

Banco de Dados I

Modelo Conceitual

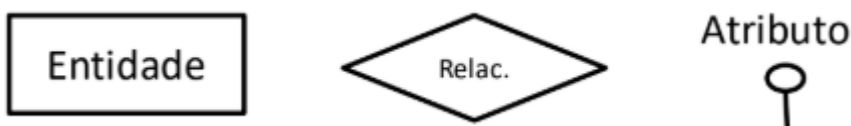
Descreve a estrutura de um BD de uma forma mais próxima da percepção dos usuários

Independente de aspectos de implementação

Representa a estrutura de um banco de dados sem considerar um SGBD específico

Modelo Entidade-Relacionamento

Três conceitos básicos



Entidade

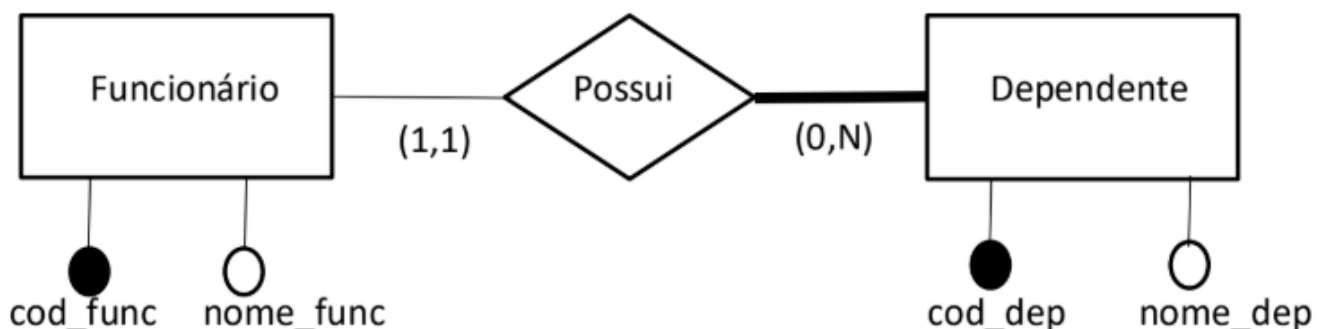
“Conjunto de objetos da realidade modelada sobre os quais deseja-se manter informações no banco de dados”

Podendo ser:

- Algo que existe fisicamente, que pode ser tocado
- Algo que existe conceitualmente (Abstrato)
- Podem ser eventos

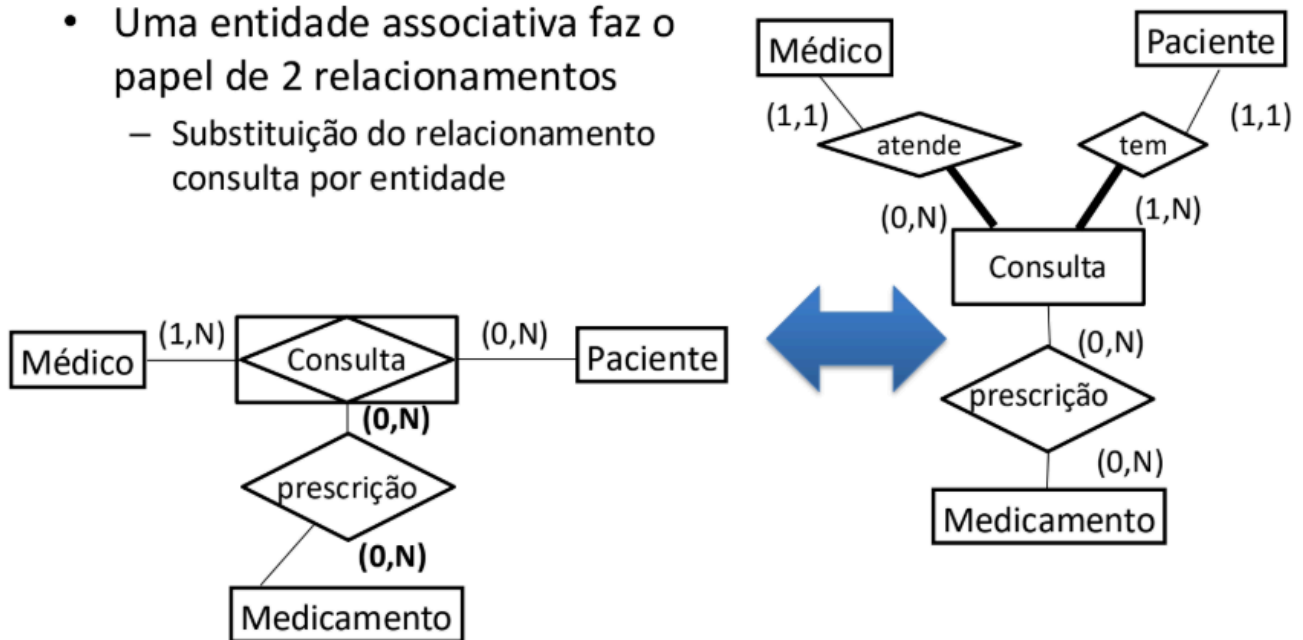
Entidade Fraca

Existem entidades que apenas existem em função de outras



Entidade Associativa

- Uma entidade associativa faz o papel de 2 relacionamentos
 - Substituição do relacionamento consulta por entidade



Relacionamento

Conjunto de associações entre ocorrências/instâncias de entidades

Cada ocorrência da entidade que participa de um relacionamento desempenha um Papel

Úteis sobretudo nos auto-relacionamentos (relacionamento unário)

Grau de Relacionamento

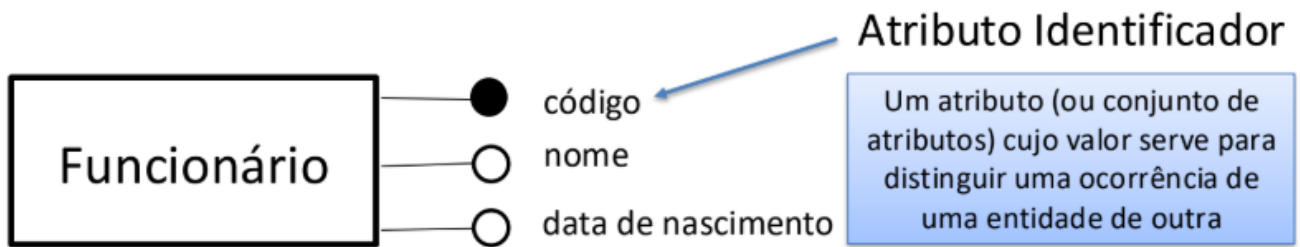
Número de (instâncias de) entidades que participam do relacionamento

Cardinalidades

Expressar o número de ocorrências/instâncias às quais outra ocorrência/instância pode ser associada através de um conjunto de relacionamentos

- (1:1)
- (1:N)
- (N:N)

Atributo

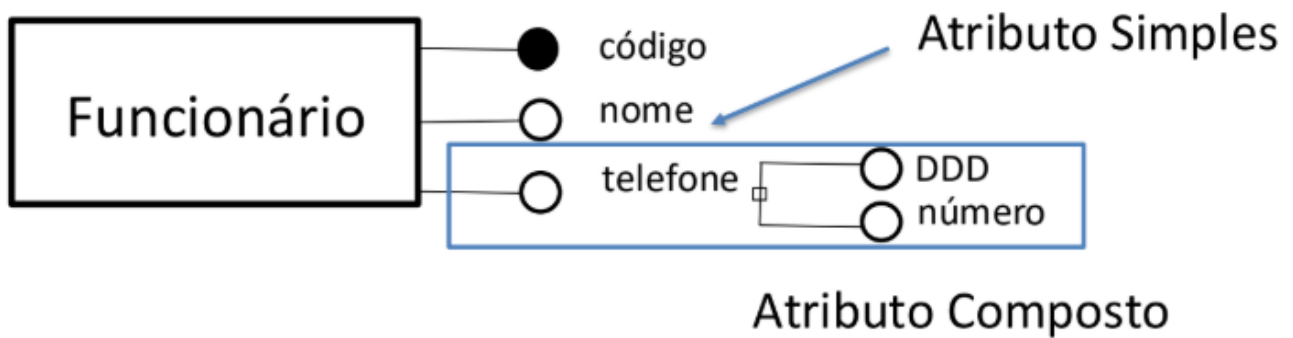


Propriedades que descrevem uma entidade

Exemplo:

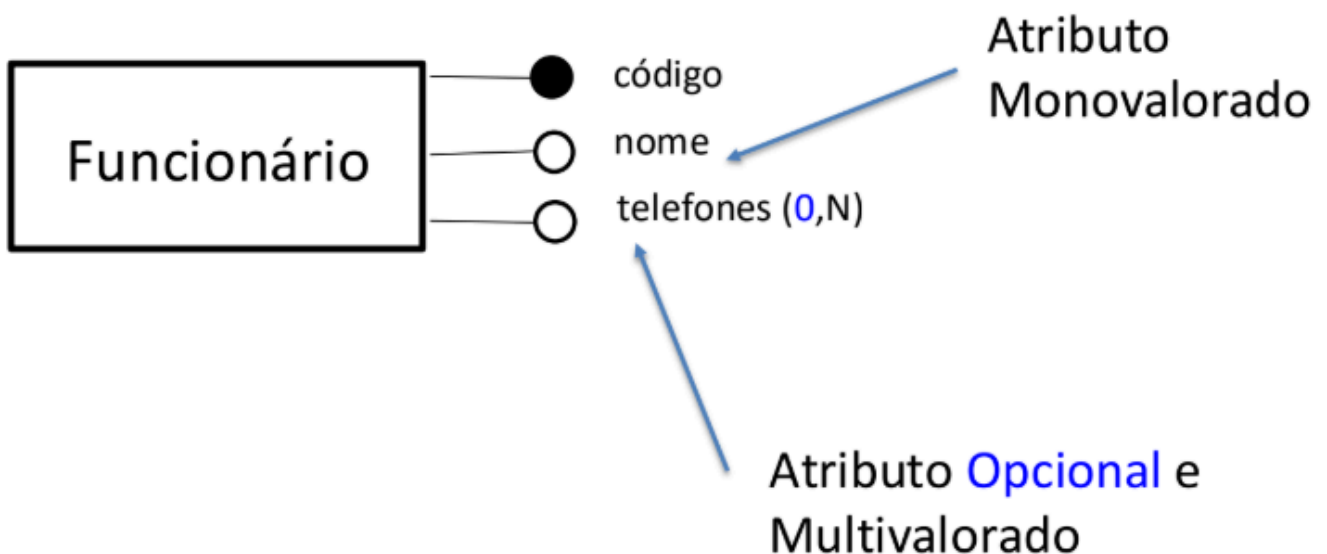
- Entidade: Funcionário
- Atributos: código, nome, data de nascimento

Atributo Composto

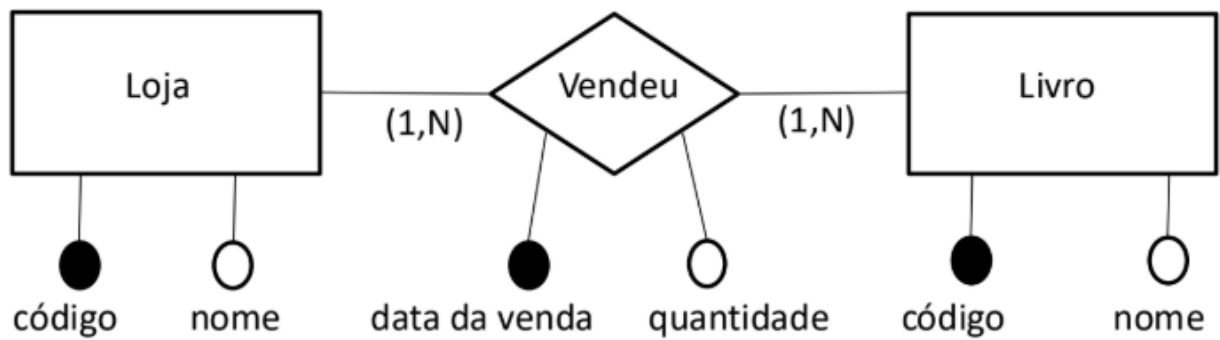


Atributo Multivalorado

Atributos multivalorados e compostos são atributos complexos



- Atributo de Relacionamento

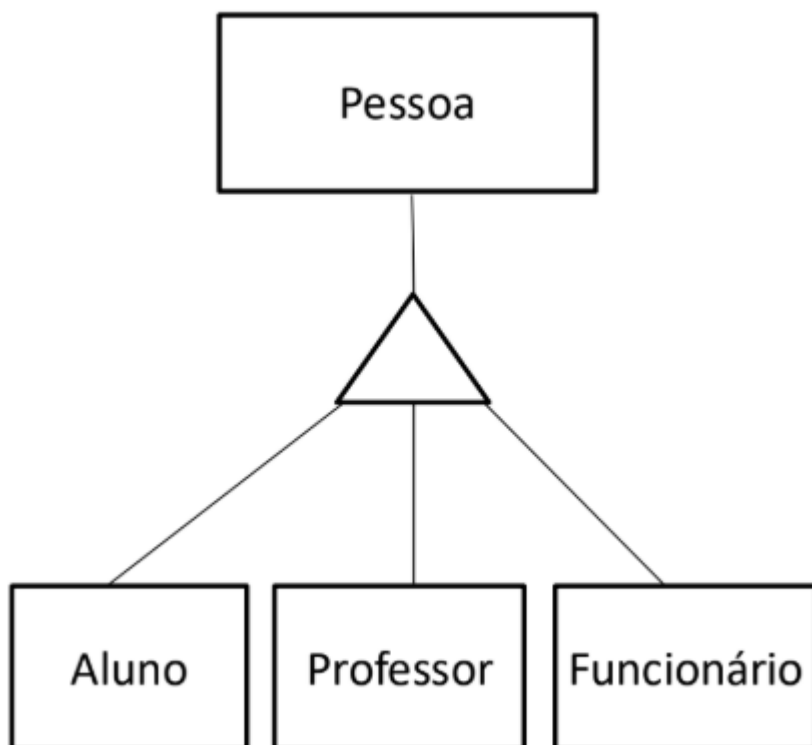


Generalização (Especialização)

É um relacionamento de classificação entre uma entidade mais geral e outra mais específica

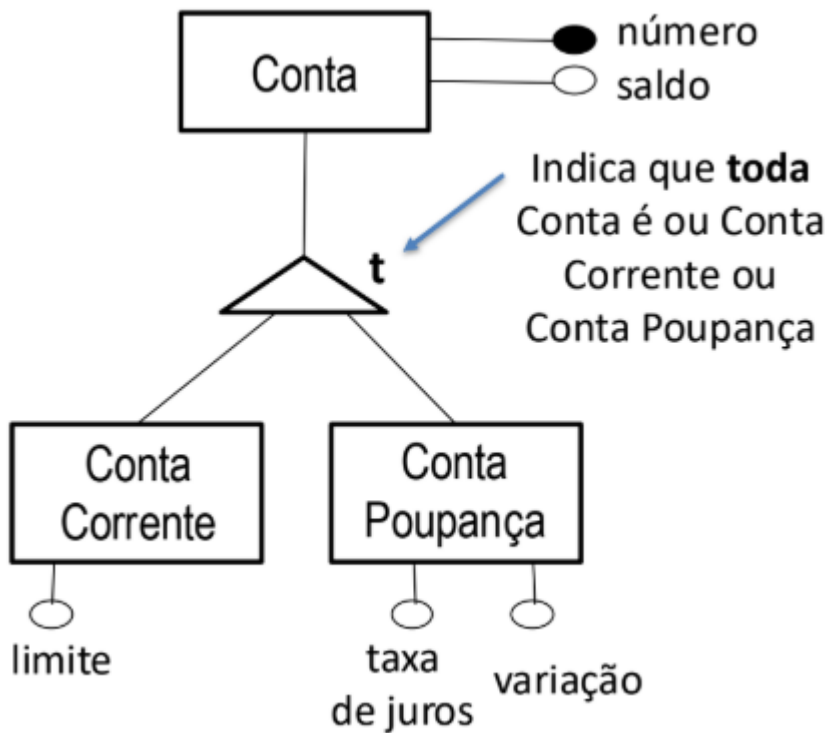
A entidade mais geral é denominada entidade de nível superior (**superclasse**) e a mais específica de entidade de nível inferior (**subclasse**)

As propriedades da superclasse são herdadas pela subclasse → **Herança**



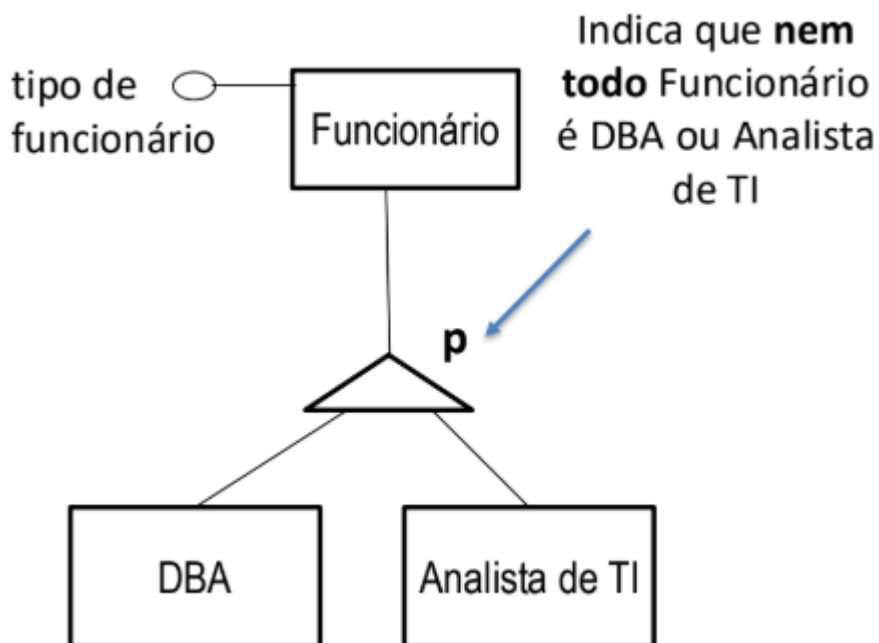
Especialização Total

Para cada ocorrência da entidade genérica existe **sempre** uma entidade especializada



Especialização Parcial

Nem toda ocorrência da entidade genérica possui uma ocorrência correspondente em uma entidade especializada






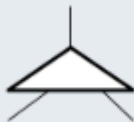
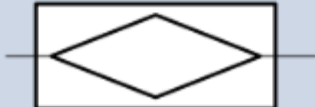


Especialização Exclusiva

Exclusiva (x): ocorrência de uma entidade genérica em apenas uma entidade especializada

Especialização Compartilhada

Compartilhada (c): Ocorrência de uma entidade genérica pode aparecer em **várias** entidades especializadas

Conceito	Símbolo									
Entidade										
Relacionamento										
Atributo										
Atributo identificador										
Relacionamento identificador										
Generalização/ especialização	<div></div> <table><tr><th></th><th>Total (t)</th><th>Parcial (p)</th></tr><tr><td>Exclusiva (x)</td><td>xt</td><td>xp</td></tr><tr><td>Compartilhada (c)</td><td>ct</td><td>cp</td></tr></table>		Total (t)	Parcial (p)	Exclusiva (x)	xt	xp	Compartilhada (c)	ct	cp
	Total (t)	Parcial (p)								
Exclusiva (x)	xt	xp								
Compartilhada (c)	ct	cp								
Entidade associativa										

Modelo ER
- Símbolos
[Heuser, 2009]

Entidade fraca

17

Construção do Modelo

Transformando Relacionamento N:N para Entidade

1. O relacionamento N:N é representado como uma entidade
2. A entidade criada é relacionada às entidades que originalmente participavam do relacionamento
3. A entidade criada tem como identificador:
 - Os relacionamentos com as entidades que originalmente participavam do relacionamento
 - Os atributos que eram identificadores do relacionamento original (caso o relacionamento original tivesse atributos identificadores)
4. A cardinalidade da entidade criada em cada relacionamento de que participa é (1,1)
5. As cardinalidades das entidades que eram originalmente associadas pelo relacionamento transformado em entidade são transcritas ao novo modelo conforme mostrado no exemplo inicial

Entidade vs. Atributo

Se objeto está vinculado a outros objetos:

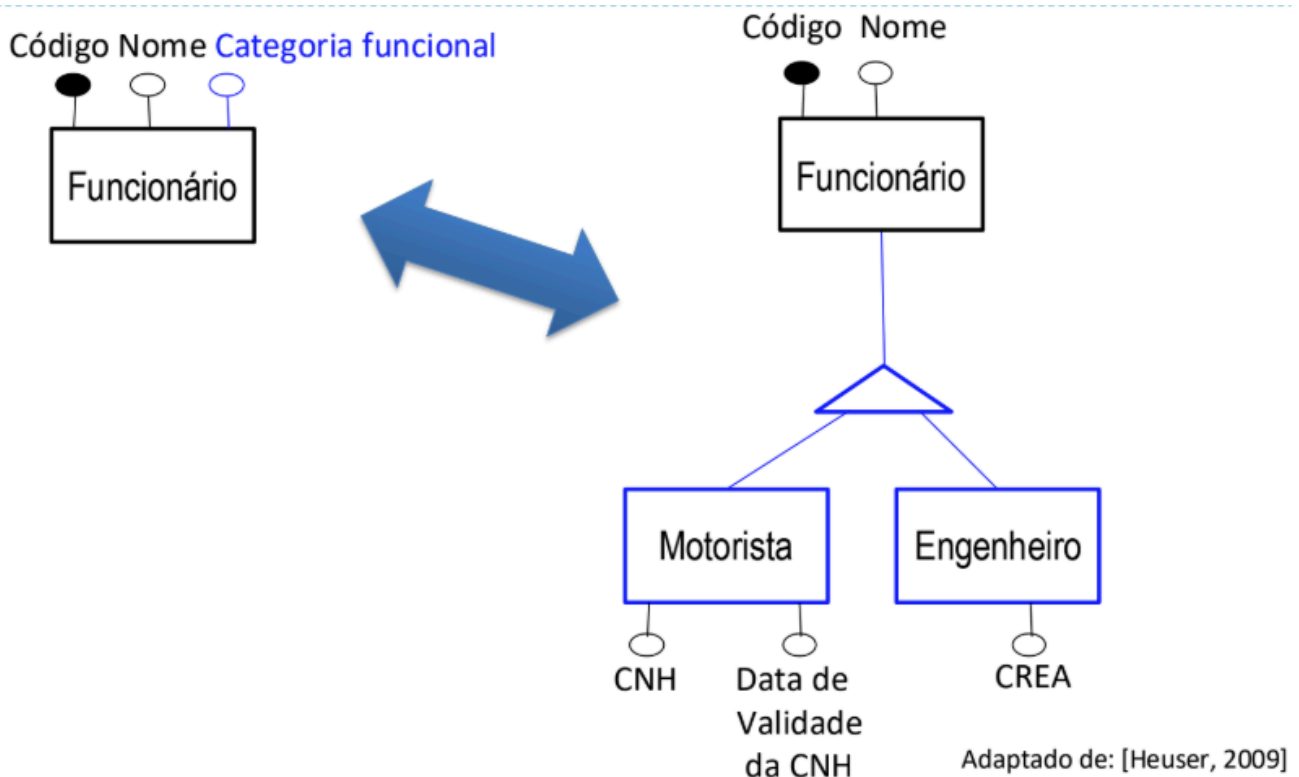
- Deve ser modelado como entidade
Caso contrário:
- Pode ser modelado como atributo
Conjunto de valores de um determinado objeto é fixo (domínio fixo):
- Pode ser modelado como atributo
Existem transações no sistema que alteram o conjunto de valores do objeto (domínio variável):
- Não deve ser modelado como atributo

Atributo vs. Especialização/Generalização

Especialização deve ser usada quando:

As classes especializadas de entidades possuem propriedades particulares:

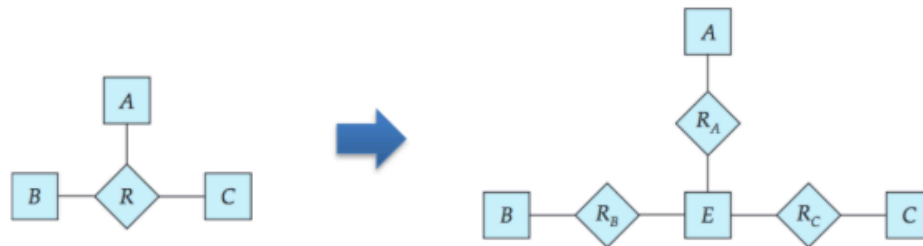
- Atributos
- Relacionamentos
- Generalizações/especializações



Conversão de relacionamentos não binários para a forma binária

- Substituir R entre entidades A , B e C por uma entidade E , e três relacionamentos:

1. R_A , relacionando E e A
 2. R_B , relacionando E e B
 3. R_C , relacionando E e C
- Criar um atributo identificador especial para E
 - Adicionar quaisquer atributos de R para E
 - Para cada relacionamento (a_i, b_i, c_i) em R , criar
 1. uma nova ocorrência e_i na entidade E
 2. adiciona (e_i, a_i) para R_A
 3. adiciona (e_i, b_i) para R_B
 4. adiciona (e_i, c_i) para R_C



Modelo Relacional

O modelo relacional representa um banco de dados como um conjunto de relações

Relação \leftrightarrow **Tabela** (de valores)

coluna (atributo)		nome do campo (nome do atributo)
<i>ID_Professor</i>	<i>Nome</i>	<i>CPF</i>
1	Silva	11111111111
2	Souza	22222222222
3	Costa	33333333333

linha (tupla)

valor do campo
(valor do atributo)

Obs.: **Domínio** = Conjunto de valores que pode aparecer em cada coluna

Relação R

$$R (A_1, A_2, A_3, \dots, A_n)$$

Onde:

- R é o nome da relação
- A_1, A_2, \dots, A_n é uma lista de atributos
- N é o grau da relação

Exemplo:

Chave Primária (PK)

Seja $K \subset R$

K é uma super chave de R se os valores de K são suficientes para identificar um única tupla de cada possível relação de $r(R)$

Super-chave K é uma chave candidata se K é mínima

Uma das chaves candidatas é selecionada para ser **chave primária**

Chave(s) candidata(s) não selecionada(s) → chave(s) alternativa(s)

Restrições de integridade de valores Null

Especifica se a um atributo é permitido ter valores Null

Chave Estrangeira

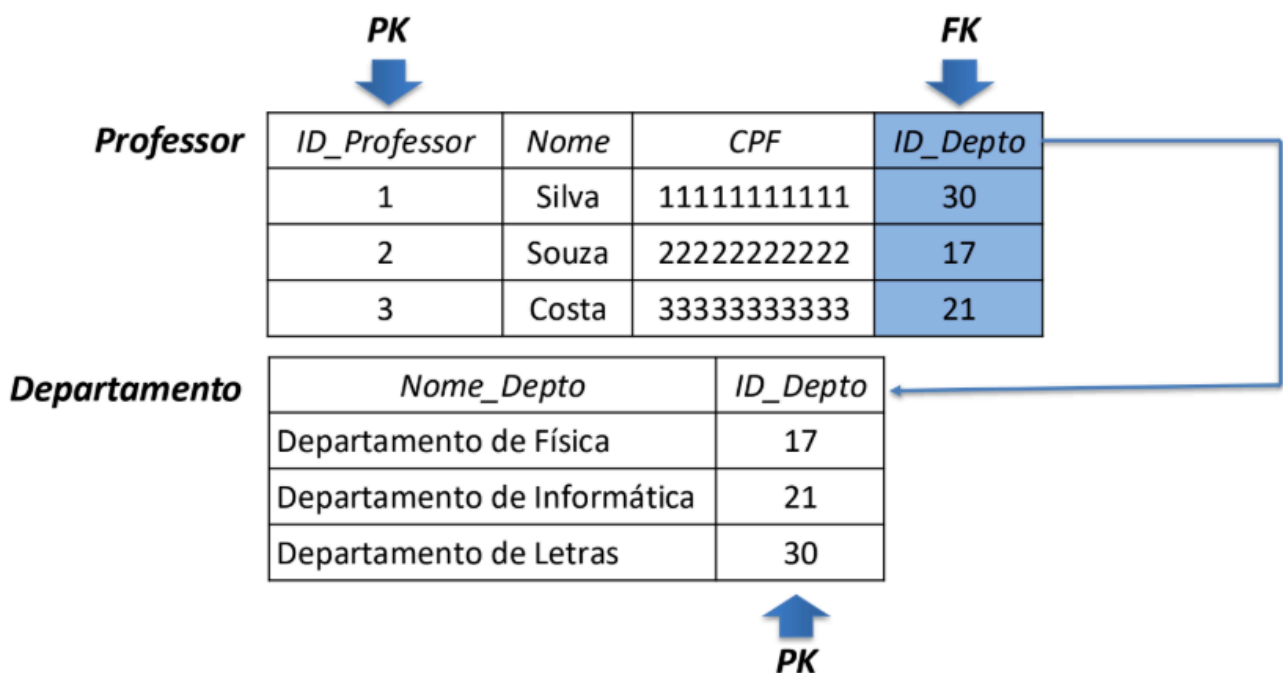
$$R_1[FK] \rightarrow R_2[PK]$$

Onde:

- PK é a chave primária
- FK é a chave estrangeira

Então, para qualquer tupla t_1 de R_1 :

$t_1[FK] = t_2[PK]$, onde t_2 é uma tupla de R_2 ou $t_1[FK]$ é Null



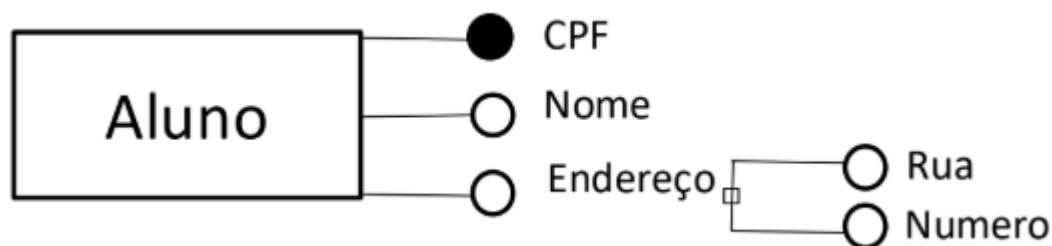
Esquema Lógico

Departamento (Nome_Depto, ID_Depto)
PK(ID_Depto)

Professor (ID_Professor, Nome, CPF, ID_Depto)
PK(ID_Professor)
FK(ID_Depto) ref Departamento(ID_Depto)

Mapeamento ER → Relacional

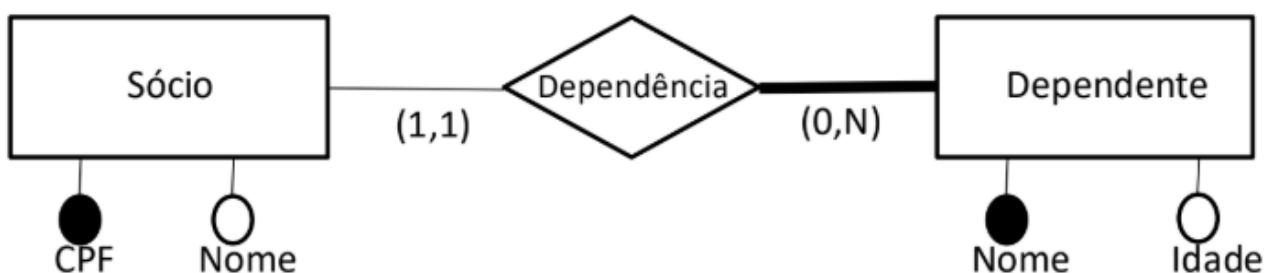
Entidades Fortes



1. Criar uma relação: `Aluno()`
2. Para todo Atributo Simples criar um atributo: `Aluno(Nome, CPF)`
3. Para atributos compostos, criar vários atributos simples: `Aluno(Nome, CPF, Rua, Numero)`
4. Criar uma chave primária

`Aluno(Nome, CPF, Rua, Numero)`
PK(CPF)

Entidades Fracas



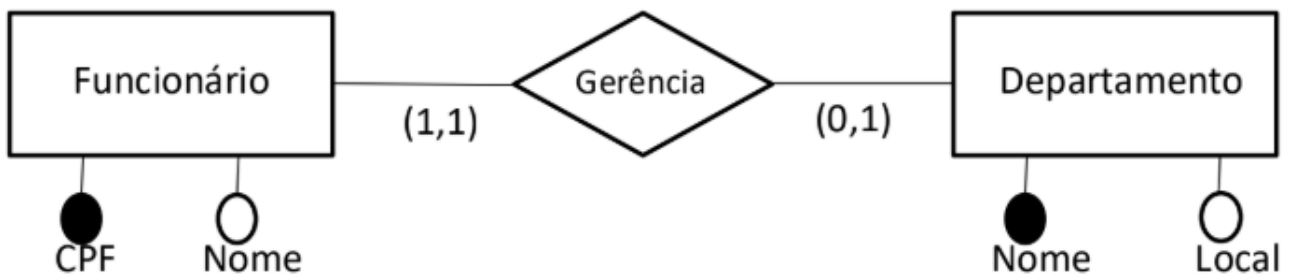
1. Criar uma relação `Dependente()`
2. Para todo Atributo Simples criar um atributo: `Dependente(Nome, idade)`
3. Para atributos compostos, criar vários atributos simples
4. Criar uma FK apontando para PK da Entidade Forte

```
Dependente(Nome, Idade, SocioCPF)
FK(SocioCPF) ref Socio(CPF)
```

5. Criar uma PK composta pelo Atributo identificador e FK

```
Dependente(Nome, Idade, SocioCPF)
FK(SocioCPF) ref Socio(CPF)
PK(Nome, SocioCPF)
```

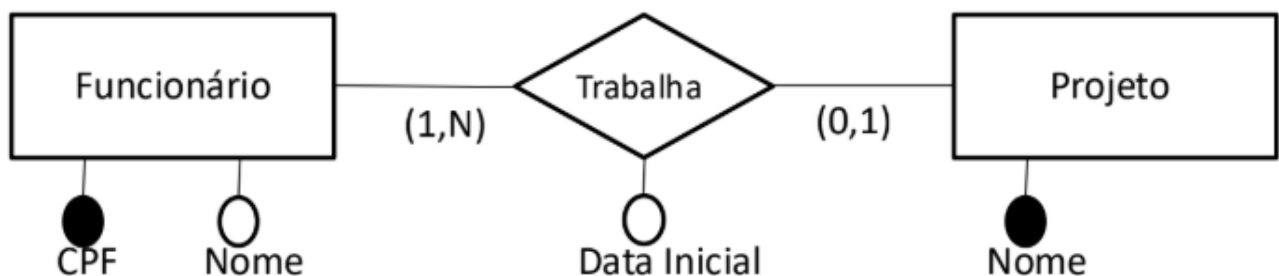
Relacionamentos Binários 1:1



1. Criar uma FK na relação com participação total (*Todo departamento tem funcionário*)

```
Funcionário (Nome, CPF)
PK(CPF)
Departamento (Nome, Local, GerenteCPF)
FK(GerenteCPF) ref Funcionário(CPF)
PK(Nome)
```

Relacionamentos Binários 1:N



1. Criar uma FK na relação com cardinalidade N

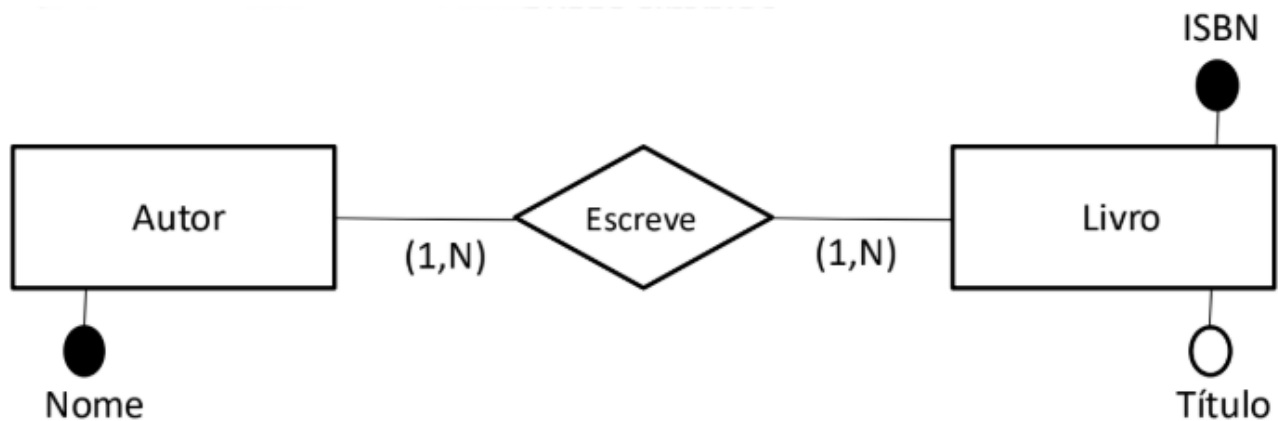
```
Funcionário (Nome, CPF, ProjetoNome)
PK(CPF)
FK(ProjetoNome) ref Projeto(Nome)
```

```
Projeto (Nome)
  PK(Nome)
```

2. Criar todos os atributos do relacionamento, se houver

```
Funcionário (Nome, CPF, ProjetoNome, DataInicial)
  PK(CPF)
  FK(ProjetoNome) ref Projeto(Nome)
Projeto (Nome)
  PK(Nome)
```

Relacionamentos Binários N:N



1. Criar um novo Relacionamento

```
Autor (Nome)
  PK(Nome)

Livro (ISBN, Título)
  PK(ISBN)

Escreve ()
```

2. Criar FK das duas relações

```
Autor (Nome)
  PK(Nome)

Livro (ISBN, Título)
  PK(ISBN)

Escreve (AutorNome, LivroISBN)
  FK(AutorNome) ref Autor(Nome)
  FK(LivroISBN) ref Livro(ISBN)
```

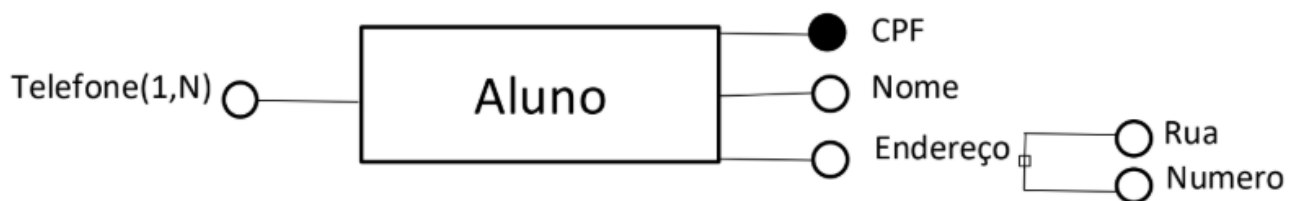
3. Criar a PK (FK1 + FK2)

```
Autor (Nome)
    PK(Nome)

Livro (ISBN, Título)
    PK(ISBN)

Escreve (AutorNome, LivroISBN)
    FK(AutorNome) ref Autor(Nome)
    FK(LivroISBN) ref Livro(ISBN)
    PK(AutorNome, LivroISBN)
```

Atributos Multivalorados



1. Criar uma nova relação: `Telefone ()`
2. Criar Atributo(s) Simples: `Telefone (Telefone)`
3. Cria FK para a relação original

```
Aluno(CPF, Nome, Endereço)
    PK(CPF)

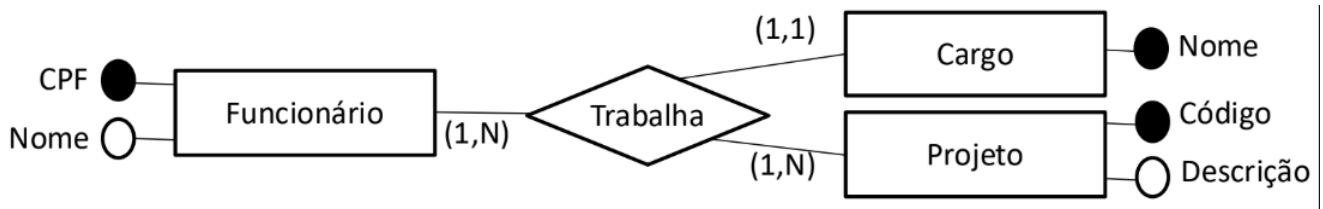
Telefone(Telefone, AlunoCPF)
    FK(AlunoCPF) ref Aluno(CPK)
```

4. Criar PK (FK + Atributos)

```
Aluno(CPF, Nome, Endereço)
    PK(CPF)

Telefone(Telefone, AlunoCPF)
    FK(AlunoCPF) ref Aluno(CPF)
    PK(Telefone, AlunoCPF)
```

Relacionamentos N-ários, N>2



1. Criar uma nova relação **Trabalha ()**
2. Criar Atributo(s) simples
3. Criar FK para todas as relações

```
Funcionário (CPF, Nome)
```

```
PK(CPF)
```

```
Cargo (Nome)
```

```
PK(Nome)
```

```
Projeto (Código, Descrição)
```

```
PK(Código)
```

```
Trabalha (FCPF, CNome, PCódigo)
```

```
FK(FCPF) ref Funcionário(CPF)
```

```
FK(CNome) ref Cargo(Nome)
```

```
FK(PCódigo) ref Projeto(Código)
```

4. Criar PK com todas as relações que não sejam 1

```
Funcionário (CPF, Nome)
```

```
PK(CPF)
```

```
Cargo (Nome)
```

```
PK(Nome)
```

```
Projeto (Código, Descrição)
```

```
PK(Código)
```

```
Trabalha (FCPF, CNome, PCódigo)
```

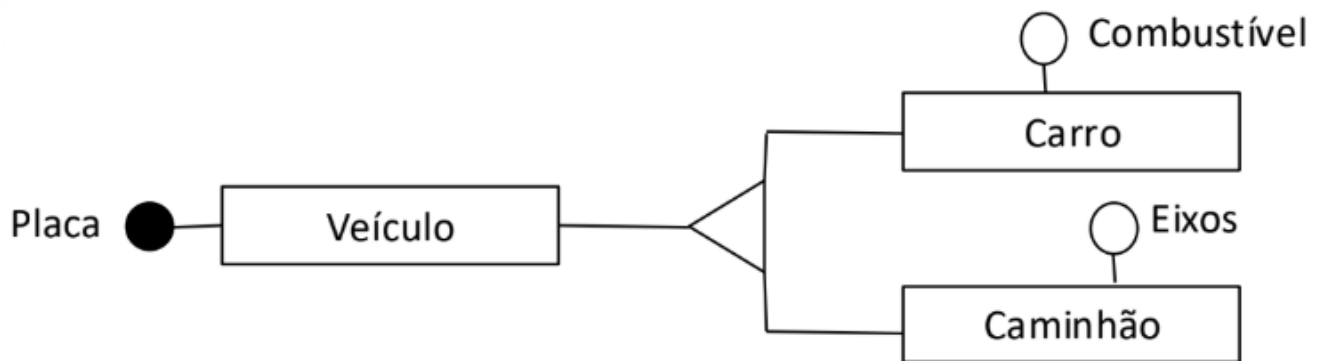
```
FK(FCPF) ref Funcionário(CPF)
```

```
FK(CNome) ref Cargo(Nome)
```

```
FK(PCódigo) ref Projeto(Código)
```

```
PK(FCPF, PCódigo)
```

Mapeamento de Heranças



Partição Única

```
Veículo (Placa, Combustível, Eixos, TipoVeículo*)
      PK(Placa)
```

Particionamento Vertical

```
Veículo (Placa)
      PK(Placa)

Carro (Combustível, VeículoPlaca)
      FK(VeículoPlaca) ref Veículo(Placa)
      PK(VeículoPlaca)

Caminhão (Eixos, VeículoPlaca)
      FK(VeículoPlaca) ref Veículo(Placa)
      PK(VeículoPlaca)
```

Particionamento Horizontal

```
Carro (Combustível, Placa)
      PK(Placa)

Caminhão (Eixos, Placa)
      PK(Placa)
```

Modelo Físico

SQL - Struct Query Language

Tipos de Dados

Numérico (principais):

- integer/int, float, real, numeric(p,n)

Cadeia de caracteres:

- char(n), varchar(n), text

Dados binários:

- blob

Data/tempo:

- date, datetime, timestamp, time, year

Booleano:

- bool, boolean, tinyint(1)

Criando Banco de Dados

```
create database nome_db
--ou
create schema nome_db
```

Criando Tabela

```
create table r (A1 D1, A2 D2, ..., An Dn,
(integrity-constraint1),
...,
(integrity-constraintk));
```

r é o nome da relação

Cada A_i é um nome de atributo no esquema da relação r

D_i é o tipo de dados dos valores no domínio do atributo A_i

Exemplo:

```
create table Departamento (Nome_Depto varchar(50), ID_Depto numeric(5,0) );
```

Restrições de Integridade (RIs)

```
-- Não Nulo
not null
-- Atributo(s) forma(m) uma chave candidata
unique(A1,...,An)
-- PK
```



```
primary key (A1, ..., An)
-- FK
foreign key (Am, ..., An) references r
```

Exemplo:

```
create table Departamento (
    Nome_Depto varchar(50) not null,
    ID_Depto numeric(5,0),-----
    primary key (ID_Depto) ); -----

create table Professor (
    ID_Professor numeric(5,0),-----
    Nome varchar(50) not null,
    CPF char(11), -----
    Salario numeric(8,2),
    ID_Depto numeric(5,0),
    unique (CPF), -----
    primary key (ID_Professor), -----
    constraint fk_depto_prof foreign key (ID_Depto) references
Departamento(ID_Depto) );
-- foreign key (ID_Depto) references Departamento(ID_Depto) );
```

Drop table

```
drop table r
```

Alterar Tabela

```
alter table r add A D
-- Onde A é o nome do atributo a ser adicionado na relação r e D é o domínio
de A
-- Todas as tuplas na relação são associados valores nulos como valor do novo
atributo
alter table r drop A
-- Onde A é o nome do atributo da relação r a ser removido
-- Remoção de atributos não é suportado por muitos SGBDs
```

Exemplo

```
create table Departamento (
    Nome_Depto varchar(50),
    ID_Depto numeric(5,0) );

alter table Departamento add Data_criacao date;
```

```
alter table Departamento add primary key (ID_Depto);
```

```
alter table Departamento drop primary key;
```

```
create table Professor (  
    ID_Professor numeric(5,0),  
    Nome varchar(50) not null,  
    CPF char(11),  
    Salario numeric(8,2),  
    ID_Depto numeric(5,0),  
    unique (CPF),  
    primary key (ID_Professor));
```

```
alter table Professor add constraint fk_depto_prof foreign key (ID_Depto)  
references Departamento(ID_Depto);
```

```
alter table Professor drop foreign key fk_depto_prof;
```

Restrições de atributos e domínios

```
not null  
default <valor>  
check <condição>
```

Exemplo

```
ID_Depto int not null  
check (ID_Depto>0 and ID_Depto<=99999)  
  
semestre varchar(6) default 'Summer' check (semestre in ('Fall',  
'Winter', 'Spring', 'Summer'));  
  
create domain D_NUM as integer  
check (D_NUM > 0 and D_NUM < 21);  
ID_Depto D_NUM not null;
```

Restrições de integridade referencial

```
-- Remoção  
on delete  
cascade (propagação)  
set null (substituição por nulos)  
set default (substituição por um valor default)  
-- Opção default: bloqueio (restrict)
```

```
-- As mesmas opções se aplicam à cláusula  
on update (alteração)
```

Select

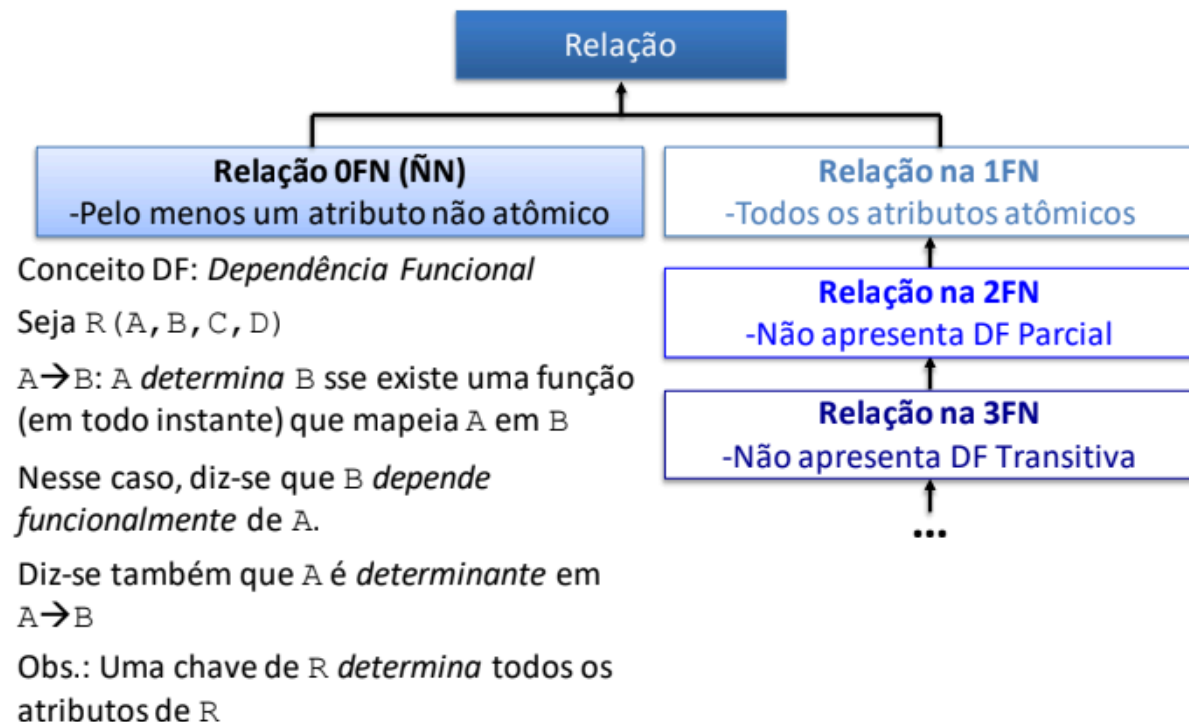
```
select A1, A2, ..., An  
from r1, r2, ..., rm  
where P  
  
-- Ai representa um atributo  
-- ri representa uma relação  
-- P é um predicado  
  
select * from r  
-- * denota todos os atributos  
-- r representa uma relação
```

Modificações do banco de dados

```
insert -- inserir  
  
insert into Departamento  
values ('Departamento de Informática', 21);  
  
insert into Professor (Nome, CPF, ID_Professor)  
values ('Costa', 33333333333, 3);  
  
--  
update -- alterar  
  
update Professor  
set Salario=10000, ID_Depto=21  
where ID_Professor=3;  
  
update Professor  
set Salario=Salario*1.1  
where ID_Depto in (select ID_Depto from Departamento  
where Nome_Depto='Departamento de Informática');  
  
--  
delete -- remover  
  
delete from Professor;  
  
delete from Professor where ID_Professor=3;  
  
delete from Professor where ID_Depto in
```

```
(select ID_Depto from Departamento
where Nome_Depto='Departamento de Informática');
```

Normalização



0FN ou ÑN

Uma relação está na 0FN se ela apresentar algum atributo não atômico

Atributos não-atômicos
(multivalorados)

Departamento

<u>ID_Depto</u>	Nome_Depto	Tel_Secret_Depto	Disciplina		
			<u>ID_Disc</u>	Nome	Créditos
DCC	Departamento de Ciência da Computação	3938-3393	MAB605	Recuperação da Informação	4
			MAB112	Sistemas de Informação	4
			MAB120	Computação I	6
			MAB489	Banco de Dados I	4

1FN

Uma relação está na 1FN se todos os seus atributos forem atômicos

Departamento ($\geq 1FN$)

<u>ID_Depto</u>	Nome_Depto	Tel_Secret_Depto
DCC	Departamento de Ciência da Computação	3938-3393

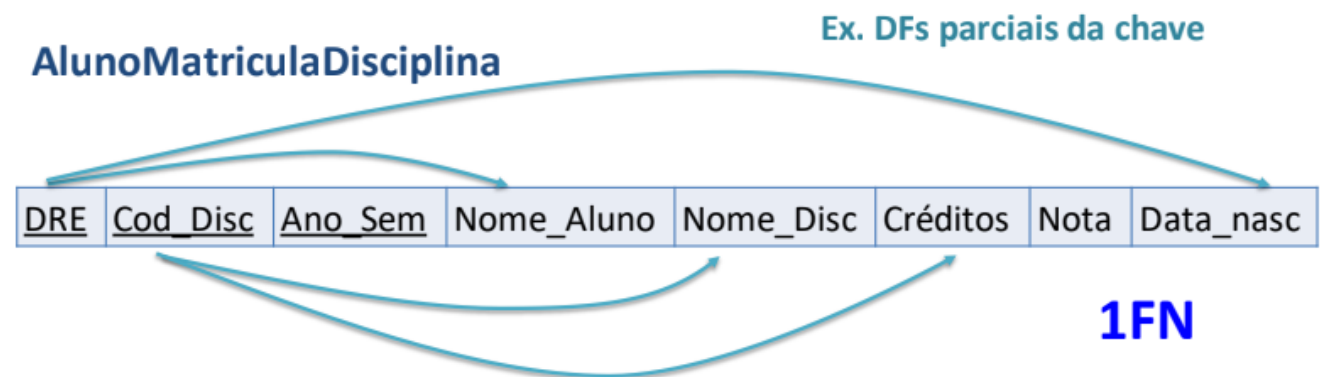
Disciplina ($\geq 1FN$)

<u>ID_Disc</u>	Nome	Créditos	ID_Depto
MAB605	Recuperação da Informação	4	DCC
MAB112	Sistemas de Informação	4	DCC
MAB120	Computação I	6	DCC
MAB489	Banco de Dados I	4	DCC

2FN

Uma relação está na 2FN se ela estiver na 1FN e se ela **não** apresentar dependências funcionais (DFs) parciais da chave

Parcial = "de uma parte"



Aluno

<u>DRE</u>	Nome_Aluno	Data_nasc
------------	------------	-----------

Disciplina

<u>Cod_Disc</u>	Nome_Disc	Créditos
-----------------	-----------	----------

Matricula

<u>DRE</u>	<u>Cod_Disc</u>	<u>Ano_Sem</u>	Nota
------------	-----------------	----------------	------

3FN

Uma relação está na 3FN se ela estiver na 2FN e se ela **não** apresentar dependências funcionais (DFs) transitivas da chave

DisciplinaDepartamento

Ex. DF transitiva da chave

<u>Cod_Disc</u>	Nome_Disc	Créditos	ID_Depto	NomeCoord_Depto
-----------------	-----------	----------	----------	-----------------

Departamento

<u>ID_Depto</u>	NomeCoord_Depto
-----------------	-----------------

Disciplina

<u>Cod_Disc</u>	Nome_Disc	Créditos	ID_Depto
-----------------	-----------	----------	----------

Resumo

0FN



(Eliminar atributos multivalorados)

1FN



(Eliminar dependência funcional parcial)

2FN



(Eliminar dependência funcional transitiva)

3FN

Álgebra Relacional

Operações Básicas

Seleção σ

Seleciona a partir da relação de entrada R, tuplas (linhas) que satisfazem a um determinado predicado:

$$\sigma_{predicado}(R)$$

Predicados permitem expressar comparações do tipo: $=, <, \leq, >, \geq, \neq$

Pode-se relacionar comparações com operadores lógicos (and, or, not):

\wedge, \vee, \neg

$$\sigma_{ID_Depto=17}(Professor)$$

$$\sigma_{ID_Depto=21 \wedge Salario > 9000}(Professor)$$

Projeção II

Copia as colunas das relações de entrada R

$$\Pi_{col1,col2,col3}(Tabela)$$

Exemplo:

$$\Pi_{Nome, Salario}(Professor)$$

Produto Cartesiano \times

Concatena cada tupla de R_1 com todas as tuplas de R_2

$$R_1 \times R_2$$

(Cada elemento de R_1 será associado a todos os elementos de R_2)

Exemplo:

<i>ID_Professor</i>	<i>Nome</i>	<i>CPF</i>	<i>Salario</i>	<i>Professor. ID_Depto</i>	<i>Nome_Depto</i>	<i>Departamento. ID_Depto</i>
1	Silva	11111111111	8.000,00	30	Departamento de Física	17
1	Silva	11111111111	8.000,00	30	Departamento de Informática	21
1	Silva	11111111111	8.000,00	30	Departamento de Letras	30
2	Souza	22222222222	5.400,00	17	Departamento de Física	17
2	Souza	22222222222	5.400,00	17	Departamento de Informática	21
2	Souza	22222222222	5.400,00	17	Departamento de Letras	30
3	Costa	33333333333	10.000,00	21	Departamento de Física	17
3	Costa	33333333333	10.000,00	21	Departamento de Informática	21
3	Costa	33333333333	10.000,00	21	Departamento de Letras	30
4	Gomes	44444444444	5.200,00	17	Departamento de Física	17
4	Gomes	44444444444	5.200,00	17	Departamento de Informática	21
4	Gomes	44444444444	5.200,00	17	Departamento de Letras	30
5	Lima	55555555555	8.700,00	21	Departamento de Física	17
5	Lima	55555555555	8.700,00	21	Departamento de Informática	21
5	Lima	55555555555	8.700,00	21	Departamento de Letras	30

Junção Natural \bowtie

Junção forçando a igualdade naqueles atributos que são comuns a R_1 e R_2 (mesmo nome e domínio nas duas relações)

$$R_1 \bowtie R_2$$

Junção Theta \bowtie_{θ}

Operação binária que combina o produto cartesiano e a seleção em uma única operação

$$R_1 \bowtie_{\theta} R_2 = \sigma_{\theta}(R_1 \times R_2)$$

A condição de junção é geralmente da forma

$$\langle cond_1 \rangle \wedge \langle cond_2 \rangle \wedge \dots \wedge \langle cond_n \rangle$$

onde $\langle \text{condi} \rangle$ é uma expressão $A \theta B$, sendo A um atributo de R_1 , B um atributo de R_2 e θ um dos operadores de comparação $\{=, <, \leq, >, \geq, \neq\}$

Renomeação ρ

Renomear relação e seus atributos

$$\rho_S(A_1, A_2, \dots, A_n)(R)$$

Renomear apenas a relação

$$\rho_S(R)$$

Renomear apenas os atributos

$$\rho_{(A_1, A_2, \dots, A_n)}(R)$$

- Onde: R é a relação original, S é o novo nome dado à relação R , A_1, A_2, \dots, A_n são os novos nomes de atributos

Renomear a relação

$$\rho_S(R)$$

Renomear um atributo

$$\rho_{A_1 \leftarrow R_1}(R)$$

- Onde: A_1 é o novo nome do atributo R_1 da relação original R

Renomear mais de um atributo

$$\rho_{A_1 \leftarrow R_1, \dots, A_n \leftarrow R_n}(R)$$

Atribuição \leftarrow / $=$

VarR ← E

VarR (A₁, A₂, ..., A_n) ← E

- O resultado da expressão **E** é atribuído à variável **VarR**;
A₁, A₂, ..., A_n são os novos nomes dos atributos
- Pode utilizar a variável nas expressões subsequentes
- Ex.: Listar os nomes e os salários dos professores com salário maior do que “Souza”

```
Souza ← σNome="Souza" (Professor)  
Resultado ← (ΠProfessor.Nome, Professor.Salario  
(σProfessor.Salario>Souza.Salario (Professor x  
Souza) ) )
```

União ∪

$$\pi_{\text{nome_coluna}} (\text{Tabela 1}) \cup \pi_{\text{nome_coluna}} (\text{Tabela 2})$$

Diferença −

$$\pi_{\text{nome_coluna}} (\text{Tabela 1}) - \pi_{\text{nome_coluna}} (\text{Tabela 2})$$

Intersecção ∩

$$\pi_{\text{nome_coluna}} (\text{Tabela 1}) \cap \pi_{\text{nome_coluna}} (\text{Tabela 2})$$

Junção Externa (Outer Join)

Computa a junção e então adiciona tuplas extras formadas por uma relação que não casam com tuplas na outra relação no resultado da junção

- Usa valores **null** :

Left Outer Join ⋈

$$\text{Tabela 1} \bowtie \text{Tabela 2}$$

Right Outer Join ⋈

$$\text{Tabela 1} \bowtie \text{Tabela 2}$$

Full Outer Join

Tabela 1 \bowtie Tabela 2

Divisão \div

Dadas as relações $r(R)$ and $s(S)$ tais que $S \subset R$

- R e S são os esquemas das relações r e s , respectivamente
 $r \div s$ é a maior relação $t(R - S)$ tal que

$$t \times s \subseteq r$$

$$r \div s = \pi_{R-S}(r) - \pi_{R-S}((\pi_{R-S}(r) \times s) - r)$$

Onde $R-S$ são as colunas únicas de r (que não estão em s)

A operação divisão é apropriada para consultas que incluem a expressão "para todo"

Exemplo: listar os nomes de todos os clientes que têm conta em todas as agências de Niterói

$$r \div s = \pi_{\text{nome_cliente}, \text{cod_agencia}}(\text{Cliente} \bowtie \text{Conta}) \div \pi_{\text{cod_agencia}}(\sigma_{\text{cidade}='Niterói'}(\text{Agencia}))$$

Agregação γ

Operação de álgebra relacional estendida, permite a utilização de funções agregadas sobre conjuntos de valores

$$G_1, G_2, \dots \gamma F_1(A_1), F_2(A_2), \dots$$

Onde:

- G_n é uma lista de atributos em cada grupo
- F é uma função de agregação
 - avg
 - min
 - max
 - sum
 - count
 - count-distinct

- A_i é um nome de atributo

Exemplo: Listar média dos salários dos professores por Departamento (ID_Depto)

SQL - Consultas

Select

```
select A1, A2, ...  
from tabela  
where P
```

- `distinct` - remove duplicatas
- `select *` - retorna todos atributos (colunas)
- Pode conter expressões aritméticas: $+$, $-$, $*$, e $/$ operando em constantes ou atributos das tuplas
Equivalente ao $\pi_{\text{Nome}}(\text{Professor})$ na álgebra relacional

From

A cláusula `from` lista as relações envolvidas na consulta
Equivalente ao $\text{Tabela 1} \times \text{Tabela 2}$ na álgebra relacional

```
select *  
from Professor, Departamento
```

Gera cada possível par professor – departamento, com todos atributos de ambas relações

Where

A cláusula where especifica condições que o resultado deve satisfazer
Resultados de comparação podem ser combinados usando conectivos lógicos: `and`, `or` e `not`

Corresponde ao predicado $\sigma_{\text{salario} > 7000}(\text{Professor})$ da seleção de álgebra relacional

```
select *  
from Professor  
where salario > 7000
```

Natural Join

```
select Nome
from Professor natural join Departamento
where Salario>9000 and Nome_Depto='Departamento de Informática'
```

Renomeação - AS

SQL permite renomear relações e atributos usando a cláusula AS

```
old_name AS new_name
```

Para renomear pode-se omitir a palavra-chave as

```
old_name new_name
```

Exemplo:

```
select Nome, Salario*12 as Salario_Anual
from Professor
```

Operações com strings - LIKE

Operador like para casamento de strings utilizando padrões descritos com o uso de caracteres especiais:

?: casa com qualquer substring

_ : casa com qualquer caracter

Exemplo: Encontrar os nomes de todos os departamentos cujo nome inclui a substring “ica”

```
select Nome_Depto
from Departamento
where Nome_Depto like '%ica%'
```

Obs: Casar string “100 %”

like '100 \% ' - Usar o caracter de escape '\'

Order By

Ordenar tuplas no resultado

- desc - decrescente

- `asc` - crescente (DEFAULT)

```
-- Listar o nome de todos os professores em ordem alfabética
select distinct Nome
from Professor
order by Nome
-- Ordenar por múltiplos atributos
order by ID_Depto desc, Nome
```

Predicados da Cláusula `Where`

`Between`

```
-- Operador de comparação de intervalos:
select Nome
from Professor
where Salario between 7000 and 9000
```

Comparação de tuplas

```
select Nome
from Professor as P, Departamento as D
where (P.ID_Depto, Nome_Depto) =
      (D.ID_Depto, 'Departamento de Informática')
```

Junções

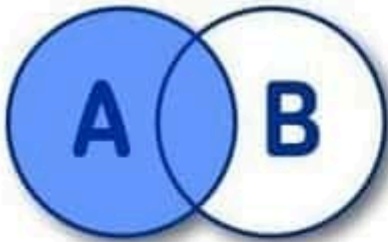
Outer Joins

```
select *
from Professor P left outer join Departamento D
on P.ID_Depto=D.ID_Depto

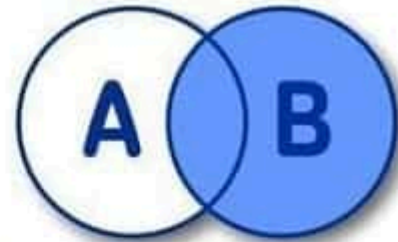
select * from Professor natural left outer join Departamento

select * from Professor left outer join
Departamento using (ID_Depto)
```

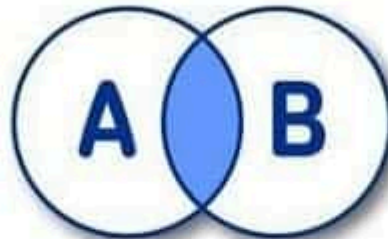
SQL JOINS



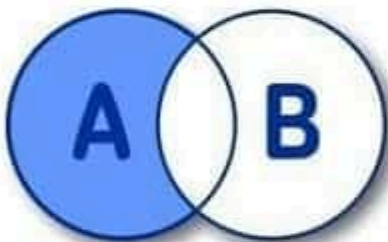
SELECT * FROM
A **LEFT** JOIN B
ON A.KEY = B.KEY



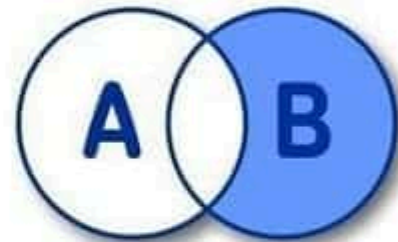
SELECT * FROM
A **RIGHT** JOIN B
ON A.KEY = B.KEY



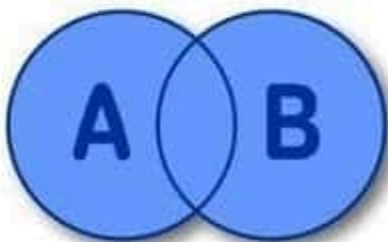
SELECT * FROM
A **INNER** JOIN B
ON A.KEY = B.KEY



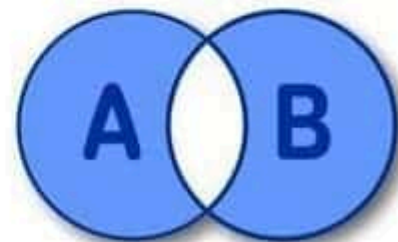
SELECT * FROM A
LEFT JOIN B
ON A.KEY = B.KEY
WHERE B.KEY IS NULL



SELECT * FROM A
RIGHT JOIN B
ON A.KEY = B.KEY
WHERE A.KEY IS NULL




SELECT * FROM A
FULL OUTER JOIN B
ON A.KEY = B.KEY



SELECT * FROM A **FULL
OUTER** JOIN B ON A.KEY =
B.KEY WHERE A.KEY IS
NULL OR B.KEY IS NULL

MySQL JOIN Types


Created by Steve Stedman



SELECT from two tables


```
SELECT *
FROM Table1;

SELECT *
FROM Table2;
```




INNER JOIN

```
SELECT *
FROM Table1 t1
INNER JOIN Table2 t2
ON t1.fk = t2.id;
```



LEFT OUTER JOIN

```
SELECT *
FROM Table1 t1
LEFT OUTER JOIN Table2 t2
ON t1.fk = t2.id;
```




RIGHT OUTER JOIN

```
SELECT *
FROM Table1 t1
RIGHT OUTER JOIN Table2 t2
ON t1.fk = t2.id;
```




SEMI JOIN – Similar to INNER JOIN, with less duplication.

```
SELECT *
FROM Table1 t1
WHERE EXISTS (SELECT 1
FROM Table2 t2
WHERE t1.fk = t2.id
);
```




ANTI SEMI JOIN

```
SELECT *
FROM Table1 t1
WHERE NOT EXISTS (SELECT 1
FROM Table2 t2
WHERE t1.fk = t2.id
);
```




LEFT OUTER JOIN with exclusion

```
SELECT *
FROM Table1 t1
LEFT OUTER JOIN Table2 t2
ON t1.fk = t2.id
WHERE t2.id is null;
```



RIGHT OUTER JOIN with exclusion

```
SELECT *
FROM Table1 t1
RIGHT OUTER JOIN Table2 t2
ON t1.fk = t2.id
WHERE t1.fk is null;
```




FULL OUTER JOIN

```
SELECT * FROM Table1 t1
LEFT OUTER JOIN Table2 t2
ON t1.fk = t2.id
UNION
SELECT * FROM Table1 t1
RIGHT OUTER JOIN Table2 t2
ON t1.fk = t2.id;
```




FULL OUTER JOIN with exclusion

```
SELECT * FROM Table1 t1
LEFT OUTER JOIN Table2 t2
ON t1.fk = t2.id
WHERE t2.id IS NOT NULL
UNION
SELECT * FROM Table1 t1
RIGHT OUTER JOIN Table2 t2
ON t1.fk = t2.id
WHERE t1.id IS NOT NULL;
```




Two INNER JOINS

```
SELECT *
FROM Table1 t1
INNER JOIN Table2 t2
ON t1.fk = t2.id
INNER JOIN Table3 t3
ON t1.fk_table3 = t3.id;
```



Two LEFT OUTER JOINS

```
SELECT *
FROM Table1 t1
LEFT OUTER JOIN Table2 t2
ON t1.fk = t2.id
LEFT OUTER JOIN Table3 t3
ON t1.fk_table3 = t3.id;
```



INNER JOIN and a LEFT OUTER JOIN

```
SELECT *
FROM Table1 t1
INNER JOIN Table2 t2
ON t1.fk = t2.id
LEFT OUTER JOIN Table3 t3
ON t1.fk_table3 = t3.id;
```

Created By Steve Stedman

<http://SteveStedman.com>

Twitter @SqlEmit

Operações sobre conjuntos

- União: `union`

```
(select Nome_Curso from Graduacao) union (select Nome_Curso from PosGrad)
```


- Diferença: `except`

```
(select ID_Area from Graduacao) except (select ID_Area from PosGrad)
```

- Intersecção: `intersect`

```
(select ID_Area from Graduacao) intersect (select ID_Area from PosGrad)
```

Cada uma das operações automaticamente elimina duplicatas

Para reter todas as duplicatas, usar versão multisets: `union all`, `intersect all` e `except all`

Aninhamento subconsultas

Uma subconsulta é uma consulta SQL aninhada dentro de outra consulta

- **Diferença:** Listar o identificador das áreas apenas com curso de graduação

```
select distinct ID_Area
from Graduacao
where ID_Area not in (
    select distinct ID_Area from PosGrad
)
```

- **Intersecção:** Listar o identificador das áreas que possuem tanto cursos de graduação quanto de pós-graduação

```
select distinct ID_Area
from Graduacao
where ID_Area in (
    select distinct ID_Area from PosGrad
)
```

Funções de Agregação

- `avg`: média aritmética dos valores

```
select ID_Depto, avg(Salario)
from Professor
group by ID_Depto
```

- `min` : mínimo valor
- `max` : máximo valor

```
select max(Salario)
from Professor
```

- `sum` : soma de valores
- `count` : quantidade de valores
- `count (distinct ...)` : quantidade de valores distintos

Having

Enquanto predicados na cláusula `where` são aplicados antes da formação dos grupos, predicados na cláusula `having` são aplicados depois da formação dos grupos

```
select ID_Depto, avg(Salario)
from Professor
group by ID_Depto
having avg(Salario) > 6000

-- OU

select ID_Depto, avg(Salario) as AvgSal
from Professor
group by ID_Depto
having AvgSal > 6000
```

Comparação de Conjuntos

Some

- `<`
- `<=`
- `>`
- `>=`
- `=`
- `<>`

Exemplo: Encontrar nomes dos professores com salário maior que algum (pelo menos um) professor do departamento com identificador igual a 21

```

select distinct T.Nome
from Professor as T, Professor as S
where T.Salario > S.Salario and S.ID_Depto=21

-- Mesma consulta usando cláusula > some
select distinct Nome
from Professor
where Salario > some (
    select Salario
    from Professor
    where ID_Depto=21
)

```

All

- `< all`
- `<= all`
- `> all`
- `>= all`
- `= all` (não é o mesmo que in)
- `<> all` (é idêntico a not in)

Exemplo: Encontrar nomes dos professores com salário maior que o salário de todos os professores do departamento com identificador igual a 17

```

select Nome
from Professor
where Salario > all (
    select Salario
    from Professor
    where ID_Depto=17
)

```

Cardinalidade de conjunto:

O construtor `exists` retorna o valor `true` se a subconsulta argumento não é vazia

- `exists`
- `not exists`

Exemplo: Listar o identificador das áreas que possuem tanto cursos de graduação quanto de pós-graduação

```
select distinct ID_Area
from Graduacao as G
where exists (
    select distinct ID_Area
    from PosGrad as P
    where P.ID_Area=G.ID_Area
)
```

Subconsultas na cláusula **From**

Exemplo: Achar dentre todos os departamentos, o máximo do salário total (de algum dos departamentos)

```
select max(tot_salarios)
from (
    select ID_Depto, sum(Salario)
    as tot_salarios
    from Professor
    group by ID_Depto
) as total_deptos;
```

View

```
create view v as <query expression>
```

Exemplo: Uma visão dos professores sem o seu salário

```
create view Cadastro_Prof as
select ID_Professor, Nome, CPF, ID_Depto
from Professor
```

Visões definidas usando outras visões

```
create view Prof_Depto (Prof, Depto) as
select Nome, Nome_Depto
from Professor natural join Departamento;

create view Prof_Depto_Letras (Nome) as
select Prof
from Prof_Depto
where Depto='Departamento de Letras';
```

Apagando visão

```
drop view v;
```

Atualizando uma visão

Exemplo: Adicionar uma nova tupla na visão Cadastro_Prof definida anteriormente

```
insert into Cadastro_Prof values (6, 'Lopes', '666666666666', 21);
```

Resumo

- Tabela virtual definida através de uma consulta
- Definição fica armazenada no catálogo
- Pode ser usada em um select
- Insert, update e delete com restrições
- Aninhamento permitido (visão sobre visão)
- Visões temporárias apenas para uso imediato em consultas, sem salvamento no catálogo

Criando usuários em MySQL

```
create user 'novousuario'@'<host>' identified by 'password';
```

Exemplo:

```
create user 'joao'@'localhost' identified by '123456';
```

```
create user 'maria'@'localhost' identified by '654321';
```

Removendo usuários

```
drop user 'maria'@'localhost';
```

Especificação de autorizações em SQL

```
-- A declaração grant é usada para conferir autorização  
grant <privilege list>  
on <relation name* or view name>  
to <user list>
```

```
-- *'nome da base de dados'.'nome da tabela'
```

Privilégios

- `select` : permite acesso de leitura para uma relação, ou a habilidade de consultar usando a visão
- `insert` : a habilidade de inserir tuplas
- `update` : a habilidade de alterar usando a declaração update em SQL
- `delete` : a habilidade de remover tuplas
- `all privileges` : usada como uma forma abreviada para conceder todos os privilégios

```
show privileges;  
  
grant select  
on Departamento  
to 'joao'@'localhost', 'maria'@'localhost';
```

A declaração `revoke` é usada para revogar autorização

```
revoke select  
on Departamento  
from 'joao'@'localhost', 'maria'@'localhost'
```

Triggers

São ações especificadas pelos usuários, e que são executadas automaticamente quando ocorrer alguma operação que cause a modificação dos dados de uma tabela

```
CREATE  
    [DEFINER = { user | CURRENT_USER }]  
    TRIGGER trigger_name  
    trigger_time trigger_event  
    ON tbl_name FOR EACH ROW  
    [trigger_order]  
    trigger_body  
  
-- trigger_time: { BEFORE | AFTER }  
-- trigger_event: { INSERT | UPDATE | DELETE }  
-- trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

Removendo gatilho

```
DROP TRIGGER trigger_name;
```

Mostrando os gatilhos definidos:

```
SHOW TRIGGERS;
```

Por Linha (FOR EACH ROW)

O código do Gatilho é executado a cada alteração em determinada linha. Variáveis especiais são criadas, dentre estas estão:

NEW: refere-se aos valores associados a uma nova linha a ser inserida (`insert`) ou aos novos valores de uma linha já existente (`update`). Não aplicável em delete.

OLD: refere-se aos valores associados a uma linha que acabou de ser removida (`delete`) ou aos antigos valores de uma linha já existente (`update`). Não aplicável em `insert`.

Não aplicável em `insert`.

Exemplos:

```
DELIMITER $$
CREATE TRIGGER Tgr_Venda_Insert
AFTER INSERT ON Venda
FOR EACH ROW
BEGIN
    UPDATE Produto SET Quant_Estoque = Quant_Estoque - NEW.Quant
    WHERE ID_Produto = NEW.ID_Produto;

CREATE TRIGGER Tgr_Venda_Delete
AFTER DELETE ON Venda
FOR EACH ROW
BEGIN
    UPDATE Produto SET Quant_Estoque = Quant_Estoque + OLD.Quant
    WHERE ID_Produto = OLD.ID_Produto;
END$$

CREATE TRIGGER Tgr_Controle_Estoque
BEFORE UPDATE ON Produto
FOR EACH ROW
BEGIN
    if NEW.Quant_Estoque < 0 then
        signal sqlstate '45000' set message_text = 'Nao ha produto suficiente
em estoque para atender o pedido, venda nao pode ser realizada.';
```

```
end if;
```

```
CREATE TRIGGER Tgr_Controle_Estoque
```

```
AFTER UPDATE ON Produto
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    DECLARE qtd integer;
```

```
    select Quant_Estoque into qtd
```

```
    from Produto
```

```
    where ID_Produto=OLD.ID_Produto;
```

```
    if qtd < 0 then
```

```
        signal sqlstate '45000' set message_text = 'Nao ha produto  
suficiente em estoque para atender o pedido, venda nao pode ser realizada.';
```

```
    end if;
```

```
END$$
```

```
DELIMITER ;
```