МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
Национальный исследовательский ядерный университет «МИФИ»

Институт
интеллектуальных кибернетических систем

Кафедра кибернетики (№ 22)

Направление подготовки 09.04.04 Программная инженерия

Лабораторная работа на тему

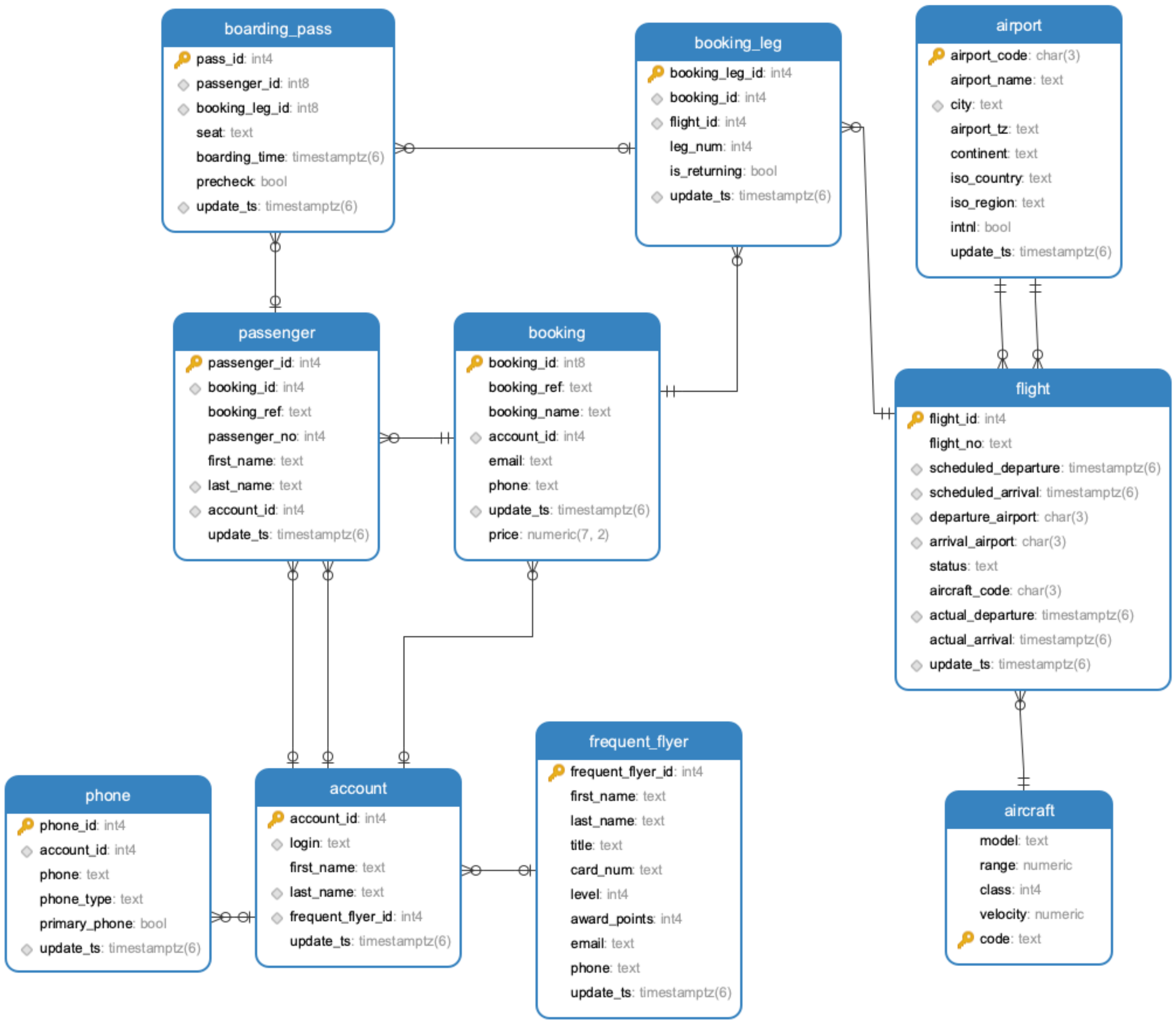
«Анализ планов запросов»

по дисциплине «Проектирование баз данных кибернетических систем»

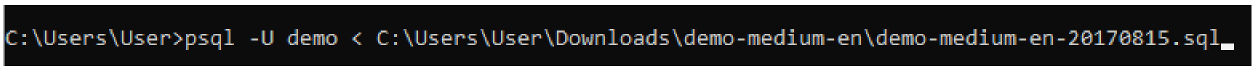
Выполнила: Клычков М.Д.
Группа: М20-504

Москва 2021

BD schema

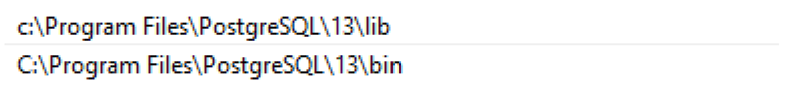


В своей работе я решил использовать тренировочную базу данных Postgres. Для этого я создал пустую базу данных “demo”, чтобы сдампить в нее базу “airplanes”. Для этого я использовал консольную команду:



```
C:\Users\User>psql -U demo < C:\Users\User\Downloads\demo-medium-en\demo-medium-en-20170815.sql
```

Так же добавил пути в переменные среды пользователя, чтобы можно было пользоваться командами из консоли.

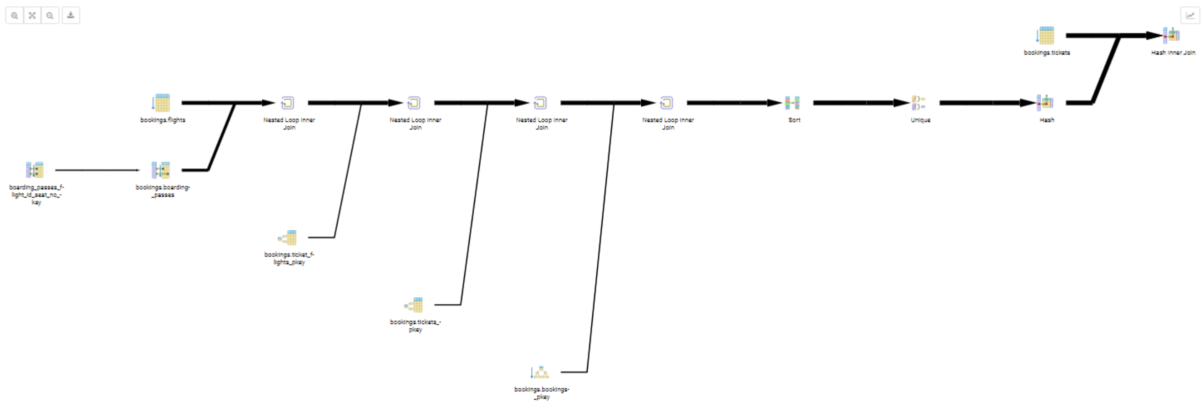


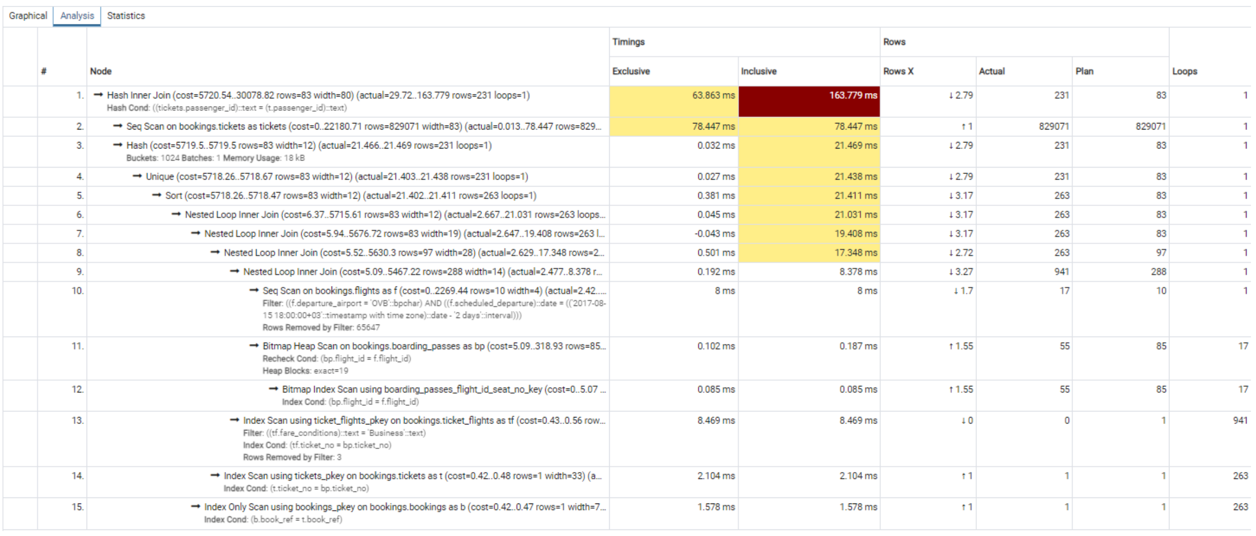
```
c:\Program Files\PostgreSQL\13\lib
C:\Program Files\PostgreSQL\13\bin
```

Contents

1. Вывести всех, кто осуществлял перелет в аэропорту OVB, будучи пассажирами бизнес-класса.

```
select
            passenger_name,
            contact_data::json->'phone' as client_phone,
            contact_data::json->'email' as client_email
            FROM tickets
            where passenger_id in (SELECT DISTINCT passenger_id
FROM bookings b
LEFT JOIN tickets AS t
    ON t.book_ref = b.book_ref
LEFT JOIN boarding_passes AS bp
    ON bp.ticket_no = t.ticket_no
LEFT JOIN ticket_flights AS tf
      ON tf.ticket_no = t.ticket_no
LEFT JOIN flights AS f
    ON f.flight_id = bp.flight_id
WHERE
tf.fare_conditions = 'Business'
AND f.departure_airport = 'OVB'
AND f.scheduled_departure::date = bookings.now()::date - INTERVAL '2 day');
```

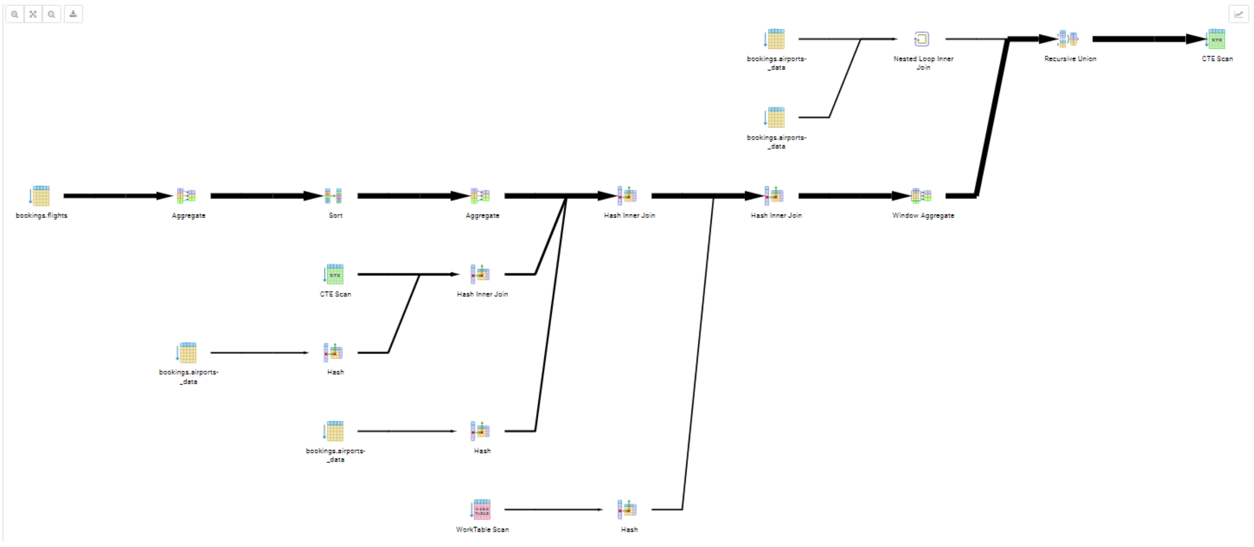


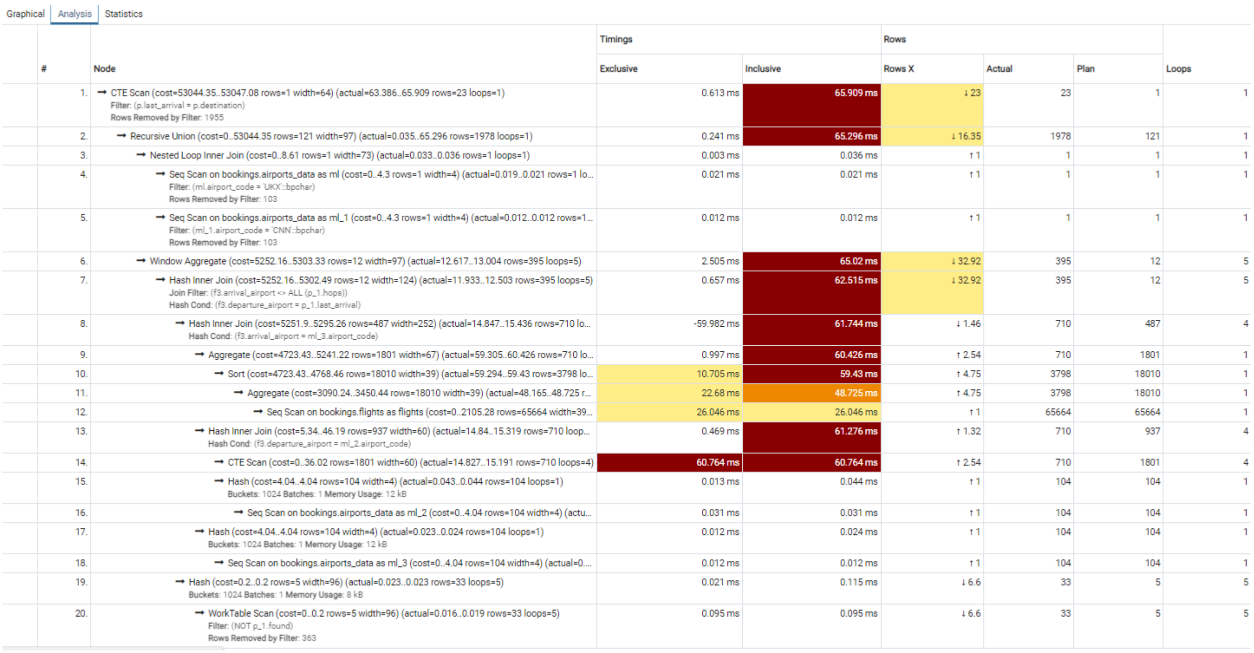

Graphical | Analysis | Statistics

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Hash Inner Join (cost=5720.54..30078.82 rows=83 width=80) (actual=29.72..163.779 rows=231 loops=1) Hash Cond: ((tickets.passenger_id)::text = (t.passenger_id)::text)	63.863 ms	163.779 ms	↓ 2.79	231	83	1
2.	→ Seq Scan on bookings.tickets as tickets (cost=0..22180.71 rows=829071 width=83) (actual=0.013..78.447 rows=829...	78.447 ms	78.447 ms	↑ 1	829071	829071	1
3.	→ Hash (cost=5719.5..5719.5 rows=83 width=12) (actual=21.466..21.469 rows=231 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 18 kB	0.032 ms	21.469 ms	↓ 2.79	231	83	1
4.	→ Unique (cost=5718.26..5718.67 rows=83 width=12) (actual=21.403..21.438 rows=231 loops=1)	0.027 ms	21.438 ms	↓ 2.79	231	83	1
5.	→ Sort (cost=5718.26..5718.47 rows=83 width=12) (actual=21.402..21.411 rows=263 loops=1)	0.381 ms	21.411 ms	↓ 3.17	263	83	1
6.	→ Nested Loop Inner Join (cost=6.37..5715.61 rows=83 width=12) (actual=2.667..21.031 rows=263 loops...	0.045 ms	21.031 ms	↓ 3.17	263	83	1
7.	→ Nested Loop Inner Join (cost=5.94..5676.72 rows=83 width=19) (actual=2.647..19.408 rows=263 l...	-0.043 ms	19.408 ms	↓ 3.17	263	83	1
8.	→ Nested Loop Inner Join (cost=5.52..5630.3 rows=97 width=28) (actual=2.629..17.348 rows=2...	0.501 ms	17.348 ms	↓ 2.72	263	97	1
9.	→ Nested Loop Inner Join (cost=5.09..5467.22 rows=288 width=14) (actual=2.477..8.378 r...	0.192 ms	8.378 ms	↓ 3.27	941	288	1
10.	→ Seq Scan on bookings.flights as f (cost=0..2269.44 rows=10 width=4) (actual=2.42.... Filter: ((f.departure_airport = 'OVB'::bpchar) AND ((f.scheduled_departure)::date = (('2017-08-15 18:00:00+03'::timestamp with time zone)::date - '2 days'::interval))) Rows Removed by Filter: 65647	8 ms	8 ms	↓ 1.7	17	10	1
11.	→ Bitmap Heap Scan on bookings.boarding_passes as bp (cost=5.09..318.93 rows=85... Recheck Cond: (bp.flight_id = f.flight_id) Heap Blocks: exact=19	0.102 ms	0.187 ms	↑ 1.55	55	85	17
12.	→ Bitmap Index Scan using boarding_passes_flight_id_seat_no_key (cost=0..5.07 ... Index Cond: (bp.flight_id = f.flight_id)	0.085 ms	0.085 ms	↑ 1.55	55	85	17
13.	→ Index Scan using ticket_flights_pkey on bookings.ticket_flights as tf (cost=0.43..0.56 row... Filter: ((tf.fare_conditions)::text = 'Business'::text) Index Cond: (tf.ticket_no = bp.ticket_no) Rows Removed by Filter: 3	8.469 ms	8.469 ms	↓ 0	0	1	941
14.	→ Index Scan using tickets_pkey on bookings.tickets as t (cost=0.42..0.48 rows=1 width=33) (a... Index Cond: (t.ticket_no = bp.ticket_no)	2.104 ms	2.104 ms	↑ 1	1	1	263
15.	→ Index Only Scan using bookings_pkey on bookings.bookings as b (cost=0.42..0.47 rows=1 width=7... Index Cond: (b.book_ref = t.book_ref)	1.578 ms	1.578 ms	↑ 1	1	1	263

2. Как добраться из Усть-Кута (UKX) в Нерюнгри (CNN) с минимальным количеством стыковок, и каково будет время полета?

```
WITH RECURSIVE p(last_arrival, destination, hops, flights, found) AS (
SELECT a_from.airport_code,
         a_to.airport_code,
         ARRAY[a_from.airport_code],
         ARRAY[]::char(6)[],
         a_from.airport_code = a_to.airport_code
  FROM   airports a_from, airports a_to
  WHERE  a_from.airport_code = 'UKX'
  AND    a_to.airport_code = 'CNN'
  UNION ALL
  SELECT r.arrival_airport,
         p.destination,
         (p.hops || r.arrival_airport)::char(3)[],
         (p.flights || r.flight_no)::char(6)[],
         bool_or(r.arrival_airport = p.destination) OVER ()
  FROM   routes r, p
  WHERE  r.departure_airport = p.last_arrival
  AND    NOT r.arrival_airport = ANY(p.hops)
  AND    NOT p.found
)
SELECT hops,
       flights
FROM   p
WHERE  p.last_arrival = p.destination;
```

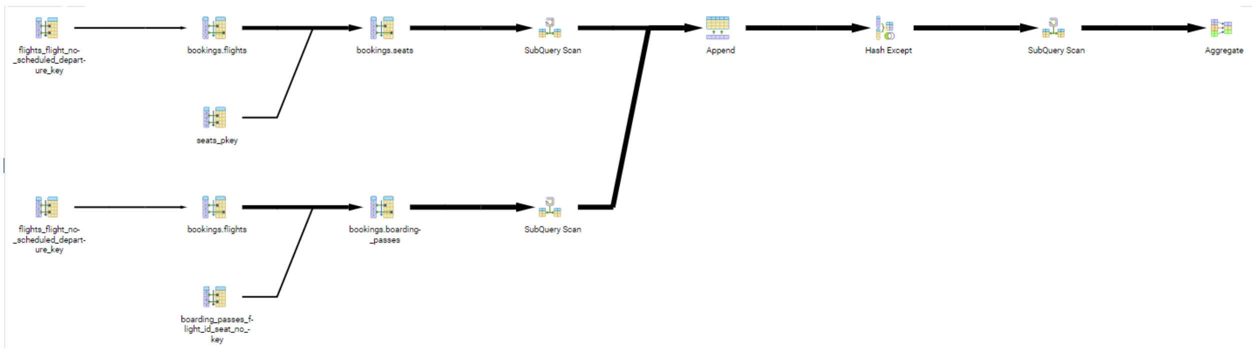


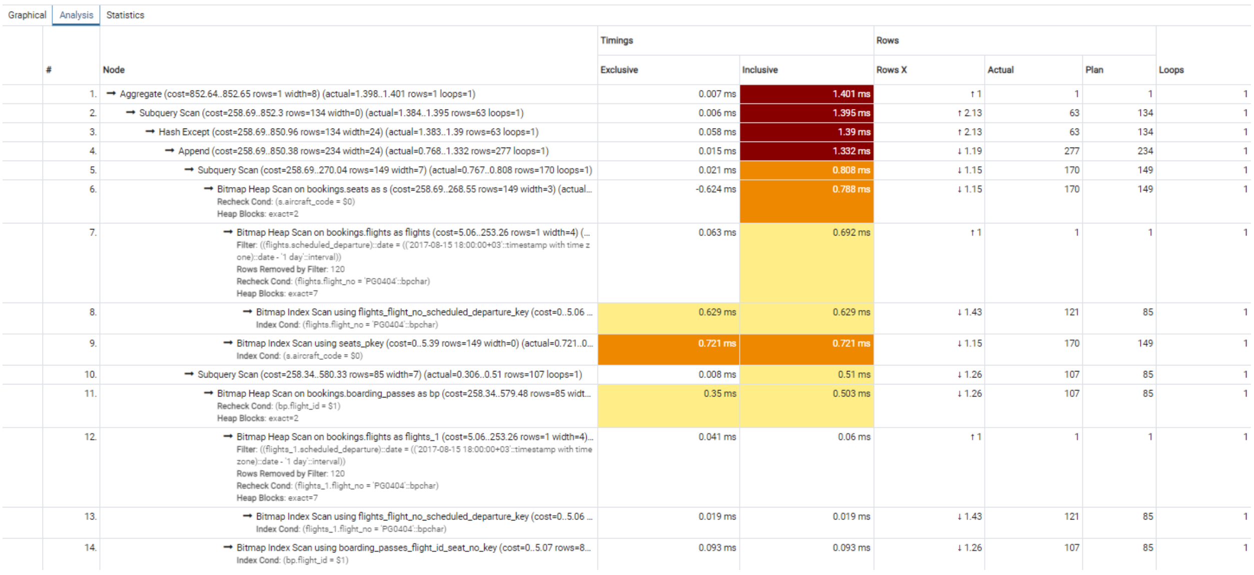

Graphical | Analysis | Statistics

#	Node	Timings		Rows			
		Exclusive	Inclusive	Rows X	Actual	Plan	Loops
1.	→ CTE Scan (cost=53044.35..53047.08 rows=1 width=64) (actual=63.386..65.909 rows=23 loops=1) Filter: (p.last_arrival = p.destination) Rows Removed by Filter: 1955	0.613 ms	65.909 ms	↓ 23	23	1	1
2.	→ Recursive Union (cost=0..53044.35 rows=121 width=97) (actual=0.035..65.296 rows=1978 loops=1)	0.241 ms	65.296 ms	↓ 16.35	1978	121	1
3.	→ Nested Loop Inner Join (cost=0..8.61 rows=1 width=73) (actual=0.033..0.036 rows=1 loops=1)	0.003 ms	0.036 ms	↑ 1	1	1	1
4.	→ Seq Scan on bookings.airports_data as ml (cost=0..4.3 rows=1 width=4) (actual=0.019..0.021 rows=1 lo... Filter: (ml.airport_code = 'UKX'::bpchar) Rows Removed by Filter: 103	0.021 ms	0.021 ms	↑ 1	1	1	1
5.	→ Seq Scan on bookings.airports_data as ml_1 (cost=0..4.3 rows=1 width=4) (actual=0.012..0.012 rows=1... Filter: (ml_1.airport_code = 'CNN'::bpchar) Rows Removed by Filter: 103	0.012 ms	0.012 ms	↑ 1	1	1	1
6.	→ Window Aggregate (cost=5252.16..5303.33 rows=12 width=97) (actual=12.617..13.004 rows=395 loops=5)	2.505 ms	65.02 ms	↓ 32.92	395	12	5
7.	→ Hash Inner Join (cost=5252.16..5302.49 rows=12 width=124) (actual=11.933..12.503 rows=395 loops=5) Join Filter: (f3.arrival_airport <> ALL (p_1.hops)) Hash Cond: (f3.departure_airport = p_1.last_arrival)	0.657 ms	62.515 ms	↓ 32.92	395	12	5
8.	→ Hash Inner Join (cost=5251.9..5295.26 rows=487 width=252) (actual=14.847..15.436 rows=710 lo... Hash Cond: (f3.arrival_airport = ml_3.airport_code)	-59.982 ms	61.744 ms	↓ 1.46	710	487	4
9.	→ Aggregate (cost=4723.43..5241.22 rows=1801 width=67) (actual=59.305..60.426 rows=710 lo...	0.997 ms	60.426 ms	↑ 2.54	710	1801	1
10.	→ Sort (cost=4723.43..4768.46 rows=18010 width=39) (actual=59.294..59.43 rows=3798 lo...	10.705 ms	59.43 ms	↑ 4.75	3798	18010	1
11.	→ Aggregate (cost=3090.24..3450.44 rows=18010 width=39) (actual=48.165..48.725 r...	22.68 ms	48.725 ms	↑ 4.75	3798	18010	1
12.	→ Seq Scan on bookings.flights as flights (cost=0..2105.28 rows=65664 width=39...	26.046 ms	26.046 ms	↑ 1	65664	65664	1
13.	→ Hash Inner Join (cost=5.34..46.19 rows=937 width=60) (actual=14.84..15.319 rows=710 loop... Hash Cond: (f3.departure_airport = ml_2.airport_code)	0.469 ms	61.276 ms	↑ 1.32	710	937	4
14.	→ CTE Scan (cost=0..36.02 rows=1801 width=60) (actual=14.827..15.191 rows=710 loops=4)	60.764 ms	60.764 ms	↑ 2.54	710	1801	4
15.	→ Hash (cost=4.04..4.04 rows=104 width=4) (actual=0.043..0.044 rows=104 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 12 kB	0.013 ms	0.044 ms	↑ 1	104	104	1
16.	→ Seq Scan on bookings.airports_data as ml_2 (cost=0..4.04 rows=104 width=4) (actu...	0.031 ms	0.031 ms	↑ 1	104	104	1
17.	→ Hash (cost=4.04..4.04 rows=104 width=4) (actual=0.023..0.024 rows=104 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 12 kB	0.012 ms	0.024 ms	↑ 1	104	104	1
18.	→ Seq Scan on bookings.airports_data as ml_3 (cost=0..4.04 rows=104 width=4) (actual=0....	0.012 ms	0.012 ms	↑ 1	104	104	1
19.	→ Hash (cost=0.2..0.2 rows=5 width=96) (actual=0.023..0.023 rows=33 loops=5) Buckets: 1024 Batches: 1 Memory Usage: 8 kB	0.021 ms	0.115 ms	↓ 6.6	33	5	5
20.	→ WorkTable Scan (cost=0..0.2 rows=5 width=96) (actual=0.016..0.019 rows=33 loops=5) Filter: (NOT p_1.found) Rows Removed by Filter: 363	0.095 ms	0.095 ms	↓ 6.6	33	5	5

3. Сколько мест оставалось свободными на рейсе PG0404 вчера?

```
SELECT
  count(*)
FROM
  (
    SELECT
      s.seat_no
    FROM
      seats s
    WHERE
      s.aircraft_code = (
        SELECT
          aircraft_code
        FROM
          flights
        WHERE
          flight_no = 'PG0404'
          AND scheduled_departure :: date = bookings.now():: date - INTERVAL
'1 day'
      )
    EXCEPT
    SELECT
      bp.seat_no
    FROM
      boarding_passes bp
    WHERE
      bp.flight_id = (
        SELECT
          flight_id
        FROM
          flights
        WHERE
          flight_no = 'PG0404'
          AND scheduled_departure :: date = bookings.now():: date - INTERVAL
'1 day'
      )
 ) t;
```

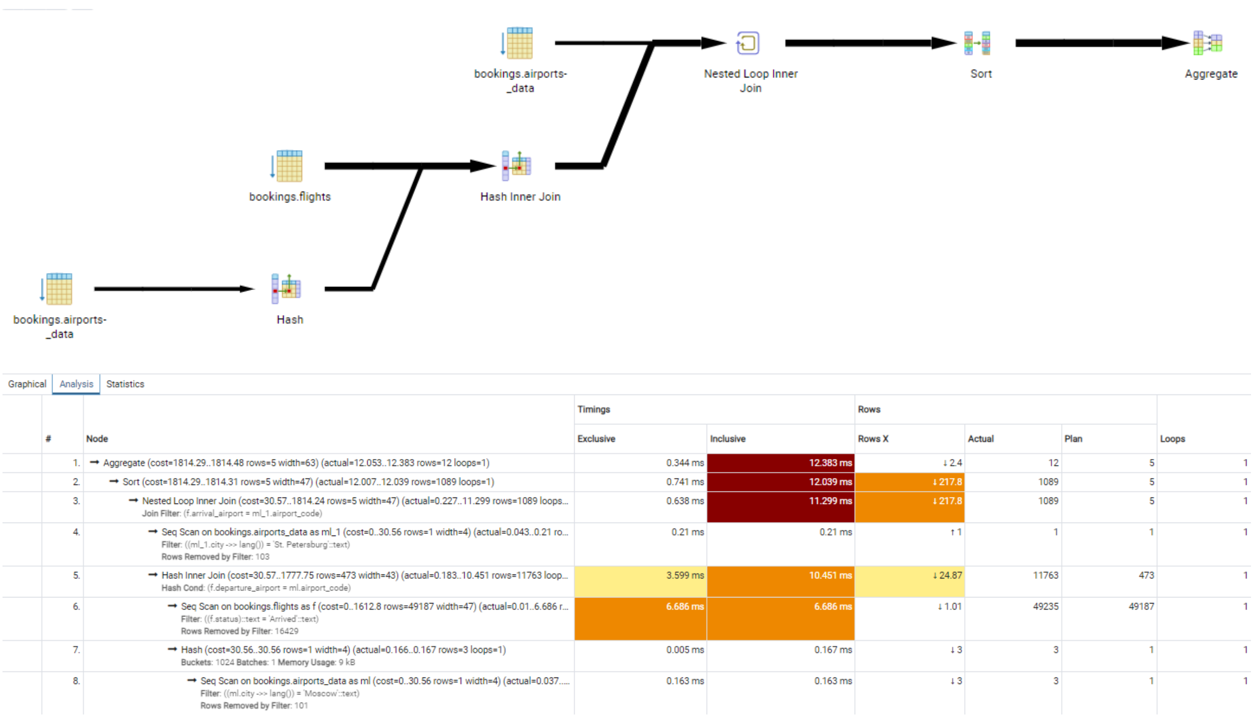

flights_flight_no-_scheduled_depart-ure_key
bookings.flights
bookings.seats
SubQuery Scan
Append
Hash Except
SubQuery Scan
Aggregate
seats_pkey
flights_flight_no-_scheduled_depart-ure_key
bookings.flights
bookings.boarding-_passes
SubQuery Scan
boarding_passes_f-light_id_seat_no_-key


Graphical | Analysis | Statistics

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Aggregate (cost=852.64..852.65 rows=1 width=8) (actual=1.398..1.401 rows=1 loops=1)	0.007 ms	1.401 ms	↑ 1	1	1	1
2.	→ Subquery Scan (cost=258.69..852.3 rows=134 width=0) (actual=1.384..1.395 rows=63 loops=1)	0.006 ms	1.395 ms	↑ 2.13	63	134	1
3.	→ Hash Except (cost=258.69..850.96 rows=134 width=24) (actual=1.383..1.39 rows=63 loops=1)	0.058 ms	1.39 ms	↑ 2.13	63	134	1
4.	→ Append (cost=258.69..850.38 rows=234 width=24) (actual=0.768..1.332 rows=277 loops=1)	0.015 ms	1.332 ms	↓ 1.19	277	234	1
5.	→ Subquery Scan (cost=258.69..270.04 rows=149 width=7) (actual=0.767..0.808 rows=170 loops=1)	0.021 ms	0.808 ms	↓ 1.15	170	149	1
6.	→ Bitmap Heap Scan on bookings.seats as s (cost=258.69..268.55 rows=149 width=3) (actual... Recheck Cond: (s.aircraft_code = $0) Heap Blocks: exact=2	-0.624 ms	0.788 ms	↓ 1.15	170	149	1
7.	→ Bitmap Heap Scan on bookings.flights as flights (cost=5.06..253.26 rows=1 width=4) (... Filter: ((flights.scheduled_departure)::date = (('2017-08-15 18:00:00+03'::timestamp with time z one)::date - '1 day'::interval)) Rows Removed by Filter: 120 Recheck Cond: (flights.flight_no = 'PG0404'::bpchar) Heap Blocks: exact=7	0.063 ms	0.692 ms	↑ 1	1	1	1
8.	→ Bitmap Index Scan using flights_flight_no_scheduled_departure_key (cost=0..5.06 ... Index Cond: (flights.flight_no = 'PG0404'::bpchar)	0.629 ms	0.629 ms	↓ 1.43	121	85	1
9.	→ Bitmap Index Scan using seats_pkey (cost=0..5.39 rows=149 width=0) (actual=0.721..0... Index Cond: (s.aircraft_code = $0)	0.721 ms	0.721 ms	↓ 1.15	170	149	1
10.	→ Subquery Scan (cost=258.34..580.33 rows=85 width=7) (actual=0.306..0.51 rows=107 loops=1)	0.008 ms	0.51 ms	↓ 1.26	107	85	1
11.	→ Bitmap Heap Scan on bookings.boarding_passes as bp (cost=258.34..579.48 rows=85 widt... Recheck Cond: (bp.flight_id = $1) Heap Blocks: exact=2	0.35 ms	0.503 ms	↓ 1.26	107	85	1
12.	→ Bitmap Heap Scan on bookings.flights as flights_1 (cost=5.06..253.26 rows=1 width=4)... Filter: ((flights_1.scheduled_departure)::date = (('2017-08-15 18:00:00+03'::timestamp with time zone)::date - '1 day'::interval)) Rows Removed by Filter: 120 Recheck Cond: (flights_1.flight_no = 'PG0404'::bpchar) Heap Blocks: exact=7	0.041 ms	0.06 ms	↑ 1	1	1	1
13.	→ Bitmap Index Scan using flights_flight_no_scheduled_departure_key (cost=0..5.06 ... Index Cond: (flights_1.flight_no = 'PG0404'::bpchar)	0.019 ms	0.019 ms	↓ 1.43	121	85	1
14.	→ Bitmap Index Scan using boarding_passes_flight_id_seat_no_key (cost=0..5.07 rows=8... Index Cond: (bp.flight_id = $1)	0.093 ms	0.093 ms	↓ 1.26	107	85	1

4. Вывести только прибывшие по факту рейсы

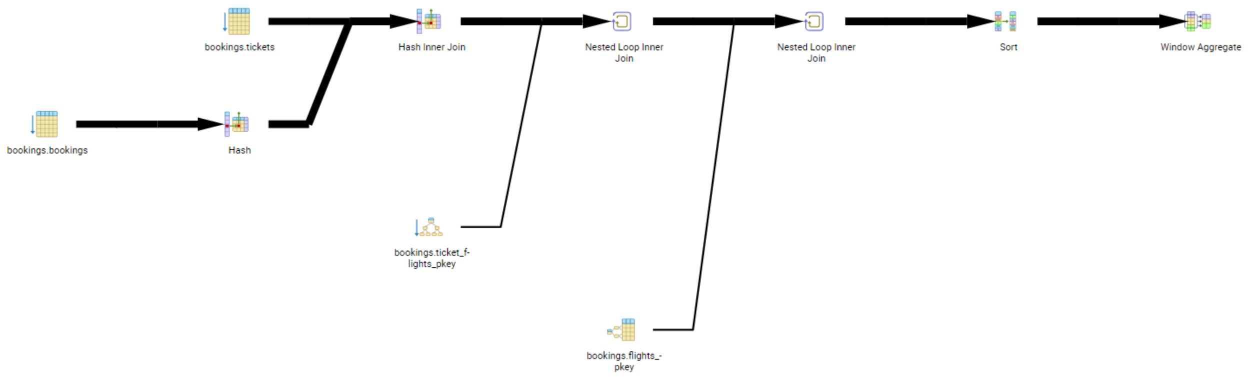
```
SELECT
  f.flight_no,
  f.scheduled_duration,
  min(f.actual_duration),
  max(f.actual_duration),
  sum(
    CASE WHEN f.actual_departure > f.scheduled_departure + INTERVAL '1 hour'
THEN 1 ELSE 0 END
  ) delays
FROM
  flights_v f
WHERE
  f.departure_city = 'Moscow'
  AND f.arrival_city = 'St. Petersburg'
  AND f.status = 'Arrived'
GROUP BY
  f.flight_no,
 f.scheduled_duration;
```

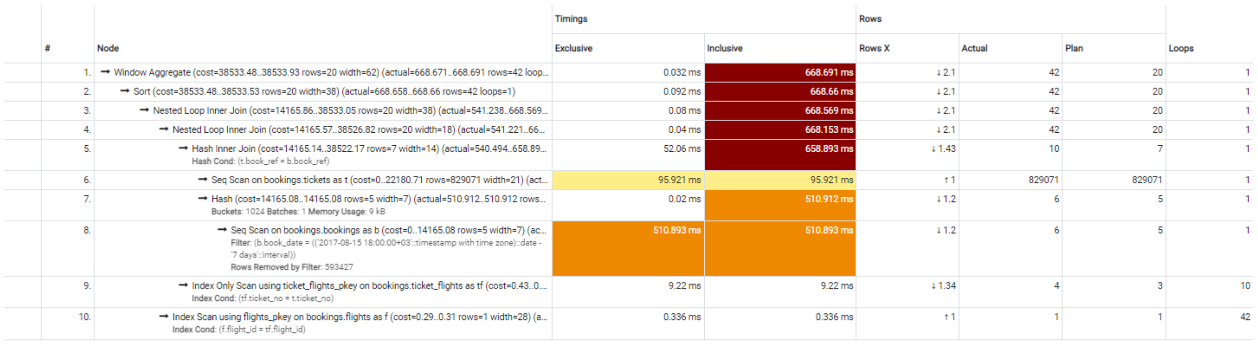


#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Aggregate (cost=1814.29..1814.48 rows=5 width=63) (actual=12.053..12.383 rows=12 loops=1)	0.344 ms	12.383 ms	↓ 2.4	12	5	1
2.	→ Sort (cost=1814.29..1814.31 rows=5 width=47) (actual=12.007..12.039 rows=1089 loops=1)	0.741 ms	12.039 ms	↓ 217.8	1089	5	1
3.	→ Nested Loop Inner Join (cost=30.57..1814.24 rows=5 width=47) (actual=0.227..11.299 rows=1089 loops... Join Filter: (f.arrival_airport = ml_1.airport_code)	0.638 ms	11.299 ms	↓ 217.8	1089	5	1
4.	→ Seq Scan on bookings.airports_data as ml_1 (cost=0..30.56 rows=1 width=4) (actual=0.043..0.21 ro... Filter: ((ml_1.city ->> lang()) = 'St. Petersburg'::text) Rows Removed by Filter: 103	0.21 ms	0.21 ms	↑ 1	1	1	1
5.	→ Hash Inner Join (cost=30.57..1777.75 rows=473 width=43) (actual=0.183..10.451 rows=11763 loop... Hash Cond: (f.departure_airport = ml.airport_code)	3.599 ms	10.451 ms	↓ 24.87	11763	473	1
6.	→ Seq Scan on bookings.flights as f (cost=0..1612.8 rows=49187 width=47) (actual=0.01..6.686 r... Filter: ((f.status)::text = 'Arrived'::text) Rows Removed by Filter: 16429	6.686 ms	6.686 ms	↓ 1.01	49235	49187	1
7.	→ Hash (cost=30.56..30.56 rows=1 width=4) (actual=0.166..0.167 rows=3 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 9 kB	0.005 ms	0.167 ms	↓ 3	3	1	1
8.	→ Seq Scan on bookings.airports_data as ml (cost=0..30.56 rows=1 width=4) (actual=0.037..... Filter: ((ml.city ->> lang()) = 'Moscow'::text) Rows Removed by Filter: 101	0.163 ms	0.163 ms	↓ 3	3	1	1

5. Для каждого билета отобразите все включенные сегменты рейса вместе со временем стыковки.

```
SELECT
  tf.ticket_no,
  f.departure_airport,
  f.arrival_airport,
  f.scheduled_arrival,
  lead(f.scheduled_departure) OVER w AS next_departure,
  lead(f.scheduled_departure) OVER w - f.scheduled_arrival AS gap
FROM
  bookings b
  JOIN tickets t ON t.book_ref = b.book_ref
  JOIN ticket_flights tf ON tf.ticket_no = t.ticket_no
  JOIN flights f ON tf.flight_id = f.flight_id
WHERE
  b.book_date = bookings.now():: date - INTERVAL '7 day' WINDOW w AS (
    PARTITION BY tf.ticket_no
    ORDER BY
      f.scheduled_departure
 );
```

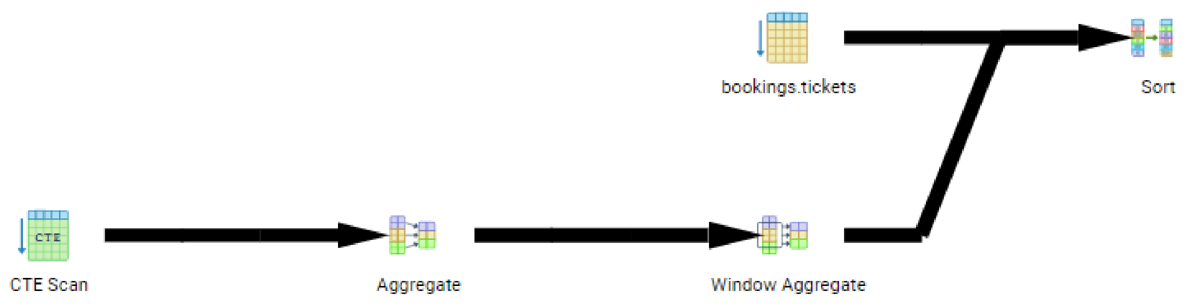


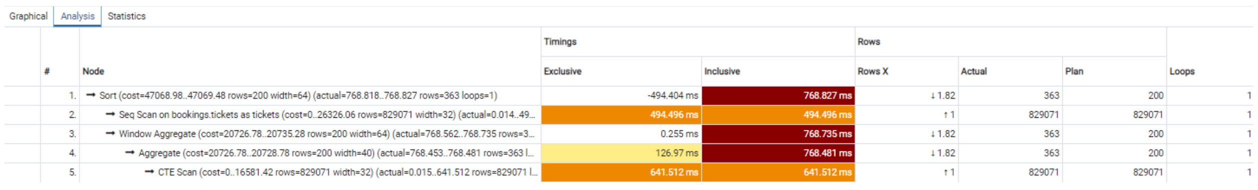


		Timings		Rows			
#	Node	Exclusive	Inclusive	Rows X	Actual	Plan	Loops
1.	→ Window Aggregate (cost=38533.48..38533.93 rows=20 width=62) (actual=668.671..668.691 rows=42 loop...	0.032 ms	668.691 ms	↓ 2.1	42	20	1
2.	→ Sort (cost=38533.48..38533.53 rows=20 width=38) (actual=668.658..668.66 rows=42 loops=1)	0.092 ms	668.66 ms	↓ 2.1	42	20	1
3.	→ Nested Loop Inner Join (cost=14165.86..38533.05 rows=20 width=38) (actual=541.238..668.569...	0.08 ms	668.569 ms	↓ 2.1	42	20	1
4.	→ Nested Loop Inner Join (cost=14165.57..38526.82 rows=20 width=18) (actual=541.221..66...	0.04 ms	668.153 ms	↓ 2.1	42	20	1
5.	→ Hash Inner Join (cost=14165.14..38522.17 rows=7 width=14) (actual=540.494..658.89... Hash Cond: (t.book_ref = b.book_ref)	52.06 ms	658.893 ms	↓ 1.43	10	7	1
6.	→ Seq Scan on bookings.tickets as t (cost=0..22180.71 rows=829071 width=21) (act...	95.921 ms	95.921 ms	↑ 1	829071	829071	1
7.	→ Hash (cost=14165.08..14165.08 rows=5 width=7) (actual=510.912..510.912 rows... Buckets: 1024 Batches: 1 Memory Usage: 9 kB	0.02 ms	510.912 ms	↓ 1.2	6	5	1
8.	→ Seq Scan on bookings.bookings as b (cost=0..14165.08 rows=5 width=7) (ac... Filter: (b.book_date = (('2017-08-15 18:00:00+03'::timestamp with time zone)::date - '7 days'::interval)) Rows Removed by Filter: 593427	510.893 ms	510.893 ms	↓ 1.2	6	5	1
9.	→ Index Only Scan using ticket_flights_pkey on bookings.ticket_flights as tf (cost=0.43..0.... Index Cond: (tf.ticket_no = t.ticket_no)	9.22 ms	9.22 ms	↓ 1.34	4	3	10
10.	→ Index Scan using flights_pkey on bookings.flights as f (cost=0.29..0.31 rows=1 width=28) (a... Index Cond: (f.flight_id = tf.flight_id)	0.336 ms	0.336 ms	↑ 1	1	1	42

6. Какие сочетания имени и фамилии встречаются чаще всегодля имен и фамилий отдельно.

```
WITH p AS (
  SELECT
    left(
      passenger_name,
      position(' ' IN passenger_name)
    ) AS passenger_name
  FROM
    tickets
)
SELECT
  passenger_name,
  round(
    100.0 * cnt / sum(cnt) OVER (),
    2
  ) AS percent
FROM
  (
    SELECT
      passenger_name,
      count(*) cnt
    FROM
      p
    GROUP BY
      passenger_name
  ) t
ORDER BY
  percent DESC;
```

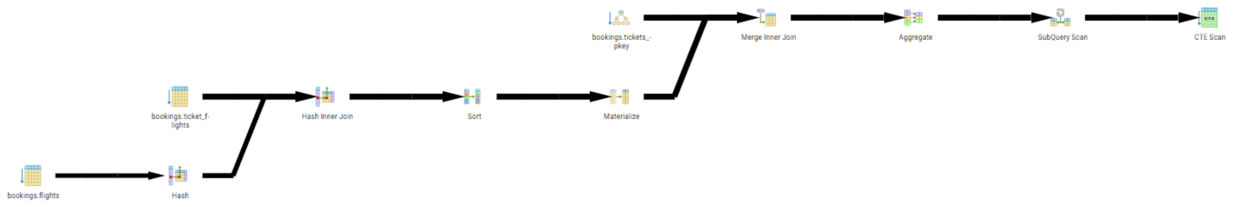


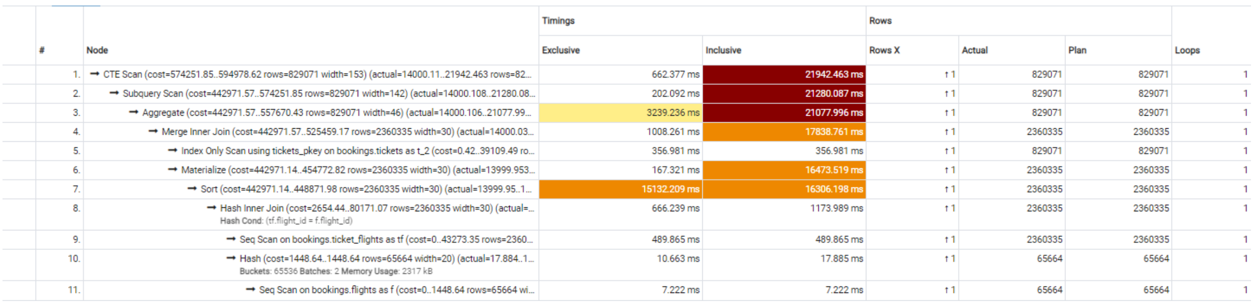


		Timings		Rows			
#	Node	Exclusive	Inclusive	Rows X	Actual	Plan	Loops
1.	→ Sort (cost=47068.98..47069.48 rows=200 width=64) (actual=768.818..768.827 rows=363 loops=1)	-494.404 ms	768.827 ms	↓ 1.82	363	200	1
2.	→ Seq Scan on bookings.tickets as tickets (cost=0..26326.06 rows=829071 width=32) (actual=0.014..49...	494.496 ms	494.496 ms	↑ 1	829071	829071	1
3.	→ Window Aggregate (cost=20726.78..20735.28 rows=200 width=64) (actual=768.562..768.735 rows=3...	0.255 ms	768.735 ms	↓ 1.82	363	200	1
4.	→ Aggregate (cost=20726.78..20728.78 rows=200 width=40) (actual=768.453..768.481 rows=363 l...	126.97 ms	768.481 ms	↓ 1.82	363	200	1
5.	→ CTE Scan (cost=0..16581.42 rows=829071 width=32) (actual=0.015..641.512 rows=829071 l...	641.512 ms	641.512 ms	↑ 1	829071	829071	1

7. аэропорты отправления и назначения для каждого билета, игнорируя стыковки, и решите, + признак билета в оба конца.

```
WITH t AS (
  SELECT
    ticket_no,
    a,
    a[1] departure,
    a[cardinality(a) ] last_arrival,
    a[cardinality(a)/ 2 + 1] middle
  FROM
    (
      SELECT
        t.ticket_no,
        array_agg(
          f.departure_airport
          ORDER BY
            f.scheduled_departure
        ) || (
          array_agg(
            f.arrival_airport
            ORDER BY
              f.scheduled_departure DESC
          )
        ) [1] AS a
      FROM
        tickets t
        JOIN ticket_flights tf ON tf.ticket_no = t.ticket_no
        JOIN flights f ON f.flight_id = tf.flight_id
      GROUP BY
        t.ticket_no
    ) t
)
SELECT
  t.ticket_no,
  t.a,
  t.departure,
  CASE WHEN t.departure = t.last_arrival THEN t.middle ELSE t.last_arrival
END arrival,
  (t.departure = t.last_arrival) return_ticket
FROM
  t;
```

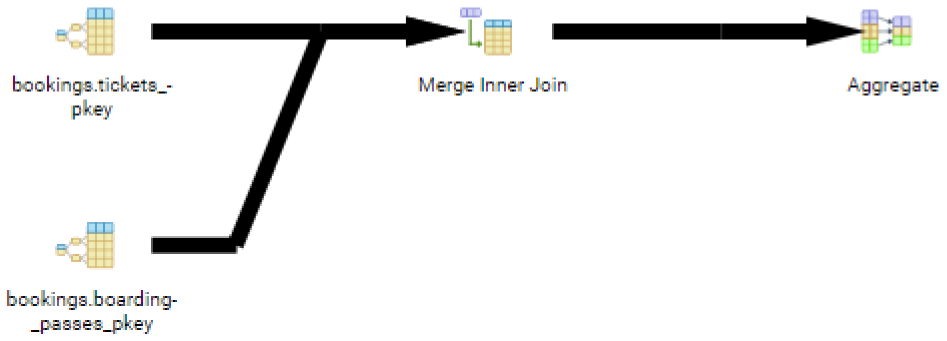


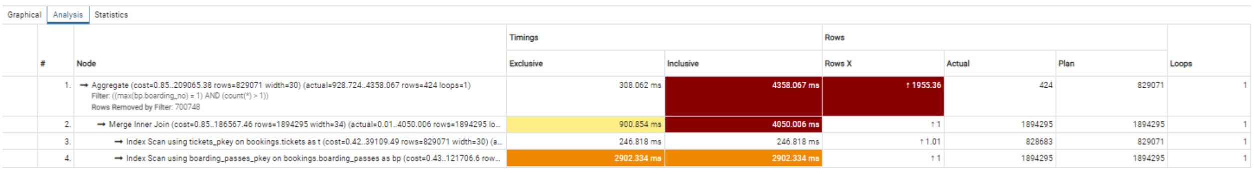


		Timings		Rows			
#	Node	Exclusive	Inclusive	Rows X	Actual	Plan	Loops
1.	→ CTE Scan (cost=574251.85..594978.62 rows=829071 width=153) (actual=14000.11..21942.463 rows=82...	662.377 ms	21942.463 ms	↑ 1	829071	829071	1
2.	→ Subquery Scan (cost=442971.57..574251.85 rows=829071 width=142) (actual=14000.108..21280.08...	202.092 ms	21280.087 ms	↑ 1	829071	829071	1
3.	→ Aggregate (cost=442971.57..557670.43 rows=829071 width=46) (actual=14000.106..21077.99...	3239.236 ms	21077.996 ms	↑ 1	829071	829071	1
4.	→ Merge Inner Join (cost=442971.57..525459.17 rows=2360335 width=30) (actual=14000.03...	1008.261 ms	17838.761 ms	↑ 1	2360335	2360335	1
5.	→ Index Only Scan using tickets_pkey on bookings.tickets as t_2 (cost=0.42..39109.49 ro...	356.981 ms	356.981 ms	↑ 1	829071	829071	1
6.	→ Materialize (cost=442971.14..454772.82 rows=2360335 width=30) (actual=13999.953...	167.321 ms	16473.519 ms	↑ 1	2360335	2360335	1
7.	→ Sort (cost=442971.14..448871.98 rows=2360335 width=30) (actual=13999.95..1...	15132.209 ms	16306.198 ms	↑ 1	2360335	2360335	1
8.	→ Hash Inner Join (cost=2654.44..80171.07 rows=2360335 width=30) (actual=... Hash Cond: (tf.flight_id = f.flight_id)	666.239 ms	1173.989 ms	↑ 1	2360335	2360335	1
9.	→ Seq Scan on bookings.ticket_flights as tf (cost=0..43273.35 rows=2360...	489.865 ms	489.865 ms	↑ 1	2360335	2360335	1
10.	→ Hash (cost=1448.64..1448.64 rows=65664 width=20) (actual=17.884..1... Buckets: 65536 Batches: 2 Memory Usage: 2317 kB	10.663 ms	17.885 ms	↑ 1	65664	65664	1
11.	→ Seq Scan on bookings.flights as f (cost=0..1448.64 rows=65664 wi...	7.222 ms	7.222 ms	↑ 1	65664	65664	1

8. Найдите наиболее дисциплинированных пассажиров, которые первыми зарегистрировались на все свои рейсы.

```
SELECT t.passenger_name,
t.ticket_no
FROM tickets t
JOIN boarding_passes bp
ON bp.ticket_no = t.ticket_no
GROUP BY t.passenger_name,
t.ticket_no
HAVING max(bp.boarding_no) = 1
AND count(*) > 1;
```



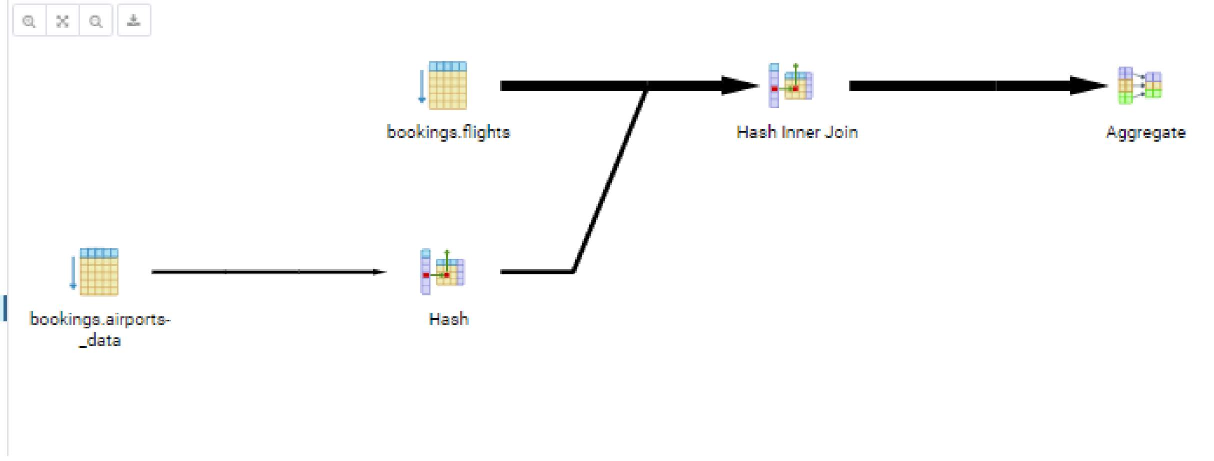


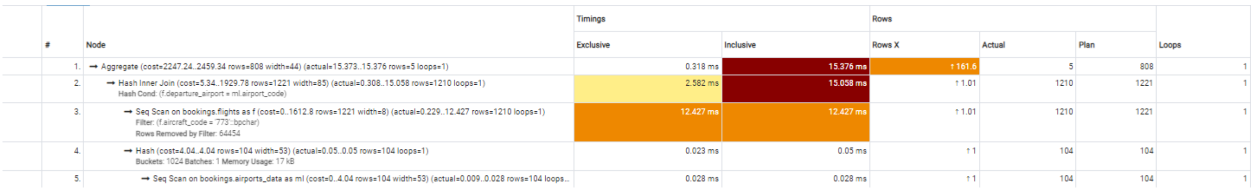
Graphical | Analysis | Statistics

		Timings		Rows			
#	Node	Exclusive	Inclusive	Rows X	Actual	Plan	Loops
1.	→ Aggregate (cost=0.85..209065.38 rows=829071 width=30) (actual=928.724..4358.067 rows=424 loops=1) Filter: ((max(bp.boarding_no) = 1) AND (count(*) > 1)) Rows Removed by Filter: 700748	308.062 ms	4358.067 ms	↑ 1955.36	424	829071	1
2.	→ Merge Inner Join (cost=0.85..186567.46 rows=1894295 width=34) (actual=0.01..4050.006 rows=1894295 lo...	900.854 ms	4050.006 ms	↑ 1	1894295	1894295	1
3.	→ Index Scan using tickets_pkey on bookings.tickets as t (cost=0.42..39109.49 rows=829071 width=30) (a...	246.818 ms	246.818 ms	↑ 1.01	828683	829071	1
4.	→ Index Scan using boarding_passes_pkey on bookings.boarding_passes as bp (cost=0.43..121706.6 row...	2902.334 ms	2902.334 ms	↑ 1	1894295	1894295	1

9. Вывести аэропорта, в которых чаще всего ротируется Боинг 777.

```
SELECT foo.city,
        count(foo.city) AS c_c
FROM
    (SELECT city,
         aircraft_code
    FROM airports AS a
    LEFT JOIN flights AS f
        ON f.departure_airport = a.airport_code
    WHERE aircraft_code = '773') AS foo
GROUP BY  city, foo.aircraft_code;
```

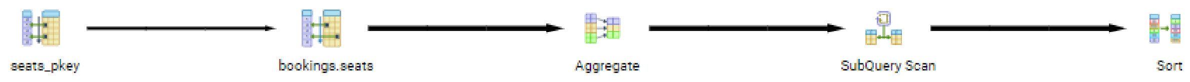


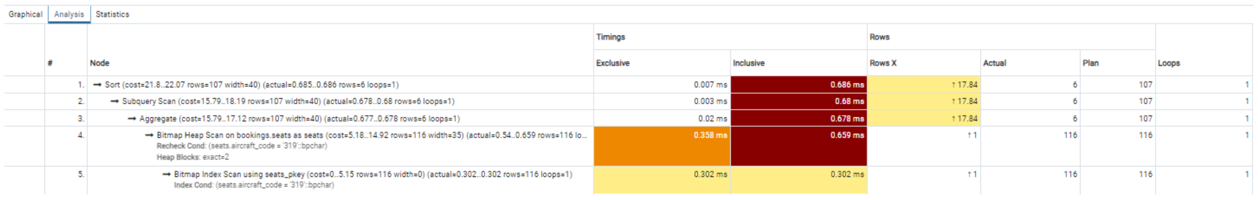


#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Aggregate (cost=2247.24..2459.34 rows=808 width=44) (actual=15.373..15.376 rows=5 loops=1)	0.318 ms	15.376 ms	↑ 161.6	5	808	1
2.	→ Hash Inner Join (cost=5.34..1929.78 rows=1221 width=85) (actual=0.308..15.058 rows=1210 loops=1) Hash Cond: (f.departure_airport = ml.airport_code)	2.582 ms	15.058 ms	↑ 1.01	1210	1221	1
3.	→ Seq Scan on bookings.flights as f (cost=0..1612.8 rows=1221 width=8) (actual=0.229..12.427 rows=1210 loops=1) Filter: (f.aircraft_code = '773'::bpchar) Rows Removed by Filter: 64454	12.427 ms	12.427 ms	↑ 1.01	1210	1221	1
4.	→ Hash (cost=4.04..4.04 rows=104 width=53) (actual=0.05..0.05 rows=104 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 17 kB	0.023 ms	0.05 ms	↑ 1	104	104	1
5.	→ Seq Scan on bookings.airports_data as ml (cost=0..4.04 rows=104 width=53) (actual=0.009..0.028 rows=104 loops...	0.028 ms	0.028 ms	↑ 1	104	104	1

10.Определить буквы с пустующими пассажирскими местами

```
SELECT foo2.final_res,
         foo2.result_W
FROM
    (SELECT foo.result_W ,
         COUNT(foo.result_W) as final_res
    FROM
        (SELECT NULLIF (regexp_replace(seat_no,
         '\D','','g'), '')::numeric AS result_D, regexp_replace(seat_no, '[0-9]', '', 'g') AS result_W
        FROM seats
        WHERE aircraft_code = '319') AS foo
        GROUP BY  result_W) AS foo2
    ORDER BY  foo2.final_res asc;
```




Graphical | Analysis | Statistics

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Sort (cost=21.8..22.07 rows=107 width=40) (actual=0.685..0.686 rows=6 loops=1)	0.007 ms	0.686 ms	↑ 17.84	6	107	1
2.	→ Subquery Scan (cost=15.79..18.19 rows=107 width=40) (actual=0.678..0.68 rows=6 loops=1)	0.003 ms	0.68 ms	↑ 17.84	6	107	1
3.	→ Aggregate (cost=15.79..17.12 rows=107 width=40) (actual=0.677..0.678 rows=6 loops=1)	0.02 ms	0.678 ms	↑ 17.84	6	107	1
4.	→ Bitmap Heap Scan on bookings.seats as seats (cost=5.18..14.92 rows=116 width=35) (actual=0.54..0.659 rows=116 lo... Recheck Cond: (seats.aircraft_code = '319'::bpchar) Heap Blocks: exact=2	0.358 ms	0.659 ms	↑ 1	116	116	1
5.	→ Bitmap Index Scan using seats_pkey (cost=0..5.15 rows=116 width=0) (actual=0.302..0.302 rows=116 loops=1) Index Cond: (seats.aircraft_code = '319'::bpchar)	0.302 ms	0.302 ms	↑ 1	116	116	1

Вывод

В ходе лабораторной работы было произведена работа с СУБД POSTGRESQL, разработано 10 запросов на русском языке, а также на SQL, проведен сравнительный анализ результатов с помощью POSTGRESQL explain.