Rekurzia a window functions

Ján Mazák

FMFI UK Bratislava

Window functions

- Použitie GROUP BY vedie k nahradeniu celej skupiny jediným riadkom.
- Niekedy však chceme zachovať pôvodné riadky a len k nim doplniť hodnotu súvisiacu s inými riadkami v skupine.
- Riešenie: window functions. Tieto funkcie vidia nielen jeden riadok, ale aj „okolité“ riadky (presná definícia je súčasťou dotazu v časti OVER).
- Window functions nevidia riadky odfiltrované vo WHERE a HAVING.
- Ak chceme filtrovať výsledok podľa vypočítanej hodnoty window funkcie, treba použiť vnorený dotaz.

Window functions

```
SELECT emp_id, name, dept_id, salary,
       avg(salary) OVER (PARTITION BY dept_id) avg_s
FROM employee;
```

```
 emp_id | name   | dept_id | salary  | avg_s
--------+--------+---------+---------+---------------
     11 | King   |      10 | 5000.00 | 2600.00
     13 | Clark  |      10 | 1500.00 | 2600.00
     24 | Miller |      10 | 1300.00 | 2600.00
     14 | Jones  |      20 | 2975.00 | 2175.00
     ...
```

Window functions

```
SELECT dept_id, name, salary,
       rank() OVER (PARTITION BY dept_id
                    ORDER BY salary DESC)
FROM employee;
```

```
 dept_id |  name   | salary   | rank
---------+---------+----------+----------------------
      10 | King    | 5000.00 |    1
      10 | Clark   | 1500.00 |    2
      10 | Miller  | 1300.00 |    3
      20 | Ford    | 3000.00 |    1
      20 | Scott   | 3000.00 |    1
      20 | Jones   | 2975.00 |    3
      20 | Adams   | 1100.00 |    4
    ...
```

Window functions

Funkcie, ktoré možno použiť ako window functions:

- bežné agregačné funkcie
  (môžu sa však správať trochu odlišne!)
- rank(), percent_rank()
- nth_value() — element in the n-th row of the window
- ...

Window functions

- Ak vynecháme PARTITION BY, partícia obsahuje všetky riadky výstupu.

```
SELECT salary, sum(salary) OVER ()
FROM employee;
```

Window functions

- Ak vynecháme PARTITION BY, partícia obsahuje všetky riadky výstupu.

```
SELECT salary, sum(salary) OVER ()
FROM employee;
```

- Okno možno pomenovať a použiť viackrát.

```
SELECT
    sum(salary) OVER w,
    avg(salary) OVER w
FROM employee
WINDOW w AS (PARTITION BY dept_id
             ORDER BY salary DESC);
```

Window frame

- Každý riadok má svoje vlastné okno (window frame) — zoznam riadkov, nad ktorými sa vykoná window function.
- Okno závisí od použitej funkcie. Default window frame možno v rámci dotazu modifikovať.
- Komplikovaná syntax. Nasledujúce dve formulácie sú ekvivalentné významovo, ale vraj nie z hľadiska efektívnosti výpočtu...

```
ROWS UNBOUNDED PRECEDING EXCLUDE CURRENT ROW
ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING
```

Window frame

```
SELECT salary, sum(salary) OVER (ORDER BY salary)
FROM employee;
```

```
 salary  |   sum
---------+---------
  800.00 |   800.00
  950.00 |  1750.00
 1100.00 |  2850.00
 1300.00 |  4150.00
 1500.00 |  7150.00   <-- pribudlo 3000, nie 1500
 1500.00 |  7150.00
 1750.00 |  8900.00
```

Okno: riadky v partícii po daný riadok plus riadky s rovnakou hodnotou.

Array datatype

Databázy bežne podporujú polia (nie však SQLite).
Dátový typ v PostgreSQL:

```
CREATE TABLE x (
    y  integer[4],
    z  text[][]
);
```

Zápis v súlade s SQL štandardom:

```
CREATE TABLE x (
    y  integer ARRAY[4],
    z  text ARRAY[][]
);
```

Aktuálna implementácia veľkosť poľa ignoruje.

Arrays

Pole možno vytvoriť priamo v rámci dotazu.

```
SELECT ARRAY[1,2] || ARRAY[3,4] AS a;
SELECT ARRAY[1, 2] || '{3, 4}';
```

Na prácu s poľom existuje množstvo funkcií, napr. `array_prepend`, `array_length`.

```
SELECT array_position(
     ARRAY['mon','tue','wed','thu','fri','sat','sun'],
     'mon'
  );
SELECT * FROM x WHERE 47 = ANY (some_array);
SELECT * FROM x WHERE 47 = ALL (some_array);
```

Arrays

Pole tiež možno získať aplikovaním agregačnej funkcie.

```
SELECT e.name, array_agg(p.name)
FROM employee e
    JOIN project p ON p.emp_id = e.emp_id
GROUP BY e.emp_id, e.name;
```

```
  name   |      array_agg
---------+--------------------
 Jones   | {Enviro2,Nuclear1}
 Smith   | {Enviro1}
 Turner  | {Nuclear1}
 Blake   | {Enviro1}
```

Rekurzia: cez CTE (Common Table Expressions)

WITH RECURSIVE umožňuje rekurzívne sa odvolať na reláciu.

```
WITH RECURSIVE cte_name AS(
    SELECT ...  -- non-recursive term
  UNION [ALL]
    SELECT ...  -- recursive term
)
SELECT * FROM cte_name;
```

Rekurzia

```
WITH RECURSIVE t(n) AS (
    SELECT 1
  UNION ALL
    SELECT n+1 FROM t WHERE n < 5
)
SELECT n FROM t;
```

Výpočet rekurzie — iterácia (seminaive evaluation)

1. Evaluate the non-recursive term. For UNION (but not UNION ALL), discard duplicate rows. Include all remaining rows in the result of the recursive query, and also place them in a temporary working table.
2. Evaluate the recursive term, substituting the current contents of the working table for the recursive self-reference. For UNION, discard duplicate rows and rows that duplicate any previous result row. Include all remaining rows in the result of the recursive query, and also place them in a temporary intermediate table.
3. If the intermediate table is empty, stop. Otherwise replace the working table with the intermediate table and go to 2.

Rekurzia

Rekurzívny odkaz možno použiť len raz a pre jednu reláciu (PostgreSQL). Nekorektné využitie:

```
WITH RECURSIVE t(n) AS (
    SELECT 1
  UNION ALL
    (SELECT n+1 FROM t WHERE n < 5
        UNION ALL
     SELECT n-1 FROM t WHERE n < 5)
)
SELECT n FROM t;
```

Rekurzia

Rekurzívny výpočet nemusí skončiť, zodpovednosť je na autorovi dotazu.

```
WITH RECURSIVE t(n) AS (
    SELECT 1
  UNION ALL
    SELECT n+1 FROM t
)
SELECT n FROM t WHERE n < 5;
```

Rekurzia

Rekurzia zvyšuje vyjadrovaciu silu jazyka SQL.

- Rekurziou sa dá vyjadriť tranzitívny uzáver, t. j. existencia cesty neobmedzenej dĺžky.
- Bez rekurzie možno vyjadriť len cestu fixnej dĺžky (pomocou opakovaného joinu).

Prehľadávanie grafov

Pomocou rekurzie možno zapísať prehľadávanie stromu.

```
WITH RECURSIVE search_tree(vFrom, vTo) AS (
    SELECT t.vFrom, t.vTo
    FROM tree t
  UNION ALL
    SELECT st.vFrom, t.vTo
    FROM search_tree st, tree t
    WHERE t.vFrom = st.vTo
)
SELECT * FROM search_tree;
```

Prehľadávanie grafov

Poradie navštívených vrcholov závisí od implementácie rekurzie, ale vhodným usporiadaním vieme dosiahnuť prehľadávanie do šírky.

```
WITH RECURSIVE search_tree(vFrom, vTo, depth) AS (
    SELECT t.vFrom, t.vTo, 1
    FROM tree t
  UNION ALL
    SELECT st.vFrom, t.vTo, depth + 1
    FROM search_tree st, tree t
    WHERE t.vFrom = st.vTo
)
SELECT * FROM search_tree ORDER BY depth;
```

Prehľadávanie grafov

Poradie navštívených vrcholov závisí od implementácie rekurzie, ale vhodným usporiadaním vieme dosiahnuť prehľadávanie do hĺbky.

```
WITH RECURSIVE search_tree(vFrom, vTo, path) AS (
    SELECT t.vFrom, t.vTo, ARRAY[t.vFrom, t.vTo]
    FROM tree t
  UNION ALL
    SELECT st.vFrom, t.vTo, path || t.vTo
    FROM search_tree st, tree t
    WHERE t.vFrom = st.vTo
)
SELECT * FROM search_tree ORDER BY path;
```

Prehľadávanie grafov

Ak chceme ustrážiť koniec rekurzie, treba sa vyhýbať zacykleniu.

```
WITH RECURSIVE
    search_graph(vFrom, vTo, depth, path, is_cycle)
AS (
    SELECT g.vFrom, g.vTo, 1,
        ARRAY[g.vFrom, g.vTo], false
    FROM graph g
  UNION ALL
    SELECT sg.vFrom, g.vTo, sg.depth + 1,
        path || g.vTo, g.vTo = ANY(path)
    FROM search_graph sg, graph g
    WHERE g.vFrom = sg.vTo AND NOT is_cycle
)
SELECT * FROM search_graph ORDER BY is_cycle, path;
```

Prehľadávanie grafov

SQL po pridaní rekurzie je turingovsky úplný jazyk (vyjadríte v ňom riešenie pre všetky veci programovateľné v Pythone či C). Náčrt dôkazu a ukážka riešenia zložitých problémov v SQL, napr. ASCII Mandelbrotovej množiny: link

Prehľadávanie grafov

SQL po pridaní rekurzie je turingovsky úplný jazyk (vyjadríte v ňom riešenie pre všetky veci programovateľné v Pythone či C). Náčrt dôkazu a ukážka riešenia zložitých problémov v SQL, napr. ASCII Mandelbrotovej množiny: link

Neznamená to však, že túto silu musíte aj využiť. Napr. pri prehľadávaní grafu treba zvážiť, či nie je jednoduchšie vypýtať si celý graf z databázy a prehľadávanie zapísať ako program vo vhodnejšom jazyku. Podobne pri iných komplikovaných problémoch (napr. riešených backtrackingom).

Hierarchické dáta sa však v databázach občas vyskytujú, treba preto mať aspoň predstavu, ako sa s nimi vysporiadať.

Literatúra

- https://www.sqltutorial.org/sql-window-functions/
- https://www.sqlite.org/windowfunctions.html
- https://www.postgresql.org/docs/current/tutorial-window.html
- https://www.postgresql.org/docs/current/arrays.html
- https://www.postgresql.org/docs/current/queries-with.html
- https://www.sqlite.org/lang_with.html

Na zamyslenie

- Dá sa každá rekurzia zapísať pomocou iterácie?
  A čo iterácia pomocou rekurzie?
- Vysvetlite rozdiel medzi naivnou a seminaivnou evaluáciou rekurzívneho dotazu.
- Vymyslite otázku, ktorá sa nedá vyjadriť v SQL ani po pridaní rekurzie. (Bežne sa pýtame na záznamy, to sa vyjadriť dá, ale skúste sa pýtať na niečo iné.)

Úlohy

Čo počíta tento dotaz?

```
WITH RECURSIVE t(s) AS (
    SELECT 'foo'
  UNION ALL
    SELECT s || 'bar' FROM t WHERE length(s) < 20
)
SELECT s FROM t ORDER BY s;
```

Úlohy

Čo počíta tento dotaz?

```
WITH RECURSIVE t(s) AS (
    SELECT 'foo'
  UNION ALL
    SELECT s || 'bar' FROM t WHERE length(s) < 20
)
SELECT s FROM t ORDER BY s;
```

Generuje reťazce v tvare `foo(bar)`* postupne od najkratších.

Úlohy

Súčet čísel od 1 do 100

Úlohy

Súčet čísel od 1 do 100

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

Úlohy

Čo počíta tento dotaz?
Zmení sa to, ak UNION nahradíme UNION ALL?

```
WITH RECURSIVE t(n) AS (
    SELECT 1
  UNION
    SELECT 10-n FROM t
)
SELECT * FROM t ORDER BY n;
```

Úlohy

Skončí výpočet tohto dotazu?

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t
)
SELECT * FROM t LIMIT 10;
```

Úlohy

Skončí výpočet tohto dotazu?

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t
)
SELECT * FROM t LIMIT 10;
```

V PostgreSQL áno, ale vo všeobecnosti to závisí od implementácie LIMIT a rekurzie.