

Agregácia v SQL

Ján Mazák

FMFI UK Bratislava

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

```
/* počet zamestnancov */
```

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

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

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

`/* počet zamestnancov v jednotlivých oddeleniach */`

```
SELECT e.dept_id, COUNT(e.emp_id) AS c
FROM employee e
GROUP BY e.dept_id
```

Agregácia

HAVING umožňuje filtrovať skupiny. (Pomenovanie atribútov vytvorených agregáciou za **SELECT** sa nedá použiť za **HAVING**.)

```
/* počet zamestnancov v oddeleniach s > 1 zamestnancom */
```

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

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).

`/* počet zamestnancov v jednotlivých oddeleniach */`

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

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

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

Agregácia — chyby

Vnárание agregovaných funkcií je neprípustné:

```
SELECT MAX(COUNT(...)) ...
```

```
/* správne: najprv COUNT,  
   potom MAX v ďalšom SELECTe */
```

Agregačnú funkciu nemožno aplikovať priamo na reláciu:

```
WHERE x = MAX(r) ...
```

```
/* správne: x = (SELECT MAX(...) FROM r) */
```


Dvojitá agregácia sa veľmi ťažko číta, neraz nefunguje (kvôli duplicite hodnôt vznikajúcej pri karteziánskom súčine) a je zdrojom chýb:

```
FROM r, s
```

```
...
```

```
HAVING COUNT(r.x) = COUNT(s.y)
```

```
/* lepšie rozdeliť do osobitných SELECTov */
```

WITH — Common Table Expressions (CTE)

WITH vytvorí reláciu existujúcu len počas výpočtu dotazu.

```
WITH pijanPocetAlkoholov(pijan, c) AS (  
    SELECT pijan, COUNT(DISTINCT alkohol)  
    FROM lubi  
    GROUP BY pijan  
)  
SELECT MAX(ppa.c)  
FROM pijanPocetAlkoholov ppa
```

WITH sa často používa pri viackrokovom agregovaní. V jednom dotaze možno za WITH vymenovať aj viacero relácií oddelených čiarkou.

Záznamy, kde sa dosahuje extrém (arg max)

```
WITH pijanPocetAlkoholov(pijan, c) AS (  
    SELECT pijan, COUNT(DISTINCT alkohol)  
    FROM lubi  
    GROUP BY pijan  
)  
SELECT ppa.pijan  
FROM pijanPocetAlkoholov ppa  
WHERE ppa.c = (  
    SELECT MAX(ppa2.c)  
    FROM pijanPocetAlkoholov ppa2  
)
```

Vnorené dotazy (subqueries)

Podľa toho, či výsledok vnoreného dotazu závisí od riadka, pre ktorý sa vyhodnocuje WHERE, rozlišujeme:

- ▶ **nekorelované** — nezávisí
- ▶ **korelované** — závisí

Nekorelované dotazy stačí počítať raz, kým korelované musíme počítať pre každý riadok nanovo. Navyše narúšajú optimalizáciu (prepis dotazu do výpočtovo výhodnejšej podoby), pretože vo všeobecnosti je ťažké hľadať súvis medzi programom a podprogramom. Korelované vnorené dotazy sú preto niekedy veľmi pomalé; vyhýbajte sa im.

Vnorené dotazy — NOT IN

Zamestnanci, ktorých nadriadený nie je priamym podriadeným prezidenta.

Uncorrelated subquery: 10^5

```
SELECT e.emp_id, e.superior
FROM employee e
WHERE e.superior NOT IN (
    SELECT s.emp_id
    FROM employee s
    JOIN employee p
      ON s.superior = p.emp_id
    WHERE p.job = 'president'
);
```

Antijoin: 10^5

```
SELECT e.emp_id, e.superior
FROM employee e
WHERE NOT EXISTS (
    SELECT 1
    FROM employee s
    JOIN employee p
      ON s.superior = p.emp_id
    WHERE p.job = 'president'
      AND s.emp_id = e.superior
);
```

Vnorené dotazy — scalar subquery

Zamestnanci zarábajúci viac ako priemer v ich oddelení.

Correlated subquery: 10^{10}

```
SELECT e.emp_id, e.salary
FROM employee e
WHERE e.salary > (
    SELECT AVG(e2.salary)
    FROM employee e2
    WHERE e2.dept_id =
        e.dept_id
);
```

Join: 10^5

```
WITH dept_avg AS (
    SELECT dept_id,
        AVG(salary) AS avg_salary
    FROM employee
    GROUP BY dept_id
)
SELECT e.emp_id, e.salary
FROM employee e
    JOIN dept_avg da
        ON e.dept_id = da.dept_id
WHERE e.salary > da.avg_salary;
```

Vnorené dotazy — ALL

Zamestnanci zarábajúci viac ako všetci ich podriadení.

Correlated subquery: 10^{11}

```
SELECT e.name
FROM employee e
WHERE e.salary > ALL (
  SELECT s.salary
  FROM employee s
  WHERE s.superior = e.emp_id
);
```

Antijoin: 10^5

```
SELECT e.name
FROM employee e
WHERE NOT EXISTS (
  SELECT 1
  FROM employee s
  WHERE s.superior = e.emp_id
  AND e.salary <= s.salary
);
```

Vnorené dotazy — ANY

Zamestnanci zarábajúci viac ako nejaký ich podriadený.

Correlated subquery: 10^{11}

```
SELECT e.name
FROM employee e
WHERE e.salary > ANY (
    SELECT s.salary
    FROM employee s
    WHERE s.superior = e.emp_id
);
```

Join: 10^5

```
SELECT e.name
FROM employee e
    JOIN employee sub
        ON sub.superior = e.emp_id
WHERE e.salary > sub.salary;
```


Vnorené dotazy

Nevýhody correlated subquery:

- ▶ Niekedy horšia čitateľnosť (inokedy lepšia).
- ▶ Ťažšie debugovanie (lebo korelovaný dotaz nie je sám osebe zmysluplný a je previazaný s kontextom).
- ▶ Komplexita sťažuje budúce udržiavanie/modifikáciu kódu.
- ▶ Horšie využívanie existujúcich indexov (vytvorenie ad-hoc zoznamu pre IN, ALL atď. preruší prepojenie na existujúce tabuľky).
- ▶ Prekážka pre optimalizáciu (ťažko sa hľadá alternatívny spôsob výpočtu); čím viac dát, tým väčší problém.
- ▶ Treba si strážiť prítomnosť NULL.

Vnorené dotazy za FROM — fuj, radšej WITH

```
SELECT outer_query.emp_id, outer_query.name, outer_query.total_salary
FROM (
  SELECT middle_query.emp_id,
    middle_query.salary + COALESCE(middle_query.bonus, 0) AS total_salary
  FROM (
    SELECT e.emp_id, e.name, e.salary,
      (SELECT SUM(b.amount)
        FROM (
          SELECT bonus.emp_id, bonus.amount
          FROM employee_bonus bonus
          WHERE bonus.amount > 1000
        ) AS filtered_bonus
      WHERE filtered_bonus.emp_id = e.emp_id
    ) AS bonus
  FROM (
    SELECT emp_id, name, salary
    FROM employee
    WHERE salary > 50000
  ) AS e
  ) AS middle_query
) AS outer_query
WHERE outer_query.total_salary > 60000;
```

- ▶ <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>

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)
- ▶ pijan, ktorý vypil najmenej druhov alkoholu
- ▶ tržby jednotlivých krčiem
- ▶ krčma s najväčšou celkovou tržbou
- ▶ priem. suma prepitá pri 1 návšteve pre jednotlivé krčmy
- ▶ koľko najviac alkoholov, ktoré nik neľúbi, je v jednej krčme v ponuke?