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, …, D_n, pak relací R rozumíme libovolnou podmnožinu kartézského součinu $D_1 \times D_2 \times \ldots \times D_n$.
- Relace tedy je množina *n*-tic $(a_1, a_2, \ldots, a_n)$, kde $a_i \in D_i$
- Příklad:
  - *klient_jmeno* = {Novák, Mates, Braun, Novotný …}
    /* množna jmen klientů */
  - *klient_ulice* = {Spálená, Hlavní, Horní, …} /* množina jmen ulic*/
  - *klient_mesto* = {Praha, Brno, Nymburk, …} /* množina jmen měst */
  - pak *r* = {
    (Novák, Spálená, Praha),
    (Mates, Horní, Brno),
    (Braun, Hlavní, Brno),
    (Novotný, Horní, Nymburk)
    }

    je relace, tj. podmnožina *klient_jmeno* x *klient_ulice* x *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/
- 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* ≡ *BRANCH_NAME* ≡ *branch_name*

SQL připouští duplikáty

- Pro zajištění dobré analogie SQL a množinového modelu potřebujeme tzv. multisety
  - 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 θ, 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 $t_1 \ldots t_2$ 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`** $A_1, A_2, \ldots, A_n$ **`from`** $r_1, r_2, \ldots, r_m$ **`where`** P je ekvivalentní multisetové verzi výrazu

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

SQL `create table`

- Relace v SQL je definována příkazem
  `create [temporary] table [if not exists]`
  jméno (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integritní-omezení$_1$), ..., (integritní-omezení$_k$)) [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`

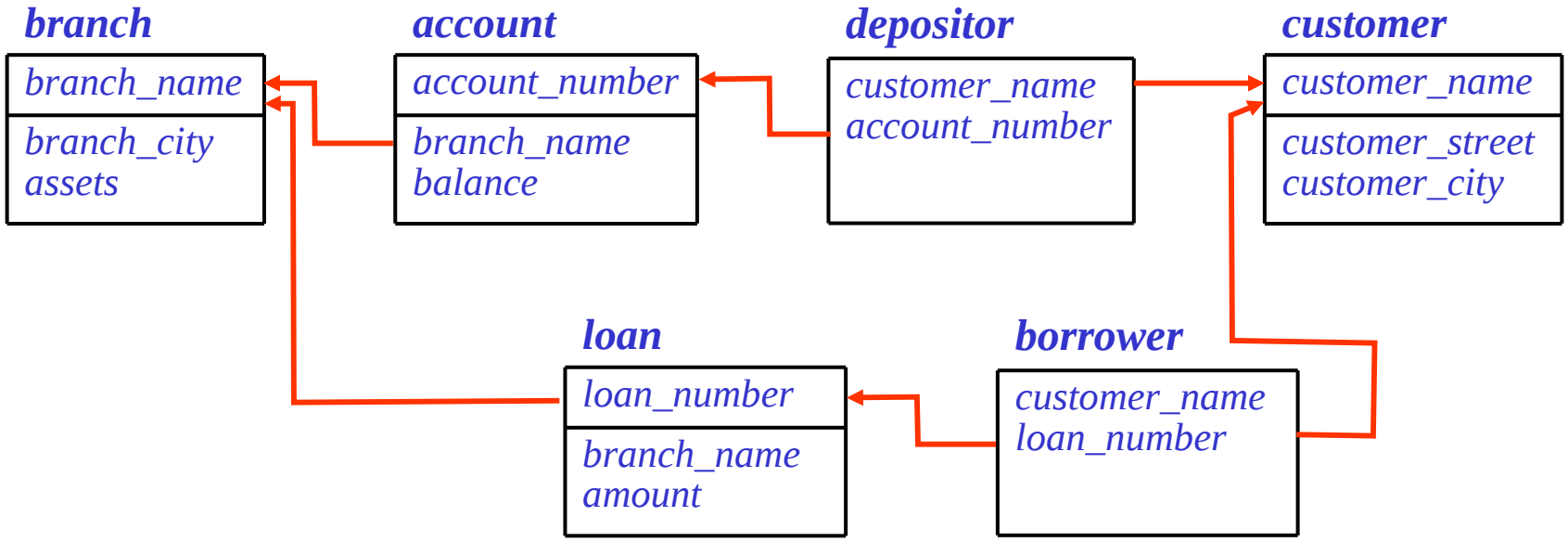
- Domény
  - Celá čísla – BIT, TINYINT,SMALLINT,MEDIUMINT, INT,INTEGER(p),BIGINT
  - Reálná čísla – REAL, DOUBLE, FLOAT(p), DECIMAL(p,s), NUMERIC
  - Časy – DATE, TIME, TIMESTAMP, DATETIME, YEAR
  - Texty – CHAR(d), VARCHAR(d), TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT
  - Množiny – SET (h1, h2, …)
  - Výčty – ENUM(h1, h2, …)
  - Data – BINARY, VARBINARY
- Omezení domény
  - NOT NULL – nesmí být NULL
  - AUTO_INCREMENT – automaticky generováné 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)

- Nechť $K \subseteq R$. 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
  - Selekce (restrikce) σ
    - Výběr jen některých prvků relace
  - Projekce: ∏
    - Výběr jen určitých atributů
  - Sjednocení: ∪
    - 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: x
    - Klasická množinová operace
  - Přejmenování: ρ
    - 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:

⟨atribut⟩ *op* ⟨atribut⟩ nebo ⟨konstanta⟩,

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

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

r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

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

A	B	C	D
α	α	1	7
β	β	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: $\Pi_{A_1, A_2, \ldots, A_k}(r)$

  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 (neuvedených) atributů
  - Duplicitní prvky (řádky) jsou odstraněny – relace jsou množiny!
- Příklad: V relaci *klient* nás nezajímá atribut *klient_ulice*

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

r

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

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

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

SQL `select`

- Univerzální příkaz SQL

`select [all|distinct|distinctrow]` $s_1, s_2, .., s_k$
`from` *jméno;*

  - *jméno* je jméno vytvářené relace
  - s_1, s_2, …, s_k je seznam sloupců relace *jméno*

- Příklad

`select` *psč* **`from`** *zákazník;*

Rozdílo mezi algebrou a programem – v algebře relace nemůže osahovat 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:

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

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* ➔

Relace *r*, *s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

r x *s*:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

- POZOR: Mohou vznikat 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 $A_1,..,A_n$ **from** $R_1,R_2,..,R_m$ **where** p;

A_i jsou atributy, R_i jsou relace a p je predikát

Tento dotaz je ekvivalentní relačnímu výrazu:

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

- 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
α	1
α	2
β	1

r

A	B
α	2
β	3

s

$r \cup s$:

A	B
α	1
α	2
β	1
β	3

SQL union

- Univerzální příkaz SQL (nejjednodušší verze)

```
select A1,..,An from jméno1
union
select A1,..,An from jméno2;
```

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

$$account \leftarrow account \cup \{(\text{“A-973”}, \text{“Benešov”}, 1200)\}$$
$$depositor \leftarrow depositor \cup \{(\text{“Kovář”}, \text{“A-973”})\}$$

SQL `insert into`

- Přidání řádku do tabulky SQL

`insert into` *jméno1* $(A_1,..,A_n)$ **`values`** $(V_1,..,V_n)$;

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

- 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*
α	1
α	2
β	1

r

A	*B*
α	2
β	3

s

r – s:

A	*B*
α	1
β	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

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

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

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
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

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

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	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

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

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

$$\Pi_{customer_{name}}\left(\sigma_{branch_{name} = Nymburk}\left(\sigma_{borrower.loan\_number = loan.loan\_number}\left(borrower \times loan\right)\right)\right)$$

    - 2. možnost

$$\Pi_{customer_{name}}\left(\sigma_{borrower.loan\_number = loan.loan\_number}\left(\left(\sigma_{branch_{name} = Nymburk}\left(borrower\right)\right) \times loan\right)\right)$$

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
α	1
α	2
β	1

r

A	B
α	2
β	3

s

$r \cap s$:

A	B
α	2

Množinové operace v SQL

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

Přirozené spojení

- Zápis: $r \bowtie s$
- Nechť 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: $\prod_{r.A, r.B, r.C, r.D, s.E} \left(\sigma_{r.B = s.B \wedge r.D = s.D} \left(r \times s \right) \right)$

Přirozené spojení – příklad

- Relace *r*, *s*:

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

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	∈

s

r ⋈ *s*:

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

- Praktický příklad

Zamestnanec

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

Oddeleni

Odddel	Manager
Finance	Jirka
Obchod	Petr
Vyroba	Karel

Zamestnanec ⋈ *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 ⟕

loan ⟕ borrower

loan_number	branch_name	amount	customer_name
L-170	Praha 1	3000	Jonáš
L-230	Nymburk	4000	Kovář
L-260	Benešov	1700	*null*

pravé vnější spojení ⟖

loan ⟖ borrower

loan_number	branch_name	amount	customer_name
L-170	Praha 1	3000	Jonáš
L-230	Nymburk	4000	Kovář
L-155	*null*	*null*	Sláma

plné vnější spojení ⟗

loan ⟗ borrower

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

Spojení relací v SQL

- Základní syntaxe je

  r_1 <Typ> **`join`** r_2 **`on`** <podmínka>

  - 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“
  - Nechť r a s jsou relace podle schémat R a S, kde $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$ a $S = (B_1, \ldots, B_n)$
  - Výsledkem $r \div s$ je relace dle schématu $R - S = (A_1, \ldots, A_m)$

    $r \div s = \{ t \mid t \in \prod_{R-S}(r) \wedge \forall u \in s\ (tu \in r) \}$, kde tu značí zřetězení „řádků“ t a u chápané jako jediná n-tice
- Vlastnost
  - Nechť $q = r \div s$, pak q je největší relace splňující $q \times s \subseteq r$
- Definice pomocí základních operací relační algebry
  - Nechť $r(R)$ a $s(S)$ jsou relace a nechť $S \subseteq R$

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

    $\forall$ $\prod_{R-S,S}(r)$ přeuspořádá atributy r

    $\forall$ $\prod_{R-S}(\prod_{R-S}(r) \times s) - \prod_{R-S,S}(r))$ dá ty n-tice t z $\prod_{R-S}(r)$, pro které platí, že některá n-tice $u \in s$ je taková, že $tu \notin r$

Operace dělení – příklad

- Relace *r*, *s*:

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

r

B
1
2

s

r÷*s*:

A
α
β

- Praktický příklad

Pracuje_pro

Jmeno	Manager
Franta	Jirka
Pavla	Petr
Josef	Karel
Petr	Karel

Šéf

Manager
Jirka
Karel

Pracuje_pro ÷ Šéf

Jmeno
Franta
Josef
Petr

Přiřazovací operace

- Přiřazovací operace (←) 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

$$temp1 \leftarrow \prod_{R\text{-}S} (r)$$

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

$$vysledek = temp1 - temp2$$

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

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

  - Možnost 2

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

    - 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

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

  - E je relační výraz a F_1, F_2, …, 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

$$G_1, G_2, \ldots, G_m \; \vartheta_{F_1(A_1),\, F_2(A_2),\, \ldots,\, F_n(A_n)} \left(E \right)$$

  - E je relační výraz
  - $G_1, G_2, \ldots, G_m$ 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*
α	α	7
α	β	7
β	β	3
β	β	10

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

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

$_{branch\_name}\vartheta_{sum(balance)}(account)$:

branch_name	sum(*balance*)
Nymburk	1300
Praha 1	1500
Benešov	700

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
groupe 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 + *null* vrací *null*
- Agregátní funkce *null* hodnoty ignorují
- Je zavedena tříhodnotová logika s *unknown*
  - Např.: 5 < *null*, *null* <> *null* nebo *null* = *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

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

  - 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 distict` *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 připouští duplikáty

- Pro zajištění dobré analogie SQL a množinového modelu potřebujeme tzv. multisety
  - 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 θ, 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 $t_1 \ldots t_2$ 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`** $A_1, A_2, \ldots, A_n$ **`from`** $r_1, r_2, \ldots, r_m$ **`where`** P je ekvivalentní multisetové verzi výrazu

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

Dotazy