Язык SQL, дз – 9

Бурмашев Григорий, БПМИ-208

19 декабря 2022 г.

Номер 3

3. **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.**

Посмотрим на запрос из одной из предыдущих домашних заданий, который содержит в себе CTE:

```
demo=# EXPLAIN WITH add_row AS
( INSERT INTO aircrafts_tmp
        SELECT * FROM aircrafts
        RETURNING *
)
INSERT INTO aircrafts_tmp
SELECT add_row.aircraft_code, add_row.model, add_row.range
FROM add_row;
```

Результат:

```
                                  QUERY PLAN
------------------------------------------------------------------------------
 Insert on aircrafts_tmp  (cost=3.36..3.54 rows=0 width=0)
   CTE add_row
     ->  Insert on aircrafts_tmp aircrafts_tmp_1  (cost=0.00..3.36 rows=9 width=52)
           ->  Seq Scan on aircrafts_data ml  (cost=0.00..3.36 rows=9 width=52)
   ->  CTE Scan on add_row  (cost=0.00..0.18 rows=9 width=52)
(5 rows)
```

Узел CTE находится на самом верху в плане, я ожидал, что он будет самым внутренним (приоритетным), поскольку CTE мы (кажется) должны создать в первую очередь.

Номер 6

6. 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.

Делаю команду и сразу смотрю на результат:

```
demo=# EXPLAIN
SELECT flight_id,
  row_number() OVER (PARTITION BY flight_no ORDER BY flight_id)
FROM flights f;
                                QUERY PLAN
---------------------------------------------------------------------------
 WindowAgg  (cost=3209.85..3872.27 rows=33121 width=19)
   ->  Sort  (cost=3209.85..3292.65 rows=33121 width=11)
         Sort Key: flight_no, flight_id
         ->  Seq Scan on flights f  (cost=0.00..723.21 rows=33121 width=11)
(4 rows)
```

Здесь WindowAdd идет самым первым, поскольку он вычисляет оконную функцию по тому (OVER) набору данных, который был отсортирован с помощью ORDER BY (Sort ...)

Номер 8

> 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.

Проверяю время выполнения первого запроса:

```
                                                                                                   QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=11769.37..11769.39 rows=9 width=56) (actual time=177.846..177.864 rows=9 loops=1)
   Sort Key: ((SubPlan 1)) DESC
   Sort Method: quicksort  Memory: 25kB
   ->  Group  (cost=1.75..11769.22 rows=9 width=56) (actual time=21.973..177.789 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=0.447..0.454 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=0.357..0.384 rows=9 loops=1)
         SubPlan 1
           ->  Aggregate  (cost=1305.76..1305.77 rows=1 width=8) (actual time=19.698..19.698 rows=1 loops=9)
                 ->  Hash Join  (cost=1106.81..1304.55 rows=97 width=240) (actual time=17.341..19.682 rows=79 loops=9)
                       Hash Cond: (flights.arrival_airport = ml_2.airport_code)
                       ->  Hash Join  (cost=1101.47..1298.71 rows=187 width=8) (actual time=17.323..19.620 rows=79 loops=9)
                             Hash Cond: (flights.departure_airport = ml_1.airport_code)
                             ->  GroupAggregate  (cost=1096.13..1288.81 rows=359 width=67) (actual time=17.311..19.570 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=1096.13..1204.47 rows=3589 width=39) (actual time=17.287..19.365 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(f
                                         ->  Sort  (cost=1096.13..1106.48 rows=4140 width=39) (actual time=17.274..17.792 rows=3680 loops=9)
                                               Sort Key: flights.flight_no, flights.departure_airport, flights.arrival_airport, ((flights.scheduled_arrival - flights.scheduled_departure)), ((to_char(flights.scheduled_d
                                               Sort Method: quicksort  Memory: 1049kB
                                               ->  Seq Scan on flights  (cost=0.00..847.41 rows=4140 width=39) (actual time=1.697..10.625 rows=3680 loops=9)
                                                     Filter: (aircraft_code = ml.aircraft_code)
                                                     Rows Removed by Filter: 29441
                             ->  Hash  (cost=4.04..4.04 rows=104 width=4) (actual time=0.065..0.066 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..0.025 rows=104 loops=1)
                       ->  Hash  (cost=4.04..4.04 rows=104 width=4) (actual time=0.129..0.130 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=0.019..0.055 rows=104 loops=1)
 Planning Time: 2.159 ms
 Execution Time: 180.089 ms
(34 rows)
```

Делаю еще пару запусков:

```
 Planning Time: 1.637 ms
 Execution Time: 103.627 ms
(34 rows)
```

```
 Planning Time: 0.589 ms
 Execution Time: 98.901 ms
(34 rows)
```

Второй запрос:


```
 Planning Time: 0.442 ms
 Execution Time: 42.541 ms
(34 rows)
```

```
 Planning Time: 0.487 ms
 Execution Time: 41.285 ms
(34 rows)
```

Видим, что для обоих запросов planning time **очень** сильно отличается (в меньшую сторону) от execution time. Planning time не учитывает в себе то время, что сервер в реальности затратит на обработку запроса и выдачу результата. При этом для второго запроса оба времени меньше, что ожидаемо, поскольку мы оптимизировали запрос путем замены коррелированного запроса на JOIN.

Теперь придумаем аналогичные запросы. Первый:

```
demo=# SELECT a.city AS a_code,
            a.airport_code,
            ( SELECT count( r.departure_airport )
FROM routes r
                WHERE r.departure_airport = a.airport_code
            ) AS num_departures
       FROM airports a
       GROUP BY 1, 2
       ORDER BY 3 DESC;
```

Время выполнения:

```
                                                                                                                                      QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
 Sort  (cost=88954.42..88954.68 rows=104 width=44) (actual time=404.627..404.640 rows=104 loops=1)
   Sort Key: ((SubPlan 1)) DESC
   Sort Method: quicksort  Memory: 33kB
   ->  HashAggregate  (cost=30.82..88950.94 rows=104 width=44) (actual time=4.954..404.563 rows=104 loops=1)
         Group Key: (ml.city ->> lang()), ml.airport_code
         Batches: 1  Memory Usage: 24kB
         ->  Seq Scan on airports_data ml  (cost=0.00..30.30 rows=104 width=36) (actual time=0.027..0.208 rows=104 loops=1)
         SubPlan 1
           ->  Aggregate  (cost=854.73..854.74 rows=1 width=8) (actual time=3.886..3.886 rows=1 loops=104)
                 ->  Nested Loop  (cost=827.75..853.43 rows=104 width=240) (actual time=3.788..3.884 rows=7 loops=104)
                       ->  Seq Scan on airports_data ml_1  (cost=0.00..4.30 rows=1 width=4) (actual time=0.005..0.011 rows=1 loops=104)
                             Filter: (airport_code = ml.airport_code)
                             Rows Removed by Filter: 103
                       ->  Hash Join  (cost=827.75..848.09 rows=104 width=4) (actual time=3.779..3.869 rows=7 loops=104)
                             Hash Cond: (flights.arrival_airport = ml_2.airport_code)
                             ->  GroupAggregate  (cost=822.41..840.21 rows=200 width=67) (actual time=3.778..3.865 rows=7 loops=104)
                                   Group Key: flights.flight_no, flights.departure_airport, flights.arrival_airport, flights.aircraft_code, ((flights.scheduled
rival - flights.scheduled_departure))
                                   ->  Group  (cost=822.41..831.12 rows=315 width=39) (actual time=3.763..3.855 rows=37 loops=104)
                                         Group Key: flights.flight_no, flights.departure_airport, flights.arrival_airport, flights.aircraft_code, ((flights.sch
led_arrival - flights.scheduled_departure)), ((to_char(flights.scheduled_departure, 'ID'::text))::integer)
                                         ->  Sort  (cost=822.41..823.20 rows=318 width=39) (actual time=3.760..3.781 rows=318 loops=104)
                                               Sort Key: flights.flight_no, flights.arrival_airport, flights.aircraft_code, ((flights.scheduled_arrival - fligh
scheduled_departure)), ((to_char(flights.scheduled_departure, 'ID'::text))::integer)
                                               Sort Method: quicksort  Memory: 53kB
                                               ->  Seq Scan on flights  (cost=0.00..809.19 rows=318 width=39) (actual time=1.795..3.434 rows=318 loops=104)
                                                     Filter: (departure_airport = ml.airport_code)
                                                     Rows Removed by Filter: 32803
                             ->  Hash  (cost=4.04..4.04 rows=104 width=4) (actual time=0.039..0.040 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=0.003..0.019 rows=104 loops=1)
 Planning Time: 0.545 ms
 Execution Time: 404.743 ms
(30 rows)
```

```
 Planning Time: 1.872 ms
 Execution Time: 366.775 ms
(30 rows)
```

Второй запрос:

```
demo=#  SELECT a.city AS a_code,
a.airport_code,
         count( r.departure_airport) AS num_departures
    FROM airports a
    LEFT OUTER JOIN routes r
      ON r.departure_airport = a.airport_code
    GROUP BY 1, 2
    ORDER BY 3 DESC;
```

Время выполнения:

```
                                                                                                                                                    QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=2807.19..2807.45 rows=104 width=44) (actual time=40.388..40.398 rows=104 loops=1)
   Sort Key: (count(flights.departure_airport)) DESC
   Sort Method: quicksort  Memory: 33kB
   ->  GroupAggregate  (cost=2774.96..2803.70 rows=104 width=44) (actual time=40.172..40.362 rows=104 loops=1)
         Group Key: ((ml.city ->> lang())), ml.airport_code
         ->  Sort  (cost=2774.96..2775.32 rows=144 width=40) (actual time=40.167..40.216 rows=710 loops=1)
               Sort Key: ((ml.city ->> lang())), ml.airport_code
               Sort Method: quicksort  Memory: 80kB
               ->  Hash Right Join  (cost=2450.39..2769.80 rows=144 width=40) (actual time=36.827..39.821 rows=710 loops=1)
                     Hash Cond: (flights.departure_airport = ml.airport_code)
                     ->  Hash Join  (cost=2445.05..2724.60 rows=276 width=240) (actual time=36.740..38.601 rows=710 loops=1)
                           Hash Cond: (flights.arrival_airport = ml_2.airport_code)
                           ->  Hash Join  (cost=2439.71..2717.84 rows=530 width=8) (actual time=36.692..38.335 rows=710 loops=1)
                                 Hash Cond: (flights.departure_airport = ml_1.airport_code)
                                 ->  GroupAggregate  (cost=2434.37..2699.57 rows=1020 width=67) (actual time=36.643..38.084 rows=710 loops=1)
                                       Group Key: flights.flight_no, flights.departure_airport, flights.arrival_airport, flights.aircraft_code, ((flights.sched
d_arrival - flights.scheduled_departure))
                                       ->  Sort  (cost=2434.37..2459.87 rows=10200 width=39) (actual time=36.638..36.925 rows=3798 loops=1)
                                             Sort Key: flights.flight_no, flights.departure_airport, flights.arrival_airport, flights.aircraft_code, ((flights.
eduled_arrival - flights.scheduled_departure)), ((to_char(flights.scheduled_departure, 'ID'::text))::integer)
                                             Sort Method: quicksort  Memory: 393kB
                                             ->  HashAggregate  (cost=1551.24..1755.24 rows=10200 width=39) (actual time=32.179..32.964 rows=3798 loops=1)
                                                   Group Key: flights.flight_no, flights.departure_airport, flights.arrival_airport, flights.aircraft_code, (fl
ts.scheduled_arrival - flights.scheduled_departure), (to_char(flights.scheduled_departure, 'ID'::text))::integer
                                                   Batches: 1  Memory Usage: 921kB
                                                   ->  Seq Scan on flights  (cost=0.00..1054.42 rows=33121 width=39) (actual time=0.010..16.789 rows=33121 loop
)
                                 ->  Hash  (cost=4.04..4.04 rows=104 width=4) (actual time=0.037..0.037 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=0.003..0.018 rows=104 loops=1)
                           ->  Hash  (cost=4.04..4.04 rows=104 width=4) (actual time=0.038..0.038 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=0.003..0.018 rows=104 loops=1)
                     ->  Hash  (cost=4.04..4.04 rows=104 width=53) (actual time=0.054..0.054 rows=104 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 17kB
                           ->  Seq Scan on airports_data ml  (cost=0.00..4.04 rows=104 width=53) (actual time=0.009..0.028 rows=104 loops=1)
 Planning Time: 0.499 ms
 Execution Time: 40.540 ms
(34 rows)
```

```
 Planning Time: 1.382 ms
 Execution Time: 47.225 ms
(34 rows)
```

```
 Planning Time: 0.503 ms
 Execution Time: 42.520 ms
(34 rows)
```

Получили аналогичный результат, время выполнения ведет себя точно также при оптимизации.