Bases de Dados

FORMAS NORMAIS

Características de um bom desenho

- Suponha que as relações *instrutor* e *department* foram combinadas numa única relação: em *in_dep,* que representa a sua junção natural

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

- Há redundância / repetição de dados
- É necessário usar valores *null* (e.g., se quisermos registar um departamento sem professores)

Decomposição

- A única forma de evitar o problema de repetição no esquema *in_dep* é decompô-lo em dois esquemas: *instrutor* e *department*.
- Nem todas as decomposições são boas. Suponha que decompomos

  funcionario(ID, nome, rua, cidade, salário)

  em

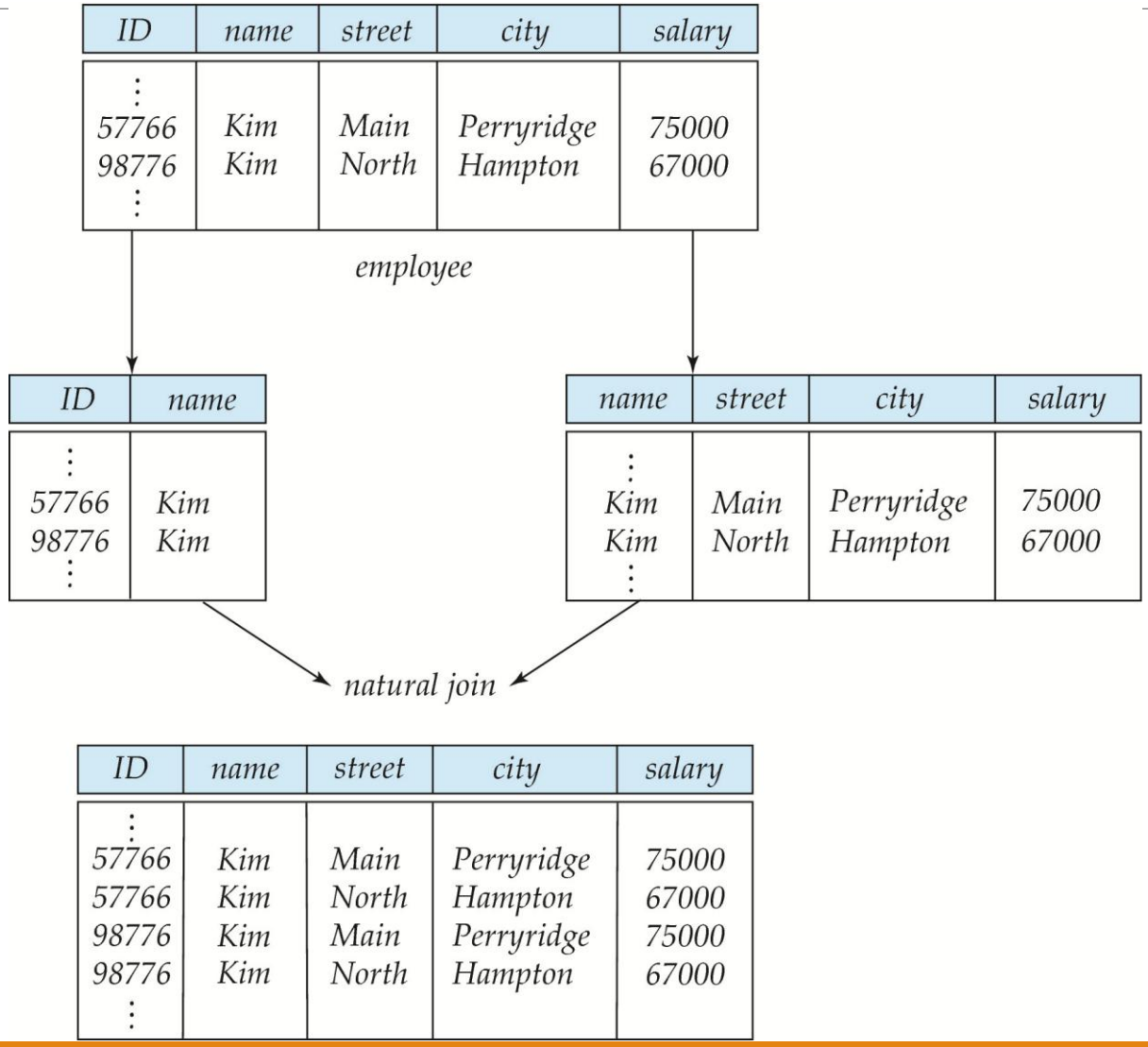
  funcionario1 (*ID* , *nome*)

  funcionario2 (*nome* , *rua, cidade, salário*)

  O problema surge quando temos dois funcionários com o mesmo nome

- O próximo slide ilustra como perdemos dados — não podemos reconstruir a relação original *do funcionario* — e, portanto, esta é uma **decomposição com perdas** .

Uma decomposição com perdas



Decomposição sem perdas

- Seja *R* um esquema de relação e sejam R_1 e R_2 duas relações que formam uma decomposição de R. Ou seja, R = R_1 U R_2

- Dizemos que a decomposição é uma **decomposição sem perdas** se não houver perda de informação ao substituir R pelos dois esquemas de relação R_1 U R_2

- Formalmente,

$$\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$$

Exemplo de decomposição sem perdas

Decomposição de *R = (A, B, C)*

R $_1$ *= (A, B)* *R* $_2$ *= (B, C)*

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_{A,B}(r) \bowtie \Pi_{B,C}(r)$

A	B	C
α	1	A
β	2	B

Teoria da Normalização

- Verifica se uma relação particular *R* está em "boa forma".

- No caso de uma relação *R* não estar em "boa" forma, deve ser decomposta num conjunto de relações { R_1, R_2, ..., R_n } tais que
  - Cada relação está em "boa forma"
  - A decomposição é uma decomposição sem perdas

- A teoria da normalização é baseada em **dependências funcionais**

Dependências Funcionais

- Geralmente, há uma variedade de restrições (regras) sobre os dados no mundo real.
- Por exemplo, algumas das restrições que se espera que existam numa BD de uma Universidade são:
  - Alunos e professores são identificados exclusivamente pelo seu documento de identificação.
  - Cada aluno e professor tem apenas um nome.
  - Cada professor e aluno está associado a apenas um departamento.
  - Cada departamento tem apenas um valor para seu orçamento e apenas um edifício associado.

Dependências Funcionais (Cont.)

- Uma instância de uma relação que satisfaz todas essas restrições do mundo real é chamada de **instância legal** da relação;
- Uma instância legal de uma BD é aquela em que todas as instâncias das relações são instâncias legais
- Restrições exigem que o valor de um determinado conjunto de atributos **determine** o valor de outro conjunto de atributos.
- Uma dependência funcional é uma generalização da noção de *chave.*

Definição de Dependências Funcionais

- Seja *R* um esquema de relação

$$\alpha \subseteq R \; e \; \beta \subseteq R$$

- A **dependência funcional**

$$\alpha \rightarrow \beta$$

verifica-se em R se e só se para quaisquer relações legais *r*(R), sempre que quaisquer dois tuplos t_1 e t_2 de *r* têm os mesmo valores para os atributos α, eles também têm os mesmos valores para os atributos β. Ou seja,

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- Exemplo: Considere *r* (A ,*B*) com a seguinte instância de *r.*

1	4
1	5
3	7

- Neste caso, verifica-se $B \rightarrow A$; mas $A \rightarrow B$ NÃO se verifica

Fecho de um conjunto de dependências funcionais

- Dado um conjunto *F* de dependências funcionais, há outras dependências funcionais que são logicamente implicadas por *F*.
  - Se $A \rightarrow B$ e $B \rightarrow C$, então podemos inferir que $A \rightarrow$ C
  - etc.
- O conjunto de **todas** as dependências funcionais logicamente implicadas por *F* é o **fecho** de *F* .
- Denotamos o *fecho* de *F* por F^+ .

Chaves e Dependências Funcionais

- *K* é uma superchave para o esquema de relação *R* se e só se $K \rightarrow R$
- *K* é uma chave candidata para *R* se e só se
  - $K \rightarrow R$, e
  - para nenhum $\alpha \subset K$, $\alpha \rightarrow R$
- As dependências funcionais permitem expressar restrições que não podem ser expressas usando superchaves. Por exemplo, considere o esquema:

  in_dep (*ID, name, salary, dept_name, building, budget*).

- Espera-se que as seguintes dependências funcionais sejam verificadas:

  dept_name $\rightarrow$ *building*

  ID $\rightarrow$ *building*

- mas não se espera que o seguinte aconteça:

  dept_name $\rightarrow$ *salary*

Uso de dependências funcionais

- Usamos dependências funcionais para:
  - Testar relações para ver se elas são legais para um determinado conjunto de dependências funcionais.
    - Se uma relação *r* é legal para um conjunto *F* de dependências funcionais, dizemos que *r* **satisfaz** *F.*
  - Para especificar restrições no conjunto de relações legais
    - Dizemos que *F* **se verifica em** *R* se todas as relações legais em *R* satisfazem o conjunto de dependências funcionais *F.*

Observação: uma instância específica de uma relação pode satisfazer uma dependência funcional mesmo que a dependência funcional não seja válida para todas as instâncias legais.

- Por exemplo, uma instância específica de *instrutor* pode satisfazer
  name $\rightarrow$ ID

Dependências Funcionais Triviais

- *Uma* dependência funcional é **trivial** se for satisfeita por todas as instâncias de uma relação
- Exemplo *:*
  - *ID, name* $\rightarrow$ *ID*
  - *name* $\rightarrow$ *name*
- Em geral, $\alpha \rightarrow \beta$ é trivial se $\beta \subseteq \alpha$

Decomposição sem perdas

- Podemos usar dependências funcionais para mostrar decomposições sem perdas.

- Para o caso de $R = (R_1, R_2)$, exigimos que para todas as relações possíveis r no esquema R

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- Uma decomposição de R em R_1 e R_2 é uma decomposição sem perdas se pelo menos uma das seguintes dependências estiver em F^+ :
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

- As dependências funcionais acima são uma condição suficiente para a decomposição de junção sem perdas

Exemplo

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$

- $R_1 = (A, B)$, $R_2 = (B, C)$
  - Decomposição sem perdas:

    $R_1 \cap R_2 = \{ B \}$ e $B \rightarrow BC$

- $R_1 = (A, B)$, $R_2 = (A, C)$
  - Decomposição sem perdas:

    $R_1 \cap R_2 = \{ A \}$ e $A \rightarrow$ A B

- *Observação:*
  - $B \rightarrow BC$

    é uma notação abreviada para
  - $B \rightarrow \{ B, C \}$

Preservação de Dependência

- Testar restrições de dependência funcional sempre que a BD é atualizada tem um custo elevado
- É útil projetar a BD de forma a que as restrições possam ser testadas com eficiência
- Se o teste de uma dependência funcional puder ser feito considerando apenas uma relação, então o custo de testar essa restrição é baixo
- Ao decompor uma relação pode já não ser possível testar uma dependência funcional sem realizar um Produto Cartesiano.
- Uma decomposição que torna computacionalmente difícil garantir uma dependência funcional é considerada NÃO **preservadora de dependência** .

Exemplo de preservação de dependência

- Considere o esquema:

  dept_advisor(s_ID, i_ID, department_name)

  Com dependências funcionais:

  $i\_ID \rightarrow dept\_name$

  $s\_ID, dept\_name \rightarrow i\_ID$

- O desenho acima força a repetição do nome do departamento sempre que um professor participe em mais do que um tuplo de *dept_advisor* .
- Para evitar a repetição é necessário decompor *dept_advisor*
- Qualquer decomposição não incluirá todos os atributos da dependência

  $s\_ID, dept\_name \rightarrow i\_ID$

- Assim, a decomposição é NÃO preservadora de dependência