Bases de Dados (2016/12/15 (v239))

Pedro Quaresma

Departamento de Matemática
Universidade de Coimbra

2016/2017

Índice

Bibliografia

Bases de Dados

- Database System Concepts (6th edition), Silberschatz, Korth and Sudarshan, McGraw-Hill, 2010.
- Tecnologia de bases de dados (3ª Edição), Pereira, José Luís Mota, FCA-Ed.Informática, 1998. (68P/PER/3aed).
- Graph Databases, Robison, Weber and Eifrem, O'Reiley, 2013. (68P/ROB.Gra).

MySQL

- Manual de referência do MySQL, `http://dev.mysql.com/doc/`.

PHP/MySQL/Web

- PHP and MySQL Web Development (Developer's Library) (5th Edition) Luke Welling, Laura Thomson. Addison-Wesley Professional, 2014.
- PHP e MYSQL Desenvolvimento Web, Luke Welling e Laura Thomson, Campus, 3ª Edição, 2005, ISBN-13: 9788535217148.
- PHP 5/MySQL Programming, Andy Harris, Premier Press, 2004.
- Programação com PHP 5.3, Carlos Serrão e Joaquim Marques, FCA, 2009.
- Manual de referência do PHP, `http://www.php.net/manual/pt_BR/`.
- Manual de referência do Neo4j, The Neo4j Manual, neotechnology. `https://neo4j.com/docs/`

Apontamentos da disciplina

`http://www.mat.uc.pt/~pedro/lectivos/BasesDados/`

Introdução

- Sistemas de Gestão de Bases de Dados
- Visão dos dados
- Modelos de dados
- Linguagem de Definição de Dados
- Linguagem de Manipulação de Dados
- Gestão de Transacções
- Gestão de Armazenamento
- Administrador da Base de Dados
- Utilizadores da Base de Dados
- Estrutura Global do Sistema

Bases de Dados!?

Objectivos na utilização de sistemas informáticos:

- Cálculo - linguagens de manipulação de dados numéricos, linguagens de programação, `Fortran`, `Lisp`, `C`, ...
- Processamento de informação - linguagens de processamento de informação não numérica, `Cobol`, ... `SQL`.

No primeiro caso o objectivos de cálculo mantém-se, as formas de programação têm evoluído.

No segundo caso o objectivo também se mantém, houve no entanto uma clara mudança na forma de o encarar.

Processamento de Dados I — Ficheiros

Uma primeira aproximação ao problema do processamento de informação (dados) é dada por:

- Linguagem de programação especializada para o processamento de sequências de caracteres (mais do que no cálculo numérico):

  `Cobol, ...`

- Sistema de ficheiros:

  Sistema de ficheiro geridos pelo Sistema Operativo da máquina aonde está a base de dados.

Sistemas de Ficheiros - Inconvenientes

- Redundância e inconsistência de dados:
  - Múltiplos formatos, duplicação de informação em ficheiros diferentes.

Sistemas de Ficheiros - Inconvenientes

- Redundância e inconsistência de dados:
  - Múltiplos formatos, duplicação de informação em ficheiros diferentes.
- Dificuldades no acesso aos dados:
  - Necessidade de escrever um novo programa para efectuar uma nova tarefa.

Sistemas de Ficheiros - Inconvenientes

- Redundância e inconsistência de dados:
  - Múltiplos formatos, duplicação de informação em ficheiros diferentes.
- Dificuldades no acesso aos dados:
  - Necessidade de escrever um novo programa para efectuar uma nova tarefa.
- Isolamento de dados — múltiplos ficheiros e formatos.

Sistemas de Ficheiros - Inconvenientes

- Redundância e inconsistência de dados:
  - Múltiplos formatos, duplicação de informação em ficheiros diferentes.
- Dificuldades no acesso aos dados:
  - Necessidade de escrever um novo programa para efectuar uma nova tarefa.
- Isolamento de dados — múltiplos ficheiros e formatos.
- Problemas de integridade:
  - Restrições de integridade (por exemplo: saldo da conta ≥ 0) estão incluídas no código dos programas.
  - Difícil alterar ou adicionar novas restrições.

Sistemas de Ficheiros - Inconvenientes

- Redundância e inconsistência de dados:
  - Múltiplos formatos, duplicação de informação em ficheiros diferentes.
- Dificuldades no acesso aos dados:
  - Necessidade de escrever um novo programa para efectuar uma nova tarefa.
- Isolamento de dados — múltiplos ficheiros e formatos.
- Problemas de integridade:
  - Restrições de integridade (por exemplo: saldo da conta ≥ 0) estão incluídas no código dos programas.
  - Difícil alterar ou adicionar novas restrições.
- Atomicidade das alterações:
  - Falhas podem colocar a base de dados num estado inconsistente com alterações parciais já efectuadas. Por exemplo; a transferência de dinheiro de uma conta para outra ou deve ser totalmente realizada ou nenhuma alteração deve ser efectuada.

Sistemas de Ficheiros - Inconvenientes

- Redundância e inconsistência de dados:
  - Múltiplos formatos, duplicação de informação em ficheiros diferentes.
- Dificuldades no acesso aos dados:
  - Necessidade de escrever um novo programa para efectuar uma nova tarefa.
- Isolamento de dados — múltiplos ficheiros e formatos.
- Problemas de integridade:
  - Restrições de integridade (por exemplo: saldo da conta ≥ 0) estão incluídas no código dos programas.
  - Difícil alterar ou adicionar novas restrições.
- Atomicidade das alterações:
  - Falhas podem colocar a base de dados num estado inconsistente com alterações parciais já efectuadas. Por exemplo; a transferência de dinheiro de uma conta para outra ou deve ser totalmente realizada ou nenhuma alteração deve ser efectuada.
- Acessos concorrentes por diversos utilizadores:
  - Acessos concorrentes necessários por motivos de eficiência
  - Os acessos concorrentes não controlados podem originar inconsistências.

Sistemas de Ficheiros - Inconvenientes

- Redundância e inconsistência de dados:
  - Múltiplos formatos, duplicação de informação em ficheiros diferentes.
- Dificuldades no acesso aos dados:
  - Necessidade de escrever um novo programa para efectuar uma nova tarefa.
- Isolamento de dados — múltiplos ficheiros e formatos.
- Problemas de integridade:
  - Restrições de integridade (por exemplo: saldo da conta ≥ 0) estão incluídas no código dos programas.
  - Difícil alterar ou adicionar novas restrições.
- Atomicidade das alterações:
  - Falhas podem colocar a base de dados num estado inconsistente com alterações parciais já efectuadas. Por exemplo; a transferência de dinheiro de uma conta para outra ou deve ser totalmente realizada ou nenhuma alteração deve ser efectuada.
- Acessos concorrentes por diversos utilizadores:
  - Acessos concorrentes necessários por motivos de eficiência
  - Os acessos concorrentes não controlados podem originar inconsistências.
- Problemas de segurança.

Processamento de Dados II — SGBDs

- Colecção de dados inter-relacionados (Base de Dados).

Processamento de Dados II — SGBDs

- Colecção de dados inter-relacionados (Base de Dados).
- Conjunto de programas para construir a base de dados (DDL).

Processamento de Dados II — SGBDs

- Colecção de dados inter-relacionados (Base de Dados).
- Conjunto de programas para construir a base de dados (DDL).
- Conjunto de programas para aceder aos dados (DML).

Processamento de Dados II — SGBDs

- Colecção de dados inter-relacionados (Base de Dados).
- Conjunto de programas para construir a base de dados (DDL).
- Conjunto de programas para aceder aos dados (DML).
- Deve fornecer um ambiente (gráficos e/ou não gráficos) de utilização conveniente e eficiente.

Processamento de Dados II — SGBDs

- Colecção de dados inter-relacionados (Base de Dados).
- Conjunto de programas para construir a base de dados (DDL).
- Conjunto de programas para aceder aos dados (DML).
- Deve fornecer um ambiente (gráficos e/ou não gráficos) de utilização conveniente e eficiente.
- Exemplos de aplicações de SGBDs:
  - **Banca:** todas as transacções e movimentos; **Companhias aéreas:** reservas, horários: **Universidades:** inscrições, notas; **Vendas:** clientes, produtos, compras; **Indústria:** produção, inventário, pedidos, cadeia de fornecimento; **Recursos humanos:** registos dos empregados, salários, impostos; **Sistema de gestão de artigos** numa conferência/revista; **lojas "on-line"**; **Bases de Dados Geográficas**; ....

Processamento de Dados II — SGBDs

- Colecção de dados inter-relacionados (Base de Dados).
- Conjunto de programas para construir a base de dados (DDL).
- Conjunto de programas para aceder aos dados (DML).
- Deve fornecer um ambiente (gráficos e/ou não gráficos) de utilização conveniente e eficiente.
- Exemplos de aplicações de SGBDs:
  - **Banca:** todas as transacções e movimentos; **Companhias aéreas:** reservas, horários: **Universidades:** inscrições, notas; **Vendas:** clientes, produtos, compras; **Indústria:** produção, inventário, pedidos, cadeia de fornecimento; **Recursos humanos:** registos dos empregados, salários, impostos; **Sistema de gestão de artigos** numa conferência/revista; **lojas "on-line"**; **Bases de Dados Geográficas**; ....
- Exemplos de SGBDs:
  - Modelo Cliente/Servidor: Oracle; MySQL; PostgreSQL; Informix; ...
  - Ficheiro Único: SQLite; Firebird; ...
  - Modelo "Consulta por Exemplos": MS-Access

SGBDs — Níveis de Abstracção

Nível Físico: descreve como um registo (e.g. cliente) é armazenado

SGBDs — Níveis de Abstracção

Nível Físico: descreve como um registo (e.g. cliente) é armazenado

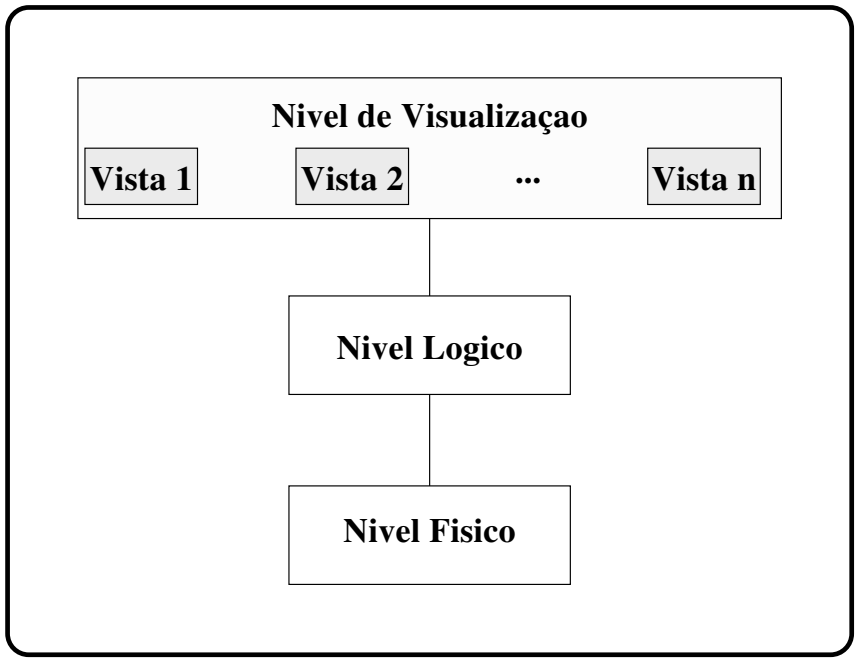
Nível Lógico: descreve os dados armazenados na base de dados, assim como as associações entre os dados

SGBDs — Níveis de Abstracção

Nível Físico: descreve como um registo (e.g. cliente) é armazenado

Nível Lógico: descreve os dados armazenados na base de dados, assim como as associações entre os dados

Nível de Visualização: as aplicações ocultam os detalhes dos tipos de dados. Por motivos de segurança alguma da informação pode ser omitida (Por exemplo: o salário de um funcionário).



Instâncias e Esquemas

- Semelhante a variáveis e constantes das linguagens de programação.
- **Esquema** — a estrutura lógica da base de dados
  - e.g., a base de dados é constituída por informação sobre clientes, contas e as associações entre si.
  - Análogo à declaração de uma variável (de um dado tipo)
  - **Esquema físico**: desenho da base de dados ao nível físico
  - **Esquema lógico**: desenho da base de dados ao nível lógico
- **Instância** — o conteúdo de uma base de dados num instante de tempo
  - Análogo ao valor de uma variável num dado instante
- **Independência física dos dados** — a capacidade de modificar o esquema físico sem alterar o esquema lógico
  - As aplicações dependem do esquema lógico
  - Em geral, as interfaces entre os vários níveis e componentes devem estar bem definidas de modo a que alterações numa parte não influenciem grandemente outras partes.

Modelos de Dados

- Um conjunto de ferramentas para descrever
  - dados
  - associações entre dados
  - semântica dos dados
  - restrições sobre os dados

(▸ Exemplo usando o modelo E-A)

Modelos de Dados

- Um conjunto de ferramentas para descrever
  - dados
  - associações entre dados
  - semântica dos dados
  - restrições sobre os dados
- Modelo Entidade-Associação (Relação) (▸ ver)

(▸ Exemplo usando o modelo E-A)

Modelos de Dados

- Um conjunto de ferramentas para descrever
  - dados
  - associações entre dados
  - semântica dos dados
  - restrições sobre os dados
- Modelo Entidade-Associação (Relação) (▸ ver)
- Modelo Relacional (▸ ver)

(▸ Exemplo usando o modelo E-A)

Modelos de Dados

- Um conjunto de ferramentas para descrever
  - dados
  - associações entre dados
  - semântica dos dados
  - restrições sobre os dados
- Modelo Entidade-Associação (Relação) (▸ ver)
- Modelo Relacional (▸ ver)
- Outros modelos (passado):

(▸ Exemplo usando o modelo E-A)

Modelos de Dados

- Um conjunto de ferramentas para descrever
  - dados
  - associações entre dados
  - semântica dos dados
  - restrições sobre os dados
- Modelo Entidade-Associação (Relação) (ver)
- Modelo Relacional (ver)
- Outros modelos (passado):
  - modelo hierárquico (ver)

(Exemplo usando o modelo E-A)

Modelos de Dados

- Um conjunto de ferramentas para descrever
  - dados
  - associações entre dados
  - semântica dos dados
  - restrições sobre os dados
- Modelo Entidade-Associação (Relação) (ver)
- Modelo Relacional (ver)
- Outros modelos (passado):
  - modelo hierárquico (ver)
  - modelo em rede (ver)

(Exemplo usando o modelo E-A)

Modelos de Dados

- Um conjunto de ferramentas para descrever
  - dados
  - associações entre dados
  - semântica dos dados
  - restrições sobre os dados
- Modelo Entidade-Associação (Relação) (ver)
- Modelo Relacional (ver)
- Outros modelos (passado):
  - modelo hierárquico (ver)
  - modelo em rede (ver)
- Outros modelos (presente/futuro!?):

(Exemplo usando o modelo E-A)

Modelos de Dados

- Um conjunto de ferramentas para descrever
  - dados
  - associações entre dados
  - semântica dos dados
  - restrições sobre os dados
- Modelo Entidade-Associação (Relação) (ver)
- Modelo Relacional (ver)
- Outros modelos (passado):
  - modelo hierárquico (ver)
  - modelo em rede (ver)
- Outros modelos (presente/futuro!?):
  - modelo orientado para objectos (ver)

(Exemplo usando o modelo E-A)

Modelos de Dados

- Um conjunto de ferramentas para descrever
  - dados
  - associações entre dados
  - semântica dos dados
  - restrições sobre os dados
- Modelo Entidade-Associação (Relação) (ver)
- Modelo Relacional (ver)
- Outros modelos (passado):
  - modelo hierárquico (ver)
  - modelo em rede (ver)
- Outros modelos (presente/futuro!?):
  - modelo orientado para objectos (ver)
  - modelos de dados semi-estruturados (ver)

(Exemplo usando o modelo E-A)

Modelos de Dados

- Um conjunto de ferramentas para descrever
  - dados
  - associações entre dados
  - semântica dos dados
  - restrições sobre os dados
- Modelo Entidade-Associação (Relação) (ver)
- Modelo Relacional (ver)
- Outros modelos (passado):
  - modelo hierárquico (ver)
  - modelo em rede (ver)
- Outros modelos (presente/futuro!?):
  - modelo orientado para objectos (ver)
  - modelos de dados semi-estruturados (ver)
  - Modelos baseados em grafos (ver)

(Exemplo usando o modelo E-A)

Modelo Entidade-Associação

- O modelo “Entity-Relationship” é baseado na percepção de que o mundo real consiste numa colecção de objectos (“entities”) e de associações entre eles (“relationships”).

◂ Os diferentes Modelos de Dados

Modelo Entidade-Associação

- O modelo “Entity-Relationship” é baseado na percepção de que o mundo real consiste numa colecção de objectos (“entities”) e de associações entre eles (“relationships”).
  - Entidades (objectos):
    - `edifícios`;
    - `salas`;
    - `equipamento`.

◂ Os diferentes Modelos de Dados

Modelo Entidade-Associação

- O modelo “Entity-Relationship” é baseado na percepção de que o mundo real consiste numa colecção de objectos (“entities”) e de associações entre eles (“relationships”).
  - Entidades (objectos):
    - ★ `edifícios`;
    - ★ `salas`;
    - ★ `equipamento`.
  - Associações entre entidades:
    - ★ `pertence`, associa `equipamento` e `salas`; o equipamento `2920` pertence ao `gabinete 6.1`.

◂ Os diferentes Modelos de Dados

Modelo Entidade-Associação

- O modelo “Entity-Relationship” é baseado na percepção de que o mundo real consiste numa colecção de objectos (“entities”) e de associações entre eles (“relationships”).
  - Entidades (objectos):
    - ★ `edifícios`;
    - ★ `salas`;
    - ★ `equipamento`.
  - Associações entre entidades:
    - ★ `pertence`, associa `equipamento` e `salas`; o equipamento `2920` pertence ao `gabinete 6.1`.

- O modelo Entidade-Associação é muito usado na concepção de bases de dados.

◂ Os diferentes Modelos de Dados

Modelo Relacional

- O modelo relacional usa uma colecção de tabelas para representar tanto os dados como as associações entre eles. Cada tabela tem múltiplas colunas, sendo que cada coluna tem uma nome diferente.

◂ Os diferentes Modelos de Dados

Modelo Relacional

- O modelo relacional usa uma colecção de tabelas para representar tanto os dados como as associações entre eles. Cada tabela tem múltiplas colunas, sendo que cada coluna tem uma nome diferente.

- O modelo relacional é um exemplo de modelo baseado em registos.

◂ Os diferentes Modelos de Dados

Modelo Relacional

- O modelo relacional usa uma colecção de tabelas para representar tanto os dados como as associações entre eles. Cada tabela tem múltiplas colunas, sendo que cada coluna tem uma nome diferente.
- O modelo relacional é um exemplo de modelo baseado em registos.
- O modelo relacional é o mais usado na actualidade sendo que a vasta maioria dos actuais sistema implementa o modelo relacional.

◂ Os diferentes Modelos de Dados

Modelo baseado em Objectos

- O modelo baseado em objectos pretende extender o modelo E-A com a noção de encapsulamento de dados, métodos (funções) e identidade de objectos.

- O modelo relacional baseado em objectos combina as características do modelo de dados orientado aos objectos com o modelo de dados relacional.

◂ Os diferentes Modelos de Dados

Modelo de dados Semi-Estruturados

- O modelo de dados semi-estruturados permite a especificação de dados aonde items individuais de informação do mesmo tipo podem ter diferentes conjuntos de atributos (“variant records”).

- A linguagem XML (eXtensible Markup Language) é usualmente usada quando se pretende representar dados semi-estruturados.

◂ Os diferentes Modelos de Dados

Modelo Hierárquico

- No modelo hierárquico, a exemplo do modelo em rede, os dados são organizados em registos e as associações representadas por ponteiros.

- Ao contrário do modelo em rede os registos organizam-se segundo uma estrutura em árvore, em vez de um grafo generalizado.

◂ Os diferentes Modelos de Dados

Modelo em Rede (grafo generalizado)

- O modelo em rede difere do modelo relacional no facto de que os dados são representados como uma colecção de registos e de ponteiros entre eles.

- Um registo,neste modelo, é similar a uma entidade do modelo E-A, um ponteiro é uma associação entre dois registos. Isto é um ponteiro pode ser visto como uma forma (binária) restrita de associação no sentido do modelo E-A.

◂ Os diferentes Modelos de Dados

Modelo Baseado na Teoria dos Grafos

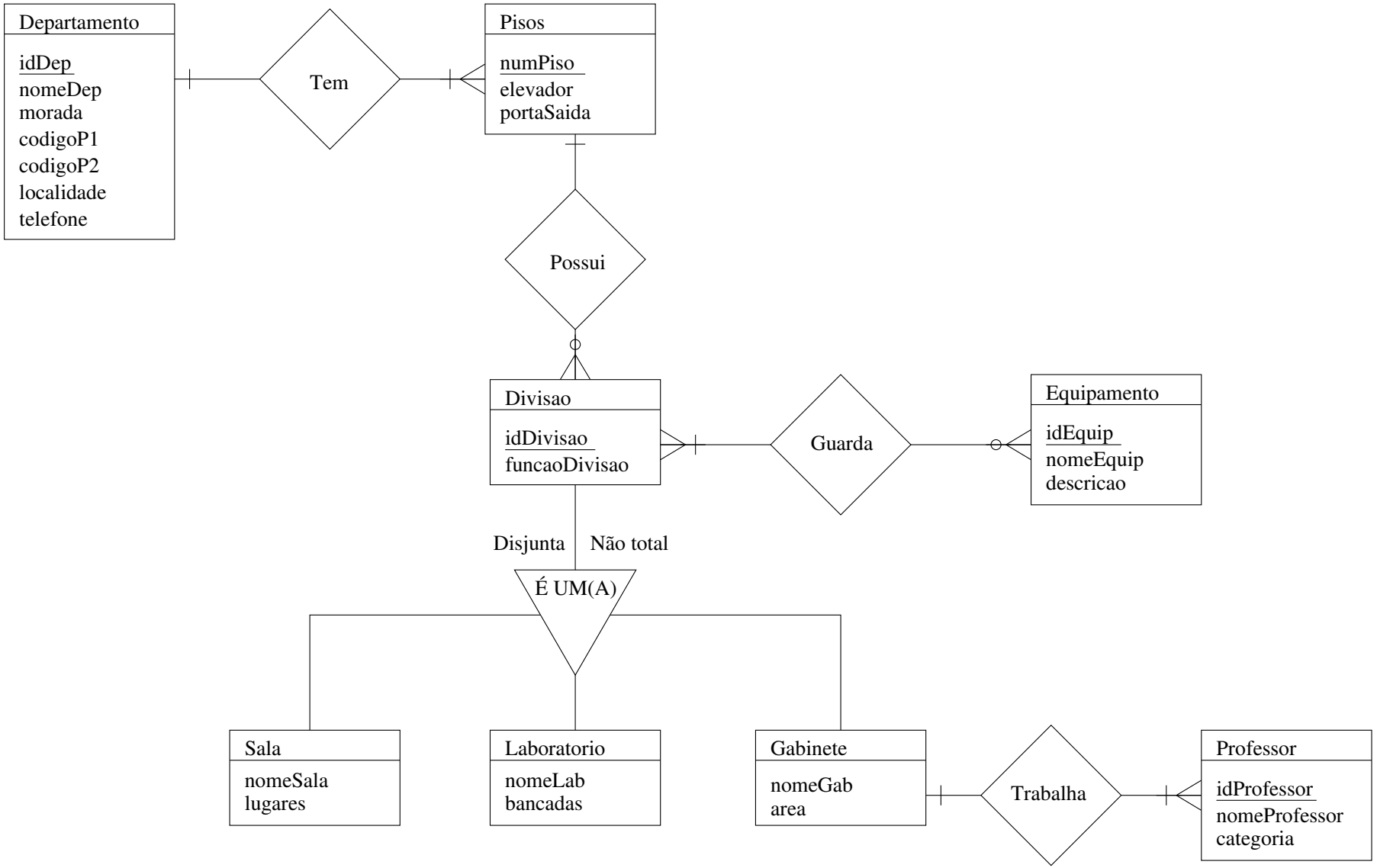
- As bases de dados orientadas a grafos (BDOG) representam a informaçã como nós de un grafo e as suas relações como arcos, de forma que se possa usar a teoria dos grafos para precorrer a base de datos.
- A informação é guardada como um grafo: os nós representam as entidades e os arcos representam as relações.

(e.g `Neo4j`)

◂ Os diferentes Modelos de Dados

Modelo E-A

Exemplo de um Diagrama Entidade-Associação (DEA)



Linguagem de Definição de Dados (DDL)

Especificação da notação para definição do esquema da base de dados.
Por exemplo:

```
CREATE TABLE Equipamento (
    IdEquip     integer,
    NomeEquip   char(10),
    Descricao   text)
```

Linguagem de Definição de Dados (DDL)

Especificação da notação para definição do esquema da base de dados.
Por exemplo:

```
CREATE TABLE Equipamento (
    IdEquip     integer,
    NomeEquip   char(10),
    Descricao   text)
```

O compilador da DDL gera um conjunto de tabelas armazenadas num dicionário de dados.

Linguagem de Definição de Dados (DDL)

Especificação da notação para definição do esquema da base de dados.
Por exemplo:

```
CREATE TABLE Equipamento (
    IdEquip     integer,
    NomeEquip   char(10),
    Descricao   text)
```

O compilador da DDL gera um conjunto de tabelas armazenadas num dicionário de dados.
O dicionário de dados contém meta-dados (dados sobre os dados):

Linguagem de Definição de Dados (DDL)

Especificação da notação para definição do esquema da base de dados.
Por exemplo:

```
CREATE TABLE Equipamento (
    IdEquip     integer,
    NomeEquip   char(10),
    Descricao   text)
```

O compilador da DDL gera um conjunto de tabelas armazenadas num dicionário de dados.
O dicionário de dados contém meta-dados (dados sobre os dados):

- Esquema de bases de dados;

Linguagem de Definição de Dados (DDL)

Especificação da notação para definição do esquema da base de dados.
Por exemplo:

```
CREATE TABLE Equipamento (
    IdEquip     integer,
    NomeEquip   char(10),
    Descricao   text)
```

O compilador da DDL gera um conjunto de tabelas armazenadas num dicionário de dados.
O dicionário de dados contém meta-dados (dados sobre os dados):

- Esquema de bases de dados;
- Linguagem de definição de dados e armazenamento:
  - Linguagem onde se especificam as estruturas de armazenamento e métodos de acesso utilizados pela base de dados;
  - Normalmente uma extensão da linguagem da definição de dados.

Linguagem de Manipulação de Dados (DML)

Linguagem para aceder e manipular os dados organizados de acordo com o respectivo modelo de dados.
A DML também é conhecida por linguagem de consulta

Linguagem de Manipulação de Dados (DML)

Linguagem para aceder e manipular os dados organizados de acordo com o respectivo modelo de dados.
A DML também é conhecida por linguagem de consulta

Duas classes de linguagens:

Procedimental o utilizador especifica quais os dados que se pretendem assim como obter os dados;

Declarativa o utilizador especifica quais os dados pretendidos sem especificar a maneira como os obter

Linguagem de Manipulação de Dados (DML)

Linguagem para aceder e manipular os dados organizados de acordo com o respectivo modelo de dados.
A DML também é conhecida por linguagem de consulta

Duas classes de linguagens:

Procedimental o utilizador especifica quais os dados que se pretendem assim como obter os dados;

Declarativa o utilizador especifica quais os dados pretendidos sem especificar a maneira como os obter

A SQL (declarativa, não procedimental) é a linguagem de consulta mais utilizada.

SQL

Linguagem não-procedimental de uso generalizado. Por exemplo: encontrar o nome do equipamento com identificação (n. de inventário) “2920”

```
SELECT NomeEquip
  FROM Equipamento
 WHERE IdEquip = 2920
```

Outro exemplo: procurar as divisões do 3º piso.

```
SELECT IdDivisao
  FROM Divisao, Piso
 WHERE NumPiso = 3
```

As aplicações geralmente acedem a bases de dados por intermédio de:

- Extensões às linguagens permitindo SQL embutido:
- Interface de aplicações (e.g. ODBC/JDBC) permitindo o envio de consultas SQL para a base de dados

Utilizadores da Base de Dados

Os utilizadores diferenciam-se pela forma esperada de interacção com o sistema:

Programadores de aplicações interagem com o sistema através de chamadas DML.

Utilizadores sofisticados constroem pedidos numa linguagem de consulta a bases de dados.

Utilizadores especializados escrevem aplicações de bases de dados especializadas que não se enquadram com o espírito do processamento de dados tradicional.

Utilizadores chamam uma das aplicações que foi construída previamente.
Por exemplo, pessoas a acederem a uma base de dados através da Rede, caixas, pessoal de secretariado.

Administrador da Base de Dados

Coordena todas as actividades do sistema de base de dados;
As funções do administrador de bases de dados incluem:

Administrador da Base de Dados

Coordena todas as actividades do sistema de base de dados;
As funções do administrador de bases de dados incluem:

- Definição do esquema;

Administrador da Base de Dados

Coordena todas as actividades do sistema de base de dados;
As funções do administrador de bases de dados incluem:

- Definição do esquema;
- Definição dos métodos de acesso e estrutura de armazenamento;

Administrador da Base de Dados

Coordena todas as actividades do sistema de base de dados;
As funções do administrador de bases de dados incluem:

- Definição do esquema;
- Definição dos métodos de acesso e estrutura de armazenamento;
- Modificação do esquema e da organização física;

Administrador da Base de Dados

Coordena todas as actividades do sistema de base de dados;
As funções do administrador de bases de dados incluem:

- Definição do esquema;
- Definição dos métodos de acesso e estrutura de armazenamento;
- Modificação do esquema e da organização física;
- **Dar aos utilizadores autorizações de acesso à base de dados**;

Administrador da Base de Dados

Coordena todas as actividades do sistema de base de dados;
As funções do administrador de bases de dados incluem:

- Definição do esquema;
- Definição dos métodos de acesso e estrutura de armazenamento;
- Modificação do esquema e da organização física;
- **Dar aos utilizadores autorizações de acesso à base de dados**;
- Especificar restrições de integridade;

Administrador da Base de Dados

Coordena todas as actividades do sistema de base de dados;
As funções do administrador de bases de dados incluem:

- Definição do esquema;
- Definição dos métodos de acesso e estrutura de armazenamento;
- Modificação do esquema e da organização física;
- **Dar aos utilizadores autorizações de acesso à base de dados**;
- Especificar restrições de integridade;
- Servir de ligação entre os utilizadores;

Administrador da Base de Dados

Coordena todas as actividades do sistema de base de dados;
As funções do administrador de bases de dados incluem:

- Definição do esquema;
- Definição dos métodos de acesso e estrutura de armazenamento;
- Modificação do esquema e da organização física;
- **Dar aos utilizadores autorizações de acesso à base de dados**;
- Especificar restrições de integridade;
- Servir de ligação entre os utilizadores;
- Monitorar a performace e responder a alterações nos requisitos.

Gestão de Transacções

Uma transacção é um conjunto de operações que efectuam uma função lógica na aplicação de base de dados

A componente de gestão de transacções garante que a base de dados se mantém num estado consistente (correcto) apesar de falhas no sistema (por exemplo: falta de energia eléctrica e paragens abruptas do sistema operativo) e de transacções falhadas.

O gestor de controlo de concorrência coordena a interacção entre transacções concorrentes para garantir a consistência da base de dados.

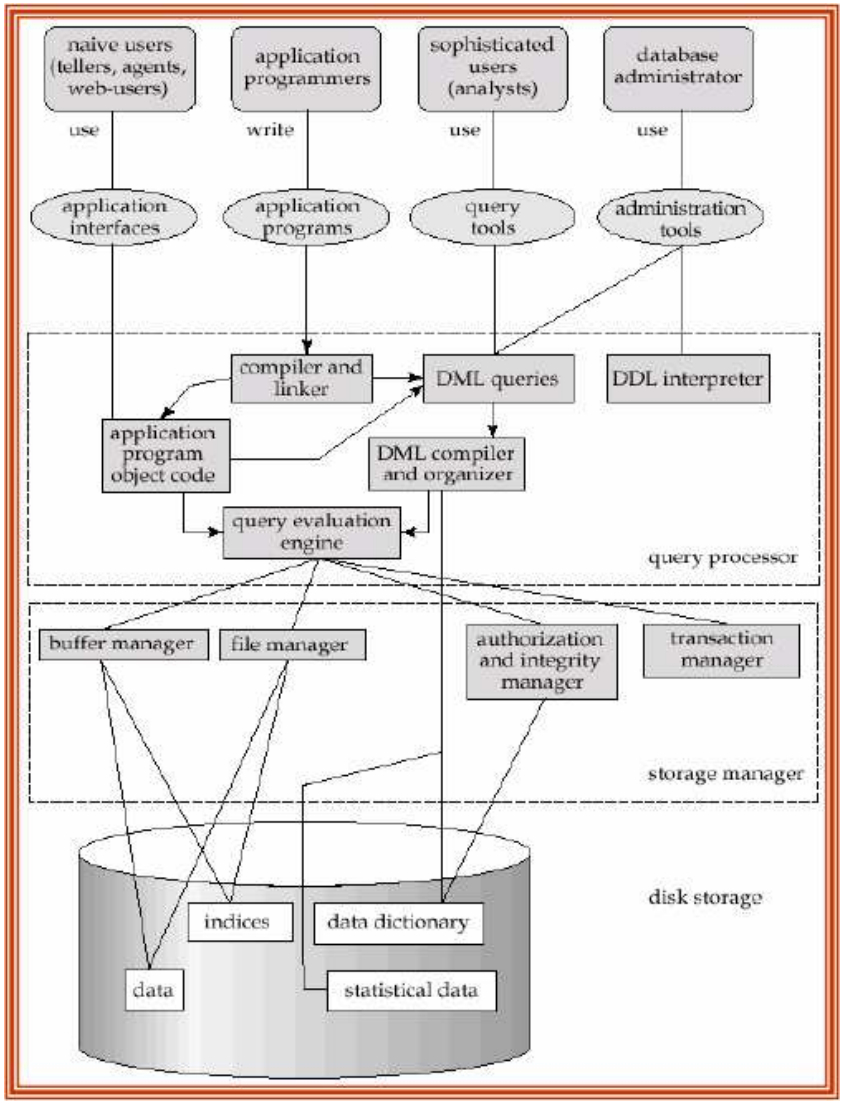
Gestão do Armazenamento

O gestor do armazenamento é um módulo de programa que fornece uma interface entre os dados de baixo nível armazenados na base de dados e as aplicações e consultas submetidas ao sistema.

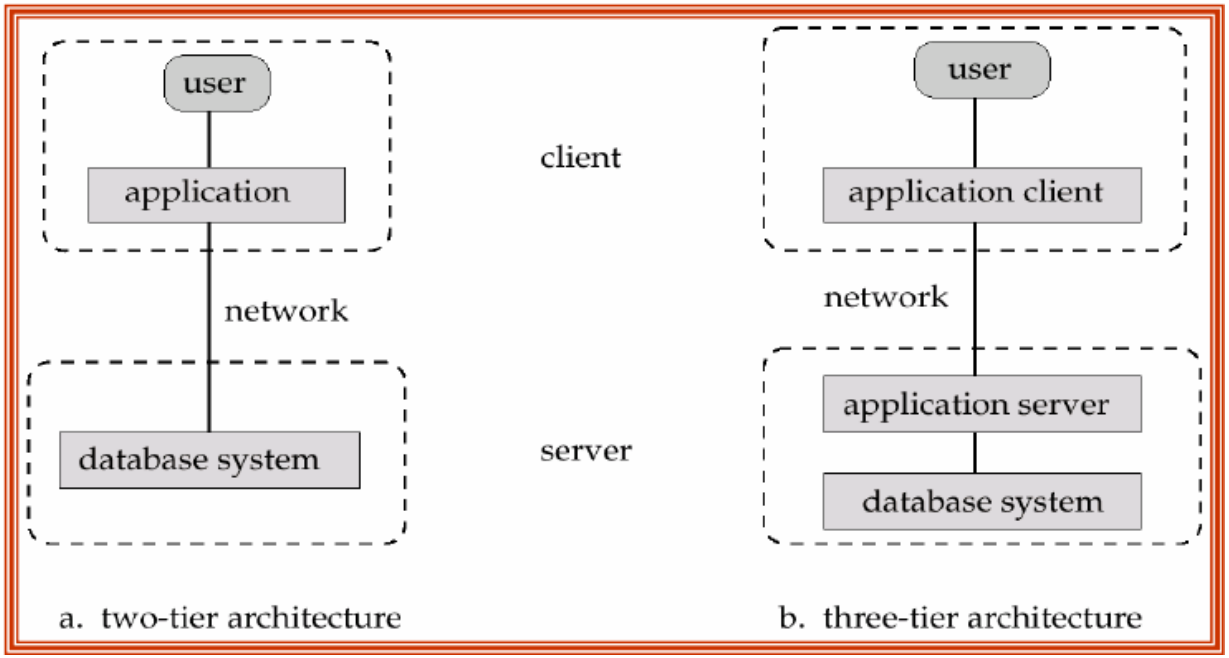
O gestor de armazenamento é responsável pelas seguintes tarefas:

- interacção com o gestor de ficheiros;
- armazenamento, recuperação e alteração eficientes dos dados.

Estrutura Genérica do Sistema



Arquitecturas de Aplicação



Arquitectura de duas camadas: os programas clientes recorrem a, por exemplo, ODBC/JDBC para comunicar com a base de dados.

Arquitectura de três camadas: por exemplo, aplicações da Rede e aplicações construídas recorrendo a "software intermediário".

Modelo Entidade-Associação

- Entidades
- Associações
- Restrições
- Chaves

Entidades e Associações

Uma base de dados pode ser modelada como:

- uma colecção de Entidades: uma entidade é um objecto existente e que é distinguível de todos os outros objectos.
  Exemplos: `Pessoa`; `Edifício`; `Sala`; `Equipamento`.

Entidades e Associações

Uma base de dados pode ser modelada como:

- uma colecção de Entidades: uma entidade é um objecto existente e que é distinguível de todos os outros objectos.
  Exemplos: `Pessoa`; `Edifício`; `Sala`; `Equipamento`.

- uma colecção de Associações entre entidades: uma associação é uma associação entre várias entidades.
  Exemplos: `contaAPrazo`, entre um banco e os seus clientes; `pertence`, entre um edíficio e as suas salas.

Entidades e Atributos

- Uma entidades pode ser vista como a abstracção de um conjunto de objectos do mesmo tipo e que partilham as mesmas propriedades. Exemplo: o conjunto de todas as `pessoas`, `empresas`, `árvores`, `feriados`.

```
Funcionario = (numero,nome,docente,correioE,bi)
Cliente = (id, nome, endereco, cidade)
Emprestimo = (numeroEmprestimo, quantia)
```

Entidades e Atributos

- Uma entidades pode ser vista como a abstracção de um conjunto de objectos do mesmo tipo e que partilham as mesmas propriedades. Exemplo: o conjunto de todas as `pessoas`, `empresas`, `árvores`, `feriados`.

- As entidades possuem atributos: propriedades descritivas possuídas por todos os membros de uma entidade. Propriedades que permitem definir a entidade.

```
Funcionario = (numero,nome,docente,correioE,bi)
Cliente = (id, nome, endereco, cidade)
Emprestimo = (numeroEmprestimo, quantia)
```

Atributos — Domínios e Tipos

Domínio - o conjunto de valores permitidos para cada atributo. Por exemplo: o domínio de `nomeCliente` pode ser o conjunto de todas as sequências de caracteres, com no máximo 50 caracteres.

Atributos — Domínios e Tipos

Domínio - o conjunto de valores permitidos para cada atributo. Por exemplo: o domínio de `nomeCliente` pode ser o conjunto de todas as sequências de caracteres, com no máximo 50 caracteres.
Tipos de atributos:

Atributos — Domínios e Tipos

Domínio - o conjunto de valores permitidos para cada atributo. Por exemplo: o domínio de `nomeCliente` pode ser o conjunto de todas as sequências de caracteres, com no máximo 50 caracteres.
Tipos de atributos:

- Atributos simples e compostos.
  Por exemplo: `sexo`, `nome`.

Atributos — Domínios e Tipos

Domínio - o conjunto de valores permitidos para cada atributo. Por exemplo: o domínio de `nomeCliente` pode ser o conjunto de todas as sequências de caracteres, com no máximo 50 caracteres.
Tipos de atributos:

- Atributos simples e compostos.
  Por exemplo: `sexo`, `nome`.
- Atributos univalor e multivalor
  Por exemplo: `altura`, `números de telefone`.

Atributos — Domínios e Tipos

Domínio - o conjunto de valores permitidos para cada atributo. Por exemplo: o domínio de `nomeCliente` pode ser o conjunto de todas as sequências de caracteres, com no máximo 50 caracteres.
Tipos de atributos:

- Atributos simples e compostos.
  Por exemplo: `sexo`, `nome`.
- Atributos univalor e multivalor
  Por exemplo: `altura`, `números de telefone`.
- Atributos derivados. Podem ser calculados a partir de outros atributos.
  Por exemplo: `idade`, calculado a partir da data de nascimento.

Chaves

- Uma Super-chave de um conjunto de entidades é um conjunto de um ou mais atributos cujos valores **determinam univocamente** cada entidade.
  A determinação unívoca, depende do contexto em causa, e é imposta como restrição.

Chaves

- Uma Super-chave de um conjunto de entidades é um conjunto de um ou mais atributos cujos valores **determinam univocamente** cada entidade.
  A determinação unívoca, depende do contexto em causa, e é imposta como restrição.
- Uma chave candidata de um conjunto de entidades é uma **super-chave minimal**.

Chaves

- Uma Super-chave de um conjunto de entidades é um conjunto de um ou mais atributos cujos valores **determinam univocamente** cada entidade.
  A determinação unívoca, depende do contexto em causa, e é imposta como restrição.
- Uma chave candidata de um conjunto de entidades é uma **super-chave minimal**.
- Apesar de poderem existir várias chaves candidatas, uma das chaves candidatas é seleccionada para ser a chave primária.

Chaves

- Uma Super-chave de um conjunto de entidades é um conjunto de um ou mais atributos cujos valores **determinam univocamente** cada entidade.
  A determinação unívoca, depende do contexto em causa, e é imposta como restrição.
- Uma chave candidata de um conjunto de entidades é uma **super-chave minimal**.
- Apesar de poderem existir várias chaves candidatas, uma das chaves candidatas é seleccionada para ser a chave primária.

Por exemplo: para a entidade `cliente = (id, nome, endereco, cidade)` temos:

Chaves

- Uma Super-chave de um conjunto de entidades é um conjunto de um ou mais atributos cujos valores **determinam univocamente** cada entidade.
  A determinação unívoca, depende do contexto em causa, e é imposta como restrição.
- Uma chave candidata de um conjunto de entidades é uma **super-chave minimal**.
- Apesar de poderem existir várias chaves candidatas, uma das chaves candidatas é seleccionada para ser a chave primária.

Por exemplo: para a entidade `cliente = (id, nome, endereco, cidade)` temos:

- Super-chave: qualquer conjunto de atributos que contenham o atributo `id`;

Chaves

- Uma Super-chave de um conjunto de entidades é um conjunto de um ou mais atributos cujos valores **determinam univocamente** cada entidade.
  A determinação unívoca, depende do contexto em causa, e é imposta como restrição.
- Uma chave candidata de um conjunto de entidades é uma **super-chave minimal**.
- Apesar de poderem existir várias chaves candidatas, uma das chaves candidatas é seleccionada para ser a chave primária.

Por exemplo: para a entidade `cliente = (id, nome, endereco, cidade)` temos:

- Super-chave: qualquer conjunto de atributos que contenham o atributo `id`;
- Chave candidata: `id`;

Chaves

- Uma Super-chave de um conjunto de entidades é um conjunto de um ou mais atributos cujos valores **determinam univocamente** cada entidade.
  A determinação unívoca, depende do contexto em causa, e é imposta como restrição.
- Uma chave candidata de um conjunto de entidades é uma **super-chave minimal**.
- Apesar de poderem existir várias chaves candidatas, uma das chaves candidatas é seleccionada para ser a chave primária.

Por exemplo: para a entidade `cliente = (id, nome, endereco, cidade)` temos:

- Super-chave: qualquer conjunto de atributos que contenham o atributo `id`;
- Chave candidata: `id`;
- Chave primária: `id`;

Representação de Entidades

Existem várias formas de representar as entidades, seus atributos, e a sua chave primária.

Representação de Entidades

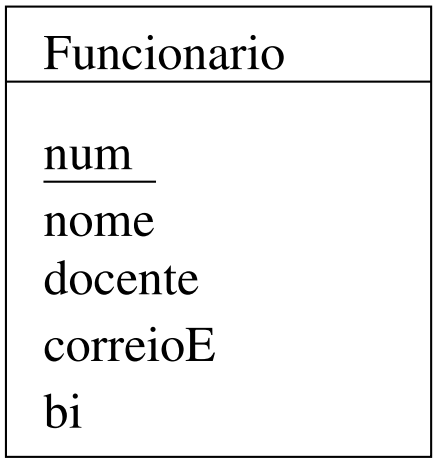
Existem várias formas de representar as entidades, seus atributos, e a sua chave primária.

- Textual: uma entidade é representada como um tuplo, sendo a chave primária individualizada de alguma forma.
  Por exemplo:
  `Funcionario = (num, nome, docente, correioE, bi)`

Representação de Entidades

Existem várias formas de representar as entidades, seus atributos, e a sua chave primária.

- Textual: uma entidade é representada como um tuplo, sendo a chave primária individualizada de alguma forma.
  Por exemplo:
  `Funcionario = (num, nome, docente, correioE, bi)`
- Gráfica: uma "caixa" contendo o nome da entidade e dos seus atributos, a chave primária tem de ser individualizada de alguma forma.



Associações

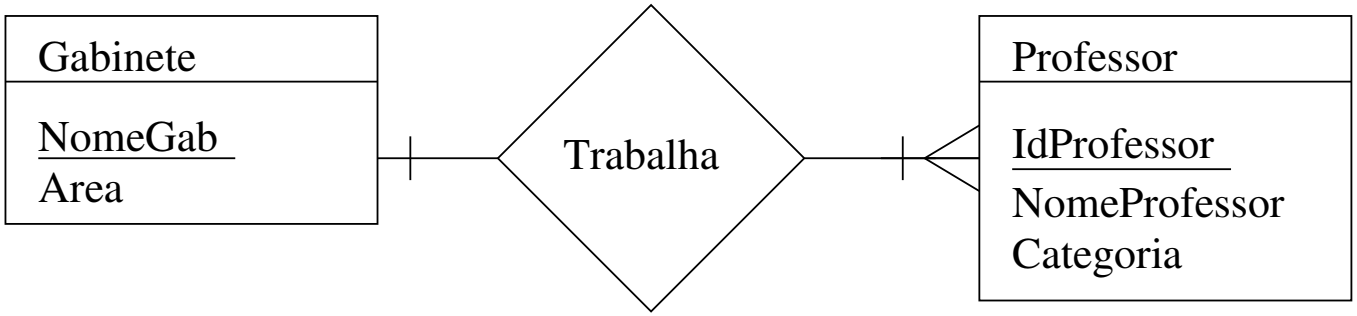
Um conjunto de associações é uma associação matemática entre $n \geq 2$ entidades, cada uma pertencente a um conj. de entidades

$$\{(e_1, e_2, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

em que $(e_1, e_2, \ldots, e_n)$ é uma associação.

Exemplos:

- `(Hugo, A-102)` $\in$ `depositante`
- `(Pedro Quaresma, 5.5)` $\in$ `Trabalha`

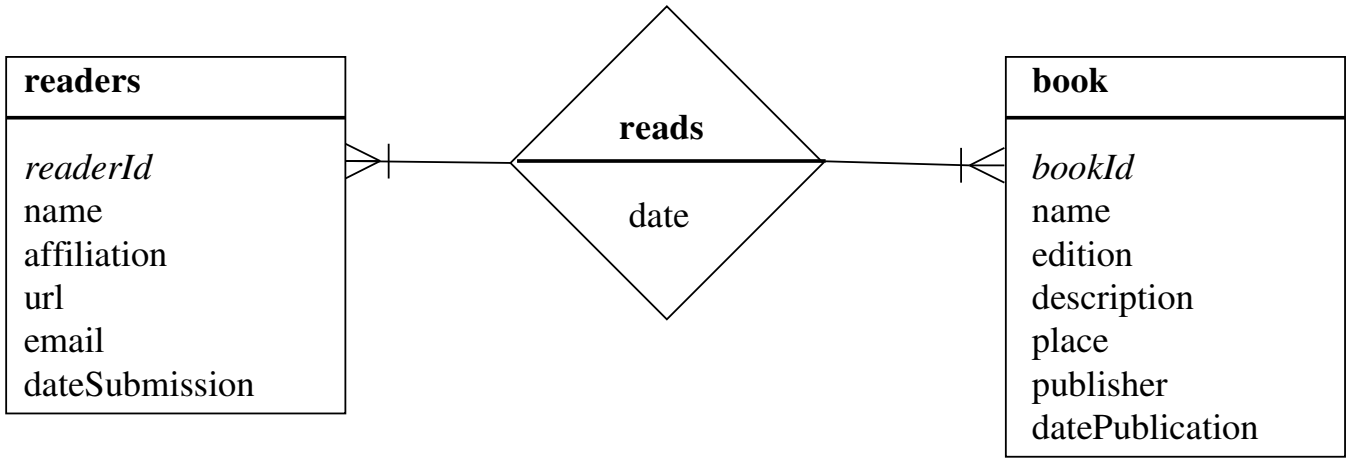


Associações (Cont.)

Um atributo pode ser uma propriedade de um conjunto de associações.

Por exemplo, o conj, de associações depositante entre os conjs. de entidades `cliente` e `conta` pode ter um atributo `dataAcesso`.

Outro exemplo:



Cardinalidades — Restrições às Associações

Restringem o número de entidades com as quais pode estar associada uma outra entidade num determinado conjunto de associações.
Para um conjunto de associações binárias a cardinalidade pode ser uma das seguintes:

Cardinalidades — Restrições às Associações

Restringem o número de entidades com as quais pode estar associada uma outra entidade num determinado conjunto de associações.
Para um conjunto de associações binárias a cardinalidade pode ser uma das seguintes:

- um para um (ou 1:1)

Cardinalidades — Restrições às Associações

Restringem o número de entidades com as quais pode estar associada uma outra entidade num determinado conjunto de associações.
Para um conjunto de associações binárias a cardinalidade pode ser uma das seguintes:

- um para um (ou 1:1)
- um para muitos (ou um para vários, ou 1:N)
- muitos para um (ou vários para 1, ou N:1)

Cardinalidades — Restrições às Associações

Restringem o número de entidades com as quais pode estar associada uma outra entidade num determinado conjunto de associações.
Para um conjunto de associações binárias a cardinalidade pode ser uma das seguintes:

- um para um (ou 1:1)
- um para muitos (ou um para vários, ou 1:N)
- muitos para um (ou vários para 1, ou N:1)

Cardinalidades — Restrições às Associações

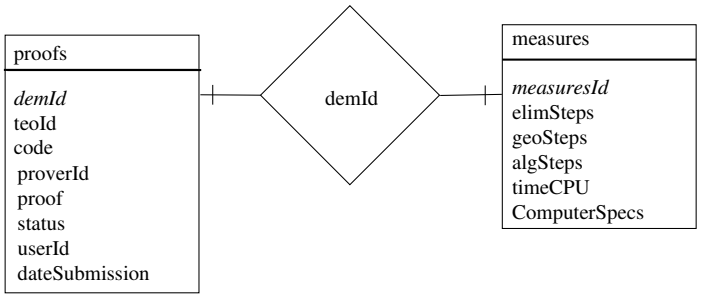
Restringem o número de entidades com as quais pode estar associada uma outra entidade num determinado conjunto de associações.
Para um conjunto de associações binárias a cardinalidade pode ser uma das seguintes:

- um para um (ou 1:1)
- um para muitos (ou um para vários, ou 1:N)
- muitos para um (ou vários para 1, ou N:1)
- muitos para muitos (ou vários para vários, N:M)

Restrição $A \xrightarrow{1:1} B$

- Proibe que uma entidade de A se relacione com mais do que uma entidade de B.
- Proibe que uma entidade de B se relacione com mais do que uma entidade de A.

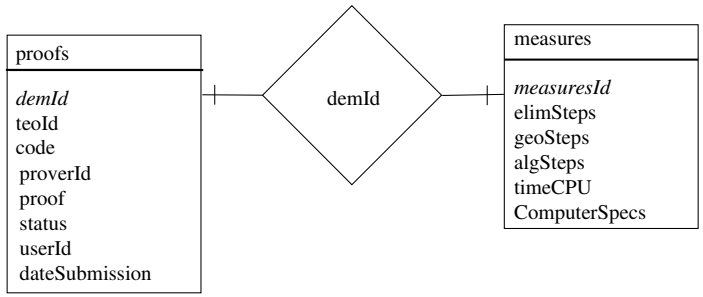
Exemplos:



Restrição $A \xrightarrow{1:1} B$

- Proibe que uma entidade de A se relacione com mais do que uma entidade de B.
- Proibe que uma entidade de B se relacione com mais do que uma entidade de A.

Exemplos:



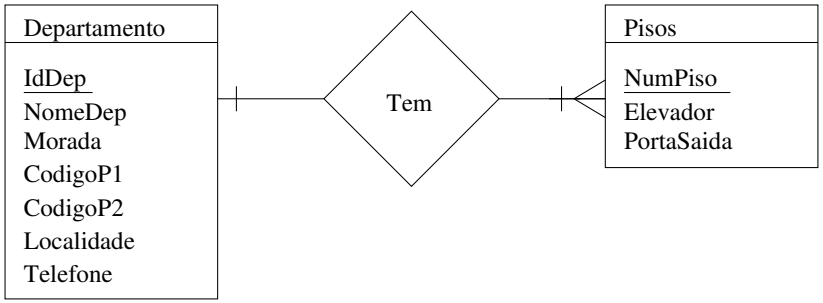
Numa empresa um `empregado` tem no máximo um `carro`, e um `carro` é atribuído a no máximo um `empregado`.

Nota: Alguns elementos de A ou B podem não estar relacionados com elementos do outro conjunto.

Restrição $A \xrightarrow{N:1} B$

- Proibe que uma entidade de A se relacione com mais do que uma entidade de B.
- Permite que uma entidade de B se relacione com mais do que uma entidade de A.

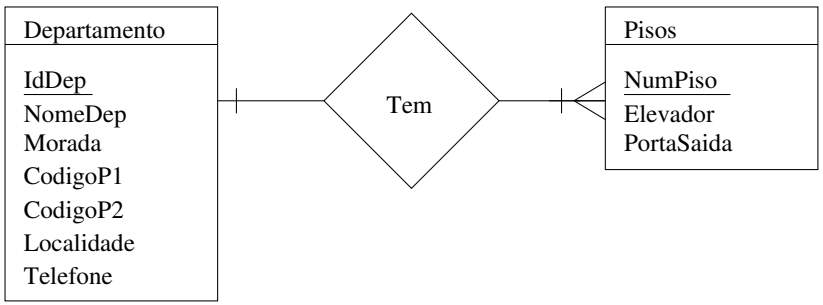
Exemplos:



Restrição $A \xrightarrow{N:1} B$

- Proibe que uma entidade de A se relacione com mais do que uma entidade de B.
- Permite que uma entidade de B se relacione com mais do que uma entidade de A.

Exemplos:

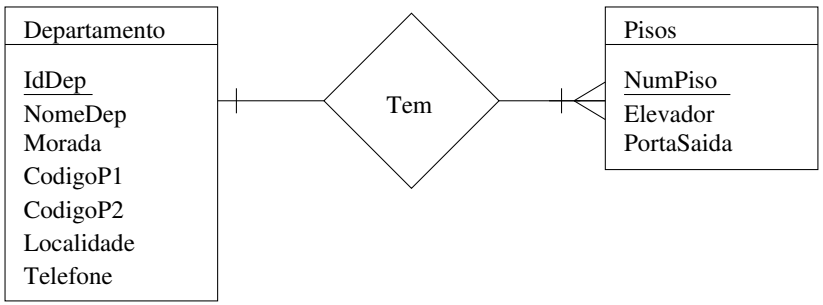


Um `aluno` está associado a no máximo uma `turma`, mas uma `turma` pode estar associada a mais que um `aluno`.

Restrição $A \xrightarrow{N:1} B$

- Proibe que uma entidade de A se relacione com mais do que uma entidade de B.
- Permite que uma entidade de B se relacione com mais do que uma entidade de A.

Exemplos:

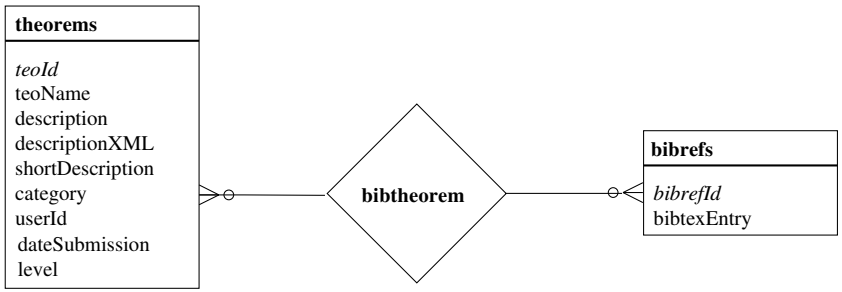


Um `aluno` está associado a no máximo uma `turma`, mas uma `turma` pode estar associada a mais que um `aluno`.
Nota: Alguns elementos de A ou B podem não estar relacionados com elementos do outro conjunto.

Restrição $A \xrightarrow{N:M} B$

- Não impõe restrições
- Permite que uma entidade de A se relacione com mais do que uma entidade de B.
- Permite que uma entidade de B se relacione com mais do que uma entidade de A.

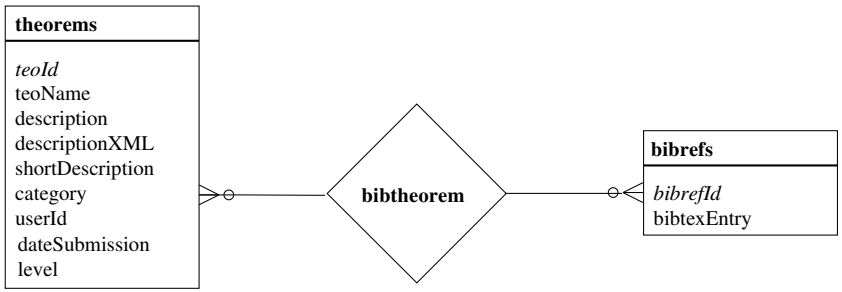
Exemplos:



Restrição $A \xrightarrow{N:M} B$

- Não impõe restrições
- Permite que uma entidade de A se relacione com mais do que uma entidade de B.
- Permite que uma entidade de B se relacione com mais do que uma entidade de A.

Exemplos:

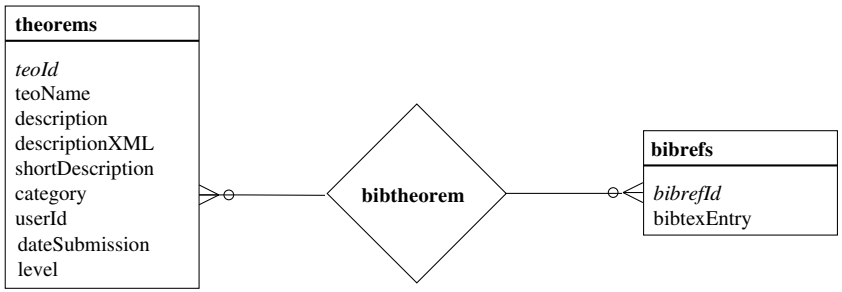


Uma `conta` pode estar associada a mais do que um `cliente`, e um `cliente` pode ter mais do que uma `conta` associada

Restrição $A \xrightarrow{N:M} B$

- Não impõe restrições
- Permite que uma entidade de A se relacione com mais do que uma entidade de B.
- Permite que uma entidade de B se relacione com mais do que uma entidade de A.

Exemplos:



Uma `conta` pode estar associada a mais do que um `cliente`, e um `cliente` pode ter mais do que uma `conta` associada

Nota: Alguns elementos de A ou B podem não estar relacionados com elementos do outro conjunto.

As cardinalidades afectam a concepção

Relembrando o exemplo:

> *o conjunto de associações* `depositante` *entre os conjuntos de entidades* `cliente` *e* `conta` *pode ter um atributo* `dataAcesso`.

O atributo da associação `dataAcesso` pode ser passar a ser um atributo de `conta`, isto se cada `conta` tiver apenas um `cliente`. Cardinalidade de $1 : N$.

No entanto, se a cardinalidade da associação for de $N : M$ tal não é possível. Nesse caso o atributo `dataAcesso` tem de pertencer à associação.

Diagramas Entidade-Associação

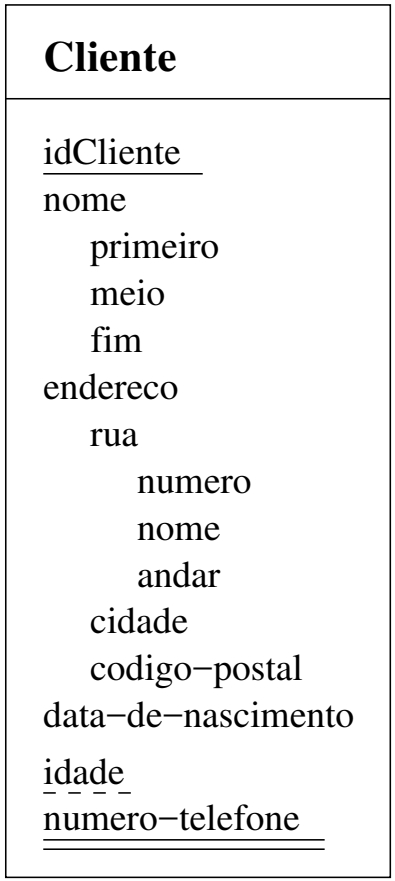
Diagramas Entidade-Associação (DEA/ERD)

Diagramas Patas de Corvos (com algumas adaptações próprias).

- **Rectângulos** representam entidades;
- **Losangos** representam associações;
- **Linhas** unem entidades e associações. Diferentes **terminações** especificam diferentes cardinalidades.

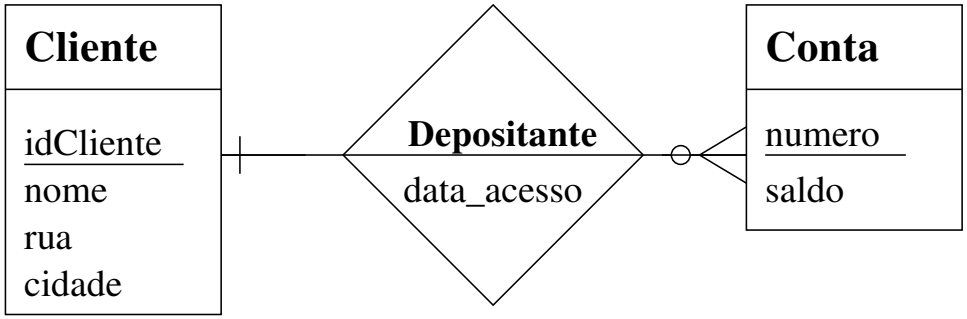
- Cada entidade e/ou associação é identificada por um identificador (acima da **linha horizontal**).
- Os atributos aparecem abaixo das linhas horizontais.
- As chaves primárias (atributos) são identificados por um **sublinhado simples**.

DEA com atributos compostos, multi-valor e derivados

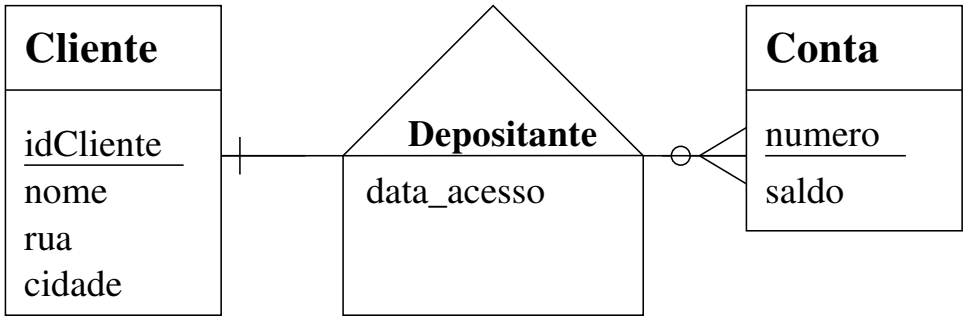


- atributos compostos são representados pela **indentação**;
- atributos derivados são representados por **tracejado**;
- atributos multi-valor são representados por um **duplo sublinhado**.

DEA — Associações com atributos

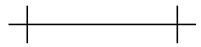
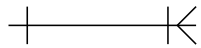
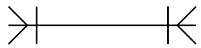
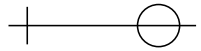


ou



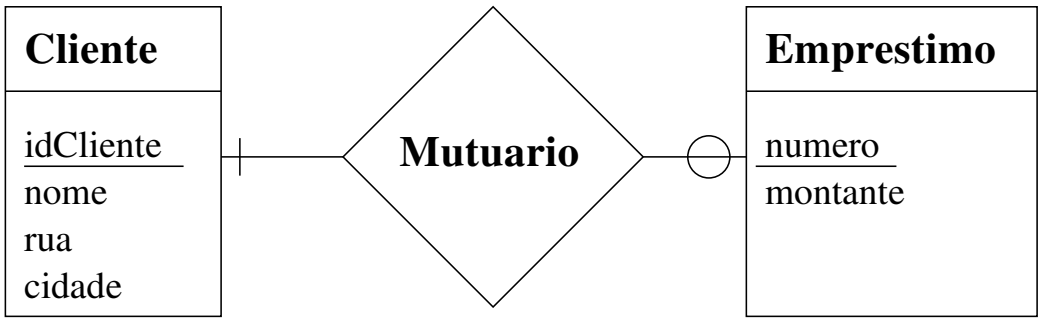
DEA - Cardinalidade das Associações

A cardinalidade das associações é expressa da seguinte forma:

1:1	
1:N	
N:M	
associações não totais	

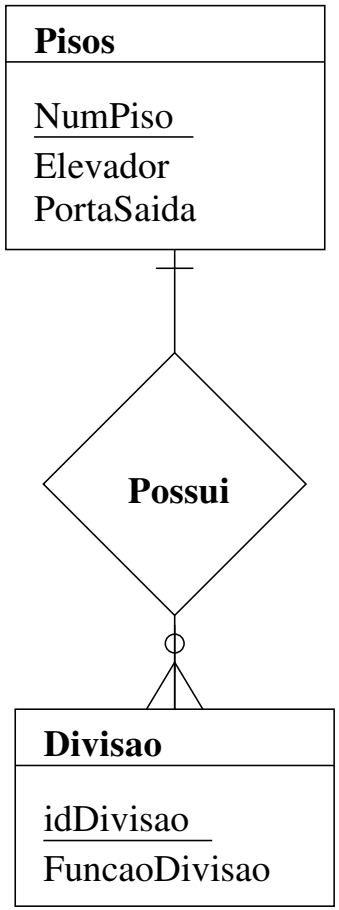
As associações não totais podem acontecer em qualquer um dos casos. Por exemplo: 1 : 1 não total num dos lados.

- um cliente está associado a, no máximo, um empréstimo.
- um empréstimo está associado a um cliente.



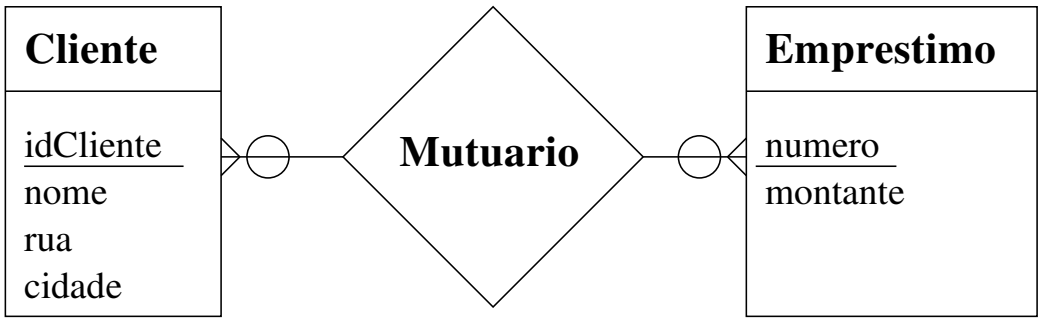
DEA - associação 1:N, não total

Pode-se ter o caso em que um piso está associado a várias divisões no entanto pode-se dar o caso de não ter nenhuma (um piso de entrada).



DEA - associação N:M, não total

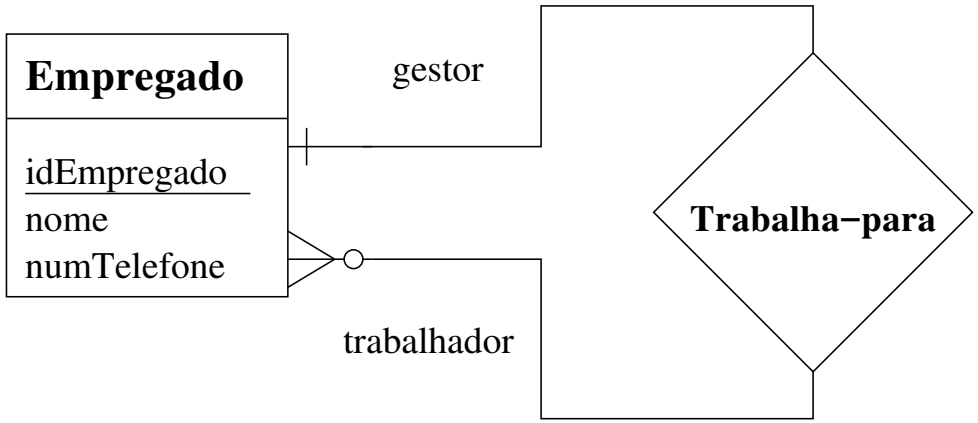
- um cliente está associado com vários empréstimos, possivelmente zero.
- um empréstimo está associado com vários clientes, possivelmente zero.



Modelo E-A/DEA — Adicionais

Papéis

- Os conjuntos de entidades participantes numa associação não são obrigatoriamente distintos:
- As etiquetas “gestor” e “trabalhador” são designadas papéis; especificam como as entidades “empregado” interagem por intermédio de associações “trabalha-para”.
- Os papéis são indicadas nos DEAs anotando as linhas que ligam os losangos aos rectângulos.
- Os papéis são opcionais, sendo utilizados para clarificar a semântica da associação.



Conjunto de Entidades Fracas

Um conjunto de entidades pode não ter atributos para formar uma chave primária. Nesse caso é designado por conjunto de entidades fracas.

Exemplo: Movimentos de conta, com nº de movimento data/hora e valor. Pode haver dois movimentos com o mesmo nº, do mesmo valor e a mesma data/hora. Têm é que ser de contas diferentes.

A existência de um conjunto de entidades fracas depende da existência de um conjunto de entidades dominante.

- o conjunto de entidades identificador deve relacionar-se com o conjunto de entidades fracas através de uma associação um para muitos, total do lado do conjunto de entidades identificador.
- Exemplo: Conta é conjunto de entidades dominante de Movimentos.

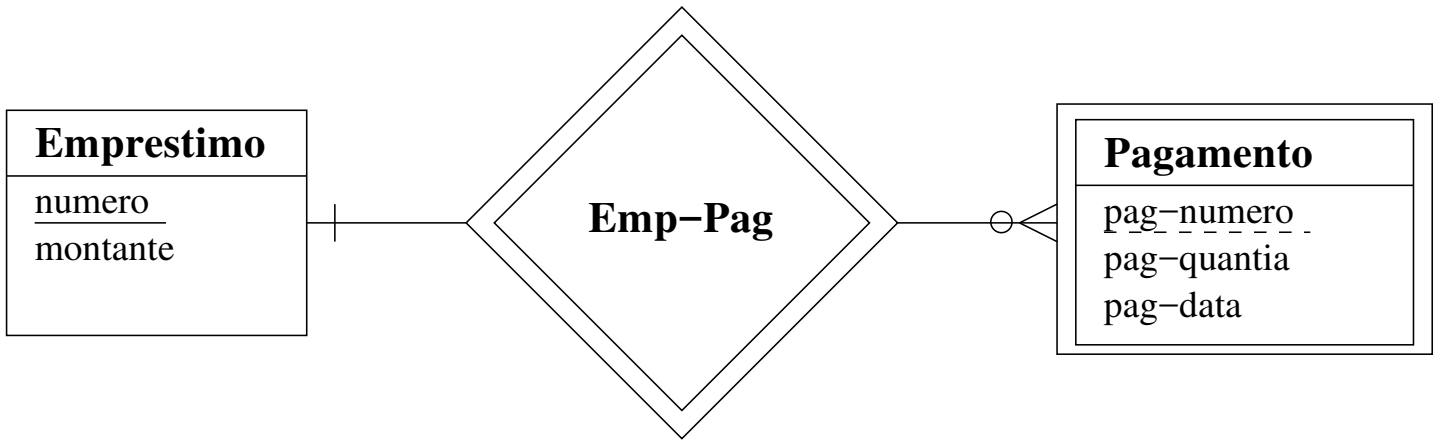
Conjunto de Entidades Fracas (cont.)

- O **discriminante** (ou chave parcial) é o conjunto de atributos que distingue as entidades de um conjunto de entidades fracas, associadas a uma mesma entidade do conjunto dominante. Exemplo: Nº de movimento é discriminante pois, para uma mesma conta, não pode haver dois movimentos com o mesmo nº.

- A chave primária de um conjunto de entidades fracas é constituída pela chave primária do conjunto de entidades dominante do qual depende e pelo discriminante do conjunto de entidades fracas.

Conjunto de Entidades Fracas (Cont.)

- Um conjunto de entidades fracas é representado por um rectângulo duplo.
- O discriminante do conjunto de entidades fracas é sublinhado a tracejado.
- A associação entre o conjunto entidades fracas e o dominante é representada por um losango duplo



Conjunto de Entidades Fracas (Cont.)

- Nota: a chave primária do conjunto de entidades identificador (ou forte) não é explicitamente representado no conjunto de entidades fracas, dado ser implícito na associação identificadora.

- Se `numero` (de empréstimo) fosse representado explicitamente, `Pagamento` poderia ser um conjunto de entidades fortes, mas assim a associação entre `Pagamento` e `Emprestimo` seria duplicada por uma associação implícita definida pelo atributo `numero` comum às duas entidades.

Especialização/Generalização

- Método de concepção descendente: designamos subgrupos dentro de um conjunto de entidades que são distintas de outras entidades nesse conjunto (Especialização).

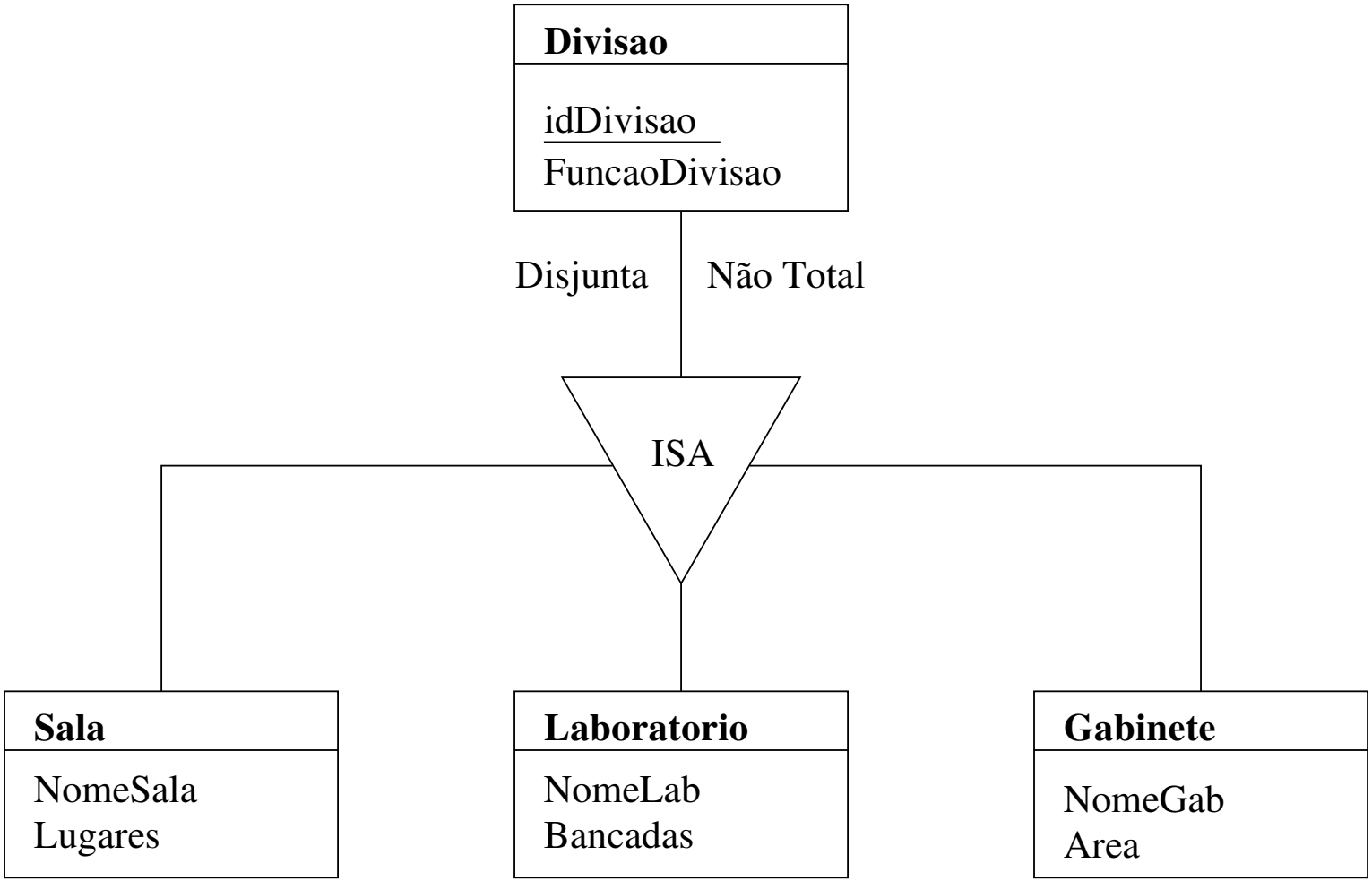
Especialização/Generalização

- Método de concepção descendente: designamos subgrupos dentro de um conjunto de entidades que são distintas de outras entidades nesse conjunto (Especialização).
- Método de concepção ascendente: combinar num conjunto de entidades de maior nível um certo número de conjuntos de entidades que partilham as mesmas características.

Especialização/Generalização

- Método de concepção descendente: designamos subgrupos dentro de um conjunto de entidades que são distintas de outras entidades nesse conjunto (Especialização).
- Método de concepção ascendente: combinar num conjunto de entidades de maior nível um certo número de conjuntos de entidades que partilham as mesmas características.
- Estes subgrupos tornam-se conjuntos de entidades de menor nível que têm atributos ou participam em associações que não se aplicam ao conjunto de entidades de maior nível. Desenhado por um triângulo anotado com ISA: um cliente é uma ("is a") pessoa.
- Herança de atributos: um conjunto de entidades de menor nível herda todos os atributos e participa em todas as associações do conjunto de entidades de maior nível ao qual está ligado.

Exemplo de Especialização



Restrições de Concepção para a Especialização/Generalização

Restrição de pertença especifica se uma entidade no conjunto de maior nível pode ou não pertencer a mais que um conjunto do nível inferior.

- disjuntas : só pode pertencer a um dos níveis inferiores (anotado com a palavra “disjunta” ao lado do triângulo).
- sobrepostas: pode pertencer a mais do que um.

Restrições de Concepção para a Especialização/Generalização

Restrição de pertença especifica se uma entidade no conjunto de maior nível pode ou não pertencer a mais que um conjunto do nível inferior.

- disjuntas : só pode pertencer a um dos níveis inferiores (anotado com a palavra “disjunta” ao lado do triângulo).
- sobrepostas: pode pertencer a mais do que um.

Restrição de completude especifica se uma entidade no conjunto de maior nível tem ou não que pertencer a pelo menos um dos conjuntos do nível inferior.

- total: tem de pertencer pelo menos a um (anotado com a palavra “total” ao lado do triângulo).
- parcial: pode não pertencer a nenhum.

Agregação

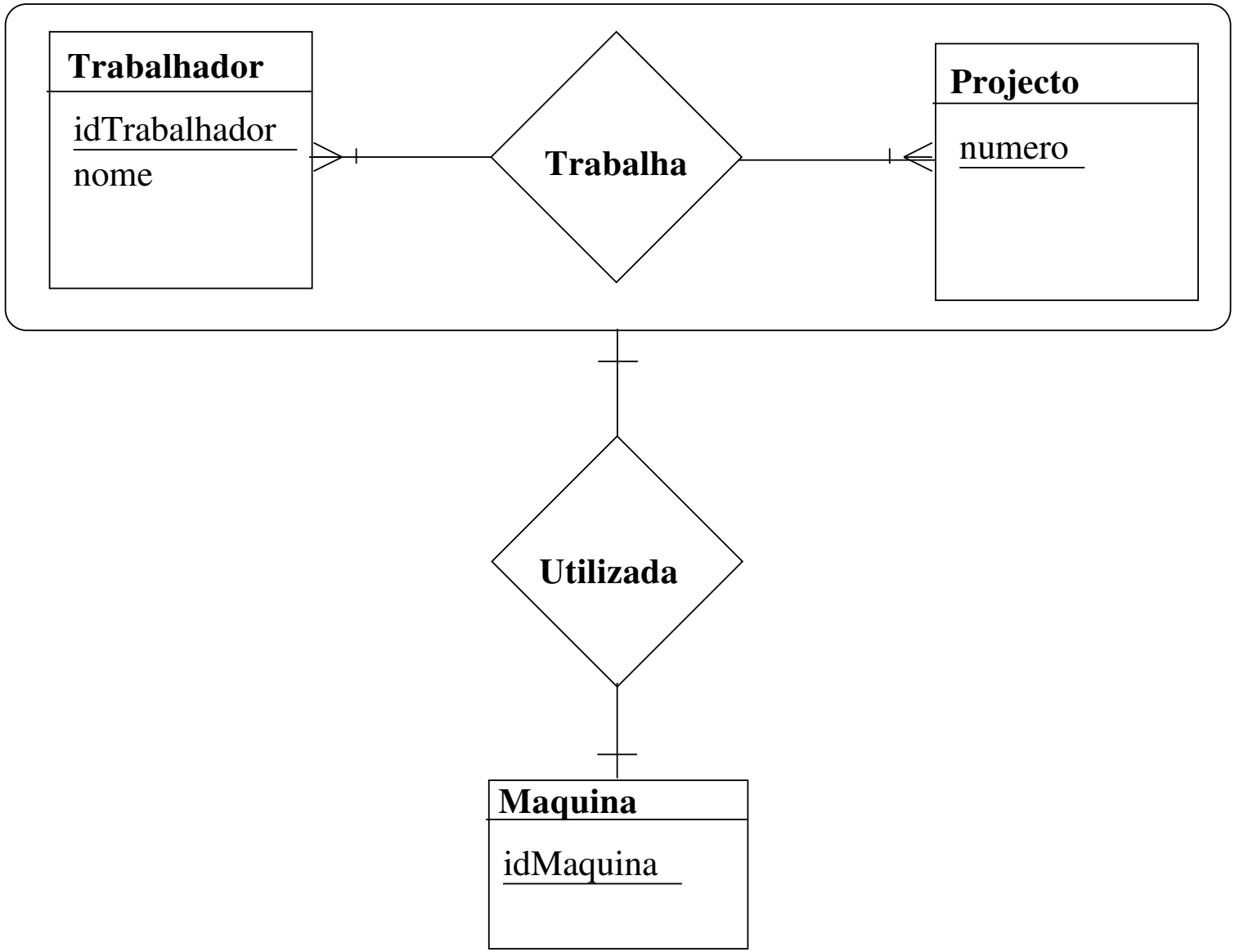
Considere o seguinte exemplo:

- Um empregado pode trabalhar em vários projectos (e num projecto pode haver vários empregados).
- Há que saber que máquinas são usadas por cada empregado em cada projecto.

A associação com máquinas não é feita com empregados nem com projectos considerados isoladamente. Deve é ser feita com a associação (par) (empregados,projectos)
Agregação:

- Trata-se a associação como uma entidade abstracta;
- Permitem-se associações entre associações (ou entre associações e entidades);
- Abstracção de uma associação numa nova entidade.

DEA com Agregação



Decisões de Concepção

- A utilização de um atributo ou conjunto de atributos para representar um objecto.
- Se um conceito da realidade é expresso mais adequadamente com um conjunto de entidades ou de associações.
- Utilização de um conjunto de entidades forte ou fracas.
- Utilização de especialização/generalização — contribui para a modularidade do desenho.
- Utilização de agregação — pode tratar-se o conjunto de entidades agregado independentemente da sua estrutura interna.

Regras para a Concepção de um DEA

- Dividir a informação em múltiplas tabelas;

Regras para a Concepção de um DEA

- Dividir a informação em múltiplas tabelas;
- Não utilizar atributos compostos (1ª Forma Normal);

Regras para a Concepção de um DEA

- Dividir a informação em múltiplas tabelas;
- Não utilizar atributos compostos (1ª Forma Normal);
- Não duplicar informação (2ª Forma Normal);

Regras para a Concepção de um DEA

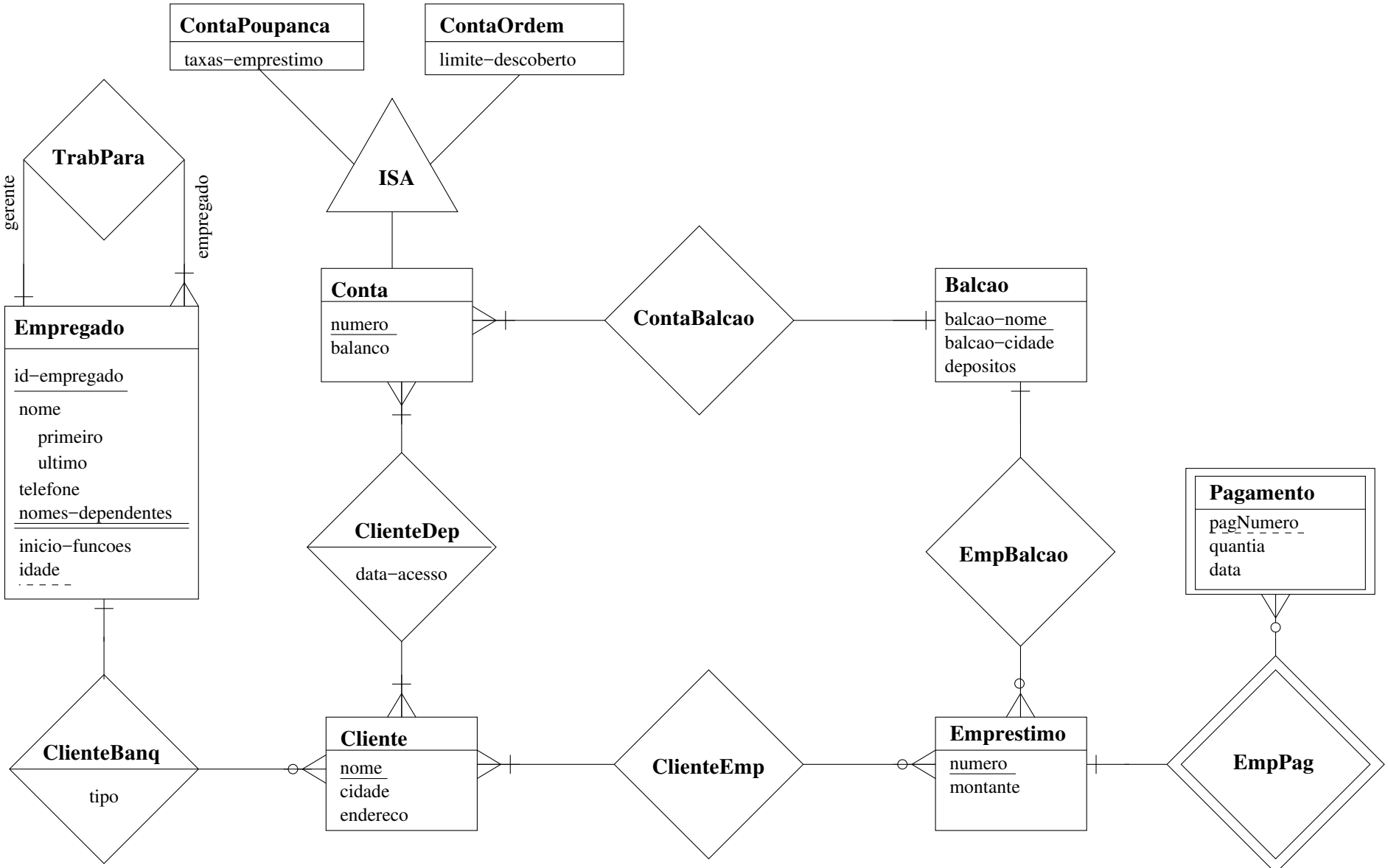
- Dividir a informação em múltiplas tabelas;
- Não utilizar atributos compostos (1ª Forma Normal);
- Não duplicar informação (2ª Forma Normal);
- As Entidade devem ser atómicas;

Regras para a Concepção de um DEA

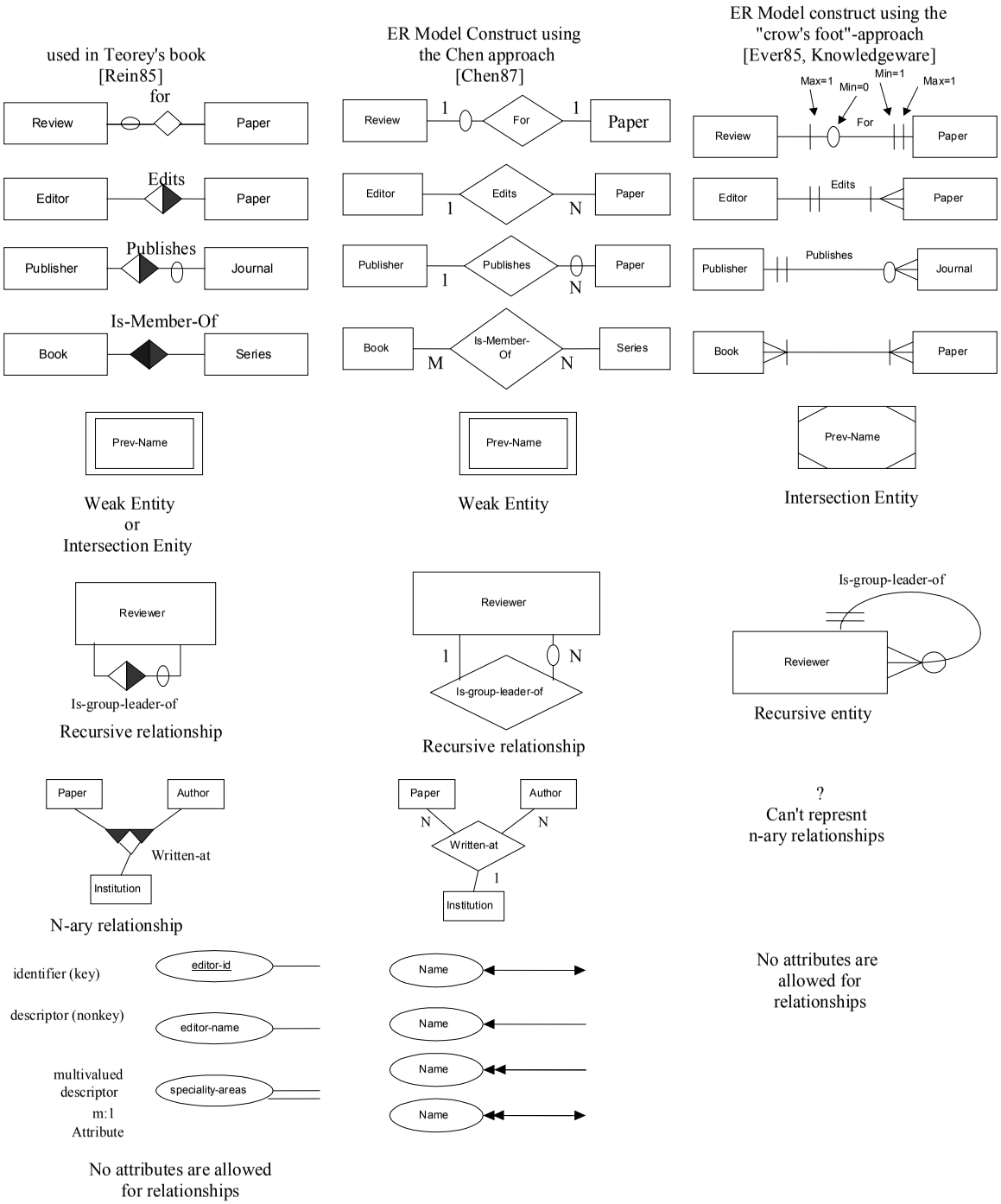
- Dividir a informação em múltiplas tabelas;
- Não utilizar atributos compostos (1ª Forma Normal);
- Não duplicar informação (2ª Forma Normal);
- As Entidade devem ser atómicas;
- Criar uma chave primária unitária (um só atributo) para cada Entidade.

Andy Harris, PHP5/MySQL Programming.

DEA para um Banco



ER Construct Comparison



Modelo Relacional

- Estrutura das Bases de Dados Relacionais
- Conversão entre Modelizações
- Álgebra Relacional
- Operações Estendidas da Álgebra Relacional
- Modificação da Base de Dados
- Vistas

Modelos Relacional

O modelo Entidade-Associação ajuda na modelização dos dados. No entanto a maioria dos SGBDs actuais implementa um modelo diferente, embora com um poder descritivo semelhante, o modelo Relacional. Felizmente é fácil passar de um para o outro.

O Modelo Relacional organiza os dados em:

- (Entidades) tabelas, ou visto de outra forma, relações matemáticas.
- (Associações) tabelas conjuntamente com o mecanismo de integridade referencial (chaves externas).

Modelos Relacional

O modelo Entidade-Associação ajuda na modelização dos dados. No entanto a maioria dos SGBDs actuais implementa um modelo diferente, embora com um poder descritivo semelhante, o modelo Relacional. Felizmente é fácil passar de um para o outro.

O Modelo Relacional organiza os dados em:

- (Entidades) tabelas, ou visto de outra forma, relações matemáticas.
- (Associações) tabelas conjuntamente com o mecanismo de integridade referencial (chaves externas).

Este modelo vai permitir responder a: Como é que os dados estão armazenados? Como consultar os dados? Como alterar os dados?

Exemplo de uma Relação

numPiso	elevador	portaSaida
6	V	F
-1	F	F
0	V	V
2	V	F
3	V	F
1	V	V
4	V	F
5	V	V

A ordem dos tuplos (linhas da tabela) é irrelevante, o ordenamento é dado simplesmente pela ordem de inserção dos elementos na base de dados.

Estrutura Básica

- Dados os conjuntos $D_1, D_2, \ldots, D_n$, tem-se que uma relação r é tal que, $r \subseteq D_1 \times D_2 \times \cdots \times D_n$.
- Uma relação é um conjunto de tuplos $(a_1, a_2, \ldots, a_n)$ com $a_i \in D_i$.

- Os conjuntos D_i são os domínios dos atributos pertencentes às entidades e associações que constituem o modelo da base de dados.
- As relações são então subconjuntos do produto cartesiano dos domínios de variação dos atributos.

Exemplo:

- domínio de `numPiso` = $\mathbb{Z}$;
- domínio de `elevador` = $\mathbb{B}$;
- domínio de `portaSaida` = $\mathbb{B}$.

Então:

$$r = \{ \text{(6,V,F), (23,V,V), (-12,F,F)} \}$$

é uma relação em `numPiso` $\times$ `elevador` $\times$ `portaSaida`

Atributos

- A todo o atributo de uma relação está associado um identificador.
- O conjunto de valores que um atributo pode tomar é chamado de domínio do atributo.
- Normalmente, obriga-se a que os valores dos atributos sejam atómicos, ou seja, indivisíveis (1ª Forma Normal):

  Por exemplo o atributo '`endereco`' é, em geral, um atributo não atómico.

Atributos com Valores Indefinidos

- Pode-se dar a situação de que numa dada relação nem todos os atributos têm um valor atribuído, por exemplo na relação.

  `Empregado =`

  `(idEmpregado,nome,telefone,nomesDependentes,inicioFuncoes,idade)`

- Os valores de `telefone` e `nomesDependentes` nem sempre estão definidos.
- Podemos ter tuplos tais como:

  `(23,João,⊥,⊥,1970-1-3,40)`

Em termos da terminologia usual em base de dados vai-se designar esse "valor" por `null`.

- O valor especial `null` pertence a todos os domínios.
- O valor `null` causa complicações na definição de muitas operações.

Ignoraremos o efeito dos valores nulos em grande parte da apresentação mas consideraremos posteriormente as suas implicações.

Esquema de Relação

- Dados os (domínios) atributos $A_1, A_2, \ldots, A_n$:
$$R = (A_1, A_2, \ldots, A_n)$$
é designado por esquema de relação, isto é, é o produto cartesiano de todos os (domínios) atributos.

  $R =$ `EsquemaCliente = (nome,endereço,cidade)`

Esquema de Relação

- Dados os (domínios) atributos $A_1, A_2, \ldots, A_n$:
$$R = (A_1, A_2, \ldots, A_n)$$
é designado por esquema de relação, isto é, é o produto cartesiano de todos os (domínios) atributos.
$$R = \texttt{EsquemaCliente = (nome,endereço,cidade)}$$

- $r(R)$ é uma relação (ou instância de relação) no esquema de relação R, isto é, um dado subconjunto de R.

$$r(R) = \texttt{Cliente(EsquemaCliente)}$$
$$\texttt{= ('João','Rua da Sofia','Coimbra')}$$

Instância de Relação

- Os valores de uma (instância de) relação são descritos por uma tabela.
- Um elemento *t* de *r* é um tuplo, representado por uma linha da tabela.

Por exemplo:

$$R = \texttt{Pisos} = (\texttt{numPiso, elevador, portaSaida})$$

$r(R)$= **atributos**

tuplos

numPiso	**elevador**	**portaSaida**
3	V	F
1	V	V
4	V	F
5	V	V

Chaves

O conceito de chave no modelo relacional é em tudo semelhante ao mesmo conceito já visto no modelo entidade-relação.
Seja $K \subseteq R$.

- K é uma super-chave de R se os valores de K são suficientes para identificar um único tuplo de toda a relação $r(R)$ possível. Por "relação possível" entende-se uma instância r que pode existir na empresa que estamos a modelar.

  Exemplo: `{nome, endereço}` e `{nome}`, são ambas super-chaves de `Cliente` (assumindo-se que não é possível existirem dois clientes com o mesmo nome).

- K é uma chave candidata, se K é minimal.

  Exemplo: `{nome}` é uma chave candidata para `Cliente` dado ser uma super-chave, e nenhum subconjunto dela é uma super-chave.

- De entre as chaves candidatas escolhe-se uma delas como sendo a chave primária da relação.

Integridade Referencial

Chaves Externas (também designadas por "estrangeiras")

- Garante que um valor que ocorre numa relação para um certo conjunto de atributos também ocorre num outro conjunto de atributos de outra relação.
  Exemplo: Se "Coimbra-central" é o nome de uma agência que ocorre num dos tuplos da relação `Conta`, então existe um tuplo na relação `Balcao` para o balcão "Coimbra-central".

- Definição Formal:
  Sejam $r_1(R_1)$ e $r_2(R_2)$ duas relações com chaves primárias K_1 e K_2 respectivamente.
  O subconjunto α de R_2 é uma chave externa referindo K_1 na relação r_1, se para todo t_2 em r_2 existe um tuplo t_1 em r_1 tal que $t_1[K_1] = t_2[\alpha]$.

Nas chaves externas usa-se o sublinhado a tracejado.

Chaves Externas ≡ Associações (no Modelo EA).

Exemplos de Relações

- As relações `Cliente` e `Conta` (entidades no modelo EA)

nome	endereco	cidade
Pedro	R. Angola 12	Coimbra
João	R. Moçambique	Coimbra

numero	balanco
C-1023	2300
C-1034	120
C-304	7500

Exemplos de Relações

- As relações `Cliente` e `Conta` (entidades no modelo EA)

nome	endereco	cidade
Pedro	R. Angola 12	Coimbra
João	R. Moçambique	Coimbra

numero	balanco
C-1023	2300
C-1034	120
C-304	7500

- A relação `ClienteDep` (associação no modelo ER)

nome	numero	dataAcesso
Pedro	C-1023	2007-6-11
João	C-304	2007-06-21
Pedro	C-1034	2006-11-1
Isabel	C-304	2005-03-29

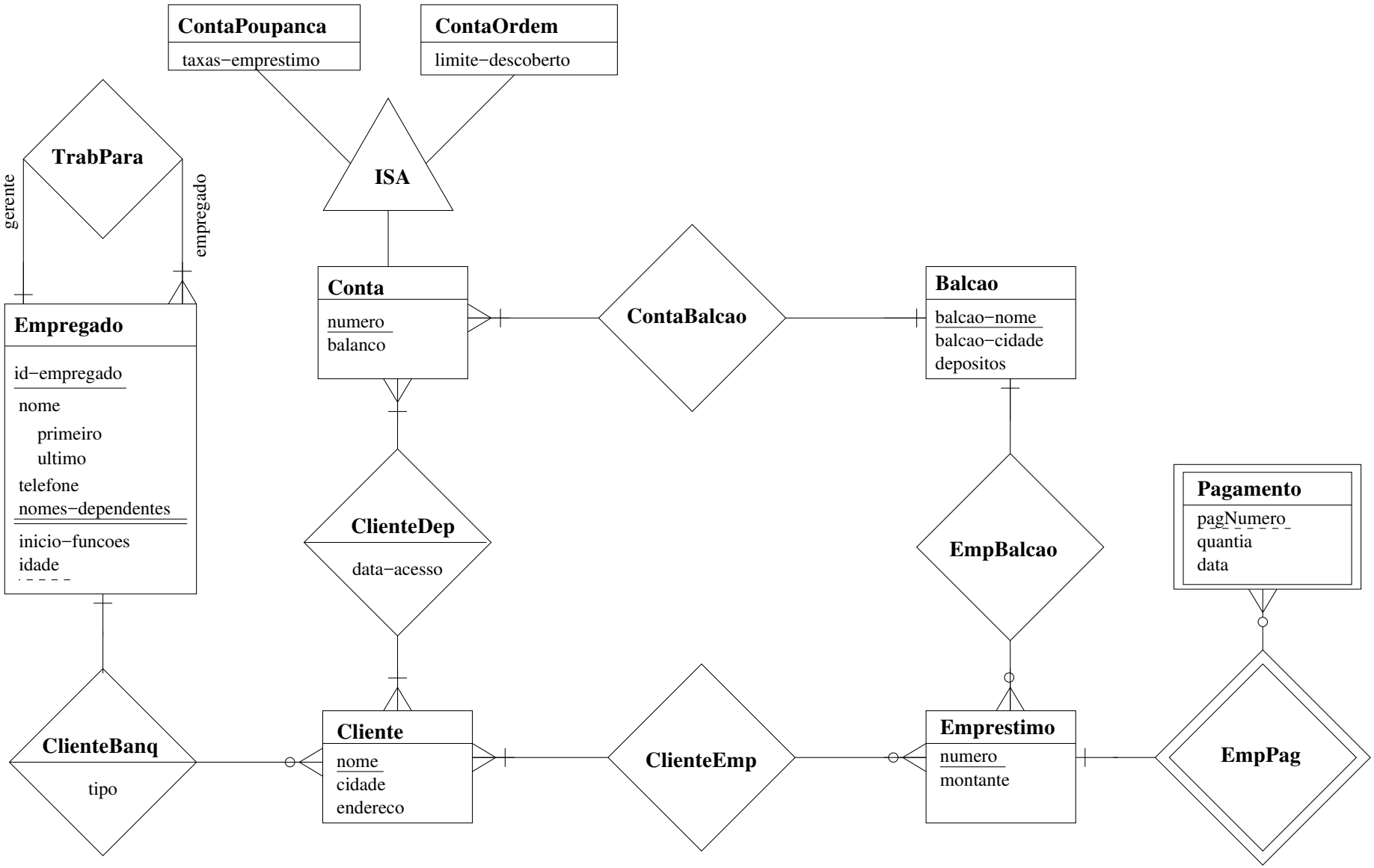
Derivação de Relações a partir de um DEA

Uma base de dados que seja representável por um DEA pode ser também representada por intermédio de um conjunto de relações.

Para cada conjunto de entidades e para cada conjunto de associações gera-se uma única relação (ou tabela) com o nome do conjunto de entidades ou conjunto de associações respectivo.

A conversão de um DEA para um esquema de tabelas constitui a base para a derivação da concepção de uma base de dados relacional a partir de um DEA.

DEA para um Banco



Conjuntos de Entidades como Tabelas

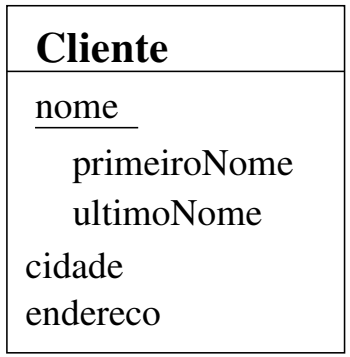
Um conjunto forte de entidades reduz-se a uma relação (tabela) com os mesmos atributos.

Balcao
balcaoNome
balcaoCidade
depositos

`Balcao(balcaoNome,balcaoCidade,depositos)`

Atributos Compostos

Atributos Compostos: cria-se um atributo para cada atributo atómico de um atributo composto.
Por exemplo, considere-se o conjunto de entidades `cliente` com o atributo composto `nome` formado por `primeiroNome` e `ultimoNome`. A tabela derivada contém os atributos `nomePrimeiroNome` e `nomeUltimoNome`.



`Cliente(nomePrimeiroNome,nomeUltimoNome,cidade,endereco)`

Atributos Multi-valor

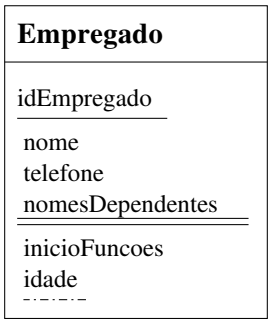
Um atributo multi-valor, `m`, de uma entidade, `E`, é representado através de uma tabela separada `EM`.

A tabela `EM` tem os atributos correspondendo à chave primária de `E` e um atributo correspondendo ao atributo multi-valor `m`.

A chave primária de `EM` é dada pela conjunção dos dois atributos. A chave primária de `E` é uma chave externa de `EM`.

Atributos Multi-valor — Exemplo

Por exemplo, o atributo multi-valor `nomesDependentes` de `Empregado` é representado pela tabela `EmpregadoNomesDependentes(idEmpregado,nomesDependentes)`.



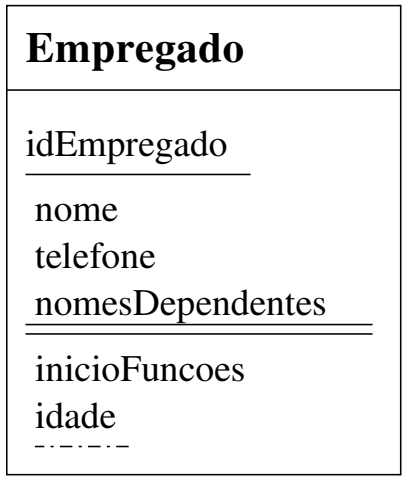
`Empregado(idEmpregado,nome,telefone,inicioFuncoes,idade)`

`NomesDependentes(idEmpregado,nomesDependentes)`

Cada valor de um atributo multi-valor é colocado numa linha separada da tabela `EM`.

Atributos Derivados

Os atributos derivados não têm uma representação directa. Os programas de acesso à informação farão o seu cálculo a partir da informação contida nos atributos dos quais ele é derivado.

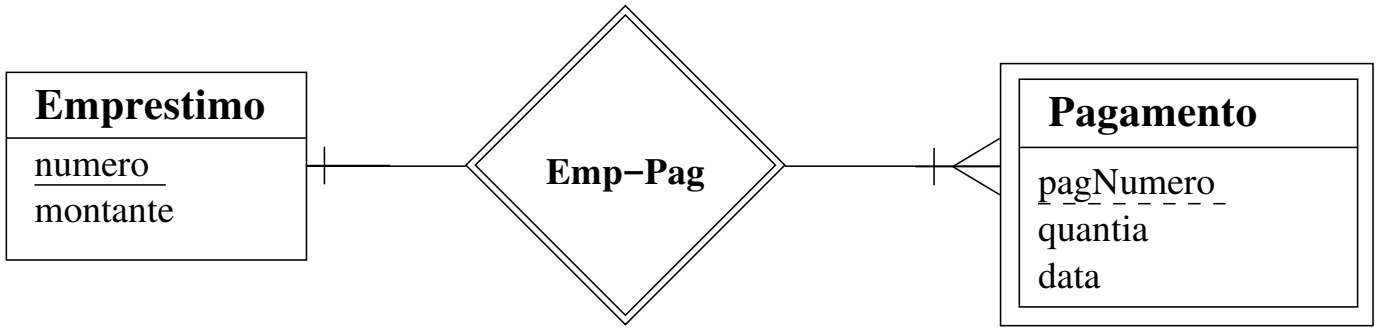


`Empregado(idEmpregado,nome,telefone,inicioFuncoes,dataNascimento)`

Isto é os atributos derivados são substituídos pelos atributos dos quais eles dependem (`idade` substituído por `dataNascimento`) ficando o seu cálculo a cargo dos programas de acesso.

Conjuntos de Entidades Fracas

Um conjunto de entidades fracas é representado por uma relação que inclui colunas para a chave primária do conjunto de entidades identificador, juntamente com as colunas para os restantes atributos do conjunto de entidades fracas. A **chave primária** desta nova entidade é a junção da **chave primária da entidade forte** com o **descritor** da entidade fraca.

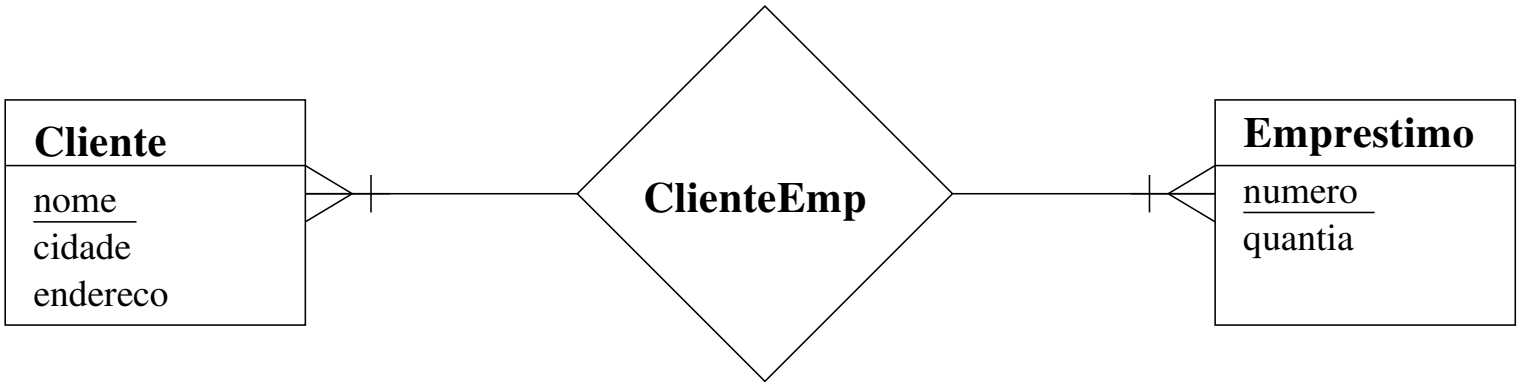


`Emprestimo(numero,montante)`
`Pagamento(numero,pagNumero,quantia,data)`

A chave primária de `Emprestimo` terá o papel adicional de chave externa em `Pagamento`

Conjuntos de Associações

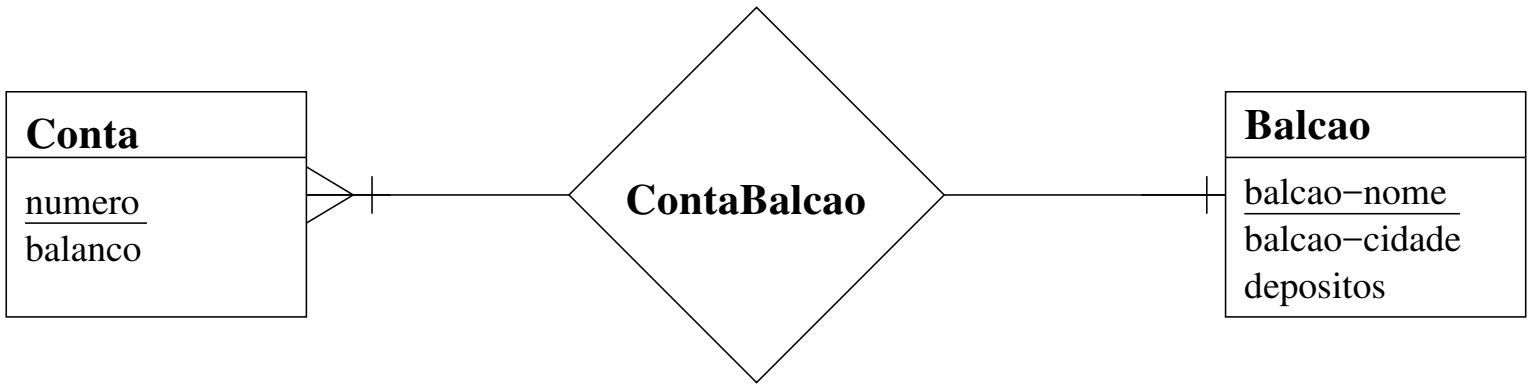
Um conjunto de associações muitos para muitos é representado com uma tabela com colunas para as chaves primárias dos dois conjuntos de entidades participantes, com colunas adicionais para os atributos próprios (ou descritivos) do conjunto de associações.




```
Cliente(nome,cidade,endereco)
Emprestimo(numero,quantia)
ClienteEmp(nome, numero)
```

Tabelas Redundantes

- Conjuntos de associações muitos-para-um (ou um-para-muitos), totais no lado "muitos" podem ser representados adicionando atributos extra ao lado "muitos". Entre esses atributos tem de estar contida a chave primária do outro conjunto participante.
- Por exemplo: Em vez de se criar uma tabela para a associação `ContaBalcao`, adiciona-se a coluna `balcaoNome` à tabela derivada a partir do conjunto de entidades `Conta`.



`Conta(numero,balanco,balcaoNome)`
`Balcao(balcaoNome,balcaoCidade,depositos)`

Redundância de Tabelas & Funções Parcias/Totais

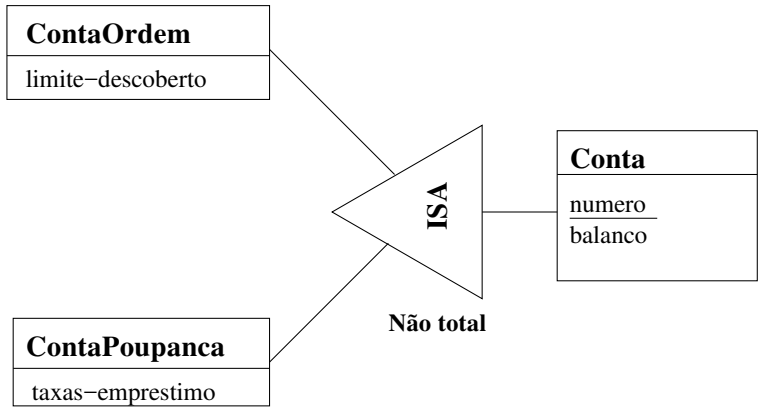
- Para conjuntos de associações um-para-um, qualquer dos lados pode receber a chave primária do outro lado.
- É redundante a tabela correspondente ao conjunto de associações relacionando um conjunto de entidades fracas com o seu conjunto identificador.
  Por exemplo: A tabela `Pagamento` já contém a informação que apareceria na tabela `EmpPag` (as colunas `numeroEmprestimo` e `numeroPagamento`).

As relações **Totais** destinguem-se das **Parcias** por não permitirem valores `NULL`.

- Relações totais têm a restrição **Not Null** activa.
- Relações parciais podem conter valores `Null`.

Derivação de Tabelas para a Especialização

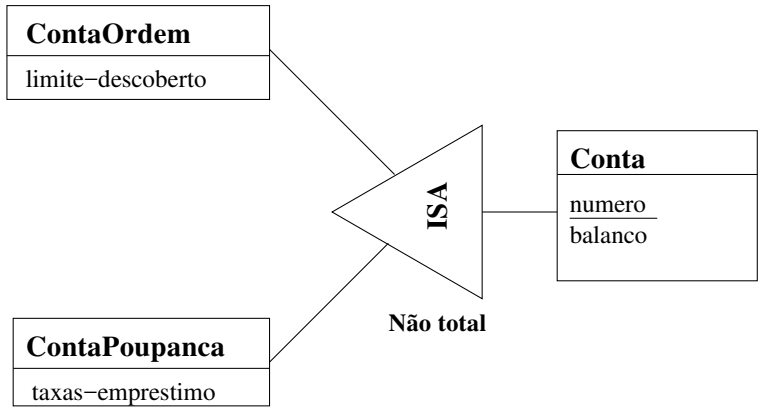
Método 1: “Descendente”



Derivação de Tabelas para a Especialização

Método 1: "Descendente"

- Formar uma tabela para a entidade de maior nível (mais geral).

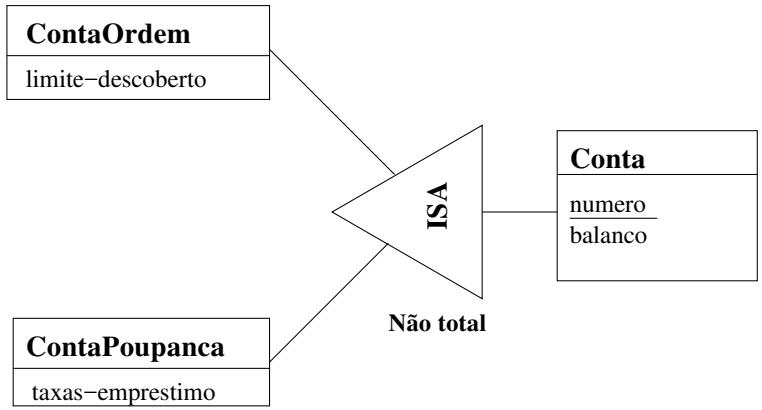


`Conta(numero,balanco)`

Derivação de Tabelas para a Especialização

Método 1: "Descendente"

- Formar uma tabela para a entidade de maior nível (mais geral).
- Criar uma tabela para cada conjunto de entidades de nível abaixo, incluindo a chave primária da entidade acima e os atributos locais.

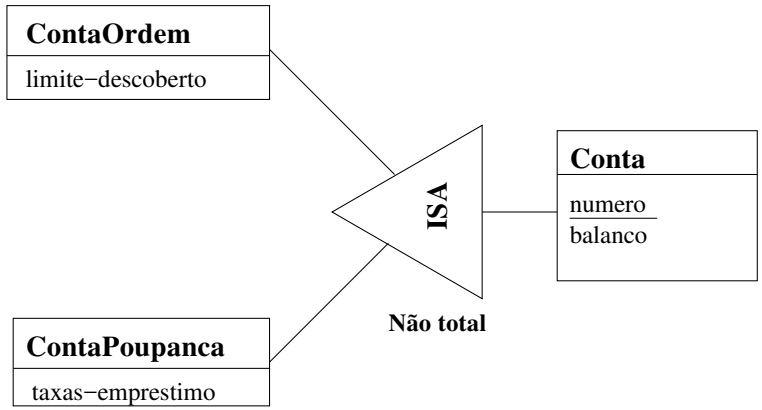


`Conta(numero,balanco)` `ContaOrdem(numero,limiteDescoberto)`
`ContaPoupanca(numero,taxasEmprestimo)`

Derivação de Tabelas para a Especialização

Método 1: "Descendente"

- Formar uma tabela para a entidade de maior nível (mais geral).
- Criar uma tabela para cada conjunto de entidades de nível abaixo, incluindo a chave primária da entidade acima e os atributos locais.

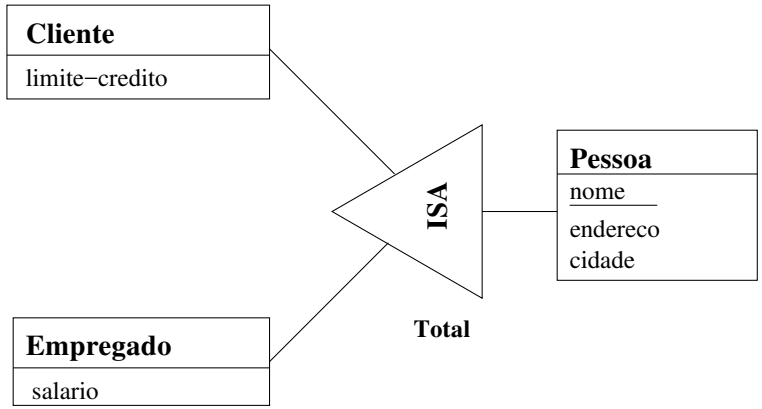


`Conta(numero,balanco)` `ContaOrdem(numero,limiteDescoberto)`
`ContaPoupanca(numero,taxasEmprestimo)`

Vantagem: Podemos guardar a informação de uma conta de outro tipo sem recorrer a valores nulos.
Desvantagem: obter a informação acerca de `ContaPoupanca` (por exemplo) obriga à consulta de duas tabelas

Derivação de Tabelas para a Especialização

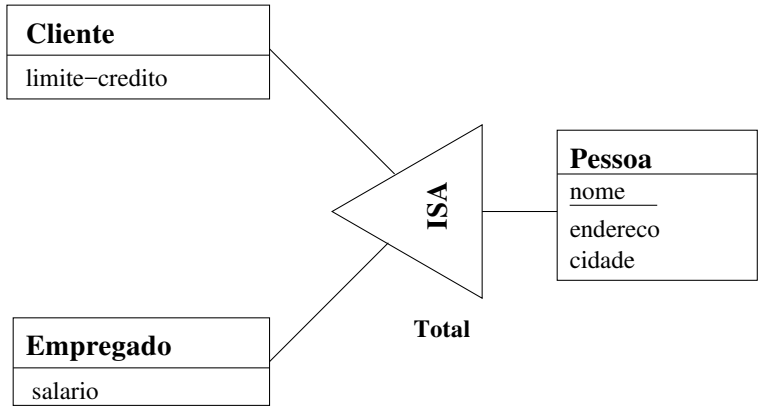
Método 2: “Ascendente”.



Derivação de Tabelas para a Especialização

Método 2: "Ascendente".

- Formar uma tabela para cada conjunto de entidades com os atributos locais e herdados.

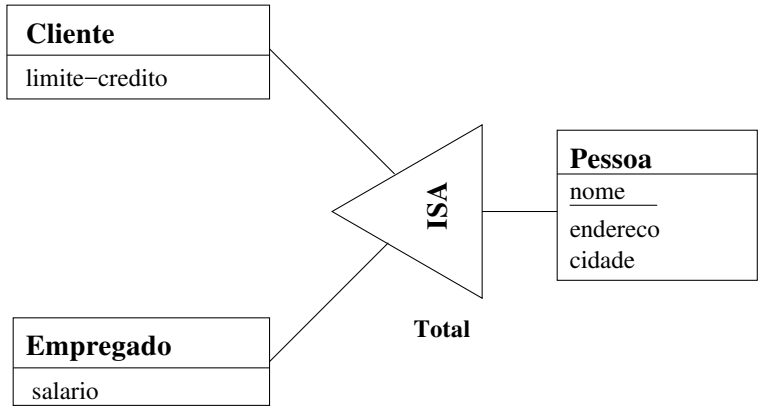



```
Cliente(nome,endereco,cidade,limiteCredito)
Empregado(nome,endereco,cidade,salario)
```

Derivação de Tabelas para a Especialização

Método 2: "Ascendente".

- Formar uma tabela para cada conjunto de entidades com os atributos locais e herdados.

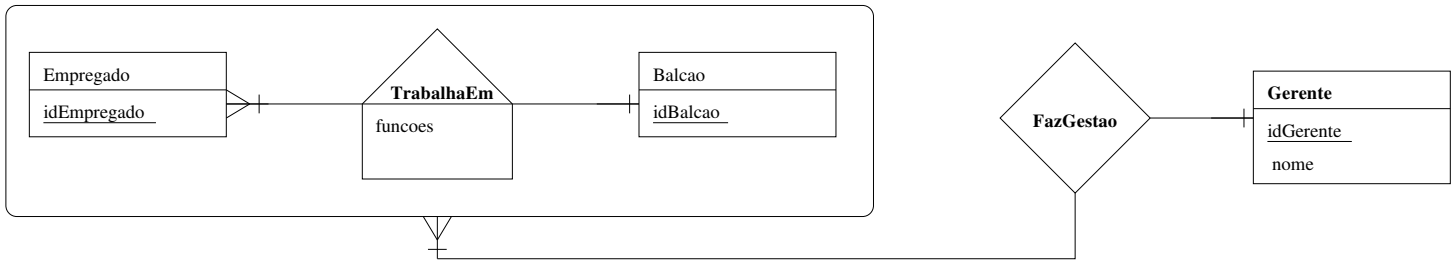



```
Cliente(nome,endereco,cidade,limiteCredito)
Empregado(nome,endereco,cidade,salario)
```

Vantagem: toda a informação sobre um dado tipo numa só tabela.
Desvantagem: no caso de especializações não disjuntas pode ocorrer incoerência na informação. Por exemplo um cliente-empregado com moradas diferentes.

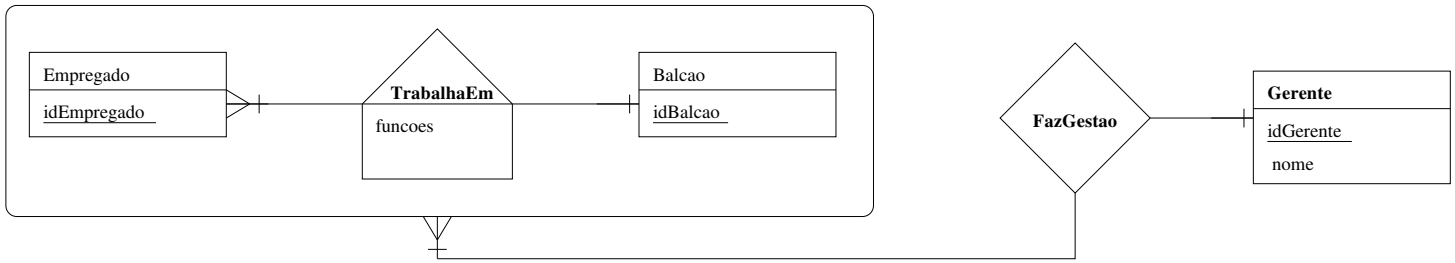
Relações Correspondendo à Agregação

Para representar agregações,



Relações Correspondendo à Agregação

Para representar agregações,

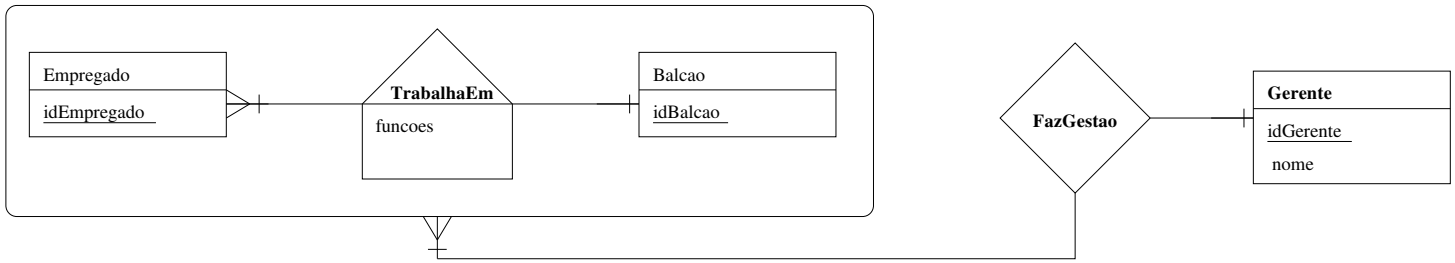


- Representar a associação da agregação explicitamente;

`TrabalhaEm(idEmpregado,idBalcao,funcoes)`

Relações Correspondendo à Agregação

Para representar agregações,

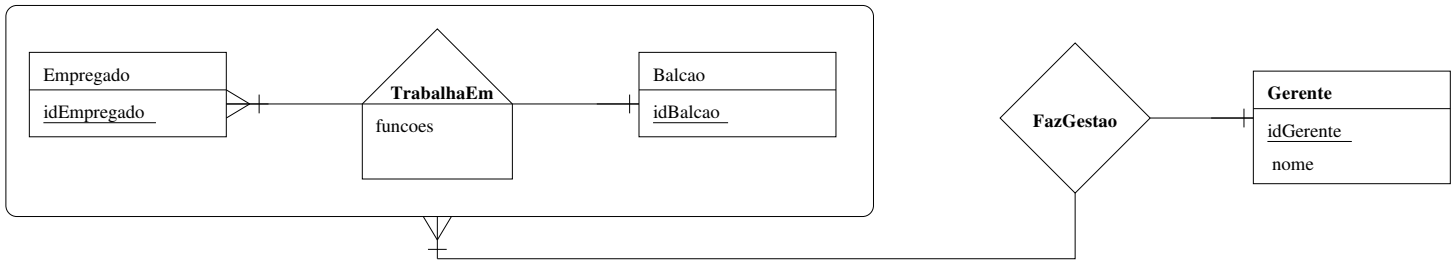


- Representar a associação da agregação explicitamente;
- Construir uma nova relação com:

```
TrabalhaEm(idEmpregado,idBalcao,funcoes)
FazGestao(                              )
```

Relações Correspondendo à Agregação

Para representar agregações,

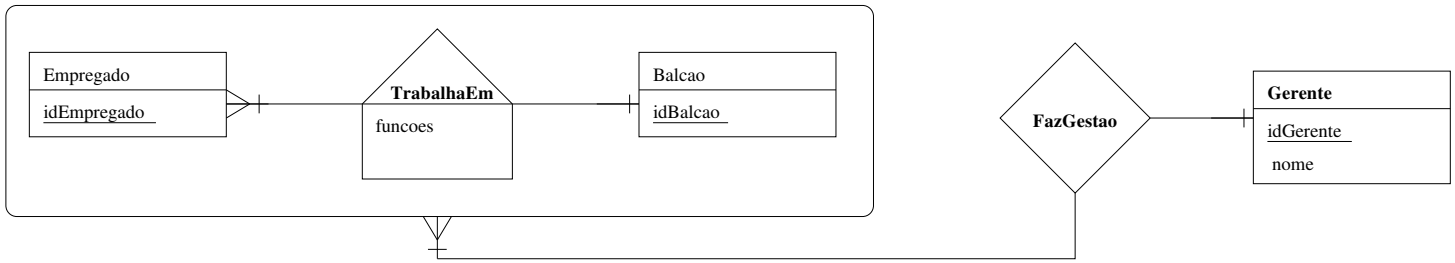


- Representar a associação da agregação explicitamente;
- Construir uma nova relação com:
  - a chave primária da associação agregada (`TrabalhaEm`), neste caso `idEmpregado,idBalcao`, como chave externa;
  - a chave primária do conjunto de entidades participante (`Gerente`), neste caso `idGerente`, como chave externa;

```
TrabalhaEm(idEmpregado,idBalcao,funcoes)
FazGestao(idEmpregado,idBalcao, idGerente)
```

Relações Correspondendo à Agregação

Para representar agregações,

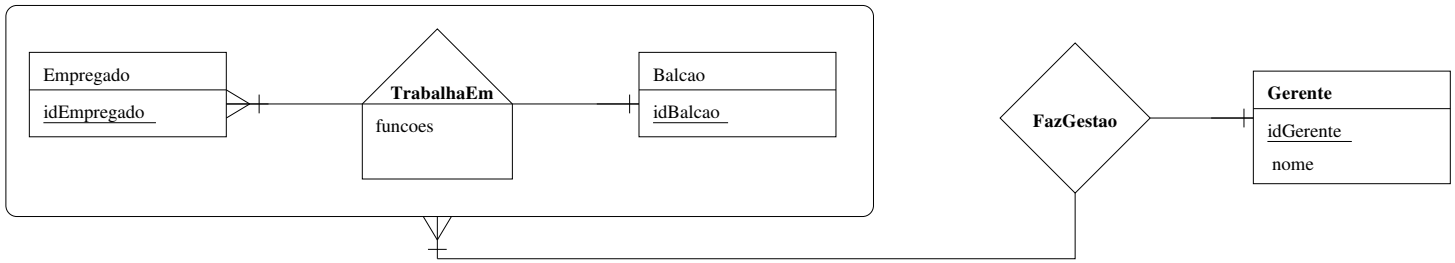


- Representar a associação da agregação explicitamente;
- Construir uma nova relação com:
  - a chave primária da associação agregada (`TrabalhaEm`), neste caso `idEmpregado,idBalcao`, como chave externa;
  - a chave primária do conjunto de entidades participante (`Gerente`), neste caso `idGerente`, como chave externa;
  - Restantes atributos descritivos de `FazGestao`, caso os haja.

`TrabalhaEm(idEmpregado,idBalcao,funcoes)`
`FazGestao(idEmpregado,idBalcao, idGerente)`

Relações Correspondendo à Agregação

Para representar agregações,



- Representar a associação da agregação explicitamente;
- Construir uma nova relação com:
  - a chave primária da associação agregada (`TrabalhaEm`), neste caso `idEmpregado,idBalcao`, como chave externa;
  - a chave primária do conjunto de entidades participante (`Gerente`), neste caso `idGerente`, como chave externa;
  - Restantes atributos descritivos de `FazGestao`, caso os haja.

`TrabalhaEm(idEmpregado,idBalcao,funcoes)`
`FazGestao(idEmpregado,idBalcao, idGerente)`

Dentro da agregação aplicam-se as regras já descritas acima.

Determinação de Chaves a partir do DEA

- Conjunto de entidades fortes. A chave primária do conjunto de entidades é a chave primária da relação.
- Conjunto de entidades fracas. A chave primária da relação consiste na união da chave primária do conjunto de entidades forte com o discriminante do conjunto de entidades fracas.
- Conjunto de relações. A união das chave primárias dos conjuntos de entidades relacionados é uma super-chave da relação.
  - Para conjuntos de associações binários um-para-muitos, a chave primária do lado “muitos” é a chave primária da relação.
  - Para conjuntos de associações um-para-um, a chave primária da relação é a chave primária de um dos conjuntos de entidades.
  - Para conjuntos de associações muitos-para-muitos, a união das chaves primárias é a chave primária da relação.

Modelo Relacional para o Banco (simplificado)

ContaPoupanca(numero, taxaEmprestimo)

ContaOrdem(numero, limiteDescoberto)

Conta(numero, balanco, balcaoNome)

Balcao(balcaoNome, balcaoCidade, depositos)

ClienteDep(nome, numero, dataAcesso)

Cliente(nome, cidade, endereco)

ClienteEmp(nome, numero)

Emprestimo(numero, quantia, balcao)

Linguagem de Consulta/Interrogação

Linguagem a que o utilizador recorre para obter informação a partir da base de dados.

Categorias de linguagens

- Procedimentais (C, . . .)
- **Declarativas** (**SQL**, . . .)

Linguagens Teóricas

- **Álgebra Relacional**
- Cálculo Relacional de Tuplos
- Cálculo Relacional de Domínios

Estas linguagens formam a base teórica das linguagens de consulta utilizadas na prática.

Álgebra Relacional

- Linguagem declarativa
- Seis operadores básicos
  - selecção — definir condições para as quais se quer obter a informação contida numa dada relação.
  - projecção — seleccionar, numa relação, quais os atributos que se quer visualizar.
  - renomeação — renomear uma dada relação
  - união
  - diferença de conjuntos
  - produto cartesiano
- Os operadores têm como argumentos relações de entrada e devolvem uma relação como resultado.

Base teórica do modelo relacional.

Operações de Consulta — Selecção

Selecção — pretende-se filtrar a informação que pretendemos obter. Para tal define-se um um predicado cujos os argumentos são atributos da relação em questão.

$$\begin{array}{rcccc} \sigma & : & \mathbb{B} \times \mathrm{R} & \longrightarrow & \mathrm{R} \\ & & (P(a_i, \ldots, a_j), r) & \longmapsto & \sigma_{P(a_i,\ldots,a_j)}(r) \end{array}$$

Exemplo

$\sigma_{A=B \wedge D>5}$

A	B	C	D
a	*a*	1	7
a	*b*	5	7
b	*b*	12	3
b	*b*	23	10

$=$

A	B	C	D
a	*a*	1	7
b	*b*	23	10

Selecção

Notação: $\sigma_{P(t)}(r)$, P é designado por predicado de selecção.

Definição: $\sigma_{P(t)}(r) = \{t \mid t \in r \wedge P(t)\}$

- Em que P é uma fórmula do cálculo proposicional constituída por termos ligados por: $\wedge$ (**e**), $\vee$ (**ou**), $\neg$ (**não**).
- Cada termo é uma expressão nos atributos da relação, a qual pode conter:
  - os atributos da relação;
  - constantes;
  - operadores e funções pré-definidas (e.g. $\sin, \cos, \ldots$);
  - operadores relacionais: $=, \neq, >, \geq, <, \leq$.

Exemplo de selecção:

$$\sigma_{\text{balcaoNome}='\text{Baixa}'}(\text{conta})$$

Projecção

Projecção — pretende-se definir quais os predicados é que vão ser visíveis para uma dada relação.

$$\begin{array}{rccc} \Pi & : & \mathbb{A} \times \mathrm{R} & \longrightarrow & \mathrm{R} \\ & & (a_i, \ldots, a_j, r) & \longmapsto & \Pi_{a_i,\ldots,a_j}(r) \end{array}$$

com $\mathbb{A}$ um sub-conjunto dos atributos de R.

Exemplo:

$\Pi_{A,C}$

A	B	C
a	10	1
a	20	1
b	30	1
b	40	2

$=$

A	C
a	1
a	1
b	1
b	2

$=$

A	C
a	1
b	1
b	2

Projecção

Notação: $\Pi_{a_1,\ldots,a_k}(r)$, com $a_i \in \mathbb{A} \subseteq R$, para $1 \leq i \leq k$. Isto é os a_is são atributos de R.

Definição: $\Pi_{a_1,\ldots,a_k}(r)$ é definida como sendo a instância de relação r, contendo somente as colunas referentes aos atributos $a_1, \ldots, a_k$.

Dado o resultado final ser uma relação (conjunto) os tuplos repetidos são automaticamente retirados.

Exemplo de projecção:

$$\Pi_{\text{numero,quantia}}(\text{conta})$$

elimina-se o atributo `balcaoNome` da relação `conta`.

União

União — pretende-se fazer a união dos tuplos de duas relações.

$$\begin{array}{rccc} \bigcup & : & \mathrm{R} \times \mathrm{R} & \longrightarrow & \mathrm{R} \\ & & (r, s) & \longmapsto & r \bigcup s \end{array}$$

Exemplo

A	B
a	1
a	2
b	1

$\bigcup$

A	B
a	2
b	3

$=$

A	B
a	1
a	2
b	1
b	3

União

Notação: $r \bigcup s$.

Definição:

$$r \bigcup s = \{t \mid t \in r \vee t \in s\}$$

Pré-condições: Para $r \bigcup s$ ser válida:

- r, s devem ter a mesma aridade (igual número de atributos);
- os atributos têm de ser compatíveis (valores de tipos compatíveis);

Exemplo de união:

$$\Pi_{\text{nomeCliente}}(\mathit{ClienteDep}) \bigcup \Pi_{\text{nomeCliente}}(\mathit{ClienteEmp})$$

determina quais os nomes dos clientes que têm uma conta, ou um empréstimo.
A utilização da operação de projecção é aqui necessária para compatibilizar duas relações que, à partida, não são compatíveis.

Diferença

Diferença — pretende-se fazer a diferença (de conjuntos) dos tuplos de duas relações.

$$\begin{array}{rccc} - & : & \mathrm{R} \times \mathrm{R} & \longrightarrow & \mathrm{R} \\ & & (r, s) & \longmapsto & r - s \end{array}$$

Exemplo

A	B
a	1
a	2
b	1

$-$

A	B
a	2
b	3

$=$

A	B
a	1
b	1

Diferença

Notação: $r - s$.

Definição:

$$r - s = \{t \mid t \in r \wedge t \notin s\}$$

Pré-condições: Para $r - s$ ser válida:

- r, s devem ter a mesma aridade (igual número de atributos);
- os atributos têm de ser compatíveis (valores de tipos compatíveis);

Exemplo de diferença:

$$\Pi_{\text{nomeCliente}}(\mathit{ClienteDep}) - \Pi_{\text{nomeCliente}}(\mathit{ClienteEmp})$$

determina quais os nomes dos clientes que têm uma conta depósito, mas não possuem nenhum empréstimo.
A utilização da operação de projecção é aqui necessária para compatibilizar duas relações que, à partida, não são compatíveis.

Produto Cartesiano

Produto Cartesiano — pretende-se fazer a junção das duas relações através do seu produto cartesiano.

$$\begin{array}{rcccc} \times & : & \mathrm{R}\times\mathrm{R} & \longrightarrow & \mathrm{R} \\ & & (r,s) & \longmapsto & r\times s \end{array}$$

Exemplo

A	B
a	1
b	2

$\times$

C	D	E
a	10	*x*
b	10	*x*
b	20	*y*
c	10	*y*

$=$

A	B	C	D	E
a	1	*a*	10	*x*
a	1	*b*	10	*x*
a	1	*b*	20	*y*
a	1	*c*	10	*y*
b	2	*a*	10	*x*
b	2	*b*	10	*x*
b	2	*b*	20	*y*
b	2	*c*	10	*y*

Produto Cartesiano

Notação: $r \times s$.

Definição:

$$r \times s = \{tq \mid t \in r \wedge q \in s\}$$

Pré-condições: Para $r \times s$ ser válida:

- as relações R, S devem ser disjuntas.

Se as relações não forem disjuntas, isto é, possuírem ambas um atributo com o mesmo nome, ter-se-á de utilizar renomeações (veremos já de seguida como).

Renomeação

- Permite dar um nome, e portanto referir, aos resultados de expressões de álgebra relacional.
- Permite que uma relação seja referida por mais do que um nome.
- Exemplo:

$$\rho_X(E)$$

  Devolve a expressão E com o nome X.
- Se uma expressão de álgebra relacional E tem aridade n, então:

$$\rho_{X(a_1,a_2,\ldots,a_n)}(E)$$

  Devolve a expressão E com o nome X, e com os atributos renomeados para $a_1, a_2, \ldots, a_n$.

Renomeação

Exemplo:

$$\rho_{X(a_1,a_2,a_3,a_4)}(r(A,B,C,D)) = X(a_1,a_2,a_3,a_4)$$

Pode-se dar o caso de só se pretender mudar o nome da relação, mantendo os nomes dos atributos. Nesse caso omite-se a componente referente aos atributos. Por exemplo:

$$\rho_X(r(A,B,C,D)) = X(A,B,C,D)$$

Composição de Operações

Pode-se construir expressões combinando várias operações
Exemplo:

$$\sigma_{A=C}\left(\begin{array}{|c|c|}\hline \textbf{A} & \textbf{B} \\ \hline a & 1 \\ b & 2 \\ \hline \end{array} \times \begin{array}{|c|c|c|}\hline \textbf{C} & \textbf{D} & \textbf{E} \\ \hline a & 10 & x \\ b & 10 & x \\ b & 20 & y \\ c & 10 & y \\ \hline \end{array}\right) = \sigma_{A=C}\left(\begin{array}{|c|c|c|c|c|}\hline \textbf{A} & \textbf{B} & \textbf{C} & \textbf{D} & \textbf{E} \\ \hline a & 1 & a & 10 & x \\ a & 1 & b & 10 & x \\ a & 1 & b & 20 & y \\ a & 1 & c & 10 & y \\ b & 2 & a & 10 & x \\ b & 2 & b & 10 & x \\ b & 2 & b & 20 & y \\ b & 2 & c & 10 & y \\ \hline \end{array}\right) =$$

$$= \begin{array}{|c|c|c|c|c|}\hline \textbf{A} & \textbf{B} & \textbf{C} & \textbf{D} & \textbf{E} \\ \hline a & 1 & a & 10 & x \\ b & 2 & b & 10 & x \\ b & 2 & b & 20 & y \\ \hline \end{array}$$

Definição Formal

Uma expressão básica na álgebra relacional é:

- Uma relação na base de dados;
- Uma relação constante.

Sejam E_1 e E_2 expressões de álgebra relacional; então todas as expressões abaixo são expressões de álgebra relacional:

- $E_1 \bigcup E_2$;
- $E_1 - E_2$;
- $E_1 \times E_2$;
- $\sigma_P(E_1)$, com P um predicado nos atributos de E_1;
- $\Pi_S(E_1)$, com S uma lista de alguns dos atributos de E_1;
- $\rho_X(E_1)$, com X um novo nome para a relação E_1;

Exemplo Bancário

Modelo Relacional

Balcao(nomeBalcao, cidadeBalcao, depositos)

Cliente(nomeCliente, enderecoCliente, tipoCliente)

Conta(nConta, nomeBalcao, balanco)

Emprestimo(nEmprestimo, nomeBalcao, quantia)

TemConta(nomeCliente, nConta)

TemEmprestimo(nomeCliente,nEmprestimo)

Exemplo Bancário – Consultas

- Determinar todos os empréstimos superiores a 1200€

Exemplo Bancário – Consultas

- Determinar todos os empréstimos superiores a 1200€

$$\sigma_{\text{quantia}>1200}(\text{emprestimo})$$

Exemplo Bancário – Consultas

- Determinar todos os empréstimos superiores a 1200€

$$\sigma_{\text{quantia}>1200}(\text{emprestimo})$$

- Encontrar os números dos empréstimos de montante superior a 1200€

Exemplo Bancário – Consultas

- Determinar todos os empréstimos superiores a 1200€

$$\sigma_{\text{quantia}>1200}(\text{emprestimo})$$

- Encontrar os números dos empréstimos de montante superior a 1200€

$$\Pi_{\text{nEmprestimo}}(\sigma_{\text{quantia}>1200}(\text{emprestimo}))$$

Exemplo Bancário – Consultas

- Determinar todos os empréstimos superiores a 1200€

$$\sigma_{\text{quantia}>1200}(\text{emprestimo})$$

- Encontrar os números dos empréstimos de montante superior a 1200€

$$\Pi_{\text{nEmprestimo}}(\sigma_{\text{quantia}>1200}(\text{emprestimo}))$$

- Listar os nomes de todos os clientes que têm um empréstimo, uma conta, ou ambas as coisas

Exemplo Bancário – Consultas

- Determinar todos os empréstimos superiores a 1200€

$$\sigma_{\text{quantia}>1200}(\text{emprestimo})$$

- Encontrar os números dos empréstimos de montante superior a 1200€

$$\Pi_{\text{nEmprestimo}}(\sigma_{\text{quantia}>1200}(\text{emprestimo}))$$

- Listar os nomes de todos os clientes que têm um empréstimo, uma conta, ou ambas as coisas

$$\Pi_{\text{nomeCliente}}(\text{temEmprestimo}) \bigcup \Pi_{\text{nomeCliente}}(\text{temConta})$$

Exemplo Bancário – Consultas (cont.)

- Encontrar os clientes que têm um empréstimo e uma conta no banco[1].

Exemplo Bancário – Consultas (cont.)

- Encontrar os clientes que têm um empréstimo e uma conta no banco[1].

$$\Pi_{\text{nomeCliente}}(\text{temEmprestimo}) \bigcap \Pi_{\text{nomeCliente}}(\text{temConta})$$

Exemplo Bancário – Consultas (cont.)

- Encontrar os clientes que têm um empréstimo e uma conta no banco[1].

$$\Pi_{\text{nomeCliente}}(\text{temEmprestimo}) \bigcap \Pi_{\text{nomeCliente}}(\text{temConta})$$

- Determinar todos os clientes que têm um empréstimo na agência de Coimbra-Baixa.

Exemplo Bancário – Consultas (cont.)

- Encontrar os clientes que têm um empréstimo e uma conta no banco[1].

$$\Pi_{\text{nomeCliente}}(\text{temEmprestimo}) \bigcap \Pi_{\text{nomeCliente}}(\text{temConta})$$

- Determinar todos os clientes que têm um empréstimo na agência de Coimbra-Baixa.

$$\Pi_{\text{nomeCliente}}(\sigma_{\text{nomeBalcao}=\text{"Coimbra-Baixa"}}(\sigma_{\text{temEmprestimo.nEmprestimo}=\text{emprestimo.nEmprestimo}}(\textit{emprestimo} \times \textit{temEmprestimo})))$$

Exemplo Bancário – Consultas (cont.)

- Encontrar os clientes que têm um empréstimo e uma conta no banco[1].

$$\Pi_{\text{nomeCliente}}(\text{temEmprestimo}) \bigcap \Pi_{\text{nomeCliente}}(\text{temConta})$$

- Determinar todos os clientes que têm um empréstimo na agência de Coimbra-Baixa.

$$\Pi_{\text{nomeCliente}}(\sigma_{\text{nomeBalcao}=\text{"Coimbra-Baixa"}}(\sigma_{\text{temEmprestimo.nEmprestimo}=\text{emprestimo.nEmprestimo}}(\mathit{emprestimo} \times \mathit{temEmprestimo})))$$

- Listar os nomes dos clientes que possuem um empréstimo na agência de Coimbra-Baixa mas que não tem nenhuma conta no banco.

Exemplo Bancário – Consultas (cont.)

- Encontrar os clientes que têm um empréstimo e uma conta no banco[1].

$$\Pi_{\text{nomeCliente}}(\text{temEmprestimo}) \bigcap \Pi_{\text{nomeCliente}}(\text{temConta})$$

- Determinar todos os clientes que têm um empréstimo na agência de Coimbra-Baixa.

$$\Pi_{\text{nomeCliente}}(\sigma_{\text{nomeBalcao}=\text{"Coimbra-Baixa"}}(\sigma_{\text{temEmprestimo.nEmprestimo}=\text{emprestimo.nEmprestimo}}(\textit{emprestimo} \times \textit{temEmprestimo})))$$

- Listar os nomes dos clientes que possuem um empréstimo na agência de Coimbra-Baixa mas que não tem nenhuma conta no banco.

$$\Pi_{\text{nomeCliente}}(\sigma_{\text{nomeBalcao}=\text{"Coimbra-Baixa"}}(\sigma_{\text{temEmprestimo.nEmprestimo}=\text{emprestimo.nEmprestimo}}(\text{emprestimo} \times \text{temEmprestimo}))) - \Pi_{\text{nomeCliente}}(\text{temConta})$$

Exemplo Bancário – Consultas (cont.)

- Determinar todos os clientes que têm um empréstimo na agência de Coimbra-Baixa.

Exemplo Bancário – Consultas (cont.)

- Determinar todos os clientes que têm um empréstimo na agência de Coimbra-Baixa.
  - primeira possibilidade

Exemplo Bancário – Consultas (cont.)

- Determinar todos os clientes que têm um empréstimo na agência de Coimbra-Baixa.
  - primeira possibilidade

$$\Pi_{\text{nomeCliente}}(\sigma_{\text{nomeBalcao}=\text{"Coimbra-Baixa"}}(\sigma_{\text{temEmprestimo.nEmprestimo}=\text{emprestimo.nEmprestimo}}(\textit{emprestimo} \times \textit{temEmprestimo})))$$

Exemplo Bancário – Consultas (cont.)

- Determinar todos os clientes que têm um empréstimo na agência de Coimbra-Baixa.
  - primeira possibilidade

$$\Pi_{\text{nomeCliente}}(\sigma_{\text{nomeBalcao}=\text{"Coimbra-Baixa"}}(\sigma_{\text{temEmprestimo.nEmprestimo}=\text{emprestimo.nEmprestimo}}(\textit{emprestimo} \times \textit{temEmprestimo})))$$

  - segunda possibilidade

Exemplo Bancário – Consultas (cont.)

- Determinar todos os clientes que têm um empréstimo na agência de Coimbra-Baixa.
  - primeira possibilidade

$$\Pi_{\text{nomeCliente}}(\sigma_{\text{nomeBalcao}=\text{"Coimbra}-\text{Baixa"}}(\sigma_{\text{temEmprestimo.nEmprestimo}=\text{emprestimo.nEmprestimo}}(\textit{emprestimo} \times \textit{temEmprestimo})))$$

  - segunda possibilidade

$$\Pi_{\text{nomeCliente}}(\sigma_{\text{temEmprestimo.nEmprestimo}=\text{emprestimo.nEmprestimo}}((\sigma_{\text{nomeBalcao}=\text{“}\textit{Coimbra}-\textit{Baixa}\text{''}}(\textit{emprestimo})) \times \text{temEmprestimo}))$$

Exemplo Bancário – Consultas (cont.)

- Determinar todos os clientes que têm um empréstimo na agência de Coimbra-Baixa.
  - primeira possibilidade

$$\Pi_{\text{nomeCliente}}(\sigma_{\text{nomeBalcao}=\text{"Coimbra}-\text{Baixa"}}(\sigma_{\text{temEmprestimo.nEmprestimo}=\text{emprestimo.nEmprestimo}}(\textit{emprestimo} \times \textit{temEmprestimo})))$$

  - segunda possibilidade

$$\Pi_{\text{nomeCliente}}(\sigma_{\text{temEmprestimo.nEmprestimo}=\text{emprestimo.nEmprestimo}}((\sigma_{\text{nomeBalcao}=\text{“}\textit{Coimbra}-\textit{Baixa}\text{''}}(\textit{emprestimo})) \times \text{temEmprestimo}))$$

- Determinar o saldo mais elevado entre todas as contas.

Exemplo Bancário – Consultas (cont.)

- Determinar todos os clientes que têm um empréstimo na agência de Coimbra-Baixa.
  - primeira possibilidade

$$\Pi_{\text{nomeCliente}}(\sigma_{\text{nomeBalcao}=\text{"Coimbra-Baixa"}}(\sigma_{\text{temEmprestimo.nEmprestimo}=\text{emprestimo.nEmprestimo}}(\textit{emprestimo} \times \textit{temEmprestimo})))$$

  - segunda possibilidade

$$\Pi_{\text{nomeCliente}}(\sigma_{\text{temEmprestimo.nEmprestimo}=\text{emprestimo.nEmprestimo}}((\sigma_{\text{nomeBalcao}=\text{"}\textit{Coimbra-Baixa}\text{"}}(\textit{emprestimo})) \times \text{temEmprestimo}))$$

- Determinar o saldo mais elevado entre todas as contas.
  - Renomear a relação `conta` como `aux`

$$\text{aux} \longleftarrow \text{conta}$$

Exemplo Bancário – Consultas (cont.)

- Determinar todos os clientes que têm um empréstimo na agência de Coimbra-Baixa.
  - primeira possibilidade

$$\Pi_{\text{nomeCliente}}(\sigma_{\text{nomeBalcao}=\text{"Coimbra-Baixa"}}(\sigma_{\text{temEmprestimo.nEmprestimo}=\text{emprestimo.nEmprestimo}}(\textit{emprestimo} \times \textit{temEmprestimo})))$$

  - segunda possibilidade

$$\Pi_{\text{nomeCliente}}(\sigma_{\text{temEmprestimo.nEmprestimo}=\text{emprestimo.nEmprestimo}}((\sigma_{\text{nomeBalcao}=\text{"}\textit{Coimbra-Baixa}\text{"}}(\textit{emprestimo})) \times \text{temEmprestimo}))$$

- Determinar o saldo mais elevado entre todas as contas.
  - Renomear a relação `conta` como `aux`

$$\text{aux} \longleftarrow \text{conta}$$

  - A consulta é:

$$\Pi_{\text{balanco}}(\text{conta}) - \Pi_{\text{conta.balanco}}(\sigma_{\text{conta.balanco}<\text{aux.balanco}}(\text{conta} \times \rho_{\textit{aux}}(\text{conta})))$$

Exemplo Clínica

medicos(nEmpr,nomeM,especialidade)

pacientes(nBI,nomeP,telefone,morada,idade)

farmacos(codF,nomeF)

consultas(nConsulta,data,nBI,nEmpr)

receitas(codF,nConsulta,quantidade)

Consultas de Exemplo

- Quais os pacientes com mais de 50 anos de idade?

Consultas de Exemplo

- Quais os pacientes com mais de 50 anos de idade?

$$\sigma_{\text{idade}>50}(\text{pacientes})$$

Consultas de Exemplo

- Quais os pacientes com mais de 50 anos de idade?

$$\sigma_{\text{idade}>50}(\text{pacientes})$$

- Quais os nomes dos pacientes com mais de 50 anos de idade?

Consultas de Exemplo

- Quais os pacientes com mais de 50 anos de idade?

$$\sigma_{\text{idade}>50}(\text{pacientes})$$

- Quais os nomes dos pacientes com mais de 50 anos de idade?

$$\Pi_{\text{nomeP}}(\sigma_{\text{idade}>50}(\text{pacientes}))$$

Consultas de Exemplo

- Quais os pacientes com mais de 50 anos de idade?

$$\sigma_{\text{idade}>50}(\text{pacientes})$$

- Quais os nomes dos pacientes com mais de 50 anos de idade?

$$\Pi_{\text{nomeP}}(\sigma_{\text{idade}>50}(\text{pacientes}))$$

- Quais os fármacos que já foram receitados em consultas da clínica?

Consultas de Exemplo

- Quais os pacientes com mais de 50 anos de idade?

$$\sigma_{\text{idade}>50}(\text{pacientes})$$

- Quais os nomes dos pacientes com mais de 50 anos de idade?

$$\Pi_{\text{nomeP}}(\sigma_{\text{idade}>50}(\text{pacientes}))$$

- Quais os fármacos que já foram receitados em consultas da clínica?

$$\Pi_{\text{nomeF}}(\sigma_{\text{receitas.codF}=\text{farmacos.codF}}(\text{receitas} \times \text{farmacos}))$$

Consultas de Exemplo

- Quais os pacientes com mais de 50 anos de idade?

$$\sigma_{\text{idade}>50}(\text{pacientes})$$

- Quais os nomes dos pacientes com mais de 50 anos de idade?

$$\Pi_{\text{nomeP}}(\sigma_{\text{idade}>50}(\text{pacientes}))$$

- Quais os fármacos que já foram receitados em consultas da clínica?

$$\Pi_{\text{nomeF}}(\sigma_{\text{receitas.codF}=\text{farmacos.codF}}(\text{receitas} \times \text{farmacos}))$$

- Quais os fármacos que nunca foram receitados?

Consultas de Exemplo

- Quais os pacientes com mais de 50 anos de idade?

$$\sigma_{\text{idade}>50}(\text{pacientes})$$

- Quais os nomes dos pacientes com mais de 50 anos de idade?

$$\Pi_{\text{nomeP}}(\sigma_{\text{idade}>50}(\text{pacientes}))$$

- Quais os fármacos que já foram receitados em consultas da clínica?

$$\Pi_{\text{nomeF}}(\sigma_{\text{receitas.codF}=\text{farmacos.codF}}(\text{receitas} \times \text{farmacos}))$$

- Quais os fármacos que nunca foram receitados?

$$\Pi_{\text{nomeF}}(\text{farmacos}) - \Pi_{\text{nomeF}}(\sigma_{\text{receitas.codF}=\text{farmacos.codF}}(\text{receitas} \times \text{farmacos})))$$

Consultas de exemplo

- Qual a idade do paciente mais velho?
  - Renomear a relação pacientes como *d*

Consultas de exemplo

- Qual a idade do paciente mais velho?
  - Renomear a relação pacientes como *d*
  - A consulta é:

$$\Pi_{\text{idade}}(\textit{pacientes}) -$$
$$- \quad \Pi_{\text{pacientes.idade}}(\sigma_{\text{pacientes.idade<d.idade}}(\text{pacientes} \times \rho_{d}(\text{pacientes}))$$

Consultas de exemplo

- Qual a idade do paciente mais velho?
  - Renomear a relação pacientes como *d*
  - A consulta é:

$$\Pi_{\text{idade}}(\textit{pacientes}) -$$
$$- \quad \Pi_{\text{pacientes.idade}}(\sigma_{\text{pacientes.idade}<\text{d.idade}}(\text{pacientes} \times \rho_{d}(\text{pacientes}))$$

- E quais os (nomes dos) pacientes com essa idade?

Consultas de exemplo

- Qual a idade do paciente mais velho?
  - Renomear a relação pacientes como *d*
  - A consulta é:

$$\Pi_{\text{idade}}(\textit{pacientes}) -$$
$$- \quad \Pi_{\text{pacientes.idade}}(\sigma_{\text{pacientes.idade<d.idade}}(\text{pacientes} \times \rho_{d}(\text{pacientes}))$$

- E quais os (nomes dos) pacientes com essa idade?
  - Seja r a relação da pergunta anterior:

Consultas de exemplo

- Qual a idade do paciente mais velho?
  - Renomear a relação pacientes como *d*
  - A consulta é:

$$\Pi_{\text{idade}}(\textit{pacientes}) -$$
$$- \quad \Pi_{\text{pacientes.idade}}(\sigma_{\text{pacientes.idade<d.idade}}(\text{pacientes} \times \rho_{d}(\text{pacientes}))$$

- E quais os (nomes dos) pacientes com essa idade?
  - Seja r a relação da pergunta anterior:

$$\Pi_{\text{nomeP}}(\sigma_{\text{pacientes.idade=r.idade}}(\text{pacientes} \times r))$$

Operações Adicionais

Definem-se outras operações que não aumentam o poder expressivo da álgebra relacional, mas simplificam algumas consultas habituais.

- Intersecção de conjuntos
- Junção Natural
- Divisão
- Atribuição

Operação de Intersecção de Conjuntos

- Notação: $r \cap s$
- Definida por:

$$r \cap s = \{t \mid t \in r \wedge t \in s\}$$

- Assume-se que:
  - r e s têm a mesma aridade;
  - Os atributos de r e s são compatíveis.
- Note que:

$$r \cap s = r - (r - s) = s - (s - r)$$

Intersecção de Conjuntos - Exemplo

$r =$

A	B
α	1
α	2
β	1

$s =$

A	B
α	2
β	3

$r \cap s =$

A	B
α	2

Operação de Junção Natural

- Notação: $r \bowtie s$
- Sejam r e s relações nos esquemas R e S respectivamente. O resultado é uma relação no esquema união dos esquemas R e S que é obtido considerando cada par de tuplos t_r de r e t_s de s. A intersecção dos esquemas deve ser não vazia.
- Se t_r e t_s têm o mesmo valor em cada um dos atributos da intersecção dos esquemas R e S, um tuplo t é adicionado ao resultado, em que:
  - O resultado é uma relação cujo conjunto de atributos é a reunião dos atributos de R e S (obtido considerando cada par de tuplos t_r de r e t_s de s).
  - Se t_r e t_s têm o mesmo valor em cada um dos atributos pertencente ao conjunto de atributos intersecção dos atributos de R e S, então o tuplo t é adicionado ao resultado. Caso contrário o tuplo não é considerado no resultado final.
- Exemplo:

$$\begin{aligned} R &= (A, B, C, D) \\ S &= (E, B, D) \end{aligned}$$

  Esquema resultado: (A, B, C, D, E)
- A junção natural $r \bowtie s$ pode-se definir por (a intersecção dos atributos é: $\{B, D\}$):

$$\Pi_{r.A,r.B,r.C,r.D,s.E}(\sigma_{r.B=s.B \wedge r.D=s.D}(r \times s))$$

Junção Natural – Exemplo

$r =$

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

$s =$

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ϵ

$r \bowtie s =$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Operação de Divisão

$$r \div s$$

- Adequada para consultas que incluam a frase “para todo”.
- Sejam r e s relações nos esquemas R e S respectivamente com
  - $R = (a_1, \ldots, a_m, b_1, \ldots, b_n)$
  - $S = (b_1, \ldots, b_n)$

  O resultado de $r \div s$ é uma relação na diferença dos esquemas R e S, isto é, $(a_1, \ldots, a_m)$

$$r \div s = \{t \mid t \in \Pi_{R-S}(r) \wedge \forall_{u \in s} tu \in r\}$$

Operação de Divisão — Exemplo

r :

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
ϵ	6
ϵ	1
β	2

s :

B
1
2

$r \div s$:

A
α
β

Outro Exemplo de Divisão

r :

A	B	C	D	E
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

s :

D	E
a	1
b	1

$r \div s$:

A	B	C
α	a	γ
γ	a	γ

Operação de Divisão (Cont.)

- Propriedade
  - Seja $q = r \div s$
  - Então q é a maior relação satisfazendo $q \times s \subseteq r$.

- Definição em termos de operações básicas da álgebra rel.
  Sejam $r(R)$ e $s(S)$ relações, com $S \subset R$

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

  Porquê?

  - $\Pi_{R-S}(r) \times s$ dá os elementos de r com todos os valores de S.
  - $\Pi_{R-S,S}(r)$ constrói uma versão de r com os atributos da expressão anterior.
  - $\Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$ dá os tuplos t em $\Pi_{R-S}(r)$ tal que para algum tuplo $u \in s$, $tu \notin r$.

Outro exemplo de divisão (Cont.)

$$\begin{aligned} r \div s &= \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)) \\ &= \Pi_{(A,B,C)}(r) - \Pi_{(A,B,C)}((\Pi_{(A,B,C)}(r) \times s) - \Pi_{(A,B,C,D,E)}(r)) \end{aligned}$$

$$R = (A, B, C, D, E) \quad S = (D, E) \quad R - S = (A, B, C)$$

- $\Pi_{(A,B,C)}(r) \times s$ dá os elementos de r com todos os valores de (D, E).
- $\Pi_{(A,B,C,D,E)}(r)$ construi uma versão de r com os atributos da expressão anterior. Neste caso: $\Pi_{(A,B,C,D,E)}(r) = r$.
- $\Pi_{(A,B,C)}((\Pi_{(A,B,C)}(r) \times s) - \Pi_{(A,B,C,D,E)}(r))$ dá os tuplos t em $\Pi_{(A,B,C)}(r)$ tal que para algum tuplo $u \in s$, $tu \notin r$.

Operação de Atribuição

- A operação de atribuição ($\leftarrow$) permite-nos expressar consultas complexas de uma forma muito conveniente. Escreve-se a consulta como um programa sequencial constituído por uma sequência de atribuições terminada com uma expressão cujo valor é o resultado da consulta.
- A atribuição é sempre efectuada para uma variável de relação temporária.
- Exemplo: escrever $r \div s$ como
  $temp1 \leftarrow \Pi_{R-S}(r)$
  $temp2 \leftarrow \Pi_{R-S}((temp1 \times s) - \Pi_{R-S,S}(r))$
  $resultado = temp1 - temp2$
  - O resultado à direita de $\leftarrow$ é atribuído à variável que se encontra à esquerda de $\leftarrow$.
  - Pode-se utilizar a variável em sub-expressões seguintes.

Consultas de Exemplo

- Listar todos os clientes que têm uma conta em todas as agências localizadas na cidade de Leiria.

$$\Pi_{cliente.nome,balcao.nome}(depositante \bowtie conta) \div \Pi_{balcao.nome}(\sigma_{balcao.cidade='Leiria'}(balcao))$$

Consultas de exemplo

- Quais os fármacos alguma vez prescritos por cardiologistas?

Consultas de exemplo

- Quais os fármacos alguma vez prescritos por cardiologistas?

$$\Pi_{\text{nomeF}}(\text{farmacos} \bowtie \text{receitas} \bowtie \text{consultas} \bowtie \bowtie \sigma_{\text{especialidade}=\text{"cardiologia"}}(\text{medicos}))$$

ou

$$\Pi_{\text{nomeF}}(\sigma_{\text{especialidade}=\text{"cardiologia}}(\text{farmacos} \bowtie \text{receitas} \bowtie \text{consultas} \bowtie \text{medicos}))$$

Consultas de exemplo

- Quais os fármacos alguma vez prescritos por cardiologistas?

$$\Pi_{\text{nomeF}}(\text{farmacos} \bowtie \text{receitas} \bowtie \text{consultas} \bowtie \bowtie \sigma_{\text{especialidade}=\text{"cardiologia"}}(\text{medicos}))$$

ou

$$\Pi_{\text{nomeF}}(\sigma_{\text{especialidade}=\text{"cardiologia}}(\text{farmacos} \bowtie \text{receitas} \bowtie \text{consultas} \bowtie \text{medicos}))$$

- Quais os (nomes dos) fármacos que já foram receitados por todos os médicos da clínica?

$$r \leftarrow \Pi_{\text{codF,nEmpr}}(\text{consultas} \bowtie \text{receitas}) \div \Pi_{\text{nEmpr}}(\textit{medicos})$$
$$\Pi_{\text{nomeF}}(\text{farmacos} \bowtie r)$$

Operações Estendidas da Álgebra Relacional

- Aumentam a expressividade da Álgebra Relacional:
  - Projecção Generalizada
  - Funções de Agregação
  - Junções Internas e Externas

Projecção Generalizada

- Permite a utilização de funções aritméticas na lista de projecção.

$$\Pi_{f_1,f_2,\ldots,f_n}(E)$$

- E é uma expressão arbitrária de álgebra relacional.
- Cada uma das expressões f_1, f_2, ..., f_n é uma expressão aritmética envolvendo constantes e atributos no esquema de E.
- Dada a relação

  `infoCredito(`<u>`nomeCliente`</u>`, limite, balancoCredito)`

  encontrar o quanto cada cliente ainda pode gastar:

$$\Pi_{\text{nomeCliente,limite}-\text{balancoCredito}}(\text{infoCredito})$$

- Há quantos dias foi cada uma das consultas

$$\Pi_{\text{nConsulta,hoje}-\text{data}}(\textit{consultas})$$

Funções de Agregação

- Funções de Agregação aplicam-se a uma colecção de valores e devolvem um único valor (relação com um só atributo e um só valor para esse atributo) como resultado.

`avg`	média dos valores
`min`	mínimo dos valores
`max`	máximo dos valores
`sum`	soma dos valores
`count`	número dos valores

Funções de Agregação

- Funções de Agregação aplicam-se a uma colecção de valores e devolvem um único valor (relação com um só atributo e um só valor para esse atributo) como resultado.

`avg`	média dos valores
`min`	mínimo dos valores
`max`	máximo dos valores
`sum`	soma dos valores
`count`	número dos valores

- Operação de Agregação na álgebra relacional

$$_{g_1,g_2,...,g_m}\mathcal{G}_{f_1(a_1),f_2(a_2),...,f_n(a_n)}(E)$$

Funções de Agregação

- Funções de Agregação aplicam-se a uma colecção de valores e devolvem um único valor (relação com um só atributo e um só valor para esse atributo) como resultado.

`avg`	média dos valores
`min`	mínimo dos valores
`max`	máximo dos valores
`sum`	soma dos valores
`count`	número dos valores

- Operação de Agregação na álgebra relacional

$$_{g_1,g_2,...,g_m}\mathcal{G}_{f_1(a_1),f_2(a_2),...,f_n(a_n)}(E)$$

  - E é uma expressão de álgebra relacional

Funções de Agregação

- Funções de Agregação aplicam-se a uma colecção de valores e devolvem um único valor (relação com um só atributo e um só valor para esse atributo) como resultado.

`avg`	média dos valores
`min`	mínimo dos valores
`max`	máximo dos valores
`sum`	soma dos valores
`count`	número dos valores

- Operação de Agregação na álgebra relacional

$$_{g_1,g_2,\ldots,g_m}\mathcal{G}_{f_1(a_1),f_2(a_2),\ldots,f_n(a_n)}(E)$$

  - E é uma expressão de álgebra relacional
  - $g_1, g_2, \ldots, g_m$ é uma lista de atributos de agrupamento (pode ser vazia)

Funções de Agregação

- Funções de Agregação aplicam-se a uma colecção de valores e devolvem um único valor (relação com um só atributo e um só valor para esse atributo) como resultado.

`avg`	média dos valores
`min`	mínimo dos valores
`max`	máximo dos valores
`sum`	soma dos valores
`count`	número dos valores

- Operação de Agregação na álgebra relacional

$$_{g_1,g_2,\ldots,g_m}\mathcal{G}_{f_1(a_1),f_2(a_2),\ldots,f_n(a_n)}(E)$$

  - E é uma expressão de álgebra relacional
  - $g_1, g_2, \ldots, g_m$ é uma lista de atributos de agrupamento (pode ser vazia)
  - Cada f_i é uma função de agregação

Funções de Agregação

- Funções de Agregação aplicam-se a uma colecção de valores e devolvem um único valor (relação com um só atributo e um só valor para esse atributo) como resultado.

`avg`	média dos valores
`min`	mínimo dos valores
`max`	máximo dos valores
`sum`	soma dos valores
`count`	número dos valores

- Operação de Agregação na álgebra relacional

$$_{g_1,g_2,\ldots,g_m}\mathcal{G}_{f_1(a_1),f_2(a_2),\ldots,f_n(a_n)}(E)$$

  - E é uma expressão de álgebra relacional
  - $g_1, g_2, \ldots, g_m$ é uma lista de atributos de agrupamento (pode ser vazia)
  - Cada f_i é uma função de agregação
  - Cada a_i é um nome de um atributo

Operação de Agregação - Exemplo

$r =$

A	B	C
a	a	7
a	b	7
b	b	3
b	b	10

$\mathcal{G}_{\mathrm{sum(C)}}(r) =$

sum(C)
27

Operação de Agregação - Exemplo

Relação conta agrupada por nomeBalcao:

conta =

nConta	nomeBalcao	quantia
$A-102$	Coimbra – Central	400
$A-201$	Coimbra – Central	900
$A-217$	Condeixa	750
$A-215$	Condeixa	750
$A-222$	Nelas	700

Operação de Agregação - Exemplo

Relação conta agrupada por nomeBalcao:

conta =

nConta	nomeBalcao	quantia
$A-102$	Coimbra – Central	400
$A-201$	Coimbra – Central	900
$A-217$	Condeixa	750
$A-215$	Condeixa	750
$A-222$	Nelas	700

$$_{\text{nomeBalcao}}\mathcal{G}_{\text{sum(quantia)}}(\text{conta}) =$$

Operação de Agregação - Exemplo

Relação conta agrupada por nomeBalcao:

conta =

nConta	nomeBalcao	quantia
$A-102$	Coimbra – Central	400
$A-201$	Coimbra – Central	900
$A-217$	Condeixa	750
$A-215$	Condeixa	750
$A-222$	Nelas	700

$$_{\text{nomeBalcao}}\mathcal{G}_{\text{sum(quantia)}}(\text{conta}) =$$

=

nomeBalcao	sum(quantia)
Coimbra – Central	1300
Condeixa	1500
Nelas	700

Funções de Agregação (Cont.)

- O resultado da agregação não tem um nome.
  - Pode-se recorrer à operação de renomeação para lhe dar um nome.
  - Por conveniência, permite-se a renomeação de atributos na operação de agregação.

$$\texttt{nomeBalcao}\mathcal{G}_{\rho_{\texttt{balancos}}(\text{sum}(\texttt{quantia}))}(\texttt{conta})$$

- Exemplo de renomeação.
  - Qual a média de idades dos pacientes de cada um dos médicos?

Funções de Agregação (Cont.)

- O resultado da agregação não tem um nome.
  - Pode-se recorrer à operação de renomeação para lhe dar um nome.
  - Por conveniência, permite-se a renomeação de atributos na operação de agregação.

$$_{\texttt{nomeBalcao}}\mathcal{G}_{\rho_{\texttt{balancos}}(\text{sum}(\texttt{quantia}))}(\texttt{conta})$$

- Exemplo de renomeação.
  - Qual a média de idades dos pacientes de cada um dos médicos?
  -

$$_{\texttt{nEmpr}}\mathcal{G}_{\rho_{\texttt{media}}(\text{avg}(\texttt{idade}))}(\texttt{consultas} \bowtie \texttt{pacientes}) =$$

Funções de Agregação (Cont.)

- O resultado da agregação não tem um nome.
  - Pode-se recorrer à operação de renomeação para lhe dar um nome.
  - Por conveniência, permite-se a renomeação de atributos na operação de agregação.

$$_{\texttt{nomeBalcao}}\mathcal{G}_{\rho_{\texttt{balancos}}(\text{sum}(\texttt{quantia}))}(\texttt{conta})$$

- Exemplo de renomeação.
  - Qual a média de idades dos pacientes de cada um dos médicos?
  -

$$_{\texttt{nEmpr}}\mathcal{G}_{\rho_{\texttt{media}}(\text{avg}(\texttt{idade}))}(\texttt{consultas} \bowtie \texttt{pacientes}) =$$

$=$

nEmpr	media
123	23
3243	68
43	4

Mais exemplos

- Quantos fármacos diferentes foram receitados em cada uma das consultas?

Mais exemplos

- Quantos fármacos diferentes foram receitados em cada uma das consultas?

$$\texttt{nConsulta}\mathcal{G}_{\rho_{\texttt{quantos}}(\text{count}(\texttt{codF}))}(\texttt{receitas})$$

Mais exemplos

- Quantos fármacos diferentes foram receitados em cada uma das consultas?

$$\texttt{nConsulta}\mathcal{G}_{\rho_{\texttt{quantos}}(\text{count}(\texttt{codF}))}(\texttt{receitas})$$

- Para cada médico, qual a quantidade média de fármacos receitados por consulta?

Mais exemplos

- Quantos fármacos diferentes foram receitados em cada uma das consultas?

$$_{\texttt{nConsulta}}\mathcal{G}_{\rho_{\texttt{quantos}}(\text{count}(\texttt{codF}))}(\texttt{receitas})$$

- Para cada médico, qual a quantidade média de fármacos receitados por consulta?

$$\text{quantCons} \leftarrow {}_{\text{nconsulta}}\mathcal{G}_{\rho_{\texttt{soma}}(\text{sum}(\texttt{quant}))}(\texttt{receitas})$$

$$_{\text{nEmpr}}\mathcal{G}_{\text{avg}(\texttt{soma})}(\texttt{quantCons} \bowtie \text{consultas})$$

Nota: Nas duas perguntas anteriores não se entra em conta com as consultas sem fármacos receitados (`null`)! Ver-se-á mais à frente como resolver esse caso.

Mais exemplos

- Quantos fármacos diferentes foram receitados em cada uma das consultas?

$$_{\texttt{nConsulta}}\mathcal{G}_{\rho_{\texttt{quantos}}(\text{count}(\texttt{codF}))}(\texttt{receitas})$$

- Para cada médico, qual a quantidade média de fármacos receitados por consulta?

$$\text{quantCons} \leftarrow {}_{\text{nconsulta}}\mathcal{G}_{\rho_{\texttt{soma}}(\text{sum}(\texttt{quant}))}(\texttt{receitas})$$

$$_{\text{nEmpr}}\mathcal{G}_{\text{avg}(\texttt{soma})}(\texttt{quantCons} \bowtie \text{consultas})$$

Nota: Nas duas perguntas anteriores não se entra em conta com as consultas sem fármacos receitados (`null`)! Ver-se-á mais à frente como resolver esse caso.

- Qual a idade do paciente mais velho?

Mais exemplos

- Quantos fármacos diferentes foram receitados em cada uma das consultas?

$$_{\texttt{nConsulta}}\mathcal{G}_{\rho_{\texttt{quantos}}(\text{count}(\texttt{codF}))}(\texttt{receitas})$$

- Para cada médico, qual a quantidade média de fármacos receitados por consulta?

$$\text{quantCons} \leftarrow {}_{\texttt{nconsulta}}\mathcal{G}_{\rho_{\texttt{soma}}(\text{sum}(\texttt{quant}))}(\texttt{receitas})$$

$$_{\texttt{nEmpr}}\mathcal{G}_{\text{avg}(\texttt{soma})}(\texttt{quantCons} \bowtie \text{consultas})$$

Nota: Nas duas perguntas anteriores não se entra em conta com as consultas sem fármacos receitados (`null`)! Ver-se-á mais à frente como resolver esse caso.

- Qual a idade do paciente mais velho?

$$\mathcal{G}_{\max(\texttt{idade})}(\texttt{pacientes})$$

Valores Nulos

- É possível que um tuplo tenha um valor nulo, denotado por `null`, para algum dos seus atributos.
- `null` significa um valor desconhecido ou que não existe.
- O resultado de qualquer expressão aritmética envolvendo um `null` é `null`.
- As funções de agregação ignoram os valores nulos.
  - Decisão arbitrária. Alternativamente, poder-se-ia retornar `null`.
  - Segue-se a semântica da SQL no tratamento de valores nulos.
- Na eliminação de duplicados e agrupamento, um `null` é tratado como um outro valor qualquer, assumindo-se que dois `nulls` são sempre iguais.
  - Decisão arbitrária. Alternativamente, poder-se-ia assumir que cada `null` é diferente de todos os outros
  - Segue-se a semântica da SQL no tratamento de valores nulos.

Valores Nulos

- Comparações com valores nulos devolvem o valor de verdade unknown.
  - Se se usasse o valor de verdade `false` em vez de unknown, então "`not(null<5)`" não seria equivalente a "`null>=5`".

Valores Nulos

- Comparações com valores nulos devolvem o valor de verdade unknown.
  - Se se usasse o valor de verdade `false` em vez de unknown, então "`not(null<5)`" não seria equivalente a "`null>=5`".
- Lógica a três valores com o valor de verdade unknown:
OR	true	false	unknown
true	true	true	true
false	true	false	unknown
unknown	true	unknown	unknown
AND	true	false	unknown
true	true	false	unknown
false	false	false	false
unknown	unknown	false	unknown
NOT(true) = false	NOT(false) = true	NOT(unknown) = unknown
  - Em SQL "`P is unknown`" é verdade se o predicado *P* tem valor de verdade unknown.

Valores Nulos

- Comparações com valores nulos devolvem o valor de verdade unknown.
  - Se se usasse o valor de verdade `false` em vez de unknown, então "`not(null<5)`" não seria equivalente a "`null>=5`".
- Lógica a três valores com o valor de verdade unknown:
  -

OR	true	false	unknown
true	true	true	true
false	true	false	unknown
unknown	true	unknown	unknown

  -

AND	true	false	unknown
true	true	false	unknown
false	false	false	false
unknown	unknown	false	unknown

  -

NOT(true) = false	NOT(false) = true	NOT(unknown) = unknown

  - Em SQL "`P is unknown`" é verdade se o predicado *P* tem valor de verdade unknown.
- Resultado do predicado de selecção é tratado como `false` se tiver valor de verdade unknown.

Junção Externa (ou exterior)

Uma extensão da operação de junção que evita a perda de informação.

- Calcula a junção e depois adiciona ao resultado os tuplos de uma relação que não estão relacionados com a outra relação na junção.
- Utiliza valores nulos (`null`).

Valores nulos (`null`) são valores cujo valor é desconhecido ou que não existe
Simplificadamente, todas as comparações com `null` são falsas por definição.

Junção Externa - Exemplo

emprestimo =

nEmprestimo	nomeBalcao	quantia
L – 170	Coimbra – central	3000
L – 230	Condeixa	4000
L – 260	Nelas	1700

temEmprestimo =

nomeCliente	nEmprestimo
Gomes	L – 170
Silva	L – 230
Costa	L – 155

Junção Externa – Exemplo

- Junção interna

  $\texttt{emprestimo} \bowtie \texttt{temEmprestimo} =$

$=$

nEmprestimo	nomeBalcao	quantia	nomeCliente
L – 170	Coimbra – central	3000	Gomes
L – 230	Condeixa	4000	Silva

Junção Externa – Exemplo

- Junção interna

  $\texttt{emprestimo} \bowtie \texttt{temEmprestimo} =$

$=$

nEmprestimo	nomeBalcao	quantia	nomeCliente
L – 170	Coimbra – central	3000	Gomes
L – 230	Condeixa	4000	Silva

- Junção externa esquerda

  $\texttt{emprestimo} \mathbin{⟕} \texttt{temEmprestimo} =$

$=$

nEmprestimo	nomeBalcao	quantia	nomeCliente
L – 170	Coimbra – central	3000	Gomes
L – 230	Condeixa	4000	Silva
L – 260	Nelas	1700	null

Junção Externa - Exemplo

- Junção externa direita

  $\texttt{emprestimo} \rtimes \texttt{temEmprestimo} =$

$=$

nEmprestimo	nomeBalcao	quantia	nomeCliente
L − 170	Coimbra − central	3000	Gomes
L − 230	Condeixa	4000	Silva
L − 155	null	null	Costa

Junção Externa - Exemplo

- Junção externa direita

  emprestimo $\bowtie\!\!\!\subset$ temEmprestimo $=$

$=$

nEmprestimo	nomeBalcao	quantia	nomeCliente
L − 170	Coimbra − central	3000	Gomes
L − 230	Condeixa	4000	Silva
L − 155	null	null	Costa

- Junção externa

  emprestimo $\supset\!\!\!\bowtie\!\!\!\subset$ temEmprestimo $=$

$=$

nEmprestimo	nomeBalcao	quantia	nomeCliente
L − 170	Coimbra − central	3000	Gomes
L − 230	Condeixa	4000	Silva
L − 260	Nelas	1700	null
L − 155	null	null	Costa

Consultas de exemplo

- Quais os fármacos que nunca foram receitados?

Consultas de exemplo

- Quais os fármacos que nunca foram receitados?

$$\Pi_{\texttt{nomeF}}(\sigma_{\texttt{nConsulta=null}}(\texttt{farmacos} \rightarrowtriangle \texttt{receitas}))$$

Consultas de exemplo

- Quais os fármacos que nunca foram receitados?

$$\Pi_{\texttt{nomeF}}(\sigma_{\texttt{nConsulta=null}}(\texttt{farmacos} \rtimes \texttt{receitas}))$$

- Quais as consultas em que não foi receitado qualquer fármaco?

Consultas de exemplo

- Quais os fármacos que nunca foram receitados?

$$\Pi_{\texttt{nomeF}}(\sigma_{\texttt{nConsulta=null}}(\texttt{farmacos} \rtimes \texttt{receitas}))$$

- Quais as consultas em que não foi receitado qualquer fármaco?

$$\Pi_{\texttt{nConsulta}}(\sigma_{\texttt{nCodF=null}}(\texttt{consultas} \rtimes \texttt{receitas}))$$

Consultas de exemplo

- Quais os fármacos que nunca foram receitados?

$$\Pi_{\texttt{nomeF}}(\sigma_{\texttt{nConsulta=null}}(\texttt{farmacos} \rtimes \texttt{receitas}))$$

- Quais as consultas em que não foi receitado qualquer fármaco?

$$\Pi_{\texttt{nConsulta}}(\sigma_{\texttt{nCodF=null}}(\texttt{consultas} \rtimes \texttt{receitas}))$$

- Quantos fármacos diferentes foram receitados em cada uma das consultas?

Consultas de exemplo

- Quais os fármacos que nunca foram receitados?

$$\Pi_{\texttt{nomeF}}(\sigma_{\texttt{nConsulta=null}}(\texttt{farmacos} \rtimes \texttt{receitas}))$$

- Quais as consultas em que não foi receitado qualquer fármaco?

$$\Pi_{\texttt{nConsulta}}(\sigma_{\texttt{nCodF=null}}(\texttt{consultas} \rtimes \texttt{receitas}))$$

- Quantos fármacos diferentes foram receitados em cada uma das consultas?

$$_{\texttt{nConsulta}}\mathcal{G}_{\text{count}(\texttt{codF})}(\texttt{consultas} \rtimes \texttt{receitas})$$

Consultas de exemplo

- Quais os fármacos que nunca foram receitados?

$$\Pi_{\texttt{nomeF}}(\sigma_{\texttt{nConsulta=null}}(\texttt{farmacos} \rtimes \texttt{receitas}))$$

- Quais as consultas em que não foi receitado qualquer fármaco?

$$\Pi_{\texttt{nConsulta}}(\sigma_{\texttt{nCodF=null}}(\texttt{consultas} \rtimes \texttt{receitas}))$$

- Quantos fármacos diferentes foram receitados em cada uma das consultas?

$${}_{\texttt{nConsulta}}\mathcal{G}_{\text{count}(\texttt{codF})}(\texttt{consultas} \rtimes \texttt{receitas})$$

- Para cada médico, qual a quantidade média de fármacos receitados por consulta?

Consultas de exemplo

- Quais os fármacos que nunca foram receitados?

$$\Pi_{\texttt{nomeF}}(\sigma_{\texttt{nConsulta=null}}(\texttt{farmacos} \rtimes \texttt{receitas}))$$

- Quais as consultas em que não foi receitado qualquer fármaco?

$$\Pi_{\texttt{nConsulta}}(\sigma_{\texttt{nCodF=null}}(\texttt{consultas} \rtimes \texttt{receitas}))$$

- Quantos fármacos diferentes foram receitados em cada uma das consultas?

$${}_{\texttt{nConsulta}}\mathcal{G}_{\text{count}(\texttt{codF})}(\texttt{consultas} \rtimes \texttt{receitas})$$

- Para cada médico, qual a quantidade média de fármacos receitados por consulta?

$$\texttt{quantCons} \leftarrow {}_{\texttt{nConsulta}}\mathcal{G}_{\rho_{\texttt{soma}}\text{sum}(\texttt{quant})}(\texttt{consultas} \rtimes \texttt{receitas})$$

$${}_{\texttt{nEmpr}}\mathcal{G}_{\text{avg}(\texttt{soma})}(\texttt{quantCons} \bowtie \texttt{consultas})$$

Modificação da Base de Dados

- O conteúdo da base de dados pode ser modificado através das seguintes operações:
  - Remoção
  - Inserção
  - Actualização

- Todas estas operação são expressas por intermédio do operador de atribuição.

Remoção

- Uma operação de remoção é expressa de uma maneira semelhante a uma consulta, excepto que os tuplos seleccionados são removidos da base de dados.
- Só se podem remover tuplos integralmente; não se podem apagar valores de determinados atributos.
- Uma remoção é expressa em álgebra relacional por:

$$r \leftarrow r - E$$

em que r é uma relação e E é uma expressão da álgebra relacional.

Exemplos de Remoção

- Apagar todas as contas na agência de Cantanhede.

$$\text{conta} \leftarrow \text{conta} - \sigma_{\text{nomeBalcao}='\textit{Cantanhede}'}(\text{conta})$$

- Apagar todos os registos de empréstimos de montante entre 0 e 50€

$$\text{emprestimo} \leftarrow \text{emprestimo} - \sigma_{\text{quantia}\geq 0 \ \wedge \ \text{quantia}\leq 50}(\text{emprestimo})$$

- Apagar todas as contas de balcões localizados em Nelas.

$$\begin{aligned} r_1 &\leftarrow \sigma_{\text{cidadeBalcao}='\text{Nelas}'}(\text{conta} \bowtie \text{balcao}) \\ r_2 &\leftarrow \Pi_{\text{cidadeBalcao,nConta,balanco}}(r_1) \\ r_3 &\leftarrow \Pi_{\text{nomeCliente,nConta}}(r_2 \bowtie \text{temConta}) \\ \text{conta} &\leftarrow \text{conta} - r_2 \\ \text{temConta} &\leftarrow \text{temConta} - r_3 \end{aligned}$$

Exemplos de Remoção (cont)

- Apagar toda a informação relativa a consultas anteriores a 2000:

$$
\begin{aligned}
r1 &\leftarrow \sigma_{\text{data}<01-01-2000}(\text{consultas}) \\
r2 &\leftarrow \Pi_{\text{codF,nConsulta,quant}}(\text{receitas} \bowtie r1) \\
\text{consultas} &\leftarrow \text{consultas} - r1 \\
\text{receitas} &\leftarrow \text{receitas} - r2
\end{aligned}
$$

Inserção

- Para inserir informação numa relação podemos:
  - especificar um tuplo a ser inserido;
  - escrever uma consulta cujo resultado é um conjunto de tuplos a inserir.
- Na álgebra relacional, uma inserção é expressa por:

$$r \leftarrow r \cup E$$

  em que r é uma relação e E é uma expressão da álgebra relacional.
- A inserção de um único tuplo é efectuada quando a expressão E é uma relação constante contendo esse tuplo.

Exemplos de Inserção

- Inserir informação na base de dados especificando que o cliente Silva tem 1200€ na conta A-973 na agência de Cantanhede.

$$\begin{aligned} \text{conta} &\leftarrow \text{conta} \cup \{(\text{A} - 973, '\text{Cantanhede}', 1200)\} \\ \text{temConta} &\leftarrow \text{temConta} \cup \{('\text{Silva}', \text{A} - 973)\} \end{aligned}$$

- Dar um bónus a todos os mutuários na agência de Cantanhede: uma conta de poupança de 200€. O número do empréstimo é utilizado para número da conta de poupança.

$$\begin{aligned} r1 &\leftarrow (\sigma_{\text{nomeBalcao}=\text{“Cantanhede”}}(\text{temEmprestimo} \bowtie \text{emprestimo})) \\ \text{conta} &\leftarrow \text{conta} \cup \Pi_{\text{nomeBalcao,nConta,200}}(r1) \\ \text{temConta} &\leftarrow \text{temConta} \cup \Pi_{\text{nomeCliente,nEmprestimo}}(r1) \end{aligned}$$

Exemplos de Inserção (cont.)

- Inserir informação na base de dados especificando que um novo paciente, com BI nº 1111 e nome Paulo, teve uma consulta (nº101) no dia 30-09-2003 com o médico João (assumindo que só há um médico com esse nome).

$$
\begin{aligned}
\text{pacientes} &\leftarrow \textit{pacientes} \cup \{(1111, \text{'Paulo'}, \text{null}, \text{null}, \text{null})\} \\
\text{consultas} &\leftarrow \text{consultas} \cup \\
&\quad \Pi_{101,30-09-2009,1111,\text{nEmpr}}(\sigma_{\text{nomeM}=\text{'João'}}(\text{medicos}))
\end{aligned}
$$

Actualização

- Um mecanismo para alterar um valor de um tuplo sem alterar todos os valores do tuplo.
- Recorre-se ao operador de projecção generalizada para efectuar este tipo de tarefa

$$r \leftarrow \Pi_{F_1,F_2,\ldots,F_n}(r)$$

- Cada F_i, ou é o i-ésimo atributo de r, se o i-ésimo atributo não for alterado, ou
- Uma expressão F_i, envolvendo apenas constantes e atributos de r, que permite calcular o novo valor do atributo.

Exemplos de Actualizações

- Pague juros de 5% em todas as contas

$$\text{conta} \leftarrow \Pi_{\text{nConta,nomeBalcao,balanco} \times 1,05}(\text{conta})$$

- Pague 6% de juros em todas as contas com saldo superior a 10.000€ e 5% às restantes contas.

$$\begin{aligned} \text{conta} \leftarrow\ & \Pi_{\text{nConta,nomeBalcao,balanco} \times 1,06}(\sigma_{\text{balanco} > 10000}(\text{conta})) \\ & \cup \Pi_{\text{nConta,nomeBalcao,balanco} \times 1,05}(\sigma_{\text{balanco} \leq 10000}(\text{conta})) \end{aligned}$$

SQL

- Linguagem de Definição dos Dados
  - Tipos em SQL
  - Manipular Tabelas (relações)
  - Integridade e Segurança
- Linguagem de Manipulação dos Dados
  - Estrutura básica
  - Operações com conjuntos
  - Funções de agregação
  - Valores nulos
  - Junções
  - Sub-consultas embutidas
  - Relações derivadas
  - Vistas
  - Modificação da Base de Dados
  - União, Diferença, Divisão em SQL
- SQL Embutido

Linguagem de Definição de Dados (DDL)

Linguagem para especificar a informação acerca de cada relação, incluindo:

Linguagem de Definição de Dados (DDL)

Linguagem para especificar a informação acerca de cada relação, incluindo:

- O esquema de cada relação.

Linguagem de Definição de Dados (DDL)

Linguagem para especificar a informação acerca de cada relação, incluindo:

- O esquema de cada relação.
- O domínio de valores associados com cada atributo.

Linguagem de Definição de Dados (DDL)

Linguagem para especificar a informação acerca de cada relação, incluindo:

- O esquema de cada relação.
- O domínio de valores associados com cada atributo.
- Restrições de integridade.

Linguagem de Definição de Dados (DDL)

Linguagem para especificar a informação acerca de cada relação, incluindo:

- O esquema de cada relação.
- O domínio de valores associados com cada atributo.
- Restrições de integridade.
- O conjunto de índices a manter para cada relação.

Linguagem de Definição de Dados (DDL)

Linguagem para especificar a informação acerca de cada relação, incluindo:

- O esquema de cada relação.
- O domínio de valores associados com cada atributo.
- Restrições de integridade.
- O conjunto de índices a manter para cada relação.
- Informação de segurança e autorização para cada relação.

Linguagem de Definição de Dados (DDL)

Linguagem para especificar a informação acerca de cada relação, incluindo:

- O esquema de cada relação.
- O domínio de valores associados com cada atributo.
- Restrições de integridade.
- O conjunto de índices a manter para cada relação.
- Informação de segurança e autorização para cada relação.
- As estruturas de armazenamento físico em disco de cada relação.

Tipos em SQL

- `char(n)`: cadeia de caracteres de comprimento fixo *n*.
- `varchar(n)`: cadeia de caracteres de comprimento variável, com o máximo *n* especificado pelo utilizador.
- `int`: inteiro (um subconjunto finito dos inteiros, dependente da máquina).
- `smallint`: inteiro pequeno (um subconjunto do tipo `int`).
- `numeric(p,d)`: número de **vírgula fixa**, com de *p* dígitos no total e com *d* casas decimais.
- `real, double precision`: Números de **vírgula flutuante**, com precisão dependente da máquina.
- `float(n)`: número de vírgula flutuante, com um mínimo de precisão de *n* dígitos.

Os valores «nulos» (`NULL`) são permitidos em todos os tipos de dados.
A declaração de um atributo como `NOT NULL` proíbe os valores nulos para esse atributo.

Tipos Data/Tempo em SQL (cont.)

- `date`: datas, contendo um ano com (4 dígitos), mês e dia, («2001-7-27»);
- `time`: tempo (diário), em horas, minutos e segundos, («09:00:30.75»);
- `timestamp`: data mais hora, («2001-7-27 09:00:30.75»).
- `Interval`: período de tempo.

A subtracção de dois valores de date/time/timestamp devolve um intervalo.
Os valores de intervalos podem ser adicionados a valores de date/time/timestamp.
Pode-se extrair campos do valor date/time/timestamp.

Tipos em MySQL

```
TINYINT[(tamanho)] [UNSIGNED] [ZEROFILL]
SMALLINT[(tamanho)] [UNSIGNED] [ZEROFILL]
MEDIUMINT[(tamanho)] [UNSIGNED] [ZEROFILL]
INT[(tamanho)] [UNSIGNED] [ZEROFILL]
INTEGER[(tamanho)] [UNSIGNED] [ZEROFILL]
BIGINT[(tamanho)] [UNSIGNED] [ZEROFILL]
REAL[(tamanho,decimais)] [UNSIGNED] [ZEROFILL]
DOUBLE[(tamanho,decimais)] [UNSIGNED] [ZEROFILL]
FLOAT[(tamanho,decimais)] [UNSIGNED] [ZEROFILL]
DECIMAL(tamanho,decimais) [UNSIGNED] [ZEROFILL]
NUMERIC(tamanho,decimais) [UNSIGNED] [ZEROFILL]
CHAR(tamanho) [BINARY | ASCII | UNICODE]
VARCHAR(tamanho) [BINARY]
DATE
TIME
TIMESTAMP
DATETIME
TINYBLOB
BLOB
MEDIUMBLOB
LONGBLOB
TINYTEXT
TEXT
MEDIUMTEXT
LONGTEXT
ENUM(value1,value2,value3,...)
SET(value1,value2,value3,...)
```

Instrução Create Table

- Uma tabela SQL é definida recorrendo ao comando `CREATE TABLE`:

$$\texttt{CREATE TABLE}\ \ r\ \ (A_1\ D_1, A_2\ D_2, ..., A_n\ D_n,$$
$$\text{restrição\_de\_integridade } 1,$$
$$\ldots,$$
$$\text{restrição\_de\_integridade } k)$$

¡

  - r é o nome da relação;
  - A_i é o nome de um atributo no esquema de relação r;
  - D_i é o tipo de dados dos valores no domínio do atributo A_i.

- Exemplo:

```
CREATE TABLE Balcao
    (balcaoNome char(15) not null,
     balcaoCidade char(30),
     depositos integer)
```

Restrições de Integridade

- not null
- primary key (A_1, ..., A_n)
- unique (A_1, ..., A_n)
- check (P), em que P é um predicado.

Exemplo:

```
CREATE TABLE Balcao
    (balcaoNome char(15) NOT NULL,
     balcaoCidade char(30),
     depositos integer,
     PRIMARY KEY (balcaoNome),
     CHECK (depositos >= 0))
```

Instrução Drop Table

O comando `DROP TABLE` remove da base de dados toda a informação sobre a relação (e não apenas os dados).

```
DROP TABLE Emprestimo
```

Caso hajam chaves externas deve-se utilizar a declaração `cascade constraints`.

```
DROP TABLE Emprestimo CASCADE CONSTRAINTS
```

Instrução Alter Table

O comando `ALTER TABLE` é utilizado para modificar o esquema, ou as restrições sobre relações já existente.

- Para adicionar novos atributos:

  ```
  ALTER TABLE r ADD A D
  ```

  em que A é o nome do atributo a adicionar à relação r e D o domínio de A.
  Todos os tuplos existentes ficam com `NULL` no novo atributo.
- Para eliminar atributos de uma relação

  ```
  ALTER TABLE r DROP A
  ```

  em que A é o nome de um atributo na relação r.

Instrução Alter Table (cont.)

- Para adicionar novas restrições:
  `ALTER TABLE r ADD CONSTRAINT N R`
  em que *N* é um nome dado à nova restrição e *R* define a restrição.
  Por exemplo:

```
ALTER TABLE Conta ADD CONSTRAINT saldoPos CHECK (balanco > 0)
```

- Para remover restrições (definidas previamente com um nome):
  `ALTER TABLE r DROP CONSTRAINT N`
  por exemplo:

```
ALTER TABLE Conta DROP CONSTRAINT saldoPos
```

Integridade e Segurança

- Restrições ao Domínio
- Integridade Referencial
- Asserções
- Segurança e Autorizações

Restrições ao Domínio

As restrições de integridade impõem-se para garantir que os dados fiquem protegidos contra «estragos» acidentais. Devem asseguram que da actualização dos dados não resulta a perda da consistência.

Restrições ao domínio são a forma mais elementar de restrição de integridade.

- Testam condições sobre valores a introduzir em atributos. Fazem-no, restringindo o domínio do atributo em causa.
- Fixam-se no momento em que se define a tabela.
- A forma mais comum de restrição ao domínio é a proibição do valor `NULL` (`NOT NULL` depois do atributo).
- Podem-se impor outras restrições usando, depois dum nome de atributo A, `CHECK(condição(`A_1`,...,` A_n`))` onde `condição(...)` denota uma condição imposta sobre os atributos $A_1,\ldots,A_n$.

Exemplos de Restrições ao Domínio

```
create table Alunos (
      numAluno number(6) not null,
      nome varchar(30) not null,
      local varchar(25),
      dataNsc date not null,
      sexo char(1) not null check (sexo in ('F','M')),
      codCurso number(3) not null);
```

```
create table Produtos (
      idProduto number(6) not null,
      nome varchar2(30) not null,
      iva number(2) not null check (iva in (5,12,21)));
```

```
create table Balcao (
      balcaoNome char(15),
      balcaoCidade char(30),
      depositos  number check (depositos >= 0));
```

Integridade Referencial (IR)

Chaves Externas (também designadas por «estrangeiras»)

- Garante que um valor que ocorre numa relação para um certo conjunto de atributos também ocorre num outro conjunto de atributos de outra relação.
  Exemplo: Se «Coimbra-central» é o nome de uma agência que ocorre num dos tuplos da relação `Conta`, então existe um tuplo na relação `Balcao` para o balcão «Coimbra-central».

- Definição Formal:
  Sejam $r_1(R_1)$ e $r_2(R_2)$ duas relações com chaves primárias K_1 e K_2 respectivamente.
  O subconjunto α de R_2 é uma chave externa referindo K_1 na relação r_1, se para todo t_2 em r_2 existe um tuplo t_1 em r_1 tal que $t_1[K_1] = t_2[\alpha]$.

Aquando da escrita do modelo relacional destigue-se as chaves externas com sublinhado a tracejado.

IR – Modificação da Base de Dados

Os testes abaixo devem ser efectuados de modo a preservar-se a seguinte restrição de integridade referencial:

$$\Pi_{\alpha}(r2) \subseteq \Pi_{K}(r1)$$

Inserção: se um tuplo t_2 é inserido em r_2, o sistema tem de garantir que existe um tuplo t_1 em r_1 tal que $t_1[K] = t_2[\alpha]$.
Ou seja

$$t_2[\alpha] \in \Pi_{K}(r1)$$

Remoção: se um tuplo t_1 é removido de r_1, o sistema deve calcular o conjunto de tuplos em r_2 que referenciam t_1:

$$\sigma_{\alpha=t_1[K]}(r_2)$$

Se este conjunto não é vazio, ou o comando de remoção é rejeitado, ou os tuplos que referenciam t_1 devem ser eles próprios removidos (remoções em cascata são possíveis).

IR – Modificação da Base de Dados (Cont.)

Actualização. Existem duas situações:
Se um tuplo t_2 é actualizado na relação r_2 em que é modificado o valor da chave externa α, então é efectuado um teste similar ao da inserção. Seja $t_{2'}$ o novo valor do tuplo t_2. O sistema deve garantir que

$$t_{2'}[\alpha] \in \Pi_K(r1)$$

Se o tuplo t_1 é actualizado em r_1, e a operação altera o valor da chave primária (K), então é efectuado um teste semelhante ao da remoção. O sistema deve calcular

$$\sigma_{\alpha=t_1[K]}(r_2)$$

usando o valor anterior de t_1 (o valor antes da actualização). Se o conjunto é não vazio, a actualização pode ser rejeitado, ou a actualização pode ser propagada em cascata, ou os tuplos podem ser removidos.

Integridade Referencial em SQL

As chaves **primárias**, **candidatas** e **externas** podem ser especificadas na instrução SQL `create table`:

Integridade Referencial em SQL

As chaves **primárias**, **candidatas** e **externas** podem ser especificadas na instrução SQL `create table`:

- A cláusula `primary key` inclui a lista dos atributos que formam a chave primária.

Integridade Referencial em SQL

As chaves **primárias**, **candidatas** e **externas** podem ser especificadas na instrução SQL `create table`:

- A cláusula `primary key` inclui a lista dos atributos que formam a chave primária.
- A cláusula `unique key` inclui a lista dos atributos que formam a chave candidata.

Integridade Referencial em SQL

As chaves **primárias**, **candidatas** e **externas** podem ser especificadas na instrução SQL `create table`:

- A cláusula `primary key` inclui a lista dos atributos que formam a chave primária.
- A cláusula `unique key` inclui a lista dos atributos que formam a chave candidata.
- A cláusula `foreign key` inclui a lista de atributos que constituem a chave externa assim como o nome da relação referenciada pela chave externa.

Integridade Referencial em SQL — Exemplos

```
CREATE TABLE Cursos (
        codCurso number(3) NOT NULL,
        nome varchar(35) NOT NULL,
        PRIMARY KEY (codCurso));
```

```
CREATE TABLE Cadeiras (
        codCadeira number(3) NOT NULL,...,
        PRIMARY KEY (codCadeira));
```

```
CREATE TABLE CursoCadeira (
        codCurso number(3) NOT NULL,
        codCadeira number(3) NOT NULL,
        semestre number(2) NOT NULL,
        PRIMARY KEY (codCurso, codCadeira),
        FOREIGN KEY (codCurso) REFERENCES Cursos,
        FOREIGN KEY (codCadeira) REFERENCES Cadeiras);
```

Acções em Cascata em SQL

```
CREATE TABLE Conta
    ...
    FOREIGN KEY (balcaoNome) REFERENCES Balcao
                                ON DELETE CASCADE
                                ON UPDATE CASCADE
    ...)
```

Acções em Cascata em SQL

```
CREATE TABLE Conta
    ...
    FOREIGN KEY (balcaoNome) REFERENCES Balcao
                                ON DELETE CASCADE
                                ON UPDATE CASCADE
    ...)
```

- Com as cláusulas `ON DELETE CASCADE`, se a remoção de um tuplo na relação `Balcao` resulta na violação da restrição da integridade referencial, a remoção propaga-se em «cascata» para a relação `Conta`, removendo o tuplo que referia a agência que tinha sido eliminada.

Acções em Cascata em SQL

```
CREATE TABLE Conta
    ...
    FOREIGN KEY (balcaoNome) REFERENCES Balcao
                                ON DELETE CASCADE
                                ON UPDATE CASCADE
    ...)
```

- Com as cláusulas `ON DELETE CASCADE`, se a remoção de um tuplo na relação `Balcao` resulta na violação da restrição da integridade referencial, a remoção propaga-se em «cascata» para a relação `Conta`, removendo o tuplo que referia a agência que tinha sido eliminada.
- Actualizações em cascata são semelhantes.

Acções em Cascata em SQL (cont.)

- Se existe uma cadeia de dependências de chaves externas através de várias relações, com um `ON DELETE CASCADE` especificado em cada dependência, uma remoção ou actualização num dos extremos pode-se propagar através de toda a cadeia.

Acções em Cascata em SQL (cont.)

- Se existe uma cadeia de dependências de chaves externas através de várias relações, com um `ON DELETE CASCADE` especificado em cada dependência, uma remoção ou actualização num dos extremos pode-se propagar através de toda a cadeia.
- Se uma remoção ou actualização em cascata origina uma violação de uma restrição que não pode ser tratada por uma outra operação em cascata, o sistema aborta a transacção. Como resultado, todas as alterações provocadas pela transacção e respectivas acções em cascata serão anuladas.

Acções em Cascata em SQL (cont.)

- Se existe uma cadeia de dependências de chaves externas através de várias relações, com um `ON DELETE CASCADE` especificado em cada dependência, uma remoção ou actualização num dos extremos pode-se propagar através de toda a cadeia.
- Se uma remoção ou actualização em cascata origina uma violação de uma restrição que não pode ser tratada por uma outra operação em cascata, o sistema aborta a transacção. Como resultado, todas as alterações provocadas pela transacção e respectivas acções em cascata serão anuladas.
- A integridade referencial é verificada apenas no final da transacção
  - Passos intermédios podem violar a integridade referencial desde que passos posteriores a reponham.
  - Caso contrário seria impossível criar alguns estados da base de dados, por exemplos, inserir dois tuplos cujas chaves externas apontam um para o outro.

Acções em Cascata em SQL (cont.)

- Alternativas às operações em cascata:
  - ON DELETE SET NULL
  - ON DELETE SET DEFAULT

- Valores nulos em atributos de chaves externas complicam a semântica de integridade referencial da SQL, devendo-se evitar recorrendo a `NOT NULL`.

- Se algum atributo de uma chave externa é nulo, o tuplo satisfaz automaticamente a restrição de integridade referencial!

Segurança

Segurança — ao contrário das restrições de integridade, que pretendiam proteger a base de dados contra estragos acidentais, a segurança preocupa-se com proteger a base de dados de estragos propositados.

- A nível do sistema operativo
- A nível da rede
- A nível físico
- A nível humano
- A nível da base de dados

Mecanismos de autenticação e autorização para permitir acessos selectivos de (certos) utilizadores a (certas) partes dos dados

Autorizações

Diferentes formas de autorização, em dados da bases de dados:

Autorizações

Diferentes formas de autorização, em dados da bases de dados:

- Autorização de leitura - permite ler, mas não modificar dados.

Autorizações

Diferentes formas de autorização, em dados da bases de dados:

- Autorização de leitura - permite ler, mas não modificar dados.
- Autorização de inserção - permite inserir novos tuplos, mas não modificar tuplos existentes.

Autorizações

Diferentes formas de autorização, em dados da bases de dados:

- Autorização de leitura - permite ler, mas não modificar dados.
- Autorização de inserção - permite inserir novos tuplos, mas não modificar tuplos existentes.
- Autorização de modificação - permite modificar tuplos, mas não apagá-los.

Autorizações

Diferentes formas de autorização, em dados da bases de dados:

- Autorização de leitura - permite ler, mas não modificar dados.
- Autorização de inserção - permite inserir novos tuplos, mas não modificar tuplos existentes.
- Autorização de modificação - permite modificar tuplos, mas não apagá-los.
- Autorização de apagar («delete») - permite apagar tuplos

Autorizações (Cont.)

Diferentes formas de autorização, para alterar esquemas:

Autorizações (Cont.)

Diferentes formas de autorização, para alterar esquemas:

- Autorização de índex - permite criar e apagar ficheiros de indexação.

Autorizações (Cont.)

Diferentes formas de autorização, para alterar esquemas:

- Autorização de índex - permite criar e apagar ficheiros de indexação.
- Autorização de recursos - permite criar novas relações.

Autorizações (Cont.)

Diferentes formas de autorização, para alterar esquemas:

- Autorização de índex - permite criar e apagar ficheiros de indexação.
- Autorização de recursos - permite criar novas relações.
- Autorização de alteração - permite criar e apagar atributos duma relação.

Autorizações (Cont.)

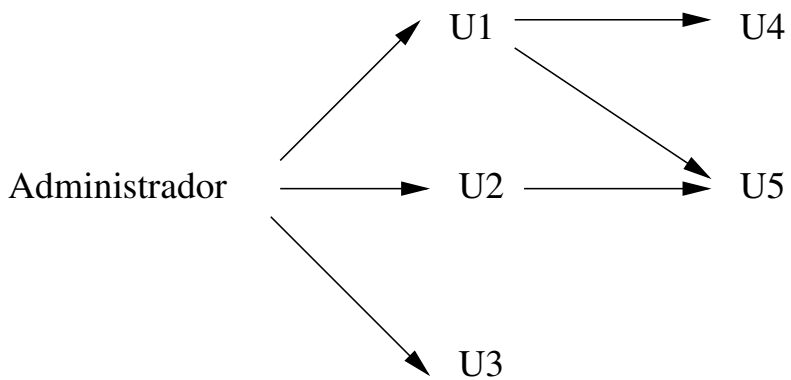
Diferentes formas de autorização, para alterar esquemas:

- Autorização de índex - permite criar e apagar ficheiros de indexação.
- Autorização de recursos - permite criar novas relações.
- Autorização de alteração - permite criar e apagar atributos duma relação.
- Autorização de remoção («drop») - permite apagar relações.

Atribuição de Privilégios

- A passagem de privilégios de um utilizador para outro pode ser representado por um grafo de autorizações.
- Os nós do grafo são utilizadores.
- A raiz é o administrador da base de dados.

Por exemplo: no grafo abaixo um arco $U_i \rightarrow U_j$ indica que o utilizador U_i atribuiu ao utilizador U_j um determinado privilégio.



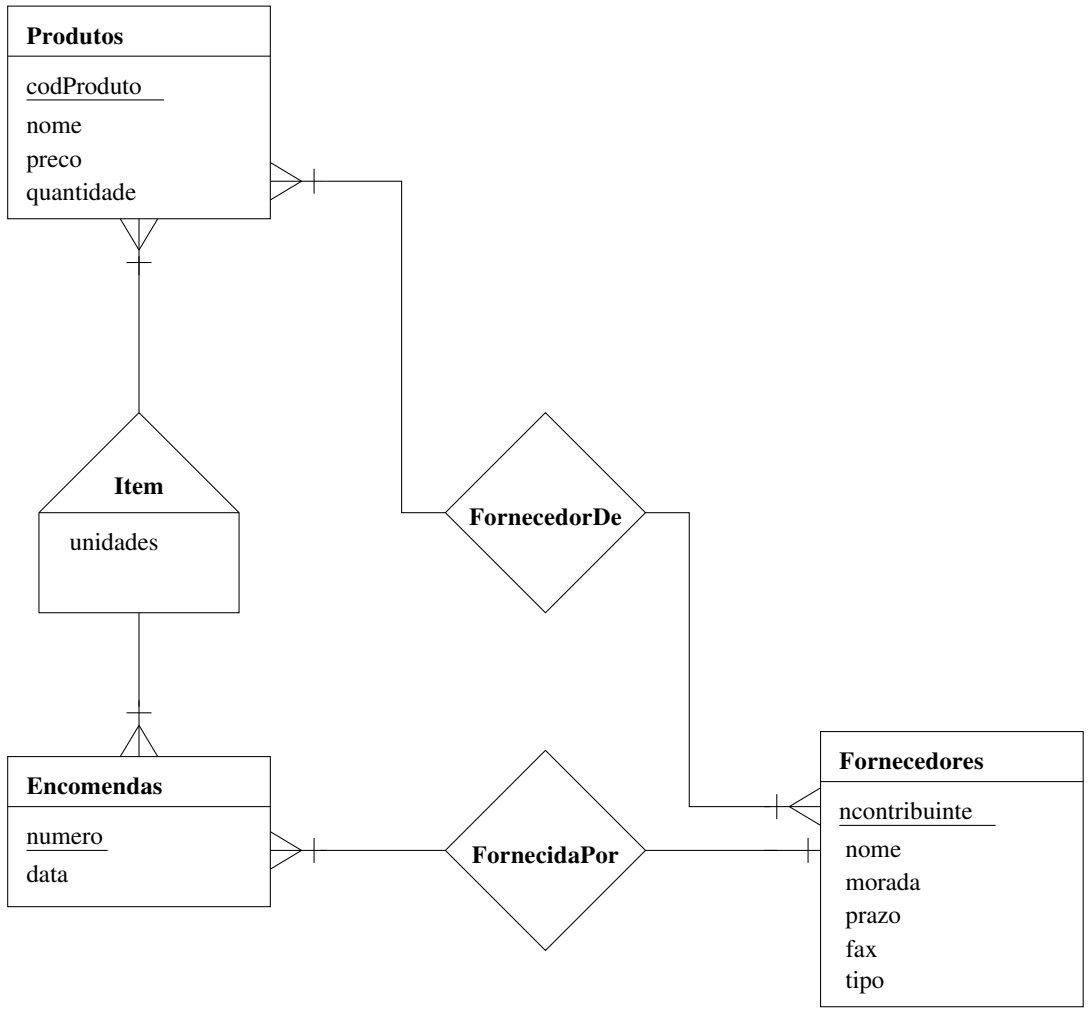
- Todos os arcos têm que fazer parte de algum caminho com origem no administrador.
- Um dado utilizador, que não o administrador, só pode atribuir privilégios a outro utilizador se para isso estiver mandatado, isto é, se tiver o privilégio de atribuir privilégios.
- O comando `GRANT` é o comando usado para atribuir privilégios.

DML — SQL

Linguagem a que o utilizador recorre para obter informação a partir da base de dados.

- Estrutura básica
- Operações com conjuntos
- Funções de agregação
- Valores nulos
- Junções
- Sub-consultas embutidas
- Relações derivadas
- Vistas
- Modificação da Base de Dados
- União, Diferença, Divisão em SQL

Esquema Utilizado em Exemplos



Pesquisa — Estrutura Básica

- SQL é baseada em operações de conjuntos e de álgebra relacional com algumas modificações e extensões.

Pesquisa — Estrutura Básica

- SQL é baseada em operações de conjuntos e de álgebra relacional com algumas modificações e extensões.
- Uma consulta SQL básica tem a forma:

$$\begin{aligned} \texttt{SELECT} \quad & A_1, A_2, \ldots, A_n \\ \texttt{FROM} \quad & r_1, r_2, \ldots, r_m \\ \texttt{WHERE} \quad & P \end{aligned}$$

Pesquisa — Estrutura Básica

- SQL é baseada em operações de conjuntos e de álgebra relacional com algumas modificações e extensões.
- Uma consulta SQL básica tem a forma:

$$\begin{aligned} \texttt{SELECT} \quad & A_1, A_2, \ldots, A_n \\ \texttt{FROM} \quad & r_1, r_2, \ldots, r_m \\ \texttt{WHERE} \quad & P \end{aligned}$$

  - os A_is representam atributos;
  - os r_is representam relações;
  - P é um predicado, nos atributos dos r_is.

Pesquisa — Estrutura Básica

- SQL é baseada em operações de conjuntos e de álgebra relacional com algumas modificações e extensões.
- Uma consulta SQL básica tem a forma:

$$\begin{aligned} \texttt{SELECT} \quad & A_1, A_2, \ldots, A_n \\ \texttt{FROM} \quad & r_1, r_2, \ldots, r_m \\ \texttt{WHERE} \quad & P \end{aligned}$$

  - os A_is representam atributos;
  - os r_is representam relações;
  - P é um predicado, nos atributos dos r_is.
- A leitura desta instrução é a seguinte:
  Seleccionar (`SELECT`) os atributos $A_1, A_2, \ldots, A_n$ de entre (`FROM`) os atributos do produto cartesiano $r_1 \times r_2 \times \cdots \times r_m$, sujeitos (`WHERE`) ao predicado P.

Pesquisa — Estrutura Básica

- SQL é baseada em operações de conjuntos e de álgebra relacional com algumas modificações e extensões.
- Uma consulta SQL básica tem a forma:

$$\begin{aligned} \texttt{SELECT} \quad & A_1, A_2, \ldots, A_n \\ \texttt{FROM} \quad & r_1, r_2, \ldots, r_m \\ \texttt{WHERE} \quad & P \end{aligned}$$

  - os A_is representam atributos;
  - os r_is representam relações;
  - P é um predicado, nos atributos dos r_is.
- A leitura desta instrução é a seguinte:
  Seleccionar (`SELECT`) os atributos $A_1, A_2, \ldots, A_n$ de entre (`FROM`) os atributos do produto cartesiano $r_1 \times r_2 \times \cdots \times r_m$, sujeitos (`WHERE`) ao predicado P.
- Corresponde à expressão da álgebra relacional:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \cdots \times r_m))$$

A Componente SELECT

- É utilizado para listar os atributos pretendidos no resultado da consulta. Por exemplo, listar o nome e o endereço dos Fornecedores:

```
mysql> SELECT nome,morada FROM Fornecedores;
+---------------+--------------------------+
| nome          | morada                   |
+---------------+--------------------------+
| Futaba        | MARYLAND                 |
| Great Planes  | Champaign, Illinois, USA |
| Tower Hobbies | Champaign, Illinois, USA |
+---------------+--------------------------+
```

- Um asterisco na cláusula `SELECT` denota «todos os atributos», por exemplo:

```
SELECT * FROM Loan
```

- NOTA: O SQL não permite o carácter '-' (hífen) nos identificadores, portanto, por exemplo, `balcaoNome` em vez de `balcao-nome`.
- NOTA: As maiúsculas e minúsculas não são distinguidas em palavras reservadas da linguagem SQL (e.g. `SELECT` é o mesmo que `select`).

A Componente SELECT (cont.)

- O SQL permite duplicados nas relações resultantes de consultas (que não chaves primárias).

A Componente SELECT (cont.)

- O SQL permite duplicados nas relações resultantes de consultas (que não chaves primárias).
- Para forçar a eliminação de duplicados, inserir a palavra-chave `DISTINCT` após o `SELECT`.
  Por exemplo: apresentar os nomes de todos os fornecedores sem repetições (o mesmo fornecedor pode fornecer vários tipos de artigos).

```
SELECT DISTINCT nome
  FROM Fornecedores
```

A Componente SELECT (cont.)

- O SQL permite duplicados nas relações resultantes de consultas (que não chaves primárias).
- Para forçar a eliminação de duplicados, inserir a palavra-chave `DISTINCT` após o `SELECT`.
  Por exemplo: apresentar os nomes de todos os fornecedores sem repetições (o mesmo fornecedor pode fornecer vários tipos de artigos).

```
SELECT DISTINCT nome
  FROM Fornecedores
```

- A palavra-chave `ALL` (valor por omissão) indica que os duplicados não devem ser removidos.

```
SELECT ALL nome
  FROM Fornecedores
```

A Componente SELECT (cont.)

- A cláusula `SELECT` pode conter expressões aritméticas envolvendo as operações, `+`, `-`, `*`, `/` com argumentos constantes e/ou atributos.
- Existem funções de agregação que permitem, entre outras, achar o valor mínimo contido num dado atributo (ver-se-à mais à frente).
- Dependendo das implementações, encontram-se normalmente definidas bibliotecas de funções.

Por exemplo:

```
mysql> SELECT nome,preco*1.23 FROM Produtos;
+--------------------------+-----------------+
| nome                     | preco*1.23      |
+--------------------------+-----------------+
| Tower Hobbies .46 ABC BB | 90.737897415161 |
| Tower Hobbies .61ABC BB  | 102.83789741516 |
| Futaba 9CAFS             | 417.43788818359 |
| Piper J-3 Cub 40         | 145.18789741516 |
+--------------------------+-----------------+
```

dá-nos o preço dos produtos já com o IVA aplicado.

A Componente FROM

- A cláusula `FROM` corresponde à operação de produto cartesiano da álgebra relacional. Indica as relações a consultar na avaliação da expressão.

Por exemplo: listar as encomendas existentes e (produto cartesiano) os fornecedores.

```
mysql> SELECT numero,data,nome,Fornecedores.nContribuinte
         FROM Encomendas,Fornecedores;
+--------+---------------------+--------------+---------------+
| numero | data                | nome         | nContribuinte |
+--------+---------------------+--------------+---------------+
|      1 | 2007-07-04 23:42:00 | Futaba       |     123456780 |
|      2 | 2007-07-04 23:43:00 | Futaba       |     123456780 |
|      1 | 2007-07-04 23:42:00 | Great Planes |     123456781 |
|      2 | 2007-07-04 23:43:00 | Great Planes |     123456781 |
```

Ou mais útil, listar as encomendas e respectivos fornecedores

```
   mysql> SELECT numero,data,nome,Fornecedores.nContribuinte
            FROM Encomendas,Fornecedores
           WHERE Encomendas.nContribuinte=Fornecedores.nContribuinte;
+--------+---------------------+-------------+---------------+
| numero | data                | nome        | nContribuinte |
+--------+---------------------+-------------+---------------+
|      2 | 2007-07-04 23:43:00 | Futaba      |     123456780 |
|      1 | 2007-07-04 23:42:00 | Great Planes |     123456781 |
+--------+---------------------+-------------+---------------+
```

A Componente WHERE

- A cláusula `WHERE` corresponde ao predicado de selecção da álgebra relacional. É formada por um predicado envolvendo atributos de relações que aparecem na cláusula `FROM`.
- Por exemplo: seleccionar todos os motores do tipo ABC de entre os produtos existentes em armazem:

```
mysql> SELECT * FROM Produtos WHERE nome LIKE '%ABC%' AND quantidade > 0;
+------------+--------------------------+-------+------------+
| codProduto | nome                     | preco | quantidade |
+------------+--------------------------+-------+------------+
|          1 | Tower Hobbies .46 ABC BB | 74.99 |         10 |
|          2 | Tower Hobbies .61ABC BB  | 84.99 |         10 |
|          3 | Tower Hobbies .75 ABC BB | 89.99 |          5 |
+------------+--------------------------+-------+------------+
```

  - Os resultados de comparações podem ser combinados por intermédio dos conectivos lógicos `AND`, `OR`, e `NOT`.
  - Os operadores relacionais habituais estão disponíveis.
  - A comparação entre sequências de caracteres pode ser feita através de expressões regulares (simples, ou complexas).

Operações com Sequências de Caracteres

- O SQL inclui um mecanismo de concordância de padrões para comparações envolvendo cadeias de caracteres. Os padrões são descritos recorrendo a dois caracteres especiais:
  - percentagem(%): O carácter % concorda com qualquer sub-cadeia.
  - sublinhado (_): O carácter _ concorda com qualquer carácter.

  A utilização deste mecanismo passa pela utilização operador relacional «`LIKE`» em vez da igualdade.

- O SQL suporta uma variedade de operações com cadeias de caracteres, tais como:
  - concatenação (utilizando «||»);
  - conversão de maiúsculas para minúsculas (e vice-versa);
  - calcular o comprimento, extracção de subcadeias, etc.

- O MySQL suporta (versão 5) expressões regulares como mecanismo de comparação de padrões.

Expressões Regulares

Uma expressão regular sobre um alfabeto finito V é definida de modo indutivo como se segue:

- ϵ (sequência vazia) é uma expressão regular;
- a é uma expressão regular, para todo o $a \in V$;
- (Iteração) se R é uma expressão regular sobre V, então também o é $(R)^*$;
- (Concatenação) se Q e R são expressões regulares sobre V, então também $(Q)(R)$.
- (União) se Q e R são expressões regulares sobre V, então também $(Q)|(R)$.

Expressões Regulares

O MySQL usa (versão 5) expressões regulares como mecanismo de comparação de padrões.

```
SELECT <seq_caracteres> REGEXP <expressão_regular>;
```

- `^` — associa com o inicio da seq. de caracteres.
- `$` — associa com o fim da seq. de caracteres.
- `.` — associa com um qualquer carácter, inclusive a mudança de linha.
- `a*` — associa com zero ou mais 'a'
- `a+` — associa com um ou mais 'a'
- `a?` — associa com zero ou um 'a'
- `de|abc` — associa com 'de' ou com 'abc'
- `[a-dX]`, `[^a-dX]` — associa (ou não) com os caracteres 'a' a 'd' e 'X'.

Expressões Regulares — Exemplos

- Para achar nomes que começam por 'b'

```
mysql> SELECT * FROM Pet WHERE name REGEXP '^b';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

- Para achar nomes contendo um 'w'

```
mysql> SELECT * FROM Pet WHERE name REGEXP 'w';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

- Para achar nomes contendo 'Carlos Seixas'

```
SELECT * FROM Enderecos WHERE rua REGEXP 'Carlos.+Seixas';
```

Expressões Regulares — Exemplos

No MySQL, os padrões SQL são, por omissão, insensíveis ao tipo do carácter (maiúscula ou minúscula). Se se pretende que a associação seja feita tendo em conta o tipo do carácter, então é necessário usar a palavra chave «Binary».

- Para achar nomes começando por um 'b' (e não por um 'B')

```
mysql> SELECT * FROM Pet WHERE name REGEXP BINARY '^b';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

Ordenando os Tuplos

Listar em ordem alfabética os nomes de todos os fornecedores de motores do tipo «ABC»

```
SELECT Fornecedores.nome
  FROM Fornecedores, Produtos
 WHERE Produtos.nome LIKE '%ABC%'
ORDER BY Fornecedores.nome
```

- Pode-se especificar «DESC» para ordenação descente ou «ASC» para ordenação ascendente, para cada atributo; por omissão, assume-se ordem ascendente.

```
ORDER BY Fornecedores.nome DESC
```

- Pode-se ter mais do que uma chave de ordenação, separando-as com vírgulas.

```
SELECT * FROM Produtos ORDER BY quantidade, preco
```

Funções de Agregação

Estas funções aplicam-se a multi-conjuntos de valores de uma coluna de uma relação, devolvendo um valor.

`avg`	valor médio
`min`	valor mínimo
`max`	valor máximo
`sum`	soma dos valores
`count`	número de valores

Por exemplo:

- Determinar o preço médio dos produtos:

```
SELECT AVG(preco) FROM Produtos
```

- Calcular o número de produtos:

```
SELECT COUNT(*) FROM Produtos
```

- Encontrar o preço máximo dos produtos:

```
SELECT MAX(preco) FROM Produtos
```

Funções de Agregação - GROUP BY

Podemos agrupar os atributos que pretendemos agregar em diferentes grupos, através do comando GROUP BY.

Listar o número de encomendas por fornecedor:

```
SELECT nome, count(*)
  FROM Encomendas, Fornecedores
 WHERE Encomendas.nContribuinte=Fornecedores.nContribuinte
GROUP BY nome
```

- Os atributos na cláusula SELECT fora de funções de agregação, têm de aparecer na lista GROUP BY.
- Se aparecer mais do que um atributo em GROUP BY, então cada grupo é formado pelos tuplo com valores iguais em todos esses atributos.

Funções de Agregação - HAVING

Podemos limitar os elementos que vão ser agregados através da inclusão de um predicado.

Listar os nomes de todas os fornecedores cujo valor médio dos preços dos seus produtos é superior a 100€.

```
SELECT Fornecedores.nome,AVG(preco)
FROM FornecedorDe,Produtos,Fornecedores
WHERE FornecedorDe.nContribuinte=Fornecedores.nContribuinte
AND FornecedorDe.codProduto=Produtos.codProduto
GROUP BY Fornecedores.nome
HAVING AVG(preco)>100
```

- Os predicados no comando HAVING são aplicados depois da formação dos grupos, enquanto que os predicados no comando WHERE são aplicados antes da formação dos grupos.

Valores Nulos

Os tuplos podem conter valores nulos, denotado por `NULL`, nalguns dos seus atributos.
`NULL` significa um valor desconhecido ou que não existe.
O predicado `IS NULL` pode ser utilizado para testar a existência de valores nulos.
Por exemplo: mostrar todos os fornecedores que possuem FAX.

```
SELECT nome
  FROM Fornecedores
 WHERE NOT(fax IS NULL)
```

- O resultado da aplicação de uma operação aritmética ao valor `NULL` é `NULL`. Por exemplo: `5 + NULL` é igual a `NULL`.
- Qualquer comparação com `NULL` retorna `UNKNOWN`.
- As funções de agregação, com excepção de `COUNT(*)`, ignoram os nulos.

Valores Nulos e Lógica Trivalente

Qualquer comparação com `NULL` retorna `UNKNOWN`.

Lógica trivalente usando o valor lógico `UNKNOWN`.

OR	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

AND	TRUE	FALSE	UNKNOWN
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

NOT	UNKNOWN
UNKNOWN	UNKNOWN

- O predicado `IS UNKNOWN` pode ser usado para testar se o valor de um dado predicado *P* é, ou não, definido. `P IS UNKNOWN` é verdadeiro se o valor de *P* não está definido (é `UNKNOWN`).
- O resultado da condição do comando `WHERE` é tratado como falso quando o seu valor é `UNKNOWN`.

Valores Nulos e Agregações

- Todas as funções de agregação excepto `COUNT(*)` ignoram tuplos com valores nulos nos atributos agregados.
  Por exemplo: calcule a média do prazo de entrega dos fornecedores.

```
SELECT AVG(prazo)
  FROM Fornecedores
```

  - os valores `NULL` serão ignorados no cálculo da média.
  - o resultado é `NULL` se não existir nenhum montante não-nulo

- A função de agregação `COUNT` também ignora os valores nulos, por exemplo:

```
SELECT COUNT(prazo)
  FROM Fornecedores
```

  não irá contabilizar os valores nulos.

- No entanto `COUNT(*)` irá contabilizar todos os tuplos, mesmo aqueles com todos os atributos a `NULL`.

Operações de Junção

- As operações de junção retornam uma relação como resultado da combinação de duas outras relações.

Operações de Junção

- As operações de junção retornam uma relação como resultado da combinação de duas outras relações.
- Estas operações adicionais são utilizadas habitualmente em consultas na componente `FROM`.

Operações de Junção

- As operações de junção retornam uma relação como resultado da combinação de duas outras relações.
- Estas operações adicionais são utilizadas habitualmente em consultas na componente `FROM`.
- Tipo de junção: define como tratar os tuplos que não estão relacionados entre si (basedados na condição de junção).

Operações de Junção

- As operações de junção retornam uma relação como resultado da combinação de duas outras relações.
- Estas operações adicionais são utilizadas habitualmente em consultas na componente `FROM`.
- Tipo de junção: define como tratar os tuplos que não estão relacionados entre si (basedados na condição de junção).
- Condição de junção: define quais os tuplos que são combinados nas duas relações, assim como quais os atributos que aparecem no resultado da junção.

Operações de Junção

- As operações de junção retornam uma relação como resultado da combinação de duas outras relações.
- Estas operações adicionais são utilizadas habitualmente em consultas na componente `FROM`.
- Tipo de junção: define como tratar os tuplos que não estão relacionados entre si (basedados na condição de junção).
- Condição de junção: define quais os tuplos que são combinados nas duas relações, assim como quais os atributos que aparecem no resultado da junção.

Tipos de Junção
inner join
left outer join
right outer join
full outer join

Condições de Junção
natural
on <predicado>
using $(A_1, A_2, \ldots, A_n)$

Operação de Junção Natural

- Sejam r e s relações nos esquemas R e S respectivamente. O resultado é uma relação no esquema $R \bigcup S$ que é obtido considerando cada par de tuplos t_r de r e t_s de s.
- Se t_r e t_s têm o mesmo valor em cada um dos atributos em $R \bigcap S$, um tuplo t é adicionado ao resultado, em que:
  - t tem os mesmos valores que t_r em r.
  - t tem os mesmos valores que t_s em s
- Exemplo:

```
R = (A, B, C, D)
S = (E, B, D)
Esquema resultado: (A, B, C, D, E)
```

em que as colunas comuns B e D foram usadas para seleccionar os tuplos a incluir no resultado.

Junção Interna (ou interior)

- As junções internas devem ser usadas conjuntamente com uma expressão condicional através dos comandos `ON` e `USING`.
- A junção interna (`INNER JOIN`) e um produto cartesiano das relações envolvidas considerando somente os tuplos que verificam a expressão condicional especificada.

```
SELECT * FROM table1 ,table2 WHERE table1.id=table2.id
```

é equivalente a:

```
SELECT * FROM table1 INNER JOIN table2 USING(id)
SELECT * FROM table1 INNER JOIN table2 ON table1.id=table2.id
```

- O condicional `ON` é uma qualquer expressão condicional tais como as que são usadas na componente `WHERE`. Deve-se usar a componente `ON` para especificar a forma de junção das tabelas, enquanto a componente `WHERE` deve ser usada para restringir os tuplos que vão ser seleccionados.

Junção Externa (ou exterior)

- Uma extensão da operação de junção que evita a perda de informação.
- Calcula a junção e depois adiciona ao resultado os tuplos de uma relação que não estão relacionados com a outra relação na junção.
- Numa junção `LEFT OUTER JOIN`, se não existe na tabela da direita uma coluna que associe através da condição `ON`, ou `USING` então todos os valores correspondentes no resultado da consulta (referente à tabela da direita) serão preenchidos com o valor `NULL`.

Junções — Relações de Exemplo

- Relação `Emprestimo`

numero	balcaoNome	quantia
L-170	Central	3000
L-230	Pólo I	4000
L-260	Pólo II	1700

- Relação `ClienteEmp`

nome	numero
João	L-170
Paulo	L-230
Raquel	L-155

Junções — Exemplos I

```
SELECT *
  FROM Emprestimo
INNER JOIN ClienteEmp
    ON Emprestimo.numero = ClienteEmp.numero
```

numero	balcaoNome	quantia	nome	numero
L-170	Central	3000	João	L-170
L-230	Pólo I	4000	Paulo	L-230

```
SELECT *
  FROM Emprestimo
LEFT OUTER JOIN ClienteEmp
    ON Emprestimo.numero = ClienteEmp.numero
```

numero	balcaoNome	quantia	nome	numero
L-170	Central	3000	João	L-170
L-230	Pólo I	4000	Paulo	L-230
L-260	Pólo II	1700	NULL	NULL

Junções — Exemplos II

```
SELECT *
  FROM Emprestimo
NATURAL INNER JOIN ClienteEmp
```

numero	balcaoNome	quantia	nome
L-170	Central	3000	João
L-230	Pólo I	4000	Paulo

- De forma equivalente podiamos escrever

```
SELECT *
  FROM Emprestimo
INNER JOIN ClienteEmp
  USING (numero)
```

neste caso explicita-se o atributo sobre o qual recaí a junção.

A junção natural (NATURAL [LEFT] JOIN) de duas tabelas é semanticamente equivalente a uma junção interna (INNER JOIN ou LEFT JOIN) com uma claúsula USING que considera todas as colunas comuns a ambas as tabelas.

Junções — Exemplos III

```
SELECT *
FROM Emprestimo
NATURAL RIGHT OUTER JOIN ClienteEmp
```

numero	balcaoNome	quantia	nome
L-170	Central	3000	João
L-230	Pólo I	4000	Paulo
L-155	NULL	NULL	Raquel

```
SELECT *
FROM Emprestimo
FULL OUTER JOIN ClienteEmp
USING (numero)
```

numero	balcaoNome	quantia	nome
L-170	Central	3000	João
L-230	Pólo I	4000	Paulo
L-260	Pólo II	1700	NULL
L-155	NULL	NULL	Raquel

Sub-consultas Embutidas

A linguagem SQL permite a inclusão de sub-consultas na cláusula `FROM` de uma consulta.

Por exemplo: achar a média dos balanços dos balcões cuja média dos balanços é maior do que 12000€.

```
SELECT balcaoNome, mediaBalanco
  FROM ( SELECT balcaoNome, avg(balanco)
           FROM Conta
         GROUP BY balcaoNome
       ) AS MediaBalcao (balcaoNome,mediaBalanco)
 WHERE mediaBalanco > 12000
```

Neste caso deixa de ser necessário a cláusula `HAVING` dado que se calcula a sub-consulta (vista temporária) e os atributos da mesma podem ser usados na consulta principal.

Relações Derivadas

A cláusula `WITH`[2] permite a definição de uma sub-consulta (vista temporária) cujo alcance é somente o da consulta aonde a mesma ocorre.

Exemplo: achar todas as contas cujo balanço seja igual ao valor máximo achado.

```
    WITH BalancoMax(valor)
      AS SELECT max(balanco)
           FROM Conta
  SELECT numero
    FROM Conta, BalancoMax
   WHERE Conta.balanco = BalancoMax.valor
```

A cláusula `WITH` é o correspondente à instrução de atribuição na álgebra relacional. No entanto é de notar que a mesma não pode ser usada sem estar associada a uma consulta.

[2] norma SQL-99, Oracle ($\geq$ 9.2), ainda não implementado no MySQL

Relações Derivadas

Outro exemplo: Achar todos os balcões aonde o valor total dos depósitos é maior do que a média dos valores dos depósitos em todos os balcões.

```
WITH TotalBalcao(balcaoNome, valor)
  AS SELECT balcaoNome, sum(balanco)
       FROM Conta
    GROUP BY balcaoNome
WITH MediaTotalBalcao(valor)
  AS SELECT avg(valor)
       FROM TotalBalcao
SELECT balcaoNome
  FROM TotalBalcao, MediaTotalBalcao
 WHERE TotalBalcao.valor > MediaTotalBalcao.valor
```

Notar que após a definição da relação derivada a mesma pode ser usada de imediato.

Vistas

- Em certas circunstâncias, não é desejável que todos os utilizadores possam aceder a todo o modelo lógico (i.e. a todas as relações armazenadas na base de dados)

- Considere o caso de um empregado que necessita de saber o número de empréstimo de um cliente, mas que não precisa de saber o montante desse empréstimo. Este empregado devera ver apenas a relação

```
(SELECT nome, numero
   FROM ClienteEmp, Emprestimo
  WHERE ClienteEmp.numero = Emprestimo.numero)
```

- Uma **vista** providencia um mecanismo de sonegação de informação.

- Qualquer relação que não pertença ao modelo conceptual mas que se torne visível ao utilizador como uma «relação virtual» é designada por **vista**.

Definição de Vistas

- Uma vista é definida por intermédio da instrução **CREATE VIEW**

  `CREATE VIEW NomeVista AS` <*consulta*>

  Em que ⟨consulta⟩ é uma consulta SQL qualquer. O nome da vista é `NomeVista`.

- Após a definição de uma vista, o seu nome pode ser utilizado para se referir à relação virtual gerada pela vista.

- Uma definição de uma vista não é o mesmo que criar uma nova relação a partir da avaliação da sua expressão. Em vez disso, a definição da vista permite guardar a expressão que depois é substituída nas consultas que utilizam a vista.

Exemplos de Vistas

- Considere-se a vista (com o nome `todosOsClientes`) contento os nomes das agências e seus clientes.

```
CREATE VIEW TodosOsClientes
       AS (SELECT balcaoNome, nome
             FROM ClienteEmp, Conta
            WHERE ClienteEmp.numero = Conta.numero)
            UNION
           (SELECT balcaoNome, nome
             FROM ClienteEmp, Emprestimo
            WHERE ClienteEmp.numero = Emprestimo.numero)
```

Exemplos de Vistas

- Considere-se a vista (com o nome `todosOsClientes`) contento os nomes das agências e seus clientes.

```
CREATE VIEW TodosOsClientes
        AS (SELECT balcaoNome, nome
              FROM ClienteEmp, Conta
             WHERE ClienteEmp.numero = Conta.numero)
             UNION
            (SELECT balcaoNome, nome
              FROM ClienteEmp, Emprestimo
             WHERE ClienteEmp.numero = Emprestimo.numero)
```

- Pode-se assim encontrar todos os clientes da agência de `Penacova` através de:

```
SELECT nome
  FROM TodosOsClientes
 WHERE balcaoNome = 'Penacova'
```

Vistas Definidas a Partir de Outras Vistas

- Uma vista pode ser utilizada na expressão de definição de outra vista.
- Uma vista v_1 **depende directamente** de uma vista v_2 se v_2 é utilizada na expressão que define v_1.
- Uma vista v_1 **depende** de uma vista v_2 se v_1 depende directamente de v_2 ou se existe um caminho de dependências entre v_1 e v_2.

Expansão de Vistas

- Forma de atribuir significado a vistas definidas em termos de outras vistas.
- Seja a vista v_1 definida em termos de uma expressão e_1 que pode ela própria conter vistas.
- Para expandir as vistas numa expressão repete-se sucessivamente o seguinte passo:

```
Repetir
  Encontrar uma vista v_i em e_1
  Substituir a vista v_i pela expressão que a define
Até que não ocorram mais vistas em e_1
```

Modificação da Base de Dados - Inserção

A inserção de tuplos numa tabela é feita em SQL com a instrução `INSERT`.

```
INSERT INTO <tabela> [(<lista_de_atributos>)]
     VALUES (<lista_de_expressões>)
```

No caso de não se especificar a lista_de_atributos subentende-se que são todos os atributos.
Exemplos:

- Adicionar um novo tuplo a Produtos

```
INSERT INTO Produtos
     VALUES (8,'F4U Corsair',403.57,5)
```

ou equivalentemente

```
INSERT INTO Produtos (nome,preco,quantidade)
     VALUES ('F4U Corsair',403.57,5)
```

Para todos os atributos da tabela não especificados são usados os valores por omissão (no caso do atributo `codProduto` é auto-incremental).

Modificação da Base de Dados - Inserção (cont.)

É também possível usar uma sintaxe alternativa.

```
INSERT INTO <tabela>
  SET <atributo1> = <expressão1>,
      <atributo2> = <expressão2>,
      ...
      <atributoN> = <expressãoN>
```

Exemplo:

```
INSERT INTO Produtos
       SET nome='P-47D Thunderbolt',
           preco=403.57,
           quantidade=3
```

Modificação da Base de Dados - Inserção (cont.)

Finalmente, é possível copiar valores entre tabelas através da instrução `INSERT ... SELECT`

Exemplo: dar como bónus a todos os clientes com empréstimos na agência Central de Coimbra da CGD, uma conta poupança de 2000€. O número do empréstimo servirá de número de conta poupança.

```
INSERT INTO Conta
    SELECT numero, 2000, balcaoNome
      FROM Emprestimo
     WHERE balcaoNome = 'Coimbra-central'
```

A instrução `SELECT ... FROM ...WHERE` é avaliada previamente à inserção de tuplos na relação (caso contrário consultas como `INSERT INTO tabela1 SELECT * FROM tabela1` causariam problemas)

Modificação da Base de Dados - Actualização

A actualização de tuplos duma tabela é feita em SQL com a instrução `UPDATE`

```
UPDATE <tabela>
   SET <atributo1> = <expressão1>,
       ...
       <atributoN> = <expressãoN>
 [ WHERE <condição> ]
```

Exemplos:

- Actualizar a quantidade em armazém de um produto após a recepção de uma encomenda:

```
UPDATE Produtos
   SET quantidade = quantidade+2
 WHERE codProduto=6;
```

- Actualizar os preços de acordo com uma taxa de inflação de 2%

```
UPDATE Produtos
   SET preco=preco*1.02;
```

Modificação da Base de Dados - Remoção

A remoção de tuplos de uma tabela é feita em SQL com a instrução `DELETE`.

```
DELETE
  FROM <tabela>
[ WHERE <condição> ]
```

Por exemplo: retirar da tabela Produtos todos os produtos dos quais não haja nenhum exemplar em armazém

```
DELETE
  FROM Produtos
 WHERE quantidade=0
```

Nota: esta instrução vai falhar (na actual implementação da base de dados) à conta de uma restrição de integridade (FornecedorDe).

Mudanças Através de Vistas

É possível usar uma **Vista** para actualizar uma tabela, utilizando as instruções `UPDATE`, `DELETE`, ou `INSERT`.

Para que seja possível actualizar uma tabela com uma **Vista** é necessário que haja uma relação injectiva (um-para-um) entre as linhas da **Vista** e as linhas da tabela que ela referência.

Além das restrições eventualmente impostas pelas permissões de acesso às tabelas subjacentes existem outras restrições que podem impedir a actualização de valores através de **Vistas**.

Mudanças Através de Vistas (cont.)

Para poder actualizar as tabelas subjacentes a **Vista** não pode conter:

- funções de agregação;
- `DISTINCT`; `GROUP BY`; `HAVING`; `UNION`;
- uma sub-consulta na lista de selecção;
- alguns tipos de junções;
- uma **Vista** não actualizável na cláusula `FROM`;
- múltiplas referências a uma dada coluna na tabela base.

Quanto à inserção de valores verifica-se ainda que:

- não pode haver duplicação nos nomes das colunas;
- a **Vista** tem de conter todas as colunas de preenchimento obrigatório e em que não esteja definido um valor por omissão;
- as colunas da **Vista** não pode ser uma coluna derivada de outras colunas.

Operação de União em SQL

Relações R_1 e R_2:

$R_1 =$

col1	col2
1	a
2	b
3	c
4	d

$R_2 =$

col1	col2
1	a
3	c
5	e

```
SELECT * from R1
UNION
SELECT * from R2
```

$R_1 \bigcup R_2 =$

col1	col2
1	a
2	b
3	c
4	d
5	e

Operação de Diferença em SQL

O MySQL não suporta directamente a operação de diferença.

$R_1 - R_2 =$

col1	col2
2	b
4	d

```
SELECT * FROM R1
 WHERE (col1,col2) NOT IN (SELECT *
                             FROM R2);
```

ou

```
SELECT * FROM R1
 WHERE NOT EXISTS (SELECT * FROM R2
                     WHERE R1.col1=R2.col1
                       AND R1.col2=R2.col2);
```

O modo mais eficiente:

```
          SELECT DISTINCT *
            FROM R1
LEFT OUTER JOIN R2 USING (col1,col2)
           WHERE R2.col1 IS NULL;
```

Operação de Intersecção em SQL

O MySQL não suporta directamente a operação de intersecção.

$R_1 \cap R_2 =$

col1	col2
1	a
3	c

```
SELECT * FROM R1
 WHERE (col1,col2)
    IN (SELECT * FROM R2);
```

ou

```
SELECT * FROM R1
 WHERE EXISTS (SELECT * FROM R2
                 WHERE R1.col1=R2.col1
                   AND R1.col2=R2.col2);
```

O modo mais eficiente:

```
    SELECT DISTINCT * FROM R1
INNER JOIN R2 USING (col1,col2);
```

Operação de Divisão em SQL

- Propriedade
  - Seja $q = r \div s$
  - Então q é a maior relação satisfazendo $q \times s \subseteq r$.
- Definição em termos de operações básicas da álgebra rel.
  Sejam $r(R)$ e $s(S)$ relações, com $S \subset R$

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

  Porquê?
  - $\Pi_{R-S}(r) \times s$ dá os elementos de r com todos os valores de S.
  - $\Pi_{R-S,S}(r)$ construi uma versão de r com os atributos da expressão anterior.
  - $\Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$ dá os tuplos t em $\Pi_{R-S}(r)$ tal que para algum tuplo $u \in s$, $tu \notin r$.

Operação de Divisão em SQL

Adequada para consultas que incluam a frase ”para todo”.

$$\begin{aligned} r \div s &= \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)) \\ &= \Pi_{(A,B,C)}(r) - \Pi_{(A,B,C)}((\Pi_{(A,B,C)}(r) \times s) - \Pi_{(A,B,C,D,E)}(r)) \end{aligned}$$

$$R = (A, B, C, D, E) \quad S = (D, E) \quad R - S = (A, B, C)$$

- $\Pi_{(A,B,C)}(r) \times s$ dá os elementos de r com todos os valores de (D, E).
- $\Pi_{(A,B,C,D,E)}(r)$ constrói uma versão de r com os atributos da expressão anterior. Neste caso: $\Pi_{(A,B,C,D,E)}(r) = r$.
- $\Pi_{(A,B,C)}((\Pi_{(A,B,C)}(r) \times s) - \Pi_{(A,B,C,D,E)}(r))$ dá os tuplos t em $\Pi_{(A,B,C)}(r)$ tal que para algum tuplo $u \in s$, $tu \notin r$.

MySQL — Exemplos de Consultas

Uma aproximação às consultas do tipo ÷ (do manual do MySQL)

- Que tipos de lojas estão presentes em uma, ou mais, cidades?

```
SELECT DISTINCT tiposLojas
  FROM Lojas
 WHERE EXISTS
        (SELECT *
           FROM CidadesLojas
          WHERE CidadesLojas.tipoLojas = Lojas.tiposLojas
        ) AS Aux;
```

- Que tipos de lojas não estão presentes em nenhuma cidade?

```
SELECT DISTINCT tipoLojas
  FROM Lojas
WHERE NOT EXISTS
       (SELECT *
          FROM CidadesLojas
         WHERE CidadesLojas.tipoLojas = Lojas.tipoLojas
       ) AS Aux;
```

MySQL — Exemplos de Consultas

- Que tipos de lojas estão presentes **em todas** as cidades?

```
SELECT DISTINCT tipoLojas FROM Lojas
 WHERE NOT EXISTS
       (SELECT * FROM Cidades
         WHERE NOT EXISTS
               (SELECT * FROM CidadesLojas
                 WHERE CidadesLojas.cidade = Cidades.cidade
                   AND CidadesLojas.tipoLojas =  Lojas.tipoLojas
               ) AS Aux1
       ) AS Aux2;
```

Este último exame é uma consulta «NOT EXISTS» duplamente embutida. Isto é, contém uma cláusula «NOT EXISTS» dentro de uma outra cláusula do mesmo tipo.
Formalmente, responde à questão «não existe uma cidade com uma loja que não está em Lojas?». É mais fácil de ver que uma consulta «NOT EXISTS» duplamente embutida responde à questão «é *x* verdade para todo o *y*?»

SQL Embutido

- SQL fornece uma linguagem declarativa para manipulação de bases de dados.
- Facilita a manipulação e permite optimizações muito difíceis se fossem programadas em linguagens imperativas.
- Mas há razões para usar SQL juntamente com linguagens de programação gerais (imperativas):
  - o SQL não tem a expressividade de uma máquina de Turing, há perguntas impossíveis de codificar em SQL (por exemplo, fechos transitivos); usando SQL juntamente com linguagens gerais é possível suprir esta deficiência;
  - nem tudo nas aplicações de bases de dados é declarativo (por exemplo, acções de afixar resultados, interfaces, etc). Essa parte pode ser programado em linguagens genéricas.
- O standard SQL define uma série de "encaixes", para várias linguagens de programação (e.g. Pascal, PHP, C, C++, Cobol, etc). À linguagem na qual se incluem comandos SQL chama-se linguagem hospedeira. Às estruturas SQL permitidas na linguagem hospedeira chama-se SQL embutido.

SQL Embutido

- Permite acesso a bases e dados SQL, via outra linguagens de programação.
- Toda a parte de acesso e manipulação da base de dados é feito através de código embutido. Todo o processamento associado é feito pelo sistema de bases de dados. A linguagem hospedeira recebe os resultados e manipula-os.
- O código tem que ser pré-processado. A parte SQL é transformada em código da linguagem hospedeira, mais chamadas a “run-time” do servidor.
- A forma particular como é feita a ligação entre a linguagem hospedeira e a linguagem SQL varia de linguagem para linguagem.

MySQL & PHP

Este exemplo simples mostra como, na linguagem PHP, fazer a ligação, executar uma consulta, mostrar as linhas do resultado e desligar do banco de dados MySQL.

Exemplo 1. Exemplo de visão geral da extensão MySQL

```
<?php
// Ligação e escolha da base de dados
$ligacao = mysqli_connect('mysql_host','mysql_user','mysql_password')
    or die('Não foi possível ligar: '.mysql_error());

echo 'Ligação bem sucedida';

mysqli_select_db($ligacao,'my_database')
    or die('Não foi possível selecionar a base de dados');
?>
```

A ligação entre o PHP e o MySQL é feita através de funções PHP próprias que estão definidas no módulo (extensão) php-mysqli:

MySQL & PHP

- `mysqli_connect` — Abre ou reutiliza uma conexão com um servidor MySQL.

  `resource mysqli_connect([string server[, string username[, string password[, bool new_link[, int client_flags]]]]])`

  Retorna um identificador de ligação MySQL em caso de sucesso, ou "FALSE" em caso de insucesso.

- `mysqli_select_db` — Selecciona uma base de dados MySQL.

  `bool mysqli_select_db (resource link_identifier,string database_name)`

  Devolve "TRUE" em caso de sucesso ou "FALSE" em caso de insucesso.

```
<?php
$ligacao = mysqli_connect('localhost','mysql_user','mysql_password');
if (!$ligacao) {
     die('Não foi possível fazer a ligação: '.mysql_error());
}

echo 'Ligação bem sucedida';
mysqli_close($ligacao);
?>
```

MySQL & PHP

```
// Executando a consulta SQL
$consulta = 'SELECT *
                FROM my_table';
$resultado = mysqli_query($ligacao,$consulta)
    or die('A consulta falhou!:'.mysql_error());

// Exibindo os resultados em HTML
echo "<table>";
while ($line = mysqli_fetch_array($resultado, MYSQL_ASSOC)) {
     echo "<tr>";
     foreach ($line as $col_value) {
           echo "<td> $col_value</td>";
     }
    echo "</tr>";
}
echo "</table>";

// Libertar o conjunto de resultados
mysqli_free_result($resultado);

// Fechar a ligação
mysqli_close($ligacao);
?>
```

MySQL & PHP

- `mysqli_query` — Realiza uma consulta MySQL

```
resource mysqli_query(resource link_identifier,string query)
```

`mysqli_query()` envia uma consulta para a base de dados activa no servidor da ligação presente `link_identifier`.
Nota: A string da query **não** deve terminar com ponto e virgula(;).

Para os comandos Select, Show, Explain ou Describe o valor de retorno é identificador de recurso ou "FALSE" se a consulta não foi executada correctamente. Para outros tipos de comandos SQL, o valor de retorno é "TRUE" em caso de sucesso e "FALSE" em erro.
O identificador de recurso pode depois ser usado em funções que lidam com os resultados das consultas (tabelas).

Com `mysqli_num_rows()` e com `mysqli_affected_rows()` podem-se obter quantas linhas foram devolvidas, ou quantas linhas foram afectadas, respectivamente.

`mysqli_free_result()` liberta (explicitamente) os recursos.

MySQL & PHP

- `mysqli_fetch_array` — Obtém uma linha como uma matriz associativa, uma matriz numérica, ou ambas.

```
array mysqli_fetch_array (resource resultado [, int result_type] )
```

Retorna uma matriz que corresponde a linha obtida **e move o ponteiro interno dos dados adiante**.

Parâmetros: resultado (o resultado obtido pela consulta)
O tipo de vector (“array”) que é obtido é uma constante e pode ter os seguintes valores: `MYSQL_ASSOC`, `MYSQL_NUM`, e o valor por omissão de `MYSQL_BOTH`.

Valores de retorno: uma vector que corresponde à linha obtida, ou FALSE se não houver mais linhas. O tipo do vector devolvido depende de como `result_type` esta definido.

Usando MYSQL_BOTH (por omissão), você terá um vector com ambos os índices, numérico e associativo. Usando `MYSQL_ASSOC`, você tem apenas os índices associativos, usando `MYSQL_NUM`, você tem apenas os índices numéricos.

ODBC

A norma ODBC (Open DataBase Connectivity) define um modo de um programa, concebido e compilado por um dado programador, comunicar com um servidor de bases de dados.

A norma define um interface entre programas em C e os SGBD (API, Application Program Interface) capaz de:

- estabelecer a ligação a uma base de dados;
- enviar consultas (de todos os tipos);
- obter os resultados.

Qualquer tipo de aplicação pode usar este tipo de interface para ter acesso a um SGBD que suporte a norma ODBC.

Todo o SGBD que suporte este protocolo define uma biblioteca com as funções de chamada necessárias para estabelecer a ligação e transferência de informação entre o programa e o servidor. O programa terá de ser compilado de forma a incluir a ligação à biblioteca.

ODBC/C

Para incorporar a ligação ODBC entre um programa e um SGBD é necessário incluir uma biblioteca apropriada.

- recebe a chamada ODBC API do programa;
- comunica com o servidor (SGBD);
- recebe o resultado que é disponibilizado ao programa.

```
#include <sql.h>
#include <sqlext.h>

gcc exemplo_odbc.c -lodbc -o exemplo
```

Dependências Funcionais e Normalização

- 1ª Forma Normal
- 2ª Forma Normal
- Objectivos na Concepção de Bases de Dados
- Dependências funcionais
- Decomposição
- Forma Normal de Boyce-Codd
- 3ª Forma Normal
- Dependências multi-valor
- 4ª Forma Normal
- Visão geral sobre o processo de concepção

1ª Forma Normal

- Um esquema R diz-se na 1ª forma normal se:
  - os domínios de todos os seus atributos são atómicos;
  - não pode haver repetição de registos.

- Um domínio é atómico se os seus elementos forem unidades indivisíveis.
  Exemplo de domínios não atómicos:
  - Atributos "naturalmente" compostos: Nomes, Endereços, etc.
  - Atributos com várias partes: Números de telefones com indicativos; B.I. com o número de validação.

- Os valores não atómicos complicam o armazenamento e encorajam repetições desnecessárias de dados.

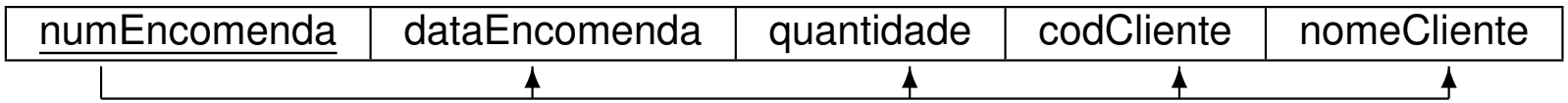
Daqui para a frente, assume-se que todas os esquemas de relações estão já na 1ª Forma Normal.

2ª Forma Normal

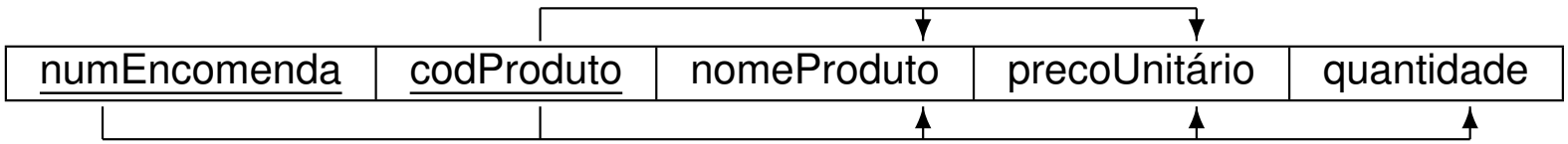
- Um esquema *R* diz-se na 2ª forma normal se:
  - está na 1ª forma normal;
  - cada atributo não chave tem de depender da chave da tabela na totalidade, e não apenas de uma parte dessa chave.
    - ★ se a chave primária é simples (um só atributo), então a relação está na 2ª forma normal.
    - ★ se a chave primária é composta (mais do que um atributo) e existe um atributo que depende somente de parte da chave primária, então a relação não está na 2ª forma normal.

Por exemplo:

Encomendas



Items-Encomendas



Daqui para a frente, assume-se que todas os esquemas de relações estão já na 2ª Forma Normal.

Objectivos na Concepção de Bases de Dados

Pretendem-se encontrar “bons” conjuntos de esquemas de relações para armazenar os dados.

Um “má” concepção pode levar a:

- Repetição de dados.
- Inconsistências devidas às operações de introdução, alteração, apagar de dados.
- Impossibilidade de representar certos tipos de informação.
- Dificuldade nas verificações de restrições de integridade.

Objectivos na Concepção:

- Evitar dados redundantes.
- Garantir que as relações relevantes sobre dados podem ser representadas.
- Facilitar a verificação de restrições de integridade.

Exemplo

Considere o esquema simples:

N.S.S.	Nome	Classificação	Vencimento/h	Horas Trab.
123-22	Abel	8	10	40
231-31	Silva	8	10	30
131-24	Sousa	5	7	30
434-26	Guiomar	5	7	32
612-67	Miguel	8	10	40

`Vencimento/h` depende de `Classificação`: este tipo de dependências (funcionais) entre atributos levanta problemas de:

- redundância:
  - Desperdiça-se espaço de armazenamento.
  - Dá azo a inconsistências.
  - Complica bastante a verificação da integridade dos dados.
- Dificuldade de representar certa informação
  - Não se pode armazenar informação de uma nova categoria de Classificação/Vencimento sem que haja um funcionário nessa categoria.

Decomposição de Esquemas de Relações

As dependências funcionais podem servir para identificar, e para indicar o caminho para uma melhor concepção global

Substituir uma (ou mais) relações por um conjunto de relações “mais pequenas”

N.S.S.	Nome	Classificação	Horas Trab.
123-22	Abel	8	40
231-31	Silva	8	30
131-24	Sousa	5	30
434-26	Guiomar	5	32
612-67	Miguel	8	40

Classificação	Vencimento/h
8	10
5	7

menos redundância; mais fácil manter a consistência dos dados; é possível acrescentar novos pares `Classificação/Vencimento`.

Problemas com a Decomposição de Esquemas de Relações

Ao fazer-se uma decomposição é necessário analisar se:

- a decomposição é necessária?
- a decomposição cria novos problemas?

- Formas normais
- Decomposição sem perdas
- Preservação das relações:
  - as restrições mantêm-se sem que seja necessário fazer junções entre relações;
  - as restrições verificam-se nas relações “menores”.

Exemplo

Considere o esquema simples:

`Amigos = (nome, telefone, codigoPostal, localidade)`

nome	telefone	codigoPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

- Redundância: os valores de `codigoPostal` e `localidade` são repetidos para cada amigo com um mesmo código postal.
  - Desperdiça-se espaço de armazenamento.
  - Dá azo a inconsistências.
  - Complica bastante a verificação da integridade dos dados.
- Dificuldade de representar certa informação.
  - Não se pode armazenar informação do código postal de uma localidade sem que hajam amigos dessa localidade. Podem usar-se valores nulos, mas estes são difíceis de gerir.

Decomposição

Decompor o esquema Amigos em:

```
Amigos1 = (nome,telefone, codigoPostal)
CPs = (codigoPostal, localidade)
```

Todos os atributos do esquema original (R) devem aparecer na decomposição em (R_1, R_2):

$$R = R_1 \cup R_2$$

Definição (Decomposição sem perdas)

Para todas as (instâncias de) relações r que "façam sentido" sobre o esquema R:

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

Note-se que o "façam sentido" depende do problema concreto.

Exemplo de Decomposição Sem Perdas

Decomposição de `Amigos` em `Amigos1` e `CPs`:

r

nome	telef.	CPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

$\Pi_{\text{Amigos1}}(r)$

nome	telef.	CPostal
Maria	1111	2815
João	2222	1000
Pedro	1112	1100
Ana	3333	2815

$\Pi_{\text{CPs}}(r)$

CPostal	localidade
2815	Caparica
1000	Lisboa
1100	Lisboa

$$\Pi_{\text{Amigos1}}(r) \bowtie \Pi_{\text{CPs}}(r) = r$$

Exemplo de Decomposição Com Perdas

Decomposição de `CPs` em: `CP1 = (CPostal)` e `Locs = (localidade)`

r

CPostal	localidade
2815	Caparica
1000	Lisboa
1100	Lisboa

$\Pi_{\text{CP1}}(r)$

CPostal
2815
1000
1100

$\Pi_{\text{Locs}}(r)$

localidade
Caparica
Lisboa

Exemplo de Decomposição Com Perdas

Decomposição de `CPs` em: `CP1 = (CPostal)` e `Locs = (localidade)`

r

CPostal	localidade
2815	Caparica
1000	Lisboa
1100	Lisboa

$\neq$

$\Pi_{CP1}(r) \bowtie \Pi_{Locs}(r)$

CPostal	localidade
2815	Caparica
2815	Lisboa
1000	Caparica
1000	Lisboa
1100	Caparica
1100	Lisboa

$\Pi_{CP1}(r)$

CPostal
2815
1000
1100

$\Pi_{Locs}(r)$

localidade
Caparica
Lisboa

Exemplo de Decomposição Com Perdas

Decomposição de `CPs` em: `CP1 = (CPostal)` e `Locs = (localidade)`

r

CPostal	localidade
2815	Caparica
1000	Lisboa
1100	Lisboa

$\neq$

$\Pi_{CP1}(r) \bowtie \Pi_{Locs}(r)$

CPostal	localidade
2815	Caparica
2815	Lisboa
1000	Caparica
1000	Lisboa
1100	Caparica
1100	Lisboa

$\Pi_{CP1}(r)$

CPostal
2815
1000
1100

$\Pi_{Locs}(r)$

localidade
Caparica
Lisboa

- Perdeu-se a informação de qual os CPs das localidades!
- Decompor parecia bom para evitar redundâncias.
- Mas decompor demais pode levar à perda de informação.

Outro Exemplo Com Perdas

Decomposição de `Amigos` em: `Amigos2 = (nome,telefone,localidade)` e `Loc = (localidade,CPostal)`.

r

nome	telef.	CPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

$\Pi_{\text{Amigos2}}(r)$

nome	telef.	localidade
Maria	1111	Caparica
João	2222	Lisboa
Pedro	1112	Lisboa
Ana	3333	Caparica

$\Pi_{\text{Loc}}(r)$

localidade	CPostal
Caparica	2815
Lisboa	1000
Lisboa	1100

Outro Exemplo Com Perdas

Decomposição de `Amigos` em: `Amigos2 = (nome,telefone,localidade)` e `Loc = (localidade,CPostal)`.

r

nome	telef.	CPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

$\neq$

$\Pi_{\text{Amigos2}}(r) \bowtie \Pi_{\text{Loc}}(r)$

nome	telef.	CPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
João	2222	1100	Lisboa
Pedro	1112	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

$\Pi_{\text{Amigos2}}(r)$

nome	telef.	localidade
Maria	1111	Caparica
João	2222	Lisboa
Pedro	1112	Lisboa
Ana	3333	Caparica

$\Pi_{\text{Loc}}(r)$

localidade	CPostal
Caparica	2815
Lisboa	1000
Lisboa	1100

Outro Exemplo Com Perdas

Decomposição de `Amigos` em: `Amigos2 = (nome,telefone,localidade)` e `Loc = (localidade,CPostal)`.

r

nome	telef.	CPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

$\neq$

$\Pi_{\text{Amigos2}}(r) \bowtie \Pi_{\text{Loc}}(r)$

nome	telef.	CPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
João	2222	1100	Lisboa
Pedro	1112	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

$\Pi_{\text{Amigos2}}(r)$

nome	telef.	localidade
Maria	1111	Caparica
João	2222	Lisboa
Pedro	1112	Lisboa
Ana	3333	Caparica

$\Pi_{\text{Loc}}(r)$

localidade	CPostal
Caparica	2815
Lisboa	1000
Lisboa	1100

- Perdeu-se a informação de qual é o CP do João (e do Pedro)!
- O que torna esta decomposição diferente da primeira?

Outro Exemplo Com Perdas

Decomposição de `Amigos` em: `Amigos2 = (nome,telefone,localidade)` e `Loc = (localidade,CPostal)`.

r

nome	telef.	CPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

$\neq$

$\Pi_{\text{Amigos2}}(r) \bowtie \Pi_{\text{Loc}}(r)$

nome	telef.	CPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
João	2222	1100	Lisboa
Pedro	1112	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

$\Pi_{\text{Amigos2}}(r)$

nome	telef.	localidade
Maria	1111	Caparica
João	2222	Lisboa
Pedro	1112	Lisboa
Ana	3333	Caparica

$\Pi_{\text{Loc}}(r)$

localidade	CPostal
Caparica	2815
Lisboa	1000
Lisboa	1100

- Perdeu-se a informação de qual é o CP do João (e do Pedro)!
- O que torna esta decomposição diferente da primeira?

Temos de ter critérios que nos permitam decompor uma relação, sem perda de informação.

Objectivo: um Bom Conjunto de Relações

Este objectivo pode ser atingido utilizando o seguinte “algoritmo”.

- Decidir se o esquema R já está num “bom” formato.
- Se não estiver, decompor R num conjunto de esquemas $\{R_1, R_2, \ldots, R_n\}$ tal que:
  - cada um deles está num “bom” formato;
  - A decomposição é sem perdas.
- A teoria é baseada em:
  - Dependências funcionais;
  - Dependências multi-valor

Dependências Funcionais

- Restrições sobre o conjunto de relações possíveis.
- Exige que os valores num conjunto de atributos determinem univocamente os valores noutro conjunto de atributos.
- São uma generalização da noção de chave.

Dependências Funcionais

- Restrições sobre o conjunto de relações possíveis.
- Exige que os valores num conjunto de atributos determinem univocamente os valores noutro conjunto de atributos.
- São uma generalização da noção de chave.

Definição (Dependência Funcional)

Seja R o esquema duma relação e $\alpha \subseteq R$ e $\beta \subseteq R$. A dependência funcional:

$$\alpha \rightarrow \beta$$

é verdadeira em R sse, para toda a relação possível (i.e. "que faça sentido") $r(R)$, sempre que dois tuplos t_1 e t_2 de r têm os mesmos valores em α, também têm os mesmos valores em β:

$$\forall_{t_1,t_2 \in r(R)} \quad t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

Dependências Funcionais (continuação)

- De forma equivalente.
  A dependência funcional $\alpha \rightarrow \beta$ é verdadeira em R sse

$$\forall_{a \in \mathrm{dom}(\alpha)} \Pi_{\beta}(\sigma_{\alpha=a}(r))$$

  tem no máximo 1 tuplo.

- Exemplo: Seja $r(A, B)$:

A	B
1	4
1	5
3	7

  Nesta instância, $A \rightarrow B$ não é verdadeira, mas $B \rightarrow A$ é.

Dependências Funcionais

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_1	d_2
a_1	b_2	c_2	d_1
a_2	b_1	c_3	d_2

$$AB \rightarrow C \qquad AB \not\rightarrow D$$

- AB não é uma chave
- A verificação para uma dada instância da relação não valida uma dependência funcional
- As dependências funcionais são restrições de integridade que tem de ser satisfeitas por todos os valores possíveis no esquema de relações.

Dependências Funcionais

Casos extremos

- $\{\} \rightarrow \alpha$
  Só se verifica se na relação *r* todos os tuplos têm o mesmo valor em α (nunca deve ser permitido).

Dependências Funcionais

Casos extremos

- $\{\} \rightarrow \alpha$
  Só se verifica se na relação *r* todos os tuplos têm o mesmo valor em α (nunca deve ser permitido).
- $\alpha \rightarrow \{\}$
  Verifica-se para toda a relação *r* e conjunto de atributos α.

Dependências Funcionais

Casos extremos

- $\{\} \rightarrow \alpha$
  Só se verifica se na relação *r* todos os tuplos têm o mesmo valor em α (nunca deve ser permitido).
- $\alpha \rightarrow \{\}$
  Verifica-se para toda a relação *r* e conjunto de atributos α.

Dependência Trivial Diz-se que uma dependência é trivial se é satisfeita por todas as relações (quer façam sentido ou não) sobre um esquema.
Por exemplo:

`nomeCliente,numEmprestimo` $\rightarrow$ `nomeCliente`
`nomeCliente` $\rightarrow$ `nomeCliente`

Em geral, $\alpha \rightarrow \beta$ é trivial se $\beta \subseteq \alpha$.

Dependências Funcionais

Chaves, são dependências funcionais.

Dependências Funcionais

Chaves, são dependências funcionais.

- K é uma **super-chave** no esquema R sse $K \rightarrow R$.

Dependências Funcionais

Chaves, são dependências funcionais.

- K é uma **super-chave** no esquema R sse $K \rightarrow R$.
- K é uma **chave candidata** em R sse $K \rightarrow R$, e para nenhum $\alpha \subset K$, $\alpha \rightarrow R$.

Dependências Funcionais

Chaves, são dependências funcionais.

- K é uma **super-chave** no esquema R sse $K \rightarrow R$.
- K é uma **chave candidata** em R sse $K \rightarrow R$, e para nenhum $\alpha \subset K$, $\alpha \rightarrow R$.

As dependências funcionais permitem expressar restrições, que não podem ser expressas somente através dos conceitos de chave.

Por exemplo, em `(nomeCliente, numEmprestimo, nomeBalcao, quantia)`.

- Espera-se que as seguintes dependências sejam verdadeiras:

  `numEmprestimo` $\rightarrow$ `quantia`
  `numEmprestimo` $\rightarrow$ `nomeBalcao`

Dependências Funcionais

Chaves, são dependências funcionais.

- K é uma **super-chave** no esquema R sse $K \rightarrow R$.
- K é uma **chave candidata** em R sse $K \rightarrow R$, e para nenhum $\alpha \subset K$, $\alpha \rightarrow R$.

As dependências funcionais permitem expressar restrições, que não podem ser expressas somente através dos conceitos de chave.

Por exemplo, em `(nomeCliente, numEmprestimo, nomeBalcao, quantia)`.

- Espera-se que as seguintes dependências sejam verdadeiras:

  `numEmprestimo` $\rightarrow$ `quantia`
  `numEmprestimo` $\rightarrow$ `nomeBalcao`

- Mas não se espera que a dependência abaixo seja verdadeira:

  `numEmprestimo` $\rightarrow$ `nomeCliente`

Dependências Funcionais Verdadeiras

Definição (Satisfação de Dependência Funcional)

Se uma relação r torna verdadeiras todas as dependências dum conjunto F, então diz-se que r **satisfaz** *F.*

Dependências Funcionais Verdadeiras

Definição (Satisfação de Dependência Funcional)

Se uma relação r torna verdadeiras todas as dependências dum conjunto F, então diz-se que r **satisfaz** *F.*

Definição (Dependência Funcional Verdadeira)

Diz-se que F **é verdadeira em** *R se todas as relações (possíveis) sobre R satisfazem as dependências em F.*

Dependências Funcionais Verdadeiras

Definição (Satisfação de Dependência Funcional)

Se uma relação r torna verdadeiras todas as dependências dum conjunto F, então diz-se que r **satisfaz** *F.*

Definição (Dependência Funcional Verdadeira)

Diz-se que F **é verdadeira em** *R se todas as relações (possíveis) sobre R satisfazem as dependências em F.*

Nota: Uma instância particular duma relação pode satisfazer uma dependência funcional mesmo que a dependência não seja verdadeira no esquema. Por exemplo, uma instância particular (em que, por acaso, nenhum empréstimo tenha mais que um cliente) satisfaz: `numEmprestimo` $\rightarrow$ `nomeCliente`.

Fecho de um Conjunto de Dependências Funcionais

Dado um conjunto F de dependências, há outras dependências que são logicamente implicadas por F. Por exemplo, se $A \rightarrow B$ e $B \rightarrow C$, então, ter-se-á $A \rightarrow C$.

Definição (Fecho de F)

Ao conjunto de todas as dependências funcionais implicadas por F chama-se **fecho** *de F (denotado por* F^+*).*

Podem encontrar-se todas as dependências em F^+ por aplicação dos Axiomas de Armstrong.

Definição (Axiomas de Armstrong)

- *Se* $\beta \subseteq \alpha$*, então* $\alpha \rightarrow \beta$ **(reflexividade)**
- *Se* $\alpha \rightarrow \beta$*, então* $\gamma\alpha \rightarrow \gamma\beta$ **(aumento)**
- *Se* $\alpha \rightarrow \beta$*, e* $\beta \rightarrow \gamma$*, então* $\alpha \rightarrow \gamma$ **(transitividade)**

Estes regras são:

- **coerentes**, isto é, só geram dependências que pertencem a F^+
- **completas**, isto é, geram todas as dependências pertencentes a F^+

Exemplo

Sejam

$$R = (A, B, C, G, H, I)$$

e

$$F = \{A \to B, A \to C, CG \to H, CG \to I, B \to H\}.$$

Podemos obter alguns dos elementos de F^+, aplicando os axiomas de Armstrong.

Exemplo

Sejam

$$R = (A, B, C, G, H, I)$$

e

$$F = \{A \to B, A \to C, CG \to H, CG \to I, B \to H\}.$$

Podemos obter alguns dos elementos de F^+, aplicando os axiomas de Armstrong.

- $A \to H$, por transitividade a partir de $A \to B$ e $B \to H$.

Exemplo

Sejam

$$R = (A, B, C, G, H, I)$$

e

$$F = \{A \to B, A \to C, CG \to H, CG \to I, B \to H\}.$$

Podemos obter alguns dos elementos de F^+, aplicando os axiomas de Armstrong.

- $A \to H$, por transitividade a partir de $A \to B$ e $B \to H$.
- $AG \to I$, por aumento de $A \to C$ com G, obtendo-se $AG \to CG$, de seguida, por transitividade com $CG \to I$.

Exemplo

Sejam

$$R = (A, B, C, G, H, I)$$

e

$$F = \{A \to B, A \to C, CG \to H, CG \to I, B \to H\}.$$

Podemos obter alguns dos elementos de F^+, aplicando os axiomas de Armstrong.

- $A \to H$, por transitividade a partir de $A \to B$ e $B \to H$.
- $AG \to I$, por aumento de $A \to C$ com G, obtendo-se $AG \to CG$, de seguida, por transitividade com $CG \to I$.
- $CG \to HI$, por aumento de $CG \to I$ inferindo $CG \to CGI$, de seguida por aumento de $CG \to H$ inferindo $CGI \to HI$, e depois transitividade.

Construção de F^+

Para calcular o fecho de um conjunto de dependências F podemos aplicar o seguinte algoritmo:

$F^+ := F$
repete
  para cada uma das dependências funcionais $f \in F^+$ **faz**
    aplicar reflexividade e aumento em f
    adicionar os resultados a F^+
  para cada par de dependências $f_1, f_2 \in F^+$ **faz**
    se f_1 e f_2 podem combinar-se por transitividade
    então adicionar a dependência resultante a F^+
até que F^+ não mude mais

NOTA: Veremos, mais tarde, outro procedimento para esta problema

Fecho de Dependências

Podemos facilitar a construção de F^+ usando mais algumas regras coerentes:

Fecho de Dependências

Podemos facilitar a construção de F^+ usando mais algumas regras coerentes:

- Se $\alpha \to \beta$ e $\alpha \to \gamma$, então $\alpha \to \beta\gamma$ **(união)**

Fecho de Dependências

Podemos facilitar a construção de F^+ usando mais algumas regras coerentes:

- Se $\alpha \to \beta$ e $\alpha \to \gamma$, então $\alpha \to \beta\gamma$ **(união)**
- Se $\alpha \to \beta\gamma$, então $\alpha \to \beta$ e $\alpha \to \gamma$ **(decomposição)**

Fecho de Dependências

Podemos facilitar a construção de F^+ usando mais algumas regras coerentes:

- Se $\alpha \to \beta$ e $\alpha \to \gamma$, então $\alpha \to \beta\gamma$ **(união)**
- Se $\alpha \to \beta\gamma$, então $\alpha \to \beta$ e $\alpha \to \gamma$ **(decomposição)**
- Se $\alpha \to \beta$ e $\gamma\beta \to \delta$, então $\alpha\gamma \to \delta$ **(pseudo-transitividade)**

Estas regras adicionais podem-se derivar dos Axiomas de Armstrong.

Fecho de um Conjunto de Atributos

Dado um conjunto de atributos α, define-se o fecho de α sobre F.

Definição (Fecho de um Conjunto de Atributos)

Dado um conjunto de dependências funcionais F, e $\alpha \subseteq R$, define-se o **fecho de α sobre F**, *denotado por α^+, como sendo o conjunto de atributos que dependem funcionalmente de α dado F, isto é:*

$$\alpha \rightarrow \beta \in F^+ \quad \text{sse} \quad \beta \subseteq \alpha^+$$

Algoritmo para calcular α^+.

$\alpha^+ := \alpha$
repete
  para cada $\beta \rightarrow \gamma \in F$ **faz**
    se $\beta \subseteq \alpha^+$ **então** $\alpha^+ := \alpha^+ \cup \gamma$
até que α^+ não mude mais

Exemplo de Fecho de Atributos

- $R = (A, B, C, G, H, I)$

Exemplo de Fecho de Atributos

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$

Exemplo de Fecho de Atributos

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- cálculo de $(AG)^+$
  1. $(AG)^+ := AG$

Exemplo de Fecho de Atributos

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- cálculo de $(AG)^+$
  1. $(AG)^+ := AG$
  2. $(AG)^+ := ABCG$ $(A \rightarrow C \text{ e } A \rightarrow B)$

Exemplo de Fecho de Atributos

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- cálculo de $(AG)^+$
  1. $(AG)^+ := AG$
  2. $(AG)^+ := ABCG$ $(A \rightarrow C \text{ e } A \rightarrow B)$
  3. $(AG)^+ := ABCGH$ $(CG \rightarrow H \text{ e } CG \subseteq AGBC)$

Exemplo de Fecho de Atributos

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- cálculo de $(AG)^+$
  1. $(AG)^+ := AG$
  2. $(AG)^+ := ABCG$ $(A \rightarrow C \text{ e } A \rightarrow B)$
  3. $(AG)^+ := ABCGH$ $(CG \rightarrow H \text{ e } CG \subseteq AGBC)$
  4. $(AG)^+ := ABCGHI$ $(CG \rightarrow I \text{ e } CG \subseteq AGBCH)$

Exemplo de Fecho de Atributos

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- cálculo de $(AG)^+$
  1. $(AG)^+ := AG$
  2. $(AG)^+ := ABCG$ $(A \rightarrow C \text{ e } A \rightarrow B)$
  3. $(AG)^+ := ABCGH$ $(CG \rightarrow H \text{ e } CG \subseteq AGBC)$
  4. $(AG)^+ := ABCGHI$ $(CG \rightarrow I \text{ e } CG \subseteq AGBCH)$

$(AG)^+$ já não muda mais dado que já inclui todos os atributos de R.

Uso de Fecho de Atributos

O cálculo do fecho de atributos pode ser usado para vários fins:

- **Testar super-Chaves:** para testar se α é super-chave, calcular α^+, e verificar se α^+ contém todos os atributos de R.

Uso de Fecho de Atributos

O cálculo do fecho de atributos pode ser usado para vários fins:

- **Testar super-Chaves:** para testar se α é super-chave, calcular α^+, e verificar se α^+ contém todos os atributos de R.
  - Será AG super-chave?

Uso de Fecho de Atributos

O cálculo do fecho de atributos pode ser usado para vários fins:

- **Testar super-Chaves:** para testar se α é super-chave, calcular α^+, e verificar se α^+ contém todos os atributos de R.
  - Será AG super-chave?
  - E algum subconjunto próprio de AG é super-chave?

Uso de Fecho de Atributos

O cálculo do fecho de atributos pode ser usado para vários fins:

- **Testar super-Chaves:** para testar se α é super-chave, calcular α^+, e verificar se α^+ contém todos os atributos de R.
  - Será AG super-chave?
  - E algum subconjunto próprio de AG é super-chave?

- **Testar dependências funcionais:** para verificar se a dependência $\alpha \rightarrow \beta$ é verdadeira (isto é pertence a F^+), basta verificar se $\beta \subseteq \alpha^+$, para um dado α^+.

Uso de Fecho de Atributos

O cálculo do fecho de atributos pode ser usado para vários fins:

- **Testar super-Chaves:** para testar se α é super-chave, calcular α^+, e verificar se α^+ contém todos os atributos de R.
  - Será AG super-chave?
  - E algum subconjunto próprio de AG é super-chave?

- **Testar dependências funcionais:** para verificar se a dependência $\alpha \rightarrow \beta$ é verdadeira (isto é pertence a F^+), basta verificar se $\beta \subseteq \alpha^+$, para um dado α^+.

- **Cálculo do fecho de F:** para cada $\gamma \subseteq R$, calcular γ^+. Para cada $S \subseteq \gamma^+$, devolver como resultado a dependência $\gamma \rightarrow S$.

Cobertura Canónica

- Um conjunto de dependências, podem conter algumas delas que são redundantes (por se inferirem das outras). Por exemplo:
  $A \to C$ é redundante em: $\{A \to B, B \to C, A \to C\}$. Porquê?

- Partes de dependências também podem ser redundantes. Por exemplo:
  - $\{A \to B, B \to C, A \to CD\}$ pode ser simplificado para $\{A \to B, B \to C, A \to D\}$. Porquê?
  - $\{A \to B, B \to C, AC \to D\}$ pode ser simplificado para $\{A \to B, B \to C, A \to D\}$. Porquê?

- Intuitivamente, uma cobertura canónica de F é um conjunto "minimal" de dependências, equivalente a F, e em que nenhuma dependência tem partes redundantes.

Atributos Dispensáveis

Considere o conjunto de dependências F e a dependência $\alpha \rightarrow \beta \in F$.

Definição (Atributo dispensável à esquerda)

O atributo A é dispensável à esquerda em α se $A \in \alpha$ e F implica $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.

Definição (Atributo dispensável à direita)

O atributo A é dispensável à direita em β se $A \in \beta$, e o conjunto $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ implica F.

Nota: a implicação na direcção oposta é trivial em ambos os casos.

Exemplos:

Atributos Dispensáveis

Considere o conjunto de dependências F e a dependência $\alpha \to \beta \in F$.

Definição (Atributo dispensável à esquerda)

O atributo A é dispensável à esquerda em α se $A \in \alpha$ e F implica $(F - \{\alpha \to \beta\}) \cup \{(\alpha - A) \to \beta\}$.

Definição (Atributo dispensável à direita)

O atributo A é dispensável à direita em β se $A \in \beta$, e o conjunto $(F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$ implica F.

Nota: a implicação na direcção oposta é trivial em ambos os casos.

Exemplos:

- Dado $F = \{A \to C, AB \to C\}$, B é dispensável em $AB \to C$ porque $A \to C$ implica $AB \to C$.

Atributos Dispensáveis

Considere o conjunto de dependências F e a dependência $\alpha \to \beta \in F$.

Definição (Atributo dispensável à esquerda)

O atributo A é dispensável à esquerda em α se $A \in \alpha$ e F implica $(F - \{\alpha \to \beta\}) \cup \{(\alpha - A) \to \beta\}$.

Definição (Atributo dispensável à direita)

O atributo A é dispensável à direita em β se $A \in \beta$, e o conjunto $(F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$ implica F.

Nota: a implicação na direcção oposta é trivial em ambos os casos.

Exemplos:

- Dado $F = \{A \to C, AB \to C\}$, B é dispensável em $AB \to C$ porque $A \to C$ implica $AB \to C$.
- Dado $F = \{A \to C, AB \to CD\}$, C é dispensável em $AB \to CD$ pois com $A \to C$, $AB \to CD$ pode ser inferido de $AB \to D$.

Teste para Atributos Dispensáveis

Considere o conjunto F de dependências, e a dependência $\alpha \rightarrow \beta \in F$.

Teste para Atributos Dispensáveis

Considere o conjunto F de dependências, e a dependência $\alpha \rightarrow \beta \in F$.

- Para testar se $A \in \alpha$ é dispensável em α, basta:
  1. calcular $(\alpha - A)^+$ usando as dependências em F;
  2. se $(\alpha - A)^+$ contém A, então A é dispensável.

Teste para Atributos Dispensáveis

Considere o conjunto F de dependências, e a dependência $\alpha \rightarrow \beta \in F$.

- Para testar se $A \in \alpha$ é dispensável em α, basta:
  1. calcular $(\alpha - A)^+$ usando as dependências em F;
  2. se $(\alpha - A)^+$ contém A, então A é dispensável.
- Para testar se $A \in \beta$ é dispensável em β, basta:
  1. calcular α^+ usando as dependências em $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$;
  2. se α^+ contém A, então A é dispensável.

Cobertura Canónica

Definição (Cobertura Canónica)

Uma cobertura canónica de F é um conjunto de dependências F_c tal que:

- *F implica todas as dependências em F_c, e*
- *F_c implica todas as dependências em F, e*
- *Nenhuma dependência em F_c contém atributos dispensáveis, e*
- *O lado esquerdo de cada dependência em F_c é único.*

Uma cobertura canónica de F é o conjunto de dependências funcionais com o mesmo poder expressivo que F e mínimo, isto é com o menor número de dependências funcionais possível.

Cálculo da Cobertura Canónica

Para calcular uma cobertura canónica de F:

$F_c := F$
repete
  Usar a regra da união para substituir as dependências em F_c,
    $\alpha_1 \to \beta_1$ e $\alpha_1 \to \beta_2$ por $\alpha_1 \to \beta_1\beta_2$
  enquanto há dependências com atributos dispensáveis **faz**
    Encontrar dependências $\alpha \to \beta$ com atributos dispensáveis (em α ou β)
    Quando se encontra um atributo dispensável, apaga-se esse atributo de $\alpha \to \beta$
  fimenquanto
até que F_c não muda.

Nota: A regra da união pode tornar-se aplicável depois de retirados alguns atributos dispensáveis. Por isso há que re-aplicá-la.

Exemplo de Cálculo de Cobertura Canónica

- $R = (A, B, C)$
- $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
- cálculo de F_c:

Exemplo de Cálculo de Cobertura Canónica

- $R = (A, B, C)$
- $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
- cálculo de F_c:
  1. Combinar $A \rightarrow BC$ e $A \rightarrow B$ para obter $A \rightarrow BC$;

Exemplo de Cálculo de Cobertura Canónica

- $R = (A, B, C)$
- $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
- cálculo de F_c:
  1. Combinar $A \rightarrow BC$ e $A \rightarrow B$ para obter $A \rightarrow BC$;
  2. $F_c = \{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$;

Exemplo de Cálculo de Cobertura Canónica

- $R = (A, B, C)$
- $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
- cálculo de F_c:
  1. Combinar $A \rightarrow BC$ e $A \rightarrow B$ para obter $A \rightarrow BC$;
  2. $F_c = \{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$;
  3. A é dispensável em $AB \rightarrow C$ porque $B \rightarrow C$ implica $AB \rightarrow C$;

Exemplo de Cálculo de Cobertura Canónica

- $R = (A, B, C)$
- $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
- cálculo de F_c:
  1. Combinar $A \rightarrow BC$ e $A \rightarrow B$ para obter $A \rightarrow BC$;
  2. $F_c = \{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$;
  3. A é dispensável em $AB \rightarrow C$ porque $B \rightarrow C$ implica $AB \rightarrow C$;
  4. $F_c = \{A \rightarrow BC, B \rightarrow C\}$;

Exemplo de Cálculo de Cobertura Canónica

- $R = (A, B, C)$
- $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
- cálculo de F_c:
  1. Combinar $A \rightarrow BC$ e $A \rightarrow B$ para obter $A \rightarrow BC$;
  2. $F_c = \{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$;
  3. A é dispensável em $AB \rightarrow C$ porque $B \rightarrow C$ implica $AB \rightarrow C$;
  4. $F_c = \{A \rightarrow BC, B \rightarrow C\}$;
  5. C é dispensável em $A \rightarrow BC$ pois $A \rightarrow BC$ é implicado por $A \rightarrow B$ e $B \rightarrow C$;

Exemplo de Cálculo de Cobertura Canónica

- $R = (A, B, C)$
- $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
- cálculo de F_c:
  1. Combinar $A \rightarrow BC$ e $A \rightarrow B$ para obter $A \rightarrow BC$;
  2. $F_c = \{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$;
  3. A é dispensável em $AB \rightarrow C$ porque $B \rightarrow C$ implica $AB \rightarrow C$;
  4. $F_c = \{A \rightarrow BC, B \rightarrow C\}$;
  5. C é dispensável em $A \rightarrow BC$ pois $A \rightarrow BC$ é implicado por $A \rightarrow B$ e $B \rightarrow C$;
  6. $F_c = \{A \rightarrow B, B \rightarrow C\}$;

Exemplo de Cálculo de Cobertura Canónica

- $R = (A, B, C)$
- $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
- cálculo de F_c:
  1. Combinar $A \rightarrow BC$ e $A \rightarrow B$ para obter $A \rightarrow BC$;
  2. $F_c = \{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$;
  3. A é dispensável em $AB \rightarrow C$ porque $B \rightarrow C$ implica $AB \rightarrow C$;
  4. $F_c = \{A \rightarrow BC, B \rightarrow C\}$;
  5. C é dispensável em $A \rightarrow BC$ pois $A \rightarrow BC$ é implicado por $A \rightarrow B$ e $B \rightarrow C$;
  6. $F_c = \{A \rightarrow B, B \rightarrow C\}$;
  7. Não há mais atributos dispensáveis. Verifica-se também que F_c não muda mais
- A cobertura canónica é: $F_c = \{A \rightarrow B, B \rightarrow C\}$.

Objectivos com a Concepção de BDs Relacionais

- Pretende-se encontrar “**bons**” conjuntos de esquemas relações, para armazenar os dados.

Objectivos com a Concepção de BDs Relacionais

- Pretende-se encontrar “**bons**” conjuntos de esquemas relações, para armazenar os dados.
- Uma “**má**” concepção pode levar a:

Objectivos com a Concepção de BDs Relacionais

- Pretende-se encontrar “**bons**” conjuntos de esquemas relações, para armazenar os dados.
- Uma “**má**” concepção pode levar a:
  - Repetição de dados;

Objectivos com a Concepção de BDs Relacionais

- Pretende-se encontrar "**bons**" conjuntos de esquemas relações, para armazenar os dados.

- Uma "**má**" concepção pode levar a:
  - Repetição de dados;
  - Impossibilidade de representar certos tipos de informação;

Objectivos com a Concepção de BDs Relacionais

- Pretende-se encontrar "**bons**" conjuntos de esquemas relações, para armazenar os dados.

- Uma "**má**" concepção pode levar a:
  - Repetição de dados;
  - Impossibilidade de representar certos tipos de informação;
  - Dificuldade na verificação da integridade.

Objectivos com a Concepção de BDs Relacionais

- Pretende-se encontrar "**bons**" conjuntos de esquemas relações, para armazenar os dados.
- Uma "**má**" concepção pode levar a:
  - Repetição de dados;
  - Impossibilidade de representar certos tipos de informação;
  - Dificuldade na verificação da integridade.
- Objectivos da concepção (para atingir um "bom" esquema):

Objectivos com a Concepção de BDs Relacionais

- Pretende-se encontrar "**bons**" conjuntos de esquemas relações, para armazenar os dados.
- Uma "**má**" concepção pode levar a:
  - Repetição de dados;
  - Impossibilidade de representar certos tipos de informação;
  - Dificuldade na verificação da integridade.
- Objectivos da concepção (para atingir um "bom" esquema):
  - Evitar dados redundantes;

Objectivos com a Concepção de BDs Relacionais

- Pretende-se encontrar "**bons**" conjuntos de esquemas relações, para armazenar os dados.
- Uma "**má**" concepção pode levar a:
  - Repetição de dados;
  - Impossibilidade de representar certos tipos de informação;
  - Dificuldade na verificação da integridade.
- Objectivos da concepção (para atingir um "bom" esquema):
  - Evitar dados redundantes;
  - Garantir que as relações relevantes sobre dados podem ser representadas;

Objectivos com a Concepção de BDs Relacionais

- Pretende-se encontrar "**bons**" conjuntos de esquemas relações, para armazenar os dados.

- Uma "**má**" concepção pode levar a:
  - Repetição de dados;
  - Impossibilidade de representar certos tipos de informação;
  - Dificuldade na verificação da integridade.

- Objectivos da concepção (para atingir um "bom" esquema):
  - Evitar dados redundantes;
  - Garantir que as relações relevantes sobre dados podem ser representadas;
  - Facilitar a verificação de restrições de integridade.

Exemplo

Concepção de um esquema de base de dados, avaliação do mesmo, e sua (se necessário) transformação num "bom" esquema.

- Concepção: Considere o esquema simples: `Amigos = (nome, telef, codPostal, localidade)`. E uma sua instância:

nome	telef	codPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

Exemplo

Concepção de um esquema de base de dados, avaliação do mesmo, e sua (se necessário) transformação num "bom" esquema.

- Concepção: Considere o esquema simples: `Amigos = (nome, telef, codPostal, localidade)`. E uma sua instância:

nome	telef	codPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

- Redundância: os valores de `(codPostal, localidade)` são repetidos para cada amigo com um mesmo código postal;
  - Desperdiça-se espaço de armazenamento;
  - Dá azo a inconsistências;
  - Complica bastante a verificação da integridade dos dados

Exemplo

Concepção de um esquema de base de dados, avaliação do mesmo, e sua (se necessário) transformação num "bom" esquema.

- Concepção: Considere o esquema simples: `Amigos = (nome, telef, codPostal, localidade)`. E uma sua instância:

nome	telef	codPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

- Redundância: os valores de `(codPostal, localidade)` são repetidos para cada amigo com um mesmo código postal;
  - Desperdiça-se espaço de armazenamento;
  - Dá azo a inconsistências;
  - Complica bastante a verificação da integridade dos dados
- Dificuldade de representar certa informação: Não se pode armazenar informação do código postal de uma localidade sem que hajam amigos dessa localidade.
  - Podem usar-se valores nulos, mas estes são difíceis de gerir.

Objectivos da Normalização

Após a concepção (e antes da implementação num dado SGBD), pretende-se obter um "bom" esquema. Temos então que:

- Avaliar: decidir se o um dado esquema R já está num "bom" formato.

Objectivos da Normalização

Após a concepção (e antes da implementação num dado SGBD), pretende-se obter um "bom" esquema. Temos então que:

- Avaliar: decidir se o um dado esquema R já está num "bom" formato.
- Transformar (normalizar): se não estiver, decompor R num conjunto de esquemas $\{R_1, R_2, \ldots, R_n\}$ tal que:
  - cada um deles está num "bom" formato;
  - a decomposição é sem perdas.

Objectivos da Normalização

Após a concepção (e antes da implementação num dado SGBD), pretende-se obter um "bom" esquema. Temos então que:

- Avaliar: decidir se o um dado esquema R já está num "bom" formato.
- Transformar (normalizar): se não estiver, decompor R num conjunto de esquemas $\{R_1, R_2, \ldots, R_n\}$ tal que:
  - cada um deles está num "bom" formato;
  - a decomposição é sem perdas.
- A **normalização** é baseada em:
  - dependências funcionais;
  - dependências multi-valor.

Exemplo - Decomposição

- Decompor o esquema Amigos em:

  `Amigos1 = (nome, telef, codPostal)`

  `CPs = (codPostal, localidade)`

Uma qualquer decomposição tem de preservar a informação, contida no esquema inicial.

Exemplo - Decomposição

- Decompor o esquema Amigos em:

  `Amigos1 = (nome, telef, codPostal)`

  `CPs = (codPostal, localidade)`

Uma qualquer decomposição tem de preservar a informação, contida no esquema inicial.

- **Não pode haver perda de atributos:** todos os atributos do esquema original (R) têm que aparecer na decomposição (R_1, R_2), isto é, $R = R_1 \cup R_2$.

Exemplo - Decomposição

- Decompor o esquema Amigos em:

  `Amigos1 = (nome, telef, codPostal)`

  `CPs = (codPostal, localidade)`

Uma qualquer decomposição tem de preservar a informação, contida no esquema inicial.

- **Não pode haver perda de atributos:** todos os atributos do esquema original (R) têm que aparecer na decomposição (R_1, R_2), isto é, $R = R_1 \cup R_2$.
- **Decomposição sem perdas:** para todas as relações possíveis r sobre o esquema R tem de se verificar que:

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

  A decomposição de R em R_1 e R_2 é sem perdas sse pelo menos uma das dependências abaixo pertence a F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

Exemplo de Decomposição Sem Perdas

Decomposição de Amigos em:

`Amigos1 = (nome,telef,codPostal)`
`CPs = (codPostal, localidade)`

r

nome	telef	codPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

$\Pi_{\text{Amigos1}}(r)$

nome	telef	codPostal
Maria	1111	2815
João	2222	1000
Pedro	1112	1100
Ana	3333	2815

$\Pi_{\text{CPs}}(r)$

codPostal	localidade
2815	Caparica
1000	Lisboa
1100	Lisboa

Verifica-se que:

$$\Pi_{\text{Amigos1}}(r) \bowtie \Pi_{\text{CPS}}(r) = r$$

Notar que é valida a dependência: `codPostal` $\rightarrow$ `localidade`, isto é, verifica-se $R_1 \cap R_2 \rightarrow R_2$.

Exemplo de Decomposição Com Perdas

Decomposição de Amigos em:

```
Amigos2 = (nome,telef,localidade)
Loc = (localidade,codPostal).
```

r

nome	telef	codPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

$\neq$

$\Pi_{\text{Amigos2}}(r) \bowtie \Pi_{\text{Loc}}(r)$

nome	telef	codPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
João	2222	1100	Lisboa
Pedro	1112	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

$\Pi_{\text{Amigos2}}(r)$

nome	telef	localidade
Maria	1111	Caparica
João	2222	Lisboa
Pedro	1112	Lisboa
Ana	3333	Caparica

$\Pi_{\text{CPs}}(r)$

localidade	codPostal
Caparica	2815
Lisboa	1000
Lisboa	1100

Note-se que nenhuma das duas dependências seguintes é válida:

- `localidade` $\rightarrow$ `nome, telefone`, isto é, $R_1 \cap R_2 \not\rightarrow R_1$.
- `localidade` $\rightarrow$ `codPostal`, isto é, $R_1 \cap R_2 \not\rightarrow R_2$.

Normalização por uso de Dependências

Quando se decompõe um esquema R com dependências F, em $R_1, R_2, \ldots, R_n$ quer-se:

Normalização por uso de Dependências

Quando se decompõe um esquema R com dependências F, em $R_1, R_2, \ldots, R_n$ quer-se:

- **Decomposição sem perdas**. Por forma a não se perder informação.

Normalização por uso de Dependências

Quando se decompõe um esquema R com dependências F, em $R_1, R_2, \ldots, R_n$ quer-se:

- **Decomposição sem perdas**. Por forma a não se perder informação.
- **Não haja redundância**. Ver-se-à mais à frente como ...

Normalização por uso de Dependências

Quando se decompõe um esquema R com dependências F, em $R_1, R_2, \ldots, R_n$ quer-se:

- **Decomposição sem perdas**. Por forma a não se perder informação.
- **Não haja redundância**. Ver-se-à mais à frente como ...
- **Preservação de dependências**. Por forma a que verificação das dependências possa ser feita de forma eficiente.

Normalização por uso de Dependências

Quando se decompõe um esquema R com dependências F, em $R_1, R_2, \ldots, R_n$ quer-se:

- **Decomposição sem perdas**. Por forma a não se perder informação.
- **Não haja redundância**. Ver-se-à mais à frente como ...
- **Preservação de dependências**. Por forma a que verificação das dependências possa ser feita de forma eficiente.

Seja F_i o conjunto de dependências de F^+ que só contêm atributos de R_i. A decomposição preserva as dependências se

$$(F_1 \cup F_2 \cup \cdots \cup F_n)^+ = F^+$$

Sem preservação de dependências, a garantia de integridade pode obrigar à computação de junções, sempre que se adicionam, apagam ou actualizam relações da base de dados. Tal pode tornar-se bastante ineficiente.

Exemplo

- Sejam $R = (A, B, C)$ e $F = \{A \rightarrow B, B \rightarrow C\}$.

Exemplo

- Sejam $R = (A, B, C)$ e $F = \{A \rightarrow B, B \rightarrow C\}$.
- Decomposição 1: $R_1 = (A, B)$, $R_2 = (B, C)$:
  - Decomposição sem perdas: $R_1 \cap R_2 = \{B\}$e $B \rightarrow BC$;
  - Preserva as dependências.

Exemplo

- Sejam $R = (A, B, C)$ e $F = \{A \rightarrow B, B \rightarrow C\}$.
- Decomposição 1: $R_1 = (A, B)$, $R_2 = (B, C)$:
  - Decomposição sem perdas: $R_1 \cap R_2 = \{B\}$e $B \rightarrow BC$;
  - Preserva as dependências.
- Decomposição 2: $R_1 = (A, B)$, $R_2 = (A, C)$:
  - Decomposição sem perdas: $R_1 \cap R_2 = \{A\}$ e $A \rightarrow AB$;
  - Não preserva as dependências. Não se pode verificar $B \rightarrow C$ sem calcular $R_1 \bowtie R_2$.

Teste de Preservação de Dependências

Para verificar se $\alpha \rightarrow \beta$ é preservada na decomposição R em $R_1, R_2, \ldots, R_n$ aplica-se o seguinte teste:

```
res := α
enquanto (houver alterações em res) faz
  para cada Ri na decomposição faz
    t := (res ∩ Ri)+ ∩ Ri
    res := res ∪ t
  fimpara
fimenquanto
```

Se `res` contém todos os atributos em β, então $\alpha \rightarrow \beta$ é preservada.
Aplica-se este teste a todas as dependências de F, para verificar se a decomposição preserva as dependências.
Ao contrário do cálculo de F^+ ou de $(F_1 \cup F_2 \cup \cdots \cup F_n)^+$, que têm ambos complexidade exponencial, este procedimento tem complexidade polinomial.

Forma Normal de Boyce-Codd

Definição (Forma Normal de Boyce-Codd)

Um esquema R diz-se na Forma Normal de Boyce-Codd, BCNF, relativamente a um conjunto de dependências F, sse para toda a dependência em F^+ da forma $\alpha \rightarrow \beta$, onde $\alpha \subseteq R$ e $\beta \subseteq R$, pelo menos uma das seguintes condições é verdadeira:

- $\alpha \rightarrow \beta$ *é trivial, isto é,* $\beta \subseteq \alpha$.
- α *é super-chave de R, isto é,* $\alpha \rightarrow R$.

Evita redundâncias

Verificação de dependências funcionais definidas sobre atributos de *R*, limita-se à verificação de chaves.

BCNF — Exemplo/Exercício

- $R = (A, B, C)$, $F = \{A \rightarrow B, B \rightarrow C\}$.

BCNF — Exemplo/Exercício

- $R = (A, B, C)$, $F = \{A \rightarrow B, B \rightarrow C\}$.
- Chave, $\{A\}$.

BCNF — Exemplo/Exercício

- $R = (A, B, C)$, $F = \{A \rightarrow B, B \rightarrow C\}$.
- Chave, $\{A\}$.
- R não está em BCNF.

BCNF — Exemplo/Exercício

- $R = (A, B, C)$, $F = \{A \rightarrow B, B \rightarrow C\}$.
- Chave, $\{A\}$.
- R não está em BCNF.
- Decomposição em $R_1 = (A, B)$, $R_2 = (B, C)$.

BCNF — Exemplo/Exercício

- $R = (A, B, C)$, $F = \{A \rightarrow B, B \rightarrow C\}$.
- Chave, $\{A\}$.
- R não está em BCNF.
- Decomposição em $R_1 = (A, B)$, $R_2 = (B, C)$.
  - R_1 e R_2 estão na BCNF.

BCNF — Exemplo/Exercício

- $R = (A, B, C)$, $F = \{A \rightarrow B, B \rightarrow C\}$.
- Chave, $\{A\}$.
- R não está em BCNF.
- Decomposição em $R_1 = (A, B)$, $R_2 = (B, C)$.
  - R_1 e R_2 estão na BCNF.
  - Decomposição sem perdas.

BCNF — Exemplo/Exercício

- $R = (A, B, C)$, $F = \{A \rightarrow B, B \rightarrow C\}$.
- Chave, $\{A\}$.
- R não está em BCNF.
- Decomposição em $R_1 = (A, B)$, $R_2 = (B, C)$.
  - R_1 e R_2 estão na BCNF.
  - Decomposição sem perdas.
  - Preserva as dependências.

Teste para BCNF

- Para determinar se uma dependência não trivial $\alpha \rightarrow \beta \in F^+$ é causa de violação de BCNF:

Teste para BCNF

- Para determinar se uma dependência não trivial $\alpha \rightarrow \beta \in F^+$ é causa de violação de BCNF:
  1. calcular α^+ (fecho de atributos em α);

Teste para BCNF

- Para determinar se uma dependência não trivial $\alpha \rightarrow \beta \in F^+$ é causa de violação de BCNF:
  1. calcular α^+ (fecho de atributos em α);
  2. Verificar se inclui todos os atributos de R, i.e. é super-chave de R. Se incluir, então não viola a condição de BCNF; caso contrário há violação da BCNF, a relação não está na BCNF.

Teste para BCNF

- Para determinar se uma dependência não trivial $\alpha \rightarrow \beta \in F^+$ é causa de violação de BCNF:
  1. calcular α^+ (fecho de atributos em α);
  2. Verificar se inclui todos os atributos de R, i.e. é super-chave de R. Se incluir, então não viola a condição de BCNF; caso contrário há violação da BCNF, a relação não está na BCNF.

- Teste simplificado: Em vez de verificar para todas as dependências de F^+, verificar apenas para as dependências numa cobertura canónica.

  Se nenhuma das dependências da cobertura canónica for contra a BCNF, então nenhuma das dependências de F^+ vai contra a BCNF.

Teste para BCNF

- Para determinar se uma dependência não trivial $\alpha \rightarrow \beta \in F^+$ é causa de violação de BCNF:
  1. calcular α^+ (fecho de atributos em α);
  2. Verificar se inclui todos os atributos de R, i.e. é super-chave de R. Se incluir, então não viola a condição de BCNF; caso contrário há violação da BCNF, a relação não está na BCNF.

- Teste simplificado: Em vez de verificar para todas as dependências de F^+, verificar apenas para as dependências numa cobertura canónica.

  Se nenhuma das dependências da cobertura canónica for contra a BCNF, então nenhuma das dependências de F^+ vai contra a BCNF.

- É no entanto necessário verificar se as dependências são preservadas.

Decomposição BCNF

Dado o esquema

(idCliente,nEmpréstimo,quantia)

Verifica-se a dependência funcional nEmpréstimo $\rightarrow$ quantia. No entanto nEmpréstimo não é uma super-chave.

Seja R um esquema que não está na BCNF, então existe pelo menos uma dependência funcional não trivial $\alpha \rightarrow \beta$ tal que α não é uma super-chave para R.

Substitui-se então R por dois esquemas:

- $(\alpha \cup \beta)$
- $(R - (\beta - \alpha))$

Ter-se-ia (para o exemplo acima)

- $(\alpha \cup \beta)$ = (nEmpréstimo,quantia)
- $(R - (\beta - \alpha))$ = (idCliente,nEmpréstimo)

Algoritmo para Decomposição BCNF

Algoritmo genérico para o cálculo de uma decomposição BCNF.

```
res := {R};
fim := false;
calcular F+;
enquanto ¬fim faz
  se há um esquema Ri em res que não está na BCNF então
    Seja α → β uma dependência (não trivial) sobre Ri
      tal que α → Ri ∉ F+ e α ∩ β = ∅;
    res := (res – Ri) ∪ (Ri – β) ∪ (α,β);
  senão
    fim := true;
  fimse
fimenquanto
```

Nota: no fim do algoritmo cada R_i está na BCNF, e a decomposição é sem perdas.

Exemplo de Decomposição BCNF

- Esquema:

  $\texttt{Amigos} = (\texttt{nome}, \texttt{telefone}, \texttt{codPostal}, \texttt{localidade})$

  $F = \{\texttt{nome} \rightarrow \texttt{telefone}, \texttt{codPostal};\ \texttt{codPostal} \rightarrow \texttt{localidade}\}$

Exemplo de Decomposição BCNF

- Esquema:

  Amigos = (nome, telefone, codPostal, localidade)
  $F = \{$nome $\rightarrow$ telefone, codPostal; codPostal $\rightarrow$ localidade$\}$

- Decomposição:

  res = {Amigos}
  codPostal $\rightarrow$ localidade $\in F^+$ e codPostal não é chave
  res = {(nome, telefone, codPostal),
  (codPostal, localidade)}

Exemplo de Decomposição BCNF

- Esquema:

  Amigos = (nome, telefone, codPostal, localidade)
  $F = \{$nome $\rightarrow$ telefone, codPostal; codPostal $\rightarrow$ localidade$\}$

- Decomposição:

  res = {Amigos}
  codPostal $\rightarrow$ localidade $\in F^+$ e codPostal não é chave
  res = {(nome, telefone, codPostal),
  (codPostal, localidade)}

- Pode-se verificar que os esquemas resultantes estão na BCNF.

Outro Exemplo de Decomposição BCNF

- Esquema:

  R = (balcao, localidade, activos, cliente, numEmprestimo, valor)
  F = {balcao $\rightarrow$ activos, localidade; numEmprestimo $\rightarrow$ valor, balcao }
  Chave = {numEmprestimo, cliente}

Outro Exemplo de Decomposição BCNF

- Esquema:

  R = (balcao, localidade, activos, cliente, numEmprestimo, valor)
  F = {balcao $\rightarrow$ activos, localidade; numEmprestimo $\rightarrow$ valor, balcao }
  Chave = {numEmprestimo, cliente}

- Decomposição:

  res = $\{R\}$
  balcao $\rightarrow$ activos, localidade $\in F^+$ e balcao não é chave em R
  R_1 = (balcao, activos, localidade)
  R_2 = (balcao, cliente, numEmprestimo, valor)
  res = $\{R_1, R_2\}$
  numEmprestimo $\rightarrow$ valor, balcao $\in F^+$ e nunEmprestimo não é chave em R_2
  R_3 = (numEmprestimo, valor, balcao)
  R_4 = (cliente, numEmprestimo)
  res = $\{R_1, R_3, R_4\}$

Outro Exemplo de Decomposição BCNF

- Esquema:

  R = (balcao, localidade, activos, cliente, numEmprestimo, valor)
  F = {balcao $\rightarrow$ activos, localidade; numEmprestimo $\rightarrow$ valor, balcao }
  Chave = {numEmprestimo, cliente}

- Decomposição:

  res = {R}
  balcao $\rightarrow$ activos, localidade $\in F^+$ e balcao não é chave em R
  R_1 = (balcao, activos, localidade)
  R_2 = (balcao, cliente, numEmprestimo, valor)
  res = {R_1, R_2}
  numEmprestimo $\rightarrow$ valor, balcao $\in F^+$ e nunEmprestimo não é chave em R_2
  R_3 = (numEmprestimo, valor, balcao)
  R_4 = (cliente, numEmprestimo)
  res = {R_1, R_3, R_4}

- Já está na BCNF.

Teste de Decomposição BCNF

Para verificar se R_i numa decomposição de R está na BCNF:

- Ou se testa R_i relativamente à restrição de F a R_i, isto é todas as dependências em F^+ que só contêm atributos de R_i;
- Ou se usa o conjunto original de dependências sobre R mas com o teste seguinte:

  Para todo $\alpha \subseteq R_i$, verificar se α^+ não contém nenhum atributo de $R_i - \alpha$, ou então α^+ contém todos os atributos de R_i.

  - Se a condição for violada por alguma $\alpha \rightarrow \beta$ em F, a dependência $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$ é verdadeira em R_i, e R_i viola BCNF.
  - Usa-se essa dependência para decompor R_i

BCNF e Preservação de Dependências

Nem sempre é possível obter uma decomposição BCNF que preserve as dependências.

- $R = (J, K, L)$, $F = \{JK \rightarrow L, L \rightarrow K\}$, existem duas chaves candidatas JK e JL.
- R não está na BCNF.
- Nenhuma decomposição de R preserva $JK \rightarrow L$.

Exemplo

Considere o esquema `Gestores = (balcao, cliente, gestorConta)`, com as dependências:

1. `gestorConta` $\longrightarrow$ `balcao`
2. `cliente, balcao` $\longrightarrow$ `gestorConta`

Não está na BCNF (a dependência 1 não é trivial e `gestorConta` não é super-chave, não implica `cliente`).

Decompor em `GestoresBal = (balcao, gestorConta)` e `Clientes = (cliente, gestorConta)`

Agora já está na BCNF.

Mas não se preservam as dependências!!! A dependência 2 não se pode verificar numa só relação.

Motivação para a 3ª Forma Normal

Há situações em que:

- a BCNF não preserva as dependências;
- a eficiência na verificação de integridade aquando de alterações é importante

Motivação para a 3ª Forma Normal

Há situações em que:

- a BCNF não preserva as dependências;
- a eficiência na verificação de integridade aquando de alterações é importante

Solução: definir uma forma normal mais fraca – 3ª Forma Normal:

- admite alguma redundância (o que pode trazer problemas, como veremos à frente);
- as dependências podem ser verificadas sem recorrer a junções;
- é sempre possível fazer uma decomposição sem perdas para a 3NF, que preserva as dependências.

Exemplo Motivador

O esquema `Gestores = (balcao, cliente, gestorConta)`, com as dependências:

1. `gestorConta` $\longrightarrow$ `balcao`
2. `cliente, balcao` $\longrightarrow$ `gestorConta`

não está na BCNF por causa da primeira dependência.

A única chave candidata de Gestores é {`cliente, balcao`}. Mas: Ao decompor Gestores com base na 1ª dependência, `balcao` vai ficar numa relação diferente daquela onde fica `cliente`.
Logo deixa de ser possível verificar a chave candidata de Gestores numa só relação!

Solução: Para se continuar a poder verificar a chave candidata da relação original, não decompor um esquema com base numa dependência que à direita contenha apenas alguns dos atributos duma chave candidata.

3ª Forma Normal

Definição (3ª Forma Normal)

Um esquema R está na 3ª Forma Normal, 3NF, sse para toda $\alpha \rightarrow \beta$ pertencente a F^+, pelo menos uma das seguintes condições é verdadeira:

- $\alpha \rightarrow \beta$ *é trivial, isto é,* $\beta \subseteq \alpha$.
- α *é super-chave de R, isto é,* $\alpha \rightarrow R$.
- *Todo atributo A em* $(\beta - \alpha)$ *está contido numa chave candidata de R, não necessariamente a mesma.*

Se *R* está na BCNF então está também na 3NF.

A 3ª condição relaxa a BCNF para garantir a preservação de dependências

3ª Forma Normal - Exemplo

$$
\begin{aligned}
R &= (J, K, L) \\
F &= \{JK \rightarrow L, L \rightarrow K\}
\end{aligned}
$$

3ª Forma Normal - Exemplo

$$\begin{aligned} R &= (J, K, L) \\ F &= \{JK \rightarrow L, L \rightarrow K\} \end{aligned}$$

Duas chaves candidatas: JK e JL.

3ª Forma Normal - Exemplo

$$\begin{aligned} R &= (J, K, L) \\ F &= \{JK \to L, L \to K\} \end{aligned}$$

Duas chaves candidatas: JK e JL.
R está na 3NF.

- $JK \to L$, JK é super-chave
- $L \to K$, K está contido numa chave candidata

3ª Forma Normal - Exemplo

$$\begin{aligned} R &= (J, K, L) \\ F &= \{JK \to L, L \to K\} \end{aligned}$$

Duas chaves candidatas: JK e JL.
R está na 3NF.

- $JK \to L$, JK é super-chave
- $L \to K$, K está contido numa chave candidata

A decomposição BCNF dá origem a $R_1 = (JL)$ e $R_2 = (LK)$.
Para esta decomposição o testar de $JK \to L$ obriga a uma junção.

3ª Forma Normal - Exemplo

$$\begin{aligned} R &= (J, K, L) \\ F &= \{JK \to L, L \to K\} \end{aligned}$$

Duas chaves candidatas: JK e JL.
R está na 3NF.

- $JK \to L$, JK é super-chave
- $L \to K$, K está contido numa chave candidata

A decomposição BCNF dá origem a $R_1 = (JL)$ e $R_2 = (LK)$.
Para esta decomposição o testar de $JK \to L$ obriga a uma junção.
Pode haver redundância em R

J	L	K
1	a	x
2	a	x
3	a	x
4	b	y

dado a dependência $L \to K$, os dados da coluna K são, sempre que haja repetição, redundantes.

Teste para 3NF

Optimização: Basta verificar para uma cobertura canónica de F (não é necessário verificar para toda a dependência em F^+).

Teste para 3NF

Optimização: Basta verificar para uma cobertura canónica de F (não é necessário verificar para toda a dependência em F^+).

- Usar o fecho de atributos para verificar, em toda a dependência $\alpha \rightarrow \beta$, se α é super-chave.

Teste para 3NF

Optimização: Basta verificar para uma cobertura canónica de F (não é necessário verificar para toda a dependência em F^+).

- Usar o fecho de atributos para verificar, em toda a dependência $\alpha \rightarrow \beta$, se α é super-chave.
- Se α não for super-chave, há que verificar se todo a atributo em β pertence a alguma chave candidata de R

Teste para 3NF

Optimização: Basta verificar para uma cobertura canónica de F (não é necessário verificar para toda a dependência em F^+).

- Usar o fecho de atributos para verificar, em toda a dependência $\alpha \rightarrow \beta$, se α é super-chave.
- Se α não for super-chave, há que verificar se todo a atributo em β pertence a alguma chave candidata de R

- Este teste é bastante ineficiente, pois envolve o cálculo de chaves candidatas.
- Pode demonstrar-se que verificar se um conjunto de esquemas está na 3NF é um problema intratável ("NP-hard").
- Existe uma decomposição para a 3NF (descrita mais à frente) com complexidade polinomial.

Algoritmo de Decomposição para 3NF

Seja F_c uma cobertura canónica de F;
i := 0;
para todo $\alpha \rightarrow \beta \in F_c$ **faz**
  se nenhum dos esquemas $R_j, 1 \leq j \leq i$ contém $\alpha\beta$ **então**
    i := i + 1;
    R_i := $\alpha\beta$;
  fimse
fimpara
se nenhum dos esquemas $R_j, 1 \leq j \leq i$ contém uma chave candidata de R **então**
  i := i + 1;
  R_i := uma (qualquer) chave candidata de R;
fimse

A decomposição resultante é $(R_1, R_2, \ldots, R_i)$.

O algoritmo descrito garante que:

- Todo o esquema R_i está na 3NF;
- A decomposição preserva as dependências e é sem perdas.

Exemplo

- Considere o esquema:
  `InfoGestores = (balcao, cliente, gestorConta, gabinete)`

- As dependências definidas sobre este esquema são:
  $$\texttt{gestorConta} \rightarrow \texttt{balcao, gabinete}$$
  $$\texttt{cliente, balcao} \rightarrow \texttt{gestorConta}$$
- A chave candidata é: {`cliente, balcao`}

- O ciclo **para todo** do algoritmo, leva à introdução dos seguintes esquemas na decomposição:
  `GestoresGab = (gestorConta, balcao, gabinete)`
  `Gestores = (cliente, balcao, gestorConta)`

- Como `Gestores` contém uma chave candidata de `InfoGestores`, o processo de decomposição termina.

BCNF versus 3NF

- É sempre possível decompor um esquema, num conjunto de esquemas na 3NF em que:
  - a decomposição é sem perdas;
  - as dependências são preservadas.

  Mas pode haver alguma redundância!!!

BCNF versus 3NF

- É sempre possível decompor um esquema, num conjunto de esquemas na 3NF em que:
  - a decomposição é sem perdas;
  - as dependências são preservadas.

  Mas pode haver alguma redundância!!!

- É sempre possível decompor um esquema, num conjunto de esquemas na BCNF em que:
  - a decomposição é sem perdas;
  - não há redundância.

  Mas nem sempre se podem preservar as dependências!!!

BCNF versus 3NF (cont.)

Exemplo de problemas causados pela redundância na 3NF

- Seja R a seguinte relação:

$$\begin{aligned} R &= (J, K, L) \\ F &= \{JK \rightarrow, L \rightarrow K\} \end{aligned}$$

com a seguinte instância

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
`null`	l_2	k_2

BCNF versus 3NF (cont.)

Exemplo de problemas causados pela redundância na 3NF

- Seja R a seguinte relação:

$$\begin{aligned} R &= (J, K, L) \\ F &= \{JK \rightarrow, L \rightarrow K\} \end{aligned}$$

com a seguinte instância

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
`null`	l_2	k_2

- A relação R, que está na 3NF mas não na BCNF, tem problemas de:

BCNF versus 3NF (cont.)

Exemplo de problemas causados pela redundância na 3NF

- Seja R a seguinte relação:

$$\begin{aligned} R &= (J, K, L) \\ F &= \{JK \rightarrow, L \rightarrow K\} \end{aligned}$$

com a seguinte instância

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
`null`	l_2	k_2

- A relação R, que está na 3NF mas não na BCNF, tem problemas de:
  - Repetição de informação (relação $L \rightarrow K$);

BCNF versus 3NF (cont.)

Exemplo de problemas causados pela redundância na 3NF

- Seja R a seguinte relação:

$$\begin{aligned} R &= (J, K, L) \\ F &= \{JK \rightarrow, L \rightarrow K\} \end{aligned}$$

com a seguinte instância

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
`null`	l_2	k_2

- A relação R, que está na 3NF mas não na BCNF, tem problemas de:
  - Repetição de informação (relação $L \rightarrow K$);
  - Necessita usar valores `null`, por exemplo para representar a dependência entre l_2, k_2 sem que haja valores correspondente em J).

BCNF versus 3NF (exemplo)

- Considere o esquema `Gestores = (balcao, cliente, gestor)`, com as dependências:

$$\begin{array}{rcl} \texttt{gestor} & \rightarrow & \texttt{balcao} \\ \texttt{cliente, balcao} & \rightarrow & \texttt{gestor} \end{array}$$

BCNF versus 3NF (exemplo)

- Considere o esquema `Gestores = (balcao, cliente, gestor)`, com as dependências:

$$\begin{array}{rcl} \texttt{gestor} & \rightarrow & \texttt{balcao} \\ \texttt{cliente, balcao} & \rightarrow & \texttt{gestor} \end{array}$$

- Está na 3NF:

BCNF versus 3NF (exemplo)

- Considere o esquema `Gestores = (balcao, cliente, gestor)`, com as dependências:

$$\begin{array}{rcl} \texttt{gestor} & \rightarrow & \texttt{balcao} \\ \texttt{cliente, balcao} & \rightarrow & \texttt{gestor} \end{array}$$

- Está na 3NF:
  - {`cliente, balcao`} é chave de Gestores;

BCNF versus 3NF (exemplo)

- Considere o esquema `Gestores = (balcao, cliente, gestor)`, com as dependências:

$$\begin{array}{rcl} \texttt{gestor} & \rightarrow & \texttt{balcao} \\ \texttt{cliente, balcao} & \rightarrow & \texttt{gestor} \end{array}$$

- Está na 3NF:
  - {`cliente, balcao`} é chave de Gestores;
  - {`balcao`} - {`gestor`} = {`balcao`} $\subseteq$ na chave candidata de Gestores (que é {`cliente, balcao`}).

BCNF versus 3NF (exemplo)

- Considere o esquema `Gestores = (balcao, cliente, gestor)`, com as dependências:

$$\begin{aligned} \texttt{gestor} &\rightarrow \texttt{balcao} \\ \texttt{cliente, balcao} &\rightarrow \texttt{gestor} \end{aligned}$$

- Está na 3NF:
  - {`cliente, balcao`} é chave de Gestores;
  - {`balcao`} - {`gestor`} = {`balcao`} $\subseteq$ na chave candidata de Gestores (que é {`cliente, balcao`}).

  Mas:

balcao	cliente	gestor
Lx	Maria	Ana
Cbr	Maria	João
Lx	Pedro	Ana
Lx	José	Ana
Cbr	null	Mário

BCNF versus 3NF (exemplo)

- Considere o esquema `Gestores = (balcao, cliente, gestor)`, com as dependências:

$$\begin{aligned} \texttt{gestor} &\rightarrow \texttt{balcao} \\ \texttt{cliente, balcao} &\rightarrow \texttt{gestor} \end{aligned}$$

- Está na 3NF:
  - {`cliente, balcao`} é chave de Gestores;
  - {`balcao`} - {`gestor`} = {`balcao`} $\subseteq$ na chave candidata de Gestores (que é {`cliente, balcao`}).

  Mas:

balcao	cliente	gestor
Lx	Maria	Ana
Cbr	Maria	João
Lx	Pedro	Ana
Lx	José	Ana
Cbr	null	Mário

  - Necessidade de ter um campo com `null` para associar gestores (ainda) sem clientes, a balcões;
  - Dados redundantes (balcao).

BCNF ou 3NF?

- Objectivos do design, numa primeira fase:
  - BCNF;
  - Decomposição sem perdas;
  - Preservação de dependências.

BCNF ou 3NF?

- Objectivos do design, numa primeira fase:
  - BCNF;
  - Decomposição sem perdas;
  - Preservação de dependências.

- Se tal não for possível, então há que optar por:
  - Não preservação de dependências;
  - Alguma redundância (devido ao uso da 3NF).

BCNF ou 3NF?

- Objectivos do design, numa primeira fase:
  - BCNF;
  - Decomposição sem perdas;
  - Preservação de dependências.

- Se tal não for possível, então há que optar por:
  - Não preservação de dependências;
  - Alguma redundância (devido ao uso da 3NF).

O SQL não fornece nenhuma forma directa de impor dependências que não sejam super-Chaves. Pode fazer-se usando `assertion` mas isso é muito ineficiente.
Mesmo quando a decomposição preserva as dependências, com o SQL não é possível testar de forma eficiente dependências cujo lado esquerdo não seja uma chave.

Dependências Multi-valor - Motivação

Há esquemas que estão na BCNF, que preservam as dependências, mas que, mesmo assim, não parecem estar suficientemente normalizadas.

Considere o seguinte esquema para o exemplo do banco:

`depositante(nConta, nomeCliente, moradaCliente)`

Se tivermos a dependência funcional

$$\texttt{nomeCliente} \rightarrow \texttt{moradaCliente}$$

então este esquema não está na BCNF.

Mas o banco quer deixar que um cliente possa ter mais do que uma morada, isto é, não quer impor esta dependência. Nesse caso, `depositante` já está na BCNF.

Motivação

Depositante

nConta	nomeCliente	moradaCliente
1	Carlos	morada1
1	Carlos	morada2
1	José	morada3
2	Carlos	morada1
2	Carlos	morada2
2	Maria	morada1
2	Maria	morada4
3	José	morada3
3	Maria	morada1
3	Maria	morada4

Está na BCNF (verificar).
Mas:

- Redundância!!!
- Problemas na inserção — Se quisermos adicionar uma nova morada (morada5) para o José, é necessário introduzir 2 tuplos:

```
(1, José, morada5) (3, José, morada5)
```

Motivação

Parece ser melhor decompor em:

nConta	nomeCliente
1	Carlos
1	José
2	Carlos
2	Maria
3	Maria
3	José

nomeCliente	moradaCliente
Carlos	morada1
Carlos	morada2
José	morada3
Maria	morada1
Maria	morada4

- Mas porquê? Que propriedades têm estes dados que permitem dizer isto? Como as exprimir?
- É certo que um dado cliente não têm sempre a mesma morada (independentemente da conta).
- Mas tem sempre o mesmo conjunto de moradas, independentemente da conta!

Dependências Multi-valor

Definição (Dependências Multi-valor)

Seja R um esquema e $\alpha \subseteq R$ e $\beta \subseteq R$. A **dependência multi-valor**

$$\alpha \twoheadrightarrow \beta$$

é verdadeira em R se em toda a relação possível $r(R)$, para todo o par de tuplo t_1, t_2 em r, se $t_1[\alpha] = t_2[\alpha]$, então existem necessariamente tuplos t_3 e t_4 em r tal que:

- $t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
- $t_1[\beta] = t_3[\beta]$
- $t_2[\beta] = t_4[\beta]$
- $t_1[R - \beta] = t_4[R - \beta]$
- $t_2[R - \beta] = t_3[R - \beta]$

Dependências Multi-valor (Cont.)

- Representação de $\alpha \twoheadrightarrow \beta$ em tabela:

	α	β	$R - \alpha - \beta$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

- Seja $R = \{\alpha, \beta, \gamma\}$.
- Diz-se que $\alpha \twoheadrightarrow \beta$ (α multi-determina β) sse para todas as possíveis $r(R)$
  Se $<\alpha_1, \beta_1, \gamma_1> \in r$ e $<\alpha_1, \beta_2, \gamma_2> \in r$
  então $<\alpha_1, \beta_1, \gamma_2> \in r$ e $<\alpha_1, \beta_2, \gamma_1> \in r$.
- Note-se que, como esta definição é simétrica em β e γ, segue-se que $\alpha \twoheadrightarrow \beta$ sse $\alpha \twoheadrightarrow \gamma$ (isto é $\alpha \twoheadrightarrow R - \alpha - \beta$).
- Note-se ainda que:
  - Se $\alpha \rightarrow \beta$ então $\alpha \twoheadrightarrow \beta$.
  - De facto, se $\alpha \rightarrow \beta$ então $\beta_1 = \beta_2$, e logo $\alpha \twoheadrightarrow \beta$.

Exemplo

- Para `Depositante` tem-se que: `nomeCliente` $\twoheadrightarrow$ `moradaCliente`:
  - Se dois tuplos têm o mesmo `nomeCliente`, tendo um `moradaCliente`$= m_1$ e `nConta` $= n_1$ e outro `moradaCliente` $= m_2$ e `nConta` $= n_2$.
  - Então têm que haver mais dois tuplos com esse `nomeCliente`:
    - ★ um com `nConta` $= n_2$ e `moradaCliente` $= m_1$;
    - ★ outro com `nConta` $= n_1$ e `moradaCliente` $= m_2$.

Depositante

nConta	nomeCliente	moradaCliente
1	Carlos	morada1
1	Carlos	morada2
1	José	morada3
2	Carlos	morada1
2	Carlos	morada2
2	Maria	morada1
2	Maria	morada4
3	José	morada3
3	Maria	morada1
3	Maria	morada4

Exemplo (Cont.)

- No nosso exemplo:

$$\texttt{nomeCliente} \twoheadrightarrow \texttt{moradaCliente}$$
$$\texttt{nomeCliente} \twoheadrightarrow \texttt{nConta}$$

- Esta definição formaliza a ideia de que cada valor particular de α (`nomeCliente`) tem associado um conjunto de valores β (`moradaCliente`) e um conjunto de valores de γ (`nConta`), e que estes dois conjuntos são independentes.

- Se são independentes, porque não metê-los em relações separadas?

Uso de Dependências Multi-valor

- Usam-se dependências multi-valor para:
  1. Testar relações, para verificar se são ou não relações válidas, dado um conjunto de dependência multi-valor.

Uso de Dependências Multi-valor

- Usam-se dependências multi-valor para:
  1. Testar relações, para verificar se são ou não relações válidas, dado um conjunto de dependência multi-valor.
  2. Especificar restrições no conjunto de (instâncias) de relações válidas. Assim, só devemos ter relações que satisfaçam o conjunto (pré-definido) de dependências funcionais e multi-valor.

Uso de Dependências Multi-valor

- Usam-se dependências multi-valor para:
  1. Testar relações, para verificar se são ou não relações válidas, dado um conjunto de dependência multi-valor.
  2. Especificar restrições no conjunto de (instâncias) de relações válidas. Assim, só devemos ter relações que satisfaçam o conjunto (pré-definido) de dependências funcionais e multi-valor.
- Se uma relação r não satisfizer uma dada dependência multi-valor, então é sempre possível construir uma relação r', por adição de tuplos em r, que satisfaz a dependência.

Teoria de Dependências Multi-valor

- Da definição de dependência multi-valor, podemos demonstrar:
  - Se $\alpha \rightarrow \beta$, então $\alpha \twoheadrightarrow \beta$, isto é, toda a dependência funcional é também dependência multi-valor.
  - $\alpha \twoheadrightarrow \beta$ é trivial sse $\beta \subseteq \alpha$ ou $\alpha \bigcup \beta = R$.

- Em geral temos um conjunto D de dependências funcionais e dependências multi-valor.

- O fecho D^+ de D é o conjunto de todas as dependências funcionais e multi-valor que são implicadas por D.
  - Pode calcular-se D^+ a partir de D, usando as definições de dependência funcional e multi-valor.
  - Tal como para dependências funcionais, há sistemas de inferência para calcular este fecho.

Inferência com Dependências Multi-valor

- Podem encontrar-se todas as dependências em D^+ por aplicação dos seguintes Axiomas (onde os primeiros 3 são os Axiomas de Armstrong) :

Definição (Axiomas Multi-valor)

- *Se* $\beta \subseteq \alpha$ *então* $\alpha \rightarrow \beta$ *(reflexividade)*
- *Se* $\alpha \rightarrow \beta$ *então* $\gamma\alpha \rightarrow \gamma\beta$ *(aumento)*
- *Se* $\alpha \rightarrow \beta$ *e* $\beta \rightarrow \gamma$ *então* $\alpha \rightarrow \gamma$ *(transitividade)*
- *Se* $\alpha \rightarrow \beta$ *então* $\alpha \twoheadrightarrow \beta$ *(replicação)*
- *Se* $\alpha \twoheadrightarrow \beta$ *então* $\alpha \twoheadrightarrow R - \beta - \alpha$ *(complemento)*
- *Se* $\alpha \twoheadrightarrow \beta$, $\gamma \subseteq R$ *e* $\delta \subseteq \gamma$ *então* $\gamma\alpha \twoheadrightarrow \delta\beta$ *(multi-aumento)*
- *Se* $\alpha \twoheadrightarrow \beta$ *e* $\beta \twoheadrightarrow \gamma$ *então* $\alpha \twoheadrightarrow \gamma - \beta$ *(multi-transitividade)*
- *Se* $\alpha \twoheadrightarrow \beta$, $\gamma \subseteq \beta$ *e existe* $\delta \subseteq R$ *tal que* $\delta \bigcap \beta = \emptyset$ *e* $\delta \rightarrow \gamma$ *então* $\alpha \rightarrow \gamma$ *(coalescência)*

Inferência com Dependências Multi-valor

- Podem encontrar-se todas as dependências em D^+ por aplicação dos seguintes Axiomas (onde os primeiros 3 são os Axiomas de Armstrong) :

Definição (Axiomas Multi-valor)

- *Se* $\beta \subseteq \alpha$ *então* $\alpha \rightarrow \beta$ *(reflexividade)*
- *Se* $\alpha \rightarrow \beta$ *então* $\gamma\alpha \rightarrow \gamma\beta$ *(aumento)*
- *Se* $\alpha \rightarrow \beta$ *e* $\beta \rightarrow \gamma$ *então* $\alpha \rightarrow \gamma$ *(transitividade)*
- *Se* $\alpha \rightarrow \beta$ *então* $\alpha \twoheadrightarrow \beta$ *(replicação)*
- *Se* $\alpha \twoheadrightarrow \beta$ *então* $\alpha \twoheadrightarrow R - \beta - \alpha$ *(complemento)*
- *Se* $\alpha \twoheadrightarrow \beta$, $\gamma \subseteq R$ *e* $\delta \subseteq \gamma$ *então* $\gamma\alpha \twoheadrightarrow \delta\beta$ *(multi-aumento)*
- *Se* $\alpha \twoheadrightarrow \beta$ *e* $\beta \twoheadrightarrow \gamma$ *então* $\alpha \twoheadrightarrow \gamma - \beta$ *(multi-transitividade)*
- *Se* $\alpha \twoheadrightarrow \beta$, $\gamma \subseteq \beta$ *e existe* $\delta \subseteq R$ *tal que* $\delta \bigcap \beta = \emptyset$ *e* $\delta \rightarrow \gamma$ *então* $\alpha \rightarrow \gamma$ *(coalescência)*

- Este conjunto de axiomas é coerente e completo.

4ª Forma Normal

Definição (4ª Forma Normal)

Um esquema R, com conjunto de dependência funcionais e multi-valor D, está na 4ª Forma Normal, 4NF, se para toda a dependência multi-valor $\alpha \twoheadrightarrow \beta$ pertencente a D^+, onde os atributos de α e β estão em R, pelo menos uma das seguintes condições é verdadeira:

- $\alpha \twoheadrightarrow \beta$ *é trivial, isto é* $\beta \subseteq \alpha$ *ou* $\alpha \cup \beta = R$.
- α *é super chave de R, isto é* $\alpha \rightarrow R$.

Se um esquema está na 4NF também está na BCNF

Restrição de Dependências Multi-valor

- A restrição do conjunto D a R_i é o conjunto de D_i com:
  - Todas as dependências em D^+ que só contêm atributos de R_i;
  - Todas as dependências multi-valor: $\alpha \twoheadrightarrow (\beta \bigcap R_i)$ onde $\alpha \subseteq R_i$ e $\alpha \twoheadrightarrow \beta \in D^+$.

- Com dependências multi-valor, a decomposição de R em R_1 e R_2 é sem perdas sse pelo menos uma das dependências abaixo pertence a D^+:
  - $R_1 \bigcap R_2 \twoheadrightarrow R_1$;
  - $R_1 \bigcap R_2 \twoheadrightarrow R_2$.

Algoritmo de Decomposição para 4NF

resultado := {R};
acabou := falso;
calcular D^+;
Seja D_i a restrição de D^+ a R_i
enquanto (**não** acabou)
  se existe esquema $R_i \in$ resultado que não está na 4NF **então**
    Seja $\alpha \twoheadrightarrow \beta$ não trivial e verdadeira em R_i tal que
      $\alpha \rightarrow R_i \notin D_i$, e $\alpha \bigcap \beta = \emptyset$;
    resultado := $(\text{resultado} - R_i) \bigcup (R_i - \beta) \bigcup (\alpha, \beta)$;
  senão acabou := verdade;
fimenquanto

Nota: A decomposição é sem perdas

Exemplo

- $R = (A, B, C, G, H, I)$
  $D = \{A \twoheadrightarrow B$
  $B \twoheadrightarrow HI$
  $CG \rightarrow H\}$

Exemplo

- $R = (A, B, C, G, H, I)$
  $D = \{A \twoheadrightarrow B$
  $B \twoheadrightarrow HI$
  $CG \rightarrow H\}$
- R não está na 4NF pois $A \twoheadrightarrow B \in F$ e A não é super-chave de R e $\{A, B\} \neq R$.

Exemplo

- $R = (A, B, C, G, H, I)$
  $D = \{A \twoheadrightarrow B$
  $B \twoheadrightarrow HI$
  $CG \rightarrow H\}$
- R não está na 4NF pois $A \twoheadrightarrow B \in F$ e A não é super-chave de R e $\{A, B\} \neq R$.
- Decomposição:
  1. $R_1 = (A, B)$ (R_1 está na 4NF. A única dep. em R_1 é trivial);
  2. $R_2 = (A, C, G, H, I)$ (R_2 não está na 4NF: $CG \rightarrow H$);

Exemplo

- $R = (A, B, C, G, H, I)$
  $D = \{A \twoheadrightarrow B$
  $B \twoheadrightarrow HI$
  $CG \rightarrow H\}$
- R não está na 4NF pois $A \twoheadrightarrow B \in F$ e A não é super-chave de R e $\{A, B\} \neq R$.
- Decomposição:
  1. $R_1 = (A, B)$ (R_1 está na 4NF. A única dep. em R_1 é trivial);
  2. $R_2 = (A, C, G, H, I)$ (R_2 não está na 4NF: $CG \rightarrow H$);
  3. $R_3 = (C, G, H)$ (R_3 está na 4NF);
  4. $R_4 = (A, C, G, I)$ (R_4 não está na 4NF: $A \twoheadrightarrow I$).

     Como $A \twoheadrightarrow B$ e $B \twoheadrightarrow HI$ então $A \twoheadrightarrow HI \in D^+$, e $A \twoheadrightarrow I$ está na restrição de D^+ a R_4.

Exemplo

- $R = (A, B, C, G, H, I)$
  $D = \{A \twoheadrightarrow B$
  $B \twoheadrightarrow HI$
  $CG \rightarrow H\}$
- R não está na 4NF pois $A \twoheadrightarrow B \in F$ e A não é super-chave de R e $\{A, B\} \neq R$.
- Decomposição:
  1. $R_1 = (A, B)$ (R_1 está na 4NF. A única dep. em R_1 é trivial);
  2. $R_2 = (A, C, G, H, I)$ (R_2 não está na 4NF: $CG \rightarrow H$);
  3. $R_3 = (C, G, H)$ (R_3 está na 4NF);
  4. $R_4 = (A, C, G, I)$ (R_4 não está na 4NF: $A \twoheadrightarrow I$).
     Como $A \twoheadrightarrow B$ e $B \twoheadrightarrow HI$ então $A \twoheadrightarrow HI \in D^+$, e $A \twoheadrightarrow I$ está na restrição de D^+ a R_4.
  5. $R_5 = (A, I)$ (R_5 está na 4NF);
  6. $R_6 = (A, C, G)$ (R_6 está na 4NF).

Exemplo

- $R = (A, B, C, G, H, I)$
  $D = \{A \twoheadrightarrow B$
  $B \twoheadrightarrow HI$
  $CG \rightarrow H\}$
- R não está na 4NF pois $A \twoheadrightarrow B \in F$ e A não é super-chave de R e $\{A, B\} \neq R$.
- Decomposição:
  1. $R_1 = (A, B)$ (R_1 está na 4NF. A única dep. em R_1 é trivial);
  2. $R_2 = (A, C, G, H, I)$ (R_2 não está na 4NF: $CG \rightarrow H$);
  3. $R_3 = (C, G, H)$ (R_3 está na 4NF);
  4. $R_4 = (A, C, G, I)$ (R_4 não está na 4NF: $A \twoheadrightarrow I$).
     Como $A \twoheadrightarrow B$ e $B \twoheadrightarrow HI$ então $A \twoheadrightarrow HI \in D^+$, e $A \twoheadrightarrow I$ está na restrição de D^+ a R_4.
  5. $R_5 = (A, I)$ (R_5 está na 4NF);
  6. $R_6 = (A, C, G)$ (R_6 está na 4NF).

Ter-se-ia então a decomposição de R em $\{R_1, R_3, R_5, R_6\}$.

4NF e Preservação de Dependências

- Tal como a BCNF, a 4NF pode não preservar as dependências:
  - $R = (A, B, C, G, H, I)$ com $D = \{A \twoheadrightarrow B, B \twoheadrightarrow HI, CG \rightarrow H\}$, foi decomposto em $\{(A, B), (C, G, H), (A, I), (A, C, G)\}$.
  - A dependência $B \twoheadrightarrow HI$ não pode ser testada apenas numa destas relações.

4NF e Preservação de Dependências

- Tal como a BCNF, a 4NF pode não preservar as dependências:
  - $R = (A, B, C, G, H, I)$ com $D = \{A \twoheadrightarrow B, B \twoheadrightarrow HI, CG \rightarrow H\}$, foi decomposto em $\{(A, B), (C, G, H), (A, I), (A, C, G)\}$.
  - A dependência $B \twoheadrightarrow HI$ não pode ser testada apenas numa destas relações.
- Aplicam-se aqui as mesmas soluções de compromisso que entre a BCNF e a 3NF:

4NF e Preservação de Dependências

- Tal como a BCNF, a 4NF pode não preservar as dependências:
  - $R = (A, B, C, G, H, I)$ com $D = \{A \twoheadrightarrow B, B \twoheadrightarrow HI, CG \rightarrow H\}$, foi decomposto em $\{(A, B), (C, G, H), (A, I), (A, C, G)\}$.
  - A dependência $B \twoheadrightarrow HI$ não pode ser testada apenas numa destas relações.
- Aplicam-se aqui as mesmas soluções de compromisso que entre a BCNF e a 3NF:
  - Objectivos numa primeira fase:
    - ★ 4NF;
    - ★ decomposição sem perdas;
    - ★ preservação de dependências.

4NF e Preservação de Dependências

- Tal como a BCNF, a 4NF pode não preservar as dependências:
  - $R = (A, B, C, G, H, I)$ com $D = \{A \twoheadrightarrow B, B \twoheadrightarrow HI, CG \rightarrow H\}$, foi decomposto em $\{(A, B), (C, G, H), (A, I), (A, C, G)\}$.
  - A dependência $B \twoheadrightarrow HI$ não pode ser testada apenas numa destas relações.
- Aplicam-se aqui as mesmas soluções de compromisso que entre a BCNF e a 3NF:
  - Objectivos numa primeira fase:
    - ★ 4NF;
    - ★ decomposição sem perdas;
    - ★ preservação de dependências.
- Se tal não for possível, então há que optar por uma de duas possíveis soluções:
  - Não preservação de dependências.
  - Alguma redundância:
    - ★ tentar BCNF;
    - ★ se tal ainda não preserva dependências, normalizar para a 3NF.

Mais Formas Normais

As **dependências de junção** generalizam as multi-valor.
Dão origem à **forma normal projecção-junção (PJNF)** (também chamada de 5ª forma normal).

Uma classe ainda mais geral de restrições leva à **forma normal de domínio-chave**.

Problemas com estas restrições muito gerais:

- é difícil raciocinar sobre elas;
- não têm conjuntos coerentes e completos de regras de inferência.

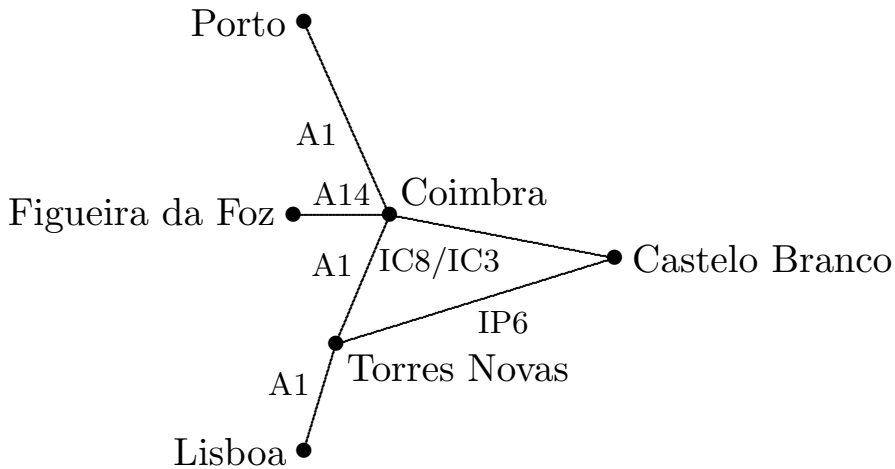
Logo, raramente são usadas.

Bases de Dados de Grafos

Um Grafo G é constituído por um conjunto N de elementos e por uma relação binária $A \subseteq N \times N$ entre esses elementos.

$$G = (N, A)$$

N – nós; A – arcos.



- Arcos dirigidos, ou não dirigidos.
- Percorrer o grafo: Caminhos; Percurso em amplitude; Percurso em profundidade.

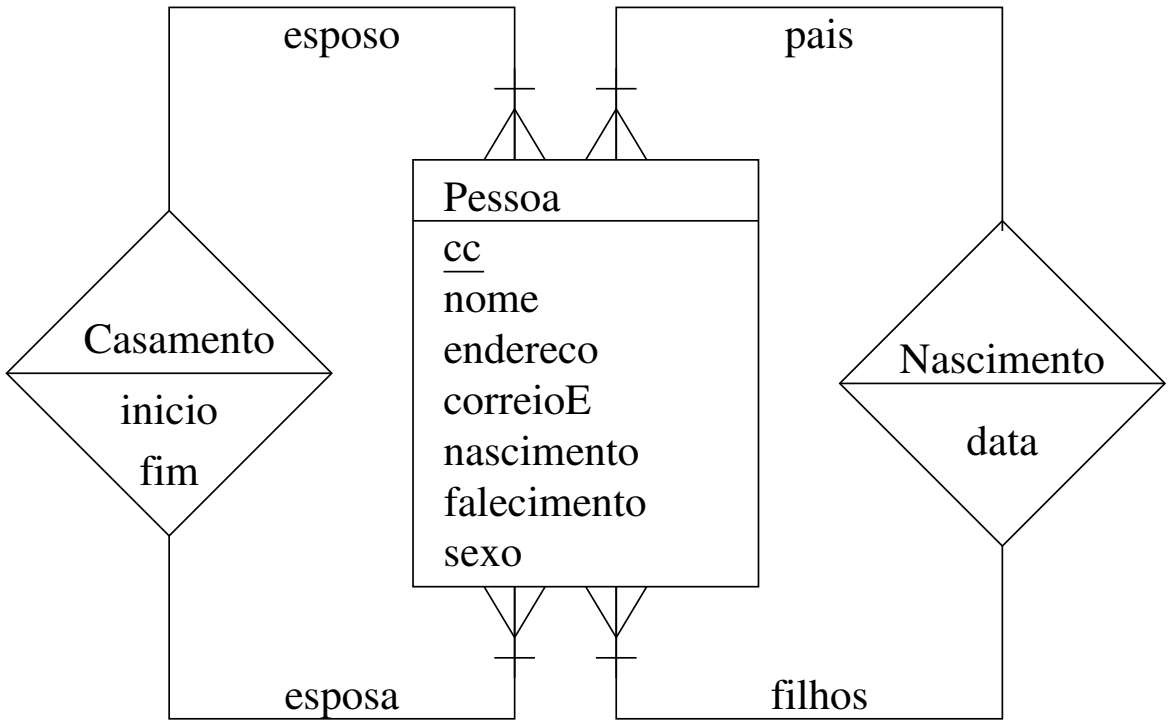
BD Grafos vs Modelo Relacional

A modelização Entidades-Associações tem como objectos de 1ª ordem Entidades e Associações.
A modelização Relacional só tem como objectos de 1ª ordem as Entidades (Relações/Tabelas).
As associações tem de ser "recuperadas" através do mecanismo de chaves externas, assim como pela junção de tabelas;

- Perde-se informação, as associações não são representadas directamente, têm de ser representadas por um mecanismo interno das bases de dados;
- A estrutura da base de dados é muito rígida, sendo difícil a sua modificação.
- Todo o tipo de informação/consultas que se baseiam nas associações são difíceis (ou mesmo impossíveis) de representar.

Solução: Bases de dados de Grafos, entidades (nós) e associações (arcos) como entidades de 1ª ordem.

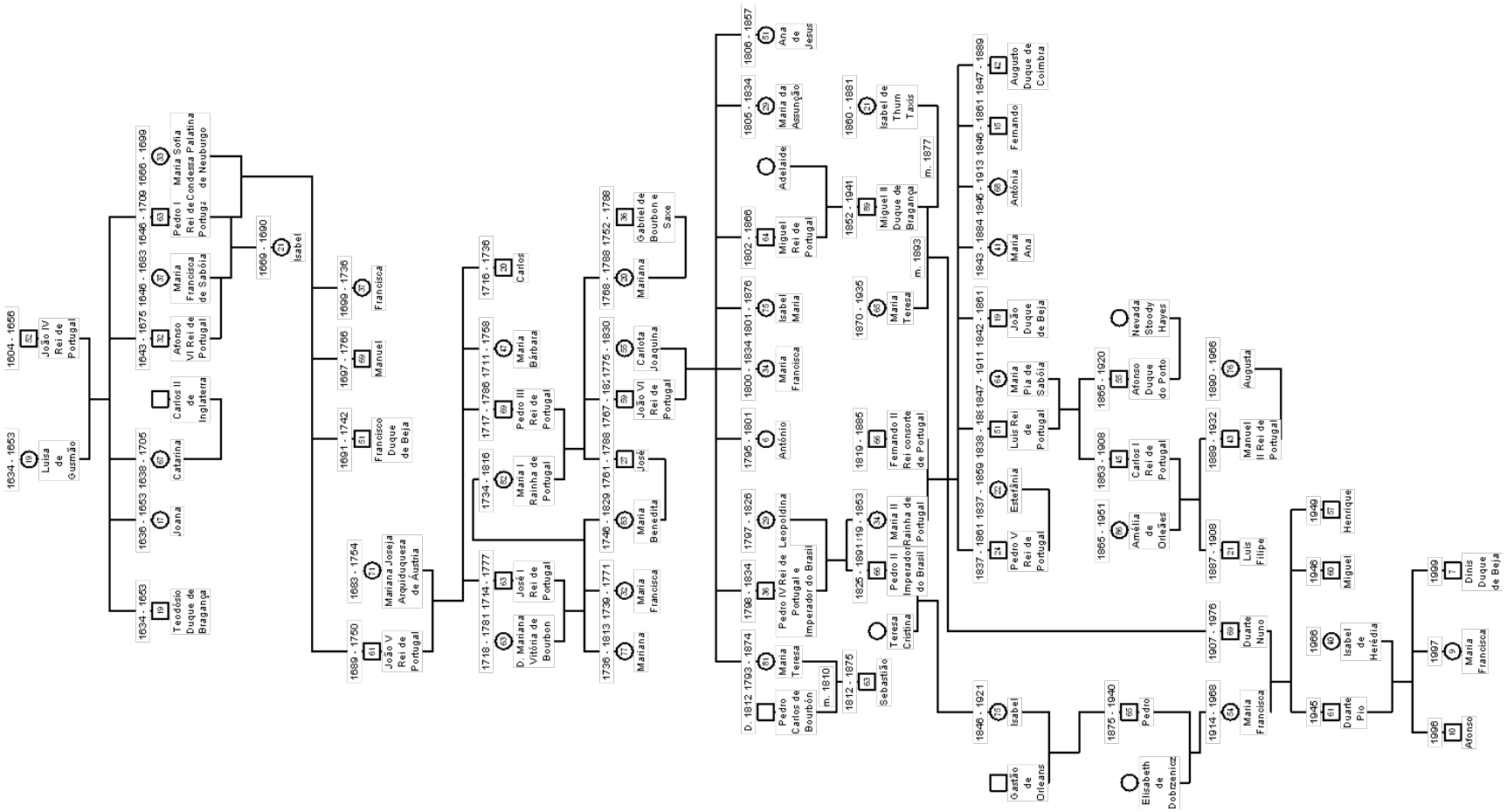
BD Grafo — Árvore Genealógica — DEA



Consultas: filhos, netos, bisnetos, . . . ; pais, avós, bisavós, . . . ; irmãos, primos, tios, . . .

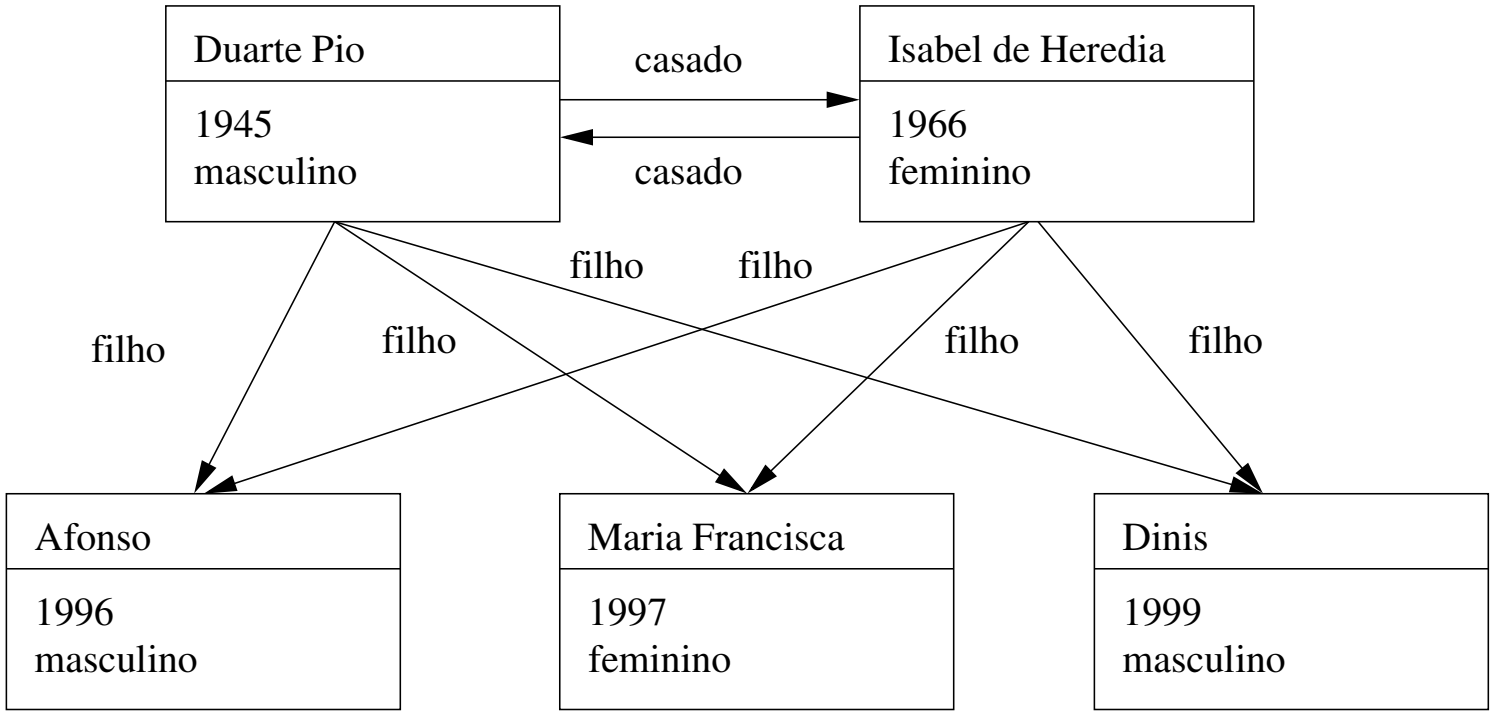
- As consultas são centradas nas associações e não na entidade;
- Há consultas recorrentes: filho do filho do filho . . .

BD Grafo — Árvore Genealógica



- As consultas recorrentes são possíveis, basta especificar caminhos no grafo.
- É possível acrescentar outras relações (e.g. irmãos) facilmente, basta acrescentar arestas adicionais ao grafo.
- As propriedades nos nós e arcos permitem guardar informação adicional.

BD Grafo — Árvore Genealógica — Grafo



- Nós – instâncias de entidades (DEA) – linhas numa tabela (MR);
- Arcos – instâncias de associações (DEA) – sem correspondência (MR).

Modelos para Bases de Dados de Grafos

Ao contrário das bases de dados relacionais, com o SQL, as bases de dados de grafos não tem uma linguagem de modelização unificadora.

PGM **P**roperty **G**raph **M**odel:

- Nós (nodes) & Arcos (edges) — Grafo;
- Propriedades (nos nós e arcos);
- Arcos dirigidos e com etiquetas.

RDF **R**esource **D**escription **F**ramework

HyperGrafos um hiper-grafo é uma generalização de um grafo em que um hiper-arco pode ligar mais do que dois nós.

O modelo (para já) mais popular é o PGM.

Bases de Dados de Grafos vs Modelo Relacional

No Modelo Relacional:

- O modelo relacional não implementa relações. As relações são representadas através dos mecanismos das chaves externas e junções.
- Muito difícil (se não impossível) consultas recorrentes: filhos de filhos de filhos ...
- Muito difícil de modificar a estrutura de uma base de dados (já com valores incluídos):
  - introduzir novas relações;
  - modificar as relações actuais.

Em contraponto as bases de dados de grafos tem relações como objectos de primeira ordem, consultas recorrentes são implementadas como percursos no grafo, a introdução, remoção de arcos não afecta a restante estrutura.

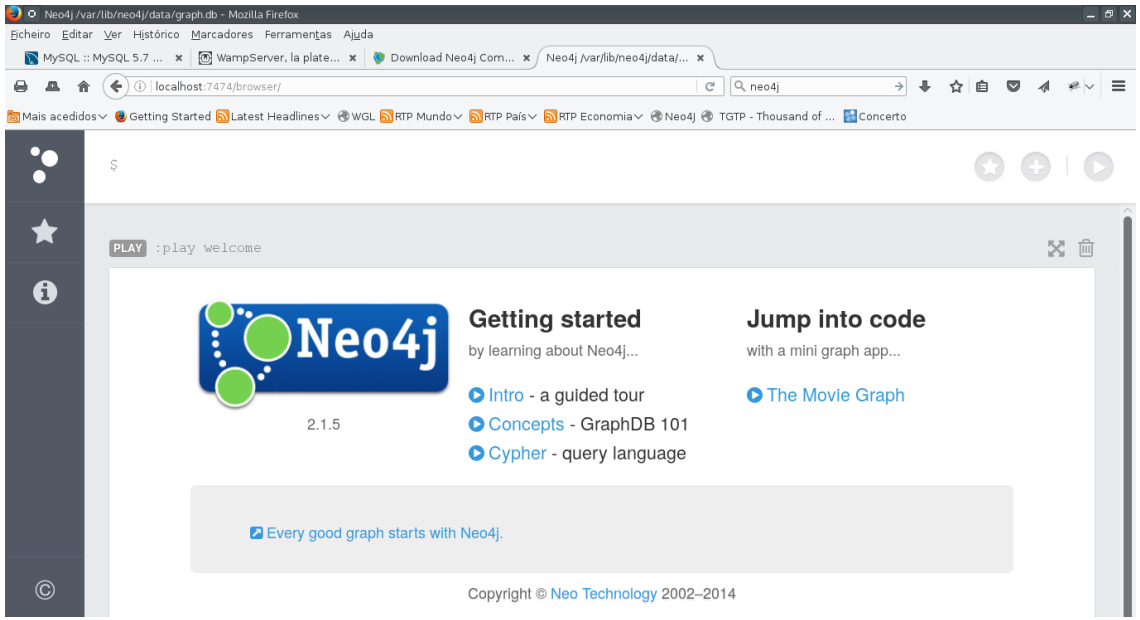
- As bases de dados relacionais estão muito mais difundidas do que as bases de dados de grafos.
- Os SGBD relacionais são mais completos, eficientes e ≪sólidos≫ que os correspondentes SGBD de grafos.

Neo4j

≪Neo4j is a highly scalable native graph database that leverages data relationships as first-class entities, helping enterprises build intelligent applications to meet today's evolving data challenges.≫

`https://neo4j.com/`

Disponível gratuitamente, para Linux, MS-Windows, Mac-OS (Neo4j Community Edition).

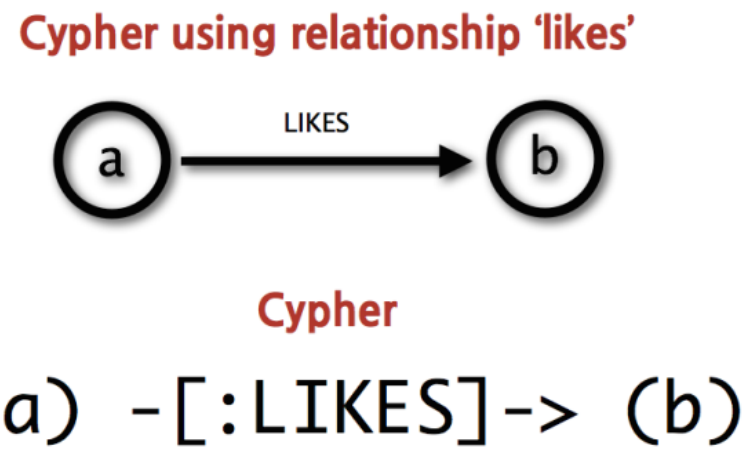


`http://localhost:7474/browser/`

Cypher — Linguagem de Consulta

«Cypher» usa «arte-ASCII» para representar padrões num grafo.

- Os **Nós** são representados como um par de parêntesis: '`()`', se é necessário referenciar o resultado posteriormente pode-se usar variáveis '`(p)`'.
- As **Relações** são representadas como setas '`-->`' entre dois nós. Pode-se colocar informação adicional, no meio da seta, usando para tal parêntesis rectos.
  - tipos de relações, por exemplo: `-[:KNOWS|:LIKE]->`;
  - nomes de variáveis: `-[rel:KNOWS]->`;
  - propriedades adicionais `-[since:2010]->`;
  - informação estrutural adicional para caminhos de comprimento variável: `-[:KNOWS*..4]->`.



Cypher — Estrutura Genérica

A estrutura genérica de uma consulta em cypher é semelhante a uma consulta em SQL, a diferença está no facto de se declararem padrões num grafo

```
MATCH (n1:Label1)-[rel:TYPE]->(n2:Label2)
WHERE rel.property > value
RETURN rel.property, type(rel)
```

- `MATCH`, define o padrão a procurar no grafo;
- `WHERE`, filtra o resultado;
- `RETURN`, define o resultado da consulta.

\+ `START` para definir um ponto de partida.

Cypher — Comandos

DDL — Comandos para manipular o grafo.

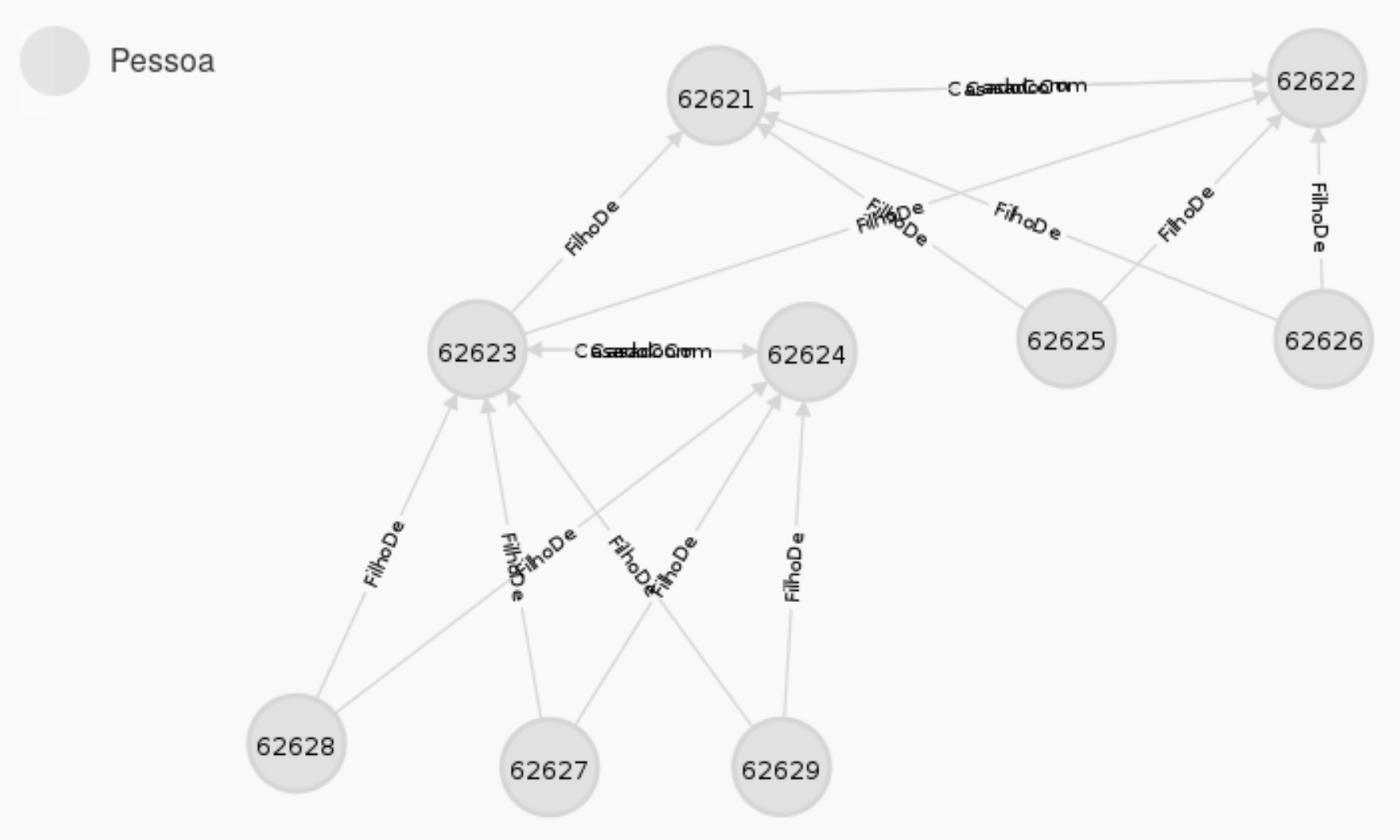
- `CREATE`/`CREATE UNIQUE`, cria nós/arcos;
- `DELETE`, apaga nós/arcos;
- `SET`, fixa o valor de uma propriedade;
- `FOREACH`, precorre uma lista de nós;

DML — Comandos para consultar o grafo.

- `START`, define um ponto de partida;
- `MATCH`, define o padrão a procurar no grafo;
- `WHERE`, filtra o resultado;
- `RETURN`, define o resultado da consulta;
- `UNION`, união de duas consultas;
- `WITH`, sequencia duas consultas (tipo «pipe» em Unix);

...

Neo4j/Cypher um Exemplo



Um Exemplo — Criar o Grafo

```
// Criar os nós (entidades): Pessoa(nome,sexo,dataNascimento,dataFalecimento)

CREATE (n1:Pessoa {nome:'Maria Francisca', sexo : 'Feminino',
        dataNasc: 1914, dataFal : 1968 })
CREATE (n2:Pessoa {nome:'Duarte Nuno', sexo : 'Masculino',
        dataNasc: 1907, dataFal : 1976 })

// Criar as arestas (relações): CasadoCom(dataCasamento,dataDivorcio)

MATCH (a:Pessoa),(b:Pessoa)
WHERE ID(a)=62621 AND b.nome = 'Duarte Nuno'
CREATE (a)-[r1:CasadoCom  dataCas:'1942-10-15' ]->(b)
CREATE (b)-[r2:CasadoCom  dataCas:'1942-10-15' ]->(a)
RETURN r1,r2

// Criar as arestas (relações): FilhoDe

// Filhos de Duarte Nuno e Maria Francisca
MATCH (a:Pessoa),(b:Pessoa),(c:Pessoa)
WHERE a.nome = 'Duarte Pio' AND b.nome = 'Duarte Nuno' AND ID(c)=62621
CREATE (a)-[r1:FilhoDe]->(b)
CREATE (a)-[r2:FilhoDe]->(c)
RETURN r1,r2
```

Um Exemplo — Consultas

- Consulta de um nível (a um só arco de distância).

```
// Pais de Afonso
MATCH (n { nome:'Afonso'})-[r:FilhoDe]->(p)
RETURN p
```

- Consulta a vários níveis (definindo um caminho no grafo).

```
// Pais e  Avós de Afonso (r:typeRel*minHop..maxHops]
MATCH (n { nome:'Afonso'})-[r:FilhoDe*1..2]->(p)
RETURN p
```

- Consulta com vários padrões.

```
// Filhos do casal: Duarte Nuno e Maria Francisca
MATCH (p)-[r1:FilhoDe]->(n1), (p)-[r2:FilhoDe]->(n2)
WHERE n1.nome = 'Duarte Nuno' AND ID(n2)=62621
RETURN p
```

Visão Global Sobre a Concepção de Bases de Dados

Dado um problema em que se quer manipular informação persistente, põe-se a questão da utilização de um Sistema de Gestão de Bases de Dados (SGBD) para a gestão da referida informação.

A concepção de uma solução para um problema de organização da informação, começa pela modelização do problema num Diagrama Entidade-Associação (DEA), e a sua posterior conversão num dado modelo de bases de dados. O passo final passa pela implementação do modelo escolhido num dado SGBD.

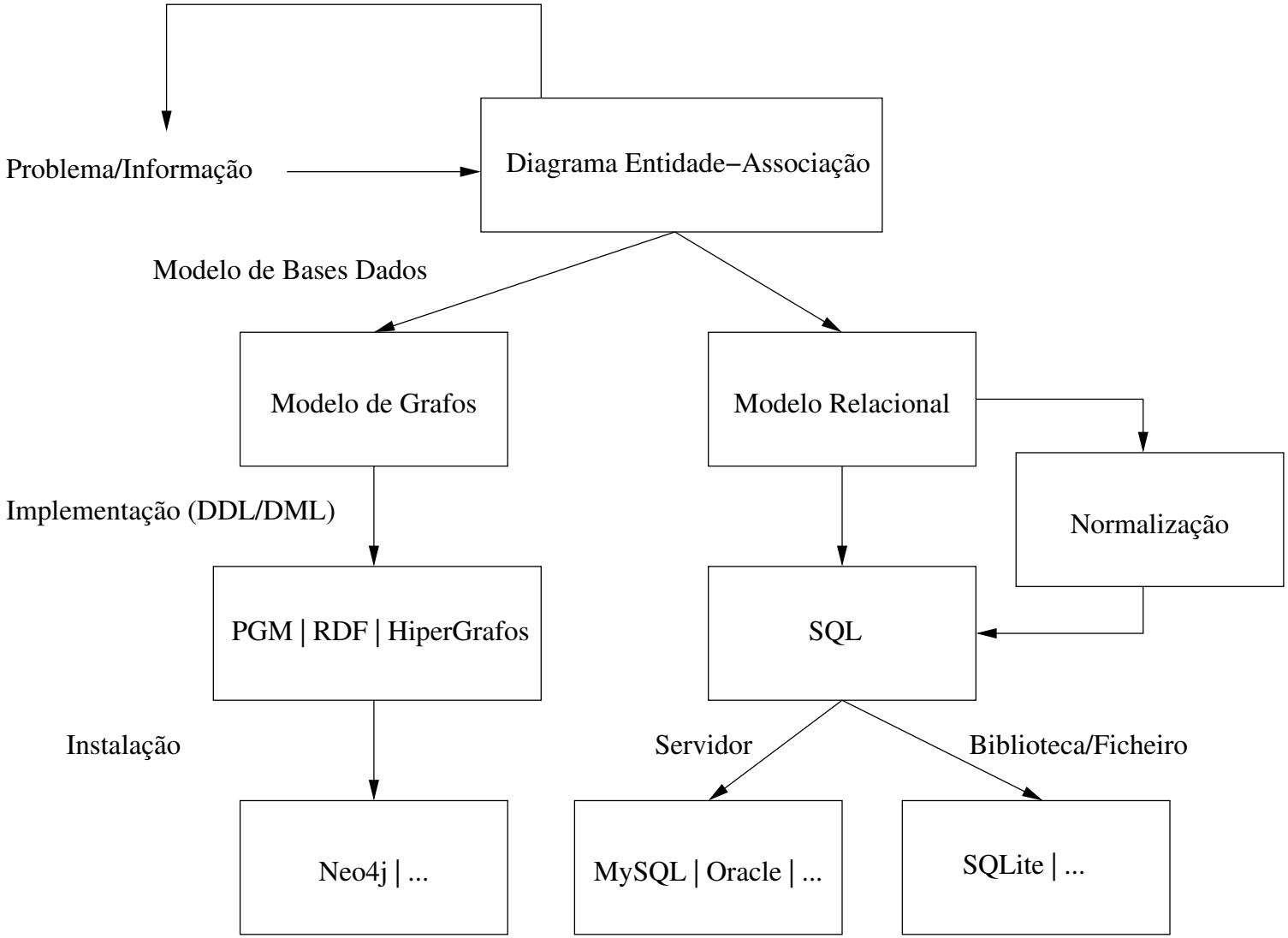
Modelização construção do DEA;

Modelos de Bases de Dados transformar o DEA num esquema relacional ou num esquema de grafo;

Implementação implementar o modelo utilizando uma linguagem de definição e manipulação de dados (DDL/DML);

Instalação escolher o SGBD apropriado para o problema em questão.

Concepção/Modelização/Implementação/Instalação



Bases de Dados Relacionais

Temos assumido que o esquema R é dado:

- R pode ter sido obtido ao passar um diagrama entidade-associação para tabelas;
- R pode ser uma única relação contendo todos os atributos de interesse para os dados (relação universal). A normalização há-de decompor R em relações mais pequenas;
- R pode ser o resultado de algum design "ad hoc".

Quando o diagrama DEA foi concebido de forma cuidadosa, o esquema gerado pode já estar numa dada forma normal. Bastará nesse caso uma simples verificação para constatar se isso é verdade ou não.

Na prática, poderão ocorrer DEAs imperfeitos que levam a que dependências que queremos impor não tenham o lado esquerdo como chave.

Por exemplo entidade `Empregado` com atributos `codDepartamento` e `moradaDep`, e a dependência $\texttt{codDepartamento} \rightarrow \texttt{moradaDep}$.
Num bom esquema `Departamentos` seria um outro conjunto de entidades.

Em algumas situações deste tipo a normalização permite corrigir situações incorrectas. Noutras ter-se-à de re-desenhar o esquema.

Instalação de uma Base de Dados Relacional

A escolha do sistema de gestão de base de dados relacional a usar vai depender de:

Qualidade o SGBD implementa transacções e chaves externas;

Robustez servidor (com gestão de utilizadores, etc.) ou biblioteca e um ficheiro contendo a base de dados;

Manutenção para soluções comerciais a manutenção da base de dados pode ser significativa.

No caso das bases de dados de grafos a escolha é mais complexa dado haver um leque mais alargado de escolhas. Ver em `https://en.wikipedia.org/wiki/Graph_database` uma lista de possíveis escolhas.