Мамонов Антон 2ИСиП-19-1

Сканирование по битовой карте

Создайте индекс по столбцу amount таблицы перелетов (ticket_flights).

```
Текущая кодовая страница: 1251
Пароль пользователя postgres:
psql (12.5)
Введите "help", чтобы получить справку.

postgres=# CREATE INDEX ON ticket_flights(amount);
CREATE INDEX
```

Напишите запрос, находящий количество перелетов стоимостью более 180 000 руб. (менее 1% строк).

```
postgres=# EXPLAIN ANALYZE SELECT * FROM ticket_flights WHERE amount > 180000;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on ticket_flights  (cost=0.00..17.13 rows=190 width=114) (actual time=0.053..0.054 rows=0 loops=1)
   Filter: (amount > '180000'::numeric)
 Planning Time: 16.615 ms
 Execution Time: 0.131 ms
(4 строки)
```

Запретите выбранный метод доступа, снова выполните запрос и сравните время выполнения. Прав ли был оптимизатор? Сканирование по битовой карте оказывается эффективнее, в случае с небольшой выборкой.

```
ostgres=# SET enable_bitmapscan = off;
ET
ostgres=# EXPLAIN ANALYZE SELECT * FROM ticket_flights WHERE amount > 180000;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
Seq Scan on ticket_flights  (cost=0.00..17.13 rows=190 width=114) (actual time=0.023..0.023 rows=0 loops=1)
  Filter: (amount > '180000'::numeric)
Planning Time: 0.209 ms
Execution Time: 0.044 ms
4 строки)


ostgres=# RESET enable_bitmapscan;
ESET
```

```
postgres=# RESET enable_bitmapscan;
RESET
postgres=# EXPLAIN ANALYZE SELECT * FROM ticket_flights WHERE amount < 44000;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on ticket_flights  (cost=0.00..17.13 rows=190 width=114) (actual time=0.021..0.022 rows=0 loops=1)
   Filter: (amount < '44000'::numeric)
 Planning Time: 0.285 ms
 Execution Time: 0.049 ms
(4 строки)


postgres=#  SET enable_seqscan = off;
SET
postgres=# EXPLAIN ANALYZE SELECT * FROM ticket_flights WHERE amount < 44000;
                                                        QUERY PLAN

-------------------------------------------------------------------------------------------------------------
------------
 Bitmap Heap Scan on ticket_flights  (cost=5.62..18.00 rows=190 width=114) (actual time=0.029..0.031 rows=0 loops=1)
   Recheck Cond: (amount < '44000'::numeric)
   ->  Bitmap Index Scan on ticket_flights_amount_idx  (cost=0.00..5.58 rows=190 width=0) (actual time=0.024..0.024 rows
=0 loops=1)
         Index Cond: (amount < '44000'::numeric)
 Planning Time: 0.475 ms
 Execution Time: 0.209 ms
(6 строк)
```

Повторите предыдущие пункты для стоимости менее 44000 руб. (чуть более 90% строк).

Соединение вложенным циклом

Создайте индекс на таблице рейсов (flights) по аэропортам отправления (departure_airport). Найдите все рейсы из Ульяновска и проверьте план выполнения запроса.

```
postgres=# CREATE INDEX ON flights(departure_airport);
CREATE INDEX
postgres=# EXPLAIN SELECT *  FROM flights f JOIN airports a ON a.airport_code = f.departure_airport  WHERE a.city
яновск';
                              QUERY PLAN
-----------------------------------------------------------------------
 Hash Join  (cost=16.54..31.72 rows=2 width=298)
   Hash Cond: (f.departure_airport = a.airport_code)
   ->  Seq Scan on flights f  (cost=0.00..14.10 rows=410 width=170)
   ->  Hash  (cost=16.50..16.50 rows=3 width=128)
         ->  Seq Scan on airports a  (cost=0.00..16.50 rows=3 width=128)
               Filter: (city = 'Ульяновск'::text)
(6 строк)


postgres=# SELECT airport_code, airport_name FROM airports WHERE city = 'Ульяновск';
 airport_code | airport_name
--------------+--------------
 11           | Восточный
 12           | Уральский
(2 строки)
```

Соедините любые две таблицы без указания условий соединения (иными словами, выполните декартово произведение таблиц). Какой способ соединения будет выбран планировщиком? Соединение вложенным циклом - единственный способ выполнения таких соединений.

```
postgres=# EXPLAIN SELECT * FROM airports a1 CROSS JOIN airports a2;
                               QUERY PLAN
-------------------------------------------------------------------------
 Nested Loop  (cost=0.00..3411.70 rows=270400 width=256)
   ->  Seq Scan on airports a1  (cost=0.00..15.20 rows=520 width=128)
   ->  Materialize  (cost=0.00..17.80 rows=520 width=128)
         ->  Seq Scan on airports a2  (cost=0.00..15.20 rows=520 width=128)
(4 строки)


postgres=# EXPLAIN SELECT *  FROM airports a1 CROSS JOIN airports a2  WHERE a2.timezone = 'Europe/Moscow';
                               QUERY PLAN
-------------------------------------------------------------------------
 Nested Loop  (cost=0.00..51.21 rows=1560 width=256)
   ->  Seq Scan on airports a1  (cost=0.00..15.20 rows=520 width=128)
   ->  Materialize  (cost=0.00..16.52 rows=3 width=128)
         ->  Seq Scan on airports a2  (cost=0.00..16.50 rows=3 width=128)
               Filter: (timezone = 'Europe/Moscow'::text)
(5 строк)
```

Постройте таблицу расстояний между всеми аэропортами (так, чтобы каждая пара встречалась только один раз). Какой способ соединения используется в таком запросе? В данном запросе используется соединение вложенным циклом.

```
postgres=# CREATE EXTENSION earthdistance CASCADE;
ЗАМЕЧАНИЕ:  установка требуемого расширения "cube"
CREATE EXTENSION
postgres=# EXPLAIN SELECT a1.airport_code "from",  a2.airport_code "to",  a1.coordinates <@> a2.coordinates "distance,
iles"  FROM airports a1 JOIN airports a2 ON a1.airport_code > a2.airport_code;
```