



**SATC**  
EDUCAÇÃO E TECNOLOGIA

## **BANCO DE DADOS**

Engenharia de Software – 2<sup>a</sup> fase

Prof. Jorge Luiz da Silva

# Dados, Informação e Conhecimento

**DADO:** Dados são **valores brutos**, literais, observações documentadas sobre o estado do mundo ou resultado de medição, **não contextualizados**. “São fatos conhecidos que podem ser registrados e não possuem significado completo. Por exemplo, nome, idade, cor, endereços...”.

Um dado pode ser uma **letra**, um **número**, uma **palavra**, bem como **conjuntos** de números e vocábulos desorganizados, o qual não transmite nenhuma informação ou conhecimento.

## **Exemplo:**

Criciúma, 1000, 2020

Içara, 500, 2022

# Dados, Informação e Conhecimento

**INFORMAÇÃO:** A informação traz significado. Quando um conjunto de dados é **agrupado, correlacionado, processado, contextualizado e interpretado**, conseguimos entender o **contexto** que o dado quer nos mostrar, passamos a chamar de **informação**. Como HENRIQUE, Kassio (2021) explica em seu artigo: “Quando um conjunto de dados é organizado de modo a passar uma mensagem dentro de um contexto, temos informação”.

**Exemplo:**

Cidade	Casos do vírus X	Ano
Criciúma	1000	2020
Itá	500	2022

# Dados, Informação e Conhecimento

**CONHECIMENTO:** O conhecimento é gerado a partir da **habilidade de analisar um conjunto de informações** correlacionadas e gerar novas informações mais aprofundadas, ou seja, trazer novos fatos úteis atrelados à experiência.

**Exemplo:**

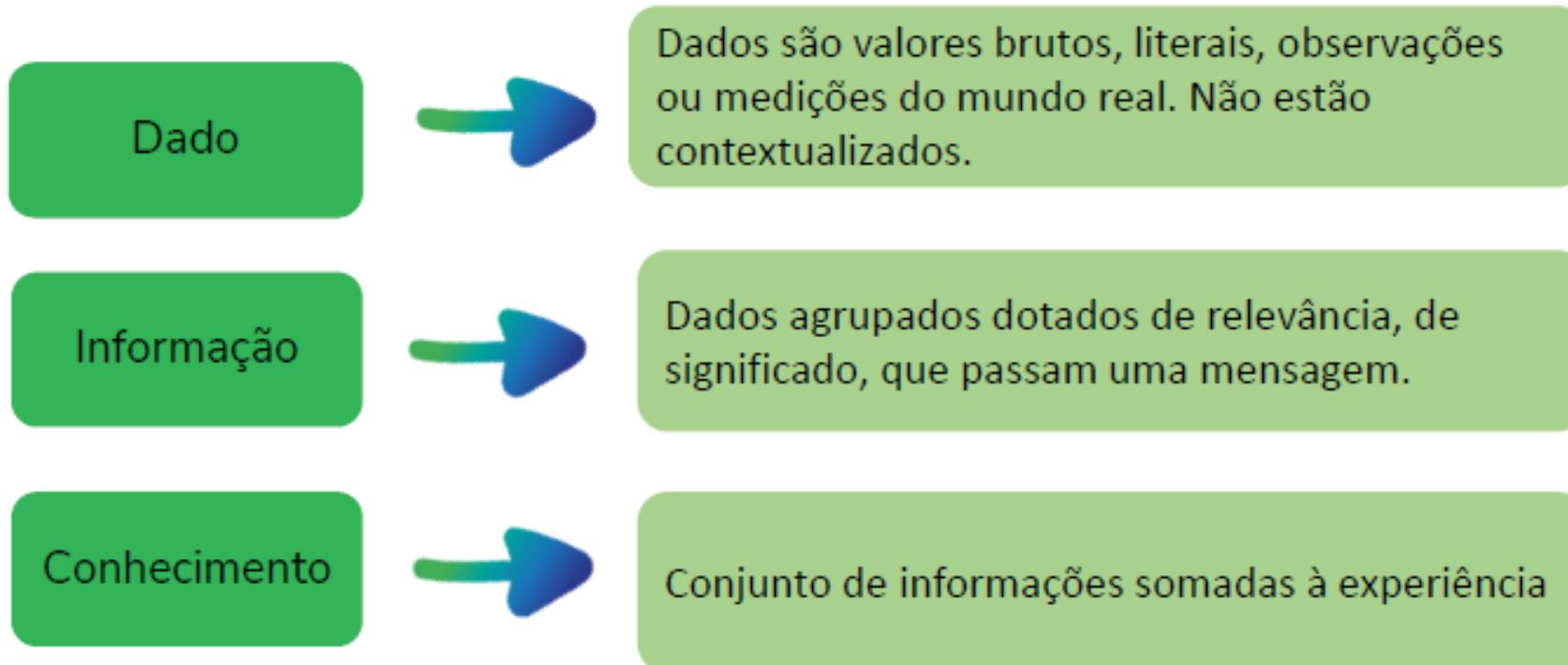
Cidade	Casos do vírus X	Ano
Criciúma	1000	2020
Íçara	500	2022

Através das informações do quadro cima, podemos concluir que em Criciúma tivemos o dobro de casos do vírus X em 2020, do que em Íçara em 2022.

# Dados, Informação e Conhecimento



# Dados, Informação e Conhecimento



# Dados, Informação e Conhecimento



- **Banco de dados** (base e meio da pirâmide)
- Processos organizacionais, **softwares**, procedimentos e **sistemas** (topo)

# Dados, Informação e Conhecimento

## DADOS

Simples observações sobre o estado do mundo

- Facilmente estruturado
- Facilmente obtido por máquinas
- Frequentemente quantificado
- Facilmente transferível

## INFORMAÇÃO

Dados dotados de relevância e propósito

- Requer unidade de análise
- Exige consenso em relação ao significado
- Exige necessariamente a mediação humana

## CONHECIMENTO

Informação valiosa da mente humana. Inclui reflexão, síntese, contexto

- De difícil estruturação
- De difícil captura em máquinas
- Frequentemente tácito
- De difícil transferência

# Tipos de dados



# Tipos de dados

**ESTRUTURADO:** Dados organizados segundo **formato pré-estabelecido**. Podemos citar como exemplo as tabelas, com suas linhas e colunas, e estas tabelas podem se relacionar entre si através de chaves. O dado estruturado pode estar armazenado em planilhas de Excel, ou algum outro SGBD relacional como SQL Server, MySQL, Oracle e outros.

# Tipos de dados

**SEMI-ESTRUTURADO:** Como o nome indica são dados que não seguem o modelo de tabelas, com colunas e linhas em formato rígido predeterminado, pré-formatado, mas possuem algum nível de formato, por exemplo utilizando tags ou chave valor.

Exemplo: XML, HTML, JSON.

```
{ } sample.json > [ ] data
1 [
2   [
3     "data": [
4       {
5         "type": "articles",
6         "id": "1",
7         "attributes": {
8           "title": "Working with JSON Data in python",
9           "description": "This article explains the various ways to work with JSON data in python.",
10          "created": "2020-12-28T14:56:29.000Z",
11          "updated": "2020-12-28T14:56:28.000Z"
12        },
13        "author": {
14          "id": "1",
15          "name": "Aveek Das"
16        }
17      }
18    ]
]
```

# Tipos de dados

**NÃO-ESTRUTURADO:** Dados que não seguem nenhuma organização ou hierarquia predeterminada.

Exemplo: foto, vídeo, áudio, postagem em rede social, localização geográfica, IoT (internet das coisas), e-mails, dentre outros.

# Tipos de dados

## ESTRUTURADOS

Estrutura homogênea e pré-definida

Estrutura independe dos dados

Clara distinção entre estrutura e dados

Estrutura sofre pouca alteração



## SEMIESTRUTURADOS

Estrutura heterogênea e nem sempre pré-definida

Estrutura embutida nos dados

Distinção entre estrutura e dados pouco clara

Estrutura sofre alteração com frequência



## NÃO ESTRUTURADOS

Sem esquema pré-definido

Estrutura independe dos dados

Estrutura irregular nem sempre presente

Estrutura sofre alteração com frequência



# Tipos de análise dos dados

Após coletar, modelar e organizar os dados, gerando informação, são feitas as **análises**.

Nesta fase, de acordo com (DAVENPORT, 2014), são empregadas técnicas para transformação destas informações em **conhecimento** sobre o cenário avaliado, a fim de encontrar padrões, compreender comportamento de pessoas, produtos, mercado e com isso **identificar novas oportunidades de negócio** ou **melhorar o relacionamento** com aqueles que interagem com o seu negócio.

Tipos de Análise:

- **Descritiva**
- **Diagnóstica**
- **Preditiva**
- **Prescritiva**

# Tipos de análise dos dados



# Trabalho sobre Bancos de dados NoSQL

- O que é? Definição. Qual finalidade?
- Quando usar NoSQL, quando não usar NoSQL?
- Tipos de modelos de dados não relacional (NoSQL).
- Comparativo / Diferenças entre modelo (banco de dados) relacional e não relacional.
  
- Apresentação 10/15min.
- Grupos de até 7 alunos.
- Até +-10 slides.

# Modelagem de Dados

**ABSTRAÇÃO DE DADOS** é a técnica de focar nos elementos principais dos dados, ou seja, no **significado**, nas **características** e como se **relacionam entre si** e menos em como **serão armazenados**, como podem ser retornados à aplicação, qual o **tipo do dado** e outras informações internas.

A **MODELAGEM DE DADOS** é exatamente a representação dos dados abstraídos em diferentes níveis, sendo o **modelo conceitual** o de maior abstração e o **físico de menor abstração**, por já se preocupar mais no formato do dado em sua camada final.

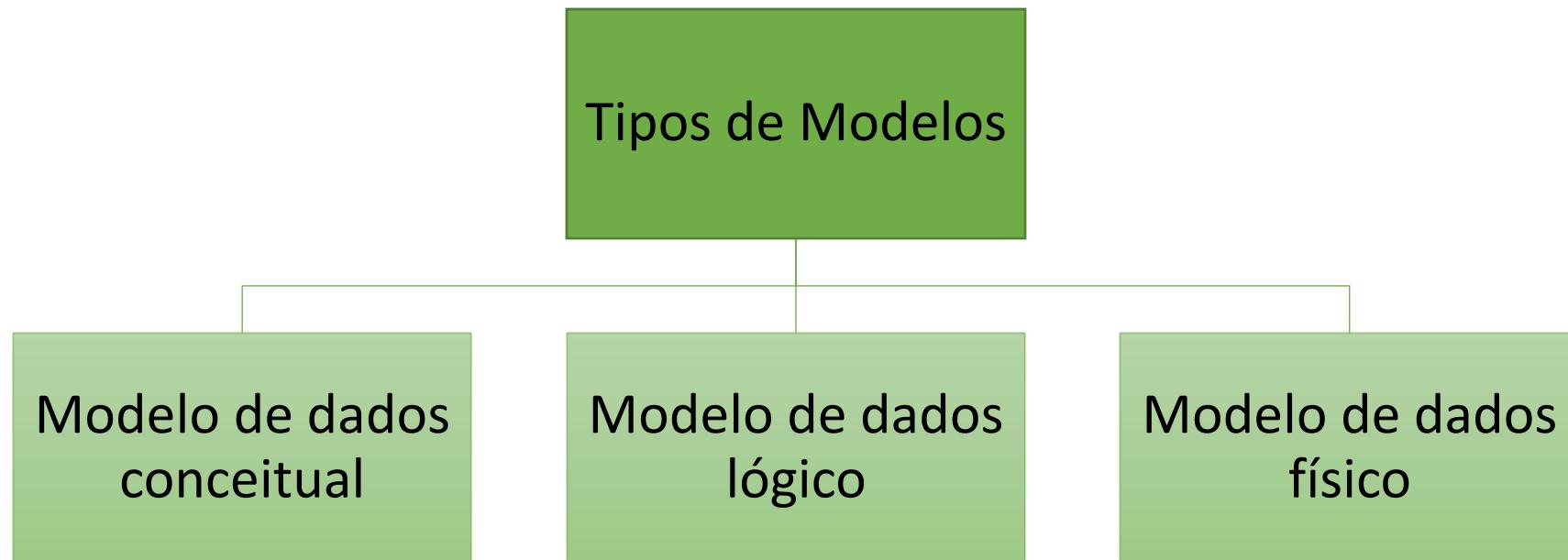
# Modelagem de Dados

**MODELO:** Abstração de um objeto ou evento da realidade. Geralmente representado por diagramas.

**MODELOS DE DADOS:** São representações visuais dos elementos de dados de um ou mais sistemas e as conexões entre eles. Ajudam a definir e estruturar dados no contexto de processos empresariais relevantes.

**MODELAGEM DE DADOS:** Conjunto de técnicas utilizadas para criar um modelo de dados que explique as características de funcionamento e de comportamento dos dados em um determinado sistema ou aplicação.

# Modelagem de Dados



# Modelagem de Dados

## Conceitual

Altamente abstrato

Independe de tecnologias

O principal diagrama é o DER

## Lógico

Abstração média

Ainda independe de tecnologia

Fase de escolha do paradigma: relacional, orientado a objetos, hierárquico

## Físico

Baixa abstração

Escolha do SGBD de acordo com o paradigma escolhido

Detalhamento máximo do dado, com seus tipos e características técnicas

Respeita as restrições do SGBD escolhido

# Modelagem de Dados



# Modelo Conceitual

Maior nível de abstração e o mais próximo da realidade dos usuários

Construído em tempo de levantamento de requisitos

Principal artefato é o MER – Modelo Entidade Relacionamento

O MER é representado pelo DER – Diagrama Entidade Relacionamento

Sem vínculo com tecnologia ou ferramenta, apenas entidades e relacionamentos

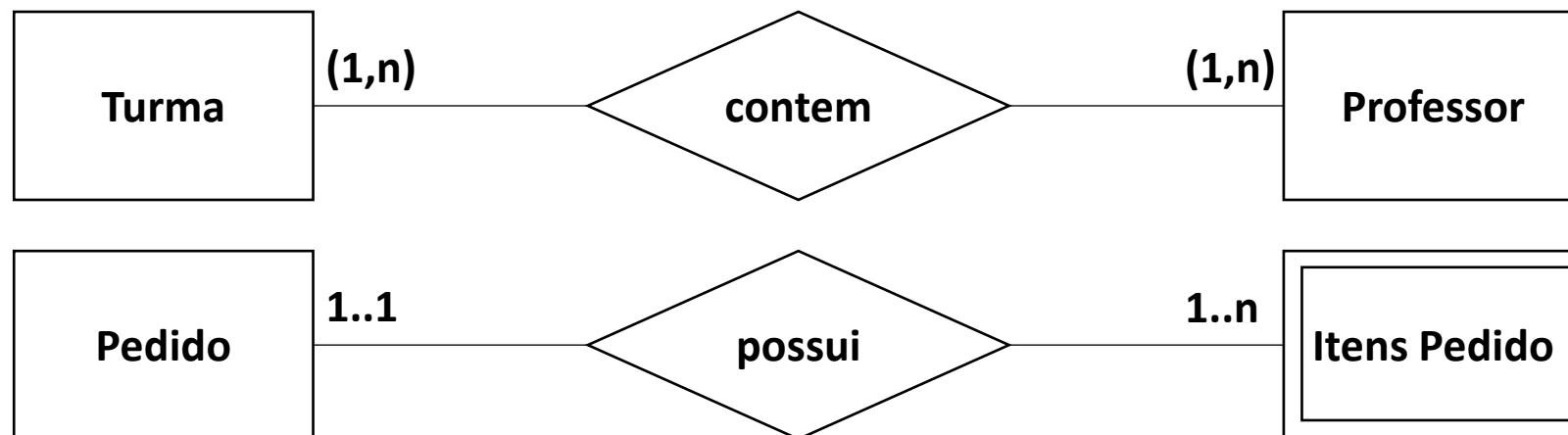
Usado para comunicação com área de negócio

Pode ter diferentes notações

# Modelo Conceitual

## Modelo Entidade Relacionamento – MER

- Criado por Peter Chen (1976) baseado na teoria relacional de Edgar Codd (1970).
- Principais elementos: **Entidade, atributos e relacionamentos.**



# Modelo Conceitual

## Modelo Entidade Relacionamento – MER (Entidade)

- **ENTIDADE** é o objeto básico do modelo ER, é algo no mundo real com uma existência independente.
- Conjunto de objetos da realidade modelada sobre os quais deseja-se manter informações no banco de dados.

# Modelo Conceitual

## Modelo Entidade Relacionamento – MER (Entidade)

CONCRETA	ABSTRATA	FORTE	FRACA	ASSOCIATIVA
Existe no mundo real	Não existe no mundo real, somente conceitualmente	Não depende de outra entidade para existir	Só existe quando parte de um relacionamento dependente	Criada a partir de um relacionamento muitos para muitos (n:n)

Funcionário

Dependente

Fornecem

# Modelo Conceitual

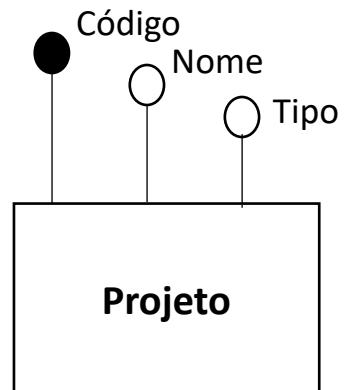
## Modelo Entidade Relacionamento – MER (Atributos)

- **ATRIBUTOS** são características, propriedades específicas que descrevem as entidades.
- Dado que é associado a cada ocorrência de uma entidade ou de um relacionamento.

# Modelo Conceitual

## MODELO ENTIDADE RELACIONAMENTO – MER (ATRIBUTOS)

SIMPLES	COMPOSTO	MONOVALORADO	MULTIVALORADO	CHAVE	ARMAZENADO	DERIVADO
Atômicos, não podem ser divididos	Podem ser divididos em várias partes	Apenas uma ocorrência por Entidade.	Permite mais de uma ocorrência por Entidade.	Representa unicamente uma ocorrência da entidade.	Valores fixos persistidos em banco	Valores obtidos através de cálculo



# Modelo Conceitual

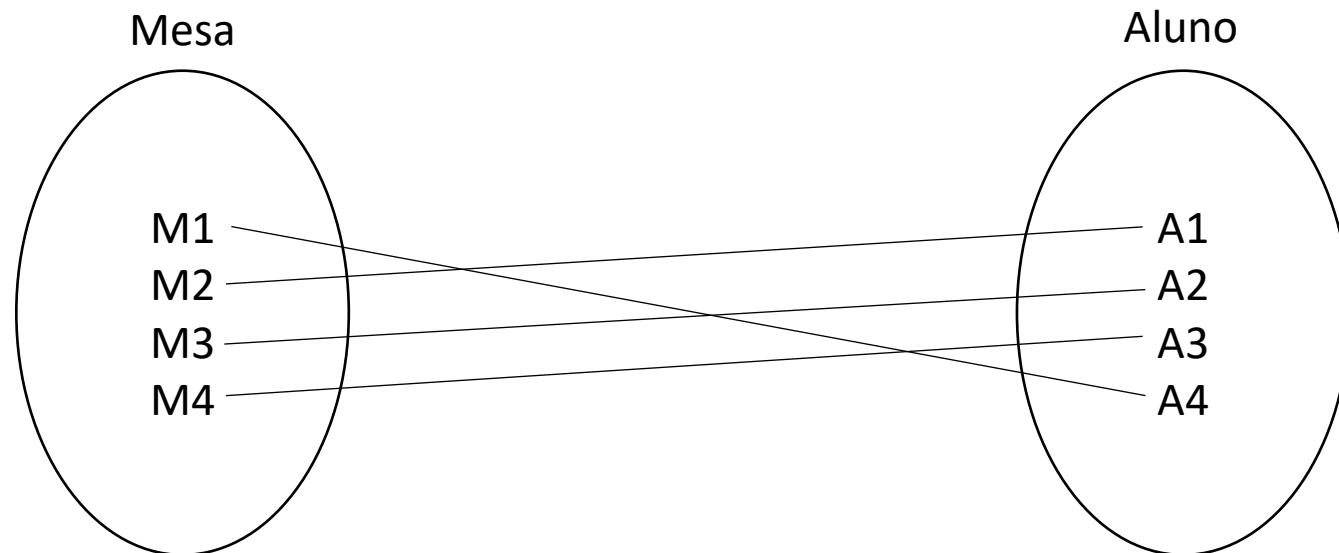
## Modelo Entidade Relacionamento – MER (Relacionamentos)

- **RELACIONAMENTOS** – As entidades são relacionadas umas às outras através de um relacionamento. É uma associação entre uma ou mais entidades.
- Em geral é expresso por verbo ou locução verbal.
- Restrições ou características de um relacionamento:
  - **Cardinalidade:** Quantidade de ocorrências (1:1, 1:n, n:n)
  - **Obrigatoriedade:** Se uma ocorrência de uma entidade é obrigatória ou não (0,1)
  - **Grau:** Auto relacionamento/recursivo (únario), Binário, Ternário, ...

# Modelo Conceitual

## Modelo Entidade Relacionamento – MER (Relacionamentos)

- **CARDINALIDADE (1,1)** – Uma ocorrência da Entidade só pode se relacionar com uma ocorrência de outra entidade.

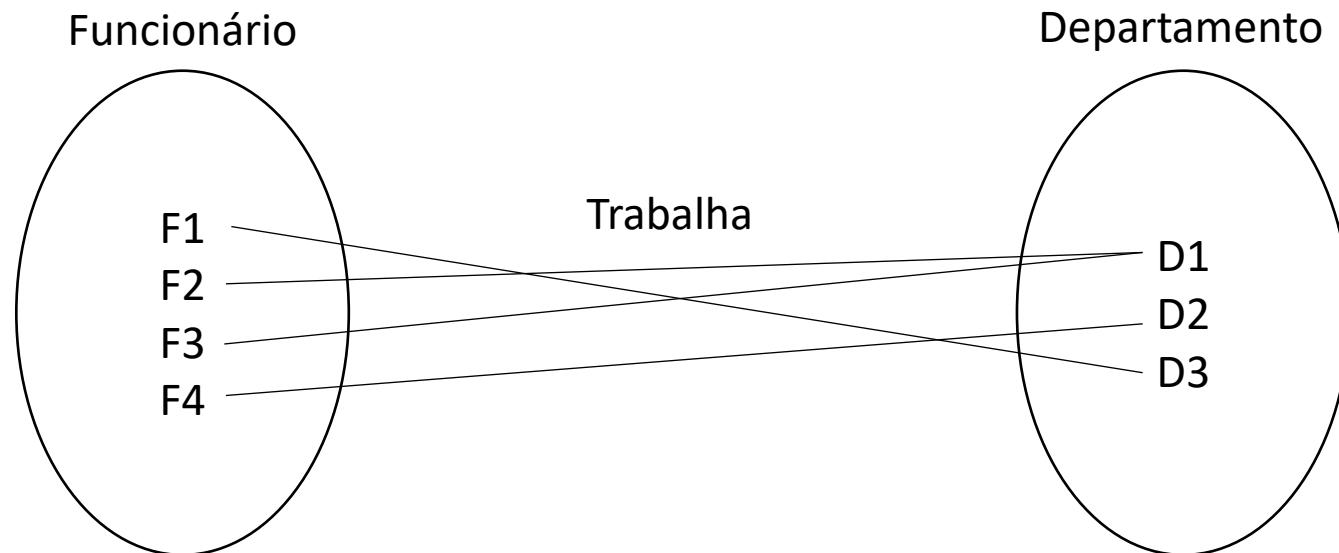


*Leitura: Um aluno ocupa uma mesa e uma mesa pode ser ocupada por um aluno.*

# Modelo Conceitual

## Modelo Entidade Relacionamento – MER (Relacionamentos)

- **CARDINALIDADE (1,n)** – Uma ocorrência da Entidade pode se relacionar com mais de uma ocorrência de outra entidade.

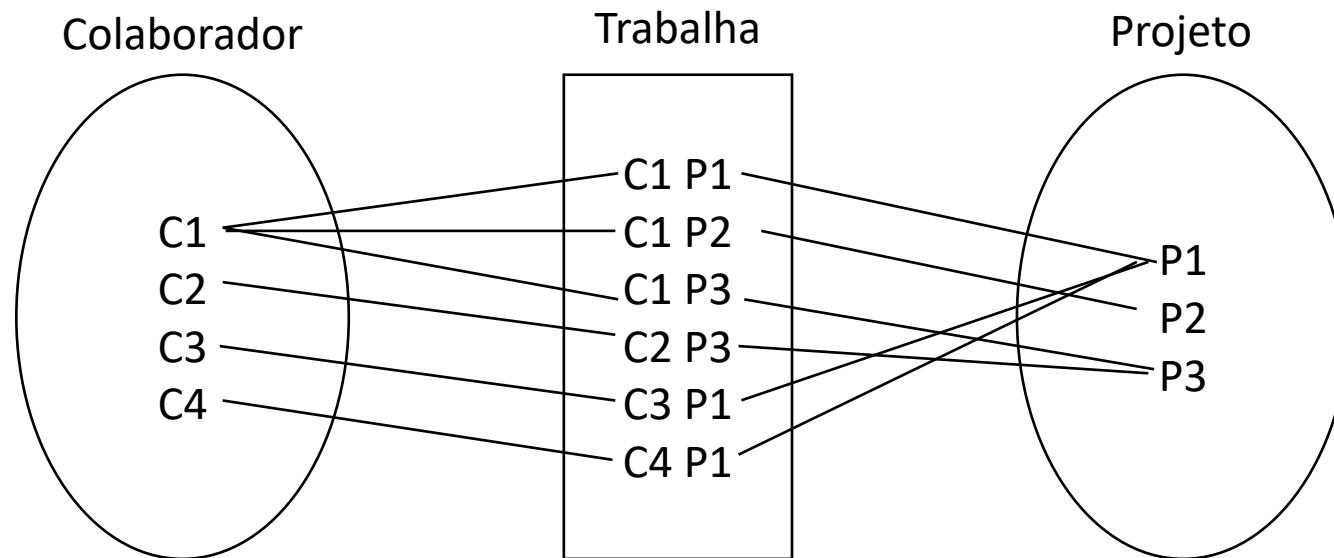


*Leitura: Um funcionário trabalha em um departamento e um departamento pode ter vários (n) funcionários.*

# Modelo Conceitual

## Modelo Entidade Relacionamento – MER (Relacionamentos)

- **CARDINALIDADE (n,n)** – Uma ocorrência da Entidade pode se relacionar com mais de uma ocorrência de outra entidade.



*Leitura:* Um colaborador trabalha em vários (n) projetos e um projeto pode possuir vários (n) colaboradores.

# Modelo Conceitual

## Modelo Entidade Relacionamento – MER (Relacionamentos)

- **OBRIGATORIEDADE** – Define se uma ocorrência de uma entidade é obrigatória ou não.
- Notação **(0,1)** – Sendo 0 para não obrigatório e 1 para obrigatório.

# Modelo Conceitual

## Modelo Entidade Relacionamento – MER (Relacionamentos)

Representação gráfica (Peter Chen)

- Cardinalidade (1,1):



- Cardinalidade (1,n):



- Cardinalidade (n,n):



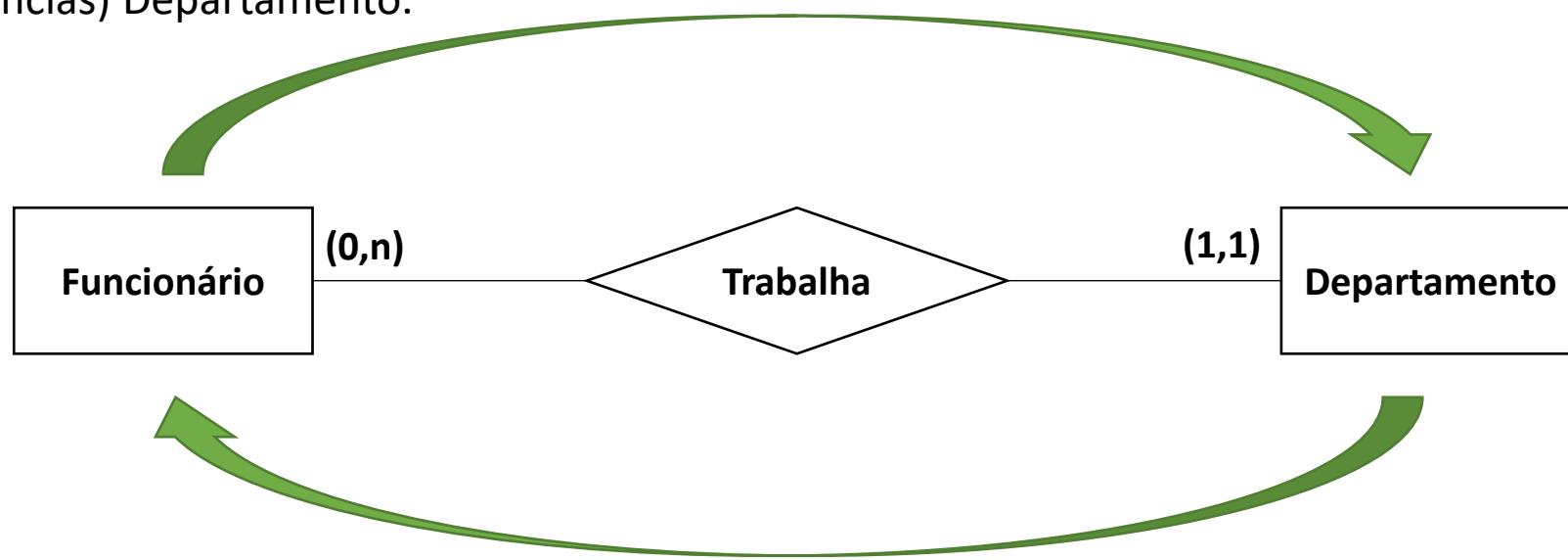
O **primeiro valor** à esquerda representa a **obrigatoriedade** (0, 1) e o **segundo** a **quantidade de ocorrências** possíveis no relacionamento (1, N, 3, 4, 5 e etc)

# Modelo Conceitual

## Modelo Entidade Relacionamento – MER (Relacionamentos)

Exemplo de como é feita a leitura das cardinalidades do DER.

- **De Funcionário para Departamento:** Um funcionário trabalha em 1 (obrigatório) ou no máximo 1 (ocorrências) Departamento.



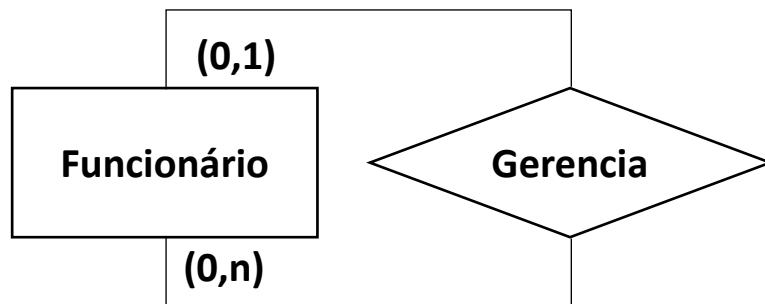
- **De Departamento para Funcionário:** Um Departamento pode possuir 0 (não obrigatório) ou muitos (ocorrências) Funcionários.

# Modelo Conceitual

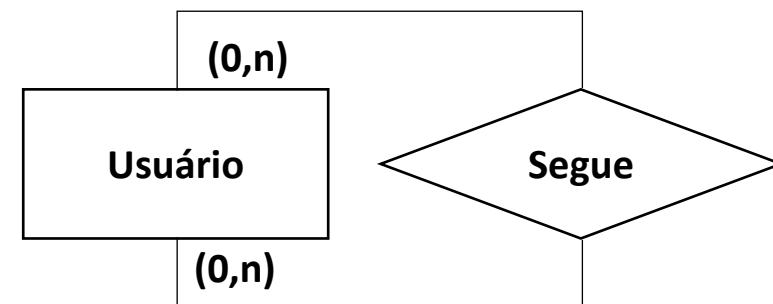
## Modelo Entidade Relacionamento – MER (Relacionamentos)

### GRAU

- **Auto relacionamento:** Relacionamento entre ocorrências de uma mesma entidade (únário).



Um funcionário pode gerenciar no mínimo 0 e no máximo N funcionários.  
Um funcionário pode ser gerenciado no mínimo por 0 e no máximo por 1 funcionário.



Um usuário pode seguir 0 ou mais usuários.  
Um usuário pode ser seguido por 0 ou mais usuários.

# Modelo Conceitual

## Modelo Entidade Relacionamento – MER (Relacionamentos)

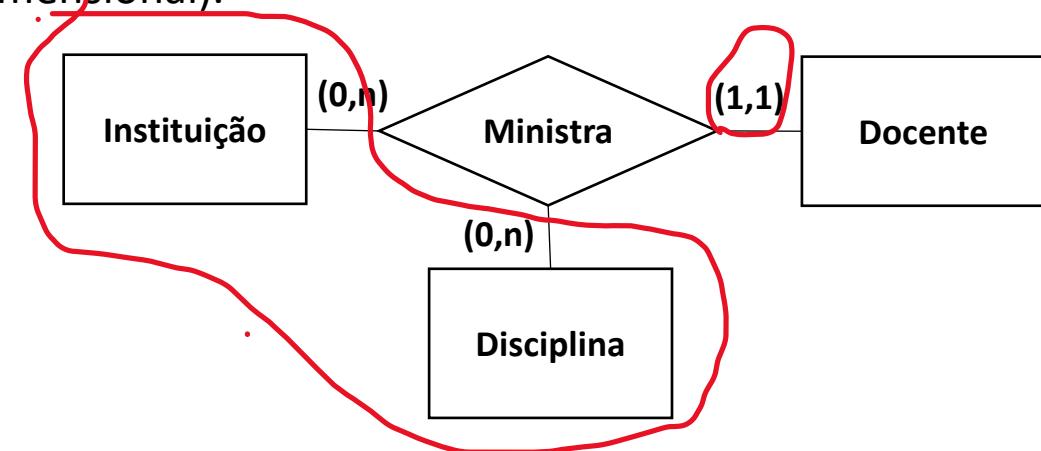
### GRAU

- **Binário:** Relação entre 2 entidades.



Um Pedido contem no mínimo 1 e no máximo N Itens Pedido.  
Um Item Pedido pode estar contido no mínimo em 1 e no máximo em 1 pedido.

- **Ternário:** Relação entre 3 entidades (dimensional).



Um disciplina em uma instituição pode ser ministrada no mínimo por 1 Docente ou no máximo 1 docente.

# Modelo Conceitual

## Modelo Entidade Relacionamento – MER (Relacionamentos)

### ENTIDADE ASSOCIATIVA

- Resultado de um relacionamento de n:n (n,n) e pode ter seus próprios atributos.

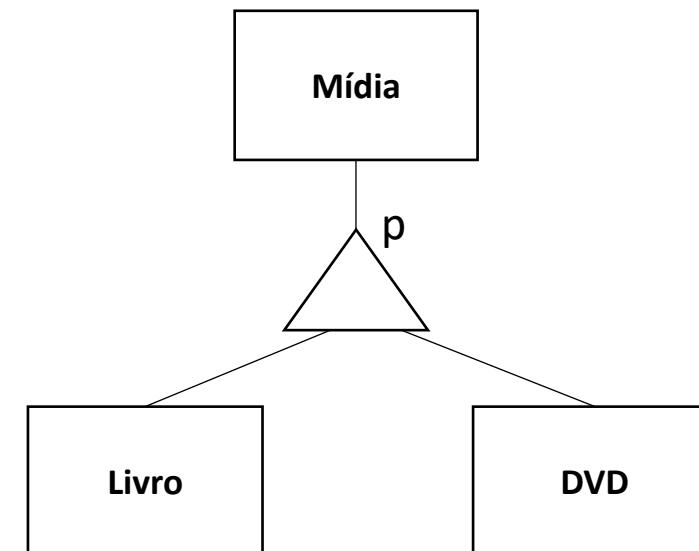
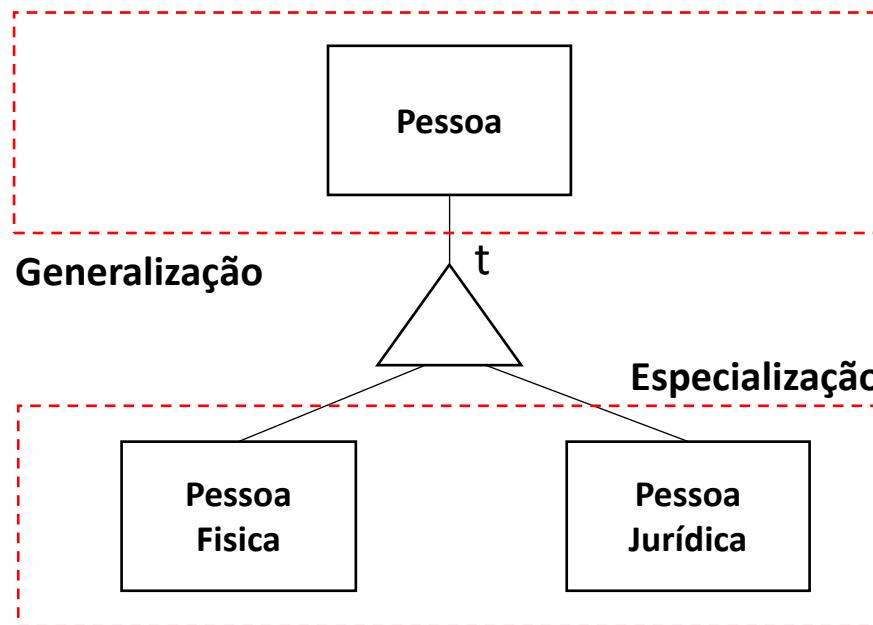


# Modelo Conceitual

## Modelo Entidade Relacionamento – MER (Relacionamentos)

### GENERALIZAÇÃO ESPECIALIZAÇÃO

- **Total:** Para cada ocorrência de entidade genérica obrigatoriamente tenho pelo menos uma ocorrência de especializada (t).
- **Parcial:** Nem toda a ocorrência da entidade genérica possui correspondência em uma entidade especializada (p).



# Modelo Entidade Relacionamento – DER

## Exercício

**Sistema de seguros de automóveis:** Cada cliente possui CPF, nome, sexo, endereço e telefones de contato (celular e fixo). Os carros possuem uma placa, marca, modelo, ano, chassi e cor. Cada carro tem um número de sinistros de acidentes associados a ele, sabendo que pode ter ocorrido múltiplos acidentes ou nenhum. Já os sinistros devem ser identificados por um código único, data e hora de ocorrência, local de ocorrência e condutor (que pode ou não ser titular da apólice). Um cliente pode possuir várias apólices (mínimo uma) vigente ou não, e cada apólice de seguro tem um identificador único e só pertence a um cliente e somente um carro, e tem data de inicio e fim da vigência, valor total assegurado, valor franquia associados a ela. É importante saber que o carro pode ter várias apólices vinculadas a ele, mas apenas uma vigente.

- Identificar e nomear:
  - Entidade
  - Atributos
  - Relacionamentos
  - Cardinalidade
- Desenhar o DER

# Modelo Entidade Relacionamento – DER

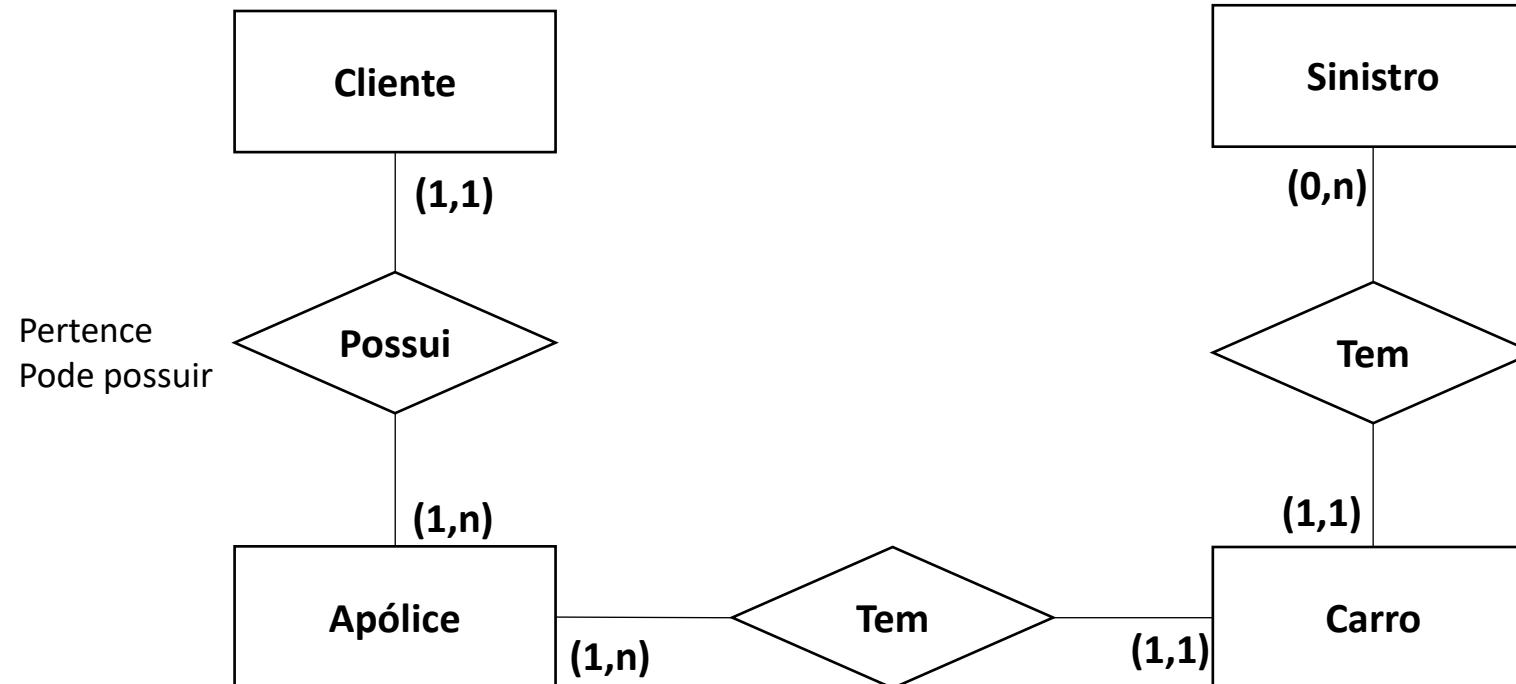
## Resolução

**Sistema de seguros de automóveis:** Cada cliente possui CPF, nome, sexo, endereço e telefones de contato (celular e fixo). Os carros possuem uma placa, marca, modelo, ano, chassi e cor. Cada carro tem um número de sinistros de acidentes associados a ele, sabendo que pode ter ocorrido múltiplos acidentes ou nenhum. Já os sinistros devem ser identificados por um código único, data e hora de ocorrência, local de ocorrência e condutor (que pode ou não ser titular da apólice). Um cliente pode possuir várias apólices (mínimo uma) vigente ou não, e cada apólice de seguro tem um identificador único e só pertence a um cliente e somente um carro, e tem data de inicio e fim da vigência, valor total assegurado, valor franquia associados a ela. É importante saber que o carro pode ter várias apólices vinculadas a ele, mas apenas uma vigente.



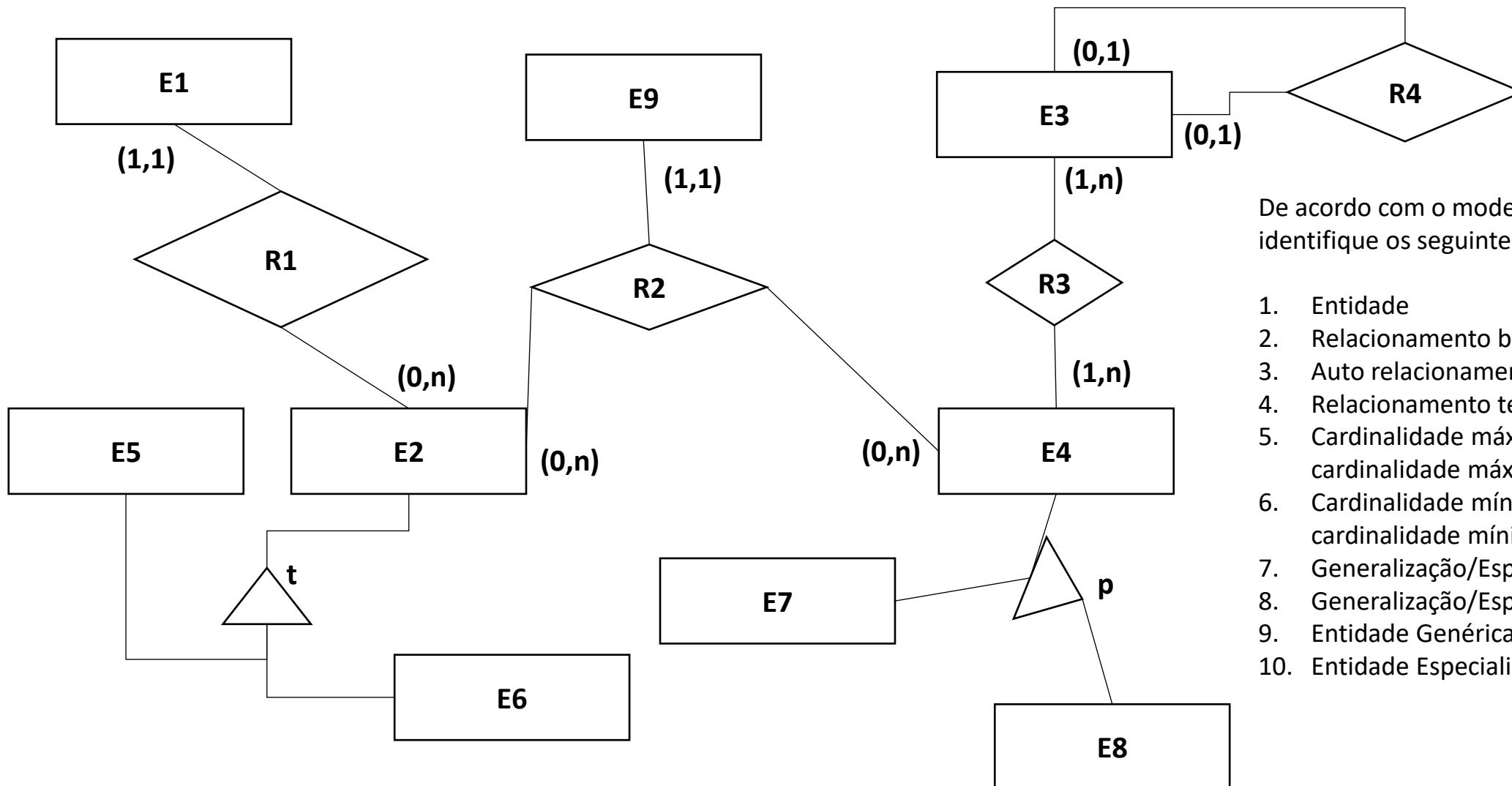
# Modelo Entidade Relacionamento – DER Resolução

Sistema de seguros de automóveis



# Modelo Entidade Relacionamento – DER

## Exercício

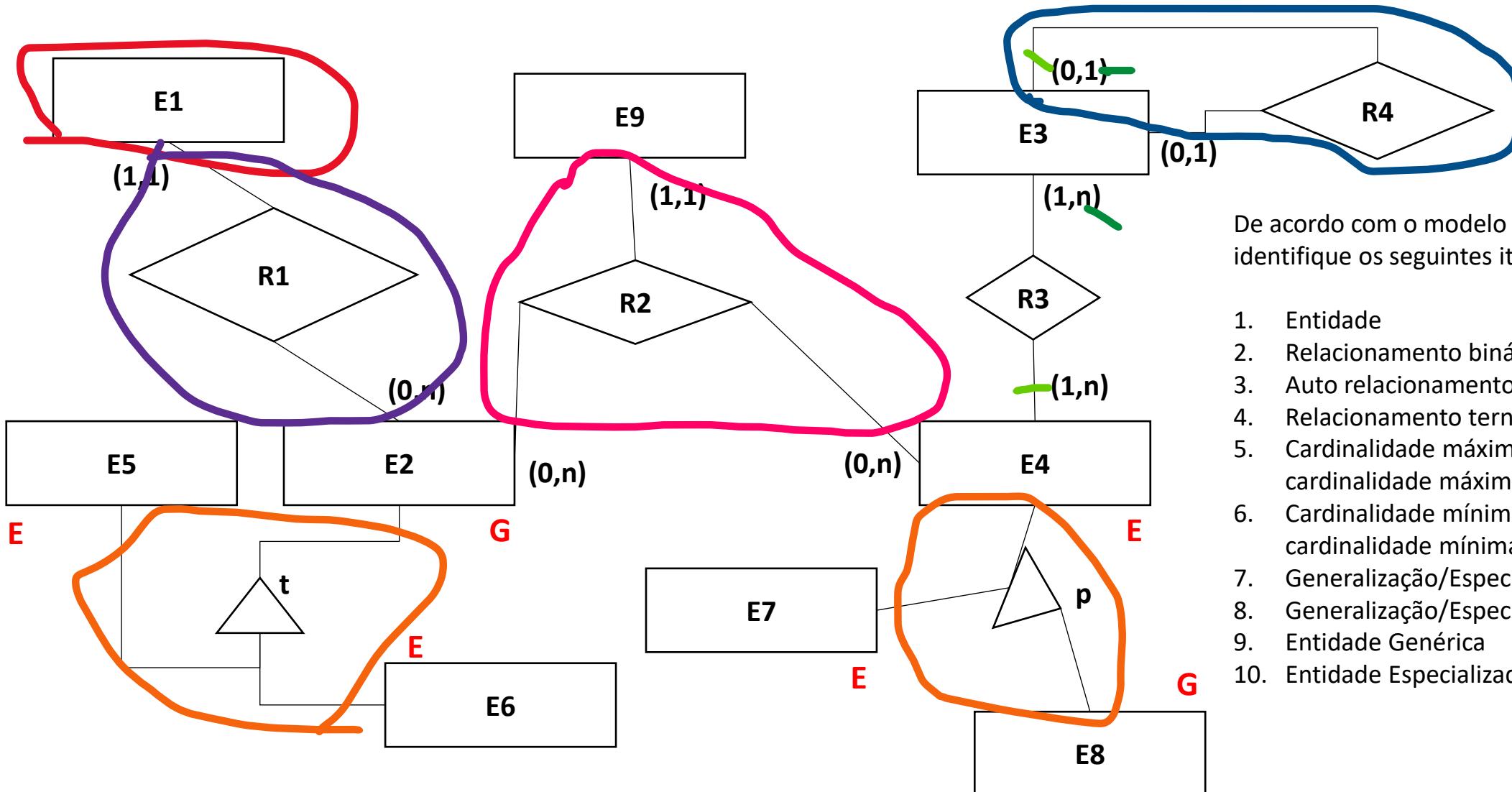


De acordo com o modelo abaixo,  
identifique os seguintes itens:

1. Entidade
2. Relacionamento binário
3. Auto relacionamento (únario)
4. Relacionamento ternário
5. Cardinalidade máxima 1 e cardinalidade máxima N
6. Cardinalidade mínima 0 e cardinalidade mínima 1
7. Generalização/Especialização total
8. Generalização/Especialização parcial
9. Entidade Genérica
10. Entidade Especializada

# Modelo Entidade Relacionamento – DER

## Resolução

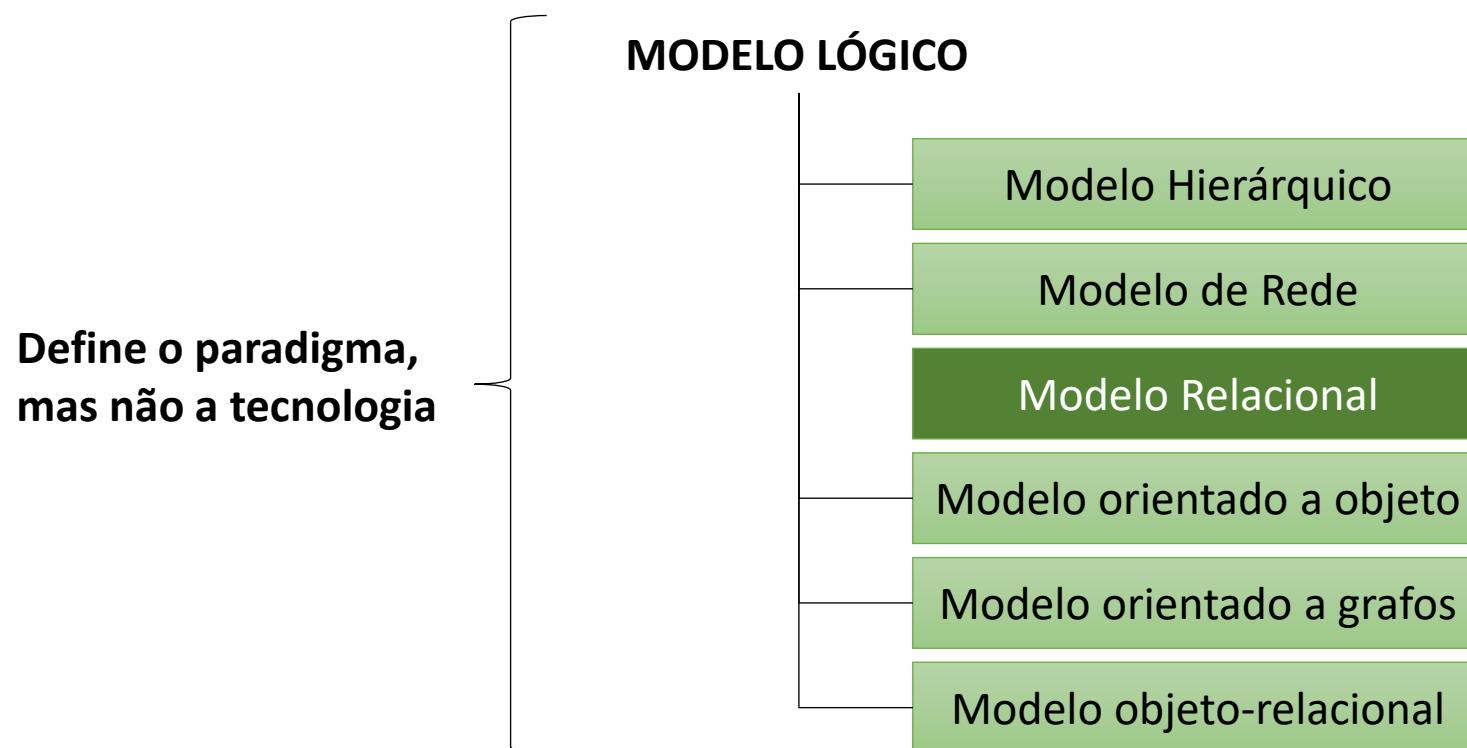


De acordo com o modelo abaixo,  
identifique os seguintes itens:

1. Entidade
2. Relacionamento binário
3. Auto relacionamento (únario)
4. Relacionamento ternário
5. Cardinalidade máxima 1 e cardinalidade máxima N
6. Cardinalidade mínima 0 e cardinalidade mínima 1
7. Generalização/Especialização total
8. Generalização/Especialização parcial
9. Entidade Genérica
10. Entidade Especializada

# Modelo Lógico

O **MODELO LÓGICO** é um aprofundamento do modelo conceitual. Neste modelo ainda não se escolheu a tecnologia, mas sim o paradigma: se será relacional, orientado a objeto, hierárquico, de rede, e a lista dos possíveis atributos vinculados a cada entidade.



# Modelo Lógico

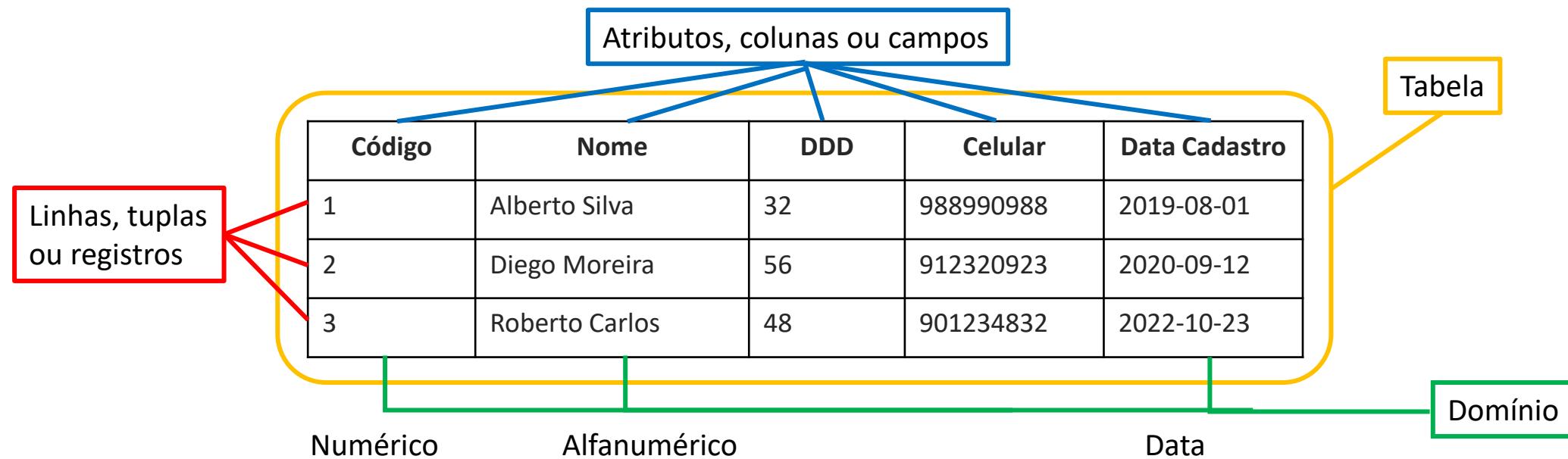
## MODELO RELACIONAL

- O **MODELO RELACIONAL** foi definido por Edgar Codd (1970).
- Representa um banco de dados com base na **teoria dos conjuntos** e seus relacionamentos.
- Em um banco de dados relacional, os dados estão organizados na forma de **tabelas**, possuem **linhas** e **colunas** e se relacionam através de **chaves**.

# Modelo Lógico

## Modelo Relacional - Tabela

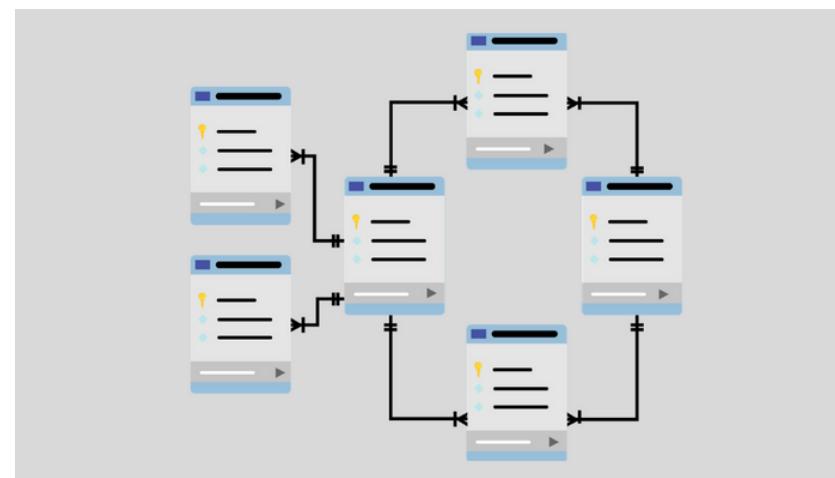
- Uma **TABELA** é um conjunto não ordenado de linhas.
- Cada **linha** pode ter um ou mais **colunas** (atributos).
- Cada **coluna** possui um único **domínio** (tipo do dado).



# Modelo Lógico

## Modelo Relacional - Chave

- Uma **CHAVE** é utilizada para **identificar linhas e estabelecer relações entre tabelas de um banco de dados** relacional.
- São estruturas que identificam **unicamente uma linha** (**chave primária** e **chave única/alternativa**) ou **promovem o relacionamento** entre as tabelas (**chave estrangeira**).
- As **tabelas** se relacionam através de **chaves**.



# Modelo Lógico

## Modelo Relacional - Chave

### **CHAVE PRIMÁRIA ou PRIMARY KEY**

- Recupera uma única linha de um conjunto de dados.
  - Único, não nulo.

Todas podem ser chave simples (uma coluna) ou chave composta (mais de uma coluna).

### **CHAVE ÚNICA/ALTERNATIVA ou UNIQUE KEY**

- Recupera uma única linha de um conjunto de dados e pode receber valores nulos.
  - Único, não nulo (ou nulo).

No entanto, o **mais comum** é ter apenas a **chave primária como composta**.

### **CHAVE ESTRANGEIRA ou FOREIGN KEY**

- Atributo chave de uma relação, cujos valores estão presente em outra tabela ligada a ela.
  - Estabelece o relacionamento entre tabelas.

# Modelo Lógico

## Modelo Relacional - Chave

### EXEMPLO DE CHAVE PRIMÁRIA E CHAVE ÚNICA / ALTERNATIVA

Código	Nome	DDD	Celular	Data Cadastro	CPF
1	Alberto Silva	32	988990988	2019-08-01	19468531578
2	Diego Moreira	56	912320923	2020-09-12	95864215874
3	Roberto Carlos	48	901234832	2022-10-23	362564197582

**Chave Primária**  
**Primary Key**  
**PK**

**Chave Única / Alternativa**  
**Unique Key**  
**UK**

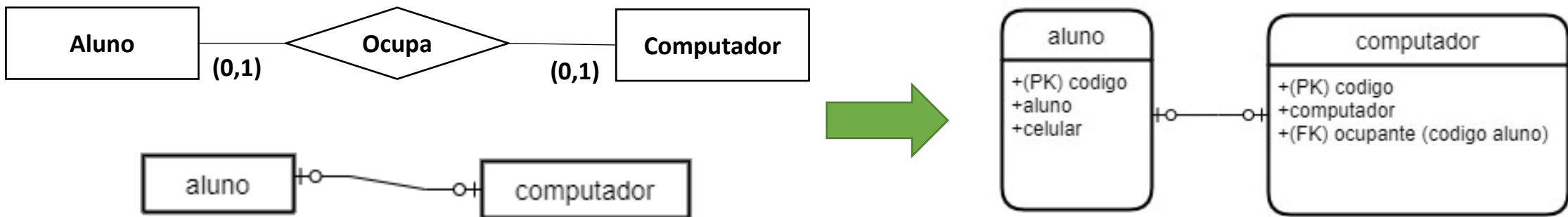
# Modelo Lógico

## Modelo Relacional - Chave

### EXEMPLO DE CHAVE ESTRANGEIRA

Código (PK)	Aluno	Celular
1	Alberto Silva	988990988
2	Diego Moreira	912320923
3	Roberto Carlos	901234832

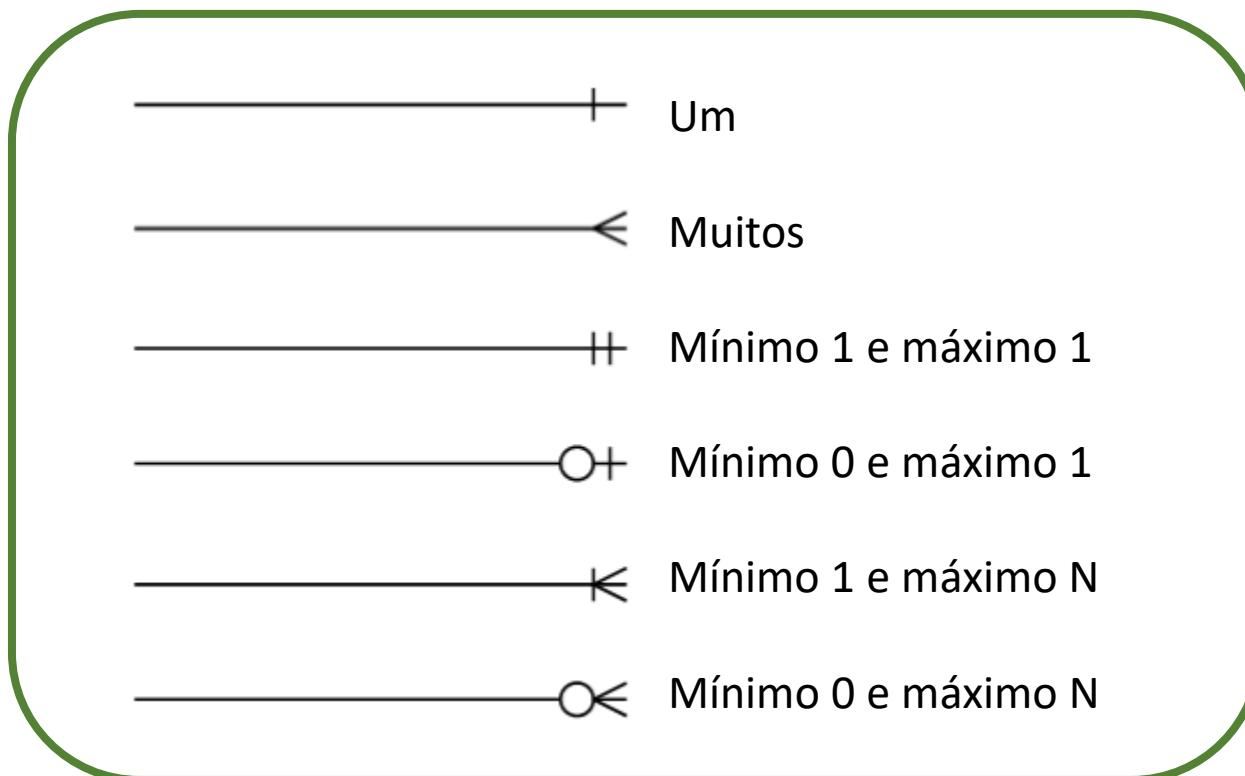
Código (PK)	Computador	Ocupante (FK) - Código Aluno -
1	Notebook 29 – Lab10.12	1
2	Desktop 18 – Lab 22.06	(nulo)
3	Notebook 33 – Lab 10.04	2



# Modelo Lógico

## Modelo Relacional - Chave

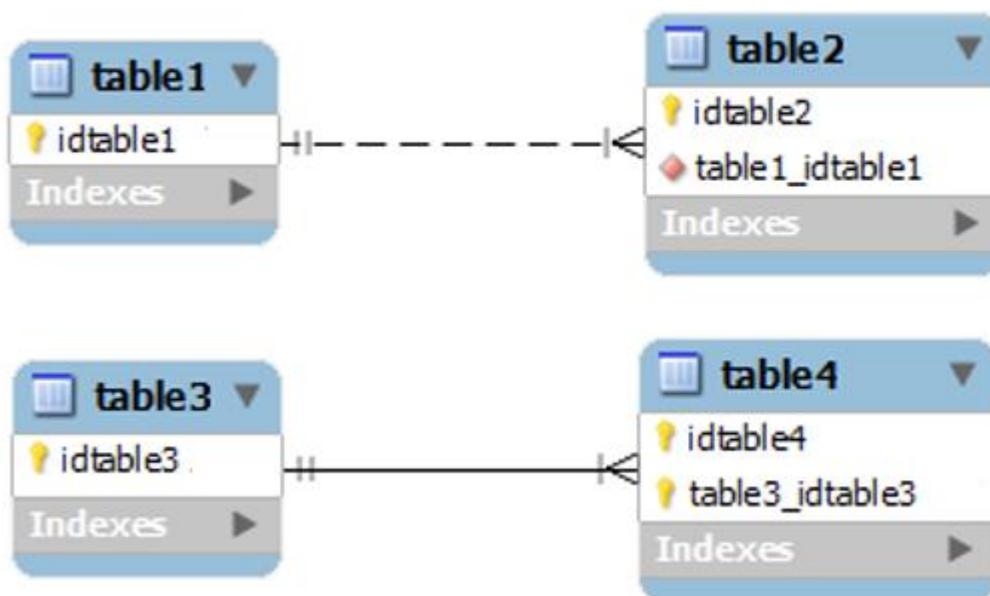
Outros exemplos de notação de relacionamentos (notação pé de galinha - engenharia da informação):



# Modelo Lógico

## Modelo Relacional - Chave

Outros exemplos de notação de relacionamentos (relacionamento tracejado e linha cheia):



Quando a estrangeira **não faz** parte da chave primária.

Quando a estrangeira **faz** parte da chave primária.

# Modelo Lógico

## Exercício

**Sistema de seguros de automóveis:** Cada cliente possui CPF, nome, sexo, endereço e telefones de contato (celular e fixo). Os carros possuem uma placa, marca, modelo, ano, chassi e cor. Cada carro tem um número de sinistros de acidentes associados a ele, sabendo que pode ter ocorrido múltiplos acidentes ou nenhum. Já os sinistros devem ser identificados por um código único, data e hora de ocorrência, local de ocorrência e condutor (que pode ou não ser titular da apólice). Um cliente pode possuir várias apólices (mínimo uma) vigente ou não, e cada apólice de seguro tem um identificador único e só pertence a um cliente e somente um carro, e tem data de inicio e fim da vigência, valor total assegurado, valor franquia associados a ela. É importante saber que o carro pode ter várias apólices vinculadas a ele, mas apenas uma vigente.

- Criar o modelo lógico com o nome dos atributos, relacionamentos e cardinalidades, utilizando a notação pé de galinha (engenharia da informação).

*Você pode utilizar o draw.io para construção do modelo ou desenhar direto em tablet/iPad.*

<https://app.diagrams.net/>

# Modelo Lógico

## Restrições de Integridade (Constraints)

**RESTRIÇÕES DE INTEGRIDADE OU CONSTRAINTS** são regras criadas para garantir a integridade dos dados de uma tabela, ou seja, garantir que as informações representem corretamente a realidade modelada.

### Integridade de Chave

Define que os valores não podem se repetir e nem nulos.

### Integridade de Domínio

Define o conjunto de valores possíveis ou permitidos que um campo pode ter.  
Podem pré-definidos ou definidos pelo usuário.

### Integridade de vazio

Define que um campo pode ou não receber valor NULL (vazio).

### Integridade Referencial

Define que os valores que aparecem em uma FK devem obrigatoriamente aparecer na PK da tabela referenciada.

# Modelo Lógico

## Normalização

- **NORMALIZAÇÃO** é o processo que visa organizar os dados em um banco de dados.
- Este processo inclui **regras** para criação de tabelas e seus relacionamentos, a fim de **proteger os dados** e tornar o banco de dados mais **flexível**, eliminando a **redundância** e a **dependência inconsistente**.
- Basicamente, é uma regra que deve ser estabelecida por uma tabela para que seja considerada “bem projetada”.
- Existem **6 (seis) regras** para normalização, mas normalmente se trabalha com **as 3 (três) primeiras**. Nós essas regras de **formas normais (FN)**.
- A partir da **3<sup>a</sup> forma normal** diz-se que o banco de dados **já se encontra normalizado**.
- Elas são acumulativas. Para se estar na 2<sup>a</sup> forma normal, por exemplo, a 1<sup>a</sup> deve ter sido também aplicada, assim como para a 3<sup>a</sup>, a 1<sup>a</sup> e a 2<sup>a</sup> devem ter sido aplicadas também.

# Modelo Lógico

## Primeira Forma Normal – 1FN

- A **PRIMEIRA FORMA NORMAL (1FN)** diz que todos os atributos de uma tabela devem ser **atômicos**, baseados em domínio simples, **sem grupos ou valores repetidos**.
  - ✓ Possuir chave primária.
  - ✓ Não possuir grupos repetitivos.
  - ✓ Quebrar atributos compostos ou multivalorados em atributos atômicos.

## Modelo Lógico

### Exemplo Primeira Forma Normal – 1FN

# Quebrar atributo multivalorado

codigo	nome	telefone	endereco
1	José	35685248 985423658	Rua das Gaivotas, 796, Novo Horizonte, Cariacica, ES, 29158-106
2	Maria	39851795 985647526 934526987	Avenida Dom Orlando Chaves, 742, Ponto Nova, Várzea Grande, MT, 78116-130
3	Joaquim	36959288	Rua Lupicínio Rodrigues, 913, Vila Cachoeirinha, Cachoeirinha, RS, 94910-160

# Quebrar atributo composto

1

codigo	nome	telefone	rua	numero	bairro	cep	cidade	uf
1	José	35685248 985423658	Rua das Gaivotas	796	Novo Horizonte	29258-958	Cariacica	ES
2	Maria	39851795 985647526 934526987	Avenida Dom Orlando Chaves	742	Ponto Nova	78111-589	Várzea Grande	MT
3	Joaquim	36959288	Rua Lupicínio Rodrigues	913	Vila Cachoeirinha	94910-996	Cachoeirinha	RS

3

1 FN

# Modelo Lógico

## Exemplo Primeira Forma Normal – 1FN

codigo	nome	telefone 1	telefone 2
1	José	35685248	985423658
2	Maria	39851795	985647526 934526987
3	Joaquim	36959288	988256901

Eliminar  
atributo  
repetido

1

codigo	nome	telefone
1	José	35685248 985423658
2	Maria	39851795 985647526 934526987
3	Joaquim	36959288 988256901

Quebrar atributo  
multivalorado

2

codigo	telefone 1
1	35685248
1	985423658
2	39851795
2	985647526
2	934526987
3	36959288
3	988256901

codigo	nome
1	José
2	Maria
3	Joaquim

3

1FN

# Modelo Lógico

## Segunda Forma Normal – 2FN

- A **SEGUNDA FORMA NORMAL (2FN)** diz que um conjunto de dados deve estar na 1FN e que todos os atributos não chaves devem depender totalmente da chave primária (não podendo ter dependências parciais).
- Desta maneira, na 2FN evitamos inconsistências devido a duplicidades.
- ✓ Estar na 1FN.
- ✓ Não possuir dependências parciais da chave primária.

# Modelo Lógico

## Exemplo Segunda Forma Normal – 2FN

A diagram illustrating a primary key constraint. A blue curved arrow points from the primary key columns (`codigo locacao`, `codigo filme`) at the top left to the primary key columns (`codigo locacao`, `codigo filme`) at the bottom right. A red curved arrow points from the primary key columns at the top right to the primary key columns at the bottom left. A yellow circle with the number 1 is positioned above the primary key columns at the top.

codigo locacao	codigo filme	titulo	data devolucao	codigo cliente
2022001	504	Os Goonies	2022-08-26	1
2022002	603	A rede social	2022-08-30	2
2022003	504	Os Goonies	2022-09-02	3

Atributo não chave,  
NÃO depende  
unicamente da PK

A diagram illustrating the decomposition of the original table into two tables. A yellow circle with the number 2 is positioned above the `codigo filme` and `titulo` columns in the first table. A yellow circle with the number 3 is positioned above the `codigo locacao` and `codigo filme` columns in the second table. A red curved arrow points from the `codigo filme` and `titulo` columns in the first table to the `codigo filme` column in the second table, indicating a foreign key relationship.

codigo filme	titulo
504	Os Goonies
603	A rede social

codigo locacao	codigo filme	data devolucao	codigo cliente
2022001	504	2022-08-26	1
2022002	603	2022-08-30	2
2022003	504	2022-09-02	3

2FN

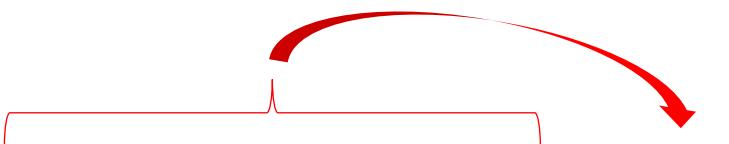
# Modelo Lógico

## Terceira Forma Normal – 3FN

- A **TERCEIRA FORMA NORMAL (3FN)** diz que um conjunto de dados deve estar na 1FN E 2FN os atributos não chave devem ser mutuamente independentes e dependentes unicamente e exclusivamente da chave primária.
  - ✓ Estar na 1FN e 2FN;
  - ✓ Garantir que todos os atributos não chave NÃO sejam funcionalmente dependentes de outros atributos não chave.

# Modelo Lógico

## Exemplo Terceira Forma Normal – 3FN



numero pedido	codigo pedido	quantidade	valor unitario	valor total
13225	PE-523	4	15,30	61,2
13226	PE-102	3	200,50	601,5
13227	PE-685	2	1050,12	2100,24
13228	PE-996	8	547,34	4378,72

1

Atributo não chave,  
depende de outro(s)  
atributos não chave  
e não somente da PK

numero pedido	codigo pedido	quantidade	valor unitario
13225	PE-523	4	15,30
13226	PE-102	3	200,50
13227	PE-685	2	1050,12
13228	PE-996	8	547,34

2

3FN

# Modelo Lógico

## Exercício Formas Normais

**Exercício:** Utilizando o Excel, **normalize** a tabela abaixo, com base no conteúdo passado.

*Um exemplo da tabela abaixo, em Excel, se encontra no AVA.*

cod cliente	nome cliente	tel 1	tel 2	endereco	cod produto	nome produto	cod fabricante	nome fabricante	preco produto	quantidade produto	Total
1	Sueli Mariane Vitória	985632588	36589885	Rua Ursa Menor, 189 - Criciuma - SC	3000	Caneta	300	Milar	50,00	2	100,00
1	Sueli Mariane Vitória	985632588	36589885	Rua Ursa Menor, 189 - Criciuma - SC	3411	Lápis	400	Clemu	150,00	3	450,00
1	Sueli Mariane Vitória	985632588	36589885	Rua Ursa Menor, 189 - Criciuma - SC	8000	Borracha	300	Milar	40,00	7	280,00
2	Luiz Jorge da Cunha	988010001 999965887	40045889	Avenida da Esperança, 10, Içara - SC	3411	Lápis	500	Fabli	150,00	5	750,00
3	Cristiane Pietra	999975252	30095877 39658745	Praça da Rota do Sol, 985 - Meleiro - SC	4522	Apontador	100	Tous	70,00	2	140,00
1	Sueli Mariane Vitória	985632588	36589885	Rua Ursa Menor, 189 - Criciuma - SC	3000	Caneta	100	Tous	50,00	4	200,00

# Modelo Físico

## Banco de dados

Um **BANCO DE DADOS** é uma coleção de dados inter-relacionados, representando informações sobre um domínio específico.

**Exemplo:** Lista telefônica, ficha de acervo de uma biblioteca, etc...



# Modelo Físico

## Sistema Gerenciador de Banco de Dados

Um **SISTEMA GERENCIADOR DE BANCO DE DADOS (SGBD ou DBMS)** é um conjunto de programas que permitem armazenar, modificar e extrair informação de um Banco de dados.

“Sistema responsável pelo gerenciamento de uma base de dados.”

Permite isolar o usuário ou aplicação da estrutura interna dos dados

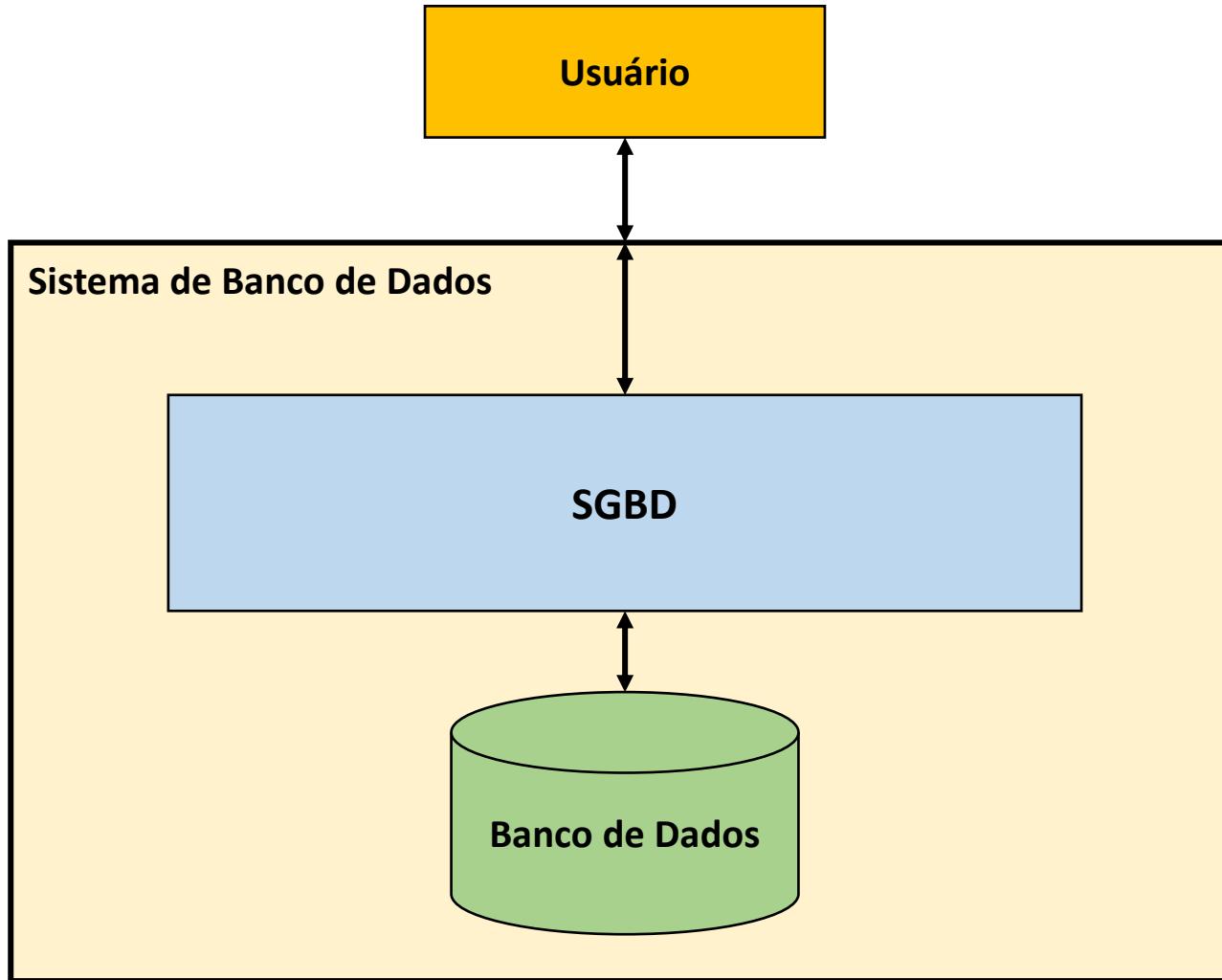
Permite o acesso e manipulação de dados: inserir, selecionar, atualizar, apagar, criar, etc

Permite o controle de permissão

Possui algoritmos para garantir performance

# Modelo Físico

## Sistema Gerenciador de Banco de Dados



# Modelo Físico SGBDs Relacionais

- ✓ Armazena dados estruturados.
- ✓ Implementam o paradigma relacional e seus elementos: tabelas, linhas, colunas e relacionamentos.
- ✓ Utiliza a linguagem SQL (Structure Query Language – linguagem de consulta estruturada) para manipulação de dados.
- ✓ Além de tabelas possuem procedimentos armazenados, visões, gatilhos, funções, índices e outros.
- ✓ Implementam o ACID:
  - Atomicidade
  - Consistência
  - Isolamento
  - Durabilidade

# Modelo Físico

## SGBDs Relacionais - ACID



# Modelo Físico

## Exemplo de GBDs Relacionais

**ORACLE®**



PostgreSQL



**IBM** DB2®



**SYBASE®**

An **SAP** Company

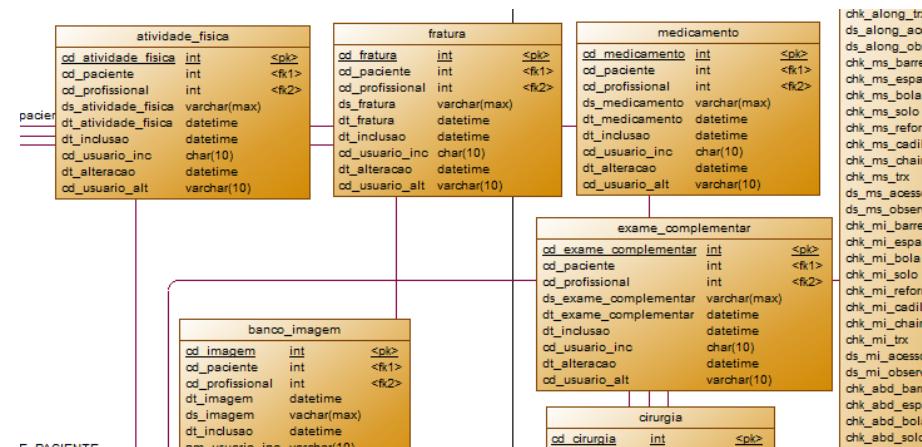


# Modelo Físico

No **MODELO FÍSICO** é onde definimos o SGBD que será utilizado e suas restrições.

É necessário detalhar os componentes da estrutura física do banco de dados no modelo relacional: tabelas, colunas, tipo de dados, visões, índices, etc.

O modelo físico definido para um SGBD não é o mesmo definido para outro SGBD, por exemplo, o modelo físico gerado para o SGBD Oracle é diferente do modelo físico gerado para o SGBD SQL Server.



# Modelo Físico

O **MODELO FÍSICO RELACIONAL** em geral é descrito por um script SQL (Structure Query Language).

Define objetos como:

- Tabelas, índices, procedimentos armazenados, visões, funções, etc.
- Relacionamentos através das chaves.
- Restrições (Constraints).
- Colunas e seus tipos de dados.

Implementa as especificidades de cada SGBD:

- Auto incremento, sequência.
- Tipos de dados podem variar.

Os **TIPOS DE DADOS** mais comuns são:

- **Numéricos exatos** (int, numeric, bit)
- **Numéricos aproximados** (float)
- **Data e hora** (date, time, datetime)
- **Caractere** (char, varchar)
- **Binário** (binary)

# Comparativo sobre os modelos de dados

## Conceitual

- ✓ Abstrato.
- ✓ Independente da tecnologia ou paradigma.
- ✓ DER.
- ✓ Conceitos:  
Entidades que se relacionam entre si.

## Lógico

- ✓ Ainda independente da tecnologia.
- ✓ Fase em que decidimos o paradigma / abordagem do modelo:  
hierárquico, orientado a objeto, relacional, não relacional.

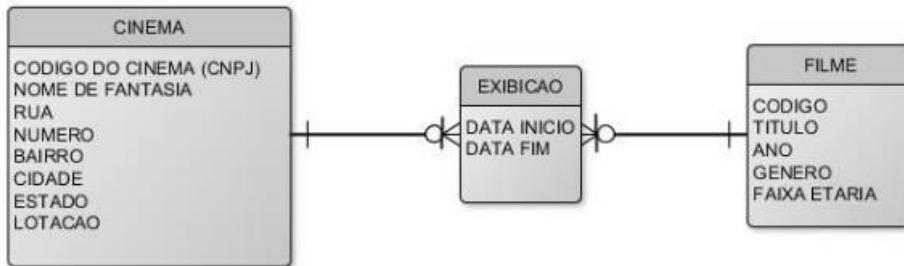
## Físico

- ✓ Escolha do SGBD
- ✓ Vinculado as restrições e características do SGBD escolhido.
- ✓ Totalmente dependente da tecnologia escolhida

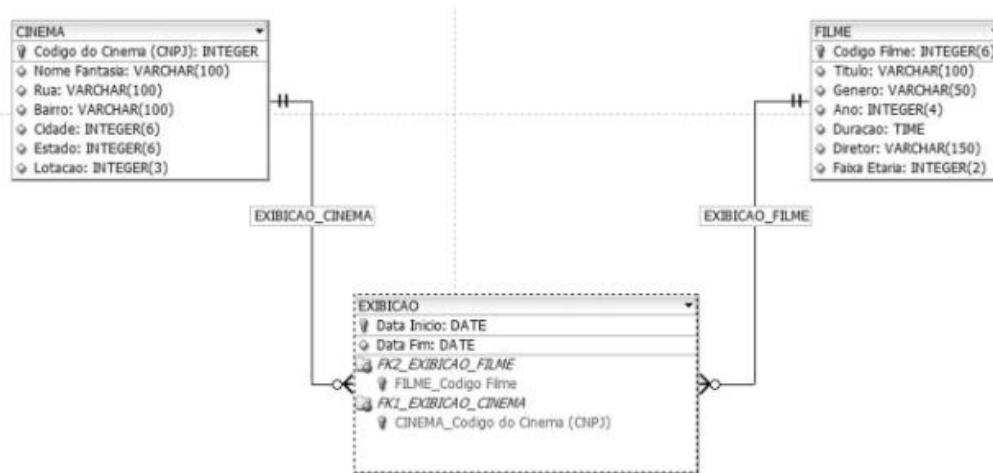
# Comparativo sobre os modelos de dados



**Modelo Conceitual**



**Modelo Lógico**



**Modelo Físico**

# Modelo Físico

## EXERCÍCIO

**Sistema de seguros de automóveis:** Cada cliente possui CPF, nome, sexo, endereço e telefones de contato (celular e fixo). Os carros possuem uma placa, marca, modelo, ano, chassi e cor. Cada carro tem um número de sinistros de acidentes associados a ele, sabendo que pode ter ocorrido múltiplos acidentes ou nenhum. Já os sinistros devem ser identificados por um código único, data e hora de ocorrência, local de ocorrência e condutor (que pode ou não ser titular da apólice). Um cliente pode possuir várias apólices (mínimo uma) vigente ou não, e cada apólice de seguro tem um identificador único e só pertence a um cliente e somente um carro, e tem data de inicio e fim da vigência, valor total assegurado, valor franquia associados a ela. É importante saber que o carro pode ter várias apólices vinculadas a ele, mas apenas uma vigente.

- Criar o modelo físico com o nome dos atributos, relacionamentos e cardinalidade.
- Gerar o script e visualizar os comando SQL gerados a partir do modelo físico.

*Você pode utilizar o MySQL WorkBench.*

# Linguagem SQL

**SQL** é a linguagem padrão para bancos de dados relacionais.

A sigla SQL significa Structure Query Language ou Linguagem de Consulta Estruturada

Criada por volta dos anos 70 e padronizada em 1986

Até 1986 existiam alguns tipos de dialetos sobre SQL

Padrão ANSI e ISO – SQL-92\*

Baseado na álgebra relacional

# Linguagem SQL

## GRUPOS DE SQL

**DDL – Data Definition Language (Linguagem de Definição de Dados):** Responsáveis por criar ou apagar objetos (CREATE/DROP), alterar o esquema, estrutura ou tipo dos objetos de banco, habilitar/desabilitar objetos (ALTER).

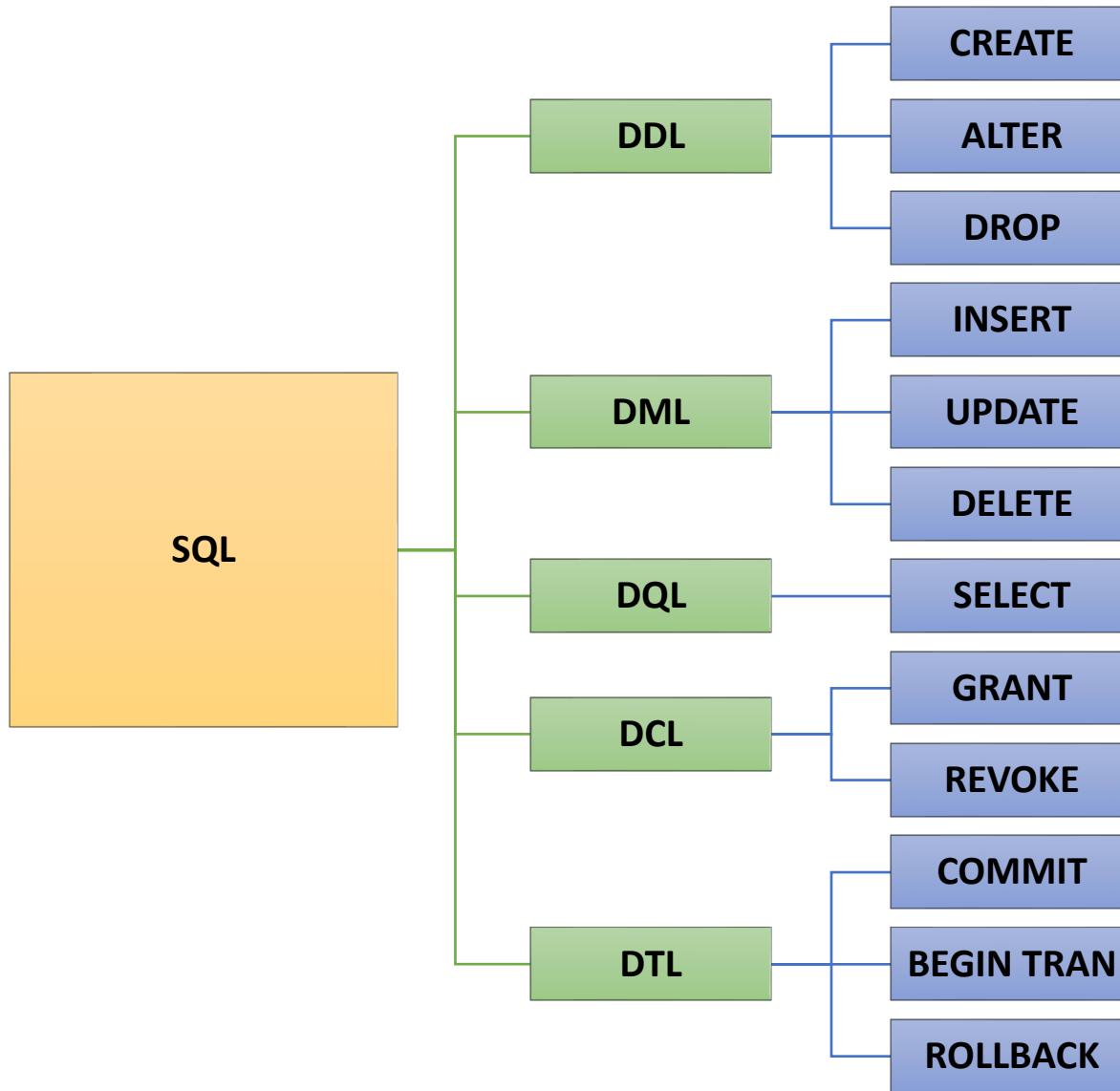
**DML - Data Manipulation Language (Linguagem de Manipulação de Dados):** Usados para modificar os dados, as linhas de uma tabela – INSERT, UPDATE, DELETE.

**DQL - Data Query Language (Linguagem de Consulta de Dados):** Usados para consultar os dados (SELECT).

**DCL - Data Control Language (Linguagem de Controle de Dados):** Utilizados para controlar o acesso aos dados. Através deles podemos conceder (GRANT) ou negar (REVOKE). Alguns SGBDs também possuem o DENY.

**DTL - Data Transaction Language (Linguagem de Transação de Dados):** Usados para controle da transação: BEGIN TRANSACTION, COMMIT E ROLLBACK.

# Linguagem SQL



# Linguagem SQL

## EXTENSÕES SQL

**T-SQL** – Microsoft SQL Server

**Transact-SQL** – Sybase ASE

**PL/SQL** – Oracle

**PL/pgSQL** – PostgreSQL

**SQL PL** – IBM DB2

# Linguagem SQL

## RANKING DOS BANCOS DE DADOS

<https://db-engines.com/en/ranking>

# Linguagem SQL

## OBJETOS DE UM BANCO DE DADOS RELACIONAL

**Visão / View:** Objeto que encapsula um ou mais comandos de SELECT.

**Índice / Index:** Chave utilizada para fins de performance.

**Procedimento Armazenado / Stored Procedure:** Objeto que possui um código procedural, podendo ter vários comandos, blocos de repetição, laço, parâmetros de entrada e saída, etc.

**Função / Function:** Similar ao procedimento armazenado, mas com a diferença de sempre retornar um valor ou um conjunto de dados.

**Gatilho / Trigger:** Procedimento armazenado que é disparado a partir de eventos especiais em tabelas ou banco de dados.

**Usuários / Logins / Users:** São os que se conectam e interagem com os bancos de dados dentro de um SGBD. Podemos conceder ou revogar permissões.

**Roles:** Agrupamento de permissões e pode ser concedida a usuários ou outras roles.

# Tipo de dados

Tipo de Dados
char(n)
varchar(n)
varchar(max)
text
nchar
nvarchar
nvarchar(max)
ntext
binary(n)
varbinary
varbinary(max)
image

Tipo de Dados	Descrição	Tamanho Máximo	Tamanho (bytes)
char(n)	Tamanho fixo, completado com espaços em bracos	8.000 caracteres	Tamanho Definido
varchar(n)	Tamanho variável com limite	8.000 caracteres	2 bytes + número de caracteres
varchar(max)	Tamanho variável com limite	1.073.741.824 caracteres	2 bytes + número de caracteres
text	Tamanho variável	2GB de dados (texto)	4 bytes + número de caracteres
nchar	Tamanho fixo com espaços em bracos	4.000 caracteres	Tamanho definido x 2
nvarchar	Tamanho variável	4.000 caracteres	
nvarchar(max)	Tamanho variável	536.870.912 caracteres	
ntext	Tamanho variável	2GB de texto	
binary(n)	Tamanho fixo (binário)	8.000 bytes	
varbinary	Tamanho variável (binário)	8.000 bytes	
varbinary(max)	Tamanho variável (binário)	2GB	
image	Tamanho variável (binário)	2GB	

Tipo de Dado	Descrição	Tamanho (bytes)
datetime	De 1 de janeiro de 1753 a 31 de dezembro de 9999 com uma precisão de 3,33 milisegundos	8 bytes
datetime2	De 1º de Janeiro de 0001 a 31 de dezembro de 9999 com precisão de 100 nanosegundos	6-8 bytes
smalldatetime	De 1 de Janeiro de 1900 a 6 de junho de 2079 com precisão de 1 minuto	4 bytes
date	Armazena apenas uma data. De 1 de Janeiro de 0001 a 31 de dezembro de 9999	3 bytes
time	Armazena um tempo apenas para uma precisão de 100 nanosegundos	3-5 bytes
datetimeoffset	O mesmo que datetime2 com a adição de um deslocamento de fuso horário	8-10 bytes
timestamp	Armazena um número único que é atualizado sempre que uma linha é criada ou modificada. O valor do timestamp é baseado em um relógio interno e não corresponde ao tempo real. Cada tabela pode ter apenas uma variável timestamp	

Tipo de Dado	Descrição	Tamanho (bytes)
bit	Número Inteiro que pode ser 0, 1 ou NULL	1 byte
tinyint	Permite números inteiros de 0 a 255	2 bytes
smallint	Permite números inteiros entre -32.768 e 32.767	4 bytes
int	Permite números inteiros entre -2.147.483.648 e 2.147.483.647	4 bytes
bigint	Permite números inteiros entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807	8 bytes
decimal(p,s)	Precisão de número flutuante e número de escala. Permite número de $-10^{38} + 1$ a $10^{38} - 1$ .  O parâmetro p indica o número total máximo de dígitos que podem ser armazenados (ambos à esquerda e à direita do ponto decimal). p deve ser um valor de 1 a 38. O padrão é 18.  O parâmetro s indica o número máximo de dígitos armazenados à direita do ponto decimal. s deve ser um valor de 0 a p. O valor padrão é 0.	5-17 bytes
numeric(p,s)	Precisão de número flutuante e número de escala. Permite número de $-10^{38} + 1$ a $10^{38} - 1$ .  O parâmetro p indica o número total máximo de dígitos que podem ser armazenados (ambos à esquerda e à direita do ponto decimal). p deve ser um valor de 1 a 38. O padrão é 18.  O parâmetro s indica o número máximo de dígitos armazenados à direita do ponto decimal. s deve ser um valor de 0 a p. O valor padrão é 0	5-17 bytes
smallmoney	Tipo de "Moeda" de -214.748.3648 a 214.748.3647	4 bytes
money	Tipo de "Moeda" de -922.337.203.685.477.5808 a 922.337.203.685.477.5807	8 bytes
float(n)	Precisão de número flutuante de $-1.79E + 308$ a $1.79E + 308$ . O parâmetro n indica se o campo deve conter 4 ou 8 bytes. float(24) contém um campo de 4 bytes e o float(53) mantém um campo de 8 bytes. O valor padrão de n é 53.	4 ou 8 bytes
real	Precisão de número flutuante de $-3.40E + 38$ a $3.40E + 38$	4 bytes

Tipo de Dado	Descrição
sql_variant	Armazena até 8.000 bytes de dados de vários tipos de dados, exceto text, ntext e timestamp
uniqueidentifier	Armazena um identificador globalmente exclusivo (GUID)
xml	Armazena dados formatados em XML. Máximo de 2GB
cursor	Armazena uma referência a um cursor usado para operações de banco de dados
table	Armazena um conjunto de resultados para processamento posterior

# Tipo de dados

# Tipo de dados

## TIPOS DE DADOS MAIS COMUNS

INTEGER

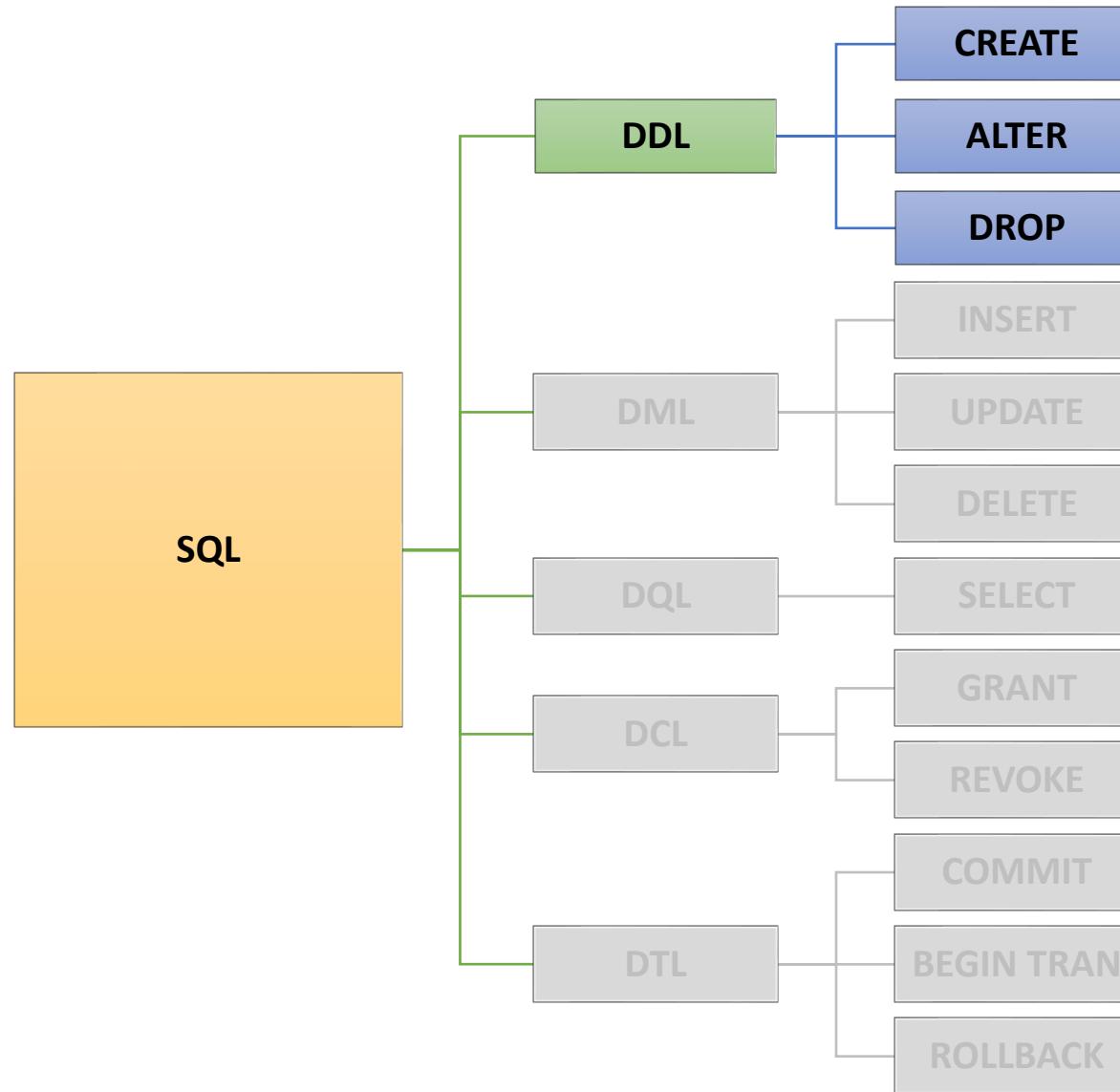
NUMBER

DATETIME

CHAR

VARCHAR

# Grupo DDL



# Criando uma tabela

## SINTAXE

```
CREATE TABLE table_name (
    column_name datatype [ NULL | NOT NULL ] ,
    column_name datatype [ NULL | NOT NULL ] , ...
    column_name datatype [ NULL | NOT NULL ] )
```

# Criando uma tabela

## EXEMPLO

```
CREATE TABLE pessoa (
    cd_coop int NOT NULL,
    cd_posto int NOT NULL,
    cd_pessoa int NOT NULL,
    cd_tp_pessoa numeric(3,0) NOT NULL,
    id_fis_jur char(1) NOT NULL,
    nm_pessoa char(65) NOT NULL,
    sc_pessoa char(26) NOT NULL )
go
```

# Excluindo uma tabela

## SINTAXE

```
DROP TABLE table_name  
go
```

## EXEMPLO

```
DROP TABLE pessoa  
go
```

# Alterando uma tabela

## SINTAXE SIMPLIFICADA DA ALTERAÇÃO

```
ALTER TABLE table_name
{ ADD column_name datatype { NULL | default = value} [, ...]
| DROP column_name [, ...]
| ALTER COLUMN column_name datatype { NULL | NOT NULL} [, ...] }
[, ...]
```

# Alterando uma tabela

## SINTAXE

```
ALTER TABLE pessoa
    ADD ds_endereco char(60) NULL,
    DROP cd_posto,
    ALTER COLUMN sc_pessoa char(30)
go
```

# Visualizando a definição de uma tabela

## SINTAXE SIMPLIFICADA

```
sp_help [object_name]
```

## EXEMPLO

```
sp_help pessoa  
go
```

# Renomeando tabelas

## SINTAXE SIMPLIFICADA

```
sp_rename old_table, new_table
```

## EXEMPLO

```
sp_rename pessoa, pessoa_bkp  
go
```

# Renomeando colunas

## SINTAXE SIMPLIFICADA

```
sp_rename "table.column" , new_column
```

## EXEMPLO

```
sp_rename "pessoa.cd_pessoal", pessoa  
go
```

# Tabelas temporárias

## SINTAXE SIMPLIFICADA

```
CREATE TABLE #table_name (
    column_name datatype [ NULL | NOT NULL ] ,
    column_name datatype [ NULL | NOT NULL ] ,...
    column_name datatype [ NULL | NOT NULL ] )
```

## EXEMPLO

```
create table #tabela_teste
(id_aluno    integer,
nm_aluno    char(10),
nm_cidade   char(15))
go
```

# Tabelas temporárias

## **TEMPORÁRIA LOCAL (SESSÃO)**

```
CREATE TABLE #table_name ...
```

## **TEMPORÁRIA GLOBAL**

```
create table tempdb..tabela_teste...
```

# Truncate Table

## SINTAXE SIMPLIFICADA

```
TRUNCATE TABLE table_name
```

## EXEMPLO

```
TRUNCATE TABLE cidade  
go
```

# Default

## SINTAXE SIMPLIFICADA

```
CREATE TABLE table_name (
    column_name datatype [ DEFAULT ] [ NULL | NOT NULL ],
    ...
```

## EXEMPLO

```
create table aluno
(id_aluno    integer,
nm_aluno    char(10),
situacao   char(1) DEFAULT 'S' NOT NULL)
go
```

# Constraints

- **PRIMARY KEY**
- **CHECK**
- **FOREIGN KEY**

# Constraint

## PRIMARY KEY

### SINTAXE SIMPLIFICADA

```
CREATE TABLE table_name (
    column_name datatype [ NULL | NOT NULL ] , ...
    CONSTRAINT constraint_name PRIMARY KEY (column_name))
go
```

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name PRIMARY KEY (column_name)
go
```

# Constraint

## PRIMARY KEY

### EXEMPLOS

```
CREATE TABLE pessoa (
    id_pessoa INT NOT NULL,
    nm_pessoa VARCHAR(100) NOT NULL,
    CONSTRAINT pk_pessoa PRIMARY KEY (id_pessoa))
go
```

```
ALTER TABLE pessoa
ADD CONSTRAINT pk_pessoa PRIMARY KEY (id_pessoa)
go
```

# Constraint CHECK

## SINTAXE SIMPLIFICADA

```
CREATE TABLE table_name (
    col datatype NOT NULL CONSTRAINT constraint_name CHECK
        (search_conditions),...
)
go
```

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name CHECK (search_conditions)
go
```

# Constraint CHECK

## EXEMPLOS

```
CREATE TABLE tabela1 (
    col1 int NOT NULL,
    col2 char(2) NOT NULL CONSTRAINT chk_col2 CHECK (col2 IN ('S','N'))
)
go
```

```
ALTER TABLE tabela1
ADD CONSTRAINT chk_test CHECK (col1 IN (1, 2))
go
```

# Constraint FOREIGN KEY

## SINTAXE SIMPLIFICADA

```
ALTER TABLE table_name_child
    ADD CONSTRAINT constraint_name foreign key (col)
        references table_name_father (col)
go
```

# Constraint FOREIGN KEY

## EXEMPLO

```
ALTER TABLE computador
ADD CONSTRAINT fk_aluno_computador FOREIGN KEY (cd_aluno)
REFERENCES aluno (cd_aluno)
go
```

# Constraint

## MODIFICANDO / EXCLUINDO CONSTRAINTS

### SINTAXE SIMPLIFICADA

```
ALTER TABLE table_name_child  
    DROP CONSTRAINT constraint_name  
go
```

# Constraint

## MODIFICANDO / EXCLUINDO CONSTRAINTS

### EXEMPLO

```
ALTER TABLE computador  
DROP CONSTRAINT fk_aluno_computador  
go
```

### SINTAXE SIMPLIFICADA DE CRIAÇÃO

```
CREATE VIEW view_name  
AS  
SELECT statement
```

### EXEMPLO

```
CREATE VIEW vw_pessoa  
AS  
SELECT cd_pessoa, nm_pessoa  
FROM pessoa  
go
```

# View EXCLUSÃO

## SINTAXE SIMPLIFICADA DE EXCLUSÃO

```
DROP VIEW view_name
```

## EXEMPLO

```
DROP VIEW vw_pessoa  
go
```

# Linguagem SQL

## EXERCÍCIO

**Sistema de avaliação:** Você deverá construir um modelo de avaliação, similar ao modelo do AVA. Cada aluno deverá ter um código único, nome completo e e-mail. Um aluno poderá ser muitas avaliações (não obrigatório). Cada avaliação deverá ter um código único, tipo de avaliação (template), data de inicio + hora, data de fim + hora. Cada avaliação preenchida pelo aluno deverá seguir um template (modelo de avaliação) com perguntas, tipos de respostas (objetiva, múltipla escolha ou aberta) e as alternativas para objetiva e múltipla escolha, bem como a sinalização resposta correta (objetiva e múltipla escolha). As respostas deverão ter relação com as perguntas de cada avaliação com base nos templates (Ex. Fulano respondeu ‘a’ na questão 5 (tipo objetiva) que possuía 5 respostas possíveis e somente 1 era a correta). As perguntas devem ter um código único, código de ordem das perguntas e vinculadas ao tipo de avaliação.

- Criar o modelo físico com o nome dos atributos e relacionamentos (PKs e FKs).
- Gerar o script e visualizar os comandos SQL gerados a partir do modelo físico.

*Você pode utilizar o MySQL WorkBench para a construir o modulo físico.*

# Opções de Campo com Auto incremento

Existem **3 opções** para campos com **auto incremento** no SQL Server:

- Campo com default usando NEWID (uniqueidentifier)
- Campo Identity
- Objeto Sequence

NOTA 1: Campos com Identity e Default devem ser removidos dos comandos de INSERT.

NOTA 2: Não é possível atualizar campos identity pelo comando UPDATE.

# Opções de Campo com Auto incremento

## DEFAULT NEWID

### SINTAXE SIMPLIFICADA

```
CREATE TABLE aluno (
    id_aluno uniqueIdentifier NOT NULL default newid(), ...
)
go
```

```
ALTER TABLE aluno ADD CONSTRAINT DF_aluno DEFAULT newid() FOR
    id_aluno;
go
```

# Opções de Campo com Auto incremento

## IDENTITY

### SINTAXE SIMPLIFICADA

```
CREATE TABLE aluno(
    id_aluno int IDENTITY NOT NULL, . . .
)
go
```

```
DBCC CHECKIDENT ('aluno', RESEED, 0)
go
```

A coluna deverá ser criada já com o tipo de auto incremento. Caso ela já tenha sido criada e for necessário mudar para tipo identity, deve-se recriar a coluna com o auto incremento identity.

**Nota:** Caso a coluna seja PK, será necessário excluir a PK, recriar a coluna com identity e depois criar novamente a PK.

# Opções de Campo com Auto incremento

## SEQUENCE

### SINTAXE SIMPLIFICADA

```
CREATE SEQUENCE aluno_seq  
    AS int  
    START WITH 1  
    INCREMENT BY 1  
go  
  
DROP SEQUENCE aluno_seq;  
  
ALTER SEQUENCE seq_Teste RESTART WITH 1;  
  
CREATE TABLE aluno ( id_aluno int NOT NULL,... )  
go
```

# Opções de Campo com Auto incremento

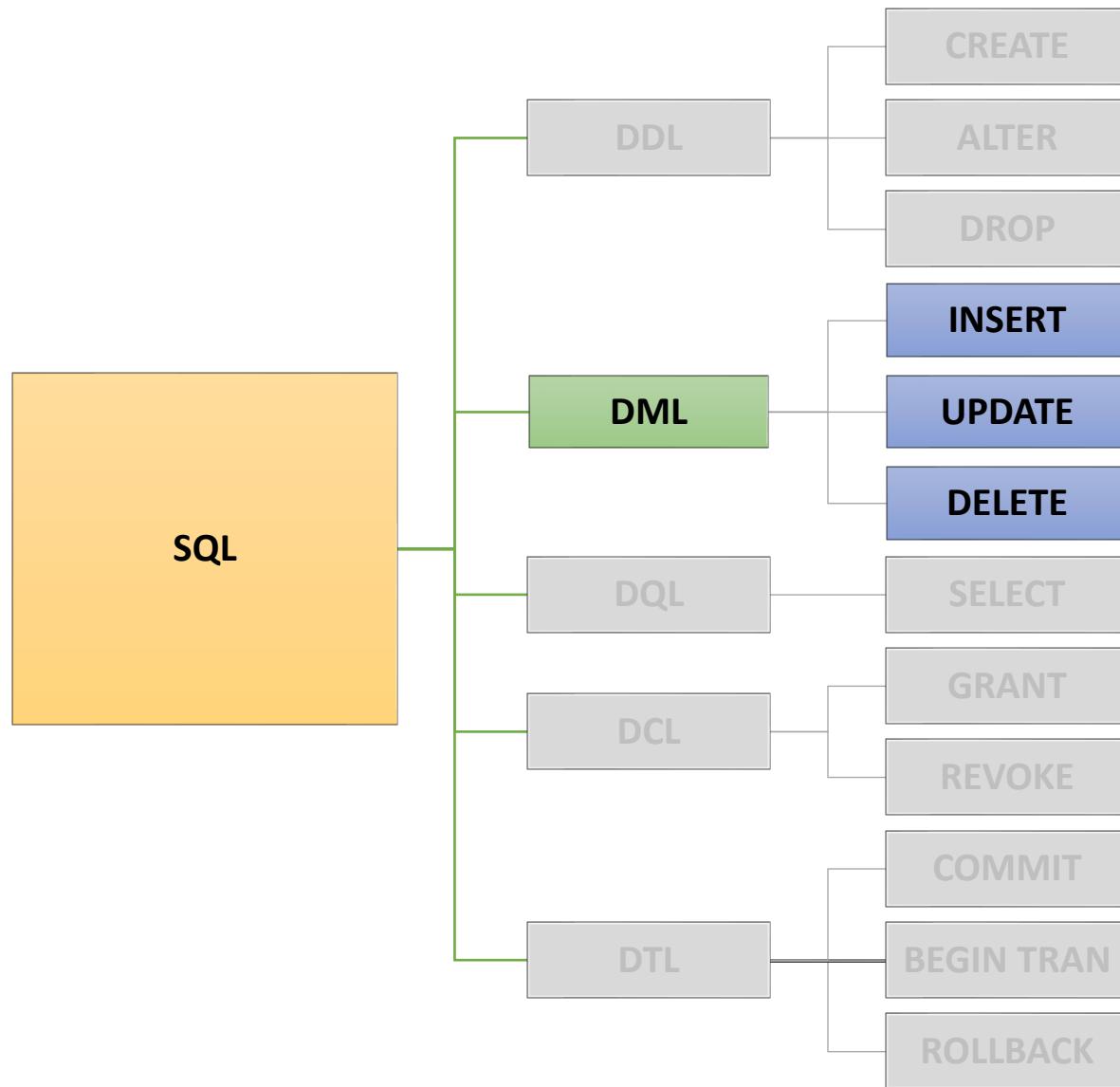
## SEQUENCE

### SINTAXE SIMPLIFICADA

```
SELECT NEXT VALUE FOR aluno_seq;
```

```
SELECT current_value FROM sys.sequences WHERE name = aluno_seq';
```

# Grupo DML



# Insert

## SINTAXE SIMPLIFICADA

```
INSERT [INTO] table_name [(column_list)]  
{values (value_list) | select statement }
```

# Insert

## EXEMPLO

```
INSERT INTO profissao (cd_profissao, ds_profissao, cd_usuario, dt_inclusao)
VALUES (1,
        'MEDICO',
        '2010-09-11',
        'ADM',
        '2007-09-11',
        NULL)
go
```

# Update

## SINTAXE SIMPLIFICADA

```
UPDATE table_name  
SET column1 = { expression | select_statement }  
[, column2 = { expression | select_statement }]  
[WHERE condition ]
```

# Update

## EXEMPLO

```
UPDATE associado  
SET qt_quotas = 154  
WHERE nr_matricula = 19  
go
```

# Delete

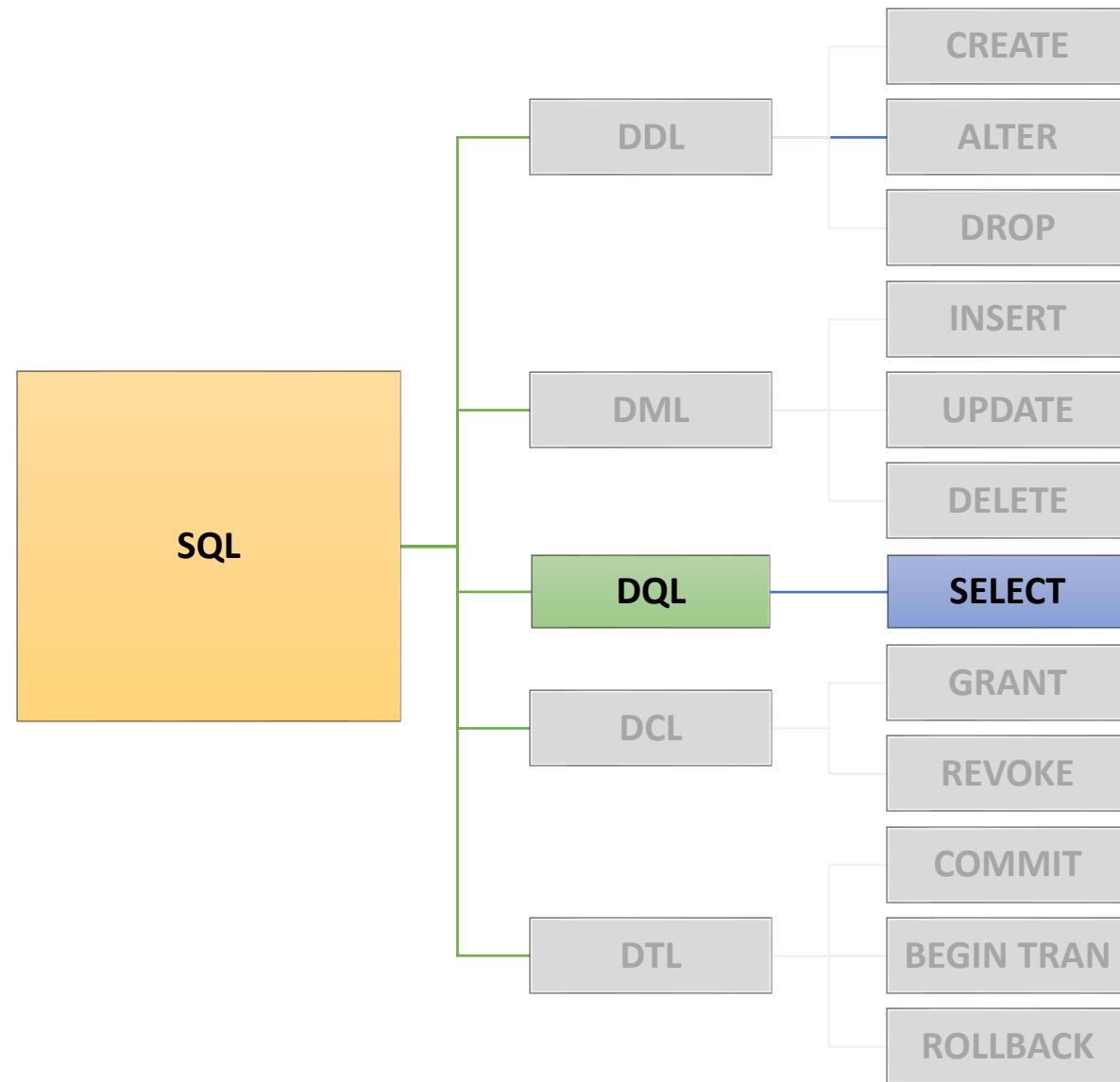
## SINTAXE SIMPLIFICADA

```
DELETE [FROM] table_name  
[WHERE condition ]
```

## EXEMPLO

```
DELETE FROM cidade  
WHERE cd_uf <> 'RS'  
go
```

# Grupo DQL



# Select

## SINTAXE BASICA SIMPLIFICADA

**SELECT** <lista de colunas / resultado visível do select>

**FROM** <conjunto de dados / tabelas>

**WHERE** <condição / filtro>

# Select

## RENOMEANDO CABEÇALHOS

### SINTAXE SIMPLIFICADA

```
SELECT column_name AS 'new_column_heading' [, ...]  
FROM ...
```

```
SELECT new_column_heading = column_name [, ...]  
FROM ...
```

```
SELECT 'new_column_heading' = column_name [, ...]  
FROM ...
```

# Select

## RENOMEANDO CABEÇALHOS

### EXEMPLOS

```
SELECT cd_pessoa AS codigo, nm_pessoa AS 'nome da pessoa'  
FROM pessoa  
go
```

```
SELECT 'Nome da pessoa' = cd_pessoa, codigo = cd_pessoa  
FROM pessoa  
go
```

# Select

## ORDENANDO COLUNAS

### SINTAXE SIMPLIFICADA

```
SELECT column_list  
FROM table_name  
ORDER BY column [ASC | DESC]  
go
```

**NOTA:** O padrão de ordenação é ASC.

# Select

## ORDENANDO COLUNAS

### EXEMPLO

```
SELECT *
FROM pessoa
ORDER BY id_fis_jur, nm_pessoa
go
```

Mostra todos os registros da tabela pessoa ordenado pelos campos id\_fis\_jur ASC e nm\_pessoa ASC.

# Select DISTINCT

É uma palavra-reservada utilizada para eliminação de valores duplicados.

## EXEMPLOS

```
SELECT DISTINCT cd_uf
FROM cidade
go
```

Mostra todos os registros distintos da tabela cidade.

# Select

## TOP

Mostra apenas um determinado conjunto de linhas do resultado da consulta.

### EXEMPLOS

```
SELECT TOP 10 cd_uf  
FROM cidade  
ORDER BY cd_uf  
go
```

Mostra os 10 primeiros registros da tabela cidade ordenados pelo código da UF ASC.

# Select

## OPERADORES ARITMÉTICOS

Os operadores aritméticos são “+” (adição), “-” (subtração), “\*” (multiplicação), “/” (divisão) e “%” (Módulo).

### EXEMPLOS

```
SELECT TOP 5 cd_conta + 10000, cd_conta * 2  
FROM conta  
ORDER BY cd_conta  
go
```

```
SELECT 38 / 5 AS ResultadoDivisao, 38 % 5 AS RestoDivisao;  
go
```

# Select

## WHERE

### SINTAXE SIMPLIFICADA

```
SELECT column_list  
FROM table  
WHERE condition
```

### EXEMPLO

```
SELECT TOP 10 dt_lancamento, vl_lancamento  
FROM lançamento_diario  
WHERE vl_lancamento > 10000  
go
```

# Select

## OPERADORES DE COMPARAÇÃO

Operador	Função
=	Igual a
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
<> ou !=	Diferente de

# Select

## VALORES NULOS (NULL)

- O usuário poderá explicitar a palavra *NULL*, sem aspas simples ou duplas;
- Um valor nulo nunca é igual a outro valor nulo;
- Algumas colunas são definidas para permitir valores nulos;
- Operações envolvendo valores nulos resultam em nulo.

**Nota I:** Lembre-se que se você usar o *NULL* com aspas simples ou duplas, ele será tratado como string cujo conteúdo é a palavra “null” e não como valor nulo.

**Nota II:** É importante lembrar que *NULL* não é sinônimo de “zero” ou “branco”.

# Select

## TRABALHANDO COM VALORES NULOS

### EXEMPLOS

```
SELECT *
FROM associado
WHERE dt_alteracao IS NULL
go
```

```
SELECT *
FROM associado
WHERE dt_alteracao IS NOT NULL
go
```

# Select

## SUBSTITUINDO VALORES NULOS

### SINTAXE

```
SELECT ISNULL(column, default_value_for_null)
FROM table_name
[WHERE condition ]
```

### EXEMPLOS

```
SELECT ISNULL(vl_renda_mes, 0)
from pessoa_fisica
go
```

Valor que substituirá o NULL na consulta.

**NOTA:** Este comando não vai mexer na tabela, apenas em mostra o valor NULL tratado tempo de consulta

Mostra todos os registros da tabela pessoa\_fisica, listando 0 quando encontrar algum valor de renda com valor NULO.

# Select

## LISTA DE VALORES: IN

### SINTAXE SIMPLIFICADA

```
SELECT column_list  
FROM table  
WHERE column IN (value1, value2, ...)
```

### EXEMPLO

```
SELECT nm_cidade,  
cd_uf  
FROM cidade  
WHERE cd_uf IN ('RS', 'SC', 'PR')  
go
```

Pode-se ter o mesmo resultado do IN com o conector lógico **OR**.

`cd_uf = 'RS' OR cd_uf = 'SC' OR cd_uf = 'PR'`

Mostra todos os nomes das cidades e códigos de UF das cidades que estejam dentro da lista: RS, SC ou PR.

# Select

## INTERVALO DE VALORES: BETWEEN

### SINTAXE SIMPLIFICADA

```
SELECT column_list  
FROM table  
WHERE column BETWEEN value1 AND value2
```

### EXEMPLO

```
SELECT *  
FROM lançamento_futuro  
WHERE dt_lanca_prev BETWEEN '2000-07-01' AND '2000-07-31'  
go
```

Mostra todos os registros da tabela lançamento\_futuro onde a data esteja entre 01/07/2000 e 31/07/2000.

# Select

## PESQUISA POR PADRÃO DE CARACTERES: LIKE

### SINTAXE SIMPLIFICADA

```
SELECT column_list  
FROM table  
WHERE column [NOT] LIKE expression
```

# Select

## PESQUISA POR PADRÃO DE CARACTERES: LIKE

### EXEMPLO

```
SELECT * FROM pessoa WHERE nm_pessoa LIKE '%LUIZ%'  
go
```

Mostra todos os nomes de  
pessoa que tenham LUIZ  
dentro do nome.

```
SELECT * FROM pessoa WHERE nm_pessoa LIKE 'JO%'  
go
```

Mostra todos os nomes de  
pessoa que comecem com JO.

```
SELECT * FROM pessoa WHERE nm_pessoa LIKE '%A'  
go
```

Mostra todos os nomes de  
pessoa que terminem com A.

# Select

## PESQUISA POR PADRÃO DE CARACTERES: LIKE

### CURINGAS

Curinga	Descrição
%	Qualquer string de zero ou mais caracteres.
_	Qualquer caractere simples.
[ ]	Qualquer caractere simples especificado no conjunto.
[ ^ ]	Qualquer caractere simples que não esteja especificado no conjunto.

# Select

## PESQUISA POR PADRÃO DE CARACTERES: LIKE

### CURINGAS

### EXEMPLOS

```
SELECT * FROM cidade WHERE cd_uf LIKE 'R_'  
go
```

Mostra todos os códigos de UF com  
comecem com R e tenham até 2 caracteres  
de tamanho do nome.

```
SELECT * FROM cidade WHERE cd_uf LIKE '[A,B]_'  
go
```

Mostra todos os códigos de UF com  
comecem com A ou B e tenham até  
2 caracteres de tamanho do nome.

# Select

## CONECTANDO CONDIÇÕES

**AND:** Quando ambas as condições devem ser satisfeitas;

**OR:** Quando somente uma das condições deve ser satisfeita.

### ORDEM DE PRECEDÊNCIA

Vários operadores são avaliados na seguinte ordem: NOT, AND e OR. Para alterar a ordem de precedência dos operadores, devemos utilizar parênteses.

# Select

## CONECTANDO CONDIÇÕES

### EXEMPLOS

```
SELECT * FROM pessoa  
WHERE cd_cidade_pes = 28 AND nr_fone_1 IS NOT NULL  
go
```

```
SELECT * FROM pessoa  
WHERE cd_cidade_pes = 28 OR nr_fone_1 IS NOT NULL  
go
```

# Select

## FUNÇÕES DE AGREGAÇÃO

Função	Valor de Retorno
COUNT(*)	Número de linhas encontradas
COUNT(column_name)	Número de valores não nulos
MAX(column_name)	Retorna o maior valor (numérica, string ou data)
MIN(column_name)	Retorna o menor valor (numérica, string ou data)
SUM(column_name)	Total numérico
AVG(column_name)	Média numérica

# Select

## FUNÇÕES DE AGREGAÇÃO

### SINTAXE SIMPLIFICADA

```
SELECT aggregate_function (column_name)
FROM table_name
[WHERE condition]
```

### EXEMPLOS

```
SELECT MAX(vl_lancamento)
FROM lancamento_diario
go
```

# Select

## FUNÇÕES DE AGREGAÇÃO

### OBSERVAÇÕES

- Exceto o **COUNT(\*)**, todas as funções de agregação ignoram valores nulos.
- **ISNULL()** substitui os valores nulos por valores especificados na função.  
Estes valores não são modificados na tabela, somente na visualização.

# Select GROUP BY

## SINTAXE SIMPLIFICADA

```
SELECT column_list, aggregate_function(column)
FROM table_name
GROUP BY column_list
go
```

```
SELECT column_list
FROM table_name
GROUP BY column_list
go
```

# Select GROUP BY

## EXEMPLO

```
SELECT dt_lancamento, COUNT(dt_lancamento)
FROM lancamento_diario
WHERE cd_conta = 9999
AND dt_lancamento BETWEEN '2001-11-01' AND '2001-11-30'
GROUP BY dt_lancamento
go
```

# Select HAVING

## SINTAXE SIMPLIFICADA

```
SELECT column_list, aggregate_function(column)
FROM table_name
GROUP BY column_list
HAVING condition
```

**NOTA:** O HAVING é chamado de WHERE do GROUP BY.

Através dele podemos efetuar filtros pelas colunas dinâmicas geradas pelas funções de agregação

# Select HAVING

## EXEMPLO

```
SELECT dt_lancamento, COUNT(dt_lancamento)
FROM lancamento_diario
WHERE cd_conta = 9999
AND dt_lancamento BETWEEN '2001-11-01' AND '2001-11-30'
GROUP BY dt_lancamento
HAVING COUNT(dt_lancamento) > 10
go
```

# Select UNIÃO (UNION)

## SINTAXE SIMPLIFICADA

query1

**UNION** [ALL]

query2

## OBSERVAÇÃO

Na operação de união de resultados, as linhas duplicadas são eliminadas por padrão. Para não eliminar as linhas duplicadas, é utilizada a palavra reservada “ALL” (UNION ALL).

# Select UNIÃO (UNION)

## EXEMPLO

```
SELECT nm_cidade as 'nome cidade' FROM cidade WHERE cd_uf = 'SC'
```

### **UNION**

```
SELECT nm_municipio FROM caf_municipio WHERE uf_municipio = 'SC'
```

```
ORDER BY nm_cidade
```

```
go
```

```
SELECT nm_cidade as 'nome cidade' FROM cidade WHERE cd_uf = 'SC'
```

### **UNION ALL**

```
SELECT nm_municipio FROM caf_municipio WHERE uf_municipio = 'SC'
```

```
ORDER BY nm_cidade
```

```
go
```

# Select UNIÃO (UNION)

## OBSERVAÇÕES

- As cláusulas GROUP BY e HAVING só poderão ser utilizados nas queries individuais
- As cláusulas ORDER BY só poderão aparecer ao final de todas as uniões de queries.
- Os alias de colunas (cabeçalhos) só poderão ser usados na primeira query do UNION.

# Select

## JUNÇÕES (JOINS)

**JUNÇÕES** são utilizados quando precisamos realizar uma consulta utilizando mais de uma tabela.

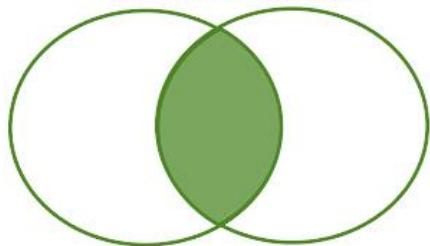
Basicamente temos 5 tipos de JOIN no SQL Server (podendo mudar de um SGBD para outro):

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN
- CROSS JOIN

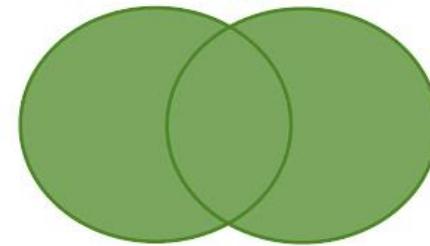
# Select

## JUNÇÕES (JOINS)

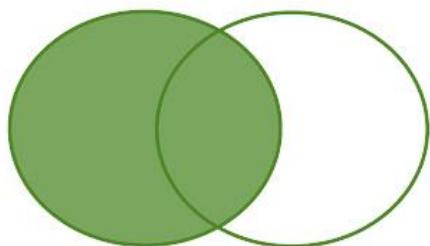
Inner Join



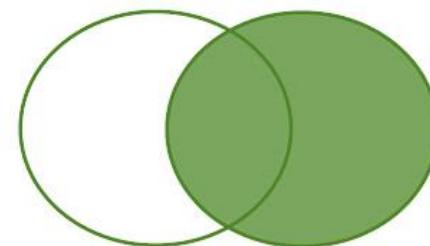
Full Join



Left Join



Right Join

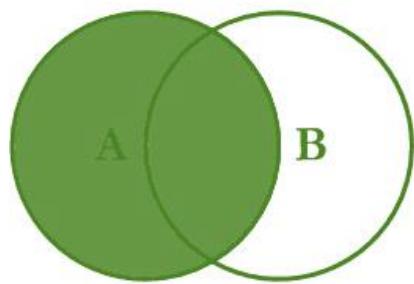


Cross Join

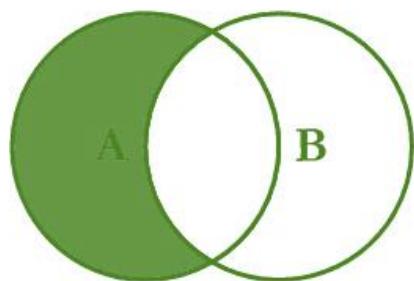
# Select

## JUNÇÕES (JOINS)

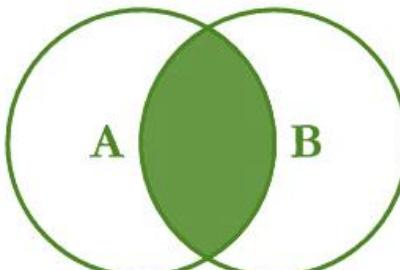
### SQL JOINS



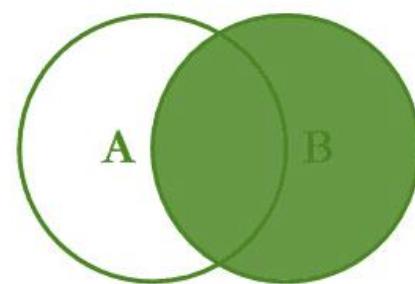
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



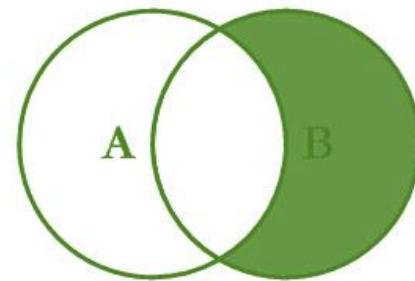
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



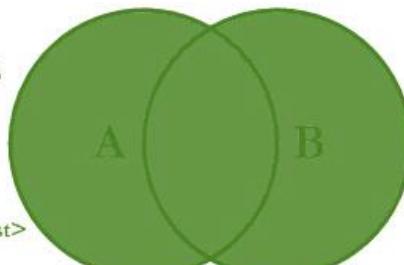
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
WHERE A.Key IS NULL
OR B.Key IS NULL
```

# Select

## JUNÇÕES (JOINS)

### SINTAXE SIMPLIFICADA

```
SELECT column_list  
FROM table1 INNER JOIN table2  
    ON table1.column_from_table1 = table2.column_from_table2  
go
```

```
SELECT column_list  
FROM table1 LEFT JOIN table2  
    ON table1.column_from_table1 = table2.column_from_table2  
go
```

```
SELECT column_list  
FROM table1 CROSS JOIN table2  
go
```

# Select

## JUNÇÕES (JOINS)

### EXEMPLO

```
SELECT usuario.cd_usuario, perfil.ds_perfil  
FROM usuario INNER JOIN perfil  
    ON usuario.cd_perfil = perfil.cd_perfil  
go
```

# Select

## JUNÇÕES (JOINS)

### OBSERVAÇÕES

- Nomes de colunas de ligação do **JOIN** não precisam ser iguais, precisam ser parte de uma FK (recomendado).
- **JOINS** entre mais de duas tabelas devem listar todas tabelas envolvidas na cláusula FROM, com seus devidos conectivos de relacionamento **ON**, mesmo que não estejam visíveis na lista do SELECT.

# Select SQL ALIAS

## SINTAXE SIMPLIFICADA

```
SELECT column_list  
FROM real_table_name alias_name
```

ALIAS é um apelido curto que damos para as tabelas para facilitar a escrita, evitar repetição do nome da tabela na consulta.

## EXEMPLO

```
SELECT ld.dt_lancamento, tl.sc_lancamento, ld.vl_lancamento  
FROM lancamento_diario ld  
INNER JOIN tipo_lancamento tl  
ON ld.cd_tp_lancamento = tl.cd_tp_lancamento  
go
```

# Select SUBQUERIES

## SINTAXE SIMPLIFICADA

```
SELECT outer_query_colum_list
FROM table
WHERE value { = | [NOT] IN }
(SELECT column_name
FROM table )
```

# Select SUBQUERIES

## EXEMPLO

```
SELECT cd_conta
FROM pessoa_conta
WHERE cd_pessoa IN (SELECT cd_pessoa
                      FROM PESSOA
                      WHERE nm_pessoa LIKE 'JOAO%')
go
```

# Select

## FUNÇÕES DE STRINGS

Função	Valor de Retorno
SUBSTRING(string, val1, val2)	Retorna parte de uma string a partir dos parâmetros inicio (val1) e fim (val2)
RIGHT(string)	Retorna parte da string da posição indicada (da dir para esq)
LEFT(string)	Retorna parte da string da posição indicada (da esq para dir)
UPPER(string)	Converte a string para maiúsculo
LOWER(string)	Converte a string para minúsculo
REVERSE(string)	Inverte os caracteres da string
LTRIM(string)	Remove os espaços à esquerda
RTRIM(string)	Remove os espaços à direita
LEN(string)	Retorna a quantidade total de caracteres
REPLICATE(string, val)	Repete string n vezes o valor de val

# Select

## FUNÇÕES DE STRING

### SINTAXE SIMPLIFICADA

```
SELECT string_funtion (column_name)
FROM table_name
[WHERE condition]
```

### EXEMPLOS

```
SELECT REVERSE(nm_pessoa)
FROM pessoa
go
```

# Select

## FUNÇÕES MATEMÁTICAS

Função	Valor de Retorno
ABS(número)	Retorna o valor absoluto do número
PI()	Retorna o valor 3,1415926...
EXP(número)	Retorna o exponencial do número
CEILING (número)	Retorna o menor valor inteiro maior ou igual ao número
FLOOR(número)	Retorna o maior valor inteiro menor ou igual ao número
POWER(número, potencia)	Retorna o valor de número elevado à potência
ROUND(número, precisão)	Arredonda o número de acordo com a precisão
SQRT(número)	Raiz quadrada do número
RAND(número)	Retorna um valor aleatório
SIGN(número)	Retorna +1 para número positivo e -1 para número negativo ou zero

# Select

## FUNÇÕES MATEMÁTICAS

### SINTAXE SIMPLIFICADA

```
SELECT math_funtion (column_name)
FROM table_name
[WHERE condition]
```

### EXEMPLOS

```
SELECT ROUND(vl_lancamento, 2)
FROM lancamento
go
```

# Select

## FUNÇÕES DE DATAS

Função	Valor de Retorno
GETDATE()	Retorna a data atual
DATEPART(tipo, data)	Retorna inteiro que é parte da data de acordo com o tipo especificado
DATEDIFF(tipo, data1, data2)	Retorna data2 – data1 de acordo com o ‘tipo’ especificado
DATEADD(tipo, número, data)	Retorna a soma do número + data de acordo com o tipo especificado

# Select

## FUNÇÕES DE DATAS

### SINTAXE SIMPLIFICADA

```
SELECT date_function (column_name)
FROM table_name
[WHERE condition]
```

### EXEMPLOS

```
SELECT GETDATE()
go
```

# Select

## FUNÇÕES DE DATAS

Tipo	Abreviação	Intervalo
Ano	YY	1793 – 9999
Mês	MM	1 – 12
Dia do ano	DY	1 – 366
Dia	DD	1 – 31
Semana	DW	1 – 7 (1 = domingo)
Hora	HH	0 – 23
Minuto	MI	0 – 59
Segundo	SS	0 – 59
Milisegundo	MS	0 – 999

# Select

## FUNÇÃO CONVERT

### SINTAXE

```
convert (<datatype>, <expressão>)
```

### EXEMPLO

```
select 'O número é : ' + convert (char(3), 515)  
go
```

-- RESULTADO --

O número é 515

# Select

## CONVERSÃO DE DATAS

Tipo c/ YY	Tipo c/ YYYY	Intervalo
-	0   100	mon dd yyyy hh:mm AM (ou PM)
1	101	mm/dd/yy
2	102	yy.mm.dd
3	103	dd/mm/yy
4	104	dd.mm.yy
5	105	dd-mm-yy
6	106	dd mon yy
7	107	mon dd yy
8	108	hh:mi:yy
9	109	mon dd yyyy hh:mm:ssss AM (ou PM)
10	110	mm-dd-yyyy
11	111	yy/mm/dd

# Select

## CONVERSÃO DE DATAS

### SINTAXE

```
convert (<datatype>, <expressão>, <tipo>)
```

### EXEMPLO

```
select convert (char(10), getdate(), 103)  
go
```

-- RESULTADO --

17/12/2004

# Select

## FUNÇÕES DE SISTEMA

Função	Valor de retorno
host_id()	Retorna o ID do processo cliente
host_name()	Retorna o nome do computador do processo cliente
suser_name()	Retorna o login do usuário no Servidor SQL Server
user_name()	Retorna o nome do usuário no banco de dados
db_name([db_id])	Retorna o nome do banco de dados
db_id([db_name])	Retorna o ID do banco de dados
object_name(obj_id)	Retorna o nome do objeto no banco de dados
object_id('obj_name')	Retorna o ID do objeto no banco de dados

# Select

## FUNÇÕES DE SISTEMA

### SINTAXE SIMPLIFICADA

```
SELECT system_funtion (column_name)
FROM table_name
[WHERE condition]
```

### EXEMPLOS

```
SELECT DB_NAME()
go
```

# Select

## EXERCÍCIO e MATERIAL DE APOIO

Exercícios Portal AVA - Exercício de Banco de dados - SELECT.pdf

### **Material de apoio**

<https://github.com/jlsilva01/exemplos-sql-satc>

# Trabalho Final

## TRABALHO FINAL COM BASE EM METODOLOGIAS ATIVAS DE APRENDIZAGEM

Construção de **projeto de banco de dados** para um dos temas abaixo:

- Reserva de Restaurantes
- Chat online
- Conta Digital
- Aplicativo de Quiz
- Sistema de Login
- Sistema de Empregos
- Sistema de Votos
- Automação de tarefas

**Premissas:**

- Criar o diagrama do modelo Conceitual (mínimo 5 entidades)
- Criar o diagrama do modelo Físico (otimizar quantidade de colunas e tipos de dados)
- Criar o dicionário de dados e documentar todas as tabelas e suas respectivas colunas de acordo com o template 1.
- Gerar script de criação do banco de dados (SGDB à sua escolha) e criar o banco de dados.
- Popular as tabelas com dados de exemplos (mínimo 5 itens principais).
- Criar e apresentar as principais consultas que possam ser utilizadas por uma aplicação que vai consumir o modelo deste projeto.

# Trabalho Final

## TRABALHO FINAL COM BASE EM METODOLOGIAS ATIVAS DE APRENDIZAGEM

- Trabalho em conjunto (Grupos a serem montados em sala) com avaliação individual
- A avaliação será baseada na **participação de cada aluno em sala** durante a construção dos trabalhos.
- Toda a jornada de construção do trabalho deverá ser **apresentado através do Powerpoint** (ou similares) - **15 min.** de apresentação.
- **Criar arquivo PDF** com os diagramas dos dois modelos, scripts de criação dos banco, script que insere os dados nas tabelas e as script das consultas (conforme premissa) e **entregar no dia da apresentação**.

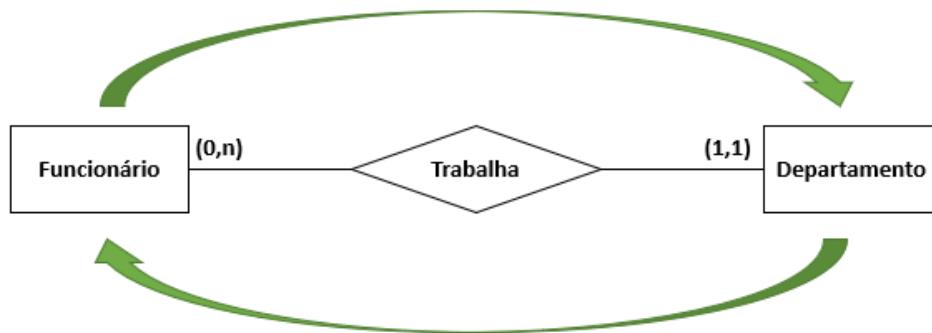
## Categorias

**DDL** : Linguagem de Definição de Dados  
**DQL** : Linguagem de Consulta de Dados  
**DML** : Linguagem de Manipulação de Dados  
**DCL** : Linguagem de Controle de Dados  
**DTL** : Linguagem de Transação de Dados

## Comandos

**DDL**  
CREATE | DROP | ALTER | RENAME  
**DQL**  
SELECT  
**DML**  
INSERT | UPDATE | DELETE  
**DCL**  
GRANT | REVOKE  
**DTL**  
COMMIT | ROLLBACK | BEGIN TRAN

## Modelo ER Conceitual



## Operadores

**Aritméticos** bit a bit  
+ - \* / % & | ^

**Comparação**  
= < > <= >= <> != !> !<

**Lógicos**  
AND | OR | NOT | IN | LIKE | ALL  
SOME | ANY | EXISTS | BETWEEN

**Compostos**  
+= -= \*= /= %= &= ^= |=

## Palavras-chave

WHERE | DISTINCT | TOP | FROM  
AS | ORDER BY | ASC | DESC | CASE  
DEFAULT | VALUES | SET

## Objetos

TABLE | VIEW | PROCEDURE | INDEX  
TRIGGER | SEQUENCE | FUNCTION

## Constraints

NOT NULL | UNIQUE | PRIMARY KEY  
FOREIGN KEY | CHECK | DEFAULT

## Funções de Agregação

AVG | MIN | MAX | COUNT | SUM

## Palavras-chave Agregação

GROUP BY | HAVING

## Joins

INNER JOIN



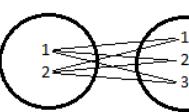
LEFT JOIN



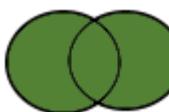
RIGHT JOIN



CROSS JOIN

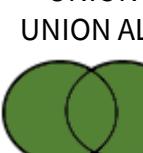


FULL JOIN

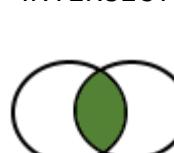


## Operações de Conjunto

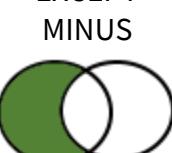
UNION



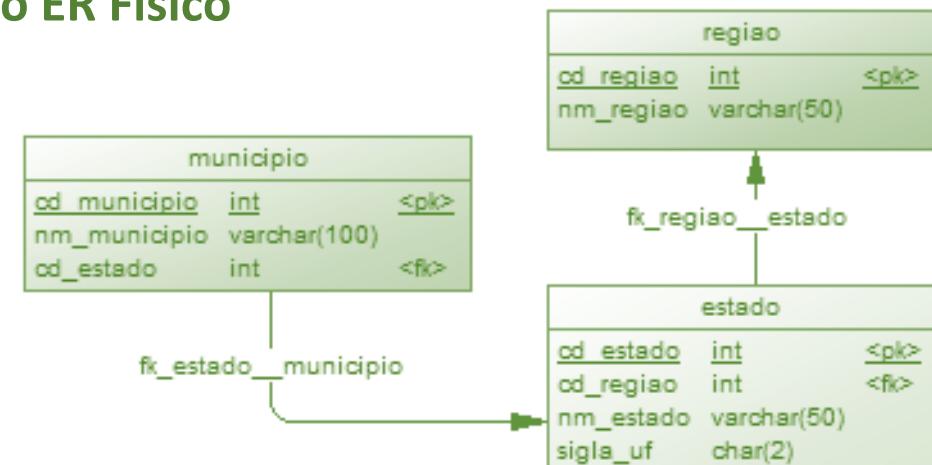
INTERSECT



EXCEPT



## Modelo ER Físico



## Exemplos DDL

### Cria uma tabela

```
CREATE TABLE aluno (
    cd_aluno int NOT NULL,
    nm_aluno varchar(100),
    idade int NOT NULL
);
```

### Adiciona uma coluna

```
ALTER TABLE aluno
ADD is_ativo bit;
```

### Modifica uma coluna

```
ALTER TABLE aluno
ALTER COLUMN idade tinyint;
```

### Remove uma coluna

```
ALTER TABLE aluno
DROP COLUMN idade;
```

### Remove uma tabela

```
DROP TABLE aluno;
```

## Exemplos DML

### Insere dados na tabela

```
INSERT INTO aluno (cd_aluno, nm_aluno, idade)
VALUES (1, 'Fulano de Tal', 23);
```

### Insere dados a partir de uma consulta

```
INSERT INTO aluno (cd_aluno, nm_aluno, idade)
SELECT codigo, nome, idade FROM colaborador;
```

### Atualiza dados de uma tabela

```
UPDATE aluno SET idade = 36 WHERE cd_aluno = 123;
```

### Apaga alguns dados de uma tabela

```
DELETE FROM aluno WHERE is_ativo = 0;
```

### Apaga todos dados de uma tabela

```
DELETE FROM aluno;
* TRUNCATE TABLE aluno;
```

### Insere dados na tabela

```
INSERT INTO aluno (cd_aluno, nm_aluno, idade)
VALUES (1, 'Fulano de Tal', 23);
```

## Exemplos DQL

### Lista todos as linhas e colunas

```
SELECT * FROM aluno;
```

### Filtre linhas de um tabela

```
SELECT * FROM aluno
WHERE cd_aluno = 123;
```

```
SELECT * FROM aluno
WHERE cd_aluno = 123
AND is_ativo = 1;
```

### Filtre colunas de uma tabela

```
SELECT cd_aluno, nm_aluno
FROM aluno
WHERE cd_aluno = 123
AND is_ativo = 1;
```

### Lista máximo de 10 linhas

```
SELECT TOP 10 cd_aluno, nm_aluno
FROM aluno
WHERE is_ativo = 1;
```

### Conta quantidade de linhas

```
SELECT COUNT(*) FROM aluno;
```

### Valores max e min de uma coluna

```
SELECT MAX(idade), MIN(idade) FROM aluno;
```

### Soma valores de uma coluna

```
SELECT SUM(idade) FROM aluno;
```

### Média dos valores de uma coluna

```
SELECT AVG(idade) FROM aluno;
```

### Ordena resultado (ascendente)

```
SELECT * FROM aluno
ORDER BY nm_aluno ASC;
```

### Ordena resultado (descendente)

```
SELECT * FROM aluno
ORDER BY nm_aluno DESC;
```

### Busca linhas e colunas de 2 tabelas

```
SELECT * FROM aluno INNER JOIN aula
ON aluno.cd_aluno = aula.cd_aluno
WHERE is_ativo = 1;
```

## Exemplos de Constraints

### Remove uma Constraint

```
ALTER TABLE aluno DROP CONSTRAINT pk_aluno;
```

### Cria uma Check Constraint

```
ALTER TABLE aluno ADD CONSTRAINT chk_1 CHECK (is_ativo IN (0, 1));
```

### Cria um Default Constraint

```
ALTER TABLE aluno ADD CONSTRAINT df_ativo DEFAULT 1 FOR is_ativo;
```

### Agregação e filtro na agregação

```
SELECT turma, COUNT(1) as quantidade
FROM turma
WHERE curso LIKE 'Engenharia%'
GROUP BY turma
HAVING quantidade > 40;
```