

UNIVERSIDADE FERNANDO PESSOA
Computer Engineering
DATABASE MANAGEMENT SYSTEMS
24/10/2016

Individual exam, access to personal notes allowed. 2h. Read carefully before answering. Unreadable answers will be discarded. Good luck. Sign your answer sheets please.

1. You are asked to design a database for a small clinic. The clinic has patients, appointments and doctors. A patient can have several appointments with doctors. An appointment takes place in a date, hour and room. There can be exams linked to an appointment. An exam can only be linked to one appointment. A patient may have health insurance. A doctor, or patient, cannot have overlapping appointments.
 - a. Identify the functional dependencies.
df1 {appointment}→{date, hour, patient, doctor, room}
df2 {exam}→{appointment}
df3 {date, hour, patient}→{appointment}
df4 {date, hour, doctor}→{appointment}
df5 {date, hour, room}→{appointment}
 - b. Propose the relations and the keys to have 3FN/BCNF.
Exam(exam, appointment) from df2

Appointment(patient, doctor, date, hour, appointment, room) from df1, df3, df4 e df5. All determinants are PK, the relation is in BCNF (df3, df4 and df5 have common attributes)
 - c. Add the constraint that a doctor cannot have more than 3 appointments in the same day.
Could use a trigger to fire in INSERT, UPDATE in APPOINTMENT.
2. Consider relation R = PROPERTY{ID, NUMBER, VENDOR_ID, NAME, CONTACT, ZONE_ID, PRICE, DATE_REGISTERED} with F={VENDOR_ID→{ZONE_ID, CONTACT, NAME}, CONTACT→{VENDOR_ID, NAME}, ID→R, NUMBER→R}
 - a. Identify the candidate keys.
ID and NUMBER are CK (they determine R).
 - b. Identify the normal form of the relation, and if necessary normalize to BCNF.
It's not in 3FN. Normalization: PROPERTY {ID, NUMBER, PRICE, DATE}, VENDOR {ID, ZONE, NAME, CONTACT}, SELLS {PROP_ID, VENDOR_ID}
3. Given R{A, B, C, D, E, F, G}, A→B, AB→C, BC→A, AC→D, ED→F, EF→G, AG→E, identify the candidate keys and normalize if necessary to BCNF.
All attributes are in both sides of the dependencies. All have to be tested. For example A+ = ABCD, BC+ = ABCD, ED+ = EDFG. Eliminating unnecessary tests, we find the candidate keys AE, AG, BCE, BCG.
4. Consider ROOM{ID, NUMBER, TYPE}, CLIENT{ID, NAME, TYPE}, RESERVATION{ROOM_ID, CLIENT_ID, DATE_IN, DATE_OUT}, and write in relational algebra:
 - a. List the rooms occupied in December.
 $\pi_{id, number} \sigma_{date1 \geq 1\text{-dec and } date2 \leq 31\text{-dec}}(ROOM \bowtie RESERVATION)$
 - b. List the type 2 rooms not occupied in October.
 $\pi_{id} (ROOM) - \pi_{room.id} \sigma_{date1 \geq 1\text{-dec and } date2 \leq 31\text{-dec}}(RESERVATION)$
 - c. List the clients that have reserved room 110.
 $\pi_{client.id, name} \sigma_{room_id=110} (CLIENT \bowtie RESERVATION)$
5. Suppose the columns ROOM.TYPE, DATE_IN and DATE_OUT are often used.
 - a. What type of indexes (primary, secondary, grouped,...) do you suggest?
Although TYPE is often used, it may not be advised to create an index. It could be a hashing index. Date1 and date2 may be a B+-Tree, and it can be used in queries involving (date1, date2).
 - b. Suppose most tuples in CLIENT have TYPE=1; what type of index do you suggest, and

for what type of queries?

A partial index, to be used in queries with `TYPE !=1`. For `type=1` queries it would not be used.

c. Can you identify a composed index for `DATE_IN` and `DATE_OUT`? For what queries?

See answer a). It can be used for range queries with dates, or for queries with `Date1` (if the order in the index is `date1` and `date2`).

6. Comment the following indexes, adding examples of queries if relevant:

- a. Clustered index in "room(id)". Follows from definition of PK. If ID is PK, the index is clustered.
- b. Non-clustered index in "room (number)". If there are queries using `NUMBER`, the index may be useful; it would be non-clustered.
- c. Clustered index in "client(name, type)". It is only clustered if (name, . type) is the PK which is not likely.
- d. Hashing index in "room (type)". If there are not many room types the usefulness of the index is questionable.

7. Comment the following statements, saying if true or false:

- a. An hashing index can be sequentially scanned by order. F, there is no ordering in hashing.
- b. A B+-Tree cannot be used for counting. F, leaves can be easily counted.
- c. Indexing foreign keys is useless. F, it may be useful for joins, and for testing on delete cascade.
- d. Using an index has no costs. F, an index has to be updated, visited, and managed in main memory

O Modelo E-R

Feliz Ribeiro Gouveia
Universidade Fernando Pessoa
fribeiro@ufp.edu.pt

Introdução

- Fases do projeto de uma Base de Dados:
 - Levantamento de requisitos
 - Projeto conceptual
 - Projeto lógico
 - Projeto físico

Levantamento de requisitos

- Tarefa complexa que consiste em conhecer o problema real, e em identificar as necessidades dos diferentes intervenientes
- Existem requisitos do utilizador, funcionais, não-funcionais, de equipamento, de software,...
- Técnicas incluem observação, questionários, entrevistas, prototipagem
- Após conhecer o problema, constrói-se um modelo (E-R, O-O, UML,...)

Levantamento de requisitos

- Os requisitos devem ser comunicados entre todos, negociados, revistos,....
- Correspondem a interpretações de uma realidade, podem ter perspetivas diferentes
- São muitas vezes interdisciplinares
 - “o cliente” pode ter uma interpretação fiscal, marketing, comercial,...
- É uma das fases mais complexas e delicadas do projeto informático
 - Alguns erros são descobertos muito tarde...na implementação

Porquê um modelo?

- A fase de levantamento de requisitos implica:
 - Lidar com vários interlocutores
 - Enfrentar realidades desconhecidas e distintas
- Existe a necessidade de
 - Documentar sem ambiguidade o que o cliente quer
 - Juntar suporte documental ao contrato do projeto
 - Importante, mitiga litígios
 - Registrar e seguir revisões
 - Comunicar a outras equipas o que se pretende
 - Por exemplo à equipa de Projeto Lógico

Porquê um modelo?

- Utilizar apenas texto para documentar tem os seus inconvenientes
 - Dificuldade de interpretação de alguns termos (o que significa “deve”, “pode”,...?)
 - Incómodo para usar como material de referência (podem ser textos longos)
 - Incómodo para partilha e trabalho colaborativo
 - É necessário muito texto mesmo para projetos de média dimensão
- Apareceram os modelos gráficos

Modelo E-R

- 1976: Chen Peter Pin-Shan publica o modelo Entidade-Relacionamento (E-R)
- Conceitos base:
 - Entidades
 - Relacionamentos
 - Atributos (de entidades e de relacionamentos)
- Os relacionamentos podem envolver:
 - 1 entidade (unários)
 - 2 entidades (binários)
 - N entidades (N-ários)

Conceitos base

- O mundo é constituído por **entidades**
- As entidades podem ser agrupadas por **tipos**
 - economia de representação
 - Todos os clientes são agrupados na entidade CLIENTE
- Uma entidade tem uma **identidade** única
 - Permite distingui-la das outras
- Uma entidade tem **propriedades**
- Uma entidade pode ser ligada a outras entidades por **relacionamentos**

O Modelo E-R

- Exemplo de Chen:

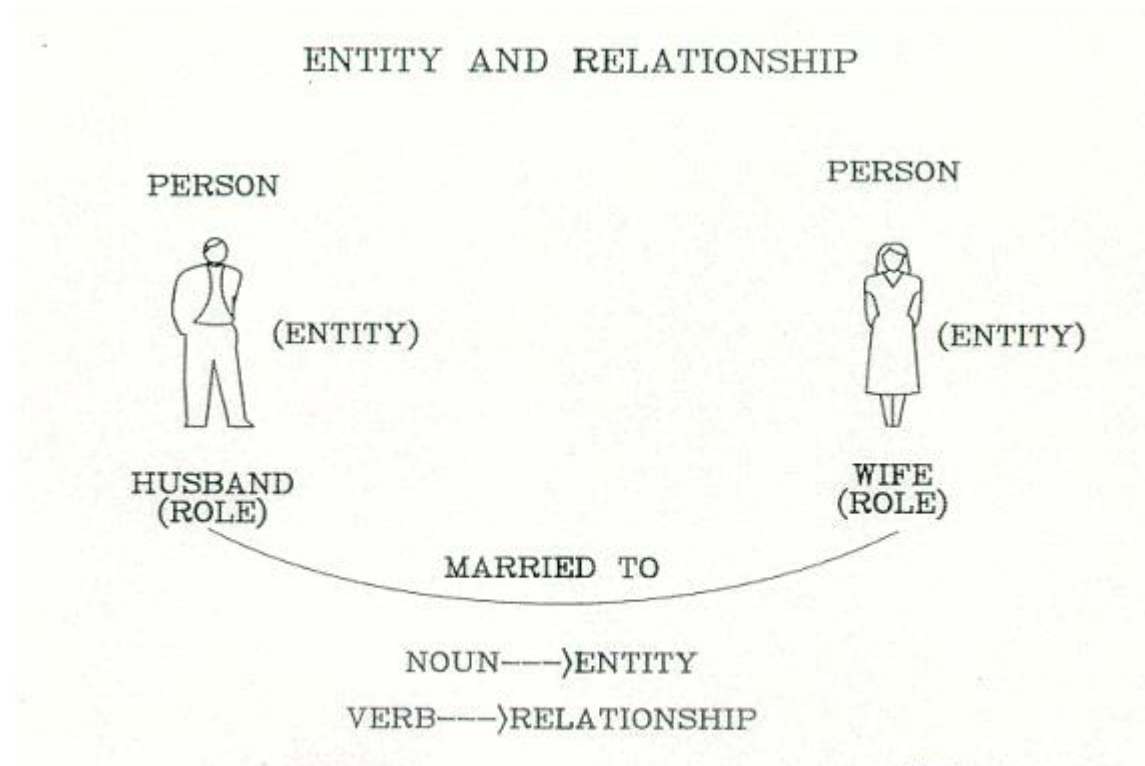


Fig. 1. The Concept of Entity and Relationship

Modelo E-R

- Representa apenas aspectos estáticos (as entidades e os seus relacionamentos, e as propriedades de ambas)
- Não representa aspectos dinâmicos (como fluxos de informação, ou restrições)
- É simples de utilizar, mesmo por não-técnicos, e revela-se bastante útil para a conceção de esquemas de BD
- Como veremos mais tarde, o mapeamento deste modelo concetual para o modelo relacional é muito simples

Modelo E-R

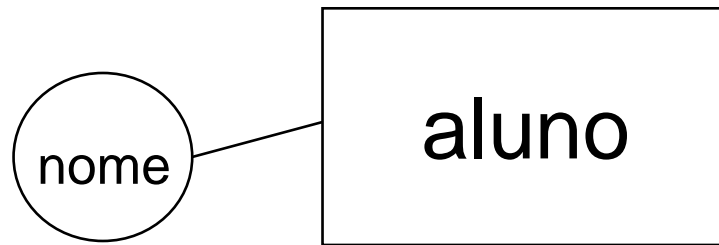
- Pertence à classe de modelos ditos “semânticos”
- Destina-se a melhorar o conhecimento do utilizador sobre os dados
- Aumenta a expressividade semântica dos dados
- Contrói-se com metodologias descendentes (parte-se do mais abstrato para o mais específico)
- Existem muitas ferramentas para utilizar este modelo
 - Gerem modificações, documentação, e facilitam a criação do modelo relacional

Conceitos (1)

- Entidade: qualquer objecto identificável (ex: *aluno*, *curso*,...). As entidades têm instâncias
- Propriedade: um dado sobre uma dada entidade (ex: *data nascimento*, *nome*, *peso*,...)
 - Os dados não devem ser calculados: por exemplo, em vez de *idade* deve-se usar *data de nascimento*
- Relacionamento: entidade que serve para ligar duas ou mais entidades (ex: *frequenta*)
 - Um relacionamento pode definir um “papel” quando lido num dado sentido (geralmente não é necessário)
- Subtipo: ex: *estudante* é subtipo de *pessoa*

Conceitos (2)

- Representamos uma entidade com um retângulo



- Uma das propriedades da Entidade deve ter valores únicos (serve de identificador)
 - Pode ser uma propriedade natural (NIF) ou artificial (código)
 - Se existir mais do que uma propriedade única, tem de se escolher uma

Conceitos (3)

- Algumas entidades “dependem” de outras entidades para existir
 - É “dependente”, isto é só pode ser identificada se outras o forem
- Chamam-se “entidades fracas” e usamos uma moldura dupla



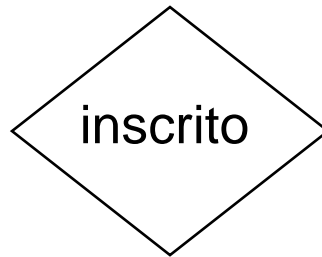
dependente

Conceitos (4)

- Exemplos de entidades fracas:
 - As turmas de uma disciplina (não faz sentido ter uma turma se a disciplina não existe)
 - A oferta de uma disciplina num dado ano letivo (não faz sentido se a disciplina não existe)
 - Um recibo (tem que ter uma fatura associada, um cliente)
 - Uma consulta (tem que ter um paciente, um médico)

Conceitos (5)

- Representamos um relacionamento com um losango

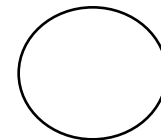
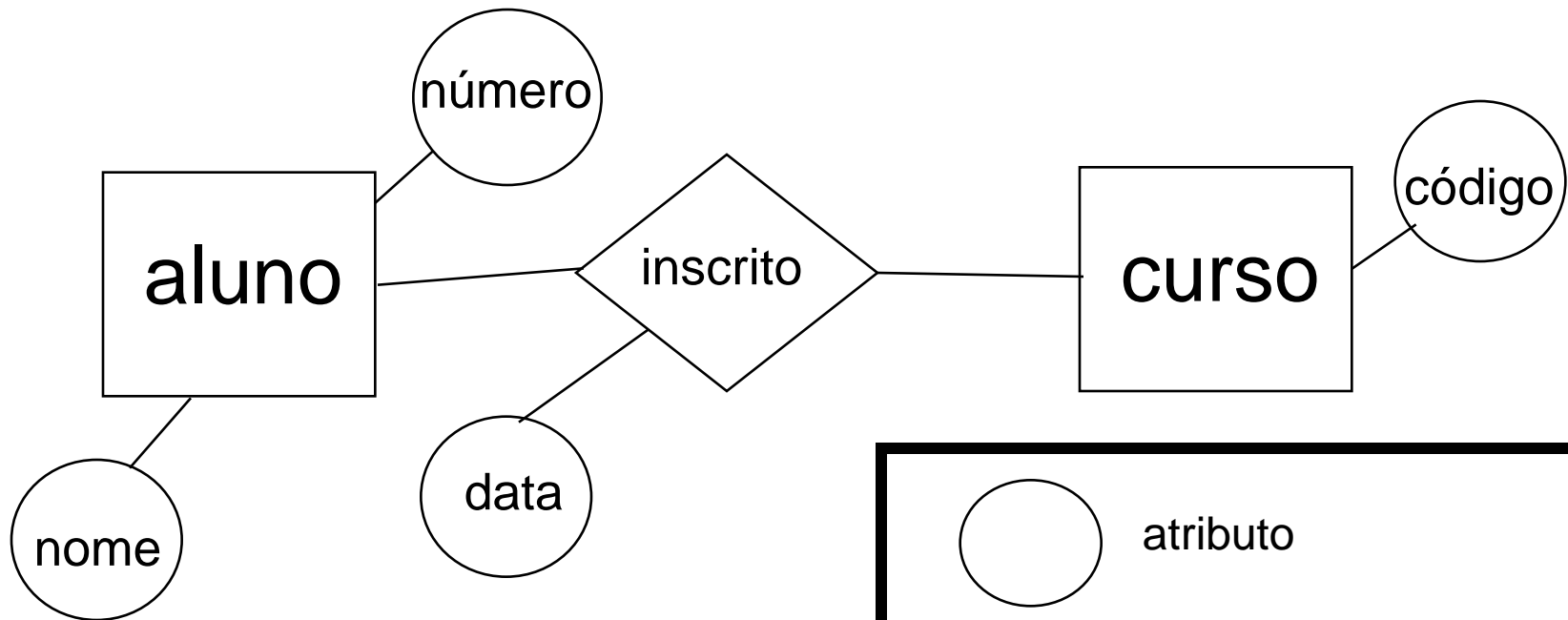


- As entidades podem participar em vários relacionamentos
- Um relacionamento tem um grau (quantos tipos de entidades envolvem)
- Um relacionamento tem multiplicidade

Construir o diagrama

- As entidades e os seus relacionamentos podem ser identificados a partir de textos, entrevistas, conhecimento prévio do problema
- Numa descrição textual deve-se ter atenção ao sujeito e aos complementos (geralmente são as entidades) e ao verbo (geralmente são os relacionamentos)
- Deve-se descrever o problema com uma linguagem coerente, mantendo o mesmo estilo
 - Facilita a leitura e a compreensão

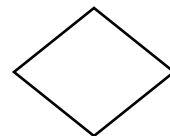
Diagrama E-R



atributo



entidade



relacionamento

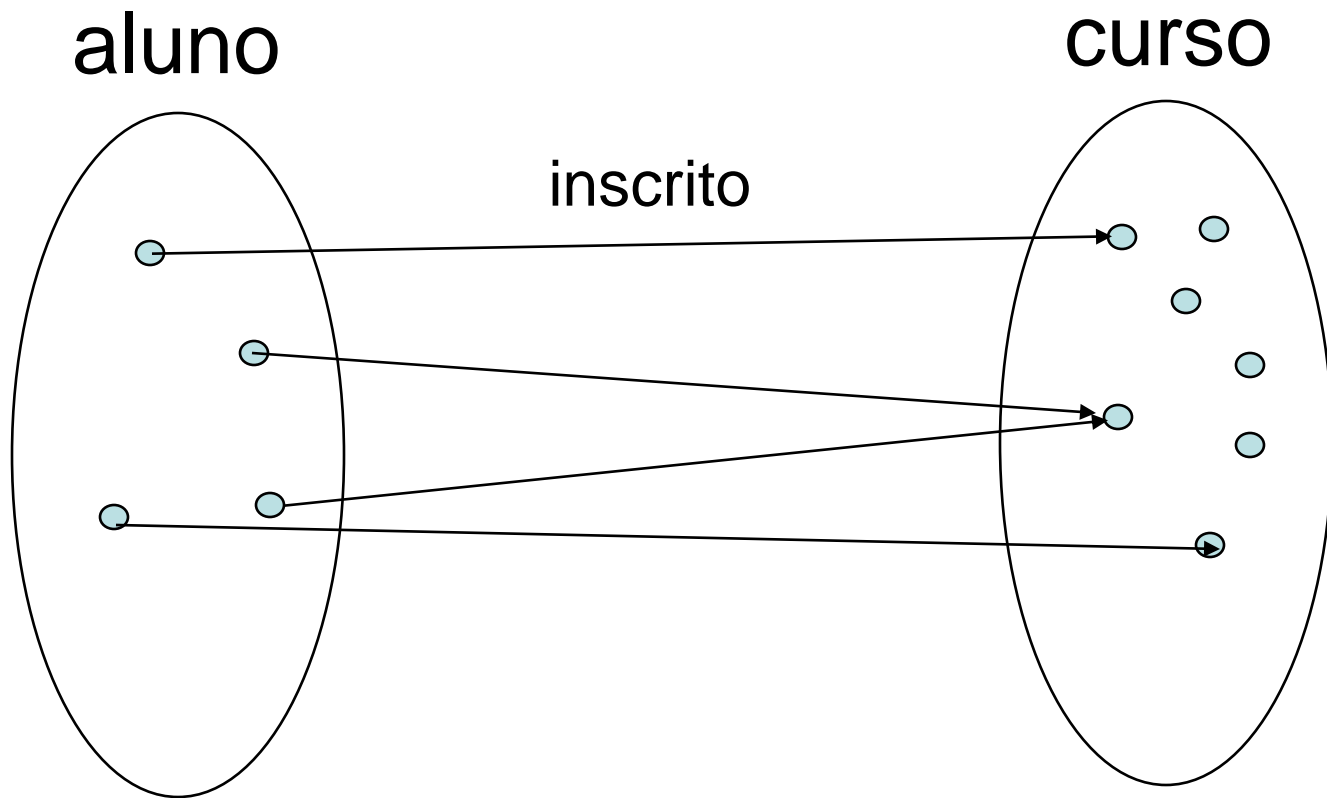
Interpretação

- Existe um relacionamento “inscrito” entre as entidades “aluno” e “curso”
- Esse relacionamento é caracterizado pelo atributo “data”
- As entidades “aluno” e “curso” têm os atributos mostrados

Multiplicidade

- Os relacionamentos têm uma multiplicidade, que especifica o número de elementos de cada entidade que participam neles
- Sabe-se que um “aluno” só se pode inscrever em um e só um “curso”
- Sabe-se que num “curso” se podem inscrever nenhum ou N alunos
- Na prática define se um relacionamento é obrigatório ou opcional

Graficamente

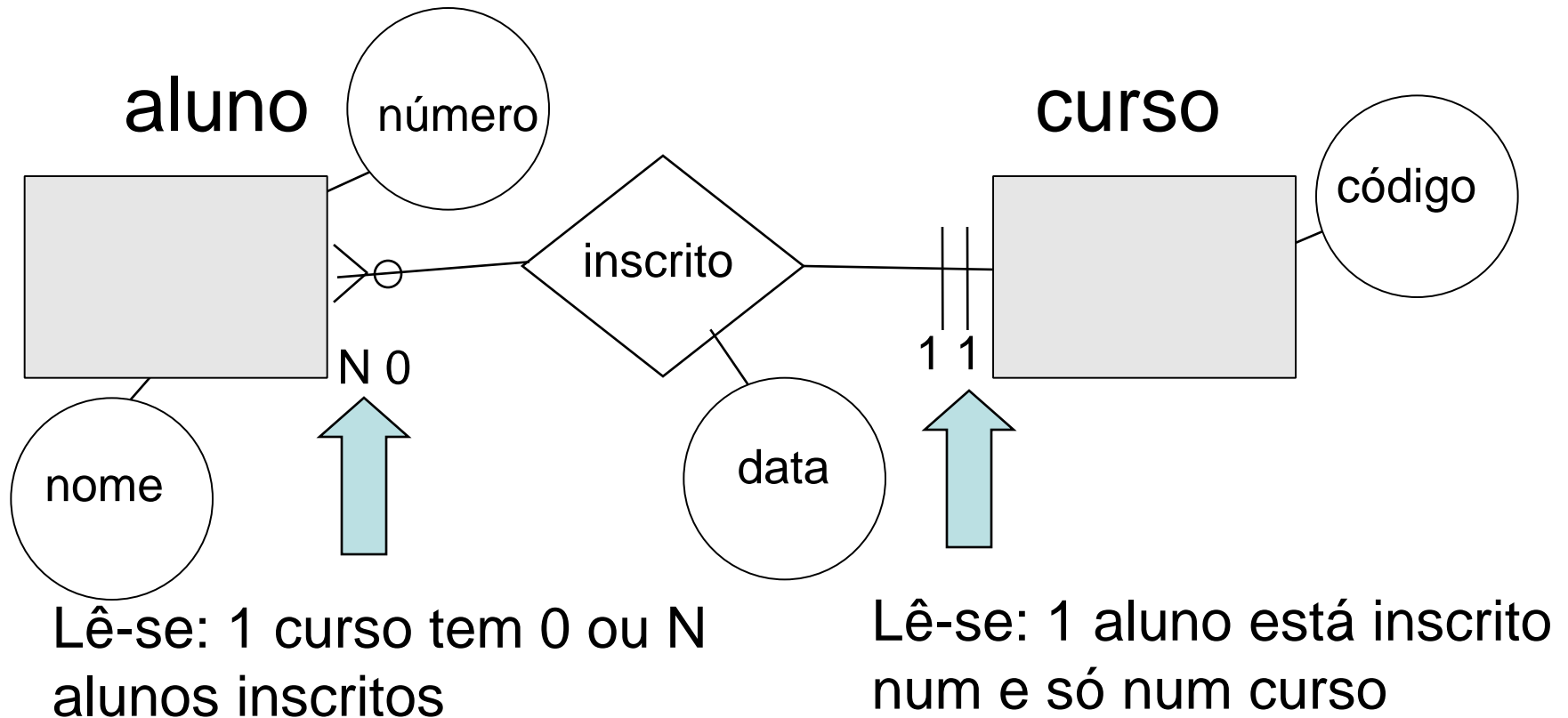


Dos conjuntos depreende-se

- Um aluno está inscrito no máximo em 1 curso, e no mínimo em 1 curso
- Um curso tem no mínimo 0 inscritos, e no máximo N alunos inscritos

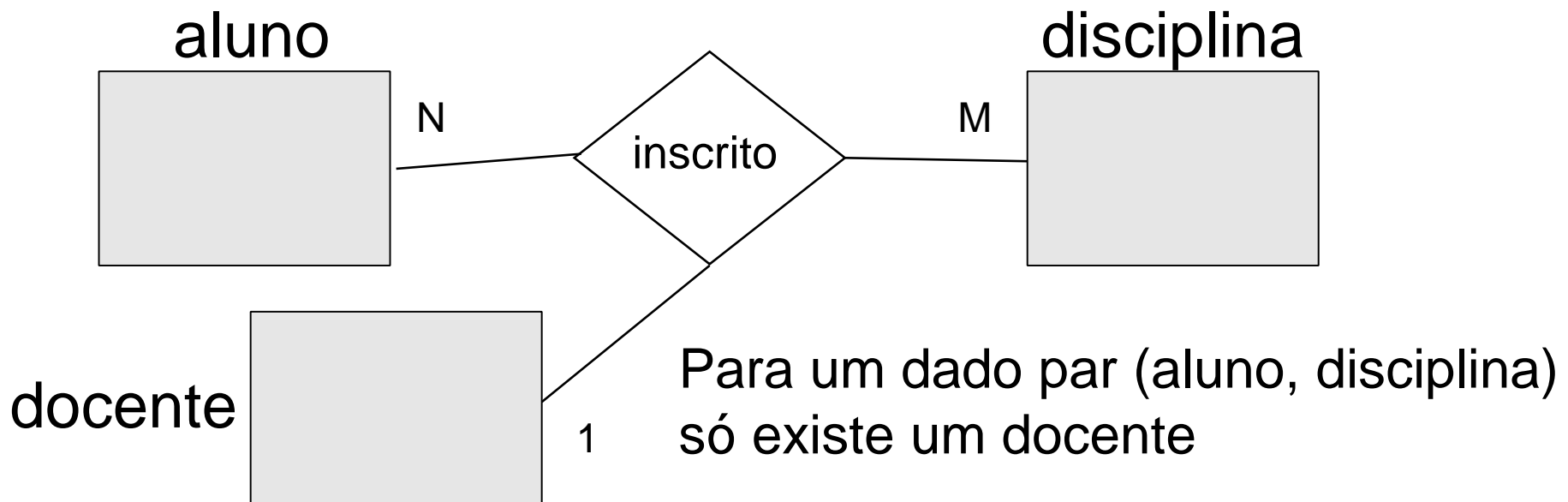
Para cada relacionamento tem que se identificar a multiplicidade máxima e mínima

Diagrama E-R



Relacionamentos ternários

- A multiplicidade deve ser representada de forma diferente
- Para cada ocorrência de um par de entidades, quantas ocorrências podem existir da outra?



Opções

- O modelo E-R é subjetivo (o mundo é subjetivo!)
- Há sempre alternativas a um dado diagrama
 - A experiência dita as escolhas
- As escolhas mais comuns incidem sobre:
 - Criar Entidades ou Atributos?
 - Criar Entidades ou Relacionamentos?

Exemplo

- Uma AULA é uma entidade ou um relacionamento?
 - Envolve DOCENTE, TURMA, SALA, DISCIPLINA, HORA_INICIO, DURAÇÃO
- Um ENDEREÇO é um atributo ou uma entidade?
 - Envolve RUA/AV, NUMERO, CODIGO_POSTAL, LOCALIDADE

O Modelo Relacional

Feliz Ribeiro Gouveia
fribeiro@ufp.edu.pt
UFP

Modelo Relacional de dados

- Um modelo refere-se a três aspectos fundamentais dos dados:
 - a sua estrutura
 - a sua integridade
 - a sua manipulação
- O Modelo Relacional (Ted Codd, IBM, 1969) admite **relações** como única estrutura

Relações

- Uma relação é definida pela sua estrutura (esquema)
- O esquema de uma relação é o conjunto do nome dos seus atributos (colunas)
- Todos os tuplos de uma relação têm todos os atributos do esquema
- Uma base de dados define-se como um conjunto de esquemas

Relações: exemplo

aluno {numero, nome, apelido, morada}

- Os valores possíveis de uma relação são a sua extensão:
 - {1000, Luis, Sousa, R. Direita}
 - {1010, Ana, Ribeiro, R. de Cima}
- Se for necessário, escreve-se:
 - {numero:1000, nome:Luis, apelido:Sousa, morada:R. Direita}
- Na prática, torna-se mais simples

Propriedades de um sistema relacional

- Só existem tabelas para representar os dados
 - Logicamente, não fisicamente
- Os valores dos atributos são todos explícitos
 - Não existem apontadores
- Todos os valores são atômicos (escalares)
 - Não há grupos de valores (Grupos de Repetição)

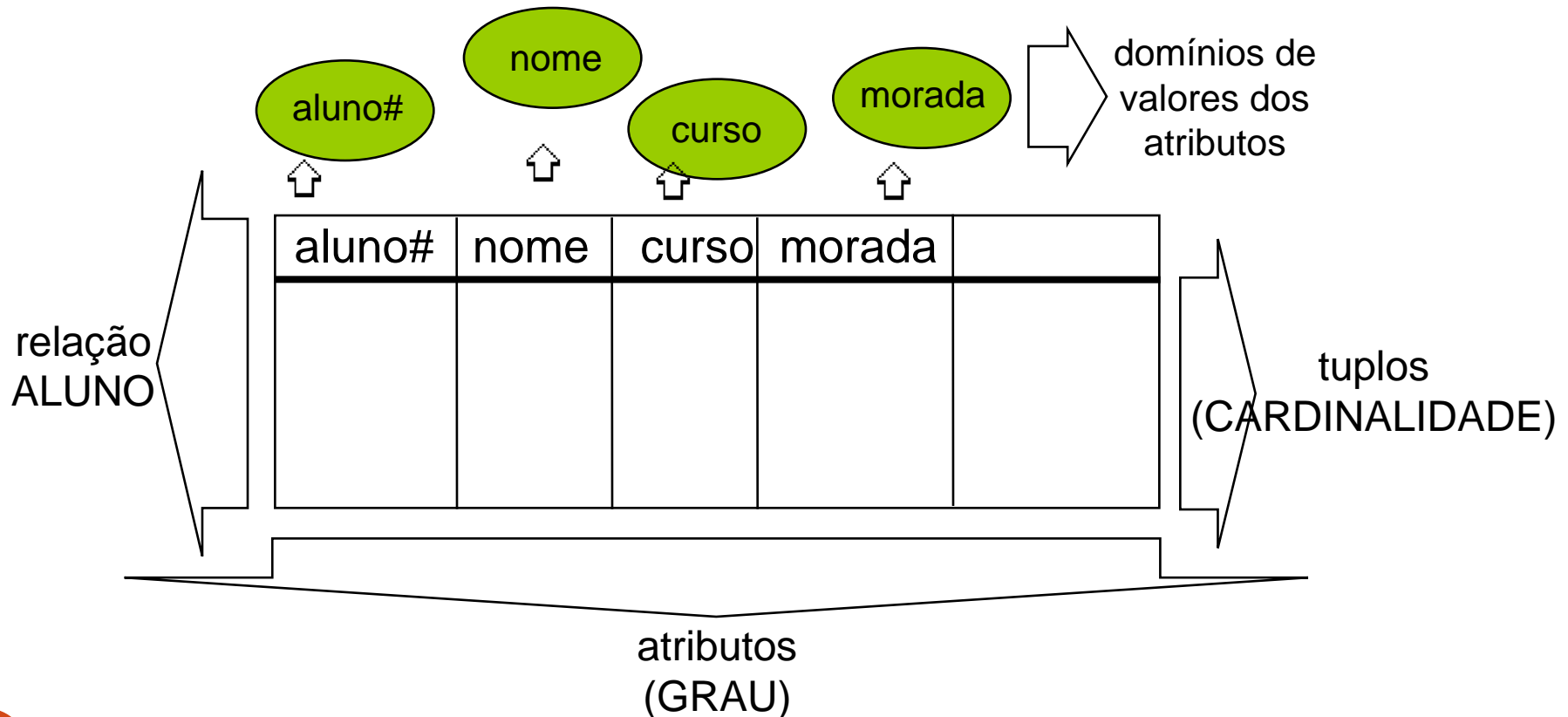
Mais propriedades

- A ordem das linhas não interessa
 - o acesso não é por posição
- A ordem das colunas não interessa
 - todas as colunas têm um nome diferente
- Não podem existir linhas duplicadas
 - é sempre possível distingui-las pelos valores de algumas das células

Terminologia relacional

- Relação => Tabela
- Tuplo => registo, linha
- Atributo => campo, propriedade, coluna
- Domínio: conjunto de valores admissíveis para os atributos de uma relação

Terminologia (cont)



Regras de integridade

- Do domínio: uma coluna só pode assumir valores do seu domínio
- Chaves: existe um conjunto de atributos único para cada tuplo: superchaves
 - Chave candidata: superchave que não pode ser reduzida
 - Chave primária: uma das chaves candidatas
 - Chave estrangeira: conjunto que é chave noutra relação
- De Entidade: a chave primária nunca é nula

Regras de integridade

- Gerais: dependem da aplicação
 - Podem ser definidas na base de dados
 - Exemplo: custo > 0, quantidade > 0
- As restrições gerais e de domínio podem ser implementadas com CHECK e CREATE ASSERTION
 - CHECK efetua verificações numa tabela
 - ASSERTION pode envolver várias tabelas
 - Mas não é implementada em quase nenhum SGBD...

Regras de integridade

- As restrições são verificadas de cada vez que uma linha é inserida ou alterada
- Por vezes é útil diferir a verificação (problema do “ovo e da galinha”: inserimos numa tabela o valor que devia existir noutra, e como não existe ainda não podemos inserir)
- Ao criar uma restrição podemos definir se queremos verificação diferida
 - a verificação de NOT NULL e CHECK não pode ser diferida

Regras de integridade

- Podemos especificar ao criar a restrição:
 - deferrable (not deferrable): significa que a verificação da restrição pode ser diferida para o fim da transação, usando SET CONSTRAINTS dentro de uma transação
 - initially deferred (initially immediate): significa que a verificação da restrição é diferida para o fim da transação
- Uma transação começa por BEGIN e termina por COMMIT (confirmar) ou ROLLBACK (anular)

Chaves

numero	nome	apelido	morada	
1000	Luis			
1010	Ana			
1020	Paulo			

Chave primária:
{numero}

Exemplo

- CREATE TABLE aluno (id serial PRIMARY KEY, nome text, apelido text, data_nasc date, genero char(1) CHECK (genero = 'F' OR genero = 'M'));
- Definem-se duas restrições nesta relação:
 - PRIMARY KEY: chave primária
 - CHECK: restrição de domínio
- Outro tipo de restrições
 - NOT NULL: coluna que não admite valores nulos
 - UNIQUE: coluna que não admite valores repetidos
 - REFERENCES : coluna é chave estrangeira

Exemplo (1)

- CREATE TABLE matricula (aluno_id int REFERENCES aluno(id), disciplina_id int REFERENCES disciplina(id), PRIMARY KEY(aluno_id, disciplina_id);
- CREATE ASSERTION tamanho_turma CHECK (**NOT EXISTS** (SELECT COUNT(*) FROM turma GROUP BY disc_id HAVING COUNT(*) > 25));
 - Sempre que se altera TURMA , verifica que o número de inscritos não ultrapassa os 25
 - A maioria dos SGBD não implementa asserções

Exemplo 2

- `CREATE DOMAIN semestre AS INTEGER
CHECK (VALUE = 1 OR VALUE = 2);`
- Este novo domínio pode ser usado:
 - `CREATE TABLE disciplina (id serial, semestre
SEMESTRE,.....)`

Triggers

- A verificação de restrições também pode ser feita com um TRIGGER
- Um *trigger* tem duas partes:
 - Uma função, definida com CREATE FUNCTION
 - A ligação desta função a uma tabela, feita com CREATE TRIGGER
- Um *trigger* pode disparar uma vez para cada linha (row level), ou uma vez para o comando (statement level)
- A sintaxe de CREATE TRIGGER é rica, com muitas possibilidades

Triggers (1)

- Um trigger pode disparar ANTES ou DEPOIS de um dado evento (INSERT, DELETE, UPDATE)
- A função tem acesso a NEW (nova linha, após o evento) e a OLD (antiga linha, antes do evento ser executado) → apenas em “row level” triggers
- NEW pode ser modificada se a condição for BEFORE
- Podem existir vários triggers para a mesma tabela e evento

Triggers (2)

- A função de um trigger que dispare ANTES para cada linha pode devolver NULL (a ação não é executada), ou a linha (NEW) no caso do evento ser INSERT ou UPDATE
- Se o trigger dispara DEPOIS, a função deve devolver NULL
- Um trigger que propaga informação para outras tabelas (ex. log) deve disparar DEPOIS para garantir que todos os ANTES já dispararam

Triggers: definir a função

- CREATE FUNCTION VerificaCurso()
RETURNS TRIGGER AS
\$\$BEGIN IF NOT (SELECT curso_id from
disciplina WHERE id = NEW.disciplina_id) =
(SELECT curso_id FROM matricula WHERE
aluno_id=NEW.aluno_id)
THEN RAISE EXCEPTION 'Erro:
disciplina de outro curso.';
END IF; RETURN NEW;
END\$\$
LANGUAGE plpgsql;

Triggers: associar à tabela e evento

- CREATE TRIGGER InscricaoCorrecta AFTER
UPDATE OR INSERT ON inscrito
FOR EACH ROW
EXECUTE PROCEDURE VerificaCurso();
- Este trigger dispara para cada linha inserida ou modificada
- Se fosse definido para o comando (statement level) dispararia uma única vez, mesmo que o comando resultasse em nenhuma linha inserida ou modificada

Triggers

```
CREATE FUNCTION log_mudanca_curso()  
  RETURNS trigger AS  
$$BEGIN  
  IF NEW.curso_id <> OLD.curso_id THEN  
    INSERT INTO mudancas_curso(aluno_id,  
      curso_id, data) VALUES (OLD.aluno_id,  
      NEW.curso_id ,now());  
  END IF;  
  RETURN NEW;  
END$$ LANGUAGE plpgsql
```

Triggers

```
CREATE TRIGGER regista_mudancas  
  BEFORE UPDATE  
  ON matricula  
  FOR EACH ROW  
  EXECUTE PROCEDURE log_mudanca_curso();
```

Vistas

- Permite apresentar ao utilizador final a informação de que este necessita, evitando expor a relação, ou as relações, de base
- É o resultado da execução de uma consulta
- Evita expor no nível conceptual a informação do nível lógico
- As vistas podem ser definidas a partir de uma ou mais relações
 - E devem ser atualizadas sempre que essas relações são atualizadas
 - Não são, por isso, materializadas (chamam-se tabelas virtuais)

Vistas (exemplo)

- `CREATE MATERIALIZED VIEW` permite materializar em memória a vista; para se verem os resultados mais recentes deve-se pedir para atualizar a vista
- `CREATE VIEW` opcionais `AS SELECT * FROM disciplinas WHERE tipo = 'opcional';`
- A vista permite esconder detalhes das relações de base
- Nem todos os SGBD permitem atualizar vistas

Álgebra Relacional

- Linguagem procedimental que usa expressões algébricas
 - Diz-se o “que se quer”, e não “como se consegue”
- Na prática o Modelo Relacional é diferente da Álgebra Relacional
- O Modelo Relacional não é baseado em conjuntos (admite duplicados)

Propriedade de fecho

- Os operandos da álgebra são as relações ou variáveis que representam relações, e os operadores permitem manipulações comuns nas relações da base de dados
- A álgebra relacional forma um “sistema fechado”: o resultado de um operador (sendo sempre uma tabela) pode constituir a entrada de outros operadores:
 - $\text{operador}_1(\dots(\text{operador}_n(\text{expressão}))\dots)$

Operadores relacionais

- SELECT/RESTRICT: seleciona linhas de uma relação
- PROJECT: seleciona colunas de uma relação
- JOIN: junta duas relações com base em valores comuns de atributos comuns (na sua forma mais utilizada, mas há outras)


SELECT: exemplo

- `select Vinho where Preço > 100`
- $\sigma_{Preço > 100}(Vinho)$

Vinho

MARCA	PREÇO	ANO	
A	20	1992	
C	50	1992	
D	120	1972	
E	200	1970	

*select também
se chama **restrict***



MARCA	PREÇO	ANO	
D	120	1972	
E	200	1970	

PROJECT: exemplo

- project *Vinho* over *Preço*, *Ano*
- $\pi_{\text{preço,ano}}(\textit{Vinho})$

Vinho

MARCA	PREÇO	ANO	
A	20	1992	
C	50	1992	
D	120	1972	
E	200	1970	



PREÇO	ANO
20	1992
50	1992
120	1972
200	1970

JOIN: exemplo

- join *Vinho* and *Região* over *Zona*
- $\text{Vinho} \bowtie \text{Região}$

Vinho

MARCA	PREÇO	ANO	ZONA
A	20	1992	Z3
C	50	1992	Z1
D	120	1972	Z3
E	200	1970	Z2

Região

ZONA	NOME
Z1	Douro
Z2	V.Real



únicas linhas com valor
comum de ZONA

MARCA	PREÇO	ANO	ZONA	NOME
C	50	1992	Z1	Douro
E	200	1970	Z2	V.Real

Mais exemplos

- $\pi_{\text{preço,ano,zona.nome}}(\sigma_{\text{Preço} > 100}(\text{Vinho}) \bowtie \text{Região})$
- Qual o resultado desta consulta?

JOIN: características

- É um operador importante pois permite fazer relacionamentos entre tabelas
- Pode ser uma operação custosa em tempo, tem de percorrer todas as tabelas envolvidas
- A sua utilização deve ser optimizada, pois é uma operação efectuada frequentemente

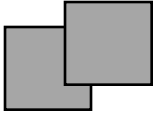
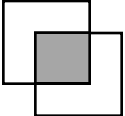
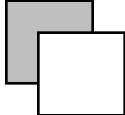
JOIN: tipos

- A junção natural compara colunas com o mesmo nome (não existe em SQL)
- Usa-se a junção *teta*: Vinho \bowtie_{marca} Região
- Equi-junção: a função teta é =
- Junção θ : INNER JOIN
- Junção externa (direita, esquerda, completa): LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN

JOIN: casos particulares

- Se R e S tiverem o mesmo esquema, o resultado de $R \bowtie S$ é a intersecção
- Se R e S não tiverem colunas comuns, a sua junção degenera num produto

Operadores sobre conjuntos

- UNION 
- INTERSECT 
- MINUS 

Operam sobre relações que devem ser compatíveis:
ter o mesmo número de atributos e estes serem do
mesmo tipo e terem o mesmo nome

Produto cartesiano

- Pouco utilizado na prática
- Dadas duas tabelas R e S o seu produto cartesiano é a combinação de todos os tuplos de R com todos os tuplos de S. A cardinalidade do resultado é o produto das cardinalidades de R e de S, e o grau é a soma dos seus graus (se houver colunas com nomes iguais, mudam-se os nomes)

SELECT genérico

```
SELECT [ALL | DISTINCT] select_expr, ...  
  FROM table_references  
  [WHERE where_definition]  
  [GROUP BY {col_name | expr | position} [ASC | DESC]]  
  [HAVING where_definition]  
  [ORDER BY {col_name | expr | position} [ASC | DESC] , ...]  
  [LIMIT {[offset,] row_count | row_count OFFSET offset}]  
  [FOR UPDATE | LOCK IN SHARE MODE]]
```

GROUP BY

- Agrupamentos:
 - max, min
 - count
 - avg
 - sum
 -

Resultado de um SELECT

- Lê todas as combinações possíveis de tuplos das tabelas no FROM
- Elimina as que não respeitam o WHERE
- Agrupa de acordo com GROUP BY
- Elimina grupos de acordo com HAVING

Mapeamento E-R em Relacional

Feliz Gouveia
fribeiro@ufp.edu.pt

Esquemas

- O esquema de uma relação é o seu nome, mais atributos e seus tipos
- Exemplo: aluno {nº, nome, endereço}
- Uma base de dados é uma coleção de relações (tabelas)
- O esquema da Base de Dados é o conjunto de todos os esquemas das relações.

Regras de Mapeamento

- Cada Entidade origina uma Tabela; as colunas da tabela são os atributos
- Se a entidade é fraca, deve incluir a chave da entidade ou entidades de suporte
- Cada Relacionamento origina uma Tabela: a sua chave primária é o conjunto das chaves primárias das entidades envolvidas
- Nos relacionamentos 1:N pode-se juntar a tabela de relacionamento com a Tabela do lado N

Se existir Herança

- Existem várias aproximações:
 - Usar uma Tabela para cada sub-entidade, e uma Tabela para a super-entidade
 - Usar uma Tabela para todas sub-entidades

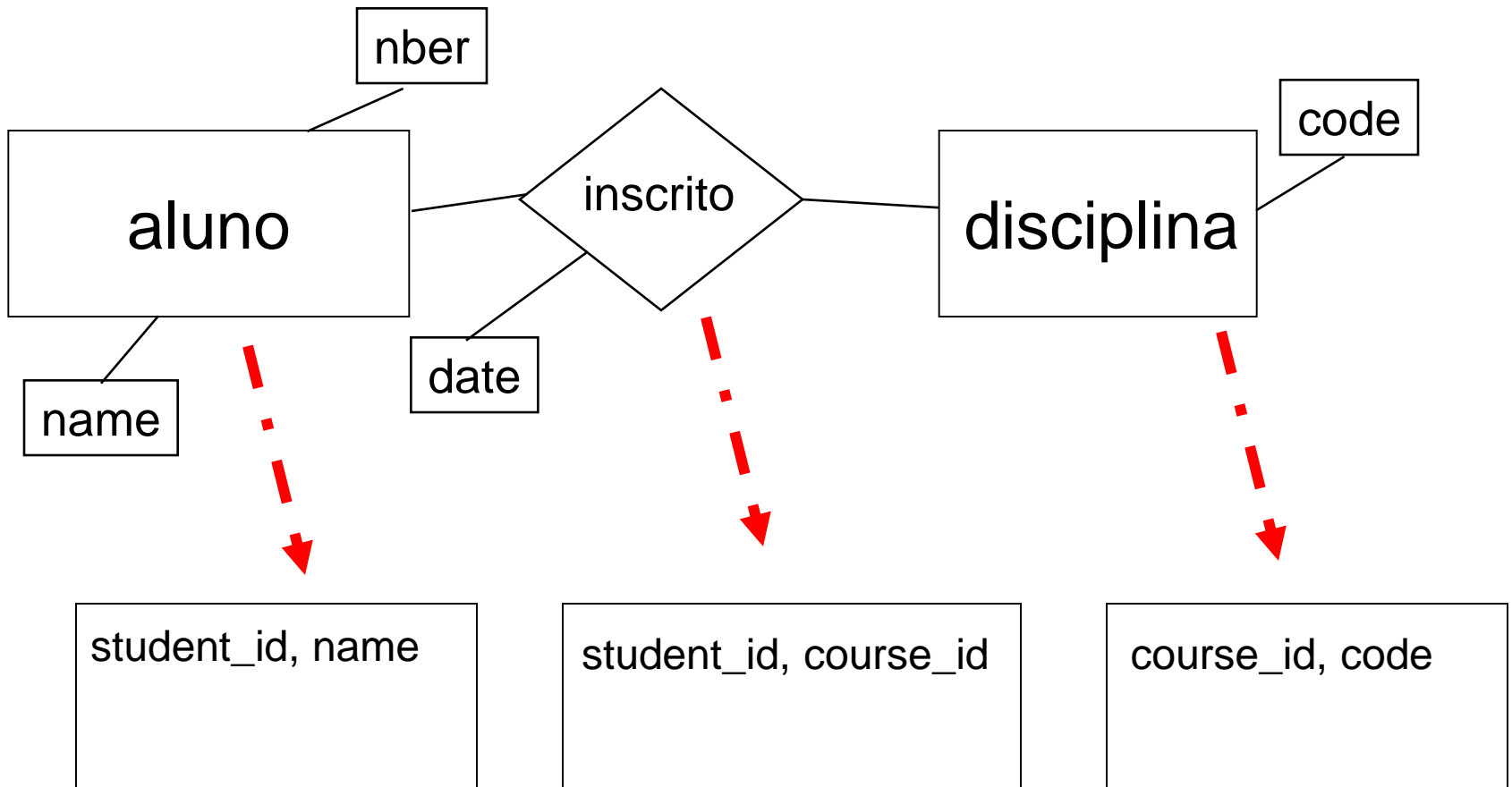
Nomes de Tabelas e Colunas

- Escolha nomes claros (exemplo, TEM não significa grande coisa...)
- Seja consistente na escolha dos nomes (por exemplo no uso de “_”)
- Não misture maiúsculas e minúsculas (prefira tudo em minúsculas)
- Escolha nomes curtos (evite “aluno_inscrito_em_curso”)
- Cuidado com os “nomes reservados”

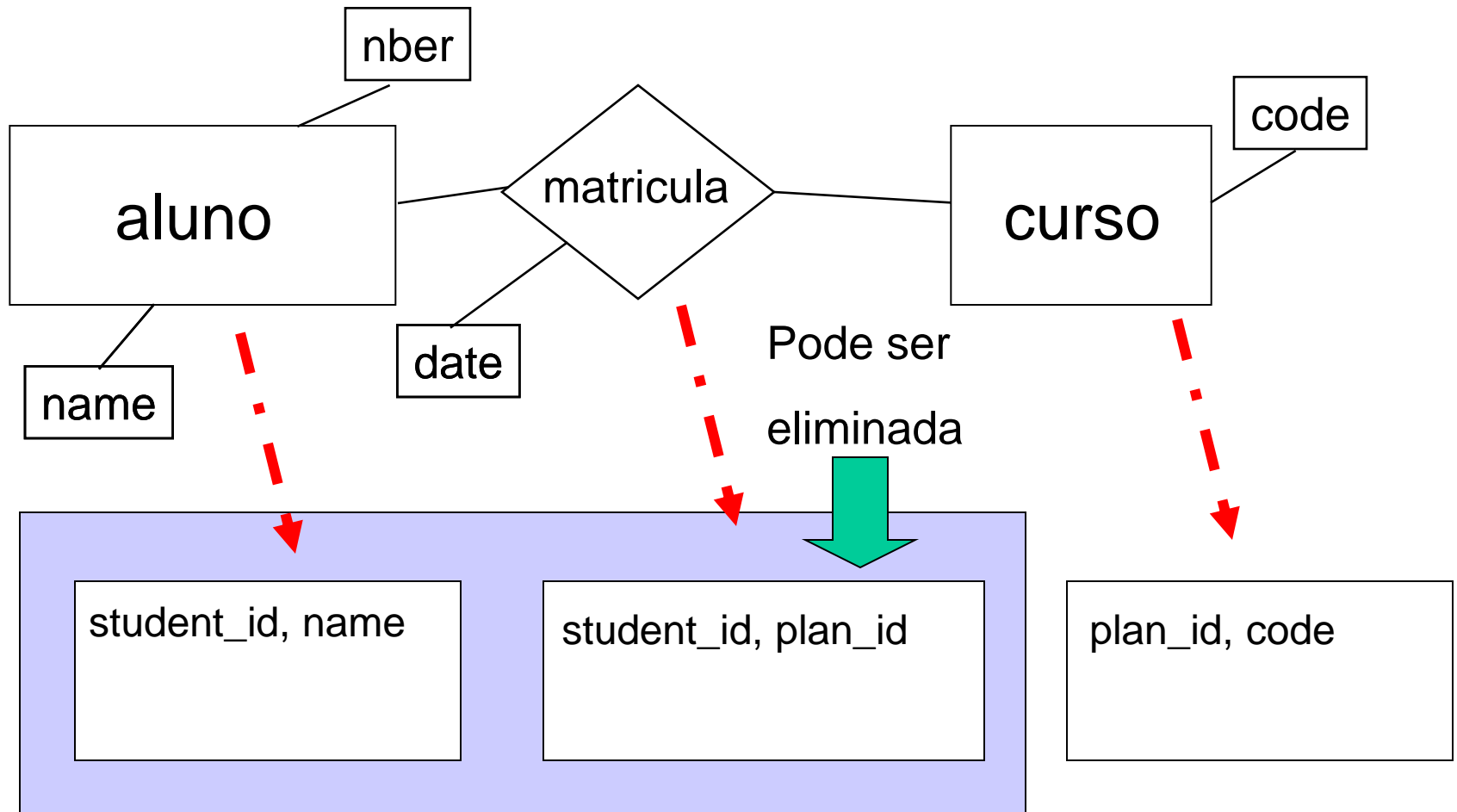
Otimização

- Como regra, deixe para a fase de normalização (a ver mais tarde)
- De momento, use tantas tabelas quantas as que necessitar

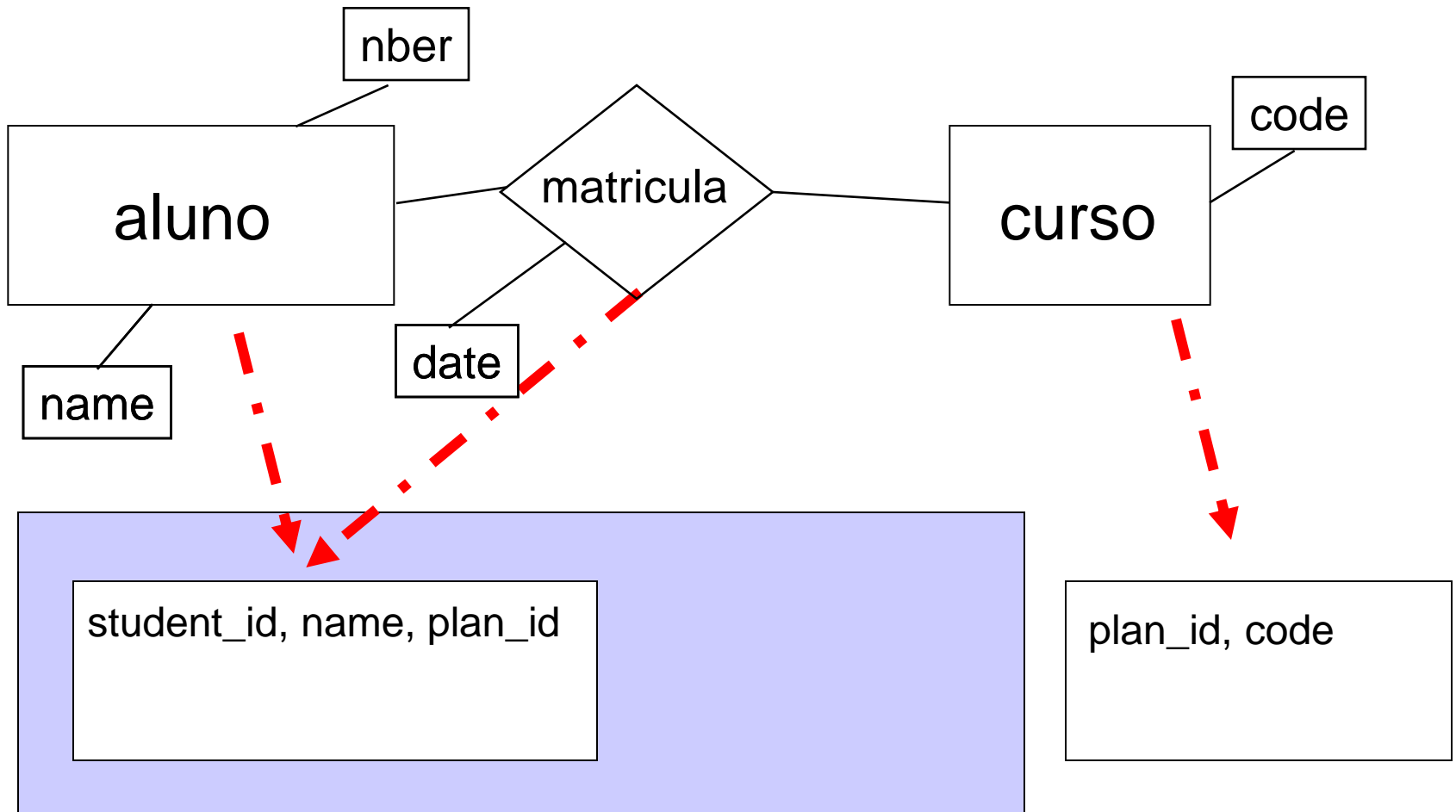
Exemplo



Exemplo com relacionamento N:1



Transforma-se em...



Chaves

- Cada entidade tem uma chave primária
 - Passa para a tabela respetiva
 - Pode ser aconselhável criar uma chave artificial (por exemplo, um contador automático)
- Cada relacionamento recebe as chaves de cada entidade envolvida
 - Passam para a tabela respetiva
 - E declare as chaves estrangeiras necessárias

Cuidado com os Domínios

- Se a mesma coluna aparece em várias tabelas, tem de ter o mesmo tipo (mas não precisa ter o mesmo nome)
- Exemplo: se **curso_id** é um inteiro na tabela **curso**, também tem que ser um inteiro na tabela **matricula**
- Cuidado com o tipo **série** (serial ou auto-increment), é um inteiro

Declare chaves

- Declare Chaves Primárias e Estrangeiras (Primary and Foreign)
- Chaves são Restrições, condicionam valores admissíveis
- Leia sempre a documentação do SGBD que utiliza

Declare restrições

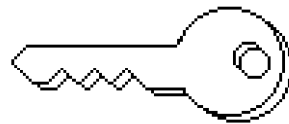
- Por exemplo UNIQUE para as chaves candidatas
- NOT NULL para colunas obrigatórias
- Restrições do tipo “salario > 0”

Dependências Funcionais, Normalização

Feliz Ribeiro Gouveia
fribeiro@ufp.edu.pt

Restrições de Entidade num SGBDR: Chave primária

- Uma chave primária é um subconjunto dos atributos de uma relação R
- Uma chave primária permite identificar linhas de forma única
- A definição de chaves primárias é crucial na construção do modelo de uma BDR

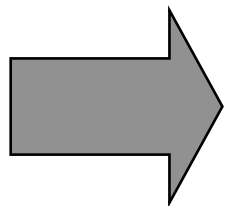


Chaves

- Superchave
 - Não há duas linhas com o mesmo valor da superchave
 - Pode ser reduzida e continua única
- Chave candidata
 - Não há duas linhas com o mesmo valor da chave
 - Não pode ser reduzida (é uma chave mínima)
- Chave primária
 - É uma das chaves candidatas; as outras designam-se “alternativas”

Chaves

- Chave estrangeira
 - Conjunto de atributos de uma relação que são chaves primárias noutras relações (ie, referenciam a linha pela chave primária)
 - Nenhuma dessas referências deve ser inválida, isto é, o valor da chave estrangeira deve existir como valor da chave primária



problema da integridade referencial

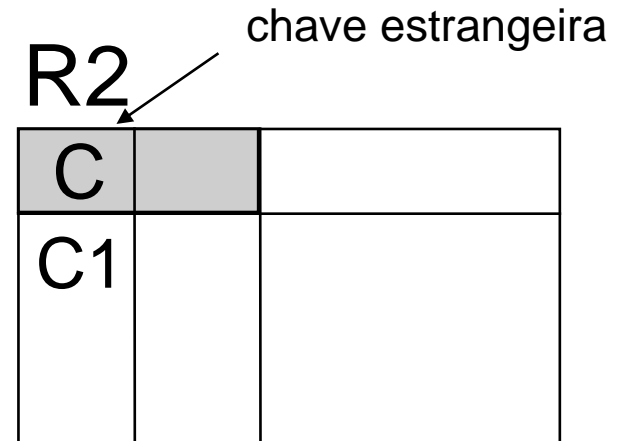
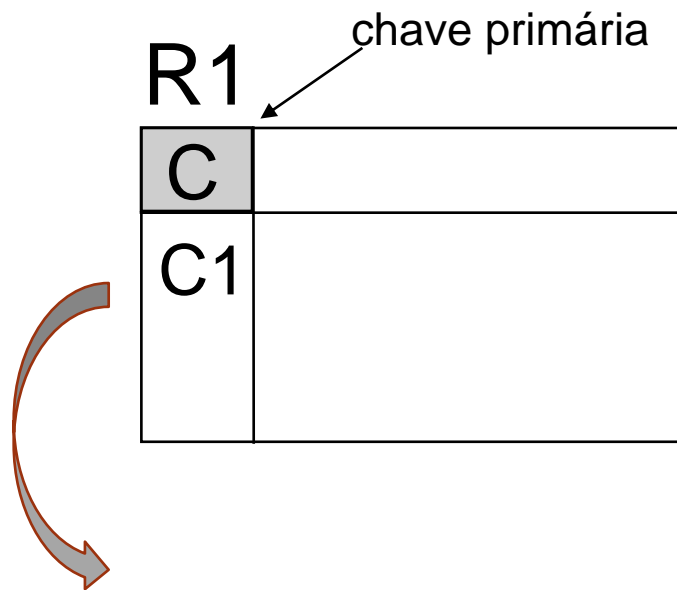
Chaves (cont)

- Atributos primos: atributos que fazem parte de uma chave candidata
- Atributos não-primos: atributos que não fazem parte de uma chave candidata

Atualização de chaves

- E quando se suprime um valor de uma chave primária referenciado numa chave estrangeira?
 - caso restrito: só se pode apagar esse valor se ele não aparecer na chave estrangeira
 - caso em cascata: apagam-se todas as referências como chave estrangeira noutras tabelas

Exemplo

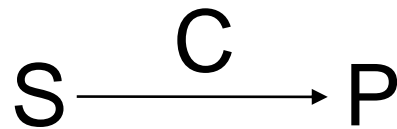


C1 na tabela R1 só se pode eliminar se:

- a) não existir em R2 (caso restrito)
- b) eliminando em R2 (efeito cascata)

Diagramas Referenciais (DR)

- Permitem representar graficamente as chaves primárias e estrangeiras
- Ex: a chave C é primária na relação P e estrangeira na relação S:



Propriedades do DR

- Pode haver referências cíclicas
 - As chaves estrangeira e primária podem estar na mesma tabela: grupo(aluno_id, colega_id)
- Um DR representa um relacionamento, mas estes não são exclusivamente representados por pares de DR

Dependências de inclusão (DI)

- O que vimos representam dependências de inclusão
- Não se limitam a chaves estrangeiras, por exemplo:
 - $\text{AVALIADO}(\text{aluno_id}) \subseteq \text{INSCRITO}(\text{aluno_id})$
- Ou herança; um ALUNO é uma PESSOA:
 - $\text{ALUNO}(\text{ID}) \subseteq \text{PESSOA}(\text{ID})$

Dependências Funcionais (DF)

- Por exemplo na tabela ALUNO temos uma dependência entre NÚMERO e NOME
 - Para um dado NÚMERO de aluno o NOME é sempre o mesmo
 - Repare que para o mesmo NOME podem existir diferentes valores de NÚMERO
- Da mesma forma, para cada NÚMERO temos sempre o mesmo valor de MORADA
- Pode identificar mais dependências?

Dependências Funcionais (DF)

- Representam restrições de integridade numa relação
 - Restringem os valores que determinados atributos podem assumir
- Podem ser temporárias (ou coincidências) ou permanentes (só estas últimas nos interessam)
 - Se uma empresa só tiver clientes do Porto, pode pensar que sabendo o produto se sabe a cidade, mas na realidade é coincidência
- A identificação das dependências pode ser difícil, principalmente em domínios complexos

DF: convenção

- As letras A, B, C, \dots denotam atributos
- As letras X, Y, Z, \dots denotam conjuntos de atributos
- ABC denota o conjunto $\{A, B, C\}$
- R denota todos os atributos da relação R

DF: definição

- Sejam X e Y subconjuntos arbitrários de atributos de uma mesma relação R . Diz-se que Y é **funcionalmente dependente** em X ,

$$X \rightarrow Y$$

- se e só se cada valor de X tiver associado precisamente um só valor de Y . Diz-se que X **determina** Y , ou que Y **depende** de X .
 - X é o determinante, Y é o dependente
- Dependência **funcional** porque é equivalente a uma função $f(X) = Y$

DF: definição

- Uma DF $X \rightarrow Y$ é trivial se $Y \subset X$ (está contido)
- Uma DF $X \rightarrow Y$ é não-trivial se $Y \not\subset X$
- Uma DF $X \rightarrow Y$ é completamente não-trivial se $Y \cap X = \emptyset$

DF: caso particular

- Se X é uma superchave da relação R , então X determina o conjunto de todos os atributos
$$X \rightarrow \{A_1, A_2, \dots, A_n\}$$
- Ou seja, se uma DF contém todos os atributos de R , então o seu determinante é uma superchave de R
- Se X determina uma superchave, então X também é uma superchave
- Se $X \rightarrow Y$, e se X não é uma chave candidata, então há redundância na relação (com possíveis problemas de atualização)

DF: redundância

aluno_id	disciplina_id	nome_disciplina
1000	201	Base de dados
1005	300	Programação
1020	400	Algoritmos
1100	201	Base de dados

- Relação **inscrito**
- Chave: {aluno_id, disciplina_id}
- A dependência {disciplina_id} → {nome_disciplina} implica redundância na relação, repetindo-se nome_disciplina

Identificação das DF

- Se as DF são importantes para detetar redundâncias, como identificar todas as DF possíveis a partir de um conjunto inicial F de dependências?
 - Ou seja, como determinar todas as DF implicadas por um conjunto F ?
- Diz-se que F **implica** $X \rightarrow Y$, e escreve-se $F \models X \rightarrow Y$ se $X \rightarrow Y$ for válida em qualquer relação que respeite F
- Uma primeira resposta foi dada por Armstrong, que mostrou que se podiam gerar novas DF

Axiomas de Armstrong

- Sejam X , Y e Z subconjuntos arbitrários de atributos de R . Prova-se o seguinte:
 - Reflexividade: se Y é um subconjunto de X , isto é $X \supseteq Y$, então $X \rightarrow Y$
 - Neste caso a DF é dita trivial. Uma DF trivial é sempre verdadeira em qualquer relação
 - Aumento: se $X \rightarrow Y$ e $Z \supseteq W$, então $XZ \rightarrow YW$
 - Por exemplo, $X \rightarrow Y$ então $XZ \rightarrow YZ$
 - Transitividade: se $X \rightarrow Y$ e $Y \rightarrow Z$ então $X \rightarrow Z$

Validade das regras

- Se uma DF pode ser derivada por estas regras, a partir de um conjunto F de DF, então ela é válida em qualquer relação que respeitar F
 - Não são geradas DF **falsas**
- As regras geram **todas** as DF possíveis
- Regra da união de DF: se $X \rightarrow Y$ e $X \rightarrow Z$
 1. $X \rightarrow XY$ (aumento de X, trivial)
 2. $YX \rightarrow YZ$ (aumento de Y)
 3. $X \rightarrow YZ$ (transitividade de 1 e 2)

Mais regras

- Exemplo anterior define a regra de união de DF:
 - se $X \rightarrow Y$ e $X \rightarrow Z$ então $X \rightarrow YZ$
- Decomposição (ou projeção):
 - Se $X \rightarrow Z$ então $X \rightarrow A, \forall A \in Z$
- Pseudo-transitividade:
 - Se $X \rightarrow Y$, e $WY \rightarrow Z$ então $WX \rightarrow Z$

Fecho de DF

- Seja F o conjunto de DF da relação R
- F^+ , o fecho de F , é o conjunto de todas as DF que são implicadas por F :
 - $F^+ = \{X \rightarrow Y \mid F \models X \rightarrow Y\}$
- F^+ é único
- É possível testar se uma DF f é implicada por F gerando F^+ e verificando se $f \in F^+$
- Gerar F^+ tem um custo importante, dependendo do número de colunas
- Uma forma simples é mostrada a seguir

Cálculo de F^+

$F^+ \leftarrow F$

Repetir

Para cada DF $f \in F^+$

aplicar aumento e reflexividade a f e juntar
as DF resultantes a F^+

Para cada par f_1 e $f_2 \in F^+$

se f_1 e f_2 puderem ser combinadas com
transitividade, juntar a DF resultante a F^+

Até F^+ não mudar

- Vamos ver um método alternativo, com um custo inferior, a seguir

Fecho de um atributo

- X^+ é por definição o conjunto de todos os atributos Y tal que $X \rightarrow Y \in F^+$

$X^+ \leftarrow X$

Repetir

Para cada DF $Y \rightarrow Z \in F$

Se $Y \subseteq X^+$ e $Z \notin X^+$

Juntar Z a X^+ , descartar $Y \rightarrow Z$

Até X^+ não mudar ou X^+ ter todos os atributos

- Escreve-se X^+_F se houver necessidade de especificar com qual conjunto de DF o fecho é calculado

Propriedades do fecho

- $X \subseteq X^+$
- Se $X \subseteq X'$, então $X^+ \subseteq (X')^+$
- $(X^+)^+ = X^+$
- Gerando o fecho de todos os atributos obtém-se, por definição de fecho X^+ , F^+
- Calcular X^+ tem, com as devidas otimizações, um custo linear em $|F|$

Utilizações de X^+

- Se X é uma **superchave**, $X^+ = R$ por definição de chave
 - Útil para identificar chaves (veremos mais tarde)
- Para saber se $X \rightarrow Y \in F^+$, calcula-se X^+ e testa-se se $Y \in X^+$
- Pode-se calcular F^+ gerando todas as DF implicadas pelo fecho de cada uma das $2^n - 1$ combinações dos atributos de R
 - Para 10 atributos temos 1023 combinações

Calcular as chaves

- Como vimos, o fecho de um atributo pode ser utilizado para identificar chaves
- X é superchave se $X \rightarrow R \in F^+$, ou seja $X^+ = R$
- X é chave candidata se não existe $X' \subset X$ tal que $X' \rightarrow R \in F^+$
- Yu e Johnson demonstraram que se $|F| = k$, o número máximo de chaves é $k!$
- Devem-se testar as $2^n - 1$ combinações de atributos de R (algoritmo força bruta)
- Deve-se usar a cobertura canónica G de F

Algoritmo força bruta

$C \leftarrow \emptyset$, no fim contém as chaves

Para todas as permutações $A_1 \dots A_{2^n-1}$ de R
ordenadas por tamanho crescente

se $A_i^+ = R$ então

$C \leftarrow C \cup A_i$

remover do teste qualquer $A_j : A_i \subset A_j$
porque nunca pode ser chave

- O algoritmo testa exaustivamente todos os 2^n-1 conjuntos de atributos, o que é desnecessário

Saiedian e Spencer 1996

- \mathcal{L} é o conjunto de atributos que só aparecem nos determinantes de algumas DF, ou em nenhuma
- \mathcal{R} é o conjunto de atributos que só aparecem nos dependentes de algumas DF
- \mathcal{B} é o conjunto de atributos que aparecem em ambos de lados de algumas DF
- Só devem ser testados os atributos de \mathcal{L} , e se necessário aumentados dos de \mathcal{B}
- Nunca é necessário testar os atributos de \mathcal{R}

Saiedian e Spencer 1996

- $\forall A \in \mathcal{L}$, A é primo, e participa em **todas** as chaves candidatas
- Se os atributos de \mathcal{L} formarem uma chave, isto é se $\mathcal{L}^+ = R$, pode-se parar
- Caso contrário, junta-se cada atributo de \mathcal{B} , um a um, a \mathcal{L} e calcula-se o fecho
- Ao contrário de Elmasri e Navathe este algoritmo encontra todas as chaves

Exemplo

- $\mathcal{F} = \{AD \rightarrow B, AB \rightarrow E, C \rightarrow D, B \rightarrow C, AC \rightarrow F\}$

$\mathcal{L} = \{A\}$ $\mathcal{B} = \{BCD\}$ $\mathcal{R} = \{EF\}$

- $\mathcal{L}^+ = A$, não é chave

Juntam-se os atributos de \mathcal{B} a \mathcal{L} :

- $\{AB\}^+ = ABECDF$, é chave
- $\{AC\}^+ = ACDBEF$, é chave
- $\{AD\}^+ = ADBCCEF$, é chave

Normalização

- Destina-se a clarificar o modelo da base de dados e a eliminar possíveis problemas de inserção, remoção e atualização
- Objectivo principal é eliminar redundâncias mantendo as restrições de integridade iniciais
- Resulta quase sempre na decomposição de relações em novas relações
- Tem que se garantir que a decomposição é adequada

Normalização: abordagens

- Em teoria, a normalização pode ser feita apenas a partir de um conjunto de atributos e de dependências funcionais (**síntese**)
 - Dispensa o diagrama E-R, as relações são definidas a partir das dependências e do conjunto de atributos
- Na prática serve para refinar a tradução de um modelo E-R em Relacional (**análise**)
 - Desenha-se o E-R
 - Faz-se o mapeamento em Relacional
 - Refina-se usando DF

Decomposição de relações

- Diz-se que uma decomposição:
 - **É sem perdas** se os tuplos originais puderem ser reconstruídos a partir das relações resultantes da decomposição, e isto sem introduzir tuplos adicionais
 - **Preserva as DF** se as relações resultantes mantiverem as mesmas restrições, sob forma de DF, que a relação original

Teorema de Heath (1971)

- Uma DF implica a existência de uma decomposição sem perdas:
 - Se $X \rightarrow Y$ e $Z = R - XY$, então R admite a decomposição sem perdas $R_1 = XY$ e $R_2 = XZ$
- Neste caso a relação original pode ser sempre reconstruída através de junções
 - $R = \pi_{R_1}(R) \bowtie \pi_{R_2}(R)$
- Note que “sem perdas” não significa obter menos tuplos na reconstrução; significa mais tuplos, isto é, menos informação

Preservação de dependências

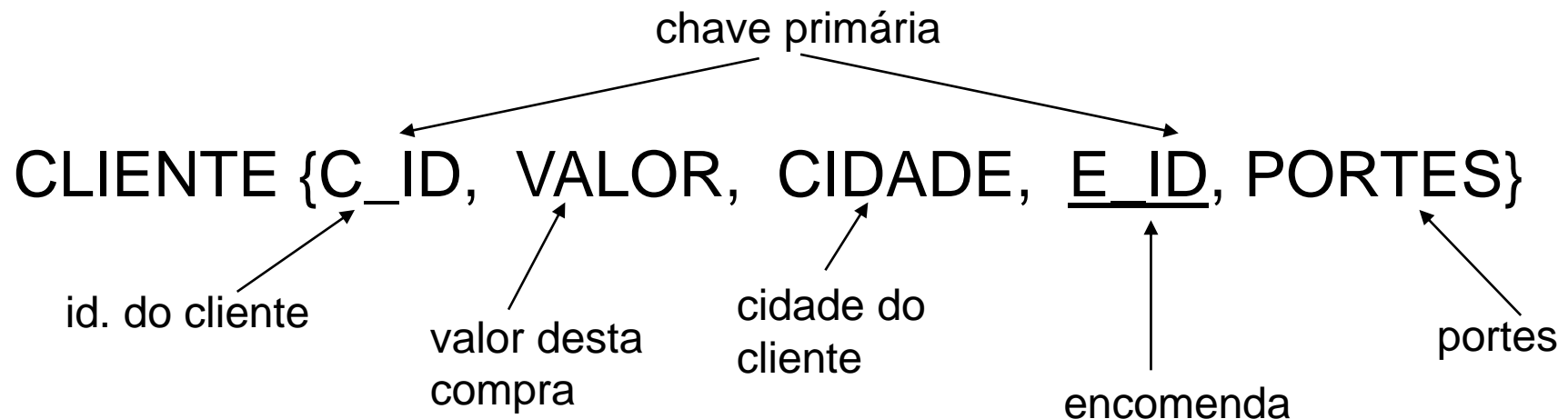
- Uma decomposição de R em R_1 e R_2 preserva as dependências se e só se:
 - F é equivalente a $F_1 \cup F_2 : F^+ = (F_1 \cup F_2)^+$
- F_i é o conjunto de DF de F^+ que só contém atributos de R_i
- Como as dependências são restrições do problema, devem ser respeitadas independentemente das relações que forem definidas
- Se não forem preservadas, a sua verificação implica junções entre relações

DF e normalização

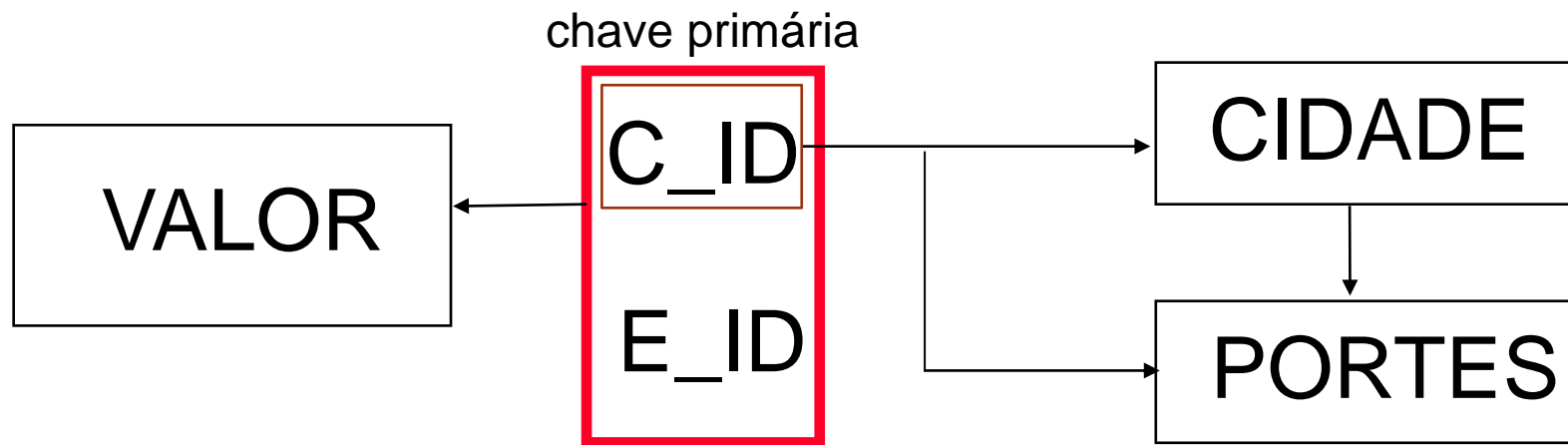
- 1FN: os valores dos atributos são atômicos, não há grupos de repetição. Deriva da própria definição do Modelo Relacional
- 2FN: está em 1FN e todos os atributos não-primos dependem da totalidade de chaves candidatas
- 3FN: está em 2FN, e todos os atributos não-primos dependem, de forma não transitiva, de chaves candidatas (não há dependências de atributos que não são chave)

DF: Exemplo

- Uma loja regista as compras dos clientes, localizados numa dada cidade, o que implica o custo dos portes; cada compra tem um valor



DF do exemplo



- Cidade depende de C_ID, que não é a chave primária: {C_ID, E_ID}
- Portes depende de Cidade e do cliente, e não da chave primária
- Valor depende da chave primária

Problemas do exemplo

- Inserção: não se pode inserir um cliente sem que se faça uma compra
- Remoção: ao remover um cliente, apaga-se informação útil (a cidade onde está)
- Atualização: se um cliente aparece várias vezes (faz várias compras), Cidade aparece várias vezes

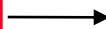
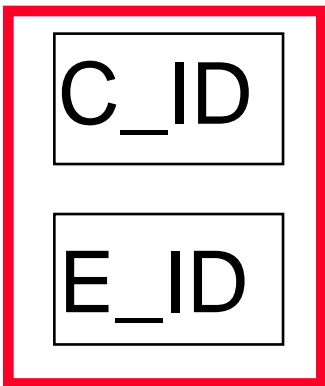
Alteração do exemplo

- Colocar os atributos a dependerem da **totalidade** da chave primária, para o que se decompõe a relação em duas:

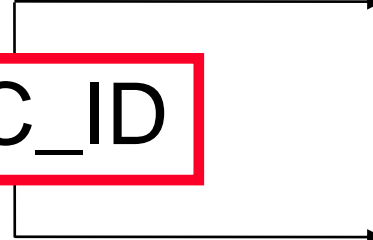
CLIENTE {C_ID, CIDADE, PORTES}
ENCOMENDA {E_ID, C_ID, VALOR}

DF para as novas relações

chave primária



chave primária



Estão em 2FN: estão em 1FN, e os atributos não-primos dependem da totalidade da chave primária

Como a chave C_ID não é composta, a relação da direita está automaticamente em 2FN

Problemas no novo exemplo

- Inserção: não se pode inserir uma cidade e os portes até haver um cliente nessa cidade
- Remoção: quando se remove um cliente, perde-se a cidade e os portes
- Atualização: quando se inserem vários clientes na mesma cidade, os portes repetem-se

Nova alteração do exemplo

- Decompõe-se a relação cujos atributos não dependem exclusivamente da chave primária:

CLIENTE {C_ID, CIDADE} C_ID → CIDADE

CIDADE {CIDADE, PORTES} CIDADE → PORTES

ENCOMENDA {C_ID, E_ID, VALOR}

Todas as relações estão em 3FN

O que vimos até agora

- Cada campo de um tuplo deve depender da Chave (1FN), de Toda a Chave (2FN) e Nada Mais do que a Chave (3FN)
- A decomposição das relações é sem perdas, isto é, a relação original pode ser reconstruída
- Vamos ver como decompor relações, usando as dependências funcionais

Definição de 3FN

- Para cada DF: $X \rightarrow Y \in F^+$
 - $Y \subseteq X$ (a DF é trivial) ou
 - X é uma superchave, ou
 - Todos os atributos em $X \multimap Y$ são atributos primos (isto é, fazem parte de uma chave candidata, podem ser chaves diferentes)
- Se uma relação não está na 3FN, pode ser decomposta em relações que respeitam a 3FN
 - Decomposição sem perdas
 - Preservando as dependências

Anomalias na 3FN

- Exemplo: os alunos inscrevem-se em turmas. A mesma disciplina tem várias turmas. As DF são:
 - $DF_1 \{disc_id, aluno_id\} \rightarrow \{turma_id\}$
 - $DF_2 \{turma_id\} \rightarrow \{disc_id\}$
- Relação possível: $\{aluno_id, disc_id, turma_id\}$
- Chaves: $\{aluno_id, turma_id\}$ e $\{aluno_id, disc_id\}$
- Está em 3FN: o determinante de DF_1 é chave, o dependente de DF_2 é primo
- No entanto tem redundância...

Anomalias na 3FN

- Repete-se {turma_id, disc_id} para cada aluno nessa turma
- Se uma turma não tiver alunos, temos que usar valores nulos (NULL) mas não podemos porque aluno_id faz parte da chave

Forma normal Boyce-Codd

- Mais geral que a 3FN, deve-se testar no caso em que há várias chaves candidatas tais que:
 - as chaves têm mais do que um atributo
 - e têm pelo menos um atributo em comum
- Caso contrário, 3FN = BCNF
 - Uma relação com 2 atributos está em BCNF
- Uma relação está em BCNF se os únicos determinantes forem superchaves

Definição de BCNF

- Uma relação R está em BCNF se o determinante de cada DF não-trivial é uma superchave de R

Para cada DF: $X \rightarrow Y \in F^+$

$Y \subseteq X$ (a DF é *trivial*)

ou

X é uma superchave

Podem-se começar a testar as DF em F e só passar para F^+ se todas forem chave

Decomposição BCNF (simples)

- Aplicar a regra da união às DF de F
- Calcular F^+
- $\mathcal{R} = R$
- Enquanto houver um $R_i \in \mathcal{R}$ não em BCNF
 - Seja $X \rightarrow Y$ não trivial, existente em R_i e F_i mas onde X não é chave e $X \cap Y = \emptyset$
 - $\mathcal{R} = (\mathcal{R} - R_i) \cup \{(R_i - Y) \cup X, XY\}$
- 1. F_i é o conjunto de dependências de F^+ que só contêm atributos de R_i (chamado projeção de F sobre R_i): $F_i = \pi(F) = \{ X \rightarrow Y \mid X \rightarrow Y \in F^+ \text{ e } XY \subseteq R_i \}$
- 2. Corre em tempo exponencial devido ao cálculo dos F_i

Otimização

- Para testar violações em R basta usar F . Para testar violações nas decomposições de R devemos usar F^+
- Evita-se fragmentação excessiva das relações ao decompor em $X \rightarrow X^+$ em vez de $X \rightarrow Y$
- Seja $X \rightarrow Y$ não trivial, onde X não é chave
- Calcular X^+
- Decompor em $R_1 = X^+$ e $R_2 = X \cup (R - X^+)$
- Se $Z \rightarrow W$, não trivial, existe em R_i e F_i mas Z não é chave, decompor R_i recursivamente

Exemplo

- Decompor em BCNF por causa de $X \rightarrow Y$ ($X \cap Y = \emptyset$):
 - $R_1 = R - Y$, $R_2 = XY$
 - $R_1 \{disc_id, turma_id\}$ $R_2 \{aluno_id, turma_id\}$
 - Não é possível colocar em BCNF sem perder a DF_1
 - Mesmo aluno em 2 turmas da mesma disciplina em R_2 !
 - Por isso, para se verificar $\{aluno_id, disc_id\} \rightarrow \{turma_id\}$ tem que se efetuar uma junção
- Por outro lado, se a relação ficar em 3FN poderemos ter que usar valores nulos quando queremos registrar turmas sem alunos inscritos

BCNF: objetivo

- A vantagem de BCNF é garantir que todas as FD são verificadas por restrições de chaves
 - Mais fácil de implementar, o próprio SGBD o faz
 - Caso contrário tem que ser programada lógica para o efeito (por exemplo com *triggers*)
- É mais restrita que 3FN porque não admite a condição adicional que os atributos em $X \twoheadrightarrow Y$ possam ser primos
- Mas nem sempre é possível decompor em BCNF e preservar as dependências

3FN e BCNF

- Qualquer esquema pode ser decomposto em 3FN e BCNF “sem perdas”
- Só 3FN preserva as dependências, com BCNF nem sempre é possível
- O risco de violar uma DF não preservada em BCNF pode ser maior do que a redundância de 3FN
 - 3FN é geralmente preferida neste caso
 - Ou então definem-se *triggers* ou asserções para poder verificar que a DF perdida é respeitada