Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ № 8

Методы оптимизации
Тема

Преподаватель _____ Е. П. Моргунов
Подпись, дата Инициалы, Фамилия

Студент КИ19-17/1Б, №031939174 _____ А. К. Никитин
Номер группы, зачетной книжки Подпись, дата Инициалы, Фамилия

Красноярск 2021

1 Ход выполнения

1.1 Вопрос 1

Для того, чтобы найти в индексе первую строку в соответствии с требуемым порядком, нужно некоторое время.

1.2 Вопрос 3

```
demo=# EXPLAIN WITH some_seats AS (
demo(# SELECT * FROM seats
demo(# WHERE aircraft_code = '319'
demo(# )
demo-# SELECT * FROM some_seats
demo-# WHERE fare_conditions = 'Business';
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 CTE Scan on some_seats  (cost=10.43..13.04 rows=1 width=74)
   Filter: ((fare_conditions)::text = 'Business'::text)
   CTE some_seats
     ->  Index Scan using seats_aircraft_code_idx on seats  (cost=0.28..10.43 rows=116 width=15)
           Index Cond: (aircraft_code = '319'::bpchar)
(5 строк)
```

Рисунок 1 – Работа планировщика в случае общих табличных выражений

Как мы видим, планировщик тратит время только на поиск внутри табличного выражения, при этом по самому табличному выражению он использует CTE Scan.

1.3 Вопрос 5

Время 3.04 – это время, которое потребуется непосредственно на фильтрацию сгруппированной таблицы, на остальные операции потребуется время, поэтому первая оценка стоимости равна 3.82.

1.4 Вопрос 7

```
demo=# BEGIN;
BEGIN
demo=# EXPLAIN
demo-# INSERT INTO seats
demo-# VALUES
demo-# (320, '10G', 'Economy'),
demo-# (320, '10H', 'Business');
                              QUERY PLAN
-----------------------------------------------------------------------
 Insert on seats  (cost=0.00..0.03 rows=2 width=74)
   ->  Values Scan on "*VALUES*"  (cost=0.00..0.03 rows=2 width=74)
(2 строки)
```

Рисунок 2 – Вставка строк

```
demo=# BEGIN;
BEGIN
demo=# EXPLAIN
demo-# UPDATE aircrafts
demo-# SET range = range + 100
demo-# WHERE model ~ '^Аэро';
                              QUERY PLAN
-----------------------------------------------------------------------
 Update on aircrafts_data ml  (cost=0.00..3.39 rows=1 width=58)
   ->  Seq Scan on aircrafts_data ml  (cost=0.00..3.39 rows=1 width=58)
         Filter: ((model ->> lang()) ~ '^Аэро'::text)
(3 строки)
```

Рисунок 3 – Обновление строк таблицы

```
demo=# BEGIN;
BEGIN
demo=# EXPLAIN
demo-# DELETE FROM aircrafts
demo-# WHERE model ~ '^Боинг';
                              QUERY PLAN
-----------------------------------------------------------------------
 Delete on aircrafts_data ml  (cost=0.00..3.39 rows=1 width=6)
   ->  Seq Scan on aircrafts_data ml  (cost=0.00..3.39 rows=1 width=6)
         Filter: ((model ->> lang()) ~ '^Боинг'::text)
(3 строки)
```

Рисунок 4 – Удаление строк таблицы

1.5 Вопрос 9

```
demo=# EXPLAIN ANALYZE
demo-# SELECT * FROM routes;
                                                                                                    QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------
 Hash Join  (cost=2705.54..3008.86 rows=276 width=252) (actual time=126.502..142.045 rows=710 loops=1)
   Hash Cond: (f3.arrival_airport = ml_1.airport_code)
   CTE f3
     ->  GroupAggregate  (cost=2401.55..2694.86 rows=1020 width=67) (actual time=108.451..112.718 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=2401.55..2427.06 rows=10202 width=39) (actual time=108.438..109.332 rows=3798 loops=1)
                 Sort Key: flights.flight_no, flights.departure_airport, flights.arrival_airport, flights.aircraft_code, ((flights.scheduled_arrival - flights.scheduled_depa
rture)), ((to_char(flights.scheduled_departure, 'ID'::text))::integer)
                 Sort Method: quicksort  Memory: 332kB
                 ->  HashAggregate  (cost=1518.24..1722.28 rows=10202 width=39) (actual time=94.101..96.264 rows=3798 loops=1)
                       Group Key: flights.flight_no, flights.departure_airport, flights.arrival_airport, flights.aircraft_code, (flights.scheduled_arrival - flights.schedule
d_departure), (to_char(flights.scheduled_departure, 'ID'::text))::integer
                       ->  Seq Scan on flights  (cost=0.00..1021.42 rows=33121 width=39) (actual time=10.504..58.487 rows=33121 loops=1)
   ->  Hash Join  (cost=5.34..28.47 rows=530 width=234) (actual time=108.577..114.316 rows=710 loops=1)
         Hash Cond: (f3.departure_airport = ml.airport_code)
         ->  CTE Scan on f3  (cost=0.00..20.40 rows=1020 width=124) (actual time=108.454..113.547 rows=710 loops=1)
         ->  Hash  (cost=4.04..4.04 rows=104 width=114) (actual time=0.113..0.113 rows=104 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 18kB
               ->  Seq Scan on airports_data ml  (cost=0.00..4.04 rows=104 width=114) (actual time=0.012..0.053 rows=104 loops=1)
   ->  Hash  (cost=4.04..4.04 rows=104 width=114) (actual time=17.674..17.674 rows=104 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 18kB
         ->  Seq Scan on airports_data ml_1  (cost=0.00..4.04 rows=104 width=114) (actual time=16.973..17.603 rows=104 loops=1)
 Planning Time: 121.172 ms
 Execution Time: 143.285 ms
(22 строки)
```

Рисунок 5 – Анализ выборки из готового представления

```
                                                                                                    QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------
 Hash Join  (cost=2705.54..3008.86 rows=276 width=252) (actual time=87.809..104.541 rows=710 loops=1)
   Hash Cond: (f3.arrival_airport = ml_1.airport_code)
   CTE f3
     ->  GroupAggregate  (cost=2401.55..2694.86 rows=1020 width=67) (actual time=87.545..92.061 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=2401.55..2427.06 rows=10202 width=39) (actual time=87.531..88.454 rows=3798 loops=1)
                 Sort Key: flights.flight_no, flights.departure_airport, flights.arrival_airport, flights.aircraft_code, ((flights.scheduled_arrival - flights.scheduled_depa
rture)), ((to_char(flights.scheduled_departure, 'ID'::text))::integer)
                 Sort Method: quicksort  Memory: 332kB
                 ->  HashAggregate  (cost=1518.24..1722.28 rows=10202 width=39) (actual time=74.265..76.061 rows=3798 loops=1)
                       Group Key: flights.flight_no, flights.departure_airport, flights.arrival_airport, flights.aircraft_code, (flights.scheduled_arrival - flights.schedule
d_departure), (to_char(flights.scheduled_departure, 'ID'::text))::integer
                       ->  Seq Scan on flights  (cost=0.00..1021.42 rows=33121 width=39) (actual time=0.012..40.119 rows=33121 loops=1)
   ->  Hash Join  (cost=5.34..28.47 rows=530 width=234) (actual time=87.665..93.596 rows=710 loops=1)
         Hash Cond: (f3.departure_airport = ml.airport_code)
         ->  CTE Scan on f3  (cost=0.00..20.40 rows=1020 width=124) (actual time=87.549..92.810 rows=710 loops=1)
         ->  Hash  (cost=4.04..4.04 rows=104 width=114) (actual time=0.107..0.107 rows=104 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 18kB
               ->  Seq Scan on airports_data ml  (cost=0.00..4.04 rows=104 width=114) (actual time=0.010..0.051 rows=104 loops=1)
   ->  Hash  (cost=4.04..4.04 rows=104 width=114) (actual time=0.096..0.096 rows=104 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 18kB
         ->  Seq Scan on airports_data ml_1  (cost=0.00..4.04 rows=104 width=114) (actual time=0.008..0.047 rows=104 loops=1)
 Planning Time: 0.346 ms
 Execution Time: 143.236 ms
(22 строки)
```

Рисунок 6 – Анализ формирующей выборки

1.6 Вопрос 11

```
demo=# EXPLAIN ANALYZE
demo-# SELECT * FROM flights_v;
                                                             QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=10.68..34830.88 rows=33121 width=255) (actual time=0.242..625.935 rows=33121 loops=1)
   Hash Cond: (f.arrival_airport = ml_1.airport_code)
   ->  Hash Join  (cost=5.34..786.03 rows=33121 width=188) (actual time=0.099..47.708 rows=33121 loops=1)
         Hash Cond: (f.departure_airport = ml.airport_code)
         ->  Seq Scan on flights f  (cost=0.00..690.21 rows=33121 width=63) (actual time=0.004..11.029 rows=33121 loops=1)
         ->  Hash  (cost=4.04..4.04 rows=104 width=129) (actual time=0.090..0.090 rows=104 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 20kB
               ->  Seq Scan on airports_data ml  (cost=0.00..4.04 rows=104 width=129) (actual time=0.002..0.041 rows=104 loops=1)
   ->  Hash  (cost=4.04..4.04 rows=104 width=129) (actual time=0.099..0.099 rows=104 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 20kB
         ->  Seq Scan on airports_data ml_1  (cost=0.00..4.04 rows=104 width=129) (actual time=0.007..0.047 rows=104 loops=1)
 Planning Time: 0.427 ms
 Execution Time: 633.153 ms
(13 строк)

demo=# EXPLAIN ANALYZE
demo-# SELECT * FROM flights_tt;
                                                     QUERY PLAN
--------------------------------------------------------------------------------------------------------------
 Seq Scan on flights_tt  (cost=0.00..1027.20 rows=17120 width=362) (actual time=0.006..10.341 rows=33121 loops=1)
 Planning Time: 0.123 ms
 Execution Time: 16.340 ms
(3 строки)
```

Рисунок 7 – Выполнение простых запросов для исходной и временной таблицы

Так как таблица flights_v является представлением, то она формируется каждый раз при обращении к ней. Временная же таблица формируется лишь один раз. Отсюда и разница во времени выполнения запроса и Query plan.

```
demo=# EXPLAIN ANALYZE SELECT arrival_city, count ( * ) FROM flights_v
demo-# WHERE arrival_city = 'Уфа'
demo-# GROUP BY arrival_city;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=35.91..899.62 rows=1 width=40) (actual time=22.662..22.662 rows=1 loops=1)
   Group Key: (ml_1.city ->> lang())
   ->  Hash Join  (cost=35.91..897.77 rows=318 width=32) (actual time=0.475..22.508 rows=384 loops=1)
         Hash Cond: (f.departure_airport = ml.airport_code)
         ->  Hash Join  (cost=30.57..811.26 rows=318 width=53) (actual time=0.387..20.900 rows=384 loops=1)
               Hash Cond: (f.arrival_airport = ml_1.airport_code)
               ->  Seq Scan on flights f  (cost=0.00..690.21 rows=33121 width=8) (actual time=0.004..9.093 rows=33121 loops=1)
               ->  Hash  (cost=30.56..30.56 rows=1 width=53) (actual time=0.376..0.376 rows=1 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 5kB
                     ->  Seq Scan on airports_data ml_1  (cost=0.00..30.56 rows=1 width=53) (actual time=0.301..0.374 rows=1 loops=1)
                           Filter: ((city ->> lang()) = 'Уфа'::text)
                           Rows Removed by Filter: 103
         ->  Hash  (cost=4.04..4.04 rows=104 width=4) (actual time=0.080..0.080 rows=104 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 7kB
               ->  Seq Scan on airports_data ml  (cost=0.00..4.04 rows=104 width=4) (actual time=0.006..0.042 rows=104 loops=1)
 Planning Time: 0.379 ms
 Execution Time: 22.709 ms
(17 строк)

demo=# EXPLAIN ANALYZE SELECT arrival_city, count ( * ) FROM flights_tt
demo-# WHERE arrival_city = 'Уфа'
demo-# GROUP BY arrival_city;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=0.00..1071.13 rows=70 width=40) (actual time=10.550..10.551 rows=1 loops=1)
   Group Key: arrival_city
   ->  Seq Scan on flights_tt  (cost=0.00..1070.00 rows=86 width=32) (actual time=0.012..10.428 rows=384 loops=1)
         Filter: (arrival_city = 'Уфа'::text)
         Rows Removed by Filter: 32737
 Planning Time: 0.146 ms
 Execution Time: 10.577 ms
(7 строк)
```

Рисунок 8 – Сложный запрос для представления и временной таблицы

1.7 Вопрос 13

```
GroupAggregate  (cost=13325.89..13337.75 rows=200 width=16) (actual time=165.432..165.432 rows=0 loops=1)
  Group Key: count_tickets.num_tickets
  ->  Sort  (cost=13325.89..13329.18 rows=1314 width=8) (actual time=165.431..165.431 rows=0 loops=1)
        Sort Key: count_tickets.num_tickets DESC
        Sort Method: quicksort  Memory: 17kB
        ->  Subquery Scan on count_tickets  (cost=13034.17..13257.83 rows=1314 width=8) (actual time=165.428..165.429 rows=0 loops=1)
              ->  Finalize GroupAggregate  (cost=13034.17..13244.69 rows=1314 width=15) (actual time=165.427..165.427 rows=0 loops=1)
                    Group Key: b.book_ref
                    ->  Gather Merge  (cost=13034.17..13223.91 rows=1528 width=15) (actual time=165.427..165.921 rows=0 loops=1)
                          Workers Planned: 2
                          Workers Launched: 2
                          ->  Partial GroupAggregate  (cost=12034.14..12047.51 rows=764 width=15) (actual time=156.379..156.379 rows=0 loops=3)
                                Group Key: b.book_ref
                                ->  Sort  (cost=12034.14..12036.05 rows=764 width=7) (actual time=156.378..156.378 rows=0 loops=3)
                                      Sort Key: b.book_ref
                                      Sort Method: quicksort  Memory: 17kB
                                      Worker 0:  Sort Method: quicksort  Memory: 17kB
                                      Worker 1:  Sort Method: quicksort  Memory: 17kB
                                      ->  Parallel Hash Join  (cost=4002.38..11997.56 rows=764 width=7) (actual time=156.314..156.314 rows=0 loops=3)
                                            Hash Cond: (t.book_ref = b.book_ref)
                                            ->  Parallel Seq Scan on tickets t  (cost=0.00..7594.05 rows=152805 width=7) (never executed)
                                            ->  Parallel Hash  (cost=3992.72..3992.72 rows=773 width=7) (actual time=156.255..156.256 rows=0 loops=3)
                                                  Buckets: 2048  Batches: 1  Memory Usage: 0kB
                                                  ->  Parallel Seq Scan on bookings b  (cost=0.00..3992.72 rows=773 width=7) (actual time=154.382..154.382 rows=0 loops=3)
                                                        Filter: (date_trunc('mon'::text, book_date) = '2016-09-01 00:00:00+07'::timestamp with time zone)
                                                        Rows Removed by Filter: 87596
Planning Time: 0.233 ms
Execution Time: 165.998 ms
```

Рисунок 9 – Исходный запрос

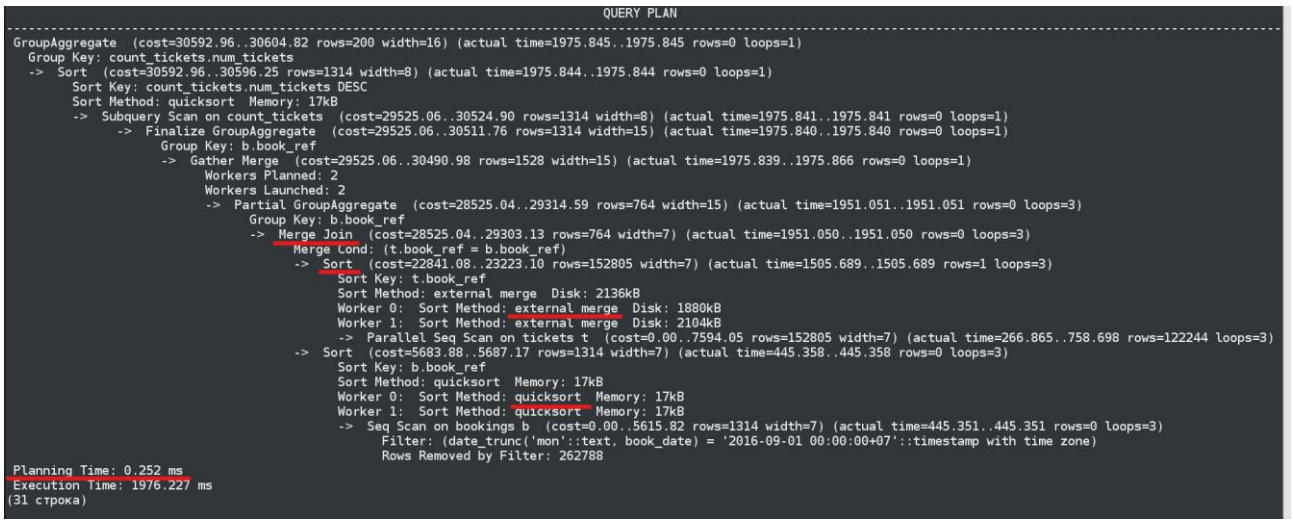

Рисунок 10 – Запрос с отключенным hashjoin

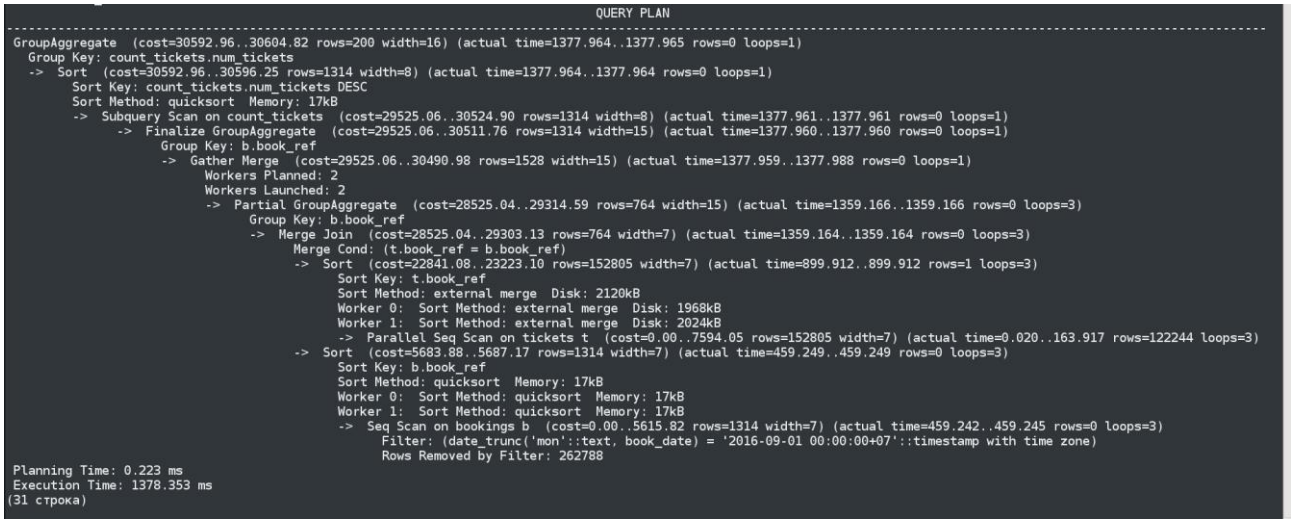

Рисунок 11 – Запрос с отключенным nested loop

Везде используется Seq scan.

1.8 Вопрос 15

```
demo=# EXPLAIN SELECT model, range,
demo-# CASE WHEN range < 2000 THEN 'Ближнемагистральный'
demo-# WHEN range < 5000 THEN 'Среднемагистральный'
demo-# ELSE 'Дальнемагистральный'
demo-# END AS type
demo-# FROM aircrafts
demo-# ORDER BY model;
                                QUERY PLAN
-------------------------------------------------------------------------
 Sort  (cost=3.55..3.57 rows=9 width=68)
   Sort Key: ((ml.model ->> lang()))
   ->  Seq Scan on aircrafts_data ml  (cost=0.00..3.41 rows=9 width=68)
(3 строки)
```

Рисунок 12 – Первый запрос

```
demo=# EXPLAIN SELECT s.seat_no, s.fare_conditions
FROM seats s
JOIN aircrafts a ON s.aircraft_code = a.aircraft_code
WHERE a.model ~ '^Cessna'
ORDER BY s.seat_no;
                                              QUERY PLAN
------------------------------------------------------------------------------------------------------
 Sort  (cost=21.26..21.63 rows=149 width=11)
   Sort Key: s.seat_no
   ->  Nested Loop  (cost=0.28..15.88 rows=149 width=11)
         ->  Seq Scan on aircrafts_data ml  (cost=0.00..3.39 rows=1 width=16)
               Filter: ((model ->> lang()) ~ '^Cessna'::text)
         ->  Index Scan using seats_aircraft_code_idx on seats s  (cost=0.28..11.00 rows=149 width=15)
               Index Cond: (aircraft_code = ml.aircraft_code)
(7 строк)
```

Рисунок 13 – Второй запрос

```
demo=# EXPLAIN SELECT a.aircraft_code AS a_code,
demo-# a.model,
demo-# r.aircraft_code AS r_code,
demo-# count( r.aircraft_code ) AS num_routes
demo-# FROM aircrafts a
demo-# LEFT OUTER JOIN routes r ON r.aircraft_code = a.aircraft_code
demo-# GROUP BY 1, 2, 3
demo-# ORDER BY 4 DESC;
                                                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
 Sort  (cost=2741.55..2741.58 rows=12 width=72)
   Sort Key: (count(f3.aircraft_code)) DESC
   ->  GroupAggregate  (cost=2738.04..2741.34 rows=12 width=72)
         Group Key: ml.aircraft_code, ((ml.model ->> lang())), f3.aircraft_code
         ->  Sort  (cost=2738.04..2738.07 rows=12 width=64)
               Sort Key: ml.aircraft_code, ((ml.model ->> lang())), f3.aircraft_code
               ->  Hash Right Join  (cost=2706.74..2737.82 rows=12 width=64)
                     Hash Cond: (f3.aircraft_code = ml.aircraft_code)
                     ->  Hash Join  (cost=2705.54..2730.10 rows=276 width=252)
                           Hash Cond: (f3.arrival_airport = ml_2.airport_code)
                           CTE f3
                             ->  GroupAggregate  (cost=2401.55..2694.86 rows=1020 width=67)
                                   Group Key: flights.flight_no, flights.departure_airport, flights.arrival_airport, flights.aircraft_code, ((flights.scheduled_arrival - fli
ghts.scheduled_departure))
                                   ->  Sort  (cost=2401.55..2427.06 rows=10202 width=39)
                                         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)
                                         ->  HashAggregate  (cost=1518.24..1722.28 rows=10202 width=39)
                                               Group Key: flights.flight_no, flights.departure_airport, flights.arrival_airport, flights.aircraft_code, (flights.scheduled_ar
rival - flights.scheduled_departure), (to_char(flights.scheduled_departure, 'ID'::text))::integer
                                               ->  Seq Scan on flights  (cost=0.00..1021.42 rows=33121 width=39)
                           ->  Hash Join  (cost=5.34..28.47 rows=530 width=32)
                                 Hash Cond: (f3.departure_airport = ml_1.airport_code)
                                 ->  CTE Scan on f3  (cost=0.00..20.40 rows=1020 width=48)
                                 ->  Hash  (cost=4.04..4.04 rows=104 width=4)
                                       ->  Seq Scan on airports_data ml_1  (cost=0.00..4.04 rows=104 width=4)
                           ->  Hash  (cost=4.04..4.04 rows=104 width=4)
                                 ->  Seq Scan on airports_data ml_2  (cost=0.00..4.04 rows=104 width=4)
                     ->  Hash  (cost=1.09..1.09 rows=9 width=48)
                           ->  Seq Scan on aircrafts_data ml  (cost=0.00..1.09 rows=9 width=48)
(27 строк)
```

Рисунок 14 – Третий запрос

```
demo=# EXPLAIN SELECT count( * )
demo-# FROM ( ticket_flights t
demo(# JOIN flights f ON t.flight_id = f.flight_id
demo(# )
demo-# LEFT OUTER JOIN boarding_passes b
demo-# ON t.ticket_no = b.ticket_no AND t.flight_id = b.flight_id
demo-# WHERE f.actual_departure IS NOT NULL AND b.flight_id IS NULL;
                                                    QUERY PLAN
---------------------------------------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=33693.92..33693.93 rows=1 width=8)
   ->  Gather  (cost=33693.70..33693.91 rows=2 width=8)
         Workers Planned: 2
         ->  Partial Aggregate  (cost=32693.70..32693.71 rows=1 width=8)
               ->  Parallel Hash Anti Join  (cost=12234.47..32353.55 rows=136063 width=0)
                     Hash Cond: ((t.ticket_no = b.ticket_no) AND (t.flight_id = b.flight_id))
                     ->  Hash Join  (cost=900.07..14636.17 rows=220865 width=18)
                           Hash Cond: (t.flight_id = f.flight_id)
                           ->  Parallel Seq Scan on ticket_flights t  (cost=0.00..12592.19 rows=435719 width=18)
                           ->  Hash  (cost=690.21..690.21 rows=16789 width=4)
                                 ->  Seq Scan on flights f  (cost=0.00..690.21 rows=16789 width=4)
                                       Filter: (actual_departure IS NOT NULL)
                     ->  Parallel Hash  (cost=6413.36..6413.36 rows=241536 width=18)
                           ->  Parallel Seq Scan on boarding_passes b  (cost=0.00..6413.36 rows=241536 width=18)
(14 строк)
```

Рисунок 15 – Четвертый запрос

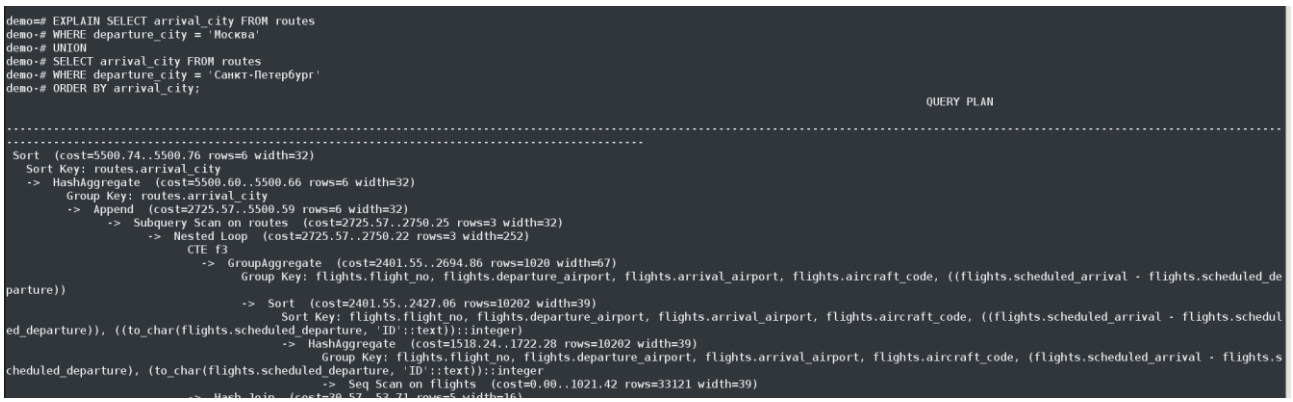

Рисунок 16 – Пятый запрос

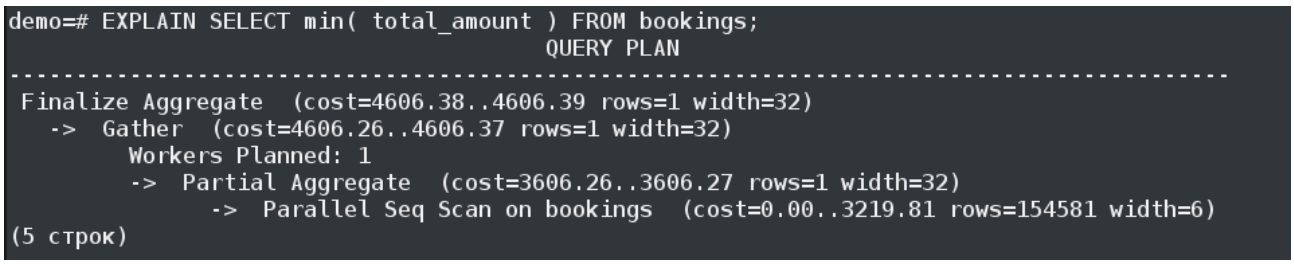

Рисунок 17 – Шестой запрос

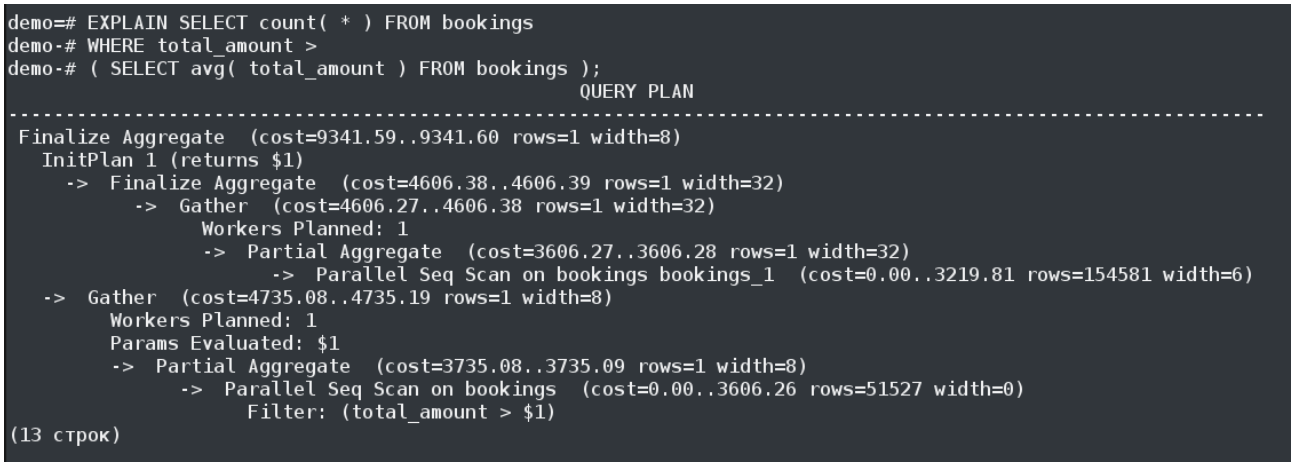

Рисунок 18 – Седьмой запрос

1.9 Вопрос 17

С разделом документации 14.2 «Статистика, используемая планировщиком» ознакомился.

1.10 Вопрос 19

С разделом документации 14.4 «Наполнение базы данных» ознакомился.

```
demo=# BEGIN;
BEGIN
demo=# COMMIT;
COMMIT
demo=#
```

Рисунок 19 – Создание транзакции перед добавлением

```
demo=# BEGIN;
BEGIN
demo=# COPY aircrafts_temp FROM '/home/postgres/some_file';
COPY 9
demo=# ROLLBACK;
ROLLBACK
demo=#
```

Рисунок 20 – Использование COPY вместо INSERT

```
demo=# DROP INDEX aircrafts_temp_model_idx;
DROP INDEX
demo=# COPY aircrafts_temp FROM '/home/postgres/some_file';
COPY 9
demo=# CREATE INDEX ON aircrafts_temp (model);
CREATE INDEX
```

Рисунок 21 – Удаление индекса и его воссоздание после добавления