

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.)

Agregácia

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

```
/* počet zamestnancov v skupinách  
podľa platu zaokrúhleného na stovky  
v jednotlivých oddeleniach */
```

```
SELECT d.name, ROUND(e.salary, -2) AS 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, ROUND(e.salary, -2)
```

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 — GROUPING SETS

Možnosť viac GROUP BY v jednom dotaze:

```
SELECT region, product, SUM(sales)
FROM sales
GROUP BY GROUPING SETS (
    (region),           -- Sales by region
    (product),          -- Sales by product
    (region, product),  -- Sales by region and product
    ()                  -- Grand total
)
```

Vhodné na rýchlejšie generovanie reportov.
Špeciálne prípady CUBE a ROLLUP.

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  
)
```

Pri výpočte **arg max** nemožno použiť ORDER BY a LIMIT, pretože LIMIT vedie k fixnému počtu záznamov vo výsledku a dopredu nevieme, pre koľko záznamov sa maximum nadobúda.

Pomocou ORDER BY a LIMIT 1 možno nájsť samotné maximum, nie je to však vhodný spôsob:

- ▶ úmysel takého dotazu je skrytý (ORDER BY a LIMIT sa používajú najmä na niečo iné, čiže akoby ste pomocou špecifického LIMIT 1 menili význam ORDER BY)
- ▶ výpočet je neefektívny (treba uložiť celý medzivýsledok, možno aj na disk, ak je veľký; až tak možno aplikovať ORDER BY)

Agregácia — chyby

Vnorenie 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 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  
a použiť join alebo jednu z agregácií  
vypočítať vopred */
```

Vnorené dotazy (inner/nested query, subquery)

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.

Vnorené dotazy

Vnorené dotazy využívajúce IN, NOT IN, ANY, ALL apod. možno prepísať pomocou joinu alebo **antijoinu** (WHERE NOT EXISTS). Antijoin počíta tie riadky z ľavej tabuľky, ktoré sa nejoinujú so žiadnymi riadkami pravej tabuľky.

Antijoin možno počítať podobne ako join *rýchlo* — napr. cez triedenie alebo hashovanie. Pre korelované vnorené dotazy takúto optimalizáciu DBMS zväčša spraviť nevie.

„Najhorší“ prípad: vnorený dotaz sa počíta úplne oddelene od vonkajšieho, napr. pre IN sa najprv zostaví celý zoznam hodnôt, hoci z nich nakoniec potrebujeme len jednu. V pláne výpočtu reprezentované ako **SubPlan**.

Vnorené dotazy

DBMS sa **nepodarilo** optimalizovať IN cez JOIN.

```
/* employees working in the same department as their superior */  
SELECT e1.name FROM employee e1  
WHERE e1.superior IS NOT NULL AND e1.dept_id IN (  
    SELECT e2.dept_id FROM employee e2  
    WHERE e2.emp_id = e1.superior  
);
```

Seq Scan on employee e1

Filter: ((superior IS NOT NULL) AND (SubPlan 1))

SubPlan 1

-> Seq Scan on employee e2

Filter: (emp_id = e1.superior)

Vnorené dotazy

DBMS sa **podarilo** optimalizovať IN cez rýchly JOIN.

```
/* employees working in the same department as their superior */  
SELECT e1.name FROM employee e1  
WHERE e1.superior IS NOT NULL AND e1.dept_id IN (  
    SELECT e2.dept_id FROM employee e2  
    WHERE e2.emp_id = e1.superior  
);
```

Hash Join

```
Hash Cond: (e2.emp_id = e1.superior)  
Join Filter: (e2.dept_id = e1.dept_id)  
-> Seq Scan on employee e1  
    Filter: (superior IS NOT NULL)  
-> Hash  
    -> Seq Scan on employee e2
```

Vnorené dotazy — NULL v NOT IN

Dotaz vracia **prázdny** výsledok kvôli NULL v NOT IN.

```
/* employees who are not superiors */  
SELECT e1.name FROM employee e1  
WHERE e1.emp_id NOT IN (  
    SELECT e2.superior FROM employee e2  
);
```

Podmienka NOT IN sa vyhodnotí ako

```
e1.emp_id <> id1 AND e1.emp_id <> id2 AND ...  
... AND e1.emp_id <> NULL
```

Pretože $x \neq \text{NULL}$ sa vyhodnotí ako UNKNOWN, celá podmienka zlyhá a dotaz nevráti žiadne riadky.

Vnorené dotazy — NULL v NOT IN

Operátor NOT EXISTS správne spracuje NULL.

```
/* employees who are not superiors (correct version) */  
SELECT e1.name FROM employee e1  
WHERE NOT EXISTS (  
    SELECT 1 FROM employee e2  
    WHERE e2.superior = e1.emp_id  
);
```

Ak e2.superior je NULL, podmienka
e2.superior = e1.emp_id
sa nevyhodnotí ako TRUE, takže dotaz funguje správne.

Vnorené dotazy — NOT EXISTS vs. EXCEPT

NOT EXISTS vždy **možno ekvivalentne prepísať cez EXCEPT**.
Niekedy je to zrozumiteľnejšie.

```
/* version with NOT EXISTS */  
SELECT e1.name FROM employee e1  
WHERE NOT EXISTS (  
    SELECT 1 FROM employee e2  
    WHERE e2.superior = e1.emp_id  
);
```

```
/* equivalent version with EXCEPT */  
SELECT e.emp_id, e.name  
FROM employee e  
EXCEPT  
SELECT e.emp_id, e.name  
FROM employee e  
    JOIN employee s ON e.emp_id = s.superior;
```

(Prečo nerobíme množinový rozdiel len pre mená, ale pridali sme aj id?)

Vnorené dotazy

Kombinácia EXCEPT a NOT EXISTS neraz čitateľnejšia ako vnáranie NOT EXISTS.
(Rýchlosť závisí od obsahu tabuliek; porovnanie je komplikované.)

```
SELECT d.dept_id FROM department d
      WHERE NOT EXISTS (
        SELECT job FROM employee j
        WHERE NOT EXISTS (
          SELECT 1 FROM employee e
          WHERE e.dept_id = d.dept_id AND e.job = j.job
        )
      );
```

```
/* verzia kombinujúca NOT EXISTS a EXCEPT */
```

```
-- všetky oddelenia
```

```
SELECT dept_id FROM department
```

```
EXCEPT
```

```
-- oddelenia, ktorým chýba aspoň jedno povolanie
```

```
SELECT d.dept_id FROM department d, employee j
```

```
WHERE NOT EXISTS (
```

```
  SELECT 1 FROM employee e
```

```
  WHERE e.dept_id = d.dept_id AND e.job = j.job
```

```
);
```

Vnorené dotazy — scalar subquery

Zamestnanci zarábajúci viac ako priemer v ich oddelení.

Correlated subquery:

```
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:

```
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 — NOT IN

Zamestnanci, ktorých nadriadený nie je priamym podriadeným prezidenta.

Uncorrelated subquery:

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

Antijoin:

```
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 — ALL

Zamestnanci zarábajúci viac ako všetci ich podriadení.

Correlated subquery:

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

Antijoin:

```
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 menej ako nejaký ich podriadený.

Correlated subquery:

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

Join:

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

Vnorené dotazy — zhrnutie

Nevýhody IN, NOT IN, ANY, ALL:

- ▶ Prekážka pre optimalizáciu (ťažko sa hľadá rýchly spôsob výpočtu); čím viac dát, tým väčší problém.
- ▶ Horšie využívanie existujúcich indexov (vytvorenie ad-hoc zoznamu pre IN, ALL atď. preruší prepojenie na existujúce tabuľky).
- ▶ Treba si strážiť prítomnosť NULL.

Používajte radšej WHERE NOT EXISTS či EXCEPT (a pre zložitý vnorený dotaz si skúste pripraviť pomocnú tabuľku so zmysluplným názvom cez WITH — to pri optimalizácii vôbec neprekáža, keďže ide len o deklaráciu a DBMS nemá povinnosť ju materializovať).

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>

Na zamyslenie

- ▶ Prejdite si túto prezentáciu pred každým písaním dotazov, najmä ukážky chýb a vhodných postupov. Niektorí študenti tieto chyby opakujú celý rok, vrátane písomky, ktorú potom musia písať opakovane.
- ▶ Aký je rozdiel medzi filtrovaním pomocou HAVING a WHERE?
- ▶ Aký je rozdiel medzi COUNT(x) a COUNT(DISTINCT x) z hľadiska výsledku a rýchlosti výpočtu?
- ▶ Ako by sme spočítali počet zamestnancov, ktorých mená začínajú jednotlivými písmenami abecedy?

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?