Лабораторная работа №9

Тема: Глава 10 из книги. Упр 3, 6, 8

Группа: М8О-109СВ-24

Выполнил: **Гимазетдинов Дмитрий Русланович**

вернуться на главную

Упражнение 3

Дано:

Самостоятельно выполните команду EXPLAIN для запроса, содержащего общее табличное выражение (CTE). Посмотрите, на каком уровне находится узел плана, отвечающий за это выражение, как он оформляется. Учтите, что общие табличные выражения всегда материализуются, т. е. вычисляются однократно и результат их вычисления сохраняется в памяти, а затем все последующие обращения в рамках запроса направляются уже к этому материализованному результату.

Решение:

Создадим общее табличное выражение:

```
demo=#
EXPLAIN WITH wt_air AS (
    SELECT * FROM aircrafts_tmp
    WHERE aircraft_code IN ('773', '763', 'CN1', 'CR2')
)
SELECT COUNT(*) FROM wt_air, wt_air AS w2 WHERE wt_air.aircraft_code =
w2.aircraft_code;
```

```
                                  QUERY PLAN
------------------------------------------------------------------------------
--
 Aggregate  (cost=26.67..26.68 rows=1 width=8)
   CTE wt_air
     ->  Seq Scan on aircrafts_tmp  (cost=0.00..25.30 rows=20 width=52)
           Filter: (aircraft_code = ANY ('{773,763,CN1,CR2}'::bpchar[]))
   ->  Hash Join  (cost=0.65..1.33 rows=20 width=0)
         Hash Cond: (wt_air.aircraft_code = w2.aircraft_code)
         ->  CTE Scan on wt_air  (cost=0.00..0.40 rows=20 width=16)
         ->  Hash  (cost=0.40..0.40 rows=20 width=16)
               ->  CTE Scan on wt_air w2  (cost=0.00..0.40 rows=20
width=16)
(9 rows)
```

Анализ плана выполнения запроса с общим табличным выражением (CTE)

Давайте подробно рассмотрим предоставленный вами план выполнения запроса и убедимся, что общее табличное выражение (CTE) действительно материализуется, как ожидается.

Ваш запрос:

```
EXPLAIN WITH wt_air AS (
    SELECT * FROM aircrafts_tmp
    WHERE aircraft_code IN ('773', '763', 'CN1', 'CR2')
)
SELECT COUNT(*)
FROM wt_air, wt_air AS w2
WHERE wt_air.aircraft_code = w2.aircraft_code;
```

Ваш план выполнения:

```
                                    QUERY PLAN
    ------------------------------------------------------------------------
------
     Aggregate  (cost=26.67..26.68 rows=1 width=8)
       CTE wt_air
         ->  Seq Scan on aircrafts_tmp  (cost=0.00..25.30 rows=20 width=52)
               Filter: (aircraft_code = ANY
('{773,763,CN1,CR2}'::bpchar[]))
       ->  Hash Join  (cost=0.65..1.33 rows=20 width=0)
             Hash Cond: (wt_air.aircraft_code = w2.aircraft_code)
             ->  CTE Scan on wt_air  (cost=0.00..0.40 rows=20 width=16)
             ->  Hash  (cost=0.40..0.40 rows=20 width=16)
                   ->  CTE Scan on wt_air w2  (cost=0.00..0.40 rows=20
width=16)
    (9 rows)
```

Разбор плана выполнения

1. **CTE wt_air**:
    - **Seq Scan on aircrafts_tmp**: Здесь выполняется последовательное сканирование таблицы `aircrafts_tmp` с фильтрацией по `aircraft_code`. Это соответствует определению CTE `wt_air`.
    - **Материализация**: Узел `CTE wt_air` указывает на то, что результат CTE был материализован. Это означает, что PostgreSQL выполнил запрос внутри CTE один раз и сохранил результат для дальнейшего использования.
2. **Hash Join**:

- **Hash Cond**: Условие соединения (`wt_air.aircraft_code = w2.aircraft_code`) используется для выполнения хэш-соединения между двумя экземплярами CTE.
- **CTE Scan on wt_air** и **CTE Scan on wt_air w2**: Эти узлы указывают на повторное использование материализованного результата CTE `wt_air` в основном запросе. Вместо повторного выполнения самого CTE, PostgreSQL обращается к уже сохранённому результату.

3. **Aggregate**:
   - **Aggregate** выполняет итоговую операцию `COUNT(*)` над результатом соединения.

Подтверждение материализации CTE

В представленном плане выполнения видно, что:

- **CTE wt_air** выполняется единожды с использованием `Seq Scan` и фильтрацией.
- **CTE Scan on wt_air** и **CTE Scan on wt_air w2** ссылаются на ранее материализованный результат CTE, избегая повторного выполнения исходного запроса CTE.

Это подтверждает, что PostgreSQL действительно материализует CTE, выполняя его один раз и используя сохранённый результат для всех последующих обращений в рамках того же запроса.

Упражнение 6

Дано:

Выполните команду EXPLAIN для запроса, в котором использована какая-нибудь из оконных функций. Найдите в плане выполнения запроса узел с именем WindowAgg. Попробуйте объяснить, почему он занимает именно этот уровень в плане.

Решение:

```
EXPLAIN
SELECT
    aircraft_code,
    model,
    range,
    ROW_NUMBER() OVER (ORDER BY range) AS row_num
FROM
    bookings.aircrafts_tmp;
```

Теоретический план запроса:

```
                                QUERY PLAN
-------------------------------------------------------------------------
---
 WindowAgg  (cost=71.17..89.02 rows=1020 width=60)
   ->  Sort  (cost=71.17..73.72 rows=1020 width=52)
```

```
         Sort Key: range
         ->  Seq Scan on aircrafts_tmp  (cost=0.00..20.20 rows=1020
width=52)
(4 rows)
```

Давайте посмотрим на выполнение команды `EXPLAIN` для запроса, использующего оконные функции. В нашем случае, предположим, что у нас есть таблица `bookings.aircrafts_tmp` с данными об авиасуднах, и мы хотим использовать оконную функцию для, например, вычисления **ранга** по дальности полёта (`range`) для каждого авиасудна.

Пример запроса с использованием оконной функции `ROW_NUMBER`:

```
EXPLAIN
SELECT aircraft_code, model, range,
       ROW_NUMBER() OVER (ORDER BY range DESC) AS range_rank
FROM bookings.aircrafts_tmp;
```

Этот запрос вычисляет порядковый номер (ранг) для каждого авиасудна, сортируя их по дальности полёта (чем выше дальность, тем выше ранг).

Пример плана выполнения

В результате выполнения команды `EXPLAIN` вы получите что-то вроде:

```
                                    QUERY PLAN
-------------------------------------------------------------------------------
-----------
 WindowAgg  (cost=XX.XX..YY.YY rows=N width=Z)
   ->  Sort  (cost=AA.AA..BB.BB rows=N width=Z)
         Sort Key: aircrafts_tmp.range DESC
         ->  Seq Scan on aircrafts_tmp  (cost=CC.CC..DD.DD rows=N width=Z)
(4 rows)
```

Разбор плана выполнения

1. **WindowAgg**:
    - Узел `WindowAgg` выполняет вычисления оконной функции. В данном случае это функция `ROW_NUMBER() OVER (ORDER BY range DESC)`, которая присваивает каждому ряду его позицию в сортированном наборе.
    - Оконные функции требуют, чтобы данные были отсортированы в соответствии с определённым критерием (`ORDER BY range DESC` в данном случае), поэтому узел `WindowAgg` находится **выше** операции сортировки (`Sort`) в плане.
2. **Sort**:

- Перед выполнением оконной функции PostgreSQL сортирует строки по столбцу `range` в порядке убывания, как указано в запросе.
- Узел `Sort` выполняет сортировку данных, после чего они передаются на обработку в узел `WindowAgg`.

3. **Seq Scan on aircrafts_tmp**:

   - Этот узел отвечает за последовательное сканирование таблицы `aircrafts_tmp`. PostgreSQL считывает строки таблицы, после чего передаёт их на сортировку в узел `Sort`.

Почему узел `WindowAgg` находится на этом уровне?

- **Оконные функции зависят от сортированных данных**, поскольку они выполняют вычисления над определённым набором строк, который часто определяется с помощью выражения `OVER (ORDER BY ...)`.
- В данном плане:
  - **Сначала** PostgreSQL считывает данные из таблицы с помощью узла `Seq Scan`.
  - **Затем** узел `Sort` сортирует строки по дальности полёта (`range`).
  - **Наконец** узел `WindowAgg` выполняет оконную функцию, вычисляя ранги на основе отсортированных данных.

Оконные функции выполняются **после сортировки** данных, так как они требуют отсортированного набора строк для корректного выполнения операций. Именно поэтому узел `WindowAgg` находится выше узла `Sort`.

Упражнение 8

Дано:

Замена коррелированного подзапроса соединением таблиц является одним из способов повышения производительности. Предположим, что мы задались вопросом: сколько маршрутов обслуживают самолеты каждого типа? При этом нужно учитывать, что может иметь место такая ситуация, когда самолеты какого-либо типа не обслуживают ни одного маршрута. Поэтому необходимо использовать не только представление «Маршруты» (routes), но и таблицу «Самолеты» (aircrafts).

Это первый вариант запроса, в нем используется коррелированный подзапрос.

```
EXPLAIN ANALYZE
SELECT  a.aircraft_code AS a_code,
        a.model,
        ( SELECT count( r.aircraft_code )
            FROM routes r
           WHERE r.aircraft_code = a.aircraft_code
        ) AS num_routes
FROM aircrafts a
GROUP BY 1, 2
ORDER BY 3 DESC;
```

А в этом варианте коррелированный подзапрос раскрыт и заменен внешним соединением:

```
EXPLAIN ANALYZE
SELECT  a.aircraft_code AS a_code,
        a.model,
        count( r.aircraft_code ) AS num_routes
  FROM aircrafts a
  LEFT
 OUTER JOIN routes r
    ON r.aircraft_code = a.aircraft_code
 GROUP BY 1, 2
 ORDER BY 3 DESC;
```

Причина использования внешнего соединения в том, что может найтись модель самолета, не обслуживающая ни одного маршрута, и если не использовать внешнее соединение, она вообще не попадет в результирующую выборку.

Исследуйте планы выполнения обоих запросов. Попытайтесь найти объяснение различиям в эффективности их выполнения. Чтобы получить усредненную картину, выполните каждый запрос несколько раз. Поскольку таблицы, участвующие в запросах, небольшие, то различие по абсолютным затратам времени выполнения будет незначительным. Но если бы число строк в таблицах было большим, то экономия ресурсов сервера могла оказаться заметной.

Предложите аналогичную пару запросов к базе данных «Авиаперевозки». Проведите необходимые эксперименты с вашими запросами.

Решение:

Выполним два запроса и получим определенные результаты:

Первый

```
QUERY PLAN
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
-------------------------------------------
 Sort  (cost=23668.18..23668.20 rows=9 width=56) (actual
time=195.800..195.803 rows=9 loops=1)
   Sort Key: ((SubPlan 1)) DESC
   Sort Method: quicksort  Memory: 25kB
   ->  Group  (cost=1.75..23668.03 rows=9 width=56) (actual
time=24.141..195.785 rows=9 loops=1)
         Group Key: ml.aircraft_code, ((ml.model ->> lang()))
         ->  Incremental Sort  (cost=1.75..14.95 rows=9 width=48) (actual
time=0.133..0.136 rows=9 loops=1)
               Sort Key: ml.aircraft_code, ((ml.model ->> lang()))
               Presorted Key: ml.aircraft_code
               Full-sort Groups: 1  Sort Method: quicksort  Average Memory:
```

```
25kB  Peak Memory: 25kB
               ->  Index Scan using aircrafts_pkey on aircrafts_data ml
(cost=0.14..14.54 rows=9 width=48) (actual time=0.062..0.108 rows=9
loops=1)
         SubPlan 1
           ->  Aggregate  (cost=2627.85..2627.86 rows=1 width=8) (actual
time=21.735..21.735 rows=1 loops=9)
                 ->  Hash Join  (cost=2239.20..2625.50 rows=188 width=240)
(actual time=20.227..21.729 rows=79 loops=9)
                       Hash Cond: (flights.arrival_airport =
ml_2.airport_code)
                       ->  Hash Join  (cost=2233.86..2619.19 rows=361
width=8) (actual time=20.214..21.699 rows=79 loops=9)
                             Hash Cond: (flights.departure_airport =
ml_1.airport_code)
                             ->  GroupAggregate  (cost=2228.52..2605.05
rows=695 width=67) (actual time=20.203..21.672 rows=79 loops=9)
                                   Group Key: flights.flight_no,
flights.departure_airport, flights.arrival_airport, flights.aircraft_code,
((flights.scheduled_arrival - flights.scheduled_departure))
                                   ->  Group  (cost=2228.52..2441.68
rows=6952 width=39) (actual time=20.187..21.589 rows=422 loops=9)
                                         Group Key: flights.flight_no,
flights.departure_airport, flights.arrival_airport, flights.aircraft_code,
((flights.scheduled_arrival - flights.scheduled_departure)),
((to_char(flights.scheduled_departure, 'ID'::text))::integer)
                                         ->  Sort  (cost=2228.52..2249.04
rows=8208 width=39) (actual time=20.155..20.416 rows=7296 loops=9)
                                               Sort Key: flights.flight_no,
flights.departure_airport, flights.arrival_airport,
((flights.scheduled_arrival - flights.scheduled_departure)),
((to_char(flights.scheduled_departure, 'ID'::text))::integer)
                                               Sort Method: quicksort
Memory: 2218kB
                                               ->  Seq Scan on flights
(cost=0.00..1694.88 rows=8208 width=39) (actual time=1.455..10.761
rows=7296 loops=9)
                                                     Filter: (aircraft_code
= ml.aircraft_code)
                                                     Rows Removed by
Filter: 58368
                             ->  Hash  (cost=4.04..4.04 rows=104 width=4)
(actual time=0.078..0.079 rows=104 loops=1)
                                   Buckets: 1024  Batches: 1  Memory Usage:
12kB
                                   ->  Seq Scan on airports_data ml_1
(cost=0.00..4.04 rows=104 width=4) (actual time=0.005..0.036 rows=104
loops=1)
                       ->  Hash  (cost=4.04..4.04 rows=104 width=4) (actual
time=0.101..0.101 rows=104 loops=1)
                             Buckets: 1024  Batches: 1  Memory Usage: 12kB
                             ->  Seq Scan on airports_data ml_2
(cost=0.00..4.04 rows=104 width=4) (actual time=0.014..0.048 rows=104
loops=1)
```

```
 Planning Time: 1.116 ms
 Execution Time: 196.025 ms
(34 rows)
```

Второй

```
QUERY PLAN
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
-------------------------------------------------
 Sort  (cost=5258.58..5258.60 rows=9 width=56) (actual time=65.781..65.784
rows=9 loops=1)
   Sort Key: (count(flights.aircraft_code)) DESC
   Sort Method: quicksort  Memory: 25kB
   ->  GroupAggregate  (cost=5255.85..5258.43 rows=9 width=56) (actual
time=65.672..65.773 rows=9 loops=1)
         Group Key: ml.aircraft_code, ((ml.model ->> lang()))
         ->  Sort  (cost=5255.85..5255.91 rows=22 width=52) (actual
time=65.660..65.686 rows=711 loops=1)
               Sort Key: ml.aircraft_code, ((ml.model ->> lang()))
               Sort Method: quicksort  Memory: 80kB
               ->  Hash Right Join  (cost=4746.77..5255.36 rows=22
width=52) (actual time=63.236..65.471 rows=711 loops=1)
                     Hash Cond: (flights.aircraft_code = ml.aircraft_code)
                     ->  Hash Join  (cost=4745.57..5242.39 rows=490
width=240) (actual time=63.167..64.515 rows=710 loops=1)
                           Hash Cond: (flights.arrival_airport =
ml_2.airport_code)
                           ->  Hash Join  (cost=4740.23..5234.52 rows=943
width=8) (actual time=63.074..64.260 rows=710 loops=1)
                                 Hash Cond: (flights.departure_airport =
ml_1.airport_code)
                                 ->  GroupAggregate  (cost=4734.89..5206.19
rows=1813 width=67) (actual time=62.991..64.022 rows=710 loops=1)
                                       Group Key: flights.flight_no,
flights.departure_airport, flights.arrival_airport, flights.aircraft_code,
((flights.scheduled_arrival - flights.scheduled_departure))
                                       ->  Sort  (cost=4734.89..4780.21
rows=18127 width=39) (actual time=62.985..63.160 rows=3798 loops=1)
                                             Sort Key: flights.flight_no,
flights.departure_airport, flights.arrival_airport, flights.aircraft_code,
((flights.scheduled_arrival - flights.scheduled_departure)),
((to_char(flights.scheduled_departure, 'ID'::text))::integer)
                                             Sort Method: quicksort
Memory: 423kB
                                             ->  HashAggregate
(cost=3090.24..3452.78 rows=18127 width=39) (actual time=54.227..54.901
rows=3798 loops=1)
                                                   Group Key:
flights.flight_no, flights.departure_airport, flights.arrival_airport,
```

```
flights.aircraft_code, (flights.scheduled_arrival -
flights.scheduled_departure), (to_char(flights.scheduled_departure,
'ID'::text))::integer
                                                    Batches: 1  Memory
Usage: 1297kB
                                                    ->  Seq Scan on flights
(cost=0.00..2105.28 rows=65664 width=39) (actual time=0.021..28.068
rows=65664 loops=1)
                                  ->  Hash  (cost=4.04..4.04 rows=104
width=4) (actual time=0.076..0.077 rows=104 loops=1)
                                        Buckets: 1024  Batches: 1  Memory
Usage: 12kB
                                        ->  Seq Scan on airports_data ml_1
(cost=0.00..4.04 rows=104 width=4) (actual time=0.004..0.034 rows=104
loops=1)
                            ->  Hash  (cost=4.04..4.04 rows=104 width=4)
(actual time=0.086..0.086 rows=104 loops=1)
                                  Buckets: 1024  Batches: 1  Memory Usage:
12kB
                                  ->  Seq Scan on airports_data ml_2
(cost=0.00..4.04 rows=104 width=4) (actual time=0.008..0.043 rows=104
loops=1)
                      ->  Hash  (cost=1.09..1.09 rows=9 width=48) (actual
time=0.032..0.033 rows=9 loops=1)
                            Buckets: 1024  Batches: 1  Memory Usage: 9kB
                            ->  Seq Scan on aircrafts_data ml
(cost=0.00..1.09 rows=9 width=48) (actual time=0.018..0.023 rows=9 loops=1)
 Planning Time: 0.878 ms
 Execution Time: 66.098 ms
(34 rows)
```

Как видно время исполнения первого запроса $\approx 196 ms$, а для второо запроса $\approx 66 ms$.

По планам выполнения обоих запросов можно увидеть ключевые строки, которые демонстрируют, почему второй запрос более эффективен. Рассмотрим это на конкретных строках планов.

Первый запрос

1. **SubPlan 1 (подзапрос)**:

   ```
   SubPlan 1
   ->  Aggregate  (cost=2627.85..2627.86 rows=1 width=8) (actual
   time=21.735..21.735 rows=1 loops=9)
   ```

   - Здесь указано, что подзапрос выполняется **9 раз** (по одному разу для каждого самолёта). Время выполнения одного подзапроса — **21.735 мс**, что при 9 повторениях даёт значительное время выполнения.

2. **Seq Scan on flights (поиск маршрутов для каждого самолёта)**:

```
->  Seq Scan on flights  (cost=0.00..1694.88 rows=8208 width=39)
(actual time=1.455..10.761 rows=7296 loops=9)
```

- Полный последовательный скан таблицы `flights` выполняется **9 раз**, так как подзапрос выполняется для каждого самолёта. Время выполнения одного сканирования — **1.455..10.761 мс**, что суммируется при 9 повторениях.
- В этом заключается ключевая неэффективность — многократный проход по данным.

Второй запрос

1. **Hash Right Join (соединение aircrafts с routes)**:

```
->  Hash Right Join  (cost=4746.77..5255.36 rows=22 width=52) (actual
time=63.236..65.471 rows=711 loops=1)
```

- Соединение таблиц `aircrafts` и `routes` выполняется **один раз**, и результат передаётся дальше в агрегатную функцию. Это ключевое преимущество, так как здесь база данных сразу собирает все данные для всех самолётов, не повторяя операции.

2. **Seq Scan on flights (последовательный скан маршрутов)**:

```
->  Seq Scan on flights  (cost=0.00..2105.28 rows=65664 width=39)
(actual time=0.021..28.068 rows=65664 loops=1)
```

- В отличие от первого запроса, здесь последовательное сканирование таблицы `flights` выполняется **один раз** (вместо 9 раз), что значительно снижает количество операций и ускоряет выполнение запроса.

Аналогичные примеры:

Поиска кол-ва мест, для всех самолетов:

```
select
      count(arc.model),
      ss.fare_conditions
  from aircrafts arc

  left
  join seats ss
    on arc.aircraft_code = ss.aircraft_code
 where ss.fare_condition in ('Comfort', 'Economy')
 group by ss.fare_conditions;
```

План:

```
demo=#
EXPLAIN ANALYZE select
      count(arc.model),
      ss.fare_conditions
  from aircrafts arc

  left
  join seats ss
    on arc.aircraft_code = ss.aircraft_code
 where ss.fare_conditions in ('Comfort', 'Economy')
 group by ss.fare_conditions;
                                                                    QUERY PLAN
-------------------------------------------------------------------------------
-------------------------------------------------
 HashAggregate  (cost=336.18..336.21 rows=3 width=16) (actual
time=5.895..5.897 rows=2 loops=1)
   Group Key: ss.fare_conditions
   Batches: 1  Memory Usage: 24kB
   ->  Hash Join  (cost=1.20..30.52 rows=1187 width=40) (actual
time=0.073..1.516 rows=1188 loops=1)
         Hash Cond: (ss.aircraft_code = ml.aircraft_code)
         ->  Seq Scan on seats ss  (cost=0.00..24.74 rows=1187 width=12)
(actual time=0.036..0.728 rows=1188 loops=1)
               Filter: ((fare_conditions)::text = ANY
('{Comfort,Economy}'::text[]))
               Rows Removed by Filter: 151
         ->  Hash  (cost=1.09..1.09 rows=9 width=48) (actual
time=0.022..0.023 rows=9 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on aircrafts_data ml  (cost=0.00..1.09 rows=9
width=48) (actual time=0.008..0.012 rows=9 loops=1)
 Planning Time: 0.366 ms
 Execution Time: 5.963 ms
(13 rows)
```

Запрос 2:

```
with wt_tt as (SELECT
    (SELECT arc.model
     FROM aircrafts arc
     WHERE arc.aircraft_code = ss.aircraft_code) AS model_count,
    ss.fare_conditions
FROM seats ss
WHERE ss.fare_conditions IN ('Comfort', 'Economy'))
select count(wt_tt.model_count), wt_tt.fare_conditions from wt_tt
group by wt_tt.fare_conditions;
```

План:

```
                                                              QUERY PLAN
-----------------------------------------------------------------------------------
-------------------------------------------
 HashAggregate  (cost=1650.93..1650.96 rows=3 width=16) (actual
time=8.307..8.309 rows=2 loops=1)
   Group Key: ss.fare_conditions
   Batches: 1  Memory Usage: 24kB
   ->  Seq Scan on seats ss  (cost=0.00..24.74 rows=1187 width=12) (actual
time=0.036..0.482 rows=1188 loops=1)
         Filter: ((fare_conditions)::text = ANY
('{Comfort,Economy}'::text[]))
         Rows Removed by Filter: 151
   SubPlan 1
     ->  Seq Scan on aircrafts_data ml  (cost=0.00..1.36 rows=1 width=32)
(actual time=0.004..0.005 rows=1 loops=1188)
           Filter: (aircraft_code = ss.aircraft_code)
           Rows Removed by Filter: 8
 Planning Time: 0.354 ms
 Execution Time: 8.371 ms
(12 rows)
```
