

# Téma 9 – Relační algebra a jazyk SQL

## Obsah

1. Relační algebra
2. Operace relační algebry
3. Rozšíření relační algebry
4. Hodnoty *null*
5. Úpravy relací
6. Stručný úvod do SQL
7. SQL a relace
8. Základní příkazy SQL
9. Hodnoty *null* a tříhodnotová logika v SQL
10. Příkazy SQL pro modifikaci obsahu databází

# Opakování - co to je *relace*?

- Matematicky: Jsou dány množiny  $D_1, D_2, \dots, D_n$ , pak relací  $R$  rozumíme libovolnou podmnožinu kartézského součinu  $D_1 \times D_2 \times \dots \times D_n$ .
- Relace tedy je množina  $n$ -tic  $(a_1, a_2, \dots, a_n)$ , kde  $a_i \in D_i$
- Příklad:
  - $klient\_jmeno = \{Novák, Mates, Braun, Novotný \dots\}$   
/\* množna jmen klientů \*/
  - $klient\_ulice = \{Spálená, Hlavní, Horní, \dots\}$  /\* množina jmen ulic\*/
  - $klient\_mesto = \{Praha, Brno, Nymburk, \dots\}$  /\* množina jmen měst \*/
  - pak  $r = \{$ 
    - (Novák, Spálená, Praha),
    - (Mates, Horní, Brno),
    - (Braun, Hlavní, Brno),
    - (Novotný, Horní, Nymburk) $\}$je relace, tj. podmnožina  $klient\_jmeno \times klient\_ulice \times klient\_mesto$
- Vzhledem k tomu, že jde vždy o konečné množiny, lze je vyjádřit výčtem, tedy tabulkami

# Strukturovaný dotazovací jazyk SQL

- Structured Query Language (SQL)
  - jazyk pro kladení dotazů do databáze
  - obsahuje jak příkazy DML (manipulace s daty), tak i pro definici dat (DDL)
- Svojí syntaxí připomíná přirozenou angličtinu
- SQL se opírá o výrazy relační algebry
- Existuje mnoho dialektů SQL
  - liší se různými rozšířeními či speciálními agregátními funkcemi
  - skladba vestavěných predikátů se rovněž může lišit
  - [www.w3schools.com/sql/](http://www.w3schools.com/sql/)
- Probereme jen základní konstrukty jazyka
  - konkrétní varianty vždy závisí na příslušném dialektu použitého databázového systému
- Poznámka k syntaxi
  - SQL identifikátory a jména atributů NEROZLIŠUJÍ malá a velká písmena (tj. *Branch\_Name*  $\equiv$  *BRANCH\_NAME*  $\equiv$  *branch\_name*)

# SQL přípouští duplikáty

- Pro zajištění dobré analogie SQL a množinového modelu potřebujeme tzv. **multiset**
  - Multiset je množina s opakujícími se prvky
- Potřebujeme multisetové verze relačních operátorů mezi relacemi  $r_1$  a  $r_2$ 
  - $\sigma_\theta(r_1)$ : Je-li  $c_1$  kopií  $n$ -tice  $t_1$  v  $r_1$ , a  $t_1$  splňuje selekční predikát  $\theta$ , pak bude  $c_1$  kopií  $t_1$  v  $\sigma_\theta(r_1)$ .
  - $\Pi_A(r)$ : Pro každou kopii  $t_1$  v  $r_1$  bude kopie  $\Pi_A(t_1)$  i v  $\Pi_A(r_1)$ 
    - $r_1 \times r_2$ : Je-li  $c_1$  kopií  $t_1$  v  $r_1$  a  $c_2$  kopií  $t_2$  v  $r_2$ , pak bude  $c_1 * c_2$  kopií  $n$ -tice  $t_1 \dots t_2$  v  $r_1 \times r_2$
- **Příklad:**
  - Multisetové relace  $r_1 (A, B)$  a  $r_2 (C)$  jsou  
 $r_1 = \{(1, a) (2, a)\}$      $r_2 = \{(2), (3), (3)\}$
  - Pak  $\Pi_B(r_1)$  bude  $\{(a), (a)\}$ ,  
a  $\Pi_B(r_1) \times r_2$  dá  $\{(a, 2), (a, 2), (a, 3), (a, 3), (a, 3), (a, 3)\}$
- SQL sémantika příkazu **select**  $A_1, A_2, \dots, A_n$  **from**  $r_1, r_2, \dots, r_m$  **where**  $P$  je ekvivalentní multisetové verzi výrazu

$$\Pi_{A_1, A_2, \dots, A_n} (\sigma_P (r_1 \dot{\cup} r_2 \dot{\cup} \dots \times r_m))$$

# SQL create table

- Relace v SQL je definována příkazem  
**create [temporary] table [if not exists]**  
*jméno* ( $A_1 D_1, A_2 D_2, \dots, A_n D_n,$   
(integritní-omezení<sub>1</sub>), ..., (integritní-omezení<sub>k</sub>)) [options]
  - $r$  je jméno vytvářené relace
  - $A_i$  jsou jména atributů schématu relace  $r$
  - $D_i$  jsou příslušné datové typy hodnot domén atributů  $A_i$
  - Integritní omezení udávají co musí tabulka splňovat a DBMS na toto splnění dohlíží (not null, primary key, ...)
  - Options – specifikují detailně některé vlastnosti tabulky

- Příklad

```
create table zákazník  
( jméno   varchar(15) not null,  
  město  varchar(30),  
  psč    integer,  
  primary key (jméno, město)  
)
```

# SQL create table

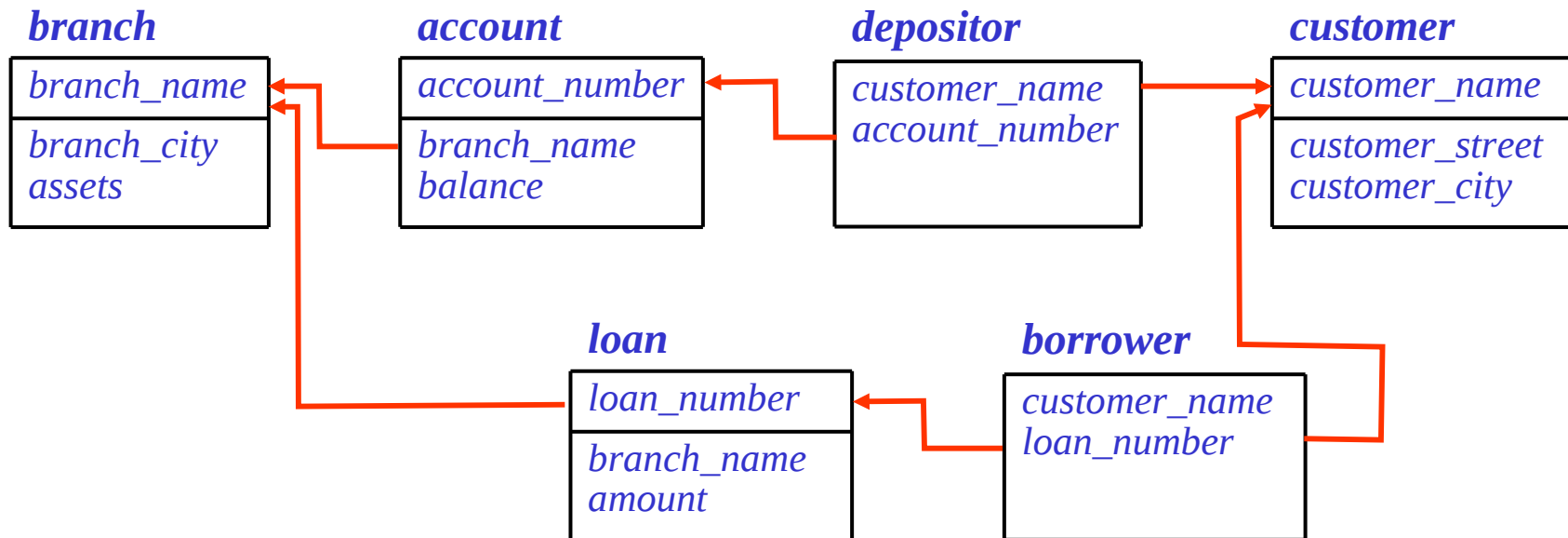
- Domény
  - Celá čísla – BIT, TINYINT, SMALLINT, MEDIUMINT, INT, INTEGER(p), BIGINT
  - Reálná čísla – REAL, DOUBLE, FLOAT(p), DECIMAL(p,s), NUMERIC
  - Časy – DATE, TIME, TIMESTAMP, DATETIME, YEAR
  - Texty – CHAR(d), VARCHAR(d), TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT
  - Množiny – SET (h1, h2, ...)
  - Výčty – ENUM(h1, h2, ...)
  - Data – BINARY, VARBINARY
- Omezení domény
  - NOT NULL – nesmí být NULL
  - AUTO\_INCREMENT – automaticky generované unikátní číslo
  - UNIQUE – musí se lišit
  - DEFAULT hod – automaticky se doplní, pokud není zadána
- Omezení v tabulce
  - PRIMARY KEY
  - FOREIGN KEY
  - INDEX
  - CHECK (výraz)

# Klíče (znovu)

- Necht'  $K \subseteq R$ .  $K$  je superklíč schématu  $R$ , když hodnoty  $K$  stačí k jednoznačné identifikaci  $r(R)$ 
  - Např.  $\{klient\_jmeno, klient\_mesto\}$  je superklíčem pro schéma  $Klient\_schema$ . Superklíčem je však i  $\{klient\_jmeno\}$
- $K$  je kandidát na klíč jestliže  $K$  je minimální superklíč
  - Např.  $\{klient\_jmeno\}$  je kandidátem na klíč pro schéma  $Klient\_schema$ , neboť je to superklíč a žádná „podmnožina“ již superklíčem není
- **Primární klíč** je vybrán mezi kandidátními klíči tak, aby se během „života“ příslušné relace neměnil
  - Např.  $\{klient\_jmeno\}$  může sloužit jako primární klíč pro naši instanci relace, avšak když přijde další *Novák*, všechno bude špatně
    - e-mailová adresa může být primárním klíčem, avšak lidé svůj e-mail občas mění (což je jiný typ komplikace)

# Cizí klíče

- Relační schéma může obsahovat atribut, který koresponduje s primárním klíčem v jiné relaci. Takový atribut se nazývá **cizí klíč**
  - Např. atributy *customer\_name* a *account\_number* relačního schématu *depositor* jsou cizí klíče do *customer* a *account*
  - Hodnotami cizího klíče v **referencující (odkazující) relaci** smí být jen ty hodnoty, které se vyskytují jako primární klíč v **relaci referencované (odkazované)**
    - Důležitý typ omezení – **referenční integrita**





# Relační algebra

- Relační algebra definuje operace nad relacemi
- Šest základních operátorů
  - Selekcce (restrikce)  $\sigma$ 
    - Výběr jen některých prvků relace
  - Projekce:  $\Pi$ 
    - Výběr jen určitých atributů
  - Sjednocení:  $\cup$ 
    - Spojení několika relací v jednu (spojované relace musí mít stejné schéma)
  - Rozdíl (množin):  $-$ 
    - Výběr těch prvků první relace, které nejsou obsaženy v druhé relaci
  - Kartézský součin:  $\times$ 
    - Klasická množinová operace
  - Přejmenování:  $\rho$ 
    - Změna jména jednoho či více atributů
- Všechny tyto operátory pracují s jednou nebo dvěma relacemi a vytváří relaci novou

# Selekce

- Zápis  $\sigma_p(r)$ 
  - $p$  je selekční predikát
- Definice

$$\sigma_p(r) = \{t \mid t \in r \wedge p(t)\}$$

Selekční predikát  $p$  je výroková formule (=podmínka) složená z termů propojených logickými operátory:  $\wedge$  (**and**),  $\vee$  (**or**),  $\neg$  (**not**)  
Každý term má tvar:

$\langle \text{atribut} \rangle \text{ op } \langle \text{atribut} \rangle$  nebo  $\langle \text{konstanta} \rangle$ ,

kde  $op$  je jeden z  $=, \neq, >, \geq, <, \leq$

- Příklad selekce:  $\sigma_{\text{klient\_mesto}=\text{"Praha"}}(r)$

	A	B	C	D
$r$	$\alpha$	$\alpha$	1	7
	$\alpha$	$\beta$	5	7
	$\beta$	$\beta$	12	3
	$\beta$	$\beta$	23	10

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

A	B	C	D
$\alpha$	$\alpha$	1	7
$\beta$	$\beta$	23	10

# SQL select

- Univerzální příkaz SQL (nejjednodušší verze)

```
select [all|distinct|distinctrow] * from  
jméno where p;
```

- *jméno* je jméno vytvářené relace
- *p* je podmínka pro řádky relace *jméno*
- Operace =, <>, >, <, >=, <=,
  - *BETWEEN* – uvedení rozmezí
  - *LIKE* – prohledávání řetězce na výskyt vzoru
  - *IN* – pro sloupce typu *SET* – množina
- Operace *AND*, *OR*, *NOT*, (, )
- *DISTINCT* – zaručí, že ve výsledku budou jen různé hodnoty
- *ALL* – naopak zahrne všechny hodnoty i vícekrát

- Příklad

```
select distinct * from zákazník where město = 'Brno'  
or psč = 11000;
```

# Projekce

- Zápis:  $\Pi_{A_1, A_2, \dots, A_k}(r)$   
kde  $A_1, A_2$  jsou jména atributů a  $r$  je jméno relace
- Výsledek je definován jako relace s  $k$  atributy („sloupci“) vytvořená z relace  $r$  výběrem pouze vyjmenovaných atributů
  - Tedy vynecháním zbývajících (neuvezených) atributů
  - Duplicitní prvky (řádky) jsou odstraněny – relace jsou množiny!
- Příklad: V relaci *klient* nás nezajímá atribut *klient\_ulice*

$\Pi_{\text{klient\_jmeno, klient\_mesto}}(\text{klient})$

	A	B	C
$r$	$\alpha$	10	1
	$\alpha$	20	1
	$\beta$	30	1
	$\beta$	40	2

	A	C
$\Pi_{A,C}(r)$	$\alpha$	1
	$\alpha$	1
	$\beta$	1
	$\beta$	2

	A	C
=	$\alpha$	1
	$\beta$	1
	$\beta$	2

# SQL select

- Univerzální příkaz SQL

```
select [all|distinct|distinctrow]  $s_1, s_2, \dots, s_k$   
from jméno;  
– jméno je jméno vytvářené relace  
–  $s_1, s_2, \dots, s_k$  je seznam sloupců relace jméno
```

- Příklad

```
select psč from zákazník;
```

*Rozdílo mezi algebrou a programem – v algebře relace nemůže obsahovat dva úplně stejné řádky, v sql dotazu ano.*

# Klauzule `select` (pokr.)

- Hvězdička v klauzuli `select` značí “všechny atributy”

**select** \* **from** *loan*

- Klauzule **select** může obsahovat aritmetické výrazy obsahující operace +, −, \*, / a konstanty nebo atributy

- Dotaz

**select** *loan\_number, branch\_name, amount \* 100*  
**from** *loan*

vrátí relaci shodnou s *loan* až na to, hodnota atributu *amount* bude vynásobena 100

- Jde vlastně o zobecněnou projekci

$\Pi_{loan\_number, branch\_name, amount * 100}(loan)$

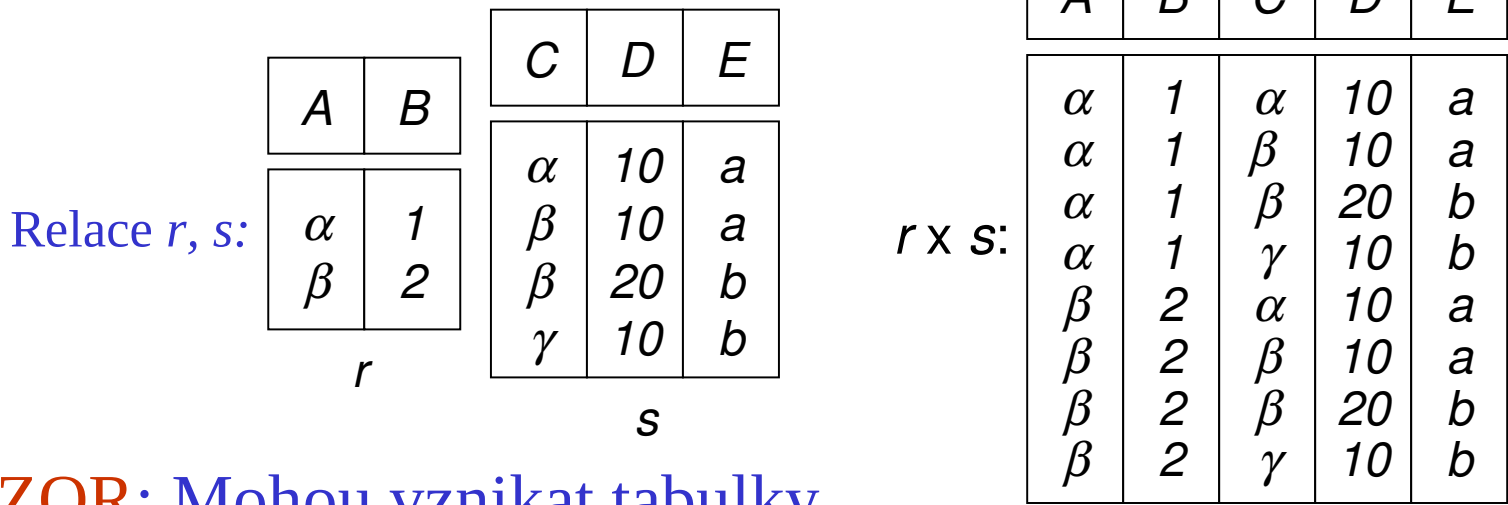
# Kartézský součin

- Zápis:  $r \times s$
- Definice:

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

Předpokládejme, že atributy  $r(R)$  a  $s(S)$  jsou disjunktní tj.,  $R \cap S = \emptyset$ .

- Lze použít i na více než dvě relace
  - Nejsou-li atributy disjunktní, tzn. některé atributy  $r(R)$  mají stejné jméno jako jména atributů v  $s(S)$ , musí se použít operace *přejmenování*
- ➔



- **POZOR:** Mohou vzniknout tabulky gigantické velikosti

# SQL select

- Univerzální příkaz SQL

```
select * from jméno1, jméno1;
```

- *jméno1* a *jméno2* jsou jména relací které tvoří kartézský součin

- Příklad

```
select * from zákazník, objednávka;
```



# Základní struktura SQL dotazu

Typický SQL dotaz má tvar:

**select**  $A_1, \dots, A_n$  **from**  $R_1, R_2, \dots, R_m$  **where**  $p$ ;

$A_i$  jsou atributy,  $R_i$  jsou relace a  $p$  je predikát

Tento dotaz je ekvivalentní relačnímu výrazu:

$$\Pi_{A_1, A_2, \dots, A_n} (\sigma_p (R_1 \times R_2 \times \dots \times R_m))$$

- **Důležité poznatky**

- SQL je deklarativní (dotazovací) jazyk, zatímco relační algebra je procedurální
- Zobrazení SQL dotazů na relační výrazy převádí deklarativní dotazy na procedury
- Provedení („výpočet výsledku“) dotazu bude implementovat procedury operací relační algebry

# Sjednocení

- Zápis:  $r \cup s$

- Definice:

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

- Relace  $r$  a  $s$  musí být **kompatibilní**, tj

1.  $r$  a  $s$  musí mít stejnou aritu (počet atributů)

2. Domény atributů musí být po řadě shodné

- Např. druhý atribut relace  $r$  a druhý atribut relace  $s$  musí mít shodný datový typ (definiční doménu)

- **Příklad:**

– najít všechny zákazníky banky, kteří mají vklad nebo půjčku

$$\Pi_{customer\_name}(depositor) \cup \Pi_{customer\_name}(borrower)$$

Relace  $r, s$ :

A	B
$\alpha$	1
$\alpha$	2
$\beta$	1

$r$

A	B
$\alpha$	2
$\beta$	3

$s$

$r \cup s$ :

A	B
$\alpha$	1
$\alpha$	2
$\beta$	1
$\beta$	3

# SQL union

- Univerzální příkaz SQL (nejjednodušší verze)

```
select  $A_1, \dots, A_n$  from jméno1
```

```
union
```

```
select  $A_1, \dots, A_n$  from jméno2;
```

- *jméno1* a *jméno2* jsou jména relací, které mají některé sloupce ( $A_1, \dots, A_n$ ) shodné

- Příklad

```
select psč from zákazník
```

```
union
```

```
select psč from objednávka;
```

# Vložení

- Vložení v relační algebře je opět přiřazení

$$r \leftarrow r \cup E$$

kde  $r$  je relace, do níž vkládáme a  $E$  je relační výraz

- Vložení jediného prvku se realizuje tak, že  $E$  bude konstantní výraz popisující prvek
  - Vložit lze najednou i více prvků, pokud  $E$  bude relační výraz kompatibilní s  $r$
- **Příklad**
    - Vlož do databáze informaci, že zákazník Kovář má účet A-973 se zůstatkem 1200 v pobočce Benešov

$$\begin{aligned} \text{account} &\leftarrow \text{account} \cup \{(\text{"A-973"}, \text{"Benešov"}, 1200)\} \\ \text{depositor} &\leftarrow \text{depositor} \cup \{(\text{"Kovář"}, \text{"A-973"})\} \end{aligned}$$

# SQL insert into

- Přidání řádku do tabulky SQL

**insert into** *jméno1* ( $A_1, \dots, A_n$ ) **values** ( $V_1, \dots, V_n$ );

- *jméno1* je jméno relace
- lze zadat jen některé sloupce ( $A_1, \dots, A_n$ )
- a pro ně hodnoty ( $V_1, \dots, V_n$ )

- Příklad

**insert into** *zákazník* **values** ( 'Jahoda',  
'Pardubice', '53002' );

# Rozdíl

- Zápis:  $r - s$
- Definice:  
$$r - s = \{t \mid t \in r \wedge t \notin s\}$$
- Relace vstupující do množinového rozdílu musí opět být vzájemně kompatibilní

Relace  $r, s$ :

$A$	$B$
$\alpha$	1
$\alpha$	2
$\beta$	1

$r$

$A$	$B$
$\alpha$	2
$\beta$	3

$s$

$r - s$ :

$A$	$B$
$\alpha$	1
$\beta$	1

# Operace přejmenování

- Pomocná operace
  - Fakticky nejde o pravou operaci relační algebry, zavádí se z pragmatických důvodů
  - Umožňuje nově pojmenovat (a tím i referencovat) výsledek jiné relační operace
  - Umožňuje též pojmenovat relaci více jmény

- **Příklad:**

$$\rho_X(E)$$

vrátí výsledek výrazu  $E$  pod jménem  $X$

- Jestliže relační výraz  $E$  má aritu  $n$ , pak

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

vrátí výsledek výrazu  $E$  pod jménem  $X$  s atributy přejmenovanými na  $A_1, A_2, \dots, A_n$ .

# Operace přejmenování

- SQL umožňuje relace a atributy pomocí klauzule **as**  
*old-name as new-name*
  - Najdi jména, čísla půjček a dlužné částky všech zákazníků a pojmenuj sloupec *loan\_number* jako *loan\_id*

```
select customer_name, borrower.loan_number as loan_id, amount
from borrower, loan
where loan_id = loan.loan_number
```

- Domácí úkol:
  - Přepište tento dotaz do formy relačního výrazu



# *n*-tice jako proměnné

- Proměnné ve tvaru *n*-tic se definují jako proměnné v klauzuli **from** s použitím klauzule **as**
- Příklad

- Najdi jména zákazníků, čísla jejich půjček a výši dluhů přes všechny pobočky

```
select customer_name, B.loan_number, L.amount  
from borrower as B, loan as L  
where B.loan_number = L.loan_number
```

- Najdi jména poboček, které mají součet vkladů (*assets*) větší některá z poboček v Praze 1

```
select distinct T.branch_name  
from branch as T, branch as S  
where T.assets > S.assets  
and S.branch_city = "Praha 1"
```

# Skládání operací

- Skutečně užitečné relační operace vzniknou teprve skládáním operací základních

$r \times s$ :

A	B	C	D	E
$\alpha$	1	$\alpha$	10	a
$\alpha$	1	$\beta$	10	a
$\alpha$	1	$\beta$	20	b
$\alpha$	1	$\gamma$	10	b
$\beta$	2	$\alpha$	10	a
$\beta$	2	$\beta$	10	a
$\beta$	2	$\beta$	20	b
$\beta$	2	$\gamma$	10	b

$\sigma_{A=C}(r \times s)$ :

A	B	C	D	E
$\alpha$	1	$\alpha$	10	a
$\beta$	2	$\beta$	10	a
$\beta$	2	$\beta$	20	b

# Příklad bankovní databáze

- Relace

- $branch(branch\_name, branch\_city, assets)$
- $customer(customer\_name, customer\_street, customer\_city)$
- $account(account\_number, branch\_name, balance)$
- $loan(loan\_number, branch\_name, amount)$
- $depositor(customer\_name, account\_number)$
- $borrower(customer\_name, loan\_number)$

- Příklady dotazů

- Najdi všechny půjčky ( $loan$ ) přes 1200

$$\sigma_{amount > 1200}(loan)$$

- Najdi čísla půjček vyšších než 1200

$$\Pi_{loan\_number}(\sigma_{amount > 1200}(loan))$$

- Najdi jména zákazníků majících vkladový účet v pobočce Nymburk

$$\Pi_{customer\_name}(\sigma_{branch\_name = Nymburk}$$

$$(\sigma_{depositor.account\_number = account.account\_number}(depositor \times account)))$$

# Příklad bankovní databáze (2)

- Další příklady dotazů

- Najdi jména zákazníků majících půjčku v pobočce ‘Nymburk’ a přitom nemají vkladový účet v žádné pobočce

$$\Pi_{customer\_name} \left( \sigma_{branch\_name = Nymburk} \left( \sigma_{borrower.loan\_number = loan.loan\_number} (borrower \times loan) \right) \right) \\ - \Pi_{customer\_name} (depositor)$$

- Najdi jména zákazníků, kteří mají půjčku vedenou v pobočce Nymburk

- 1. možnost

$$\Pi_{customer\_name} \left( \sigma_{branch\_name = Nymburk} \left( \sigma_{borrower.loan\_number = loan.loan\_number} (borrower \times loan) \right) \right)$$

- 2. možnost
- $$\Pi_{customer\_name} \left( \sigma_{borrower.loan\_number = loan.loan\_number} \left( \sigma_{branch\_name = Nymburk} (borrower) \right) \times loan \right)$$

# Příklad bankovní databáze (3)

- Příklady dotazu (použití operace *přejmenování*)
  - Najdi největší zůstatek vkladového účtu
  - Strategie:
    - Najdi zůstatky, které nejsou největší
    - K tomu účelu přejmenuj relaci *account* na *temp*, abychom mohli porovnávat jednotlivé zůstatky se všemi ostatními
    - Použij množinový rozdíl k nalezení těch zůstatků, které nejsou mezi těmi, které jsme určili v předchozím kroku
  - Dotaz pak vypadá takto:

$$\Pi_{balance}(account) - \Pi_{account.balance}(\sigma_{account.balance < temp.balance}(account \times \rho_{temp}(account)))$$

# Doplňkové operace, průnik

- Z praktických důvodů se definují další operátory, které umožňují zjednodušení častých dotazů do databáze
  - Průnik
  - Přirozené spojení (spojení přes rovnost)
  - Dělení
  - Přiřazení
- Průnik
  - Zápis:  $r \cap s$
  - Definice:
$$r \cap s = \{ t \mid t \in r \wedge t \in s \}$$
  - Předpoklad: Relace  $r$  a  $s$  jsou vzájemně kompatibilní
  - Poznámka:  $r \cap s = r - (r - s)$

Relace  $r, s$ :

A	B
$\alpha$	1
$\alpha$	2
$\beta$	1

$r$

A	B
$\alpha$	2
$\beta$	3

$s$

$r \cap s$ :

A	B
$\alpha$	2

# Množinové operace v SQL

- Množinové operátory **union**, **intersect** a **except** jsou SQL ekvivalentem relačních (množinových) operací

$\cup$ ,  $\cap$  a  $-$

- Najdi zákazníky mající vkladový účet nebo půjčku (nebo oboje)

```
(select customer_name from depositor)
union
(select customer_name from borrower)
```

- Najdi zákazníky mající jak vkladový účet tak půjčku

```
(select customer_name from depositor)
intersect
(select customer_name from borrower)
```

- Najdi zákazníky mající vkladový účet a nemající půjčku

```
(select customer_name from depositor)
except
(select customer_name from borrower)
```

- SQL má dále operátor **in**, který testuje příslušnost či členství v množině

- ekvivalent  $\in$  Relační model dat a jazyk SQL

# Přirozené spojení

- Zápis:  $r \bowtie s$
- Necht'  $r$  a  $s$  jsou relace podle schémat  $R$  a  $S$ .  
 $r \bowtie s$  je pak relace podle schématu  $R \cup S$  vytvořená jako:
  - Uvažme všechny páry  $n$ -tic  $t_r$  z  $r$  a  $t_s$  z  $s$
  - Jestliže  $t_r$  a  $t_s$  mají stejné hodnoty všech atributů z  $R \cap S$ , pak  $n$ -tice  $t$  se objeví ve výsledku, přičemž  $t$  má stejné hodnoty atributů jako  $t_r$  na  $r$  a  $t$  má stejné hodnoty atributů jako  $t_s$  na  $s$
- Výsledek přirozeného spojení je tedy množina všech kombinací „řádků“ z  $R$  a  $S$ , které mají shodné hodnoty stejnojmenných atributů
- Příklad:  
 $R = (A, B, C, D)$   
 $S = (E, B, D)$ 
  - Výsledné schéma =  $(A, B, C, D, E)$
  - $r \bowtie s$  pak je  $\{ r.C, r.D, s.E \mid r.B=s.B \wedge r.D=s.D \}$  ( $r \times s$ )



# Přirozené spojení – příklad

- Relace  $r, s$ :

A	B	C	D
$\alpha$	1	$\mu$	a
$\beta$	2	$\gamma$	a
$\gamma$	4	$\beta$	b
$\alpha$	1	$\gamma$	a
$\delta$	2	$\beta$	b

B	D	E
1	a	$\alpha$
3	a	$\beta$
1	a	$\gamma$
2	b	$\delta$
3	b	$\epsilon$

$r \bowtie s$ :

A	B	C	D	E
$\alpha$	1	$\mu$	a	$\alpha$
$\alpha$	1	$\alpha$	a	$\gamma$
$\alpha$	1	$\gamma$	a	$\alpha$
$\alpha$	1	$\gamma$	a	$\gamma$
$\delta$	2	$\beta$	b	$\delta$

$r$

$s$

- Praktický příklad

*Zamestanec*

Jmeno	ZamId	Oddel
Franta	1235	Finance
Pavla	2241	Obchod
Josef	3401	Výroba
Petr	2202	Výroba

*Oddeleni*

Odddel	Manager
Finance	Jirka
Obchod	Petr
Vyroba	Karel

*Zamestanec*  $\bowtie$  *Oddeleni*

Jmeno	ZamId	Oddel	Manager
Franta	1235	Finance	Jirka
Pavla	2241	Obchod	Petr
Josef	3401	Výroba	Karel
Petr	2202	Výroba	Karel

# Vnější spojení

- **Vnější spojení** je operace, která rozšiřuje přirozené spojení a zamezuje „ztrátě informace“
  - Určí se přirozené spojení a pak se přidají prvky z jedné ze spojovaných relací, které nesplňují požadavky na rovnost stejnojmenných atributů
  - Podle toho, ze které relace se přidávají prvky, rozlišuje se **levé vnější spojení** a **pravé vnější spojení**
  - Lze též přidat prvky z obou spojovaných relací a pak jde o **plné vnější spojení**
  - Při doplňování mohou vznikat prvky s neznámými nebo nedefinovanými hodnotami, pro jejichž reprezentaci se zavádí hodnota ***null***

# Typy a příklady vnějšího spojení

*loan*

loan_number	branch_name	amount
L-170	Praha 1	3000
L-230	Nymburk	4000
L-260	Benešov	1700

*borrower*

customer_name	loan_number
Jonáš	L-170
Kovář	L-230
Sláma	L-155

přirozené spojení  
⋈

*loan* ⋈ *borrower*

loan_number	branch_name	amount	customer_name
L-170	Praha 1	3000	Jonáš
L-230	Nymburk	4000	Kovář

levé vnější spojení  
⋈<sub>L</sub>

*loan* ⋈<sub>L</sub> *borrower*

loan_number	branch_name	amount	customer_name
L-170	Praha 1	3000	Jonáš
L-230	Nymburk	4000	Kovář
L-260	Benešov	1700	<i>null</i>

pravé vnější spojení  
⋈<sub>R</sub>

*loan* ⋈<sub>R</sub> *borrower*

loan_number	branch_name	amount	customer_name
L-170	Praha 1	3000	Jonáš
L-230	Nymburk	4000	Kovář
L-155	<i>null</i>	<i>null</i>	Sláma

plné vnější spojení  
⋈<sub>LR</sub>

*loan* ⋈<sub>LR</sub> *borrower*

loan_number	branch_name	amount	customer_name
L-170	Praha 1	3000	Jonáš
L-230	Nymburk	4000	Kovář
L-260	Benešov	1700	<i>null</i>
L-155	<i>null</i>	<i>null</i>	Sláma

# Spojení relací v SQL

- Základní syntaxe je
$$r_1 \langle \text{Typ} \rangle \mathbf{join} r_2 \mathbf{on} \langle \text{podmínka} \rangle$$
  - Typicky se používá jako součást pod-dotazu v klauzuli **from**.
- **Typ** spojení – "přívlastek" klíčového slova **join**
  - Jde o úplnou ekvivalenci se spojeními z relační algebry
  - Typy: **inner join**, **left outer join**, **right outer join**, **full outer join**
- **Spojovací podmínka**
  - určuje, na základě čeho má dojít ke spojení a které atributy budou ve výsledném spojení
  - Pokud se neuvede provede se na základě rovnosti sloupců se shodným jménem
- **Příklad**
  - Najdi všechny zákazníky, kteří mají buď půjčku nebo vkladový účet, ale ne oboje

```
select customer_name
from (depositor full outer join borrower )
where account_number is null
       or loan_number is null
```

# Operace dělení

- Zápis:  $r \div s$ 
  - Určeno pro dotazy obsahující frázi „pro všechny“
    - Necht'  $r$  a  $s$  jsou relace podle schémat  $R$  a  $S$ , kde  $R = (A_1, \dots, A_m, B_1, \dots, B_n)$  a  $S = (B_1, \dots, B_n)$
    - Výsledkem  $r \div s$  je relace dle schématu  $R - S = (A_1, \dots, A_m)$   
 $r \div s = \{ t \mid t \in \Pi_{R-S}(r) \wedge \forall u \in s (tu \in r) \}$ , kde  $tu$  značí zřetězení „řádků“  $t$  a  $u$  chápané jako jediná  $n$ -tice
- Vlastnost
  - Necht'  $q = r \div s$ , pak  $q$  je největší relace splňující  $q \times s \subseteq r$
- Definice pomocí základních operací relační algebry
  - Necht'  $r(R)$  a  $s(S)$  jsou relace a necht'  $S \subseteq R$   
$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$
    - ∇  $\Pi_{R-S,S}(r)$  přeuspořádá atributy  $r$
    - ∇  $\Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)$  dá ty  $n$ -tice  $t$  z  $\Pi_{R-S}(r)$ , pro které platí, že některá  $n$ -tice  $u \in s$  je taková, že  $tu \notin r$

# Operace dělení – příklad

- Relace  $r, s$ :

A	B
$\alpha$	1
$\alpha$	2
$\alpha$	3
$\beta$	1
$\gamma$	1
$\delta$	1
$\delta$	3
$\delta$	4
$\varepsilon$	6
$\varepsilon$	1
$\beta$	2

$r$

$B$
-----

1
2

$s$

$r \div s$ :

$A$
-----

$\alpha$
$\beta$

- Praktický příklad

*Pracuje\_pro*

Jmeno	Manager
Franta	Jirka
Pavla	Petr
Josef	Karel
Petr	Karel

*Šéf*

Manager
Jirka
Karel

*Pracuje\_pro  $\div$  Šéf*

Jmeno
Franta
Josef
Petr

# Přiřazovací operace

- Přiřazovací operace ( $\leftarrow$ ) umožňuje pohodlný zápis složitých výrazů
  - Dovoluje zapisovat „dotazy“ ve formě sekvence programových příkazů ve tvaru série přiřazení následovaných snáze čitelnými výrazy
  - Přiřazuje se vždy vhodné pracovní „proměnné typu relace“
  - Pracovní proměnné jsou pak dostupné v dalších výrazech

- **Příklad:** Operaci dělení  $r \div s$  lze zapsat jako

$$temp1 \leftarrow \Pi_{R,S}(r)$$

$$temp2 \leftarrow \Pi_{R,S}((temp1 \times s) - \Pi_{R,S,S}(r))$$

$$vysledek = temp1 - temp2$$

# Příklad bankovní databáze – další dotazy

- Najdi jména všech zákazníků, kteří mají současně vkladový účet a půjčku

$$\Pi_{customer\_name} (borrower) \cap \Pi_{customer\_name} (depositor)$$

- Najdi jména zákazníků, kteří mají půjčku, a výši této půjčky

$$\Pi_{customer\_name, loan\_number, amount} (borrower \bowtie loan)$$

- Najdi jména všech zákazníků, kteří mají vkladový účet v pobočce Nymburk nebo Benešov

– Možnost 1

$$\begin{aligned} & \Pi_{customer\_name} (\sigma_{branch\_name = \text{“Nymburk”}} (depositor \bowtie account)) \\ & \cup \Pi_{customer\_name} (\sigma_{branch\_name = \text{“Benešov”}} (depositor \bowtie account)) \end{aligned}$$

– Možnost 2

$$\begin{aligned} & \Pi_{customer\_name, branch\_name} (depositor \bowtie account) \\ & \div \rho_{temp(branch\_name)} (\{ \text{“Nymburk”}, \\ & \text{“Benešov”} \}) \end{aligned}$$

- Všimněme si, že Možnost 2 používá „konstantní relaci“ *temp* ve funkci dělitele při dělení – ptáme se totiž „pro všechny uvedené pobočky“



# Pragmatická rozšíření relačních operátorů

- Pro často kladené dotazy se zavádějí rozšířené operace
  - Zobecněná projekce
  - Agregátní funkce
  - Vnější spojení (*Outer Join*)
- Zobecněná projekce zavádí aritmetické funkce do seznamu možných výstupních atributů

$$\prod_{F_1, F_2, \dots, F_n} (E)$$

- $E$  je relační výraz a  $F_1, F_2, \dots, F_n$  jsou aritmetické výrazy zahrnující atributy ze schématu výrazu  $E$  a konstanty
- Takto se získají odvozené (počítané) atributy
- **Příklad:**
  - Relace *credit\_info(customer\_name, limit, credit\_balance)*,
  - Urči, kolik může každá osoba ještě utratit:

$$\prod_{customer\_name, limit - credit\_balance} (credit\_info)$$

# Agregátní funkce a operace

- **Agregátní funkce** pracují s kolekcí hodnot a vrací jedinou výslednou hodnotu

**avg**: průměrná hodnota

**min**: minimum

**max**: maximum

**sum**: součet hodnot

**count**: počet hodnot

- **Agregátní operace** relační algebry vytvářejí relaci se „syntetickými“ atributy a případným seskupováním prvků

$$G_1, G_2, \dots, G_m \quad \theta \quad F_1(A_1), F_2(A_2), \dots, F_n(A_n) \quad (E)$$

- $E$  je relační výraz
- $G_1, G_2, \dots, G_m$  je seznam atributů, podle nich se má seskupovat (může být i prázdný)
- $F_i$  jsou agregátní funkce
- $A_i$  jsou jména atributů ze schématu, podle něhož je tvořen  $E$

# Příklad agregátních operací a funkcí

- Relace  $r$ :

A	B	C
$\alpha$	$\alpha$	7
$\alpha$	$\beta$	7
$\beta$	$\beta$	3
$\beta$	$\beta$	10

$\vartheta_{\text{sum}(C)}(r)$ :

sum(C)
27

- Relace  $account$  seskupená podle  $branch\_name$ :

$branch\_name$	$account\_number$	$balance$
Nymburk	A-102	400
Nymburk	A-201	900
Praha 1	A-217	750
Praha 1	A-215	750
Benešov	A-222	700

$branch\_name$	sum( $balance$ )
Nymburk	1300
Praha 1	1500
Benešov	700

$\vartheta_{\text{sum}(balance)}(account)$

# Agregátní funkce v SQL

- Tyto funkce pracují s multisety hodnot a vrací hodnotu jedinou
  - jinak jsou shodné s dříve uvedenými agregátními funkcemi **avg**, **min**, **max**, **sum** a **count**

- Najdi průměrný vklad v pobočce Benešov

```
select avg(balance)  
from account  
where branch_name = "Benešov"
```

- Urči počet vkladatelů

```
select count (distinct customer_name)  
from depositor
```

# Agregátní funkce v SQL

- Tyto funkce pracují s multisety hodnot a vrací hodnotu jedinou
  - jinak jsou shodné s dříve uvedenými agregátními funkcemi **avg**, **min**, **max**, **sum** a **count**

- Najdi průměrný vklad v pobočce Benešov

```
select avg(balance)  
from account  
group by branch_name;
```

- Urči počet vkladatelů

```
select count (distinct customer_name)  
from depositor
```

# Hodnoty *Null*

- *null* se užívá pro neznámou hodnotu nebo pro označení situace, že hodnota neexistuje
  - Aritmetický výraz obsahující *null* dává výsledek *null*
  - Agregátní funkce ignorují hodnoty *null*
  - Pro eliminaci duplikátů a seskupování se *null* uvažuje jako jakákoliv jiná hodnota; dvě *null* hodnoty se považují za identické
- Predikáty zahrnující *null* vyžadují tříúrovňovou logiku s doplňkovou hodnotou *unknown*
  - Logika s pravdivostní hodnotou *unknown*:
    - OR: (*unknown* **or** *true*) = *true*,  
(*unknown* **or** *false*) = *unknown*  
(*unknown* **or** *unknown*) = *unknown*
    - AND: (*true* **and** *unknown*) = *unknown*,  
(*false* **and** *unknown*) = *false*,  
(*unknown* **and** *unknown*) = *unknown*
    - NOT: (**not** *unknown*) = *unknown*
  - Selekční predikát vyhodnocený jako *unknown* se považuje za *false*

# Hodnoty *null* v SQL

- Predikát **is null** slouží k testu *null* hodnot
  - Např.: V relaci *loan* vyhledej čísla půjček s *null* hodnotou atributu *amount*  
**select** *loan\_number* **from** *loan*  
**where** *amount* **is null**
- Aritmetické operace zahrnující *null* dávají *null*
  - Např.:  $5 + \text{null}$  vrací *null*
- Agregátní funkce *null* hodnoty ignorují
- Je zavedena tříhodnotová logika s *unknown*
  - Např.:  $5 < \text{null}$ ,  $\text{null} <> \text{null}$  nebo  $\text{null} = \text{null}$  se vždy vyhodnotí jako *unknown*
- Konstrukt ***p* is unknown** se vyhodnotí jako pravdivý, pokud predikát *p* má hodnotu *unknown*

# Modifikace relací v databázi

- K modifikaci obsahu databáze potřebujeme operace
  - Deletion (výmaz = odstranění prvku z relace)
  - Insertion (vložení prvku do relace)
  - Updating (aktualizace – změna prvku v relaci)
- Vše se realizuje operátorem přiřazení
- Výmaz (*deletion*)

$$r \leftarrow r - E$$

kde  $r$  je relace a  $E$  je relační výraz určující mazané prvky

- Příklady

- Vymaž všechny záznamy v pobočce Benešov

$$account \leftarrow account - \sigma_{branch\_name = \text{“Benešov”}}(account)$$

- Vymaž všechny záznamy o půjčkách se zůstatkem 0 až 50

$$loan \leftarrow loan - \sigma_{amount \geq 0 \text{ and } amount \leq 50}(loan)$$



# Aktualizace

- Mechanismus pro změnu hodnoty zvolených atributů, aniž by se měnily hodnoty všech atributů

- Použije se zobecněná projekce

$$r \leftarrow \prod_{F_1, F_2, \dots, F_l} (r)$$

- $F_i$  je buď

- $i$ -tý atribut  $r$ , pokud  $i$ -tý atribut nemá být změněn, nebo
- $F_i$  je výraz sestavený z konstant a atributů  $r$ , který dává novou hodnotu atributu

- Příklady

- Připočti úrok 5%

$$account \leftarrow \prod_{account\_number, branch\_name, balance * 1.05} (account)$$

- Přičti úrok 6% k účtům se zůstatkem přes 10.000 a 5% ke všem ostatním

$$account \leftarrow \prod_{account\_number, branch\_name, balance * 1.06} (\sigma_{balance > 10000} (account)) \\ \cup \prod_{account\_number, branch\_name, balance * 1.05} (\sigma_{balance \leq 10000} (account))$$

# SQL příkazy pro modifikaci databáze

- **Výmaz** (*deletion*)
  - Příkaz má strukturu **delete-from-where** s argumenty analogickými konstruktu **select-from-where**
  - Vymaž všechny vkladové účty v pobočce Nymburk  
**delete from** *account* **where** *branch\_name* = 'Nymburk'
- **Vložení** (*insertion*)
  - **insert into** relace **values** <kompatibilní\_relace>
  - Přidej záznam do tabulky *account*  
**insert into** *account* (*branch\_name*, *balance*, *account\_number*) **values** ('Beroun', 1200, 'A-9732')
- **Aktualizace** (*update*)
  - **update** relace **set** atribut = výraz **where** podmínka
  - Přidej 6% prémie ke vkladovým účtům přes 1000  
**update** *account* **set** *balance* = *balance* \* 1.06  
**where** *balance* > 1000

# Vnořené dotazy

- SQL má mechanismus pro vnořování dotazů (*subquery*)
  - někdy zvané pod-dotazy
- **Vnořený dotaz** má obvyklý tvar **select-from-where**, je však zanořen do jiného dotazu
  - Nejčastěji se používá k realizaci testu členství v relaci, porovnávání množin a určování kardinality relací
- **Příklad:**
  - Najdi zákazníky mající jak vkladový účet tak i půjčku

```
select distinct customer_name from borrower
```

```
where customer_name in (select customer_name from depositor)
```

Vnořený dotaz

# Pohledy

- Často je nevhodné poskytovat uživateli všechna data
  - tedy celý logický model databáze a všechny uložené relace
  - Bankovní úředník na jisté pozici potřebuje znát jméno zákazníka a pobočku, kde má půjčku, ne však výši půjčky.  
(**select** *customer\_name, branch\_name* **from** *borrower, loan*  
**where** *borrower.loan\_number = loan.loan\_number* )
- Mechanismus **pohledů** (*view*) umožňuje skrýt určitá data
  - Lze tak vytvořit jakoukoliv relaci, která není součástí konceptuálního modelu a zpřístupnit ji uživateli jako "virtuální relaci". Taková "virtuální relace" se nazývá **pohled**.
- Zavede se příkazem **create view** ve tvaru  
**create view v as** <formulace dotazu>  
kde v je jméno pohledu
  - Jakmile je pohled definován, jeho jméno lze používat jako zkratku celého definičního dotazu

# SQL přípouští duplikáty

- Pro zajištění dobré analogie SQL a množinového modelu potřebujeme tzv. **multiset**
  - Multiset je množina s opakujícími se prvky
- Potřebujeme multisetové verze relačních operátorů mezi relacemi  $r_1$  a  $r_2$ 
  - $\sigma_\theta(r_1)$ : Je-li  $c_1$  kopií  $n$ -tice  $t_1$  v  $r_1$ , a  $t_1$  splňuje selekční predikát  $\theta$ , pak bude  $c_1$  kopií  $t_1$  v  $\sigma_\theta(r_1)$ .
  - $\Pi_A(r)$ : Pro každou kopii  $t_1$  v  $r_1$  bude kopie  $\Pi_A(t_1)$  i v  $\Pi_A(r_1)$ 
    - $r_1 \times r_2$ : Je-li  $c_1$  kopií  $t_1$  v  $r_1$  a  $c_2$  kopií  $t_2$  v  $r_2$ , pak bude  $c_1 * c_2$  kopií  $n$ -tice  $t_1 \dots t_2$  v  $r_1 \times r_2$
- **Příklad:**
  - Multisetové relace  $r_1 (A, B)$  a  $r_2 (C)$  jsou  
 $r_1 = \{(1, a) (2, a)\}$      $r_2 = \{(2), (3), (3)\}$
  - Pak  $\Pi_B(r_1)$  bude  $\{(a), (a)\}$ ,  
a  $\Pi_B(r_1) \times r_2$  dá  $\{(a, 2), (a, 2), (a, 3), (a, 3), (a, 3), (a, 3)\}$
- SQL sémantika příkazu **select**  $A_1, A_2, \dots, A_n$  **from**  $r_1, r_2, \dots, r_m$   
**where**  $P$  je ekvivalentní multisetové verzi výrazu

$$\Pi_{A_1, A_2, \dots, A_n} (\sigma_P (r_1 \dot{\cup} r_2 \dot{\cup} \dots \times r_m))$$



# Dotazy