Base de Dados

1 – Intro

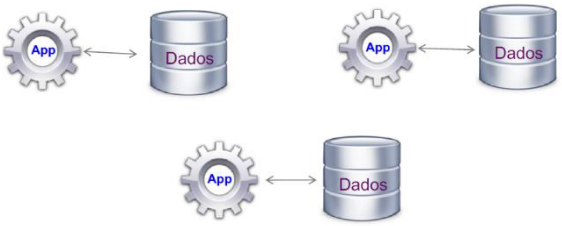
Base de Dados (BD): uma coleção organizada de dados que estão relacionados e que podem ser partilhados por múltiplas aplicações.

- ## Processamento isolado de dados

Dados isolados: cada aplicação gere os seus próprios dados.

Problemas:

- Dados podem estar replicados
- Dados de diferentes organizações e formatos
- Sync problems -> Incoerências



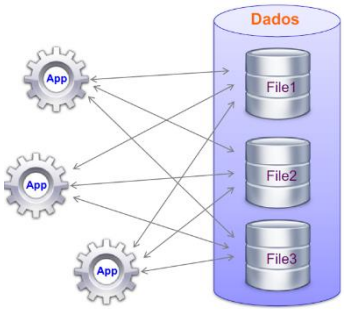
- ## Sistema de gestão de ficheiros

Dados organizados e armazenados em ficheiros partilhados por várias aplicações.

Cada aplicação acede diretamente aos ficheiros e usa uma interface própria.

Problemas:

- Acesso concorrente aos dados
- Integridade
- Segurança



- ## Sistema de Gestão de Base de Dados (SGBD)

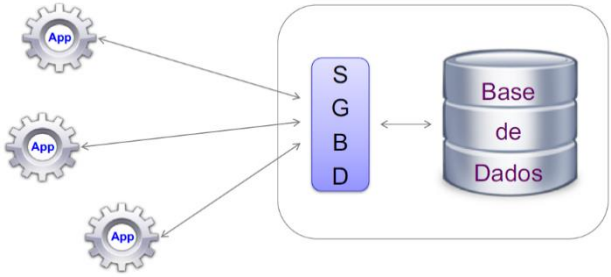
SGBD: generalpurpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications.

- Defining: Especificação do tipo de dados, estruturas de dados e restrições

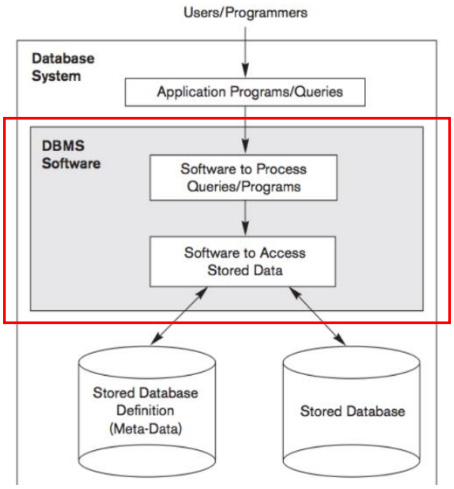
- Construction: Processo de armazenamento de dados

- Manipulating: Envolve operações como a pesquisa e obtenção de dados

- Sharing: Acesso simultâneo aos dados por parte de vários utilizadores e programas



- **Características SGBD**



- Acesso mediado pelo SGBD (Entidade única a operar com a BD)
- Interface de acesso (escolhe detalhes)
- Elevada abstração
- Dados integrados numa mesma unidade de armazenamento

-> Data Independence

- **Vantagens SGBD**

- Independência entre programas e dados
- Integridade dos dados: Controlo dos dados de acordo com as regras definidas
- Consistência dos dados
- Eficiência no acesso aos dados
- Isolamento users: Cada user parece ser o único
- Melhor gestão do acesso concorrencial
- Serviços de segurança (Controlo de Acessos/Permissões & Codificação de Dados)
- Mecanismos de backup e recuperação de dados
- Administração de dados
- Linguagem de desenho e manipulação de dados

- **Desvantagens SGBD**

- Maiores custos e complexidade na instalação e manutenção
- Centralização dos dados pode ter problemas de:
    -> Tolerância a falhas
    -> Escalabilidade
- Não respondem aos requisitos de alguns cenários

- **Users SGBD**

- Final Users: Usam o sistema c/ determinada finalidade.

- App Programmers: Desenvolvem apps que permitem que users interajam com a BD.

- Administradores da BD: Tratam processos de gestão e manutenção

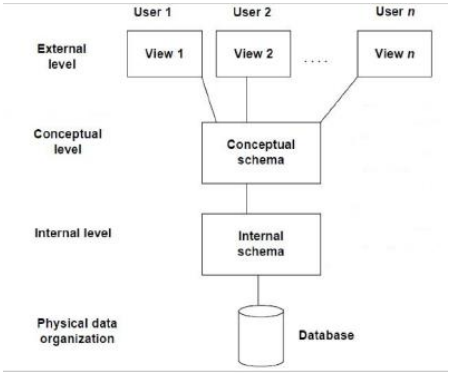
- **Dicionário de dados - SGBD**

- Descritores de objetos da BD: Tabelas, users, regras, vistas, etc.

- Locks – Info sobre os dados em uso e por quem

- Schemas e Mappings

- **SGBD – Arquitetura ANSI/SPARC**



- External level: users da BD

- Conceptual level: designers e administradores da BD

- Internal level: system designers

- **ANSI/SPARC – Nível interno**

Lida c/ implementação física da BD

- Estrutura dos registos em disco

- Indexes e ordenação dos registos

- **ANSI/SPARC – Nível Conceptual**

Descreve a estrutura da BD para os users

- Descreve entidades, tipos de dados, relações, operações, restrições, etc

- Utiliza um modelo de dados para a descrição do esquema conceptual

Abstração: Oculta detalhes de implementação física

- **ANSI/SPARC – Nível Conceptual**

Ofereca vistas da BD adaptadas a cada user

Apresentação dos dados pode ser trabalhada, parte dos dados ocultados, etc.

- **ANSI/SPARC –Independência dos dados**

2 níveis de independência:

Nível Físico: Alterações no nível físico -> s/ impacto no esquema conceptual

Nível Lógico: Alterações no esquema conceptual -> s/ repercutições nos esquemas externos.

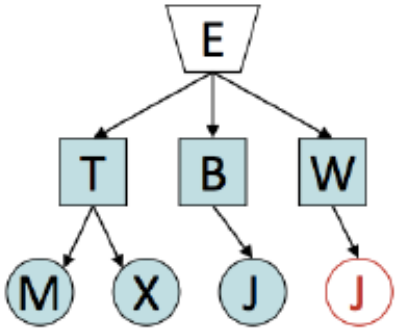
- **Modelo de Base de Dados**

Modelo da BD: coleção de conceitos para descrição lógica de dados (Modelo Lógico)

Esquema (Schema): a descrição de um conjunto particular de dados com recurso a um determinado modelo.

-> Modelo Relacional

- **Modelo Hierárquico**



Dados numa estrutura hierárquica (árvore)

Nós das árvores -> registos, ligados por ponteiros (links)

Registo => conjunto de atributos

Link = Associação Pai-Filho

1:N registo pai -> N registos filhos

- **Desvantagens**

- Adaptado a cenários de acesso sequencial aos dados.

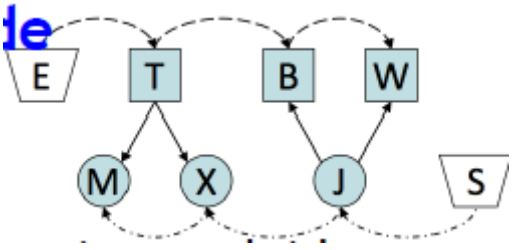
  - Qualquer acesso passa sempre pela root

- Redundância de informação

- Restrições de integridade: Remover um segmento pai significa remover todos os filhos

- Não permite estabelecer associações N:M

- **Modelo de Rede**



1 registo pode estar envolvido em várias associações

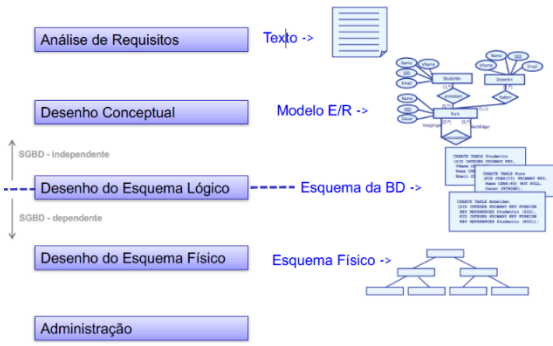
Relações representadas através grafos

- Melhorias na capacidade de navegação.

- Relacionamento 1:N entre dois tipos de registo.

2 – Desenho de BD (Diagrama E/R)

- ## Análise de Requisitos



Processo de comunicação com o cliente

1. Levantamento detalhado de toda a info: entidades, relações, restrições, etc.
2. Filtragem da info
3. Discussão sobre aspetos dúbios e falhas no ponto 1
4. Distinção entre dados e operações

- ## Desenho Conceptual: Conceptualização do mundo real

Modelação trata do mapeamento das entidades e relações do mundo real para conceitos de base de dados

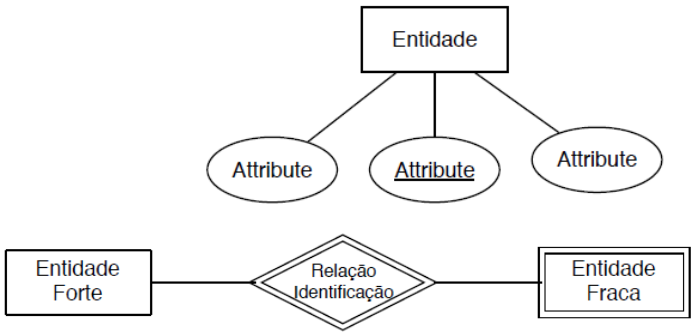
- Não é determinístico

- Nem sempre é claro (óbvio)

Visão abstrata da estrutura da BD.

-> Modelo Entidade/Relação

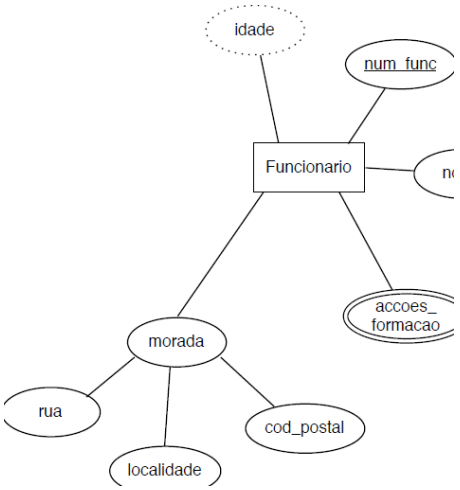
- ## Modelo Entidade/Relação (E/R): Diagrama E/R (DER)



- **Entidades**: algo que existe
- **Atributos**: propriedades das entidades
  - **Atributos chave**: sublinhado
- **Relações**: entre 2 ou mais entidades

- Entidades
  - **Fortes**: Não dependem de outras
  - **Fracas**: Dependem de outras

- Atributos
  - **Derivados** (idade)
  - **Multivalor** (accoes_formacao)
  - **Compostos** (morada)



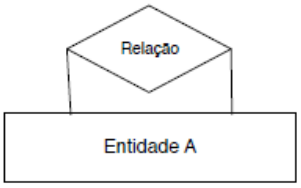
Relações podem ter atributos

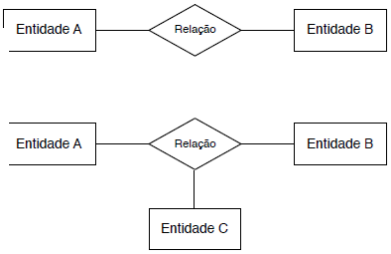
• Classificação relacionamentos

- **Grau**: Num de entidades envolvidas
- **Obrigatoriedade:** Da participação na relação
- **Cardinalidade:** Relação entre o número de ocorrências numa entidade com as respetivas ocorrências na outra com a qual estabelece o relacionamento.

○ Grau da relação

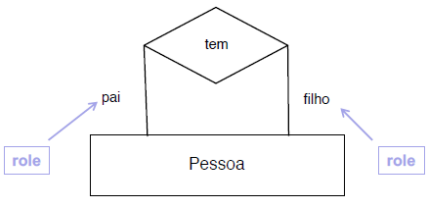
- Uniária
- Binária (mais comum)
- Terciária

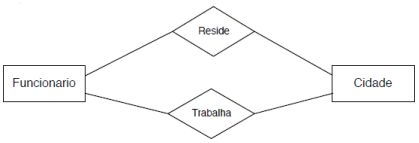




- Relações **Múltiplas**

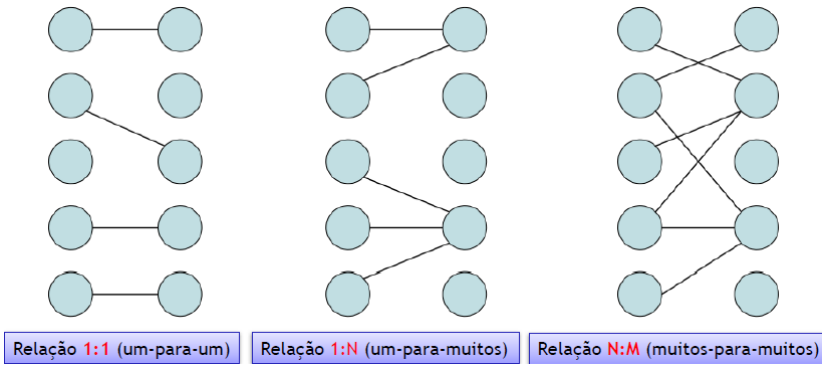
- Relações **Recursivas** (unárias)





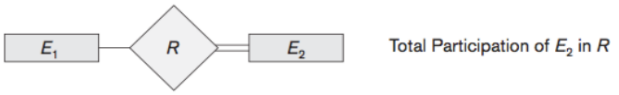
  - Assimétrica – Necessário indicar os **roles**

• Cardinalidade

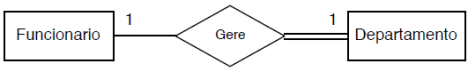


- ## Obrigatoriedade de Participação na Relação

- Participação total (obrigatório): cada instância da entidade participa em pelo menos uma relação do conjunto de relações (linha dupla).



- Participação parcial (opcional): alguma(s) instância(s) da entidade podem não participar em qualquer relação do conjunto de relações.



Notação Alternativa:

- Mínimo:

-> "0" = Opcional

-> "1" = Obrigatória

- Máximo:

-> "1" = No máximo, cada entidade associada a 1 instância

-> "N" = Cada instância associada a várias instâncias.

- Relação 1:1 => 1 funcionário gere 1 departamento && 1 departamento só tem um gestor (funcionário):

- Relação 1:N => 1 funcionário trabalha para 1 só departamento && 1 departamento tem 1 ou + funcionários

- Relação N:M => 1 funcionário pode trabalhar em 1 ou + projetos && 1 projeto tem 1 ou + funcionários a trabalhar nele

- ## Restrições de Integridade

Restrições nos **Atributos**

Cada atributo só tem um valor

Atributos chave são únicos

Atributo (deve / pode ter) ter um valor

Valor do atributo pode ter restrições (>, <, !=, not null, etc)

Cardinalidade

Relação 1:1 (um-para-um)

Relação 1:N (um-para-N)

Relação N:M (muitos-para-muitos)

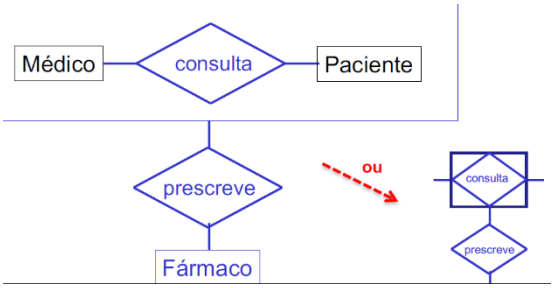
Obrigatoriedade

participação das entidades nas associações

- ## Agregação

Modelar uma relação entre uma entidade e outra relação envolvendo outras entidades.

Entidade Associativa - Permite associar entidades a relacionamentos.



- ## Generalização/Especialização

Classificação de entidades em hierarquia de classes

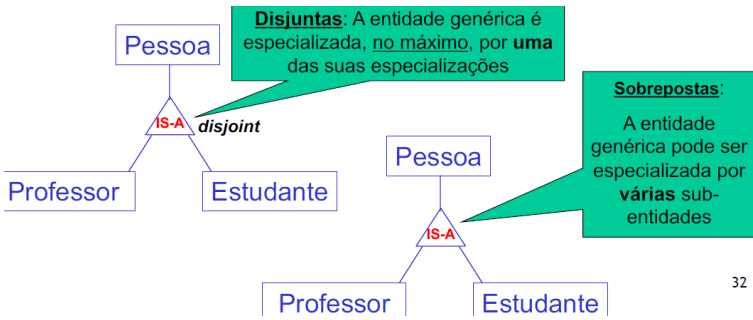
As sub-entidades herdam os atributos das super-entidades

- **Restrições (tipo de especialização)**

Sobreposição (overlaping)

Disjuntas: uma entidade só pode pertencer, no máximo, a uma subclasse de especialização (disjoint – ao lado do Δ).

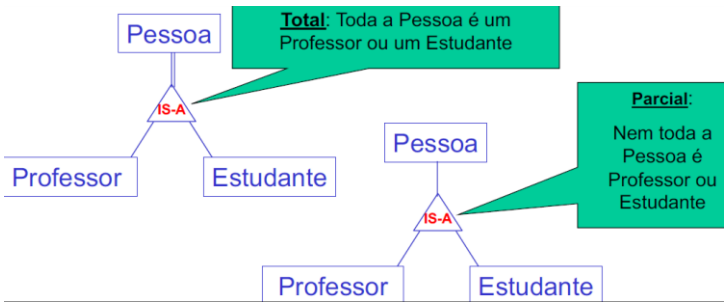
Sobrepostas: uma ocorrência de entidade genérica pode ter mais de uma especialização.



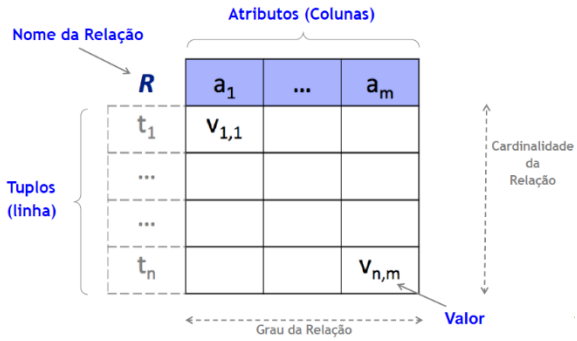
Completude (covering)

Total: uma entidade de nível superior tem de pertencer a pelo menos um subclasse de especialização (linha dupla).

Parcial: pode não pertencer a nenhuma.



3 – Modelo Relacional



- **Conceitos**

Base do Modelo Relacional – **Relação (Tabela)**

- Atributo (A1, A2,..., An):

-> Representam tipo de dados a armazenar

-> Número de atributos de 1 relação => Grau da Relação

-> Devem ter nomes distintos

- Domínio (D1, D2,...,Dn):

-> Tipo de dados

-> Gama de valor possíveis para 1 atributo (Sexo: 'M', 'F')

-> Valores desconhecidos ou não existentes (NULL)

- Esquema da Relação R(A1, A2,...,An): Relational Schema

-> Nome do esquema e lista de atributos: Pessoa (nome, bi)

-> Opcional: inclui o tipo de atributos: Pessoa (nome: String, bi: integer)

- Relação r(R):

-> Estrutura bidimensional c/ determinado esquema e 0 ou + instâncias (tuplos)

-> Subconjunto do prod. Cartesiano: r(R) ⊆ (dom(A1) X dom(A2) X ... X dom(An))

- Tuplo: Linha de uma relação

-> Distintos numa relação

-> Número de tuplos => Cardinalidade da relação

-> Ex: t = < (Nmec, 65022), (Nome, João), (Curso, LEI) >

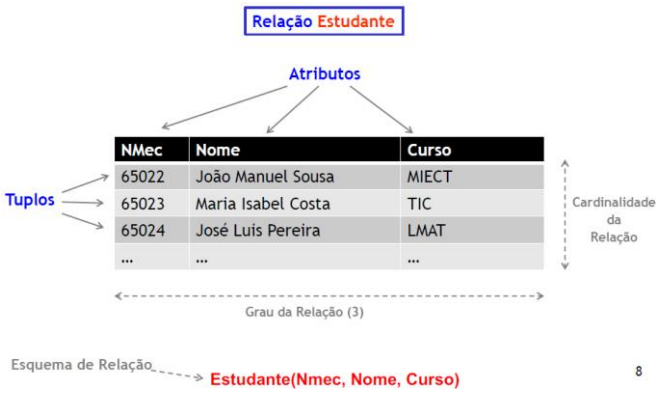
- Atomicidade:

-> valor de um atributo num tuplo é atómico

- Esquema da Base de Dados (Database Schema):

-> conjunto de todos os esquemas

D = {R1(X1), ..., Rn(Xn)}



- Relação – **Chaves**

- Superchave: conjunto de atributos que identificam de forma única os tuplos da relação

- Chave Candidata: subconjunto de atributos de uma superchave que não pode ser reduzido sem perder essa qualidade de superchave

- Chave Primária: chave principal selecionada de entre as chaves candidatas

Não pode ter valor NULL

Elemento "natural" de identificação; Atributo cujo valor nunca é alterado.

- Chave Única: chave candidata não eleita como primária

- Chave Estrangeira: conjunto de um ou mais atributos que é chave primária noutra relação

- **Restrições de Integridade:** Regras p/ garantir a integridade dos dados

<u>TIPOS:</u>

- Domínio - dos atributos. Forma mais elementar de integridade. Os campos devem obedecer ao tipo de dados e às restrições de valores admitidos para um atributo

- Entidade - cada tuplo deve ser identificado de forma única com recurso a uma chave primária que não se repete e não pode ser null (condição de set)

- Referencial - o valor de uma chave estrangeira ou é null ou contém um valor que é chave primária na relação de onde foi importada

1. Representação da Informação
   - Numa base de dados relacional, todos os dados, incluindo o próprio dicionário de dados, são representados de uma só forma, em tabelas bidimensionais.

2. Acesso garantido
   - Cada elemento de dados fica bem determinado pela combinação do nome da tabela onde está armazenado, valor da chave primária e respectiva coluna (atributo).

3. Suporte sistemático de valores nulos (NULL)
   - Valores NULL são suportados para representar informação não disponível ou não aplicável, independentemente do domínio dos respectivos atributos.

4. Catálogo activo e disponível
   - Os metadados são representados e acedidos da mesma forma que os próprios dados.

17

5. Linguagem completa
   - Apesar de um sistema relacional poder suportar várias linguagens, deverá existir pelo menos uma linguagem com as seguintes características:
     - Manipulação de dados, com possibilidade de utilização interativa ou em programas de aplicação.
     - Definição de dados.
     - Definição de views.
     - Definição de restrições de integridade.
     - Definição de acessos (autorizações).
     - Manipulação de transações (commit, rollback, etc.).

6. Regra da atualização de vistas (view)
   - Numa vista, todos os dados modificados (em atributos actualizáveis) devem ver essas modificações traduzidas nas tabelas base.

7. Operações de alto-nível
   - Capacidade de tratar uma tabela (base ou virtual) como se fosse um simples operando (ou seja, utilização de uma linguagem set-oriented), tanto em operações de consulta como de atualização ou eliminação.

18

8. Independência física dos dados
   - Alterações na organização física dos ficheiros da base de dados ou nos métodos de acesso a esses ficheiros (nível interno) não devem afectar o nível lógico.

9. Independência lógica dos dados
   - Alterações no esquema da base de dados (nível lógico), que não envolvam remoção de elementos, não devem afectar o nível externo.

10. Restrições de integridade
    - As restrições de integridade devem poder ser especificadas numa linguagem relacional, independentemente dos programas de aplicação, e armazenadas no dicionário de dados.

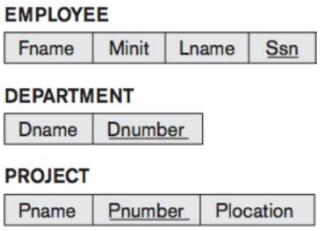
11. Independência da localização
    - O facto de uma base de dados estar centralizada numa máquina, ou distribuída por várias máquinas, não deve repercutir-se ao nível da manipulação dos dados.

12. Não subversão
    - Se existir no sistema uma linguagem de mais baixo-nível (tipo record-oriented), ela não deverá permitir ultrapassar as restrições de integridade e segurança.

- **Conversão DER -> Modelo Relacional**

Cada conjunto de entidades e relações do DER vai gerar uma única relação (tabela)

1. Entidade Regular

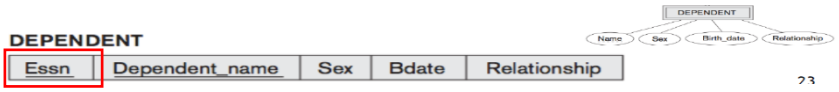


Para cada entidade -> criar 1 relação (tabela)

- Incluir todos os atributos

- Selecionar 1 das chaves E para chave primária

2. Entidade Fraca



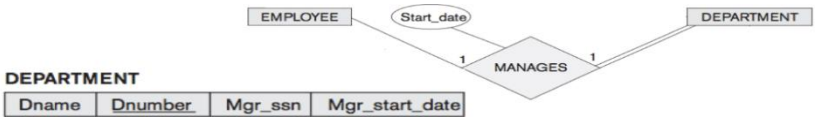
Cada entidade fraca -> 1 relação

- Chave estrangeira: atributos + chave primária da entidade dominante

- Atributos compostos da entidade fraca ficam como elementos singulares

- Chave primária: Combinação chave primária dominante + chave parcial fraca

3. Relacionamento 1:1



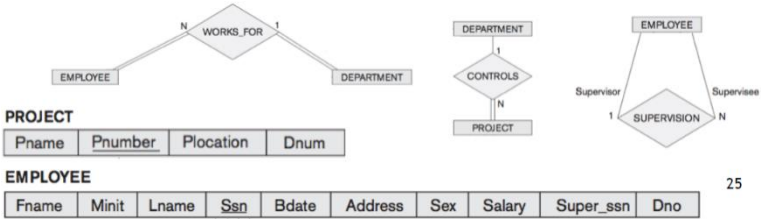
Escolher uma das relações

- chave estrangeira a chave primária da outra relação.

- Incluir nela eventuais atributos do relacionamento

- Deve ser escolhida a relação com participação **total**.

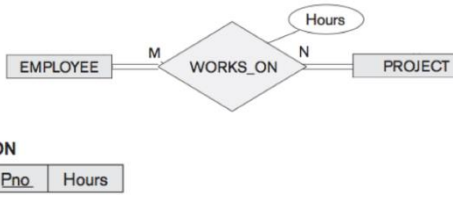
4. Relacionamento 1:N



- Lado N: incluir como chave estrangeira a chave primária do lado 1

+ incluir os atributos do relacionamento

5. Relacionamento N:M

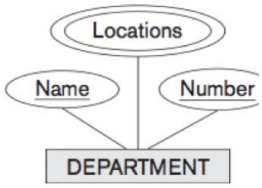


Criar uma relação para cada relacionamento

- Chave estrangeira: Chaves primárias das relações que participam. => Chaves combinadas formam a chave primária da relação

- Incluir atributos do relacionamento.

6. Atributo Multi-Valor



Para cada atributo criar uma tabela

- Incluir um atributo correspondente ao multi-valor

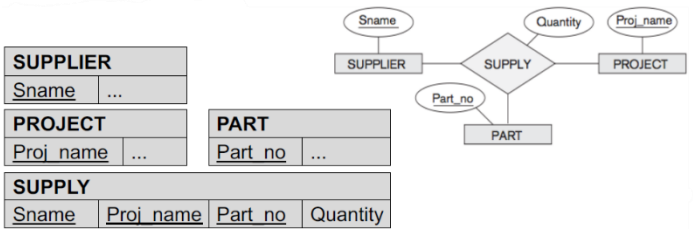
- Incluir chave primária da relação que tem o multi-valor

- Chave primária é a combinação das ^^^^^

DEPT_LOCATIONS

Dnumber	Dlocation

7. Relacionamento n-ário



SUPPLIER

Sname	...

PROJECT

Proj_name	...

PART

Part_no	...

SUPPLY

Sname	Proj_name	Part_no	Quantity

Para cada criar uma tabela

- Chaves estrangeiras: chaves primárias das relações que representam as entidades participantes.

- Incluir atributos do relacionamento

- Chave primária: Combinação chaves estrangeiras

8. Especialização

EMPLOYEE

Ssn	Fname	Minit	Lname	Birth_date	Address	Job_type

SECRETARY

Ssn	Typing_speed

TECHNICIAN

Ssn	Tgrade

ENGINEER

Ssn	Eng_type

a. Especialização: Método 1

Relação para entidade maior nível

- Criar relação para cada entidade nível inferior

- **Incluir** (cada uma): chave primária da maior + atributos locais

Funciona com qualquer tipo de especialização: Total/Parcial, Disjunta/Sobreposta

b. Especialização: Método 2

Criar uma relação para cada entidade nível inferior

- **Incluir:** atributos da superclasse + atributos locais

CAR

Vehicle_id	License_plate_no	Price	Max_speed	No_of_passengers

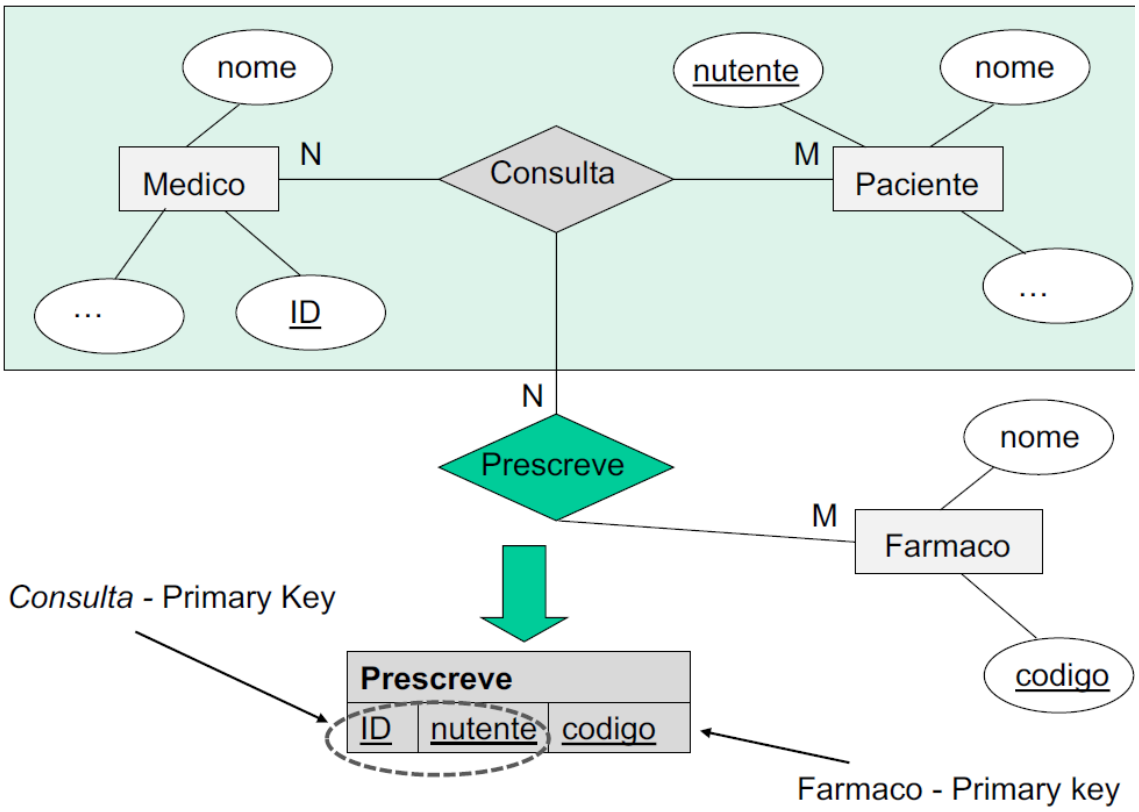
TRUCK

Vehicle_id	License_plate_no	Price	No_of_axles	Tonnage

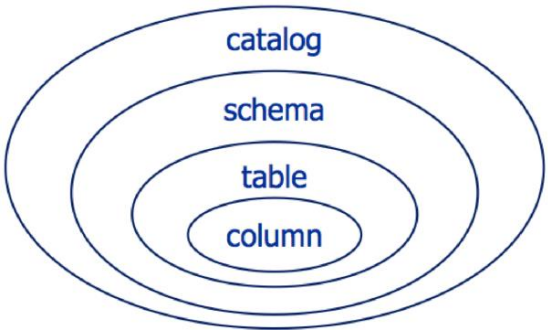
<u>Só funciona com especialização total.</u>

<u>Só se recomenda em especializações disjuntas pois nas sobrepostas há duplicação de informação da mesma entidade por várias relações (tabelas).</u>

9. Agregação



4 – SQL – Data Definiton Language (DDL – Sublinguagem)



SQL: Linguagem para definir, manipular e questionar uma Base de Dados Relacional

DDL: Data Definition Language

Tabela – Relação; Linha – Tuplo; Coluna – Atributo.

- **Criar e Eliminar BD**

CREATE DATABASE dbname;

DROP DATABASE dbname;

```
CREATE DATABASE dbname;
dbname - nome da base de dados a criar

CREATE DATABASE COMPANY;
```

```
DROP DATABASE dbname;
dbname - nome da base de dados a eliminar

DROP DATABASE COMPANY;
```

- **Schema**

Schema: "namespace" que agrupa tabelas e outros elementos pertencentes à mesma aplicação

CREATE SCHEMA schemaname [AUTHORIZATION username];

DROP SCHEMA schemaname;

```
CREATE SCHEMA schemaname [AUTHORIZATION username];

CREATE SCHEMA COMPANY AUTHORIZATION 'CCosta';
```

```
DROP SCHEMA schemaname;

DROP SCHEMA COMPANY;
```

- **Tipo de dados SQL**

- Numbers

- Characters, Strings

- Data e time

- Binary Objects

char(n)
- cadeia de caracteres de tamanho fixo n

varchar(n)
- cadeia de caracteres com tamanho máximo n

int
- números inteiros (4 bytes)

numeric(precisão, escala)
- números reais "sem limite" de tamanho

date e time
- data e hora

boolean*
- valores booleanos

- **Definição de Domínio**

Domain pode conter um valor de defeito (default) e restrições do tipo not null e check

CREATE DOMAIN domainname

```
CREATE DOMAIN domainname
Criação...
CREATE DOMAIN compsalary INTEGER
       NOT NULL CHECK (compsalary > 475);

Utilização...
CREATE TABLE EMPLOYEE (
   ...
   Salary              compsalary,
```

- **Definição de Novo Tipo**

Tipo (alias) é uma alternativa ao domain, mas é mais limitado

CREATE Type... em SQL SERVER

```
Criação...
CREATE TYPE SSN FROM varchar(9) NOT NULL;

Utilização...
CREATE TABLE EMPLOYEE (
   ...
   Ssn                  SSN,
```

- **Criar uma Tabela**

```
CREATE TABLE tbname ( A1 D1, A2 D2, …, An Dn,
                      (integrity-constraint1),
                      ...
                      (integrity-constraintK) );
```

```
tbname - nome da relação (tabela)

CREATE TABLE COMPANY.EMPLOYEE (...)
CREATE TABLE EMPLOYEE (...)

COMPANY - nome do schema
```

```
A1 D1, A2 D2, ..., An Dn
A1...An – Atributos da relação
D1...Dn – Domínio dos atributos
```

```
Restrições de Integridade
integrity-constraint1,
```

CREATE TABLE...
definindo atributos e respectivo domínio.

```
CREATE TABLE EMPLOYEE (
   Fname                VARCHAR(15),
   Minit                CHAR,
   Lname                VARCHAR(15),
   Ssn                  CHAR(9),
   Bdate                DATE,
   Address              VARCHAR(30),
   Sex                  CHAR,
   Salary               DECIMAL(10,2),
   Super_ssn            CHAR(9),
   Dno                  INT);
```

- **Atributos – Valores por Omissão**

DEFAULT: Valores por omissão

CREATE com default ...

```
CREATE TABLE EMPLOYEE (
   Fname                VARCHAR(15),
   ...
   Salary               DECIMAL(10,2)     DEFAULT 0,
```

- **Restrições de Integridade**

Restrições podem ser:

- Coluna: referem-se a apenas uma coluna e são descritas em frente à coluna

- Tabela: referem-se a mais do que a uma coluna e ficam separadas da definição das colunas

  - **Check (P):** impor uma regra a um atributo

Restrição aplicada a cada atributo referenciado sempre que um tuplo é introduzido ou modificado.

Restrição CHECK na tabela...

```
CREATE TABLE DEPARTMENT (
   ...
   Dept_create_date     DATE              NOT NULL,
   Mgr_start_date       DATE,
   ...
   CHECK (Dept_create_date <= Mgr_start_date);
```

Restrição CHECK na coluna...

```
CREATE TABLE EMPLOYEE (
   ...
   Salary               DECIMAL(10,2)     CHECK (Salary > 12),
```

  - **Not Null:** atributo não pode ser null

- **Primary key (A1, …, An):** definir chave primária

Chave primária: Não pode conter valores repetidos ou nulos

Restrição PRIMARY KEY na coluna...

```
CREATE TABLE EMPLOYEE (
   ...
   Ssn                  CHAR(9)              PRIMARY KEY,
```

Restrição PRIMARY KEY na tabela...
(obrigatório se PK for composta por mais do que um atributo)

```
CREATE TABLE EMPLOYEE (
   ...
   Ssn                CHAR(9),
   ...
   PRIMARY KEY (Ssn));
```

- **Unique (A1, …, An):** chaves candidatas não primárias

Restrição UNIQUE na coluna...

```
CREATE TABLE DEPARTMENT (
   Dname                VARCHAR(15)  UNIQUE  NOT NULL,
```

Restrição UNIQUE na tabela...

```
CREATE TABLE DEPARTMENT (
   Dname                VARCHAR(15)
   Dnumber              INT
   PRIMARY KEY (Dnumber),
   UNIQUE (Dname), ... );
```

- Usado para chaves candidatas alternativas

- Não pode conter valores repetidos MAS pode ter valores null

- **Foreign Key:** definir chave estrangeira

Chave estrangeira: deve referenciar uma chave primária ou única

Restrição FOREIGN KEY na coluna...

```
CREATE TABLE EMPLOYEE (
   ...
   Super_ssn    CHAR(9)   REFERENCES EMPLOYEE(Ssn),
   Dno          INT       REFERENCES DEPARTMENT(Dnumber)
```

Restrição FOREIGN KEY na tabela...

```
CREATE TABLE EMPLOYEE (
   ...
   Ssn          CHAR(9),
   Dno          INT                          NOT NULL,
   ...
   FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
   FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber)
```

Integridade Referencial:

Violação quando são inseridos ou eliminados tuplos ou quando os atributos chave estrangeira ou primária são modificados.

Ações alternativas - **On delete/On update:**

restrict - não deixa efetuar a operação

cascade - apaga os registos associados (delete) ou altera a chave estrangeira (update)

set null - a chave estrangeira passa a null.

set default - a chave estrangeira passa a ter o valor por omissão

Restrição FOREIGN KEY

```
CREATE TABLE EMPLOYEE (
   ...
   Ssn          CHAR(9),
   Dno          INT                          NOT NULL,
   ...
   FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn)
               ON DELETE SET NULL   ON UPDATE CASCADE,
   FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber)
               ON DELETE SET DEFAULT ON UPDATE CASCADE);
```

Restrições com nome...

```
CREATE TABLE EMPLOYEE (
   ...
   ...
   CONSTRAINT EMPPK
     PRIMARY KEY (Ssn),
   CONSTRAINT EMPSUPERFK
     FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn)
                 ON DELETE SET NULL   ON UPDATE CASCADE,
   CONSTRAINT EMPDEPTFK
     FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber)
                 ON DELETE SET DEFAULT ON UPDATE CASCADE);
```

- **Tabeça – Drop**

Eliminar a tabela EMPLOYEE

```
DROP TABLE EMPLOYEE;
```

Drop table – remove da BD toda a informação sobre a tabela e os dados (tuplos)

Eliminar a tabela EMPLOYEE com opção CASCADE

```
DROP TABLE EMPLOYEE CASCADE;
```

> Caso gaja violação de restrições, operação é rejeitada

> **CASCADE:** Permite eliminar a tabela + os elementos referenciados

- **Tabela - Alter**

Alter table – modificar o esquema da tabela//restrições existentes

- Adicionar atributos

Todos os tuplos existentes ficam com valor null no novo atributo.

ALTER TABLE tablename ADD Attribute Domain

```
ALTER TABLE EMPLOYEE ADD nofiscal INT;
```

- Adicionar restrições

ALTER TABLE tablename ADD CONSTRAINT name theconstraint

```
ALTER TABLE EMPLOYEE ADD CONSTRAINT salarymin CHECK (Salary >475);
```

- Eliminar atributos

ALTER TABLE tablename DROP COLUMN attributename

```
ALTER TABLE EMPLOYEE DROP COLUMN nofiscal;
```

- Eliminar Restrições

ALTER TABLE tablename DROP CONSTRAINT name

```
ALTER TABLE EMPLOYEE DROP CONSTRAINT salarymin;
```

- Alterar um atributo

ALTER TABLE tablename ALTER Attribute Domain

```
ALTER TABLE EMPLOYEE ALTER COLUMN noFiscal CHAR(9);
```

5 – Álgebra Relacional: Linguagem de Consulta/Interrogação

- Álgebra Relacional: Lnguagem formal do Modelo Relacional

- **Seleção:** σ <selection condition> (R)

- Selecionar um subconjunto de tuplos da relação (t ∈ R) que satisfazem os critérios de seleção

Select Condition é uma expressão boleana

Relation2 ← σ<selection condition>(Relation1): Nova relação (R2) c/ esquema relacional = à original (R1)

- Operadores de comparação:

Comparar 2 atributos//1 atributo c/ 1 valor

- Operandos: Nomes dos atributos e constantes

- Operradores: =, ≠, ≤, ≥ , <, >

$\sigma_{Dno=4}(EMPLOYEE)$

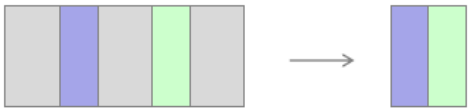
$\sigma_{Salary>30000}(EMPLOYEE)$

- Condições Booleanas:

- AND, OR e NOT

$\sigma_{(Dno=4\ AND\ Salary>25000)\ OR\ (Dno=5\ AND\ Salary>30000)}(EMPLOYEE)$

- **Projeção:** π <attribute list> (R)



<attribute list> = A1, A2, ... Ak, nomes dos atributos da relação R

- Resultado: Nova relação só c/ os k atributos selecionados

-> Removidas as linhas duplicadas do resultado

$\pi_{Lname,\ Fname,\ Salary}(EMPLOYEE)$

- **Renomeação:** ρ R2(B1, B2, ..., Bn) (R1) ou

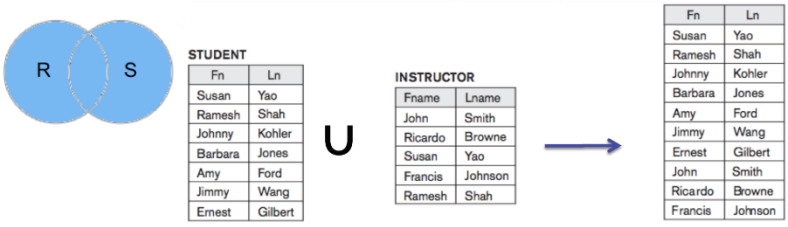
ρ R2(R1) ou ρ (B1, B2, ..., Bn) (R1)

1º - Resultado: Nova relação R2 c/ atributos renomeados (B1, B2, …, Bn)

2º - Só renomeamos a relação

3º - Só renomeamos os atributos

- **União:** $R \cup S = \{t : t \in R \vee t \in S\}$

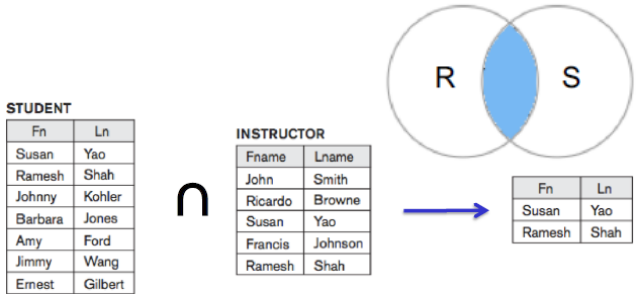


Tabelas têm de ser compatíveis:

- Mesmo número de atributos
- Atributos c/ domínios compatíveis

Resultado: Relação que inclui todos os tuplos de R e de S (Tuplos duplicados são eliminados)

- **Interseção:** $R \cap S = \{t : t \in R \wedge t \in S\}$

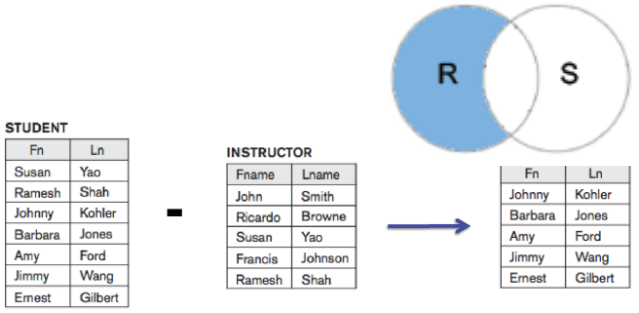


Tabelas têm de ser compatíveis

- Mesmo número de atributos
- Atributos c/ domínios compatíveis

Resultado: Relação que inclui os tuplos que existem simultaneamente em R e S (Tuplos duplicados são eliminados)

- **Diferença:** $R - S = \{t : t \in r \wedge t \notin s\}$



Tabelas têm de ser compatíveis

- Mesmo número de atributos
- Atributos c/ domínios compatíveis

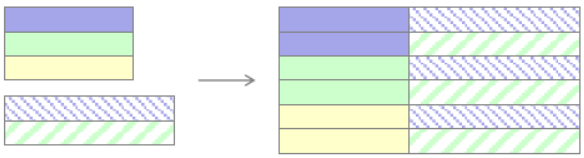
- **União, Interseção e Diferença**

União e Interseção comutativas: $R \cup S = S \cup R$ e $R \cap S = S \cap R$

Diferença não é comutativa: $R - S \neq S - R$

União e Interseção associativas: $R \cup (S \cup T) = (R \cup S) \cup T$ e $(R \cap S) \cap T = R \cap (S \cap T)$

- **Produto Cartesiano:** R X S



Permite-nos combinar tuplos de relações diferentes.

Resultado: nova relação (Q) que combina cada elemento (tuplo) de uma relação (R) com um elemento (tuplo) da outra relação (S)

-> Q(A1, A2, ..., An, B1, B2, ..., Bm) = R(A1, A2, ..., An) X S(B1, B2, ..., Bm)

- **Theta Join:** R $\bowtie_C$ S

$R3 \leftarrow R1 \times R2$ (produto cartesiano)

σ_c (R3) (seleção com condição c)

Resultado das seguintes operações:

C é <join condition> AKA <condition> AND <condition> AND.....

Em <condition> podemos aplicar operadores de comparação

  - Variações da Theta Join

$$DEPT\_MGR \leftarrow DEPARTMENT \bowtie_{Mgr\_ssn=Ssn} EMPLOYEE$$

Figura 1 - nomes dos funcionários gestores de departamentos

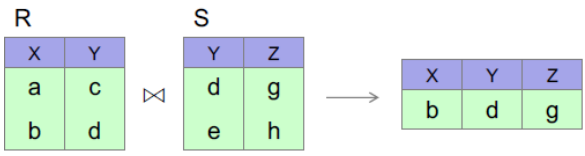
- Equi-Junção (EquiJoin):

Usado c/ o operador "=" na condição de junção

- Junção Natural (Natural Join): R ⋈ S

Condição: Igualdade dos atributos c/ o mesmo nome

Atributos repetidos -> removidos



- **Divisão:** R ÷ S -> R(A1 ,...,Ar,B1,...,Bk) e S(B1,...,Bk)

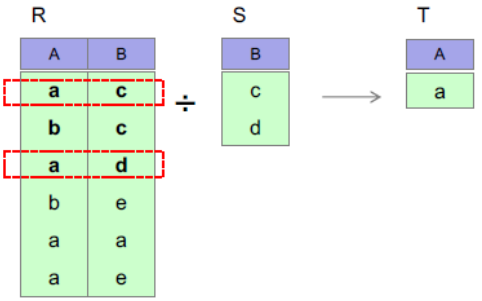
Resultado: todos os tuplos de R1(A1 ,...,Ar) que tenham correspondência com todos os tuplos de S em R2(B1,...,BK)

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

onde π_{R-S} -> $\pi_{(A1,\ldots,Ar)}$

R1 e R2 são projeções de R

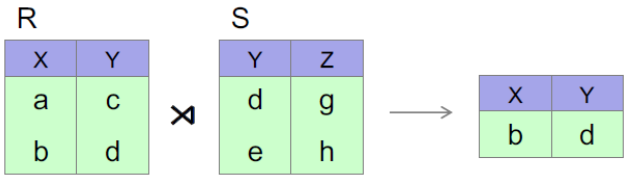
Num atributos de R > num atributos de S



OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{<\text{selection condition}>}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{<\text{attribute list}>}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{<\text{join condition}>} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{<\text{join condition}>} R_2$, OR $R_1 \bowtie_{(<\text{join attributes 1}>),\ (<\text{join attributes 2}>)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{<\text{join condition}>} R_2$, OR $R_1 *_{(<\text{join attributes 1}>),\ (<\text{join attributes 2}>)} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

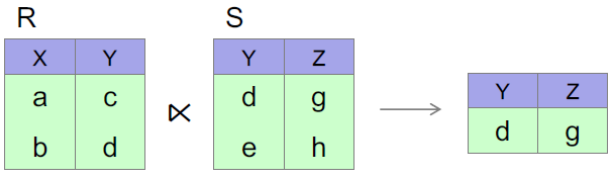
- **Operações Estendidas**
- **Semi-Join**
  - **Left Semi Join:** R ⋉ S = πR (R ⋈ S)

Projeção dos atributos de R na junção natural de R com S



  - **Right Semi Join:** R ⋊ S = πS (R ⋈ S)

Projeção dos atributos de S na junção natural de R com S



- **Inner vs Outer Join**

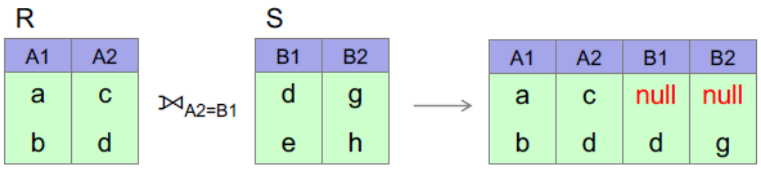
- Inner Join: Operações combinam dados de duas tabelas para serem apresentadas na forma de uma única tabela

Tuplos não relacionados são descartados (Incluindo os tuplos c/ valores Null)

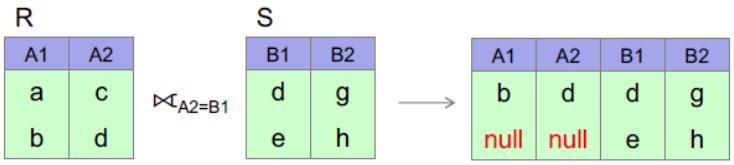
- Outer Join: Incluímos todos os tuplos de uma (ou de ambas) das relações componentes

Atributos que não fazem matching são preenchidos c/ Null.

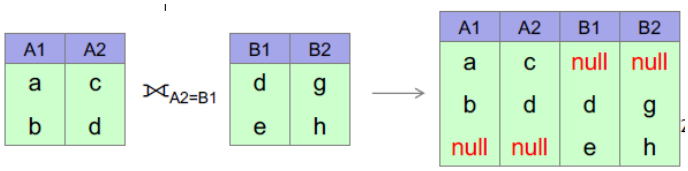
- **Outer Join**
  - **Left Outer Join:** R ⟕ S



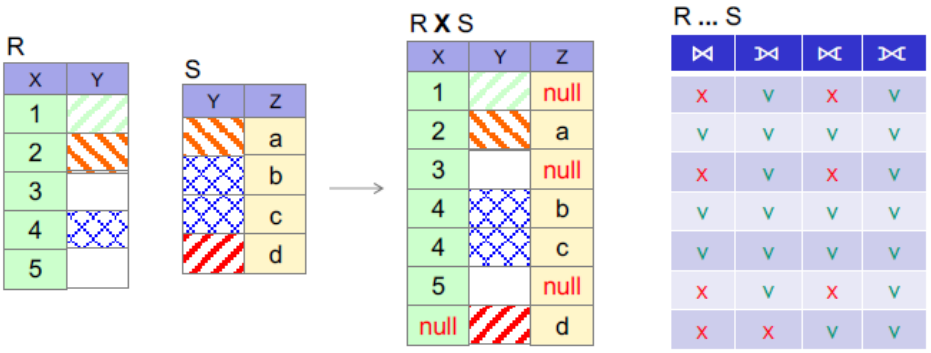
  - **Right Outer Join:** R ⟖ S



  - **Full Outer Join:** R ⟗ S



->Resumo JOIN: Natural | Left Outer | Right Outer | Full Outer



- **Agregação:** <grouping attributes> ℑ <function list> (R)
  - **Avg:** Média dos valores
  - **Min:** mínimo dos valores
  - **Max:** máximo dos valores
  - **Sum:** soma dos valores
  - **Count:** número dos valores

$\pi_{Dno,\ Avg\_Salary=avg(Salary)}(EMPLOYEE)$

$\Im_{count(Ssn),\ avg(Salary)}(EMPLOYEE)$

$Dno\ \Im_{count(Ssn),\ avg(Salary)}(EMPLOYEE)$

$\rho_{R(Dno,\ No\_of\_employees,\ Average\_sal)}\ (Dno\ \Im_{count(Ssn),\ avg(Salary)}(EMPLOYEE))$

Também podem ser <u>usadas em projeções:</u>

- **criar** atributos agregados

- atributos não agregados são **agrupados** de forma a **não** haver valores **repetidos**

6 – SSL DML (Data Manipulation Language)

- **Insert:** Inserir um novo tuplo numa relação

INSERT INTO tablename VALUES (v1,v2,…,vn);

Colunas não indicadas, valores inseridos respeitam ordem de criação dos atributos. Podemos usar NULL//DEFAULT

```
INSERT INTO tablename VALUES (v1,v2,…,vn);
```

```
INSERT INTO EMPLOYEE VALUES
   ('Richard','K','Marini','653298653',NULL,'98
     Oak Forest, Katy, TX', 'M', 37000, '653298653', 4);
```

INSERT INTO tablename (A1,A4,A8,…,An) VALUES (v1,v4,v8,…,vn);

Colunas indicadas, Restantes ficam c/ o seu valor NULL//DEFAULT

```
INSERT INTO tablename (A1,A4,A8,…,An) VALUES (v1,v4,v8,…,vn);
```

```
INSERT INTO EMPLOYEE (Dno, Fname, Lname, Ssn) VALUES
   (4, 'Richard', 'Marini', '653298653');
```

- **Delete:** Remover 1 ou + tuplos de uma relação

Só afeta uma relação. Mas pode propagar-se c/ CASCADE

```
DELETE FROM tablename WHERE match_condition;
```

```
-- remoção (potencial) de um tuplo:
DELETE FROM EMPLOYEE WHERE Ssn='123456789';

-- remoção (potencial) de n tuplos:
DELETE FROM EMPLOYEE WHERE Dno = 5;
-- ou
DELETE FROM EMPLOYEE WHERE Dno > 5 AND Dno < 8;

-- remoção de todos os tuplos da relação:
DELETE FROM EMPLOYEE;
```

- **Update:** Atualizar 1 ou + tuplos de uma relação

Só afeta uma relação. Mas pode propagar-se c/ CASCADE

```
UPDATE tablename SET A1=v1,...,An=vn WHERE match_condition;
```

```
-- atualiza um tuplo:
UPDATE   PROJECT
SET      Plocation = 'Bellaire', Dnum = 5
WHERE    Pnumber=10;

-- atualização (potencial) de n tuplos:
UPDATE  EMPLOYEE
SET     Salary = Salary * 1.1
WHERE   Dno = 5;
```

----------CONSULTAS SIMPLES----------

- **Projeção – SELECT FROM**

```
-- Forma Básica:
SELECT <attribute_list> FROM <table_list>;
```

Seleciona um conjunto de atributos (colunas) de 1 ou + tabelas

```
SELECT * FROM EMPLOYEE;          -- Todas as colunas

SELECT Fname, Ssn FROM EMPLOYEE;  -- Duas colunas
```

$\Pi_{\langle attribute\_list \rangle}(R1)$

  - **SELECT ALL vs DISTINCT**

```
-- Todos os tuplos (por defeito):
SELECT All <attribute_list> FROM <table_list>;

-- Eliminar tuplos repetidos:
SELECT DISTINCT <attribute_list> FROM
<table_list>;
```

Selecionar todos os tuplos vs Eliminar os duplicados

```
SELECT ALL Salary FROM EMPLOYEE;

SELECT DISTINCT Salary FROM EMPLOYEE;
```

- **Seleção – WHERE** $\Pi_{\langle attribute\_list \rangle}(\sigma_{\langle condition \rangle}(R1))$

```
SELECT <attribute_list> FROM <table_list> WHERE <condition>;
```

Selecionar um subconjunto de tuplos da/s tabela/s de acordo c/ uma expressão condicional

```
SELECT Bdate, Address FROM EMPLOYEE
WHERE  Fname='John' AND Minit='B' AND Lname='Smith';
```

- Condição pode conter operadores de comparação AND, OR, NOT

- **Renomeação – Relação, Atributo e Aritmética**

```
-- Renomear
-- Renomear Tabela*
SELECT E.Fname, E.Ssn FROM EMPLOYEE AS E;
ou
SELECT E.Fname AS Fn, E.Ssn AS Ssname FROM EMPLOYEE AS E;

-- Renomear Atributo
SELECT Dno AS DepNumber FROM EMPLOYEE;

-- Renomear Resultado de Operação Aritmética**
SELECT Salary * 0.35 AS SalaryTaxes FROM EMPLOYEE;
```

$\rho_{R2}(R1)$

$\rho_{R2(B1,..,Bn)}(R1)$

$\rho_{B1,..,Bn}(R1)$

Renomear:

- Relações e atributos

- Resultado de uma operação aritmética

- **Reunião, Interseção e Diferença**

<u>Requisitos:</u>

- 2 relações c/ o mesmo número de atributos

- Domínio de cada atributo compatível

$R1 \cup R2$

$R1 \cap R2$

$R1 - R2$

-> UNION, INTERSECT e EXCEPT

Colocados entre 2 queries

Tuplos duplicados eleiminados

<u>Manter duplicados</u>: UNION ALL, EXCEPT ALL* e INTERSECT ALL*

- **UNION**

Projetos (número) que têm 1 funcionário **OU** 1 gestor do departamento que controla o projeto com o último nome Smith?

```
SELECT FROM .....
UNION (ALL)
SELECT FROM .....
```

```
(SELECT DISTINCT Pnumber
FROM   PROJECT, DEPARTMENT, EMPLOYEE
WHERE  Dnum=Dnumber AND Mgr_ssn=Ssn AND Lname='Smith'
UNION
(SELECT DISTINCT Pnumber
FROM   PROJECT, WORKS_ON, EMPLOYEE
WHERE  Pnumber=Pno AND Essn=Ssn AND Lname='Smith' );
```

- **Produto Cartesiano**

Ao usar mais do que uma relação na instrução SELECT FROM o resultado é o produto cartesiano dos 2 conjuntos

```
SELECT * FROM table1, table2, …, tableN;
```

```
-- Exemplo de Produto Cartesiano
SELECT * FROM EMPLOYEE, DEPARTMENT;
```

$R1 \times R2 \times .. \times RN$

```
-- Exemplo de Produto Cartesiano só com dois atributos
-- >> Pode ser visto com Prod. Cartesiano seguido de Projeção
SELECT Ssn, Dname FROM EMPLOYEE, DEPARTMENT;
```

- **Junção de Relações – WHERE**

Produto Cartesiano tem pouco interesse prático

- WHERE permite junção de relações

```
SELECT <atribute_list> FROM <table_list> WHERE <join_condition>;
```

```
-- Exemplo de "select-project-join query"
SELECT Fname, Lname, Address
FROM   EMPLOYEE, DEPARTMENT
WHERE  Dname='Research' AND Dnumber=Dno;
```

Join Condition

- **Junção Várias Relações**

```
/* Questão: Para cada projeto localizado em 'Stafford', queremos
saber o seu número, o número do departamento que o controla e
último nome, endereço e data de nascimento do gestor desse
departamento. */
```

```
SELECT Pnumber, Dnum, Lname, Address, Bdate
FROM   EMPLOYEE, DEPARTMENT, PROJECT
WHERE  Dnum=Dnumber AND Mgr_ssn=Ssn AND Plocation='Stafford';
```

Join Condition 1 Join Condition 2

- **Ambiguidade de Nomes de Atributos**

Full Qualified Name (fqn): relation_name.attribute

```
/* Exemplo: Vamos pegar num dos exemplos anteriores e imaginar
que o atributo Dno de EMPLOYEE se chamava Dnumber... */
```

```
SELECT Fname, Lname, Address
FROM   EMPLOYEE, DEPARTMENT
WHERE  Dname='Research' AND EMPLOYEE.Dnumber=DEPARTMENT.Dnumber;
```

- **Ambiguidade + Renomeação**

Alias: Renomeação de relações

```
/* Exemplo: Para cada Funcionário, pretendemos obter o seu
primeiro e último nome, assim como do seu supervidor. */
```

```
SELECT E.Fname, E.Lname, S.Fname, S.Lname
FROM   EMPLOYEE AS E, EMPLOYEE AS S
WHERE  E.Super_ssn=S.Ssn;
```

- **Comparação de Strings - LIKE**

Wildcards:

% - Zero ou + caracteres

_ - 1 qualquer caracter

```
/* Obter o primeiro e último nome dos funcionários cujo endereço contém a
substring 'Houston,TX'. */
SELECT   Fname, Lname
FROM     EMPLOYEE
WHERE    Address LIKE '%Houston,TX%';

/* Obter o primeiro e último nome dos funcionários nascidos nos anos 50 */
SELECT   Fname, Lname
FROM     EMPLOYEE
WHERE    Bdate LIKE '_ _ 5 _ _ _ _ _ _ _';
```

LIKE ... ESCAPE

ESCAPE: pesquisar wildcards na string

```
/* Nome dos funcionários cujo endereço contém a substring 'Houston%,TX'. */
SELECT   Fname, Lname
FROM     EMPLOYEE
WHERE    Address LIKE '%Houston@%,TX%' ESCAPE '@';
```

- **Operadores Aritméticos & BETWEEN**

Operações aritméticas: +, -, X, ÷

BETWEEN: verificar núm. Entre gama de valores

```
/* Obter o salário, com um aumento de 10%, de todos os trabalhadores do
projeto GalaxyS. */

SELECT   E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal
FROM     EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
WHERE    E.Ssn=W.Essn AND W.Pno=P.Pnumber AND P.Pname='GalaxyS';

/* Funcionários do departamento nº 5 com salário entre 3k e 4k */
SELECT * FROM EMPLOYEE
WHERE  (Salary BETWEEN 30000 AND 40000) AND Dno = 5;
```

- **Ordenação de Resultados: ORDER BY A1, …, Ak**

A1, …, Ak - atributos a ordenar.

1,2,..,k – também podemos usar o número da coluna

ASC (Ascendente) && DESC (Descendente)

```
/* Lista de funcionários e projetos em que trabalham, ordenado por
departamento e, dentro deste, pelo último nome (descendente) e depois o
primeiro */

SELECT    D.Dname, E.Lname, E.Fname, P.Pname
FROM      DEPARTMENT AS D, EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
WHERE     D.Dnumber= E.Dno AND E.Ssn= W.Essn AND W.Pno= P.Pnumber
ORDER BY  D.Dname, E.Lname DESC, E.Fname;

/* ... ORDER BY 1, 2 DESC, 3; */
```

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

- **Tratamento dos NULL**

NULL: valor desconhecido ou que não existe

- Expressão aritmética com null é null

- Verificar se determinado atributo é nulo: IS NULL

- funções de agregação ignoram o null

- **NULL – Lógica de 3 valores**

Retorno comparação lógica: TRUE, FALSE

Comparação com NULL retorna UNKNOWN

```
/* Exemplo 1 */
SELECT  Fname, Salary
FROM EMPLOYEE
WHERE Salary > 40000;

/* Exemplo 2 */
SELECT  Fname, Salary
FROM EMPLOYEE
WHERE Salary > 40000 OR Fname='James';
```

NULL > 40000
UNKNOWN

UNKNOWN OR TRUE

- **IS NOT NULL**

IS NULL: tuplos com determinado atributo a NULL

IS NOT NULL: tuplos com determinado atributo diferente de NULL

```
-- IS NOT NULL
SELECT * FROM EMPLOYEE
WHERE  Super_ssn IS NOT NULL;

-- IS NULL
SELECT * FROM EMPLOYEE
WHERE  Super_ssn IS NULL;
```

- **Junções – JOIN ON** $R \bowtie_C S$

JOIN ON: especificar simultaneamente as tabelas a juntar e a condição de junção

```
SELECT ... FROM (.. [INNER] JOIN .. ON ..) ...;
-- [INNER] é opcional

-- exemplo de Equi-join:
SELECT   Fname, Lname, Address
FROM     (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
WHERE    Dname='Research';
```

- **NATURAL JOIN** $R \bowtie S$

NATURAL JOIN: atributos de junção têm todos o mesmo nome nas duas relações

```
SELECT ... FROM (.. NATURAL JOIN ..) WHERE <condition>;

-- exemplo de Natural Join com renomeação:
SELECT   Fname, Lname, Address
FROM     (EMPLOYEE NATURAL JOIN
                (DEPARTMENT AS DEPT (Dname, Dno, Mssn, Msdate)))
WHERE    Dname='Research';
```

Não disponível e
SQL Server!

- **OUTER JOIN**

LEFT, RIGHT, FULL

$R \bowtie_{A1=B2} S$ $R \bowtie_{A1=B2} S$

$R \bowtie_{A1=B2} S$

```
SELECT .. FROM (.. LEFT|RIGHT|FULL [OUTER] JOIN ..) ...;

/* exemplo de Outer Join com renomeação das relações e
atributos */

SELECT E.Lname AS Employee_name, S.Lname AS Supervisor_name
FROM   (EMPLOYEE AS E LEFT OUTER JOIN EMPLOYEE AS S
        ON E.Super_ssn=S.Ssn);

-- RIGHT OUTER JOIN
-- FULL  OUTER JOIN
```

○ JOIN – Encadeamento

```
SELECT .. FROM (.. JOIN .. JOIN .. JOIN ..) ...;
```

Várias opções de JOIN

uma das relações da junção resulta de outra operação de junção

```
/* Exemplo do slide 17: Para cada projeto localizado em
'Stafford', queremos saber o seu número, o número do
departamento que o controla e último nome, endereço e data de
nascimento do gestor desse departamento. */
-- Nota: Neste caso as join conditions estão à frente do ON

SELECT   Pnumber, Dnum, Lname, Address, Bdate
FROM     ((PROJECT JOIN DEPARTMENT ON Dnum=Dnumber)
          JOIN EMPLOYEE ON Mgr_ssn=Ssn)
WHERE    Plocation='Stafford';
```

• Agregações

<u>Funções de Agregação:</u>

- COUNT, SUM, MAX, MIN, AVG

- Em geral, não são utilizados os tuplos com valor NULL no atributo na função.

<u>Efetuar agregação por atributos:</u>

- GROUP BY <grouping attributes>

<u>Efetuar seleção sobre dados agrupados:</u>

- HAVING <condition>

```
/* Exemplo 1: relativamente aos salários dos funcionários, obter
o valor total, o máximo, o mínimo e o valor médio */
SELECT   SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
FROM     EMPLOYEE;

/* Exemplo 2: Nº de funcionários do departamento 'Research' */
SELECT   COUNT (*)
FROM     EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER
WHERE    DNAME='Research';

/* Exemplo 3: Nº de vencimentos distintos */
SELECT   COUNT (DISTINCT Salary)
FROM     EMPLOYEE;
```

○ GROUP BY

Se existirem valores NULL nos "grouping attribute", então é criado um grupo com todos os tuplos contendo NULL nesses atributos

Exemplos... agregação de atributo(s)

```
/* Exemplo 1: para cada departamento, obter o seu número, o
número de funcionários e a sua média salarial */
SELECT    Dno, COUNT(*), AVG(Salary)
FROM      EMPLOYEE
GROUP BY  Dno;
```

Os "grouping attributes" devem aparecer na cláusula SELECT
Exemplo: Dno

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Borg	888665555		55000	NULL	1

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

```
/* Exemplo 2: agregação com junção de duas relações */
SELECT    Pnumber, Pname, COUNT(*)
FROM      PROJECT JOIN WORKS_ON ON Pnumber=Pno
GROUP BY  Pnumber, Pname;
```

○ GROUP BY… HAVING

- A condição da cláusula WHERE é aplicada antes da criação dos grupos.

- A condição do HAVING é executada depois da criação dos grupos

- Na cláusula HAVING só podemos ter atributos que aparecem em GROUP BY ou funções de agregação

Exemplo... agregação de atributo(s) com seleção

```
/* Exemplo 1: Para cada projeto, com mais de dois funcionários,
obter o seu nome e nº de funcionários que trabalham no projeto
*/
SELECT    Pname, COUNT(*)
FROM      PROJECT join WORKS_ON
          ON Pnumber=Pno
GROUP BY  Pname
HAVING    COUNT(*) > 2;
```

Junção

Pname	Pnumber	Essn	Pno	Hours
ProductX	1	123456789	1	32.5
ProductX	1	453453453	1	20.0
ProductY	2	123456789	2	7.5
ProductY	2	453453453	2	20.0
ProductY	2	333445555	2	10.0
ProductZ	3	666884444	3	40.0
ProductZ	3	333445555	3	10.0
Computerization	10	333445555	10	10.0
Computerization	10	999887777	10	10.0
Computerization	10	987987987	10	35.0
Reorganization	20	333445555	20	10.0
Reorganization	20	987654321	20	15.0
Reorganization	20	888665555	20	NULL
Newbenefits	30	987987987	30	5.0
Newbenefits	30	987654321	30	20.0
Newbenefits	30	999887777	30	30.0

Pname	Count (*)
ProductY	3
Computerization	3
Reorganization	3
Newbenefits	3

```
SELECT    A1,..,An, FAgr1,..Fagrh
FROM      R1,R2,..,Rm
WHERE     <condition_W>
GROUP BY  A1,..,An
HAVING    <condition_H>;
```

$$\Pi_{A1,..,An,\ FAgr1,..Fagrh}\left(\sigma_{<condition\_H>}\left(_{A1,..,An}\Im_{FAgr1,..,Fagrh}\left(\sigma_{<condition\_W>}(R1\times..\times Rm)\right)\right)\right)$$

ORDEM DAS OPERAÇÕES

- **SubConsultas**
  - **Cláusula FROM**

Utilizar o resultado de uma subquery como uma tabela na cláusula FROM, dando-lhe um nome (alias).

Exemplo... agregação de atributo(s) com selecção

```
/* Exemplo 1: Obter uma lista de funcionários com mais de dois
dependentes */
SELECT    Fname, Minit, Lname, Ssn
FROM      Employee JOIN ( SELECT   Essn
                          FROM     DEPENDENT
                          GROUP BY Essn
                          HAVING   count(Essn)>2) AS Dep
          ON Ssn=Dep.Essn;
```

○ Operador IN/NOT IN

WHERE A1,..,An IN (SELECT B1,..,Bn FROM ...):

Selecionar os tuplos em que os atributos existem na subconsulta

Retorna B1, ..., Bn

A1, …, An e B1, …, Bn têm de ter mesmo núm de atributos e domínios compatíveis

Exemplos...

```
/* Exemplo 1: Obter o nome de todos os funcionários que não têm
dependentes */
SELECT    Fname, Minit, Lname
FROM      EMPLOYEE
WHERE     Ssn NOT IN (SELECT Essn FROM DEPENDENT);

/* Exemplo 2: Obter o Ssn de todos os funcionários que trabalham
no mesmo projeto, e o mesmo número de horas, que o funcionário
com o Ssn = '123456789'*/
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     (Pno, Hours) IN ( SELECT  Pno, Hours
                            FROM    WORKS_ON
                            WHERE   Essn='123456789');

/* Exemplo 3: Obter o Ssn de todos os funcionários que trabalham
no projeto nº 1, 2 ou 3 */
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     Pno IN (1, 2, 3);
```

SQL Server não suporta múltiplas colunas!

- **Comparação de Conjuntos (ANY e ALL)**

Operadores que pode ser utilizados para comparar um valor simples com um set ou multiset

ANY (=CASE):

- Selecionar os resultados cujos atributos indicados sejam = > < ou <> (diferentes) do que pelo menos um tuplo da subquery

- ANY é o mesmo que IN

ALL:

combinada com os operadores = > < <>

```
/* Exemplo 1:  Obter o nome dos funcionários cujo salário é
maior do que o salário de todos os trabalhadores do departamento
5 */
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     Salary > ALL ( SELECT  Salary
                         FROM    EMPLOYEE
                         WHERE   Dno=5);

/* Exemplo 2:  Obter o nome dos funcionários cujo salário é
maior do que o salário de algum trabalhador do departamento 5 */
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     Salary > ANY ( SELECT  Salary
                         FROM    EMPLOYEE
                         WHERE   Dno=5);
```

- **Teste de Relações Vazias – EXISTS**

EXISTS retorna:

- True, subconsulta não vazia

- False, subconsulta vazia

Possibilidade de retornar NOT EXISTS

SQL – (NOT) EXISTS

```
/* Exemplo 1: Nomes dos funcionários que não têm dependentes */
SELECT    Fname, Lname
FROM      EMPLOYEE
WHERE     NOT EXISTS ( SELECT  *
                       FROM    DEPENDENT
                       WHERE   Ssn=Essn );
```

- **Tuplos Duplicados – UNIQUE**

UNIQUE: verificar se o resultado de uma subconsulta possui tuplos duplicados

verificar se determinado resultado (relação) é um conjunto ou um multiconjunto

SQL – (NOT) EXISTS

```
/* Exemplo 1: Nomes dos funcionários que gerem um departamento.
(supondo que o mesmo funcionário pode gerir mais do que um
departamento...) */
SELECT    Fname, Lname
FROM      EMPLOYEE
WHERE     UNIQUE     ( SELECT  Mgr_ssn
                       FROM    DEPARTMENT
                       WHERE   Ssn=Mgr_ssn );
```

Não disponível SQL Server

- **SubConsultas Não Correlacionadas**

A subquery (query interior) **não depende** de dados lhe são fornecidos pela query exterior

Query interior é executada uma única vez e o resultado é utilizado no SELECT exterior

SubConsulta Correlacionada

```
/* Exemplo 1: Nome dos funcionário que são gestores de
departamento */

SELECT    Fname, Lname
FROM      EMPLOYEE
WHERE     Ssn IN    (    SELECT  Mgr_ssn
                         FROM    DEPARTMENT
                         WHERE   Mgr_ssn IS NOT NULL);
```

- **SubConsultas Correlacionadas**

A subquery (query interior) depende de dados lhe são fornecidos pela query exterior

- Query interior é executada uma vez para cada resultado do SELECT exterior

SubConsulta Correlacionada

```
/* Exemplo 1: Nome dos funcionários que tem um dependente com o
primeiro nome e sexo igual ao próprio funcionário */

SELECT    E.Fname, E.Lname
FROM      EMPLOYEE AS E
WHERE     E.Ssn IN  (    SELECT  Essn
                         FROM    DEPENDENT AS D
                         WHERE   E.Fname=D.Dependent_name
                                 AND E.Sex=D.Sex );
```