Как CTE превращают SQL в настоящий ЯП...

20 сентября 2022 г

PostgresPro

PGMeetup.NN

Иван Панченко

PostgresPro

CTE Common Table Expressions

- Появились в SQL:1999
- Есть практически во всех СУБД
- Могут быть рекурсивными
- Иногда похожи на VIEW, иногда на подзапросы, иногда на временную таблицу
- Имеют много применений

```
WITH [ RECURSIVE ]
t1 [(field_name1, …)] AS [[NOT] MATERIALIZED] (query1),
t2                    AS [[NOT] MATERIALIZED] (query2), …
SELECT|UPDATE|DELETE...
```

PostgresPro

CTE Материализация CTE

- [NOT] MATERIALIZED
- До v12 материализовалось всё
- Теперь:
- Пишущие и рекурсивные — всегда материальны
- Неиспользуемые не исполняются
- Материализованные CTE - «барьер» для оптимизатора.

```
WITH yy   AS (SELECT * FROM cte WHERE y>1),
   noexec AS (SELECT * FROM cte),
   always_exec AS (INSERT INTO cte VALUES (1,2,3) RETURNING *)
SELECT * FROM yy WHERE x = 2;
```

ПОЛЬЗА ОТ CTE

- Делать запрос более читаемым (выделяя этапы исполнения запроса)
- Влиять на план запроса и ускорять его
- Рекурсивные запросы (алгоритмы на деревьях, графах)
- Избегать повторения подзапросов в запросе
- Избегать динамического формирования SQL
- Объединять несколько DML в один, анализировать результат выполнения DML.

PostgresPro

ПОЕХАЛИ. ПРОСТОЙ ПРИМЕР.

```
WITH graduates  AS (
     SELECT id, name, grade
     FROM students
     WHERE status = 'graduate'
)
SELECT count(*), AVG(grade)
FROM graduates;
```

Пока непонятно, зачем...
станет понятно по мере усложнения

PostgresPro

НЕКОТОРАЯ ПОЛЬЗА: СТРУКТУРИРУЕМ SQL

```
WITH graduates  AS (
     SELECT id, name, grade
     FROM students
     WHERE status = 'graduate'
)
SELECT name, grade,
    (SELECT count(*)
       FROM graduates b WHERE b.grade < a.grade
    )   -- сколько учатся хуже, чем этот?
FROM graduates a;
```

Тут подзапрос фигурирует 2 раза. Уже экономия.
Какая именно? Зависит от **materialize**.

PostgresPro

ПРОСТАЯ РЕКУРСИЯ

```
WITH RECURSIVE seq  AS (
     SELECT 1 AS n                    -- старт рекурсии
     UNION ALL
     SELECT n+1 AS n FROM seq  -- шаг рекурсии
     WHERE n < 10
)
SELECT SUM(n*n) FROM seq;
```

На практике так вряд ли кто-то будет делать. Хотя...

Disclaimer: я показываю, как можно, а не как нужно делать.

PostgresPro

ТИПОВОЙ ОБХОД ДЕРЕВА

```
CREATE TABLE section (
    id      int PRIMARY KEY,
    parent int,
    pos     int,
    name    text
);
CREATE TABLE book (
    name text,
    sections int[]
);
```

Структура таблицы, в которой хранится дерево.

PostgresPro

ДАННЫЕ ДЛЯ ПРИМЕРА

id	parent	pos	name
1		1	История
2		2	Физика
3	1	2	История средних веков
4	1	1	История древнего мира
5	4	1	История древнего Китая
6	4	2	История древней Персии
7	2	2	Физика твердого тела
8	2	2	Гидродинамика

name	sections
Ударные волны в насыщенных парах	{8}
Сыма Цянь. Исторические записки	{5}
Турбулентность и фракталы	{8}
Физика в древнем Иране	{8,6}
Сверхпроводимость двухмерных кристаллов	{7}

PostgresPro

ТИПОВОЙ ОБХОД ДЕРЕВА

```
WITH RECURSIVE tree
     (id,parent, name, level, id_path, pos_path)  AS (
  SELECT id, parent, name, 0, ARRAY[id], ARRAY[pos]
    FROM section WHERE parent IS NULL

  UNION ALL

  SELECT s.id, s.parent, s.name, tree.level+1,
    tree.id_path  || ARRAY[s.id],
    tree.pos_path || ARRAY[s.pos]
   FROM section s, tree WHERE s.parent = tree.id
)
SELECT * FROM tree ORDER BY pos_path;
```

PostgresPro

РЕЗУЛЬТАТ

```
 id | parent |           name           | level | id_path | pos_path
----+--------+--------------------------+-------+---------+----------
  1 |        | История                  |     0 | {1}     | {1}
  4 |      1 | История древнего мира    |     1 | {1,4}   | {1,1}
  5 |      4 | История древнего Китая   |     2 | {1,4,5} | {1,1,1}
  6 |      4 | История древней Персии   |     2 | {1,4,6} | {1,1,2}
  3 |      1 | История средних веков    |     1 | {1,3}   | {1,2}
  2 |        | Физика                   |     0 | {2}     | {2}
  8 |      2 | Гидродинамика            |     1 | {2,8}   | {2,2}
  7 |      2 | Физика твердого тела     |     1 | {2,7}   | {2,2}
(8 rows)
```

ДОБАВИМ ПОДРАЗДЕЛЫ ВСЕХ УРОВНЕЙ

```
WITH RECURSIVE tree …,
  node_link AS (     --  связь «потомок-предок»
    SELECT id          AS descendent_id,
     unnest(id_path) AS ancestor_id
    FROM tree
  ),
  node_descendants AS ( --  связь «предок-[потомки]»
    SELECT ancestor_id,
      array_agg (descendent_id) AS descendants
    FROM tree GROUP BY ancestor_id
  )
SELECT tree.*, node_descendants.descendants
  FROM tree JOIN node_descendants ON ancestor_id = id
ORDER BY pos_path;
```

PostgresPro

РЕЗУЛЬТАТ

id	parent	name	level	id_path	pos_path	**descendants**
1		История	0	{1}	{1}	**{1,3,4,5,6}**
4	1	История древнего мира	1	{1,4}	{1,1}	**{4,5,6}**
5	4	История древнего Китая	2	{1,4,5}	{1,1,1}	**{5}**
6	4	История древней Персии	2	{1,4,6}	{1,1,2}	**{6}**
3	1	История средних веков	1	{1,3}	{1,2}	**{3}**
2		Физика	0	{2}	{2}	**{2,8,7}**
8	2	Гидродинамика	1	{2,8}	{2,2}	**{8}**
7	2	Физика твердого тела	1	{2,7}	{2,2}	**{7}**

(8 rows)

PostgresPro

ПОСЧИТАЕМ, СКОЛЬКО КНИГ В КАЖДОМ РАЗДЕЛЕ

```
WITH RECURSIVE
   tree …,
   node_link …,
   node_descendants …,
SELECT tree.*,
   (SELECT count(*)
    FROM book
    WHERE sections && node_descendants.descendants
   )
   FROM tree JOIN node_descendants ON ancestor_id = id
   ORDER BY pos_path;
```

PostgresPro

РЕЗУЛЬТАТ

id	parent	name	level	id_path	pos_path	**count**
1		История	0	{1}	{1}	**2**
4	1	История древнего мира	1	{1,4}	{1,1}	**2**
5	4	История древнего Китая	2	{1,4,5}	{1,1,1}	**1**
6	4	История древней Персии	2	{1,4,6}	{1,1,2}	**1**
3	1	История средних веков	1	{1,3}	{1,2}	**0**
2		Физика	0	{2}	{2}	**4**
8	2	Гидродинамика	1	{2,8}	{2,2}	**3**
7	2	Физика твердого тела	1	{2,7}	{2,2}	**1**

(8 rows)

PostgresPro

ВАЖНО

1) Последовательное конструирование выборки
2) Фактически, алгоритмическое программирование
3) Можно и отлаживать последовательно
4) И не забывайте писать комментарии

PostgresPro

ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОГО РЕШЕНИЯ

1) Выдать поддерево разделов, в которых (или подразделах которых) есть книги на букву «А».
2) Найти книги, которые отнесены одновременно к разделу и его предку.
3) Отцепить такие книги от таких предков.
4) Найти междисциплинарные книги (т.е. относящиеся к разным разделам верхнего уровня или их подразделам)

Конкурс!

Первый, приславший правильные решения этих задач, получает билет на PGConf.Russia 2023
Решения присылать на info@postgrespro.ru до 01.01.2023

PostgresPro

ПРОЩЕ КОНСТРУИРОВАТЬ ЗАПРОСЫ

```
WITH args (section, name, time) AS (
    SELECT $1, $2,
      $3 + '100minutes' -- тут и предвычисления можно делать
)
SELECT (сложный запрос, в котором параметры
 упоминаются много раз)
```

PostgresPro

ЕЩЁ ЗАДАЧКА

```
CREATE TABLE task (
      id         int PRIMARY KEY,
      prio       int NOT NULL, -- от 1 до 5
      is_active bool NOT NULL
);
CREATE TABLE action (
     task       int NOT NULL REFERENCES task(id),
     is_activation   bool,
     is_deactivation bool,
     time       timestamptz
);
```

Построить график количества активных задач по времени

PostgresPro

НАЧНЁМ...

```
WITH present AS (
  SELECT prio, count(*) AS n
  FROM task GROUP BY prio
),
changes AS (
  SELECT action.time::date AS day, prio,
    SUM(CASE WHEN is_activation THEN 1 ELSE -1 END) dn
  FROM action
  JOIN task ON action.task = task.id
  WHERE is_activation OR is_deactivation
  GROUP BY day, prio
)
```

PostgresPro

КАЗАЛОСЬ БЫ...

```
WITH …
SELECT prio, now()::date AS day, n FROM present
UNION ALL
SELECT changes.prio,
       changes.day – '1day'::interval AS day,
       present.n – SUM(changes.dn)
   OVER (PARTITION BY changes.prio ORDER BY changes.day
DESC)
FROM changes
JOIN present ON changes.prio = present.prio;
```

Но тут есть ошибка.

В таблице present какой-то из приоритетов может отсутствовать.

И тогда по нему график не будет построен.

PostgresPro

ПОЭТОМУ...

```
WITH …,
   prios AS (SELECT DISTINCT prio FROM task),
   present_full AS (
      SELECT prios.prio,
             COALESCE(present.n, 0) AS n
       FROM prios
      LEFT JOIN present ON prios.prio = present.prio
   )
SELECT … present_full …  ;
```

Так правильно.

PostgresPro

НАПОСЛЕДОК

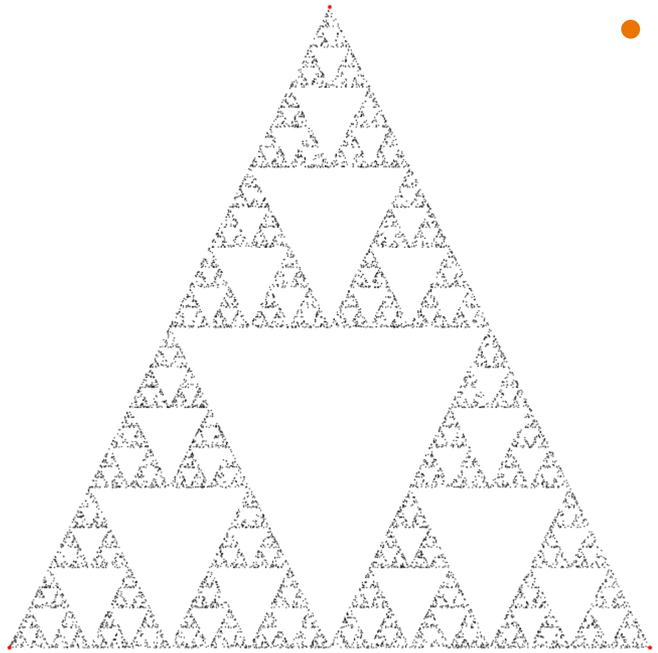
```
WITH updates AS (
  UPDATE task SET prio = 3 WHERE prio = 2;
  RETURNING id, created_at
)
SELECT COUNT(*), MAX(created_at) FROM updates;
```

А мог бы быть и DELETE.
Так можно анализировать выдачу UPDATE и DELETE.

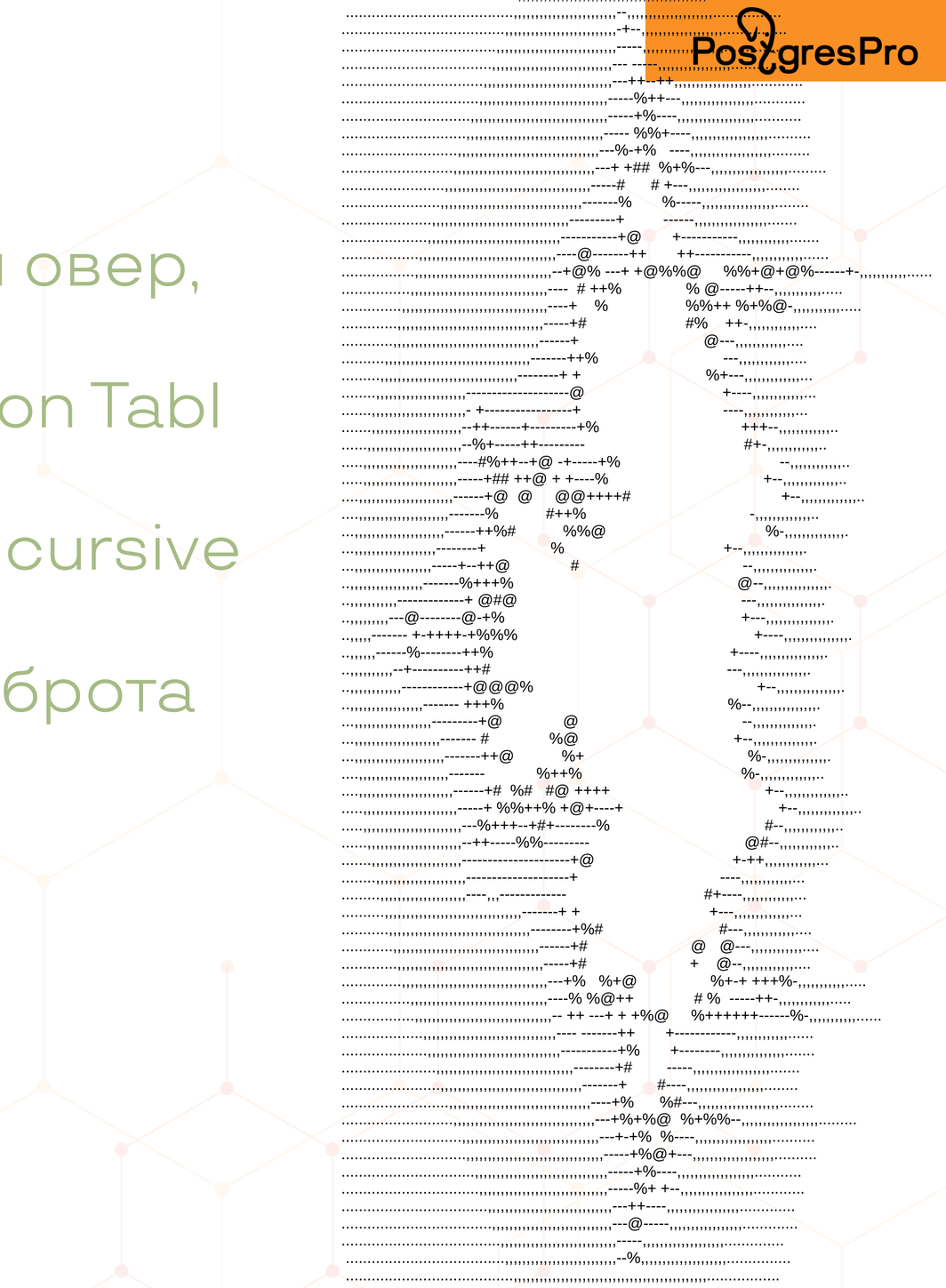
PostgresPro

ПОЧИТАТЬ

- Игра в прятки с оптимизатором. Гейм овер, это CTE PostgreSQL 12
- Programming the SQL Way with Common Table Expressions
- Drawing the Sierpiński Triangle With Recursive SQL and SVG



- Множество Мандельброта



PGMeetup.NN

СПАСИБО!

postgrespro.ru