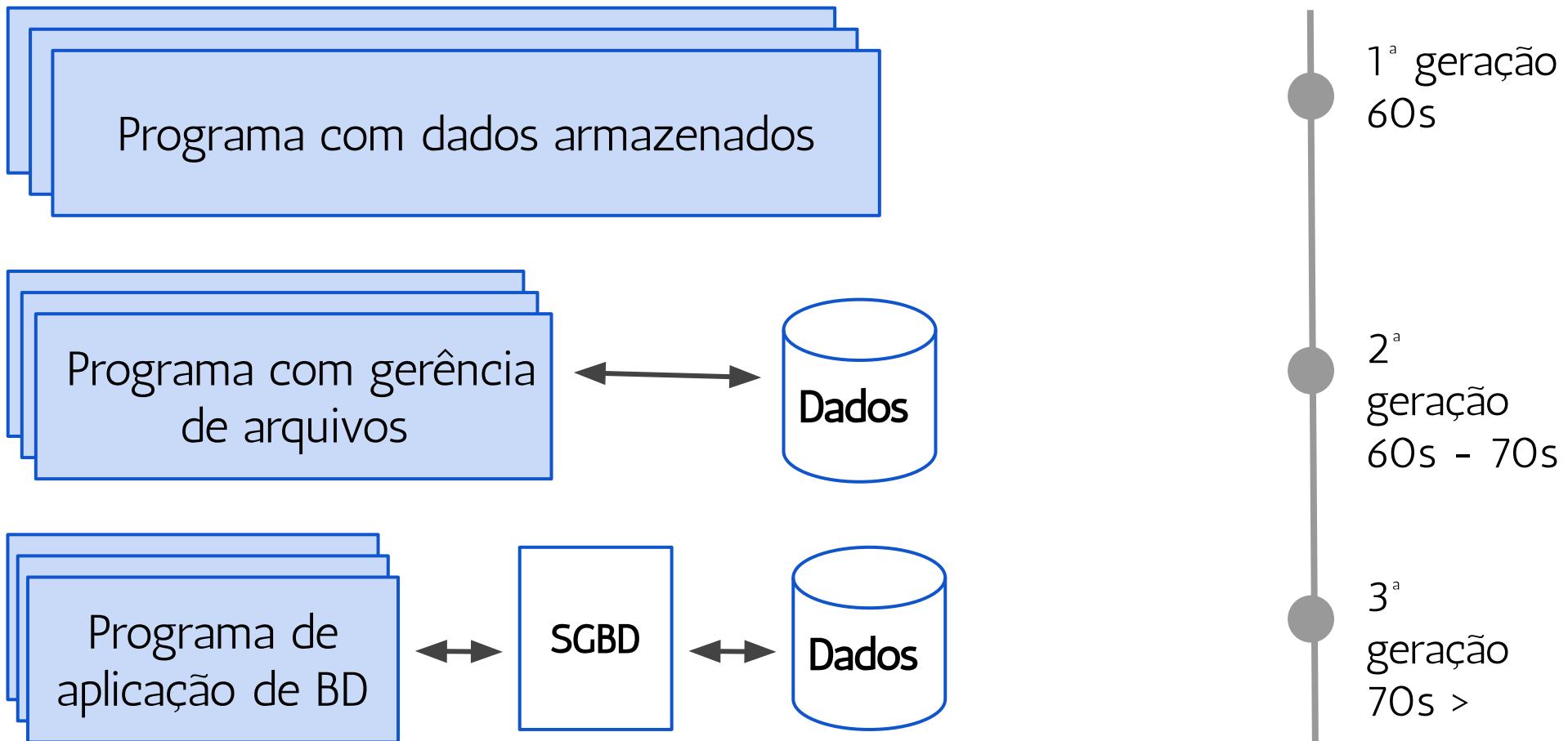
Surgimento de abordagens NoSQL (não só SQL ou não relacional)

- Orientados a documentos, como XML, Json, MongoDB  
Dados semi-estruturados
- Orientados a grafos: Neo4J  
Relacionamentos são mais *naturais* (vértices e arestas)
- Chave - valor: Redis  
Conjunto de pares de chave - valor em que uma chave funciona como um identificador exclusivo



# Introdução

## Resumindo



# Introdução

## Como estamos hoje

### The DB-Engines Ranking

- **Number of mentions of the system on websites**, measured as number of results in search engines queries. At the moment, we use [Google](#), [Bing](#) and [Yandex](#) for this measurement. In order to count only relevant results, we are searching for <system name> together with the term database, e.g. "Oracle" and "database".
- **General interest in the system**. For this measurement, we use the frequency of searches in [Google Trends](#).
- **Frequency of technical discussions about the system**. We use the number of related questions and the number of interested users on the well-known IT-related Q&A sites [Stack Overflow](#) and [DBA Stack Exchange](#).
- **Number of job offers, in which the system is mentioned**. We use the number of offers on the leading job search engines [Indeed](#) and [Simply Hired](#).
- **Number of profiles in professional networks, in which the system is mentioned**. We use the internationally most popular professional networks [LinkedIn](#) and [Upwork](#).
- **Relevance in social networks**. We count the number of [Twitter](#) tweets, in which the system is mentioned.



# Introdução

## Como estamos hoje

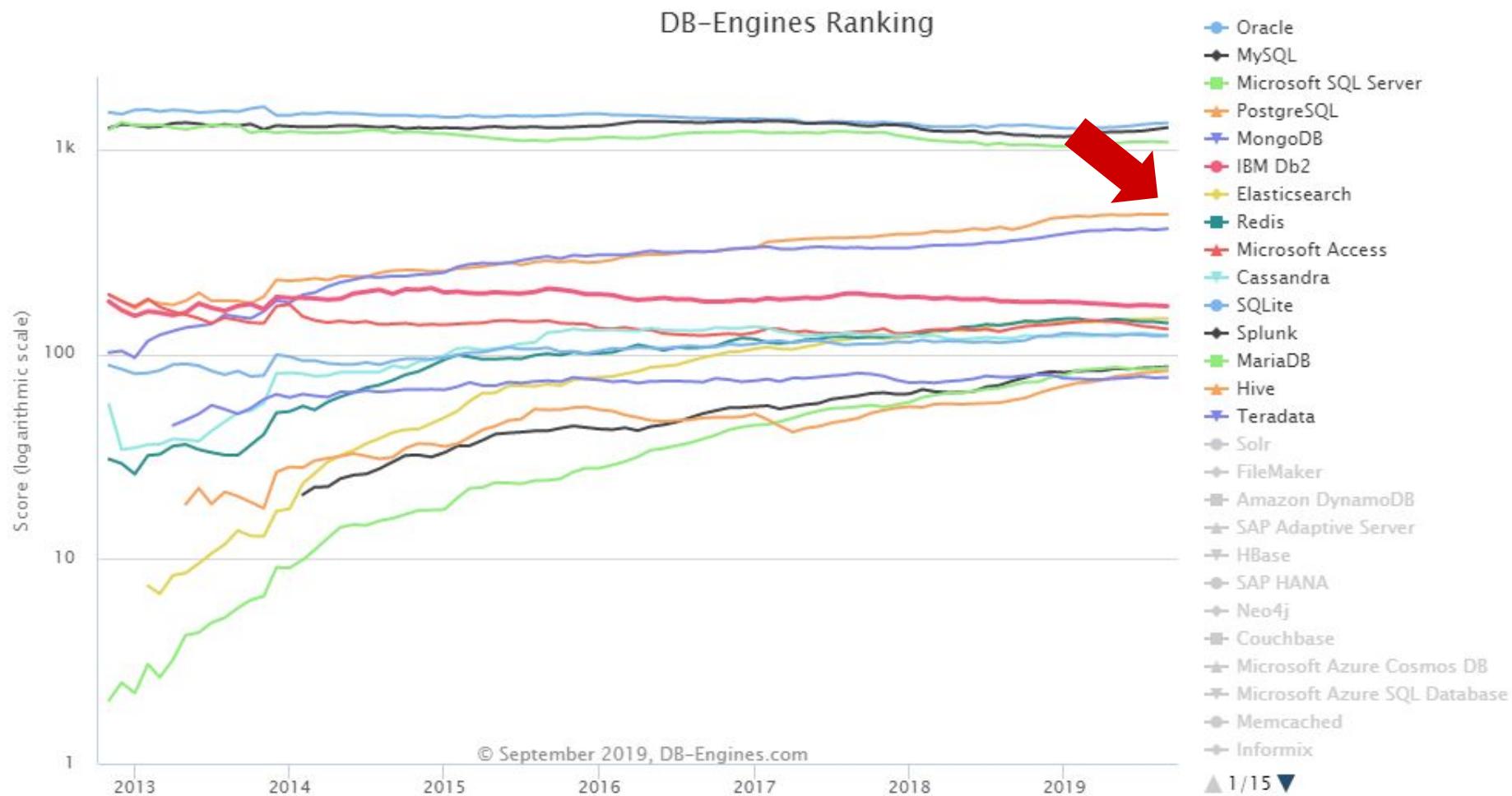
358 systems in ranking, July 2020

Rank			DBMS	Database Model	Score		
Jul 2020	Jun 2020	Jul 2019			Jul 2020	Jun 2020	Jul 2019
1.	1.	1.	Oracle 	Relational, Multi-model 	1340.26	-3.33	+19.00
2.	2.	2.	MySQL 	Relational, Multi-model 	1268.51	-9.38	+38.99
3.	3.	3.	Microsoft SQL Server 	Relational, Multi-model 	1059.72	-7.59	-31.11
4.	4.	4.	PostgreSQL 	Relational, Multi-model 	527.00	+4.02	+43.73
5.	5.	5.	MongoDB 	Document, Multi-model 	443.48	+6.40	+33.55
6.	6.	6.	IBM Db2 	Relational, Multi-model 	163.17	+1.36	-10.97
7.	7.	7.	Elasticsearch 	Search engine, Multi-model 	151.59	+1.90	+2.77
8.	8.	8.	Redis 	Key-value, Multi-model 	150.05	+4.40	+5.78
9.	9.	↑ 11.	SQLite 	Relational	127.45	+2.64	+2.82
10.	10.	10.	Cassandra 	Wide column	121.09	+2.08	-5.91



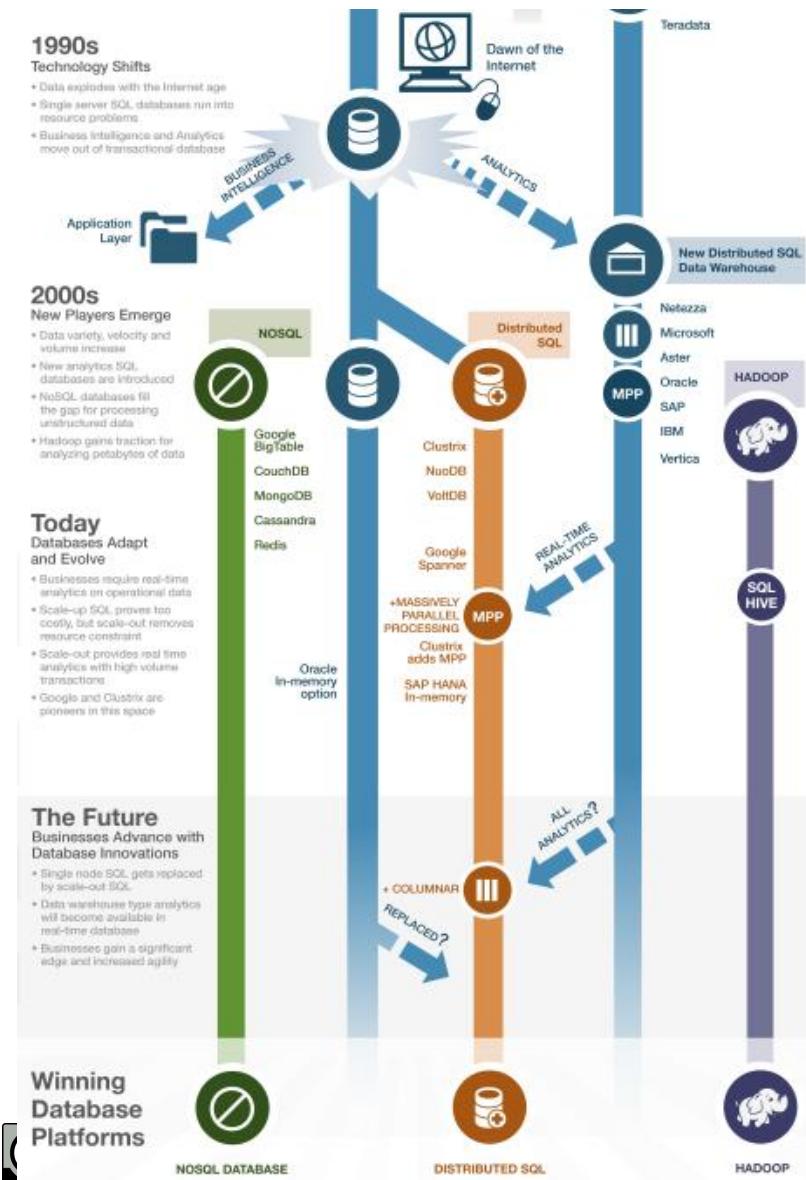
# Introdução

## Como estamos hoje



# Introdução

What's next?



## O futuro do banco de dados está na nuvem, segundo Gartner

Conforme estudo da consultoria, 75% das bases de dados terão migrado ou sido implementadas na cloud em 2022

Da Redação

03/07/2019 às 14h00



# Uma primeira prática

Pra dar uma ideia do que vamos ver

1. [Acessar esse site aqui](#)

2. Criar a tabela aluno

```
create table aluno (matricula integer primary key, nome varchar(20))
```

3. Inserir dados

```
insert into aluno (matricula, nome) values (123, 'Maria'), (124, 'Gabriel')
```

4. Selecionar dados

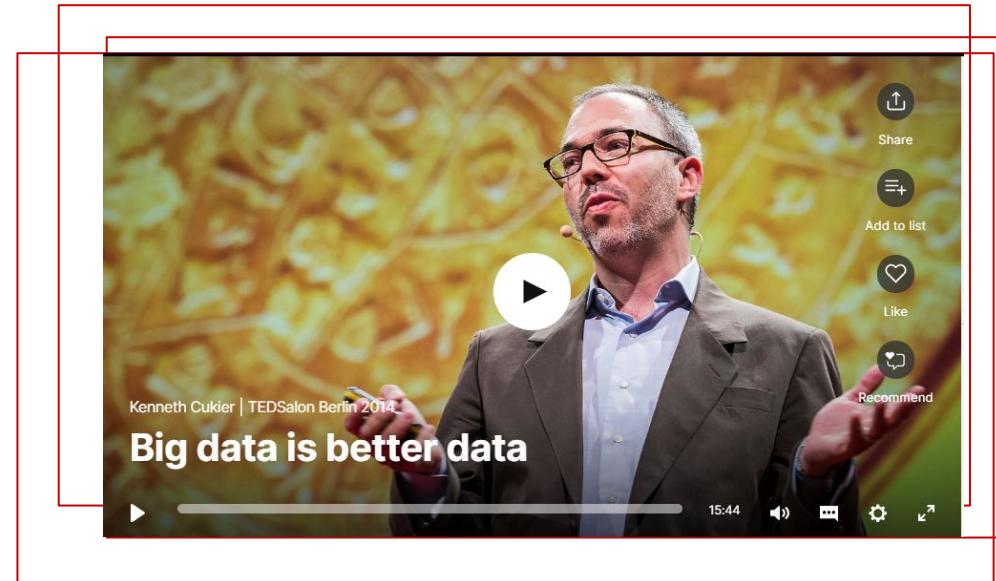
```
select * from aluno
```

aluno	
matricula	nome
123	Maria
124	Gabriel



# Para assistir

uma outra hora ;)



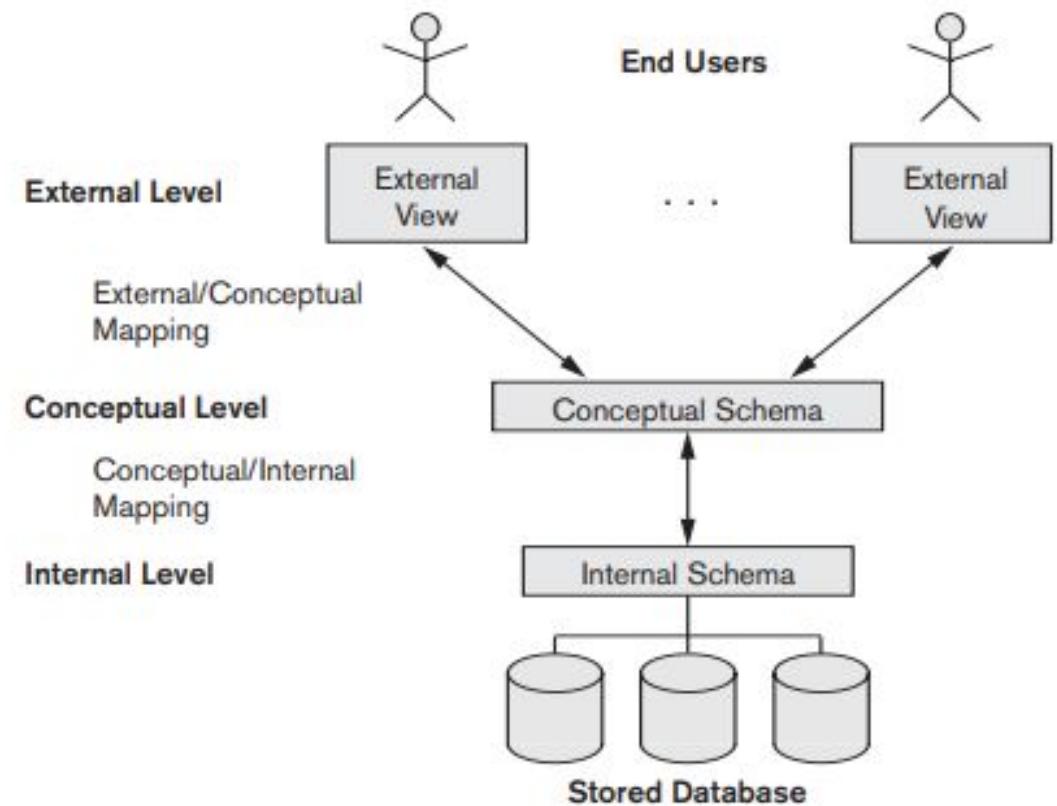
# Introdução

## Voltando aos SGBDs

### Arquitetura

Separar aplicações do usuário dos dados físicos

- **Nível interno:** usa um modelo de dados que mostra a estrutura de **armazenamento físico** do banco de dados, os detalhes dos dados guardados e os caminhos de acesso
- **Nível conceitual:** descreve o **modelo** do banco de dados, sem detalhar como os dados serão armazenados, num **alto-nível** de abstração
- **Nível externo:** descreve as **visões** do banco de dados que um grupo de usuários, por exemplo, pode ter acesso



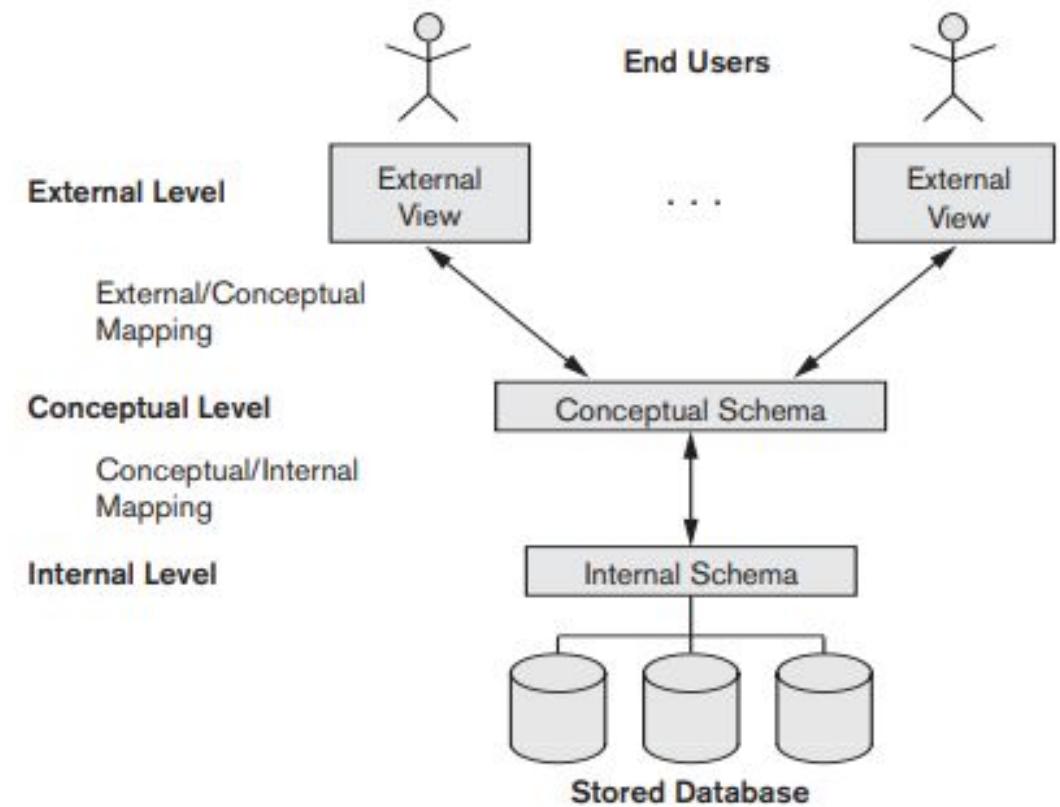
# Introdução

## Voltando aos SGBDs

### Independência de dados

Capacidade de mudar o esquema num nível de um sistema de banco de dados sem ter que mudar o esquema no nível seguinte mais alto

- **Independência de dados lógica:** alterar apenas o nível conceitual, sem que o nível externo ou nas aplicações do usuário sejam alterados
- **Independência de dados física:** alterar o nível interno sem ter que alterar o nível conceitual, nível externo ou as aplicações do usuário



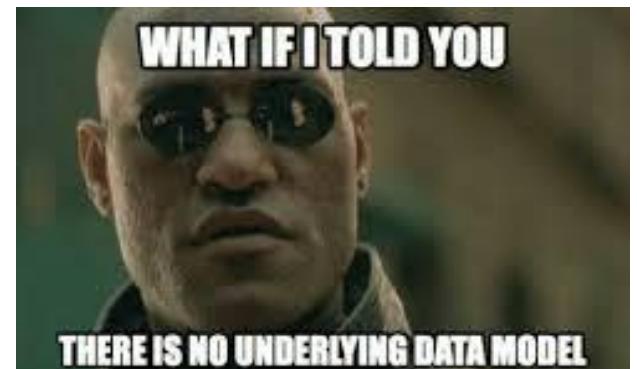
# Modelos de Banco Dados

## Modelo de dados

- Pode ser entendido como um meio para **comunicar** os conceitos que serão implementados no banco de dados.
- Permite a descrição, análise e especificação de ideias
- Deve ser **informativo** - detalhamento suficiente para que possa ser construído

## Modelagem de dados

- **Processo de criação** de um modelo de dados de acordo com o processo de negócio que está sendo considerado
- Depende de **técnicas** específicas - que vamos ver mais à frente
- Fornece a **estrutura** na qual os dados serão utilizados por um sistema/aplicação



# Modelos de Banco de Dados

Tipos de modelos

1. Conceitual
2. Lógico
3. Físico



# Modelos de Banco de Dados

## Tipos de modelos

### 1. Conceitual

- Primeira fase da modelagem
- Representação do modelo em um alto nível de abstração, independente do SGBD que será utilizado
- Registra **quais** dados **podem** aparecer no banco, mas **não** registra **como** estes dados estão armazenados no SGBD

### Exemplos

- Cadastro de alunos em um curso  
Dados necessários: nome, nascimento, endereço, etc.
- Cadastro de pedidos  
Dados necessários: código do produto, quantidade, código do cliente, código do vendedor.



Descreva um modelo conceitual para armazenar os dados de livros.

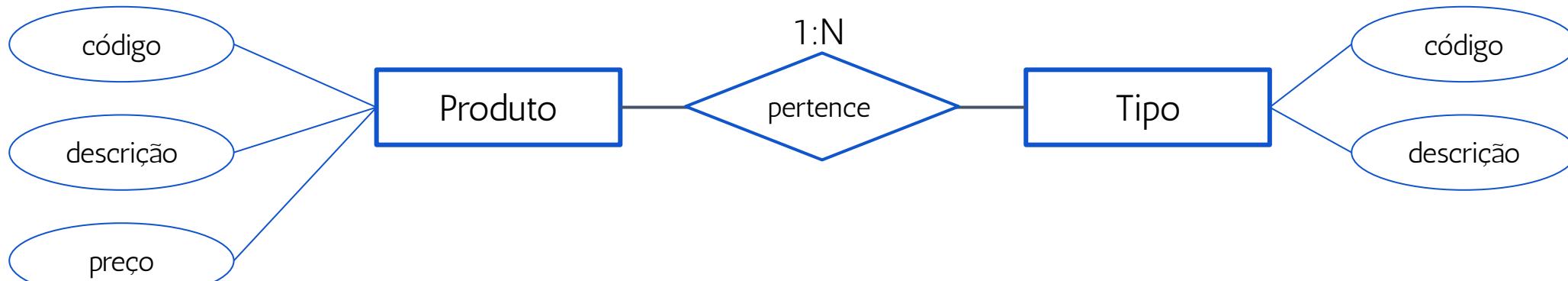


# Modelos de Banco de Dados

## Tipos de modelos

### 2. Lógico

- Descreve as **estruturas** que serão armazenadas no banco de dados
- Utiliza representação gráfica dos dados de uma maneira lógica, nomeando os componentes e ações que exercem uns sobre os outros
- Ainda é **independente** do SGBD



# Modelos de Banco de Dados

## Tipos de modelos

### 3. Físico

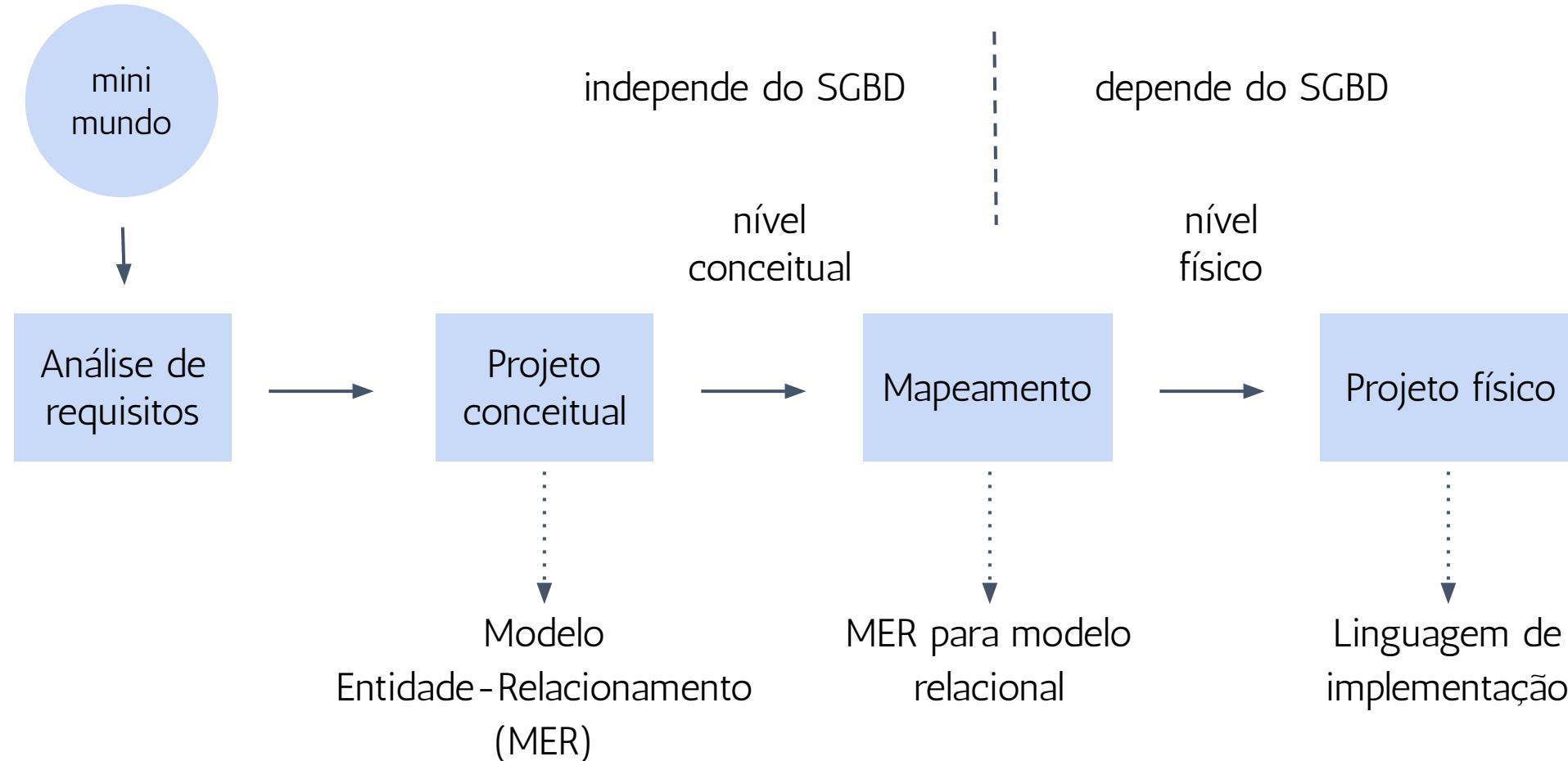
- Descreve de um banco de dados no **nível de abstração** visto pelo usuário do **SGBD**
- Diferente do conceitual e lógico, esse **depende** do SGBD que está sendo usado
- Indica quais tabelas, campos, tipos de valores, etc. serão utilizados

```
CREATE TABLE turma (
    idturma INTEGER(4) NOT NULL AUTO_INCREMENT,
    capacidade INTEGER(2) NOT NULL,
    idProfessor INTEGER(4) NOT NULL,
    PRIMARY KEY (idturma),
    FOREIGN KEY(idProfessor) REFERENCES professor(idProfessor),
    UNIQUE KEY idturma (idturma)
)
```

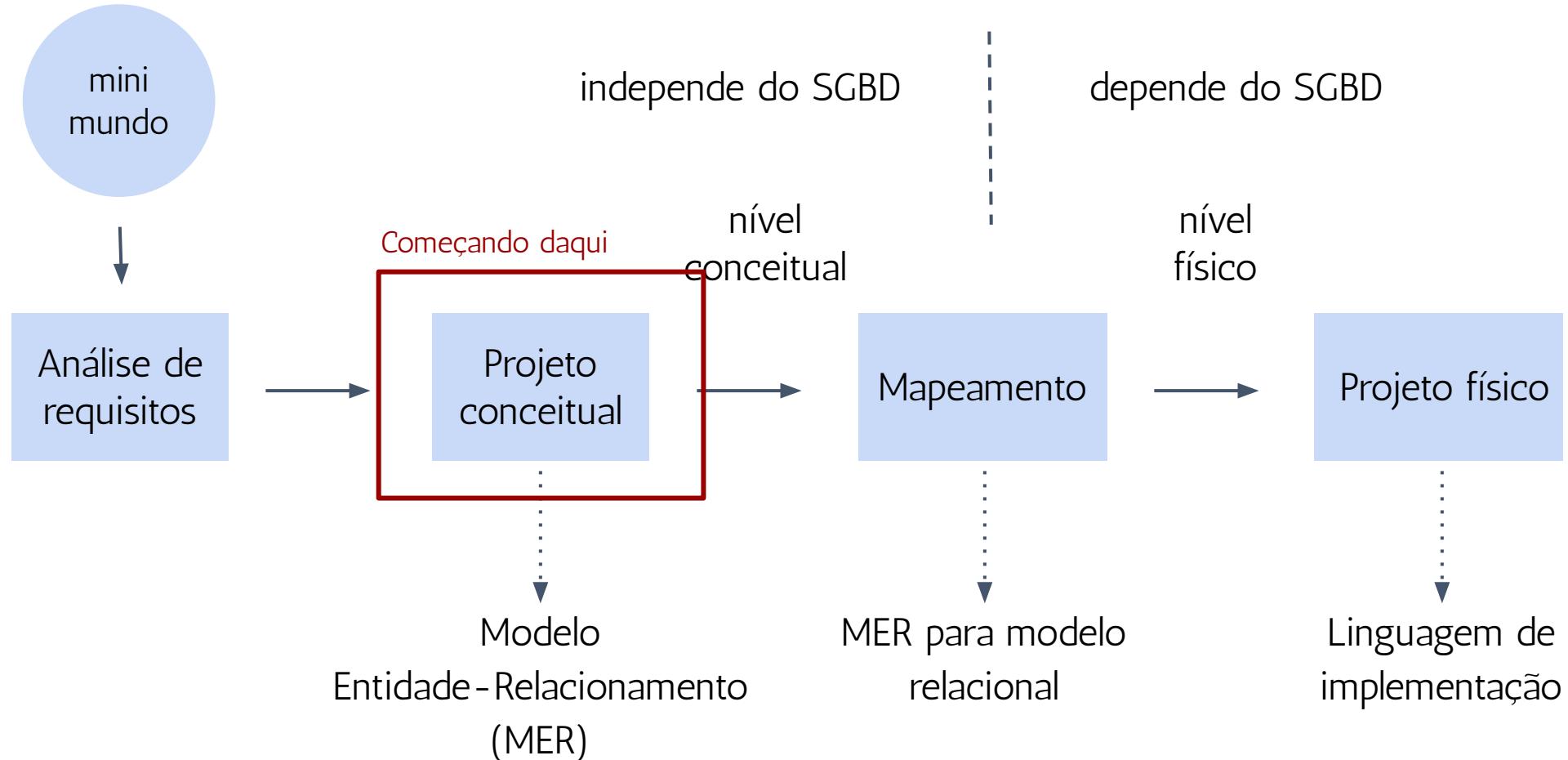
```
CREATE TABLE professor (
    idProfessor INTEGER(4) NOT NULL AUTO_INCREMENT,
    telefone INTEGER(10) NOT NULL,
    nome CHAR(80) COLLATE NOT NULL DEFAULT '',
    PRIMARY KEY (idProfessor),
    FOREIGN KEY(idTurma) REFERENCES turma(idTurma),
    UNIQUE KEY idProfessor (idProfessor)
)
```



# Projeto de Banco de Dados



# Projeto de Banco de Dados



# MER

- É um modelo de dados **conceitual** que representa de forma sistemática um modelo de negócios
- **Descreve** os objetos envolvidos no domínio (mini-mundo)  
Entidades, atributos e relacionamentos
- É uma forma abstrata de indicar qual vai ser a estrutura do banco de dados
- O diagrama entidade-relacionamento (DER) é utilizado para representar **graficamente** esses objetos

Qual a diferença entre modelo e diagrama?

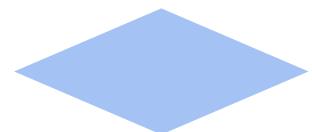
Componentes do DER



Entidade



Atributo

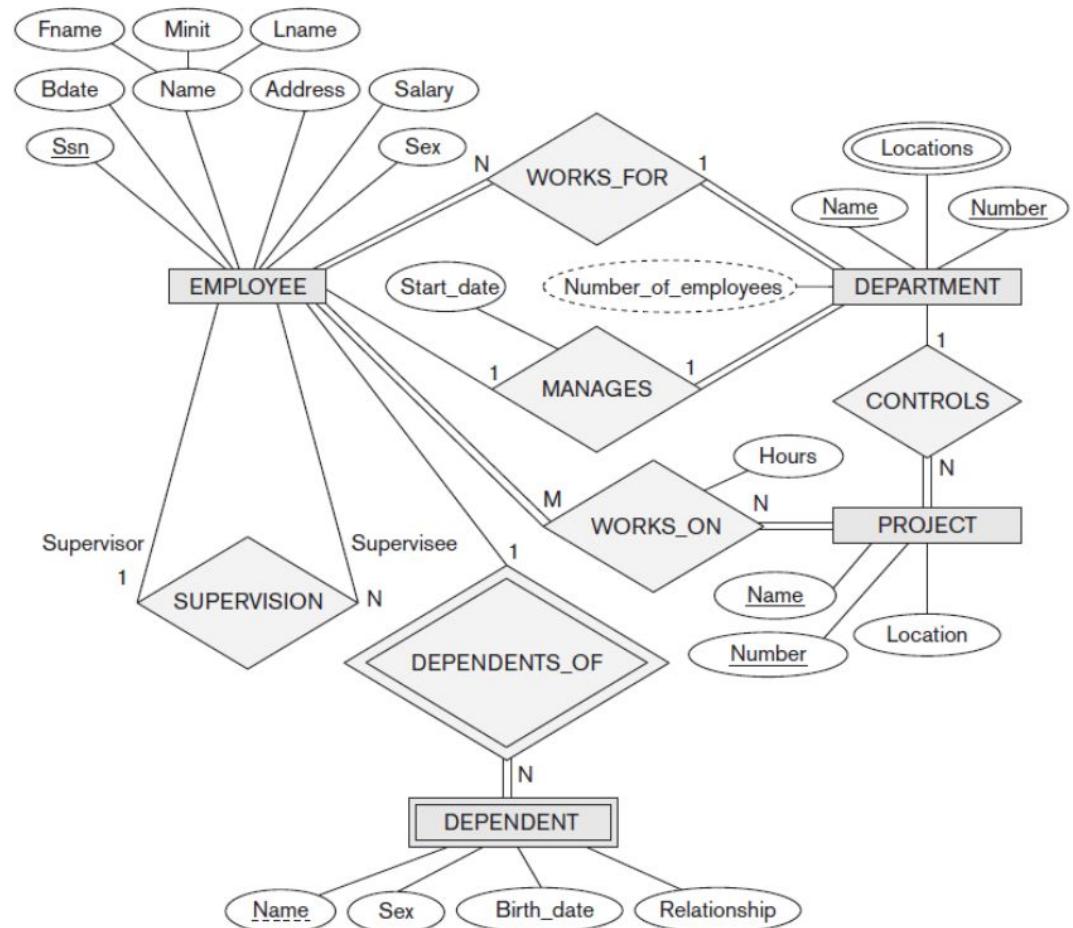
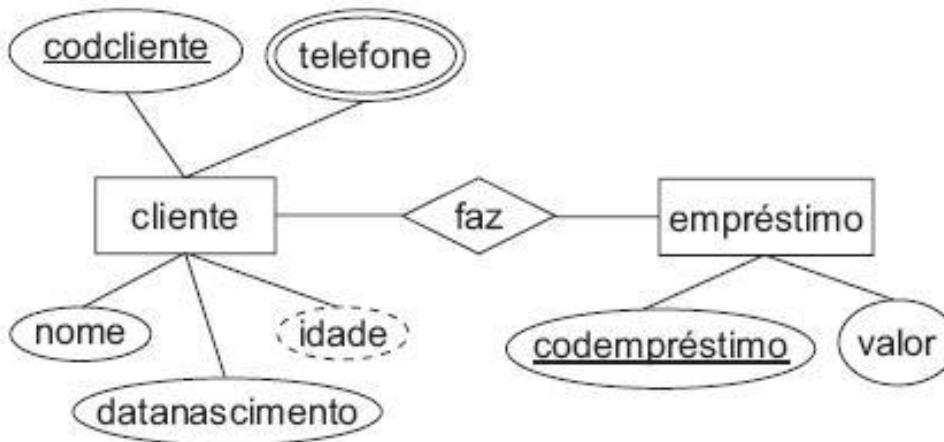


Relacionamento



# MER

## Exemplos



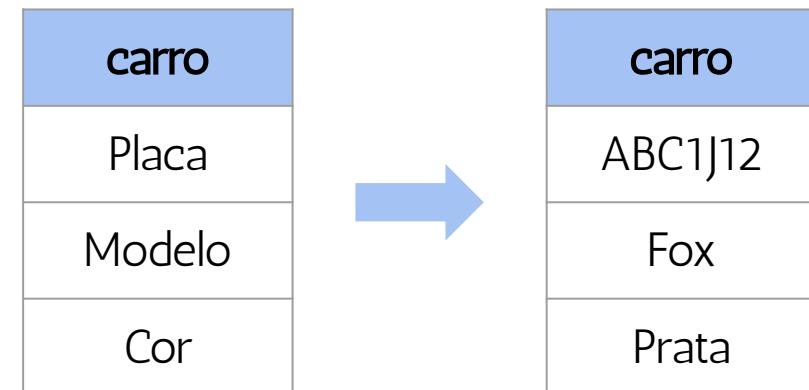
# MER

## Entidades

- Item, *coisa*, do mini-mundo que precisa ser representado em um banco de dados
- Representação de temas, tópicos ou um conceito
- Pode existir de forma física ou conceitual  
Exemplo: Empregado, aluno, livro, produto
- Precisam ser nomeados de forma clara e objetiva, representando sua função (substantivos)
- Instâncias são ocorrências da entidade, são valores que elas podem admitir

### Boas práticas

- Começar com uma letra e estar no singular
- De preferência letras minúsculas
- Sem espaços ou caracteres especiais



# MER

## Entidades

### Tipos de entidades

- Forte: sua existência independe de outras entidades. São entidades que por si só possuem total sentido de existir.

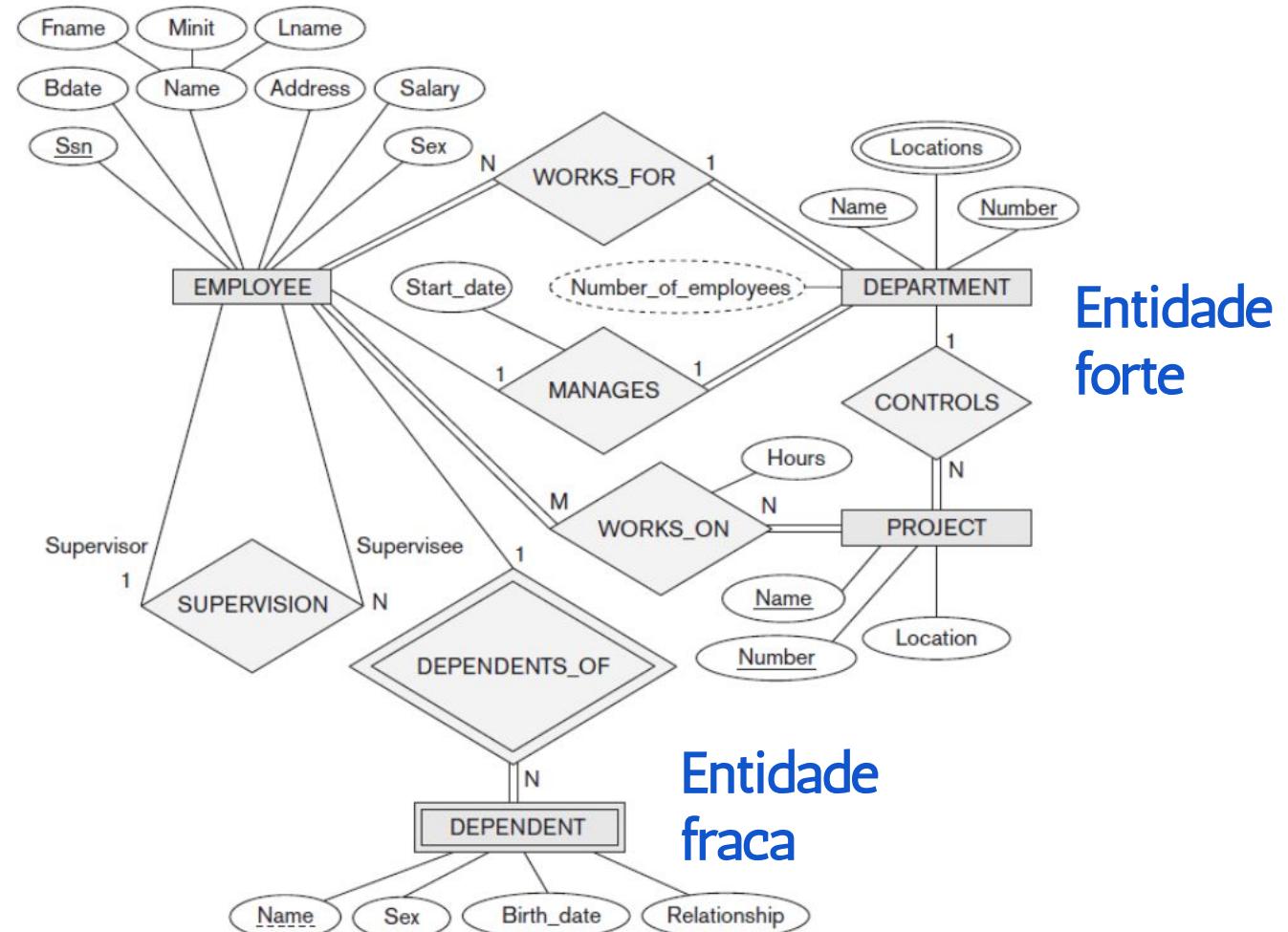


- Fraca (ou dependente): precisa de outra entidade para garantir a sua existência. O identificador de uma entidade fraca possui em sua composição o(s) atributo(s) identificador(es) da entidade forte à qual está associada.



# MER

## Entidades



# MER

## Identificando entidades



# MER

## Atributos

- Descrevem **características** de determinada entidade ou relação  
Cor, nome, modelo, telefone
- Possuem um nome, que identifica o que o atributo representa, um tipo de dados, ou domínio, e um valor  
Exemplo: Nome, cadeia de caracteres, João

### Tipos de atributos

- Simples: atômico, indivisível. Ex.: nome
- Composto: pode ser subdividido em outros atributos. Ex.: endereço
- Multivalorado: vários valores para um mesmo registro. Ex.: idioma
- Determinante: define de forma única a instância de uma entidade. Ex.: matrícula
- Derivado: atributo que tem relação com outro. Ex. idade (nascimento)
- Chaves: identifica uma instância da entidade. Ex.: CPF



# MER

Composto

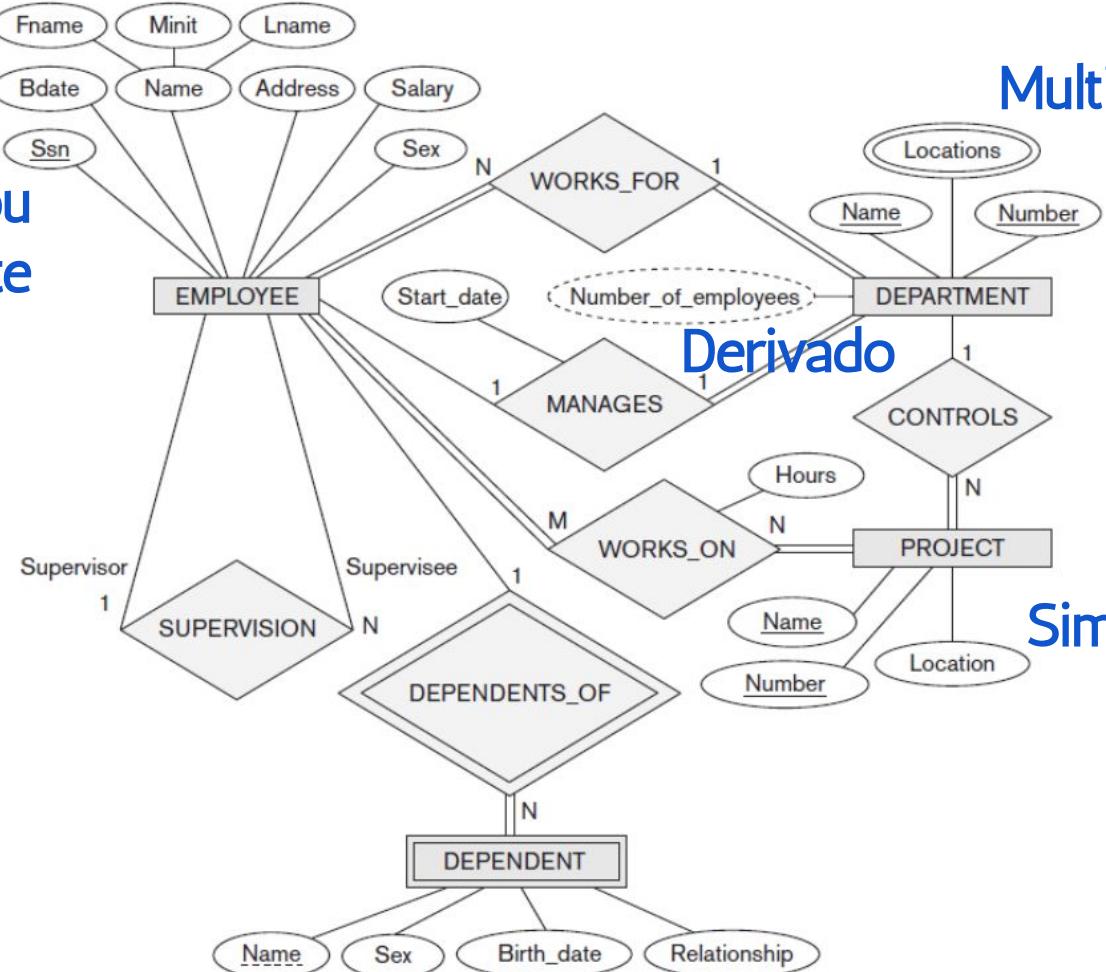
Atributos

Chave ou  
determinante

Multivalorado

Derivado

Simples



# MER

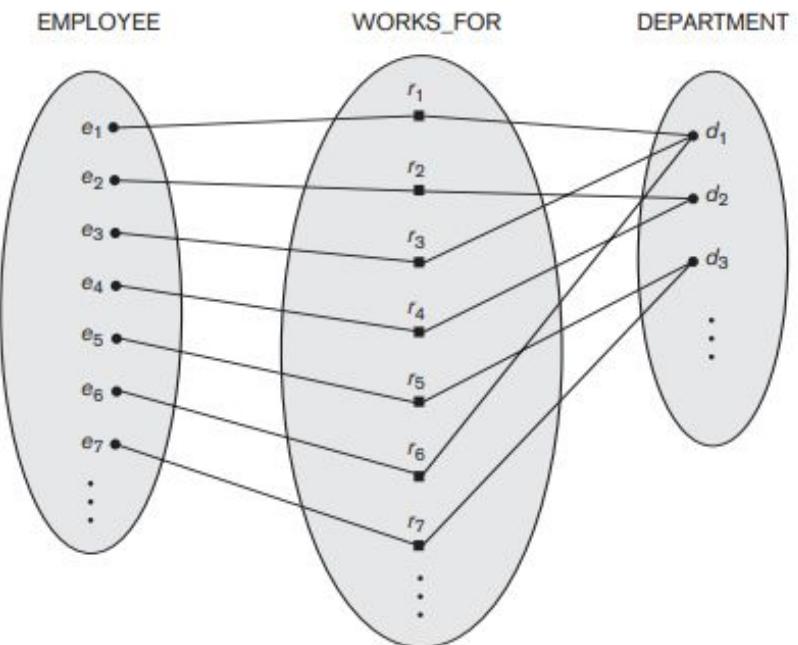
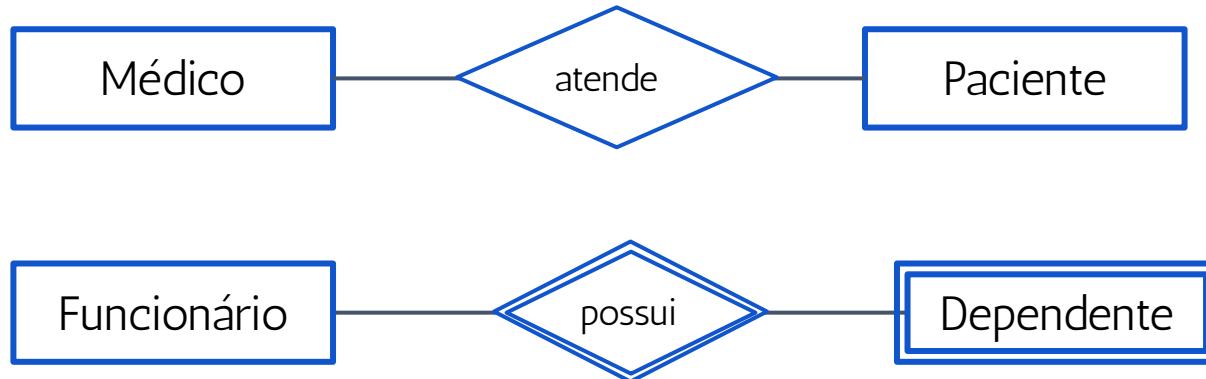
## Identificando atributos



# MER

## Relacionamentos

- São utilizados para indicar a associação entre as entidades
- Essa associação é feita entre **uma** ou mais entidades
- Entidades modelam itens de forma separada, e por isso os relacionamentos nos permitem combinar informações de diferentes



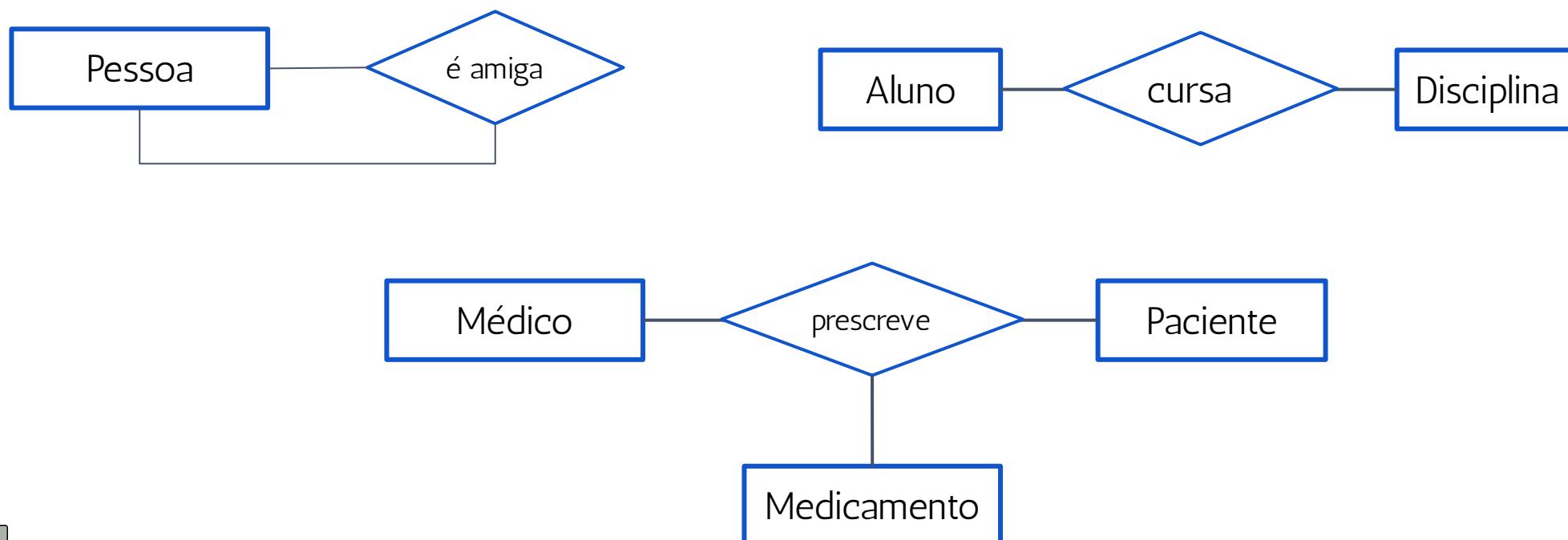
# MER

## Relacionamentos

### Grau de um relacionamento

Número de entidades que participam de um relacionamento

Unário, binário, ternário...



# MER

## Relacionamentos

### Cardinalidade

Indica o número mínimo/máximo de valores que o atributo pode assumir em cada instância da entidade ou relacionamento.

Ou seja, ajuda a definir o relacionamento, pois define o número de ocorrências possíveis em um relacionamento.

### Tipos de cardinalidade

- Mínima: número mínimo de instâncias de entidade que devem participar de um relacionamento  
0 - opcional ou 1 - obrigatório
- Máxima: número máximo de instâncias de entidade que podem participar de um relacionamento  
1 ou N - muitos



# MER

## Relacionamentos

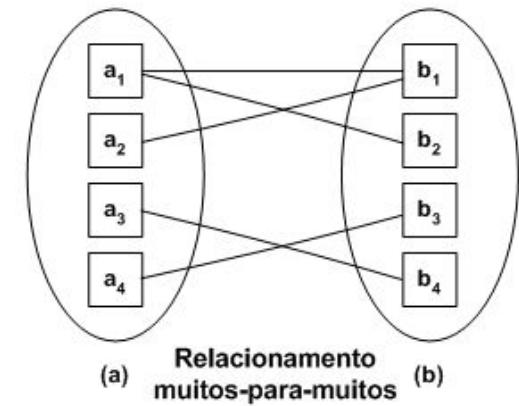
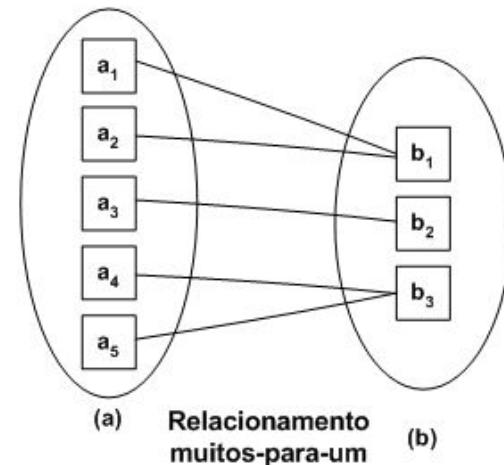
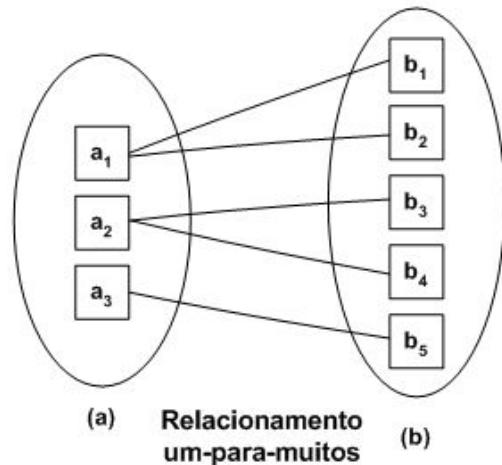
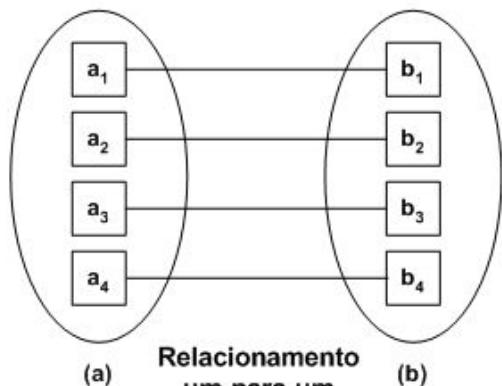
### Tipos de relacionamento

- **Relacionamento 1:1 (um para um)**: cada uma das duas entidades envolvidas referenciam obrigatoriamente **apenas uma instância** da outra.  
Usuário - Currículo
- **Relacionamento 1:n ou 1:\*** (**um para muitos**): uma das entidades envolvidas pode referenciar **várias instâncias** da outra, porém, do outro lado cada uma das várias unidades referenciadas só pode estar ligada **uma instância** da outra entidade.  
Usuário - Dependente
- **Relacionamento n..n ou \*..\*** (**muitos para muitos**): cada entidade, de ambos os lados, podem referenciar **múltiplas instâncias** da outra.  
Livro - Autor



# MER

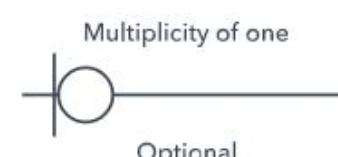
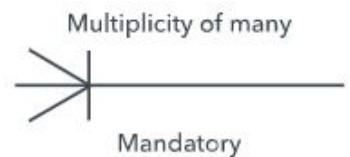
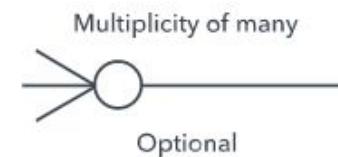
## Relacionamentos



# MER

## Relacionamentos

### Tipos de notação comuns



# MER

## Relacionamentos

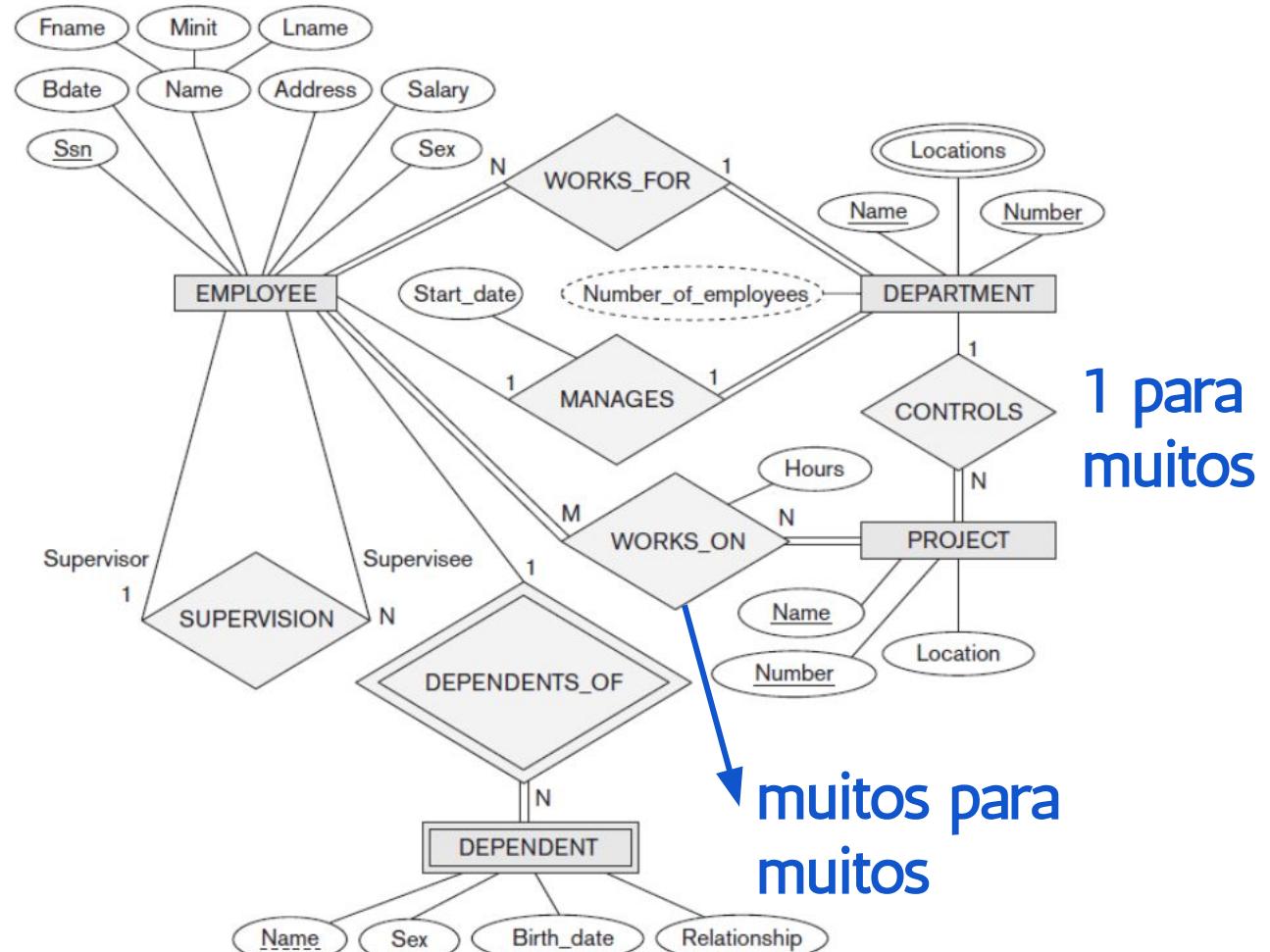
### Detalhe sobre relacionamentos N:M

Relacionamentos N:M vão sempre resultar na criação de uma nova entidade. O relacionamento N:M deixa de existir e passamos a ter dois relacionamentos 1:N.



# MER

## Relacionamentos



# MER

## Resumo

Chen's Notation		
Meaning	Notation	Alternative notation
Entity	Name	Name
Weak Entity	Name	Name
Attribute (mandatory)	Name	Name ○ —
Primary identifier attribute	Name	Name ● —
Partial identifier attribute	Name	✗
Alternate identifier attribute	✗	Name ● —
Multi-valued attribute	Name	Name ○ —
Derived attribute	Name ○ —	Name ○ — Name ○ — Tag
Composite attribute	Name ○ — Name ○ — Name	Name ○ — Name ○ — Name ○ — Name
Optional attribute	✗	Name ○ —

(cc) BY-NC-SA

Outros tipos de notações e mais detalhes::



[Don't get wrong! Explained guide to choosing a database design notation for ERD in a while](#)



# MER

## Exercício

- Uma biblioteca deseja manter informações sobre seus livros.
- Inicialmente, quer armazenar para os livros as seguintes características: ISBN, título, ano editora e autores deste livro.
- Para os autores, deseja manter: nome e nacionalidade.
- Cabe salientar que um autor pode ter vários livros, assim como um livro pode ser escrito por vários autores.
- Cada livro da biblioteca pertence a uma categoria.
- A biblioteca deseja manter um cadastro de todas as categorias existentes, com informações como: código da categoria e descrição.
- Uma categoria pode ter vários livros associados a ela.



# MER

## Exercício

**Entidades:** Peças, Depósitos, Fornecedor, Projeto, Funcionário e Departamento.

### Requisitos:

- Cada Funcionário pode estar alocado a somente um Departamento;
- Cada Funcionário pode pertencer a mais de um Projeto;
- Um Projeto pode utilizar-se de vários Fornecedores e de várias Pecas;
- Uma Peca pode ser fornecida por vários Fornecedores e atender a vários Projetos;
- Um Fornecedor pode atender a vários Projetos e fornecer várias Pecas;
- Um Depósito pode conter várias Pecas;
- Deseja-se ter um controle do material utilizado por cada Projeto, identificando inclusive o seu Fornecedor. Gravar as informações de data de Início e Horas Trabalhadas no Projeto.

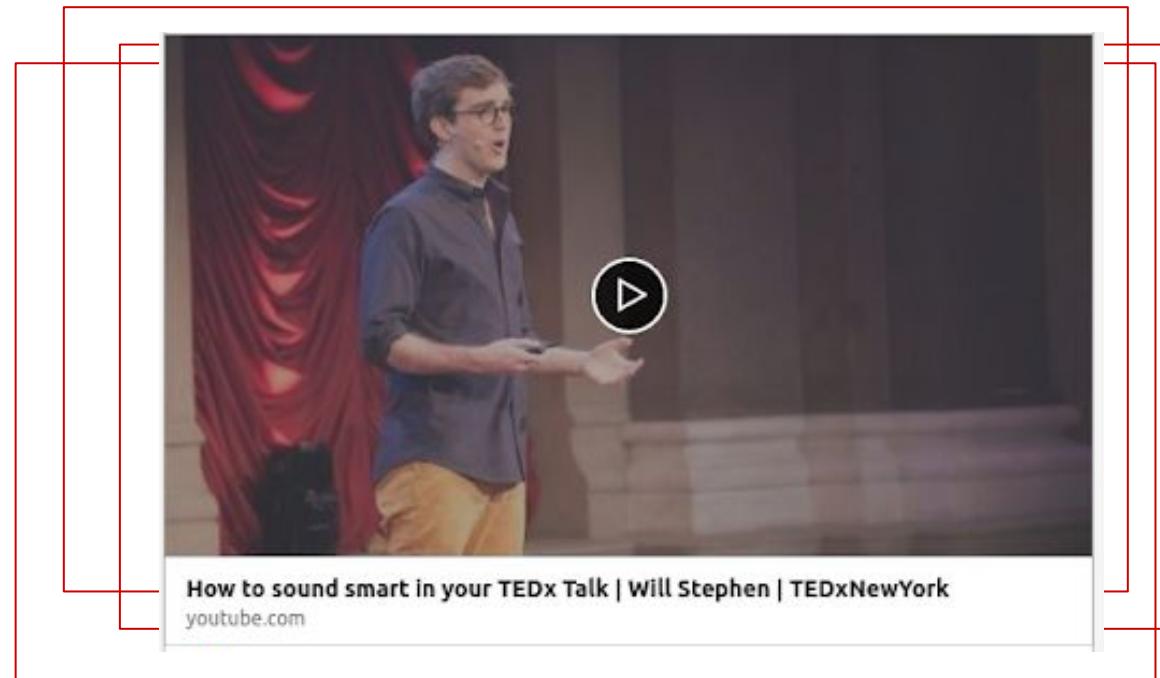
### Atributos:

- Peças: Número, Peso e Cor;
- Depósito: Número e Endereço;
- Fornecedor: Número e Endereço;
- Projeto: Número e Orçamento;
- Funcionário: Número, Salário e Telefone;
- Departamento: Número e Setor.

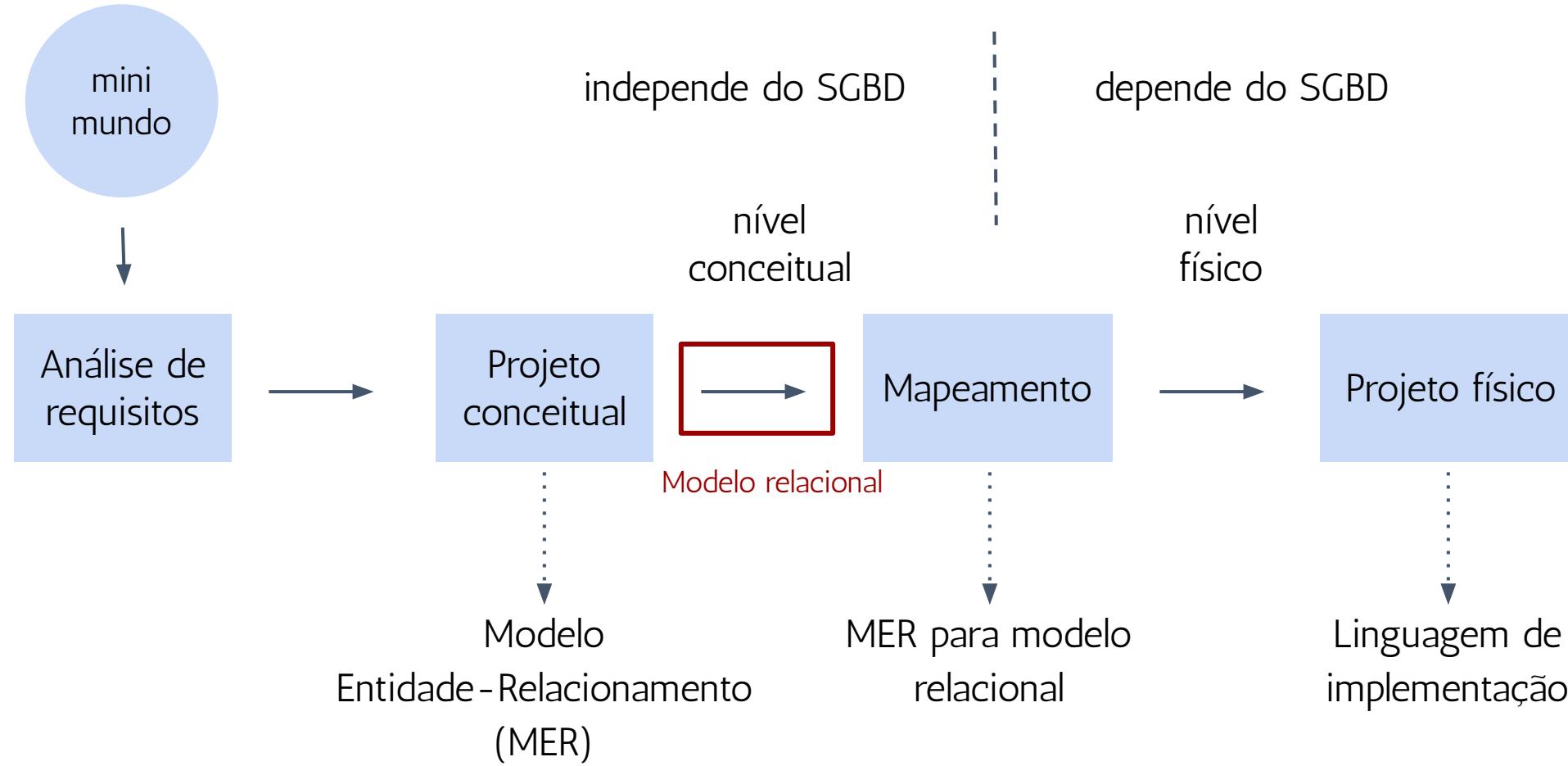


# Para assistir

uma outra hora ;)



# Projeto de Banco de Dados



# Modelo Relacional

## Modelo Relacional

- Relembrando o histórico: surgiu em 1970 pelo pesquisador da IBM, E. Codd
- Dados são organizados em tabelas - relações
- Baseado em lógica e teoria de conjuntos

## Componentes

- Tabela: estrutura básica que armazena os dados sobre algo do mini-mundo
- Tupla: linha/registro que define a ocorrência de uma entidade
- Colunas: armazena um tipo de dado na tabela - pode ser nulo
  - Chave primária (PK): identifica um registro de forma única
- Relacionamento: associação entre as tabelas
  - Chave estrangeira (FK): define como as tabelas se relacionam de fato  
Uma FK faz referência a uma PK de outra tabela



# Modelo Relacional

Tuplas

Atributos

Nome	Endereço	Telefone
José de Almeida	Rua do Imperador, Centro, 100, Petrópolis	2222 - 2222
João da Silva	Av. Getúlio Vargas, Quitandinha, 300, Petrópolis	2222 - 3333
Mariana Ferreira	Rua União Indústria, Corrêas, 100, Petrópolis	2222 - 4444

Domínio



# Modelo Relacional

## Restrições de integridade

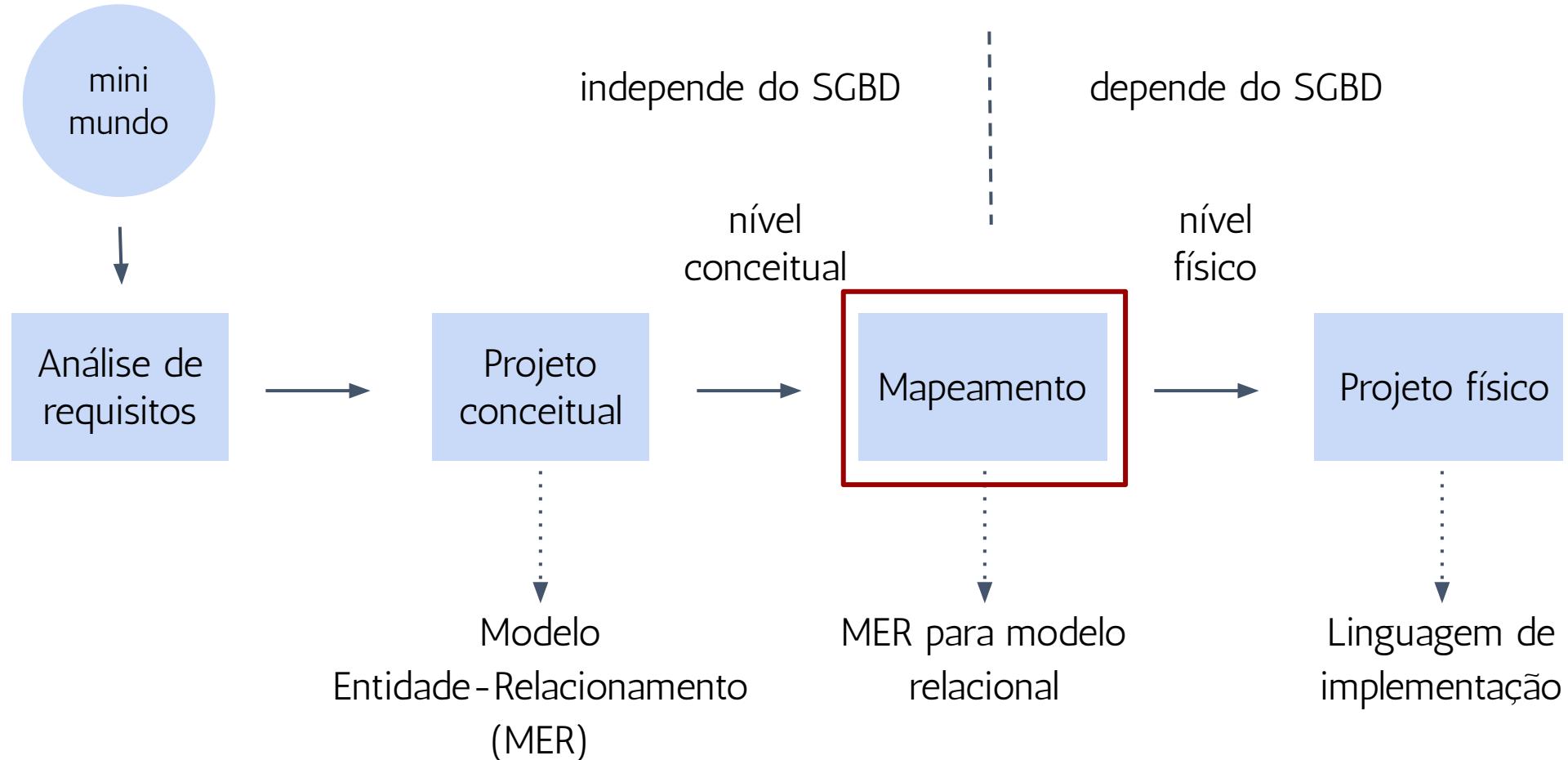
São condições para garantir a consistência dos dados em um banco de dados

Garantem que os dados representem o modelo/regras de negócio

- Restrição de chave: Impede que uma chave primária se repita. A chave primária diferencia de forma única os registros (linhas) de uma relação (tabela)
- Restrição de domínio: Define o conjunto de valores possíveis ou permitidos que um campo pode ter.
- Integridade de vazios: Verifica se um campo pode ou não receber valor nulo (NULL)
- Integridade referencial: Uma chave estrangeira de uma relação tem que coincidir com uma chave primária da tabela origem a que a chave estrangeira se refere
- Integridade definida pelo usuário: Permite definir regras que não se encaixam em outras categorias de integridade



# Projeto de Banco de Dados



# Mapeamento MER - Relacional

## Regras básicas para o mapeamento de entidades

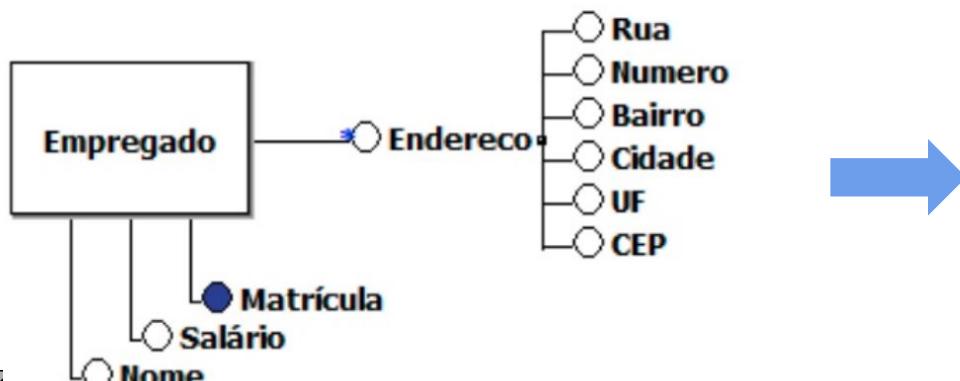
- Toda entidade vira uma relação
- Atributo identificador se torna chave primária (PK) na relação
- Atributos simples se tornam colunas (campos)
- Atributos compostos se tornam atributos simples, mapeados em colunas, uma coluna para cada atributo
- Atributos derivados não são mapeados
- Atributos multi-valorados podem ser mapeados de duas formas
  - Como n colunas, onde n é o número máximo de valores do atributo
  - Criando-se uma nova relação



# Mapeamento MER - Relacional

## Regra 1: Mapeamento de entidades fortes

- Cada entidade forte deve ser transformada em uma relação
- Atributos simples devem ser incluídos na relação
- Apenas componentes simples dos atributos compostos devem ser incluídos na relação
- Um dos atributos chaves da entidade deve ser escolhido como chave-primária



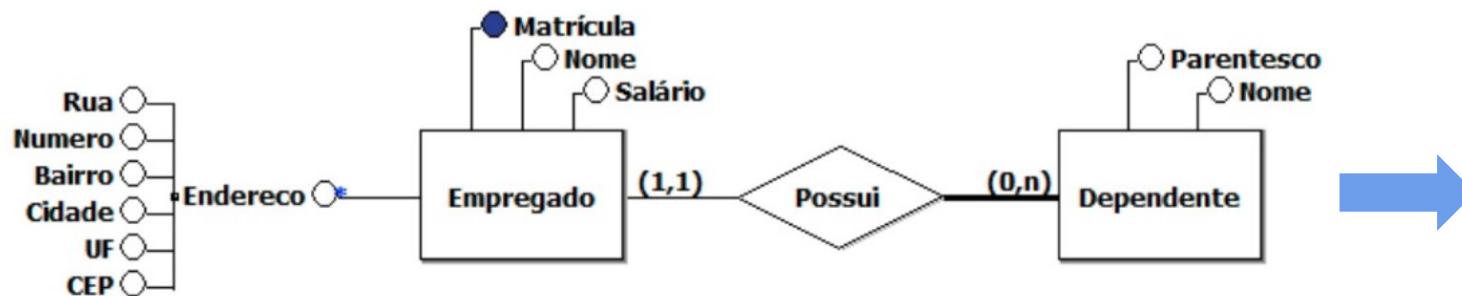
Empregado (matrícula, nome, salário, rua, número, bairro, cidade, uf, cep)



# Mapeamento MER - Relacional

## Regra 2: Mapeamento de entidades fracas

- Cada entidade fraca deve ser transformada em uma relação (= Regra 1)
- Incluir os atributos da chave primária da relação principal como chave estrangeira da relação
- Chave primária: FK da entidade principal e PK da entidade fraca



Empregado (matrícula, nome, salário, rua, número, bairro, cidade, uf, cep)

Dependente (matrícula\_empregado, nome, parentesco)

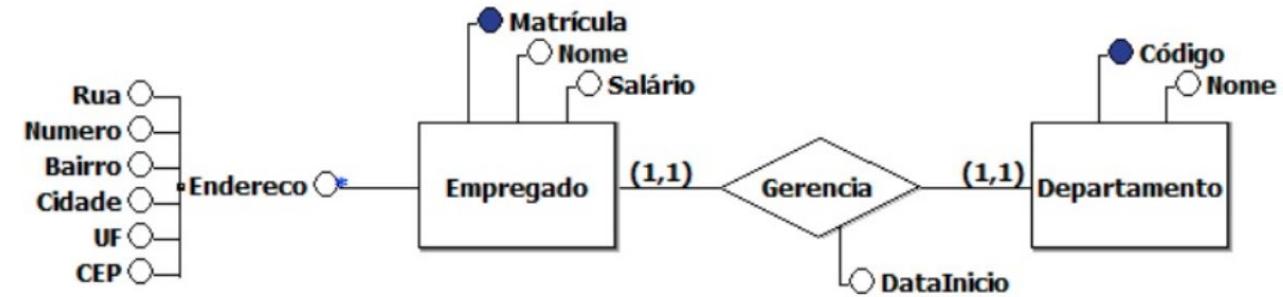


# Mapeamento MER - Relacional

## Regra 3: Mapeamento de relacionamentos 1:1

Existem 3 opções:

- Escolha da chave estrangeira
- Relacionamento incorporado
- Relação de relacionamento

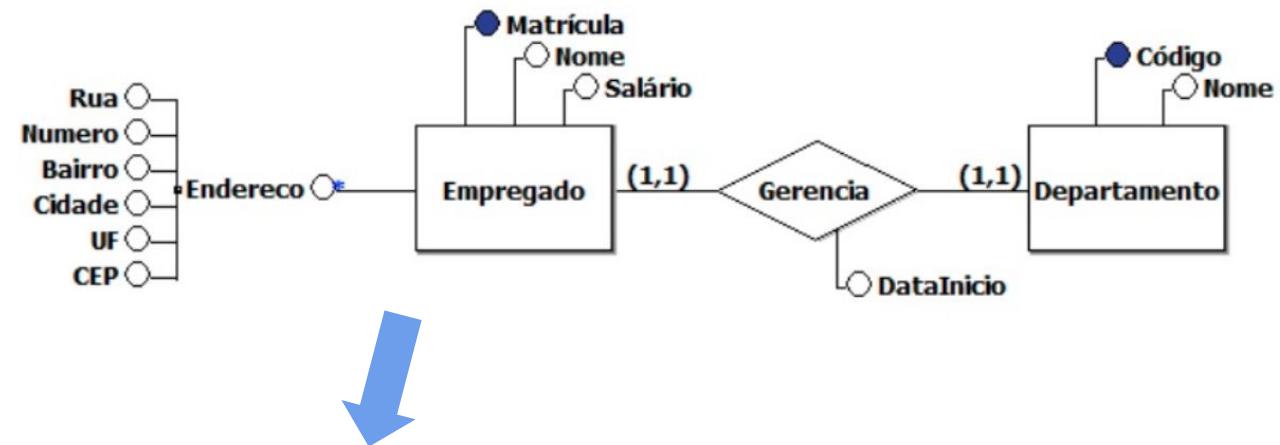


# Mapeamento MER - Relacional

## Regra 3: Mapeamento de relacionamentos 1:1

Existem 3 opções:

- **Escolha da chave estrangeira**
- Relacionamento incorporado
- Relação de relacionamento



Empregado (matrícula, nome, salário, rua, número, bairro, cidade, uf, cep)

Departamento (código, nome, gerente, data\_inicio)

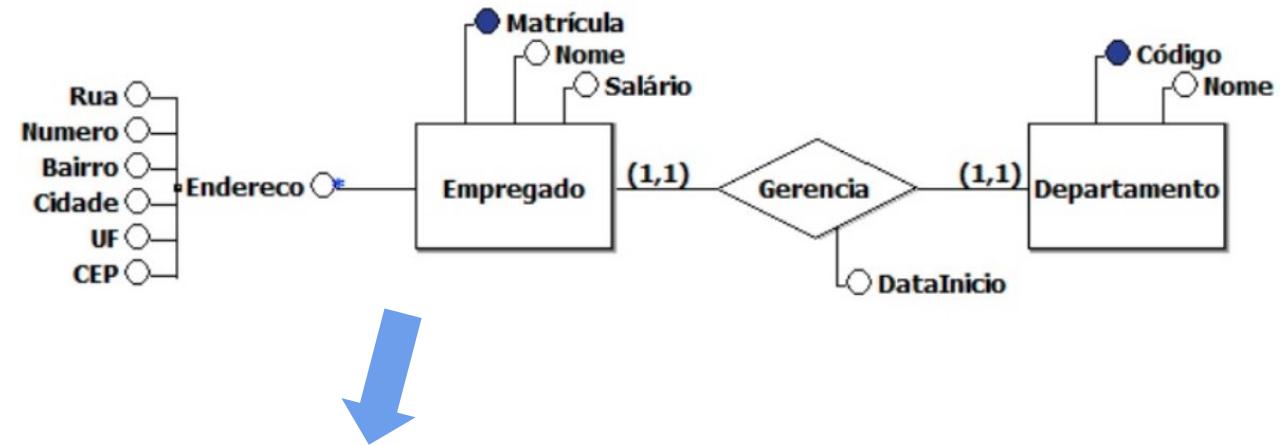


# Mapeamento MER - Relacional

## Regra 3: Mapeamento de relacionamentos 1:1

Existem 3 opções:

- Escolha da chave estrangeira
- **Relacionamento incorporado**
- Relação de relacionamento



Empregado (matrícula, nome, salário, rua, número, bairro, cidade, uf, cep,  
nome\_departamento, data\_inicio, gerente)

\* gerente poderia ser um valor booleano (verdadeiro ou falso)

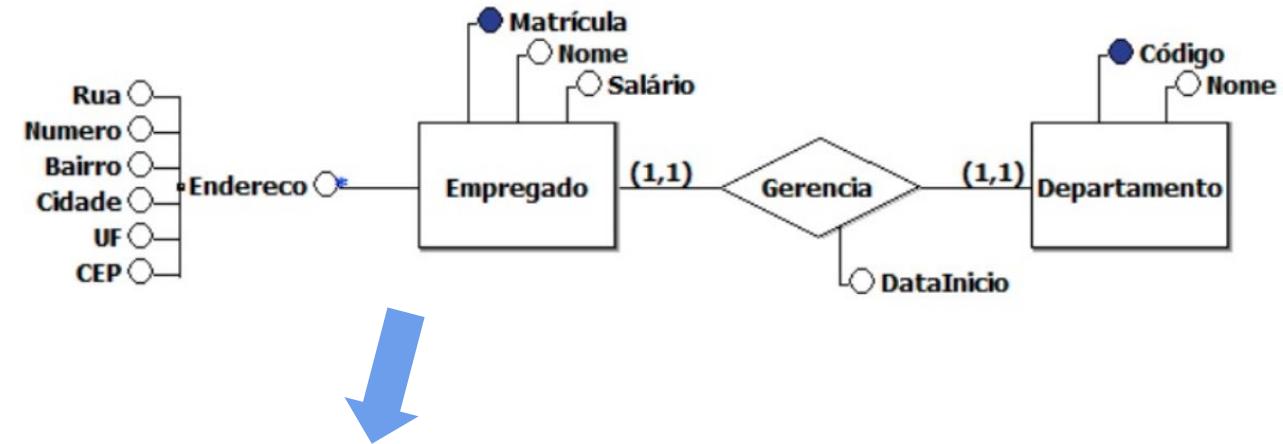


# Mapeamento MER - Relacional

## Regra 3: Mapeamento de relacionamentos 1:1

Existem 3 opções:

- Escolha da chave estrangeira
- Relacionamento incorporado
- **Relação de relacionamento**



Empregado (matrícula, nome, salário, rua, número, bairro, cidade, uf, cep)

Gerência (matricula\_empregado, codigo\_departamento)

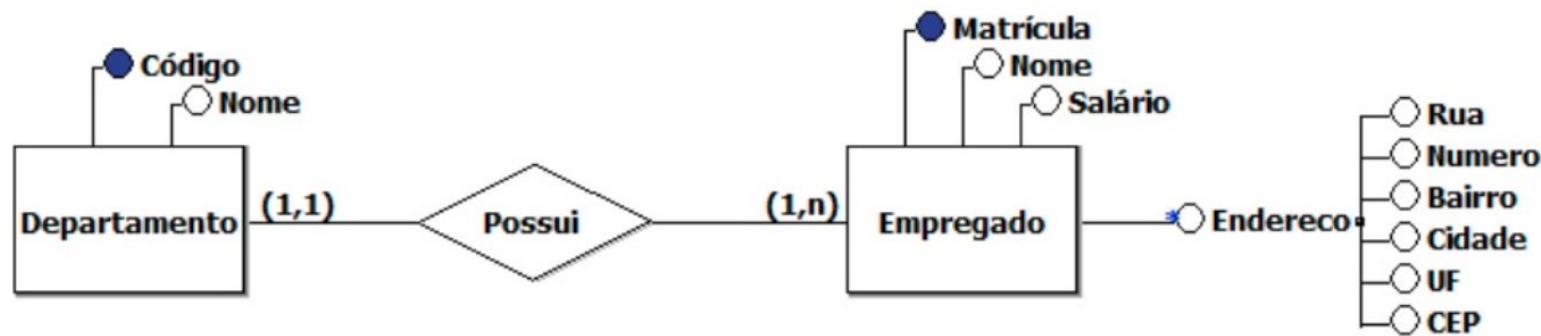
Departamento (código, nome, data\_inicio)



# Mapeamento MER - Relacional

## Regra 4: Mapeamento de relacionamentos 1:N

- Incluir a chave primária da relação do lado 1 como chave estrangeira da relação do lado N



Empregado (matrícula, nome, salário, rua, número, bairro, cidade, uf, cep,  
*codigo\_departamento*)

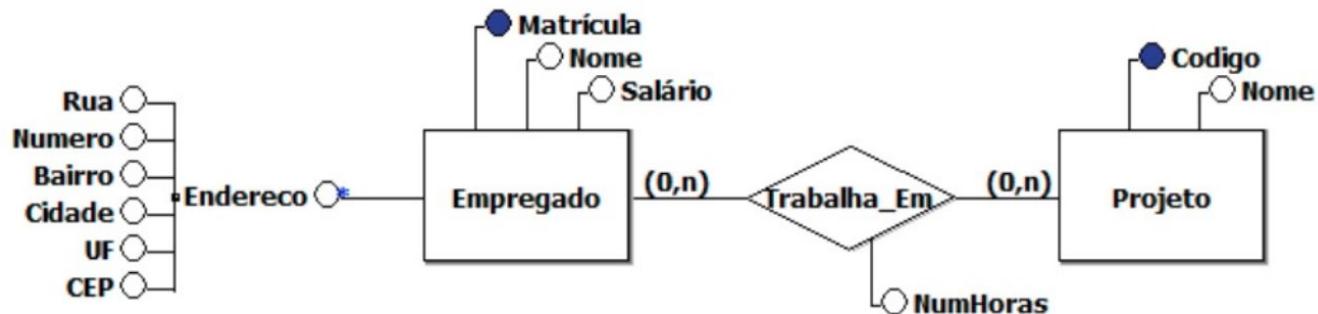
Departamento (código, nome)



# Mapeamento MER - Relacional

## Regra 5: Mapeamento de relacionamentos N:M

- Criar nova relação para o relacionamento
- Incluir chaves primárias das duas entidades que participam do relacionamento na nova relação
- Incluir na nova relação os atributos do relacionamento



Empregado (matrícula, nome, salário, rua, número, bairro, cidade, uf, cep)

Projeto (código, nome)

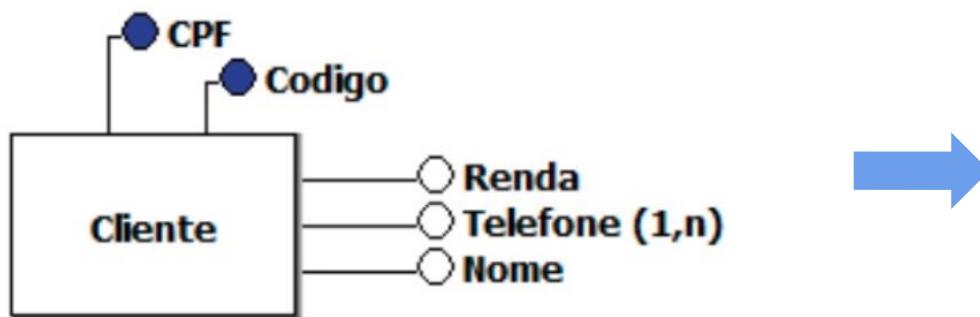
Trabalha (código, matrícula, num\_horas)



# Mapeamento MER - Relacional

## Regra 6: Mapeamento de atributos multivalorados

- Criar nova relação para o atributo multivalorado
- Incluir na relação o atributo multivalorado
- A chave primária da relação original será chave estrangeira na nova relação



Cliente (codigo, nome, cpf, renda)

Telefone\_cliente (codigo\_cliente, telefone)

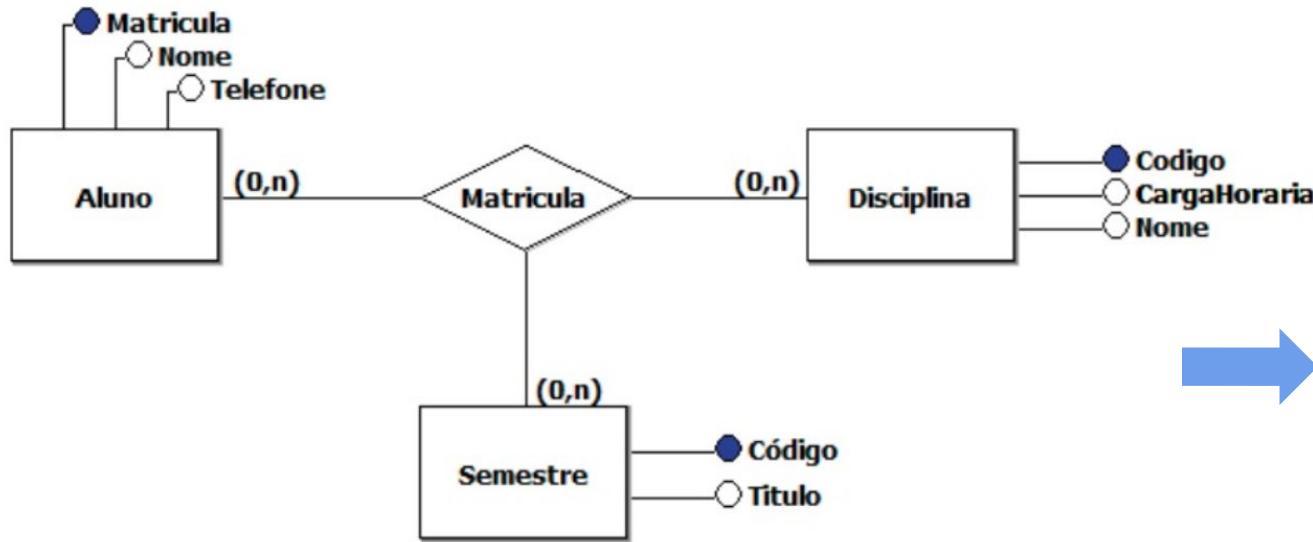
\*não é tão comum implementarmos dessa forma  
vamos ver na parte de normalização



# Mapeamento MER - Relacional

## Regra 7: Mapeamento de relacionamentos n-ários

- Criar nova relação para representar o relacionamento
- Incluir na relação as chaves primárias das relações que participam do relacionamento
- Incluir os atributos do relacionamento na nova relação
- A chave primária da relação original será chave estrangeira na nova relação



Aluno (matrícula, nome, telefone)  
 Disciplina (código, nome, carga\_horária)  
 Semestre (código, título)  
 Matrícula (matrícula, código\_disc,  
doc\_semestre)



# Mapeamento MER - Relacional

## Resumindo

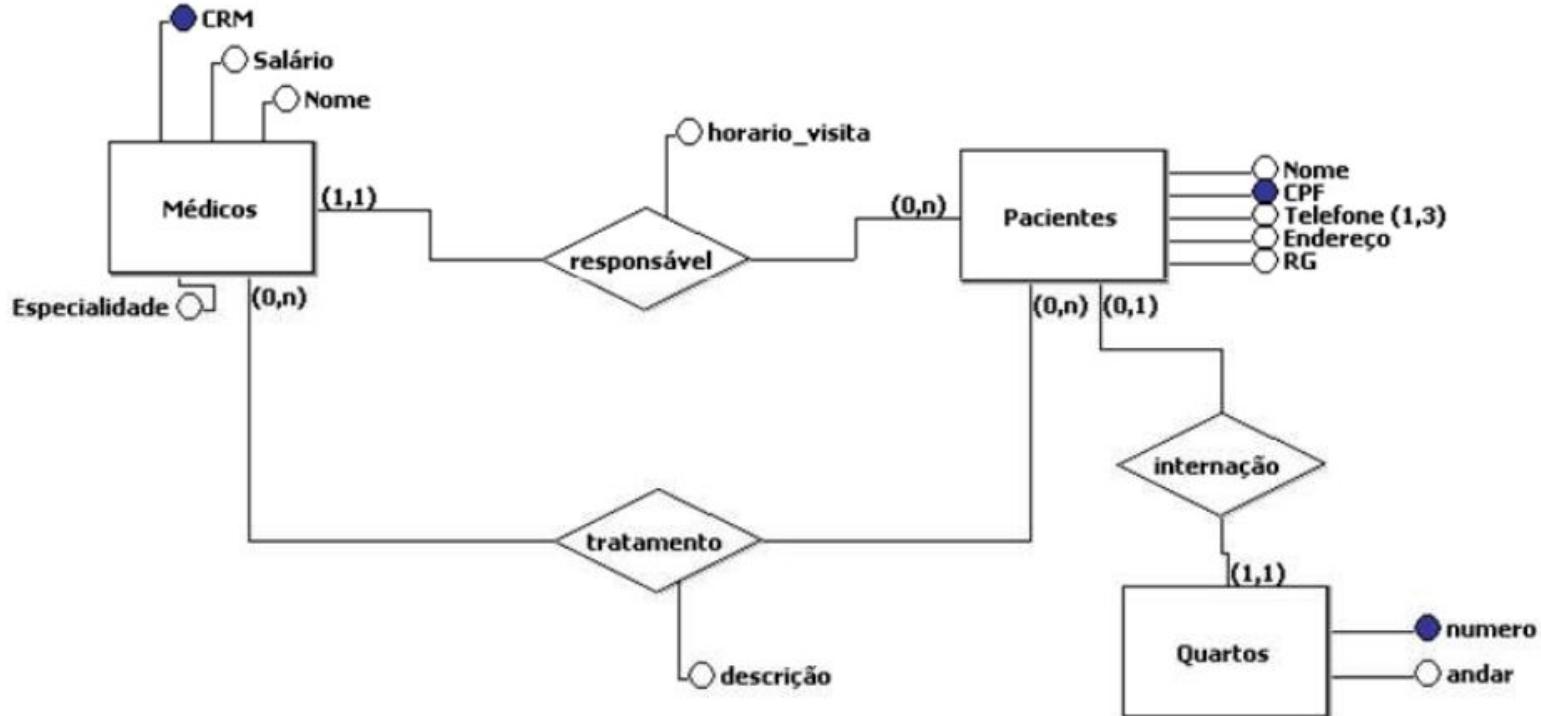
1. Mapeamento de entidades fortes
2. Mapeamento de entidades fracas
3. Mapeamento de relacionamentos 1:1
4. Mapeamento de relacionamentos 1:N
5. Mapeamento de relacionamentos N:M
6. Mapeamento de atributos multivalorados
7. Mapeamento de relacionamentos n-ários



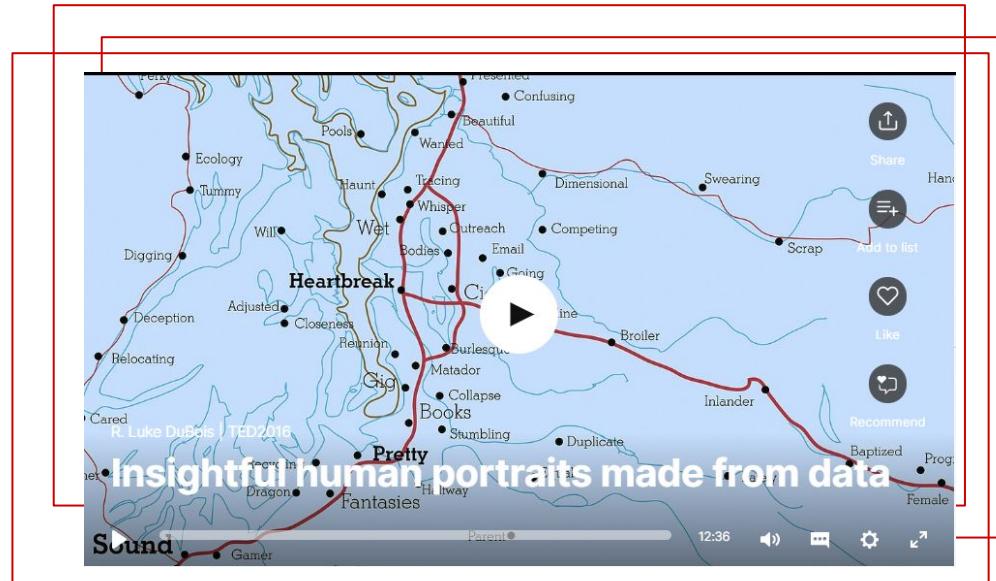
# Mapeamento MER - Relacional

## Exercício

Fazer o mapeamento para o modelo relacional



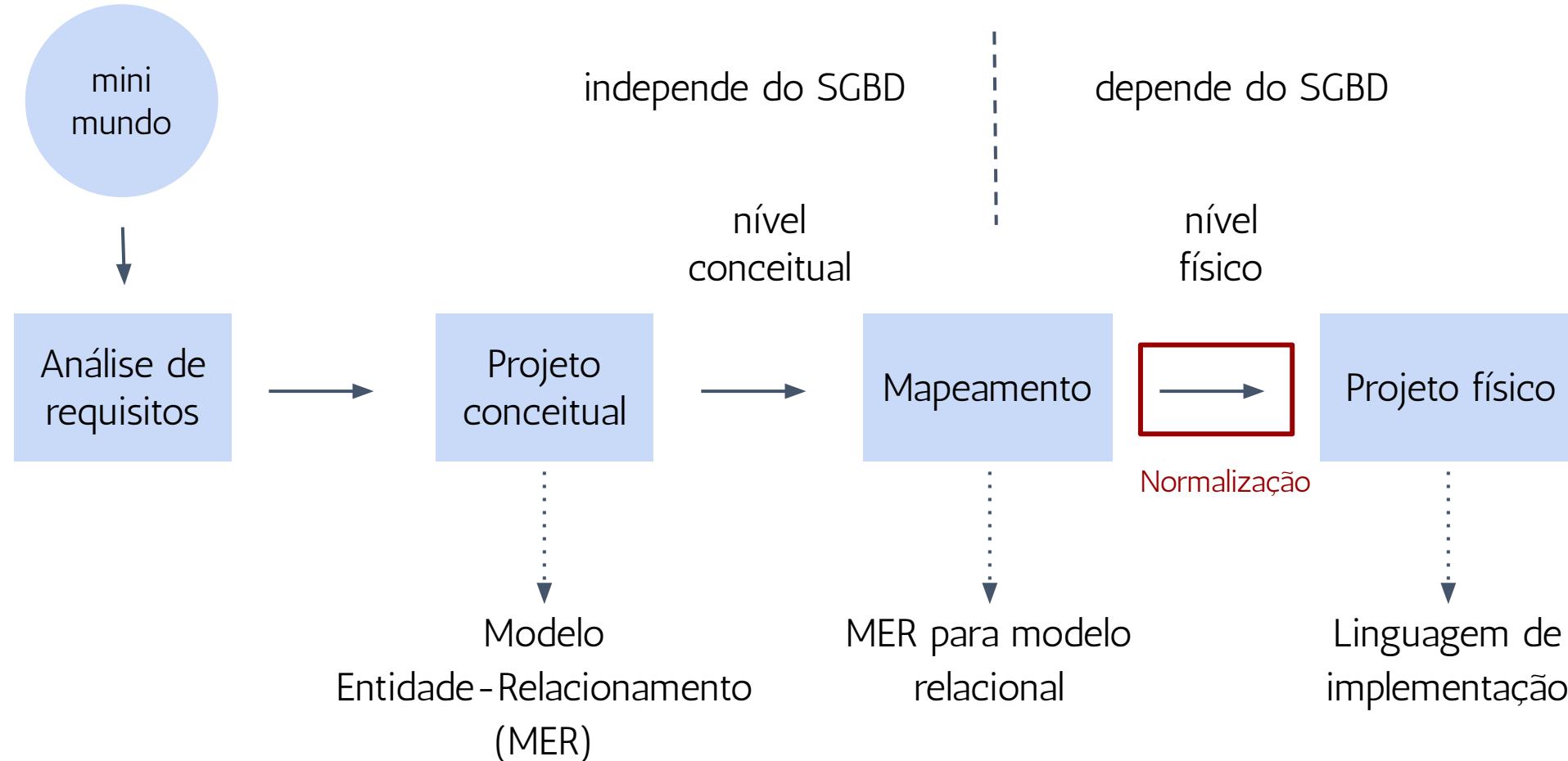
# Para assistir uma outra hora ;)



**TED**  
Ideas worth spreading

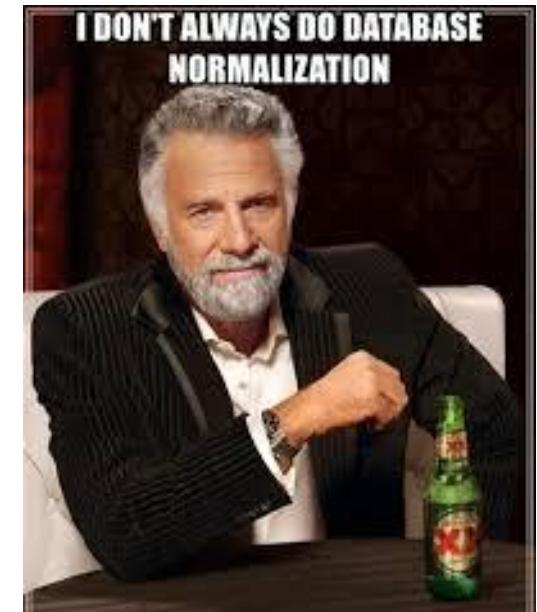


# Projeto de Banco de Dados



# Normalização de dados

- É um processo que aplica um **conjunto de regras** sobre o modelo do banco de dados relacional
  - Verificar se o modelo está corretamente projetado
  - Permitir o armazenamento consistente
  - Permitir eficiente acesso aos dados
- **Evita** problemas como
  - Inconsistência nos dados
  - Redundância
  - Falta de integridade
- As tabelas que atendem a um determinado conjunto de regras estão em uma determinada forma normal



# Normalização de dados

Vamos considerar o seguinte cenário

Placa	Marca	Modelo	Ano de fabricação
DKL4598	Wolkswagen	Gol	2006
DAE6534	Ford	Fiesta	2001
JDM8776	Wolkswagen	Santana	1999
DMZ1122	Wolkswagen	Gol	2009
JJM3692	Ford	Fiesta	2014
DMN1012	Ford	Gol	2007

Se a tabela estivesse exatamente assim, quais são os problemas que podem acontecer?



# Normalização de dados

Vamos considerar o seguinte cenário

Membro			Tarefa			
ID	Nome	Função	ID	Descrição	Início	Horas alocadas
1243	Andréa	Gerente	1700	Planejamento	10/01/2018	40
			1710	Projeto	10/04/2018	100
7868	Vinícius	Analista	1700	Detalhamento	10/06/2018	20
9868	Mariana	Programador	1720	Implementação	18/05/2018	60
4548	Ana	Programador	1750	Implementação	14/02/2018	50

Se a tabela estivesse exatamente assim, quais são os problemas que podem acontecer?



# Normalização de dados

## 1<sup>a</sup> Forma Normal

Uma relação está na 1FN quando todos os seus atributos são **atômicos** e **monovalorados**. Para isso:

1. Identificar a chave primária da entidade
2. Identificar o grupo repetitivo e removê-lo da entidade
3. Criar uma nova tabela secundária com chave primária para armazenamento do dado repetido
4. Relacionar a tabela principal e a tabela secundária

**DEPARTMENT**

Dname	<u>Dnumber</u>	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}



# Normalização de dados

## 1<sup>a</sup> Forma Normal

Devemos **sempre** aplicar a 1FN?

- Quando a quantidade de possíveis valores é pequena e conhecida, *nem sempre*:

Pessoa

Matrícula	Nome	Telefones



Pessoa

Matrícula	Nome	Telefone fixo	Celular

- Quando a quantidade de possíveis valores é muito grande, variável ou desconhecida, *sim*:

Pessoa

Matrícula	Nome	Telefones



Pessoa

Matrícula	Nome

Contato

idTelefone	Matrícula	Telefone



# Normalização de dados

## 2<sup>a</sup> Forma Normal

Uma relação está na 2FN se, e somente se, **estiver na 1FN** e cada **atributo não-chave for dependente da chave primária** inteira, isto é, cada atributo não-chave não poderá ser dependente de apenas parte da chave.

1. Identificar colunas que não são funcionalmente dependentes da chave primária da tabela
2. Remover a coluna da tabela e criar uma nova tabela com esses dados

**Pedidos**

<u>id_pedido</u>	data	<u>id_produto</u>	nome	qtd	valor	total
1	20/05/2016	1548	Mesa	2	100	200
2	27/05/2016	1847	Almofada	1	50	50
3	31/05/2016	1698	Cadeira	3	75	225



# Normalização de dados

## 2<sup>a</sup> Forma Normal

**Pedido**

<u>id_pedido</u>	data	<u>id_produto</u>	nome	qtd	valor	total
1	20/05/2016	1548	Mesa	2	100	200
2	27/05/2016	1847	Almofada	1	50	50
3	31/05/2016	1698	Cadeira	3	75	225



**Pedido**

<u>id_pedido</u>	data	<u>id_produto</u>	qtd	valor	total
1	20/05/2016	1548	2	100	200
2	27/05/2016	1847	1	50	50
3	31/05/2016	1698	3	75	225

**Produto**

<u>id_produto</u>	nome
1548	Mesa
1847	Almofada
1698	Cadeira

# Normalização de dados

## 3<sup>a</sup> Forma Normal

Uma relação está na 3FN se, e somente se, **ela estiver na 2FN** e cada atributo não-chave da relação **não possuir dependência transitiva**, para cada chave candidata existente.

1. Identificar as colunas que são funcionalmente dependentes de outras colunas não-chave na tabela
2. Remover essas colunas

Pedido

<u>id_pedido</u>	data	<u>id_produto</u>	qtd	valor	total
1	20/05/2016	1548	2	100	200
2	27/05/2016	1847	1	50	50
3	31/05/2016	1698	3	75	225



# Normalização de dados

## 3<sup>a</sup> Forma Normal

**Pedido**

<u>id_pedido</u>	data	id_produto	qtd	valor
1	20/05/2016	1548	2	100
2	27/05/2016	1847	1	50
3	31/05/2016	1698	3	75

**Produto**

<u>id_produto</u>	nome
1548	Mesa
1847	Almofada
1698	Cadeira



# Normalização de dados

## Resumindo

- 1<sup>a</sup> FN  
Quando a tabela só possui atributos **atômicos** (não multivalorados)
- 2<sup>a</sup> FN  
Quando estiver na 1FN e todo **atributo não-chave for dependente da chave primária**
- 3<sup>a</sup> FN  
Quando estiver na 2FN e **não tiver** atributos **não-chave dependendo** de atributos **não-chave**



# Normalização de dados

## Exemplo - Spotify

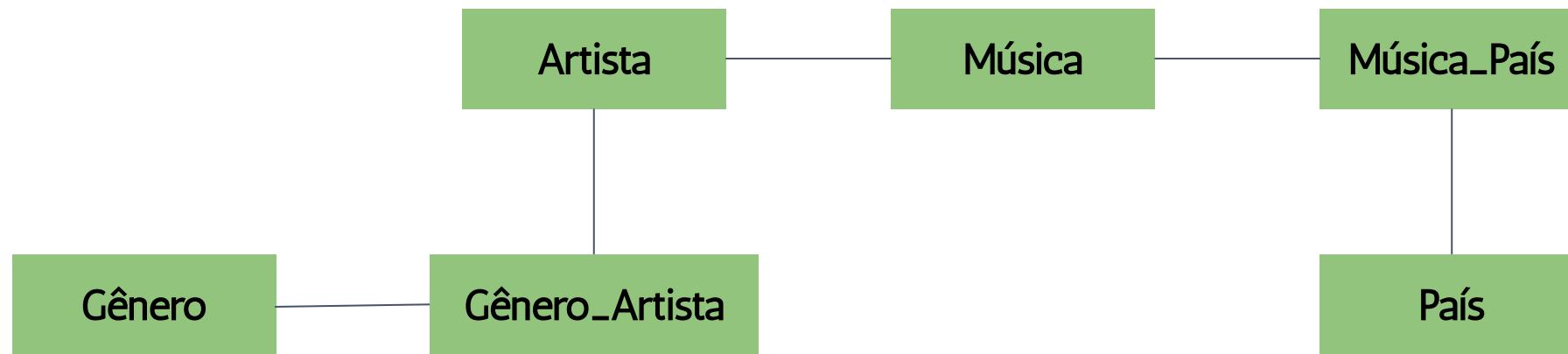
A	B	C	D	E	F	G	H	I	J	K	L
	artist	mk	position	track	streams	url	id	popularity	followers	genres	type
1	1Kilo	br	41	Deixe Me Ir - Acoustic	144732	<a href="https://open.spotify.com/track/6E2st8OqlaS7Pl">https://open.spotify.com/track/6E2st8OqlaS7Pl</a>	6E2st8OqlaS7Pl	70	77903	deep funk carioca	artist
2	21 Savage	us	108	X (feat. Future)	236421	<a href="https://open.spotify.com/track/1URnnhqYAYcrq">https://open.spotify.com/track/1URnnhqYAYcrq</a>	1URnnhqYAYcrq	88	1136786	dwn trap,pop rap,rap,trap	artist
3	21 Savage	us	169	No Heart	179652	<a href="https://open.spotify.com/track/1URnnhqYAYcrq">https://open.spotify.com/track/1URnnhqYAYcrq</a>	1URnnhqYAYcrq	88	1136786	dwn trap,pop rap,rap,trap	artist
4	21 Savage	ca	113	X (feat. Future)	25926	<a href="https://open.spotify.com/track/1URnnhqYAYcrq">https://open.spotify.com/track/1URnnhqYAYcrq</a>	1URnnhqYAYcrq	88	1136786	dwn trap,pop rap,rap,trap	artist
5	257ers	de	195	Holz	33540	<a href="https://open.spotify.com/track/6ihLfpY3cmdGyV">https://open.spotify.com/track/6ihLfpY3cmdGyV</a>	6ihLfpY3cmdGyV	65	230614	deep german hip hop,german	artist
6	257ers	de	170	Holland	36300	<a href="https://open.spotify.com/track/6ihLfpY3cmdGyV">https://open.spotify.com/track/6ihLfpY3cmdGyV</a>	6ihLfpY3cmdGyV	65	230614	deep german hip hop,german	artist
7	2 Chainz	ch	186	4:00 AM	2846	<a href="https://open.spotify.com/track/17lZAZA2AI0HwC">https://open.spotify.com/track/17lZAZA2AI0HwC</a>	17lZAZA2AI0HwC	91	1435305	dwn trap,pop rap,rap,soul	artist
8	2 Chainz	us	42	It's A Vibe	441763	<a href="https://open.spotify.com/track/17lZAZA2AI0HwC">https://open.spotify.com/track/17lZAZA2AI0HwC</a>	17lZAZA2AI0HwC	91	1435305	dwn trap,pop rap,rap,soul	artist
9	2 Chainz	us	34	4:00 AM	553507	<a href="https://open.spotify.com/track/17lZAZA2AI0HwC">https://open.spotify.com/track/17lZAZA2AI0HwC</a>	17lZAZA2AI0HwC	91	1435305	dwn trap,pop rap,rap,soul	artist
10	2 Chainz	ca	60	4:00 AM	43323	<a href="https://open.spotify.com/track/17lZAZA2AI0HwC">https://open.spotify.com/track/17lZAZA2AI0HwC</a>	17lZAZA2AI0HwC	91	1435305	dwn trap,pop rap,rap,soul	artist
11	2 Chainz	nz	89	It's A Vibe	6866	<a href="https://open.spotify.com/track/17lZAZA2AI0HwC">https://open.spotify.com/track/17lZAZA2AI0HwC</a>	17lZAZA2AI0HwC	91	1435305	dwn trap,pop rap,rap,soul	artist
12	2 Chainz	ca	66	It's A Vibe	40267	<a href="https://open.spotify.com/track/17lZAZA2AI0HwC">https://open.spotify.com/track/17lZAZA2AI0HwC</a>	17lZAZA2AI0HwC	91	1435305	dwn trap,pop rap,rap,soul	artist
13	2 Chainz	us	121	Good Drank	216344	<a href="https://open.spotify.com/track/17lZAZA2AI0HwC">https://open.spotify.com/track/17lZAZA2AI0HwC</a>	17lZAZA2AI0HwC	91	1435305	dwn trap,pop rap,rap,soul	artist
14	5 After Midnight	gb	83	Up In Here	70266	<a href="https://open.spotify.com/track/3x9lfYx4a2FPsx">https://open.spotify.com/track/3x9lfYx4a2FPsx</a>	3x9lfYx4a2FPsx	58	7843		artist
15	5 After Midnight	ie	103	Up In Here	4658	<a href="https://open.spotify.com/track/3x9lfYx4a2FPsx">https://open.spotify.com/track/3x9lfYx4a2FPsx</a>	3x9lfYx4a2FPsx	58	7843		artist
16	6cyclemind	ph	93	Prinsesa	29598	<a href="https://open.spotify.com/track/3nZa8vRD64ueq">https://open.spotify.com/track/3nZa8vRD64ueq</a>	3nZa8vRD64ueq	58	32274	neo mellow,opm,pinoy al	artist

Como ficaria o modelo normalizado?

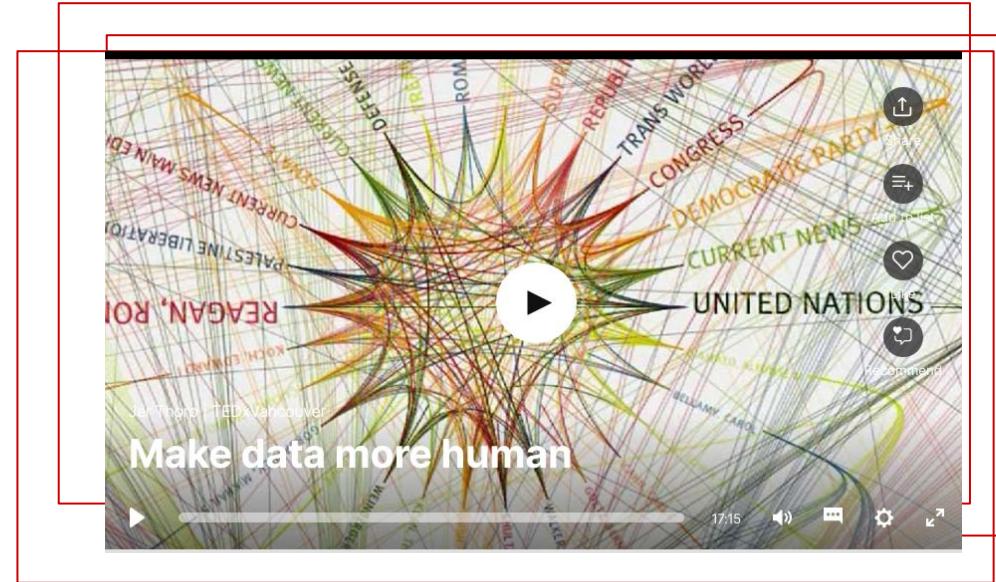


# Normalização de dados

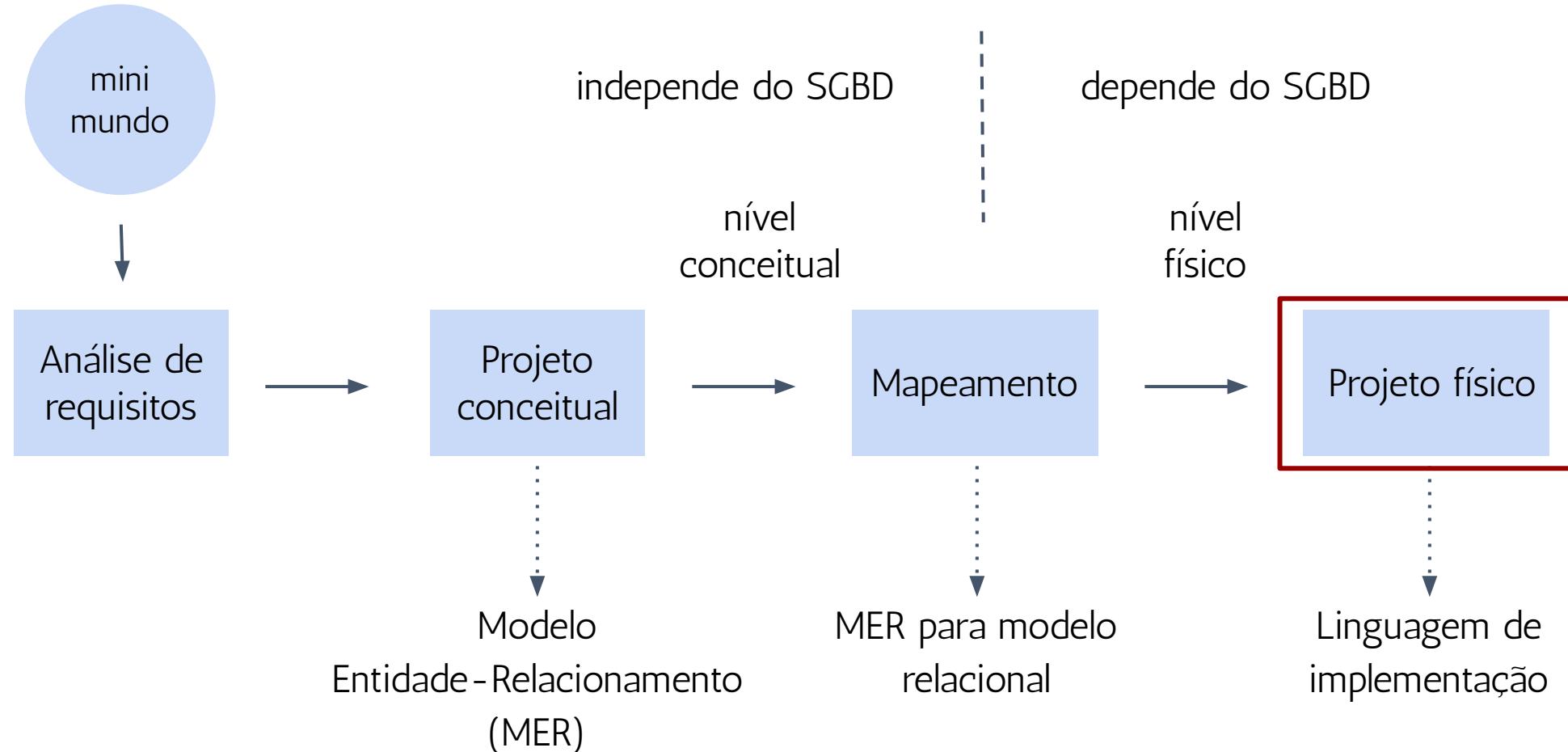
Exemplo - Spotify



# Para assistir uma outra hora ;)



# Projeto de Banco de Dados



# SQL

## Structured Query language

- É uma linguagem de pesquisa **declarativa** padrão para banco de dados relacionais
  - Tipo de paradigma onde o **foco** não está no "como" e sim **no "que"**  
Mais detalhes sobre as diferenças de paradigmas [aqui](#).
- Criada no início dos anos 70 com o objetivo de demonstrar que era viável implementar o modelo relacional
  - Proposto por E. F. Codd, membro do laboratório de pesquisa da IBM em San Jose, CA
- Padrão utilizado pelos sistemas de banco de dados relacionais
- Pode ser dividida nas seguintes categorias:
  - DQL - Data Query Language
  - DDL - Data Definition Language
  - DML - Data Manipulation Language
  - DCL - Data Control Language
  - DTL - Data Transaction Language



# SQL

## Resumindo o que vamos ver

- DDL

CREATE TABLE  
CREATE INDEX  
CREATE VIEW  
ALTER TABLE  
ALTER INDEX  
DROP INDEX  
DROP VIEW

- DML

INSERT  
UPDATE  
DELETE

- DQL

SELECT

- Cláusulas
  - FROM, WHERE
  - GROUP BY, HAVING
  - ORDER BY, DISTINCT
  - UNION

- Operadores lógicos e relacionais

AND, OR, NOT  
>, >=, <, <=, ==><>  
BETWEEN, IN, LIKE

- Funções de agregação

MAX, MIN  
AVG, SUM, COUNT

- Junções

INNER JOIN  
LEFT JOIN  
RIGHT JOIN  
FULL OUTER JOIN

- DCL

GRANT  
REVOKE

- Backup e Restauração



# Volte aqui sempre que estiver agarrado num erro de SQL

(aqui vai um checklist de verificações, pode ser que esse slide dê uma luz)

- Olhe a mensagem de erro. Ela dá alguma dica?
- Execute uma declaração SQL por vez
  - seu cursor está na linha correta a ser executada?
  - tente selecionar apenas bloco de linhas que precisa ser executado, e execute.
  - pode ser uma boa adicionar um ';' ao final da sua consulta.
- Você está conectado à base de dados correta?
- Erros de digitação são mais comuns do que a gente pensa
  - os nomes das tabelas/colunas que escreveu estão de acordo com a estrutura de tabelas/colunas que de fato existem no banco de dados?
  - as declarações SQL estão escritas corretamente?
- A sua consulta SQL está escrita seguindo a estrutura/ordem correta?



# SQL

## DDL - Data Definition Language Tabelas

### CREATE TABLE

Utilizado para criar novas tabelas no banco de dados. É necessário definir o nome da tabela, assim como o nome e o tipo de dados de cada coluna.

```
CREATE TABLE nome_tabela (coluna_1 datatype, coluna_2 datatype, coluna_3 datatype);
```

### ALTER TABLE

Permite que a estrutura da tabela seja modificada, adicionando ou excluindo novas colunas ou modificando o tipo de dados de alguma coluna, por exemplo.

```
ALTER TABLE nome_tabela ADD nome_coluna datatype;  
ALTER TABLE nome_tabela DROP COLUMN nome_coluna;  
ALTER TABLE nome_tabela MODIFY COLUMN nome_coluna datatype;
```



# SQL

## DDL - Data Definition Language Tabelas

### DROP TABLE

Permite a exclusão de tabelas. Se existirem registros na tabela, os mesmos também serão excluídos. Deve-se tomar atenção no que diz respeito às violação das restrições de integridade (chaves estrangeiras, por exemplo)

```
DROP TABLE nome_tabela;  
DROP DATABASE nome_database;
```



# SQL

## DML - Data Manipulation Language

### Constraints

Usadas para criar regras para os atributos nas tabelas do banco de dados

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified
- **INDEX** - Used to create and retrieve data from the database very quickly

Mais detalhes [aqui](#).



# SQL

## DDL - Data Definition Language Índices

- Os índices são usados para recuperar dados mais rapidamente. Os usuários não podem ver os índices, eles são usados apenas para acelerar consultas.

### CREATE INDEX

Permite a criação de índices a partir de uma ou mais colunas.

```
CREATE INDEX nome indice ON nome_tabela (nome_coluna);
CREATE UNIQUE INDEX nome indice ON nome_tabela (nome_coluna1, nome_coluna2);
```



# SQL

## DDL - Data Definition Language Índices

**ALTER** INDEX  
(extensão do Postgres, pode não ser o mesmo para outros SGBDs)  
Permite alterar a definição de um índice.

```
ALTER INDEX nome indice RENAME TO novo nome;
```

**DROP** INDEX  
Permite a exclusão de índices. As colunas continuarão existindo, apenas o índice associado será removido.

```
DROP INDEX nome indice;
```



# SQL

## DDL - Data Definition Language Views

- Views são tabelas virtuais formadas a partir de consultas SQL, contendo linhas e colunas como se fosse uma tabela do modelo do banco de dados. Também podem ser entendidas como consultas armazenadas.

### CREATE VIEW

Permite a criação de views a partir da definição de uma consulta.

Detalhes sobre consultas serão apresentados em slides mais à frente..

```
CREATE VIEW nome_view AS
    SELECT coluna1, coluna2
    FROM nome_tabela
    WHERE condição;
```



# SQL

## DDL - Data Definition Language Views

### CREATE OR REPLACE VIEW

Permite a atualização de views a partir da definição de uma consulta.

```
CREATE OR REPLACE VIEW nome_view AS
  SELECT coluna1, coluna2
    FROM nome_tabela
   WHERE condição;
```

### DROP

### VIEW

Permite a exclusão de views. As tabelas utilizadas na criação da view não são removidas..

DROP

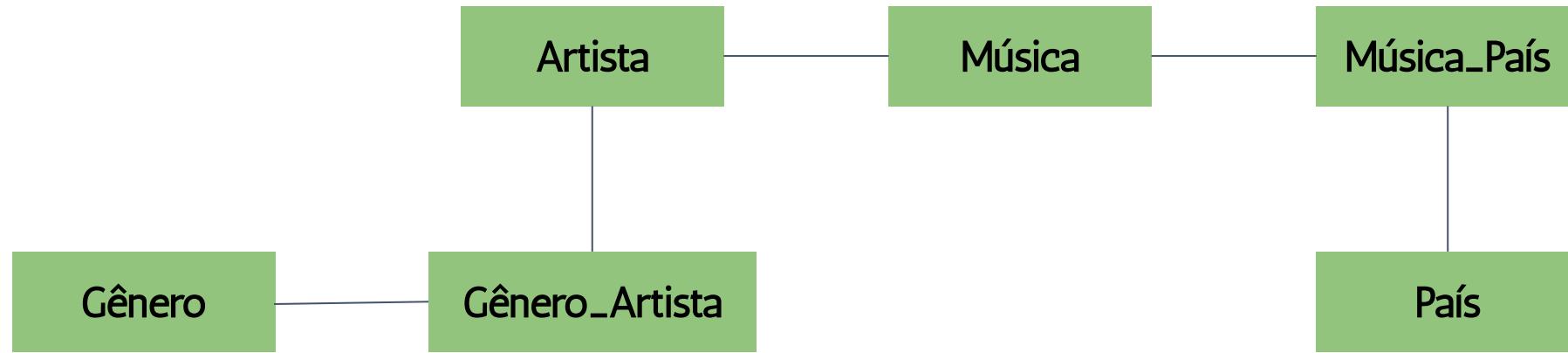
VIEW

nome\_view;



# Criação do Modelo

Exemplo - Spotify



# SQL

## DML - Data Manipulation Language

### INSERT

Utilizado na inserção de novos registros em uma tabela.

```
INSERT INTO nome_tabela (coluna_1,coluna_2,coluna_3)
VALUES (valor_1, 'valor_2', valor_3);
```

### UPDATE

Permite a edição/alteração de registros em uma tabela.

```
UPDATE
SET                 alguma_coluna
WHERE alguma_coluna = algum_valor;
```

### DELETE

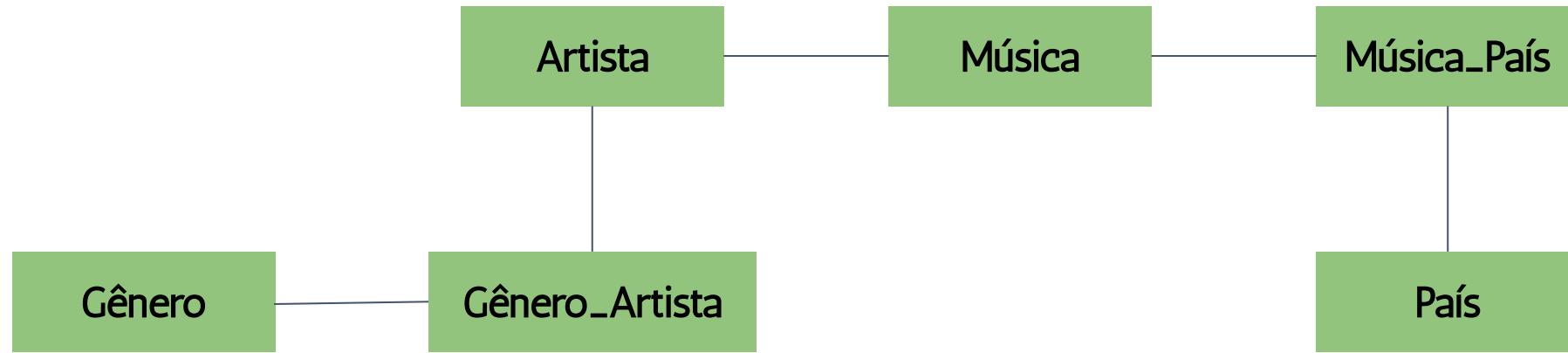
Permite que registros sejam deletados de uma tabela.

```
DELETE FROM nome_tabela    nome_tabela
wHERE alguma_coluna = algum_valor;lor
```



# Inserção de Dados

Exemplo - Spotify



# SQL

## DQL - Data Query Language

- Compreende o comando responsável por realizar consultas ao banco de dados: **SELECT**
- Pode ser combinado com diferentes **cláusulas** e **operadores** para a construção de consultas tanto simples quanto mais complexas

**Exemplo** (mais simples possível, sem cláusulas/operadores)

`SELECT 1 as id;`



Neste exemplo estamos selecionando o *número 1* com o 'apelido' *id*.

No entanto, o ideal e mais comum é utilizarmos as cláusulas para indicar qual será a *origem* das informações que estamos consultando/buscando no banco de dados.



# SQL

## Cláusulas

**aluno**

matricula	nome	idade
123456789	João Silveira	22

- Condições de modificação que definem/restringem os dados que serão selecionados ou modificados na consulta

**FROM**

Especifica a fonte (tabela) dos dados a serem recuperadas, podendo ser apenas uma ou várias.  
 É utilizada no comando SELECT e forma a base de qualquer consulta SQL.

```
SELECT  
FROM nome_tabela;
```



```
SELECT matricula, nome  
FROM aluno;
```

**WHERE**

Restringe os dados através de operações que testam se cada registro satisfaz ou não a condição estabelecida.

```
SELECT  
FROM nome_tabela  
WHERE nome_coluna operador valor;
```



```
SELECT matricula, nome  
FROM aluno  
WHERE idade > 18;
```

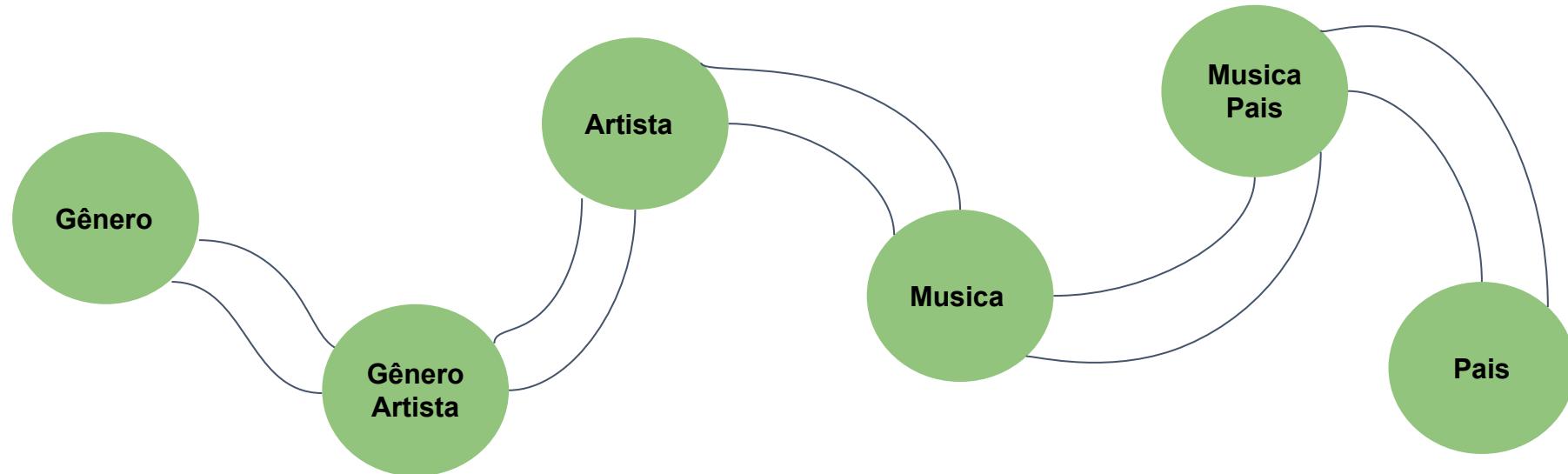


# SQL

## DQL - Data Query Language

Como pensar nas consultas? **Um caminho a ser percorrido.**

Esse caminho determina as **junções** de tabelas que precisam ser feitas.

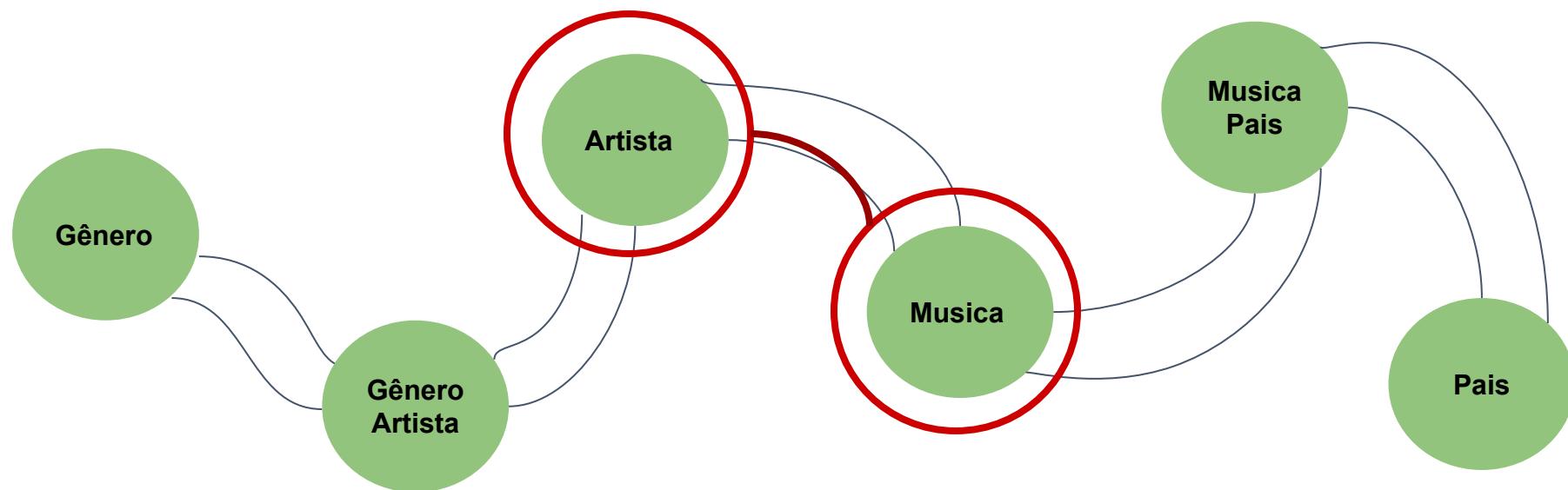


# SQL

## DQL - Data Query Language

O **que** precisa ser selecionado? De **onde** precisa ser selecionado?

Exemplo: Selecionar nome do artista e nome da música.

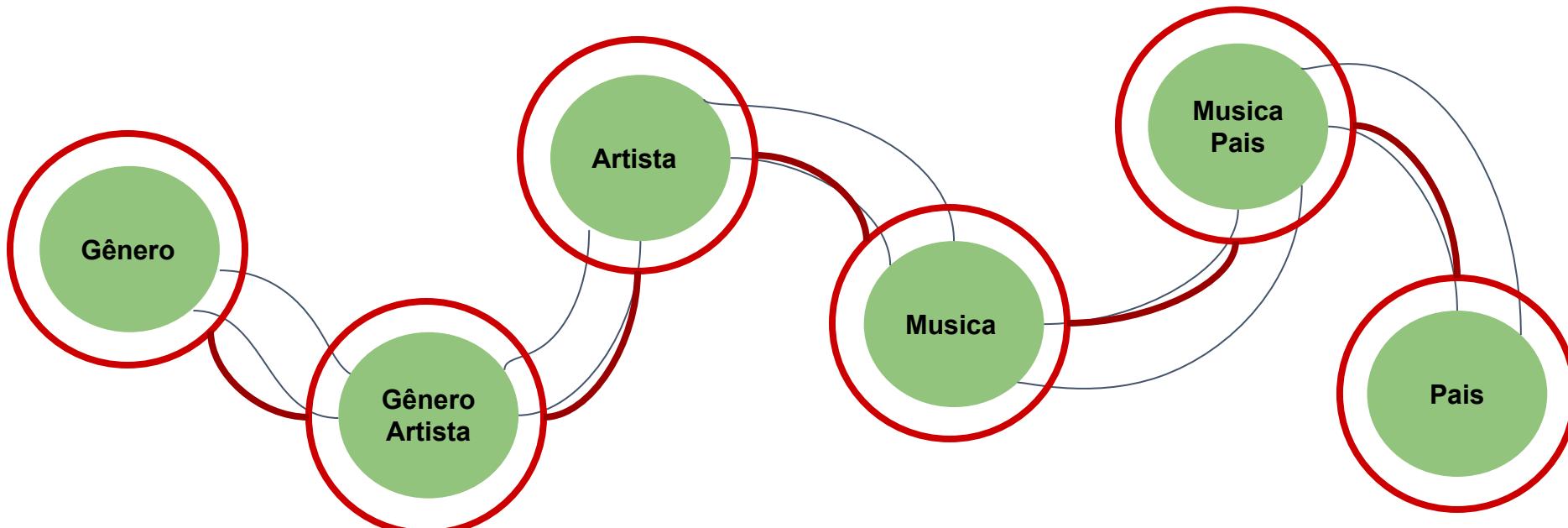


# SQL

## DQL - Data Query Language

O que precisa ser selecionado? De onde precisa ser selecionado?

Exemplo: Selecionar os países que tocam o gênero *latino*  
ou ainda, selecionar a listagem de gêneros que tocam em cada país



# SQL

## Cláusulas

### GROUP BY

Utilizada em conjunto com funções de agregação para o agrupamento do resultado por uma ou mais colunas.

```
SELECT COUNT(*)  
      FROM nome_tabela  
     GROUP BY nome_coluna;
```

### HAVING

Utilizada apenas quando aplicada a cláusula GROUP BY, a fim de limitar os dados recuperados de acordo com alguma restrição.

```
SELECT nome_coluna, função_agregação(nome_coluna)  
      FROM nome_tabela  
     WHERE nome_coluna operador valor  
     GROUP BY nome_coluna  
    HAVING função_agregação(nome_coluna) operador valor;
```



# SQL

## Cláusulas

### ORDER BY

Utilizada para ordenar os registros, levando em consideração uma ou mais colunas

```
SELECT  
FROM  
ORDER BY nome_coluna ASC|DESC;
```

nome\_coluna  
nome\_tabela

### DISTINCT

Utilizada para selecionar dados sem que haja repetição.

```
SELECT DISTINCT nome_coluna  
FROM nome_tabela;
```

### LIMIT

Especifica um valor máximo de linhas que será retornado.

```
SELECT  
FROM  
LIMIT valor;
```

nome\_coluna(s)  
nome\_tabela



# SQL

## Operadores lógicos e relacionais

- **AND:** E lógico.  
Avalia as condições e retorna as colunas se todas as condições forem verdadeiras.
- **OR:** OU lógico.  
Avalia as condições e retorna as colunas se alguma das condições for verdadeira..
- **NOT:** Negação lógica.  
Retorna as colunas o valor contrário da expressão seja verdadeiro.

```
SELECT nome_coluna(s)
FROM nome_tabela
WHERE nome_coluna_1 = valor_1
AND|OR nome_coluna_2 = valor_2;
```

```
SELECT nome_coluna(s)
FROM nome_tabela
WHERE NOT nome_coluna = valor;
```



# SQL

## Operadores lógicos e relacionais

Operador	Função
<	Menor
>	Maior
<=	Menor ou igual
>=	Maior ou igual
==	Igual
<>	Diferente

Operador	Função
BETWEEN	<code>WHERE nome_coluna BETWEEN valor_1 AND valor_2;</code>
LIKE	<code>WHERE nome_coluna LIKE padrão;</code>
IN	<code>WHERE nome_coluna IN (valor_1,valor_2,..., valor_n);</code>



# SQL

## Funções de agregação

### AVG

Retorna a média dos valores de uma determinada coluna.

```
SELECT      AVG(nome_coluna)
FROM nome_tabela;
```

### SUM

Retorna a soma dos valores de uma determinada coluna

```
SELECT      SUM(nome_coluna)
FROM nome_tabela;
```

### COUNT

Retorna o número de linhas onde a coluna possui valor diferente de NULL.

```
SELECT      COUNT(nome_coluna)
FROM nome_tabela;
```

### MIN E MAX

Retorna o menor ou maior valor encontrado na coluna passada como argumento.

```
SELECT MIN(nome_coluna)
FROM nome_tabela;
```

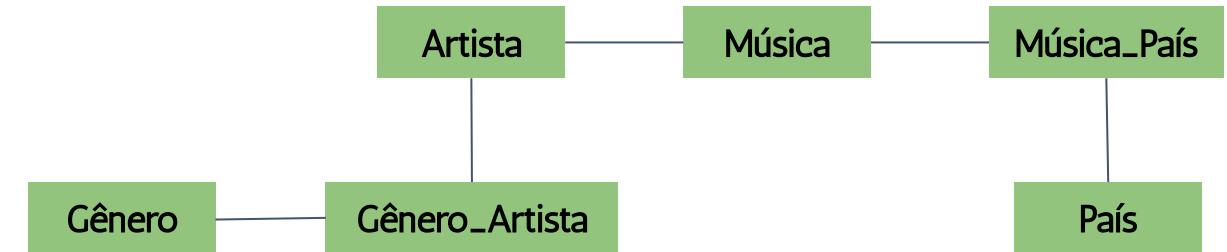


# Consultas Básicas

## Exemplo - Spotify

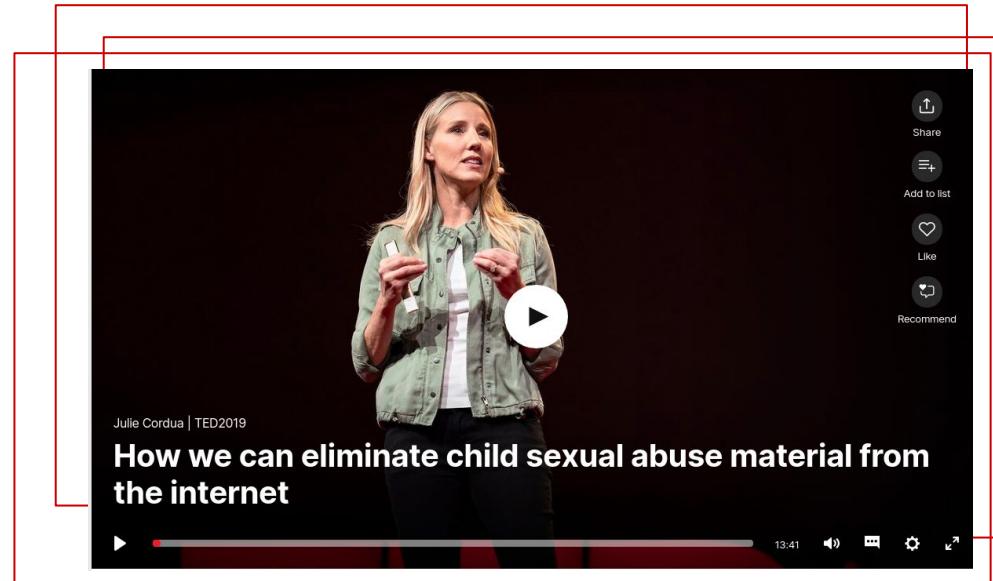
- Consultar nome dos artistas
- Consultar nome dos artistas sem repetição
- Quantidade total de gêneros distintos
- Quantidade total de músicas distintas
- Listar nome dos artistas e popularidade, ordenando pela popularidade (asc)

Ideias?



# Para assistir

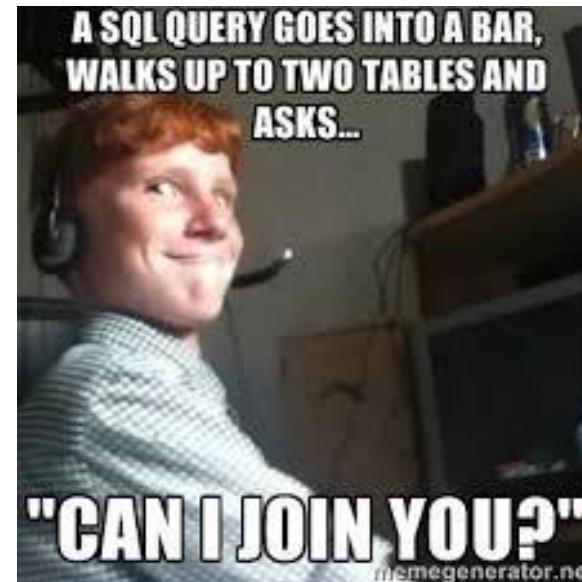
uma outra hora ;)



# SQL

## Junções

- Em um banco de dados já vimos que podemos ter duas ou mais tabelas relacionadas entre si
- Muitas vezes as consultas precisam **recuperar** dados de **tabelas diferentes**
- Por isso, precisamos **definir critérios de junções** para obter estes dados



\* Lembra da ideia de que uma consulta pode envolver mais de uma tabela?

E de como podemos pensar nas junções de tabelas como um *caminho a ser percorrido*?

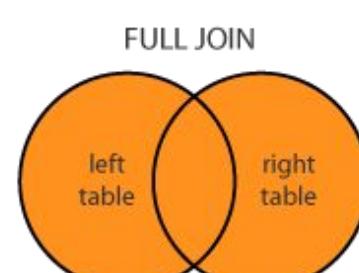
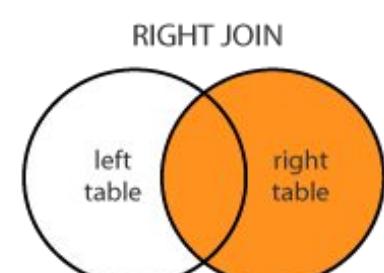
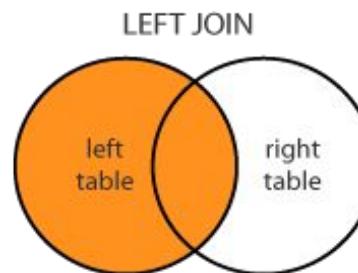
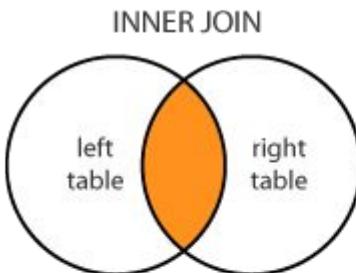


# SQL

## Junções

- Em um banco de dados já vimos que podemos ter duas ou mais tabelas relacionadas entre si
- Muitas vezes as consultas precisam **recuperar** dados de **tabelas diferentes**
- Por isso, precisamos definir **critérios de junções** para obter estes dados

Tipos de junções possíveis:

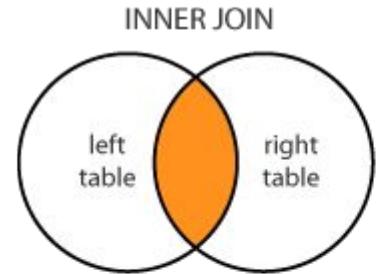


```
SELECT      nome_coluna(s)      FROM      tabela_1  
RIGHT      |      LEFT      |      FULL      JOIN      tabela_2  
ON tabela_1.nome_coluna = tabela_2.nome_coluna;
```



# SQL

## Junções



### INNER JOIN

Retorna todas as linhas quando há pelo menos um registro correspondente em ambas as tabelas.

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	1	1996-09-19
10310	1	1996-09-20

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

```

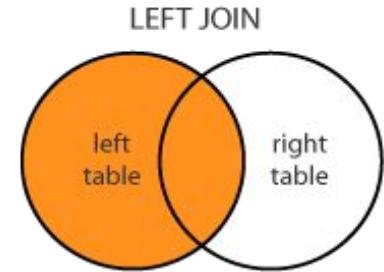
SELECT Orders.OrderID,
       Customers.CustomerName,
       Orders.OrderDate
  FROM Orders
 INNER JOIN Customers
    ON Orders.CustomerID=Customers.CustomerID
  
```

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados	1996-09-18
10309	Alfreds Futterkiste	1996-09-19
10310	Alfreds Futterkiste	1996-09-20



# SQL

## Junções



### LEFT JOIN

Retorna todas as linhas da tabela à direita, e os registros correspondentes da tabela esquerda..

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	1	1996-09-19
10310	1	1996-09-20

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

```

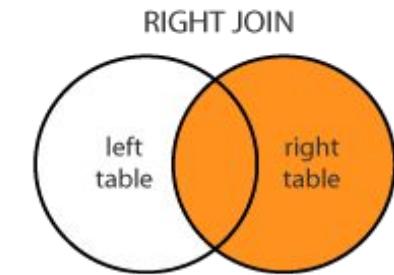
SELECT Orders.OrderID,
       Customers.CustomerName,
       Orders.OrderDate
  FROM Orders
 LEFT JOIN Customers
        ON Orders.CustomerID=Customers.CustomerID
    
```

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados	1996-09-18
10309	Alfreds Futterkiste	1996-09-19
10310	Alfreds Futterkiste	1996-09-20



# SQL

## Junções



### RIGHT JOIN

Retorna todas as linhas da tabela à direita, e os registros correspondentes da tabela esquerda..

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	1	1996-09-19
10310	1	1996-09-20

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

```

SELECT Orders.OrderID,
       Customers.CustomerName,
       Orders.OrderDate
  FROM Orders
 RIGHT JOIN Customers
    ON Orders.CustomerID=Customers.CustomerID
  
```

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados	1996-09-18
10309	Alfreds Futterkiste	1996-09-19
10310	Alfreds Futterkiste	1996-09-20
	Antonio Moreno Taquería	



# SQL

## Subconsulta

- É uma instrução SELECT dentro de outra instrução SQL, que pode ser:
  - SELECT, INSERT, UPDATE ou DELETE
- Pode ser extremamente útil ou complicar consultas que poderiam ser facilmente resolvidas apenas com junções, por exemplo.
- Podemos pensar na declaração de subconsultas como uma 'combinação de blocos', exemplo:

```
select .... from .... where ... in
```

Bloco externo

```
    select .... from .... where ....
```

Bloco interno



# SQL

## Formas de usar subconsultas

### 1. Como nova coluna da consulta

```
externo   SELECT P.titulo,  
interno   (SELECT COUNT(C.id_projeto)  
           FROM comentarios C  
           WHERE C.id_projeto = P.id ) AS qtd_com  
           FROM projetos P  
           GROUP BY P.id
```



# SQL

## Formas de usar subconsultas

### 2. Como filtro de uma consulta externa (usando operadores de comparação, in ou exists)

```
externo   SELECT P.id,  
              P.titulo,  
              P.data  
        FROM projetos P  
 WHERE P.id IN  
           (SELECT C.id_projeto  
              FROM comentarios C  
             WHERE P.id = C.id_projeto  
           );  
  
interno
```



# SQL

## Formas de usar subconsultas

3. Como fonte de dados de uma consulta externa (ou seja, fazendo o papel de uma nova tabela)

```

externo   SELECT F.titulo,
              F.Quantidade_Comentarios
FROM
  interno  (SELECT P.titulo,
                  (SELECT COUNT(C.id_projeto)
                   FROM comentarios C
                   WHERE C.id_projeto = P.id ) AS Quantidade_Comentarios
                  FROM projetos P
                ) as F
WHERE F.Quantidade_Comentarios > 2
  
```

Você pode fazer uma infinidade de subconsultas.  
 Só precisa avaliar: é realmente necessário?  
 Vai ajudar ou complicar mais?



# SQL

## Subconsultas com outras instruções SQL

### INSERT

```
externo   INSERT INTO EMPREGADOS(CODIGO, NOME, SALARIO, SECAO)
interno    SELECT CODIGO, NOME, SALARIO, SECAO
                  FROM EMPREGADOS_FILIAL
                     WHERE DEPARTAMENTO = 2
```

### UPDATE

```
externo   UPDATE EMPREGADOS
              SET SALARIO = SALARIO * 1.1
              WHERE SALARIO = (
interno      SELECT MIN(SALARIO)
                      FROM EMPREGADOS
                    )
```

### DELETE

```
externo   DELETE FROM EMPREGADOS
              WHERE SALARIO = (
interno      SELECT MIN(SALARIO)
                      FROM EMPREGADOS
                    )
```



# SQL

## DCL - Data Control Language

### GRANT

Atribui privilégios de acesso do usuário à objetos do banco de dados.

```
GRANT privilegio(s) ON objeto(s) TO usuario(s);
```

### REVOKE

Remove os privilégios de acesso aos objetos obtidos com o comando GRANT.

```
REVOKE privilegio(s) ON objeto(s) TO usuario(s);
```

Privilégios: ALL, SELECT, UPDATE, INSERT, DELETE, DROP, CREATE, ALTER, e outros.

A definição de privilégios estabelece o que cada usuário ou grupo de usuário pode realizar ou acessar no banco de dados.



# SQL

## Backup e Restore

Exemplo no Postgres (pelo terminal)

**Backup**

```
pg_dump -h localhost -U postgres -W -F t <nome_bd> > <nome_arquivo>.tar
```

**Restore**

```
pg_restore -h localhost -U postgres -W -F t -d <nome_bd> <nome_backup>.tar
```



\*No caso do SQLite, o banco de dados fica salvo em um arquivo com extensão .db

Fazer um backup (Ctrl+c, Ctrl+v) desse arquivo já é suficiente



# SQL

## Resumindo o que vimos

- DDL

CREATE TABLE  
CREATE INDEX  
CREATE VIEW  
ALTER TABLE  
ALTER INDEX  
DROP INDEX  
DROP VIEW

- DML

INSERT  
UPDATE  
DELETE

- DQL

SELECT

- Cláusulas
  - FROM, WHERE
  - GROUP BY, HAVING
  - ORDER BY, DISTINCT
  - UNION

- Operadores lógicos e relacionais

AND, OR, NOT  
>, >=, <, <=, ==><>  
BETWEEN, IN, LIKE

- Funções de agregação

MAX, MIN  
AVG, SUM, COUNT

- Junções

INNER JOIN  
LEFT JOIN  
RIGHT JOIN  
FULL OUTER JOIN

- DCL

GRANT  
REVOKE

- Backup e Restauração



# Links e materiais úteis

- [Vídeos tutoriais sobre instalação de ferramentas](#)
- [Documentação do PostgreSQL](#)
- [W3Schools - SQL Tutorial](#)
- [URI Online Judge](#)
- [Hacker Rank](#)



Esse material foi produzido por Maria Luiza Mondelli e está licenciado com a licença Attribution-NonCommercial 4.0 International (CC BY-NC 4.0)

Para mais informações sobre a licença, acesse: <https://br.creativecommons.org/licencas/>



### Atribuição-NãoComercial

#### CC BY-NC

Esta licença permite que outros remixem, adaptem e criem a partir do seu trabalho para fins não comerciais, e embora os novos trabalhos tenham de lhe atribuir o devido crédito e não possam ser usados para fins comerciais, os usuários não têm de licenciar esses trabalhos derivados sob os mesmos termos.



RESIDÊNCIA DE SOFTWARE

