ЗАДАНИЕ 1 (коэффициент сложности = 1)

Путем создания индексов добейтесь уменьшения времени выполнения запросов из работы 8 курса «Проектирование баз данных» [10 штук].

```sql
CREATE INDEX idx_aircrafts_model_en
ON aircrafts_data((model->>'en'));

CREATE INDEX idx_airports_name_len
ON airports_data(LENGTH(airport_name->>'ru'));

CREATE INDEX idx_aircrafts_range
ON aircrafts_data(range);

CREATE INDEX idx_airports_timezone
ON airports_data(timezone);

CREATE INDEX idx_boarding_passes_ticket_no
ON boarding_passes(ticket_no);

CREATE INDEX idx_seats_fare_conditions
ON seats(fare_conditions);

CREATE INDEX idx_ticket_flights_fare_conditions
ON ticket_flights(fare_conditions);

CREATE INDEX idx_flights_departure_city
ON flights(departure_airport);

CREATE INDEX idx_flights_arrival_city
ON flights(arrival_airport);

CREATE INDEX idx_bookings_total_amount
ON bookings(total_amount);
```

ЗАДАНИЕ 4 (коэффициент сложности = 2)

Выполните задания из раздела «10.5 Оптимизация запросов»: Моргунов Е.П., «PostgreSQL. Основы языка SQL».

1. Как вы думаете, почему при сканировании по индексу оценка стоимости ресурсов, требующихся для выдачи первых результатов, не равна нулю, хотя используется индекс, совпадающий с порядком сортировки?

```
EXPLAIN
  SELECT *
  FROM bookings
  ORDER BY book_ref;

                            QUERY PLAN
-------------------------------------------------------------
 Index Scan using bookings_pkey on bookings (cost=0.42..8511.24
   rows=262788 width=21)
(1 строка)
```

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

2. Как вы думаете, если в запросе присутствует предложение ORDER BY, и создан индекс по тем столбцам, которые фигурируют в предложении ORDER BY, то всегда ли будет использоваться этот индекс или нет? Почему? Проверьте ваши предположения с помощью команды EXPLAIN.

Нет, не всегда. Если таблица маленькая, то использование индекса лишь добавит сверху операции чтения индекса, тем самым замедляя запрос.
Например

```
demo=# EXPLAIN SELECT *
demo-# FROM aircrafts
demo-# ORDER BY aircraft_code;
                               QUERY PLAN
-------------------------------------------------------------------------
 Sort  (cost=3.51..3.53 rows=9 width=52)
   Sort Key: ml.aircraft_code
   ->  Seq Scan on aircrafts_data ml  (cost=0.00..3.36 rows=9 width=52)
(3 rows)
```

Т.к. таблица маленькая индекс не используется.

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.

```
demo=# EXPLAIN WITH RECURSIVE ranges ( min_sum, max_sum ) AS
( VALUES ( 0, 100000 )
UNION ALL
SELECT min_sum + 100000, max_sum + 100000
FROM ranges
WHERE max_sum <
( SELECT max( total_amount ) FROM bookings )
)
SELECT * FROM ranges;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 CTE Scan on ranges  (cost=254429.17..254429.79 rows=31 width=8)
   CTE ranges
     ->  Recursive Union  (cost=0.00..254429.17 rows=31 width=8)
           ->  Result  (cost=0.00..0.01 rows=1 width=8)
           ->  WorkTable Scan on ranges ranges_1  (cost=25442.59..25442.85 rows=3 width=8)
                 Filter: ((max_sum)::numeric < $2)
                 InitPlan 1 (returns $2)
                   ->  Finalize Aggregate  (cost=25442.58..25442.59 rows=1 width=32)
                         ->  Gather  (cost=25442.36..25442.57 rows=2 width=32)
                               Workers Planned: 2
                               ->  Partial Aggregate  (cost=24442.36..24442.37 rows=1 width=32)
                                     ->  Parallel Seq Scan on bookings  (cost=0.00..22243.29 rows=879629 width=6)
 JIT:
   Functions: 7
   Options: Inlining false, Optimization false, Expressions true, Deforming true
(15 rows)
```

4. Прокомментируйте следующий план, попробуйте объяснить значения всех его узлов и параметров.

```
EXPLAIN
  SELECT total_amount
  FROM bookings
  ORDER BY total_amount DESC
  LIMIT 5;

                              QUERY PLAN
-----------------------------------------------------------------------
 Limit (cost=8666.69..8666.71 rows=5 width=6)
   -> Sort (cost=8666.69..9323.66 rows=262788 width=6)
        Sort Key: total_amount DESC
        -> Seq Scan on bookings (cost=0.00..4301.88 rows=262788
             width=6)
(4 строки)
```

Сканирование последовательное, без индекса.
cost=0.00..4301.88 - оценка стоимости выполнения
rows=262788 - оценка числа строк в таблице
width=6 - размер total_amount 6 байт

Сортировка
cost=8666.69..9323.66 - оценка стоимости выполнения. Дорогая операция

LIMIT

5. В подавляющем большинстве городов только один аэропорт, но есть и такие города, в которых более одного аэропорта. Давайте их выявим.

```
EXPLAIN
  SELECT city, count( * )
  FROM airports
  GROUP BY city
  HAVING count( * ) > 1;
```

```
                          QUERY PLAN
---------------------------------------------------------------------
 HashAggregate (cost=3.82..4.83 rows=101 width=25)
   Group Key: city
   Filter: (count(*) > 1)
     -> Seq Scan on airports (cost=0.00..3.04 rows=104 width=17)
(4 строки)
```

Для подсчета числа аэропортов в городах используется последовательное сканирование и формирование хеш-таблицы (`HashAggregate`), ключом которой является столбец `city`. Затем из нее отбираются те записи, значения которых соответствуют условию

```
Filter: (count(*) > 1)
```

Как вы думаете, чем можно объяснить, что вторая оценка стоимости в параметре `cost` для узла `Seq Scan`, равная 3,04, не совпадает с первой оценкой стоимости в параметре `cost` для узла `HashAggregate`?

HashAggregate помимо чтения данных, также включает в себя затраты на создание хэш-таблицы и агрегирование данных

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.

```
demo=# EXPLAIN SELECT
        book_ref,
        book_date,
        EXTRACT(MONTH FROM book_date),
        EXTRACT(DAY FROM book_date),
        COUNT(*) OVER (
                PARTITION BY date_trunc('month', book_date) ORDER BY book_date
        )
FROM bookings
        NATURAL JOIN tickets
        NATURAL JOIN ticket_flights
WHERE flight_id = 2
ORDER BY book_date;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Sort  (cost=115043.66..115043.92 rows=104 width=95)
   Sort Key: bookings.book_date
   ->  WindowAgg  (cost=115025.47..115040.18 rows=104 width=95)
         ->  Gather Merge  (cost=115025.47..115037.58 rows=104 width=23)
               Workers Planned: 2
               ->  Sort  (cost=114025.44..114025.55 rows=43 width=23)
                     Sort Key: (date_trunc('month'::text, bookings.book_date)), bookings.book_date
                     ->  Nested Loop  (cost=0.86..114024.28 rows=43 width=23)
                           ->  Nested Loop  (cost=0.43..114003.81 rows=43 width=7)
                                 ->  Parallel Seq Scan on ticket_flights  (cost=0.00..113640.56 rows=43 width=14)
                                       Filter: (flight_id = 2)
                                 ->  Index Scan using tickets_pkey on tickets  (cost=0.43..8.45 rows=1 width=21)
                                       Index Cond: (ticket_no = ticket_flights.ticket_no)
                           ->  Index Scan using bookings_pkey on bookings  (cost=0.43..0.47 rows=1 width=15)
                                 Index Cond: (book_ref = tickets.book_ref)
 JIT:
   Functions: 14
   Options: Inlining false, Optimization false, Expressions true, Deforming true
(18 rows)
```

оконные функции выполняются после этапа получения данных из базы данных, поэтому WindowAgg занимает верхний уровень в плане выполнения запроса.

ЗАДАНИЕ 5 (коэффициент сложности = 1)

Выполните задания из раздела «Глава 12. Выполнение и оптимизация запросов»: Новиков Б.А., Горшкова Е. А., Графеева Н. Г., «Основы технологий баз данных».

Упражнение 12.1. В демонстрационной базе данных постройте индексы для поиска бронирований по имени пассажира и интервалу дат вылета.

```
demo=# CREATE INDEX idx_passenger_name ON tickets (passenger_name);

CREATE INDEX
```

```
demo=# CREATE INDEX idx_departure_date ON flights (scheduled_departure);
CREATE INDEX
demo=#
```

Упражнение 12.2. В демонстрационной базе данных постройте индексы для поиска вариантов перелета по названию пунктов отправления и прибытия и дате вылета.

```
demo=# CREATE INDEX idx_flights_departure_airport ON flights (departure_airport);
CREATE INDEX
demo=# CREATE INDEX idx_flights_arrival_airport ON flights (arrival_airport);
CREATE INDEX
```

```
demo=# CREATE INDEX idx_flights_departure_date ON flights (scheduled_departure);
CREATE INDEX
demo=# CREATE INDEX idx_flights_combined ON flights (departure_airport, arrival_airport, scheduled_departure);
CREATE INDEX
demo=#
```

Упражнение 12.3. В контексте предыдущих упражнений отключите использование индексов и сравните планы выполнения запросов с индексами и без них.

```
 Index Scan using idx_flights_combined on flights  (cost=0.42..8.45 rows=1 width=63)
   Index Cond: ((departure_airport = 'SFO'::bpchar) AND (arrival_airport = 'LAX'::bpchar) AND (scheduled_departure >= '2024-03-
16 00:00:00+03'::timestamp with time zone) AND (scheduled_departure <= '2024-03-23 00:00:00+03'::timestamp with time zone))
(2 rows)
```

```
demo=# SET enable_indexscan = False;
SET
demo=# SET enable_indexonlyscan = off;
SET enable_bitmapscan = off;
SET
SET
```

```
 Gather  (cost=1000.00..6151.95 rows=1 width=63)
   Workers Planned: 1
   ->  Parallel Seq Scan on flights  (cost=0.00..5151.85 rows=1 width=63)
         Filter: ((scheduled_departure >= '2024-03-16 00:00:00+03'::timestamp with time zone) AND (scheduled_departure <= '2024-03-23
 00:00:00+03'::timestamp with time zone) AND (departure_airport = 'SFO'::bpchar) AND (arrival_airport = 'LAX'::bpchar))
(4 rows)
```

с индексами cost гораздо меньше

Упражнение 12.4. Найдите в демонстрационной базе данных статистическую информацию.

```
demo=# SELECT AVG(range)
demo-# FROM aircrafts;
          avg
-----------------------
 5344.4444444444444444
(1 row)
```

Упражнение 12.5. Установите параметры оптимизатора таким образом, чтобы изменение порядка операций соединения никогда не производилось. Получите разные планы выполнения запросов, содержащих 3, 5, 8 и 13 операций соединения, изменяя порядок соединений вручную. Изучите и сравните оценки стоимости для различных порядков операций соединения каждого запроса.

```
demo=# SET join_collapse_limit = 1;
SET from_collapse_limit = 1;
SET geqo_threshold = 9999;
SET
SET
SET
demo=#
```

```
demo=# EXPLAIN SELECT *
FROM flights
JOIN ticket_flights ON flights.flight_id = ticket_flights.flight_id
JOIN tickets ON ticket_flights.ticket_no = tickets.ticket_no;
                                      QUERY PLAN
---------------------------------------------------------------------------------------
 Hash Join  (cost=171646.29..778546.50 rows=8391852 width=199)
   Hash Cond: (ticket_flights.ticket_no = tickets.ticket_no)
   ->  Hash Join  (cost=9767.51..302691.07 rows=8391852 width=95)
         Hash Cond: (ticket_flights.flight_id = flights.flight_id)
         ->  Seq Scan on ticket_flights  (cost=0.00..153851.52 rows=8391852 width=32)
         ->  Hash  (cost=4772.67..4772.67 rows=214867 width=63)
               ->  Seq Scan on flights  (cost=0.00..4772.67 rows=214867 width=63)
   ->  Hash  (cost=78913.57..78913.57 rows=2949857 width=104)
         ->  Seq Scan on tickets  (cost=0.00..78913.57 rows=2949857 width=104)
 JIT:
   Functions: 17
   Options: Inlining true, Optimization true, Expressions true, Deforming true
(12 rows)
```