

# Agregácia v SQL

Ján Mazák

FMFI UK Bratislava

prakt. cvicenia: jedn. agregacia + create view

P4

create table DML insert, update, delete CTE / WITH rekurzia

teor. cv: insert, update, delete; prakt. cv:

# Množinové operácie

- ▶ EXCEPT — rozdiel množín
- ▶ EXCEPT ALL? — rozdiel množín
- ▶ UNION — zjednotenie (bez zachovania duplicity riadkov)
- ▶ UNION ALL — zjednotenie (so zachovaním duplicity riadkov)
- ▶ INTERSECT — prienik TODO
- ▶ INTERSECT ALL — prienik TODO

# Množinové operácie

```
SELECT nazov  
FROM kriky
```

UNION

```
SELECT nazov  
FROM stromy
```

```
SELECT rodne_cislo  
FROM employee
```

EXCEPT

```
SELECT rodne_cislo  
FROM customer  
WHERE sales > 100
```

# Zoradenie záznamov

Zoznam zamestnancov usporiadaný od najvyššieho platu po najnižší, pri rovnakom plate abecedne:

```
SELECT name, salary, department  
FROM employee  
ORDER BY salary DESC, name ASC
```

# Vnorené dotazy (subqueries)

Okrem operátora EXISTS sa možno pomocou operátora IN (resp. NOT IN) pýtať na prítomnosť v množine.

```
SELECT name
FROM employee e
WHERE e.deptno IN (SELECT deptno
                  FROM department d
                  WHERE d.location = 'New York')
```

IN sa optimalizuje horšie ako join a antijoin (NOT EXISTS), dá sa však v prípade potreby prepísať pomocou iných operátorov.

## Vnorené dotazy (subqueries)

Ak je výsledkom vnoreného dotazu tabuľka 1x1, možno ju použiť ako skalár.

```
SELECT name
FROM employee e
WHERE e.deptno = (SELECT deptno
                  FROM department d
                  WHERE d.location = 'New York')
```

Ak by vo vnorenom dotaze vyšlo viac riadkov (alebo žiaden), dôjde k chybe.

(Takýmto chybám treba čo najviac predchádzať, napr. použitím UNIQUE už pri vytváraní tabuľky.)

# VIEW

Používateľom možno uľahčiť prístup k dátam vytvorením náhľadu (VIEW), ktorý je iný ako samotné definície tabuliek.

```
CREATE VIEW employee_details AS (  
    SELECT e.empno, e.name, e.deptno,  
           d.name, d.location  
    FROM employee e, department d  
    WHERE e.deptno = d.deptno  
)
```

```
SELECT * FROM employee_details
```

VIEW je permanentný objekt v db. Možno cezeň aj vkladať dáta, ale nerobte to (veľa komplikácií s NULL).



# CREATE TEMPORARY TABLE

Používateľ si v rámci 1 transakcie vie sám vytvoriť dočasnú tabuľku (nemusí byť materializovaná).

```
CREATE TEMPORARY TABLE employee_details AS (  
    SELECT e.empno, e.name, e.deptno,  
           d.name, d.location  
    FROM employee e, department d  
    WHERE e.deptno = d.deptno  
);  
SELECT * FROM employee_details;
```

Niekedy nás nezaujímajú jednotlivé záznamy relácie, ale jedna agregátna hodnota.

```
/* pocet zamestnancov */
```

```
SELECT COUNT(e.empno)  
FROM employee e
```

Agregačné funkcie: COUNT, SUM, AVG, MAX, MIN...

# Agregácia

Riadky možno rozdeliť do viacerých skupín pomocou GROUP BY. Na výstupe bude pre každú skupinu 1 riadok.

```
/* pocet zamestnancov v jednotlivych oddeleniach */
```

```
SELECT e.deptno, COUNT(e.empno) AS c  
FROM employee e  
GROUP BY e.deptno  
HAVING COUNT(e.empno) > 1
```

HAVING umožňuje filtrovať skupiny.

(Pomenovanie atribútov vytvorených agregáciou za SELECT sa nedá použiť za HAVING.)

# Agregácia

Mimo dotazu môžu ísť len agregované hodnoty a atribúty, na ktorých sa všetky záznamy v skupine zhodujú (toto je z pohľadu databázy zaručené len vtedy, ak sa ten atribút nachádza za GROUP BY).

`/* pocet zamestnancov v jednotlivych oddeleniach */`

```
SELECT d.name, COUNT(e.empno) AS c
FROM employee e
      JOIN department d ON e.deptno = d.deptno
GROUP BY d.deptno, d.name
```

(Pridanie d.name nemá vplyv na rozdelenie riadkov do skupín.)

Do skupín možno deliť aj podľa viacerých atribútov súčasne (riadky v skupine sa musia zhodovať na všetkých).

```
/* pocet zamestnancov v skupinách podľa platu  
v jednotlivých oddeleniach */
```

```
SELECT d.name, d.salary, COUNT(e.empno) AS c  
FROM employee e  
      JOIN department d ON e.deptno = d.deptno  
GROUP BY d.deptno, d.name, e.salary
```

Postupujte opatrne pri aplikovaní agregáčnych funkcií na NULL a na potenciálne prázdnu množinu záznamov. Výsledky neraz nie sú intuitívne, treba podrobne naštudovať dokumentáciu a overiť správanie pre konkrétny DBMS. Napríklad:

- ▶ `COUNT(stĺpec)` ignoruje NULL, ale `COUNT(*)` ich zaráta (aspoň v MySQL)
- ▶ `AVG` pre neprázdnu množinu ignoruje NULL a výsledok tak môže byť veľmi nereprezentatívny; pre prázdnu vráti NULL

- ▶ <https://www.postgresqltutorial.com/postgresql-aggregate-functions/>
- ▶ <https://www.postgresqltutorial.com/> (Sections 4, 5, 7)
- ▶ <https://learnsql.com/blog/error-with-group-by/>
- ▶ <https://www.postgresql.org/docs/current/functions-aggregate.html>
- ▶ <https://www.postgresqltutorial.com/postgresql-views/managing-postgresql-views/>
- ▶ <https://drive.google.com/file/d/1HCq2KMZ05UvtXGe1nTqNmkwhc3X-NLI3/view>

# Úlohy: SQL

Databáza: *lubi*(Pijan, Alkohol), *capuje*(Krcma, Alkohol, Cena),  
*navstivil*(Id, Pijan, Krcma), *vypil*(Id, Alkohol, Mnozstvo)

- ▶ počet čapovaných alkoholov
- ▶ priemerná cena piva
- ▶ najdrahší čapovaný alkohol (všetky, ak ich je viac)
- ▶ max. objem alkoholu vypitý jedným pijanom
- ▶ max. suma prepitá pri jednej návšteve krčmy
- ▶ krčma s najväčšou celkovou tržbou
- ▶ pijan, ktorý vypil najmenej druhov alkoholu
- ▶ koľko najviac alkoholov, ktoré nik neľúbi, je v jednej krčme v ponuke?