**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение**

**Высшего профессионального образования**

**Московский технический университет связи и информатики**

Факультет повышения квалификации

**Лабораторная работа №2**

**по дисциплине**

**«Интеллектуальные системы»**

Выполнил:

магистрант гр. 3МПП1901

Галузинский И.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись дата

Проверил:

к.т.н., доцент каф. МКиИТ

Махров С.С.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись дата

**Москва 2020**

# Цель работы:

Целью работы является изучение возможностей профилирования и оптимизации запросов в реляционных базах данных.

# Задание:

1. Скачать выгрузку БД об авиаперелетах за <https://postgrespro.ru/education/demodb> . Необходима база за год demo-big.zip
2. Изучить цикл статей про оптимизацию и профилирование запросов:

* <https://habr.com/ru/post/203320/>
* <https://habr.com/en/post/203386/>
* <https://habr.com/en/post/203484/>

1. Придумать 1 запрос, в котором участвует несколько таблиц, и чтобы его выполнение длилось не менее 1 секунды.
2. Для запроса показать результат EXPLAIN, а затем EXPLAIN (ANALYZE).
3. Создайте индексы на поля, которые участвуют запросе и примените последовательно следующие стратегии оптимизации:

* Index Scan
* Bitmap Index Scan
* Index Only Scan

1. Исследовать, как меняется время исполнения запроса, если его исполнять несколько раз подряд? Как используется кэш? Делать запрос с указанием опции BUFFERS.
2. Предоставить отчет в direct slack <https://join.slack.com/t/intsystems/shared_invite/zt-dgvb13xu-VJ1itP7HN33PoJXcMLGb4Q>

# Ход выполнения работы:

(Содержимое отчета:

1. ER-диаграмма;
2. Указать созданный запрос, время исполнения которого не менее 1 секунды;
3. Тестовые запросы CRUD и результат выполнения каждого запроса.
4. Для запроса показать результат EXPLAIN, а затем EXPLAIN (ANALYZE).
5. Описание того, на какие поля создавали индексы и как применяли различные стартегии оптимизации? Какой получался результат?
6. Результаты исследования использования кэша.

# Ключевые вопросы:

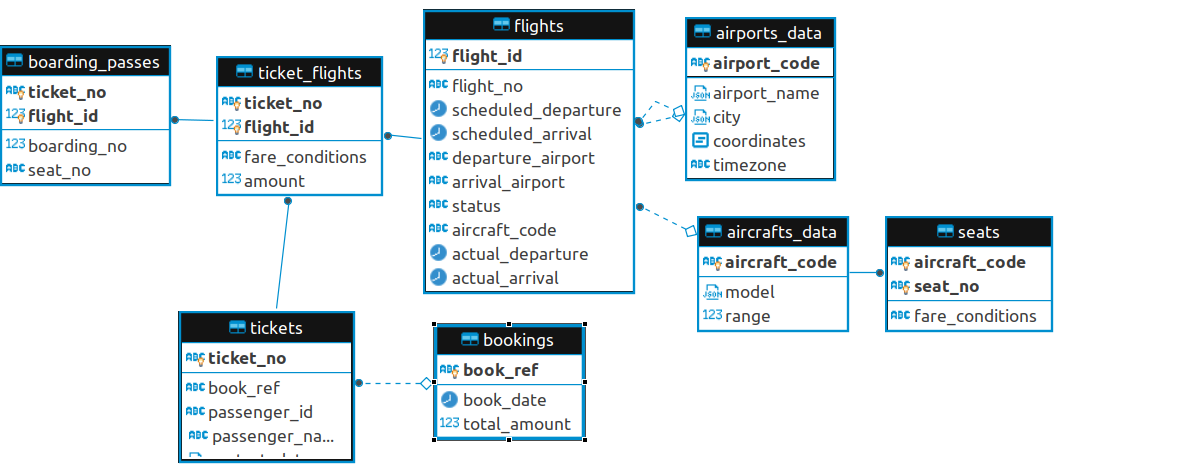
1. Что такое EXPLAIN PLAN?
2. Зачем нужны операторы EXPLAIN и ANALYZE? Можно ли их применять отдельно?
3. Зачем нужен BUFFERS?
4. Зачем нужны индексы?
5. Какие типы индексов есть в PostgreSQL?
6. Как работает кэш в PostgreSQL?

# Вывод:

В результате выполнения лабораторной работы, были изучены… появились практические навыки….. освоены….

Отчет по лабораторной работе № 2

1. ER-диаграмма

рисунок 1. ER-диаграмма

2.Указать созданный запрос, время исполнения которого не менее 1 секунды;

**explain** **analyze**

**SELECT**

demo.bookings.ticket\_flights.ticket\_no **as** Номер\_билета,

demo.bookings.ticket\_flights.flight\_id **as** Идентификатор\_рейса,

demo.bookings.ticket\_flights.fare\_conditions **as** Класс\_обслуживания,

demo.bookings.ticket\_flights.amount **as** Стоимость\_перелета,

-----------------------

demo.bookings.flights.departure\_airport **as** Аэропорт\_отправления,

demo.bookings.flights.actual\_arrival **as** дата\_прилётa,

----------------------------------------

demo.bookings.airports\_data.airport\_code **as** Код\_аэропорта,

demo.bookings.airports\_data.airport\_name **as** Название\_аэропорта,

demo.bookings.airports\_data.city **as** Город,

demo.bookings.airports\_data.timezone **as** Часовой\_пояс\_аэропорта

----------------------------------------

**from** demo.bookings.ticket\_flights

**join** demo.bookings.flights

**on** demo.bookings.flights.flight\_id =demo.bookings.ticket\_flights.flight\_id

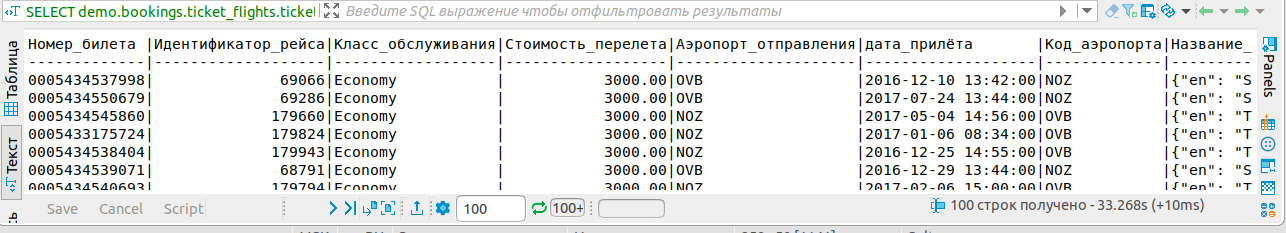
**join** demo.bookings.airports\_data

**on** demo.bookings.airports\_data.airport\_code =demo.bookings.flights.arrival\_airport

--------------------------------------------------------------------

**and** demo.bookings.ticket\_flights.amount < 15000

**order** **by** demo.bookings.ticket\_flights.amount **asc**;



3. Тестовые запросы CRUD и результат выполнения каждого запроса.

--найдём эконом места

**SELECT**

demo.bookings.boarding\_passes.ticket\_no **AS** Номер\_билета,

demo.bookings.boarding\_passes.flight\_id **AS** Идентификатор\_рейса,

demo.bookings.boarding\_passes.boarding\_no **AS** Номер\_посадочного\_талона,

demo.bookings.boarding\_passes.seat\_no **AS** Номер\_места,

---------------------------------------------------------------

demo.bookings.ticket\_flights.amount **as** Стоимость\_перелета,

demo.bookings.ticket\_flights.fare\_conditions **as** Класс\_обслуживания

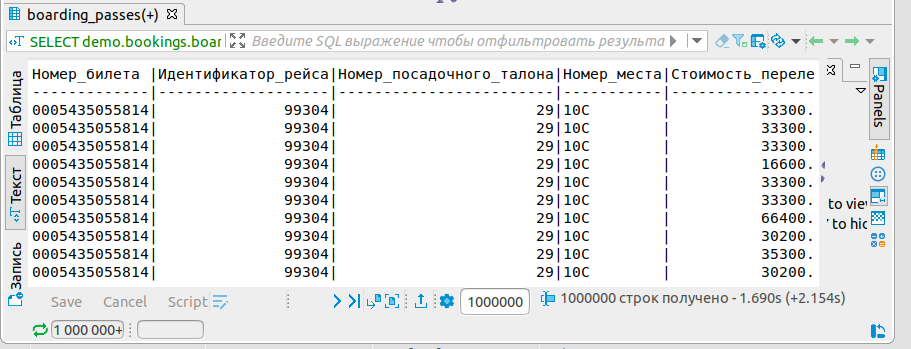
-------------------------------------------------------------

**FROM** demo.bookings.boarding\_passes

**join** demo.bookings.ticket\_flights

**on** demo.bookings.boarding\_passes.flight\_id =demo.bookings.ticket\_flights.flight\_id

**where** demo.bookings.ticket\_flights.fare\_conditions ='Economy';

1000000 строк получено - 1.690s (+2.154s)

----------найдём самолты которые смогут пролететь больше 11 000

**SELECT**

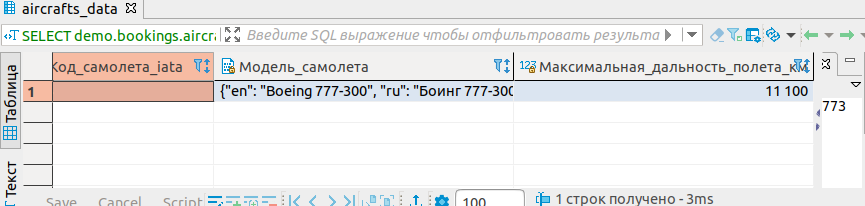
demo.bookings.aircrafts\_data.aircraft\_code **as** Код\_самолета\_IATA,

demo.bookings.aircrafts\_data.model **as** Модель\_самолета,

demo.bookings.aircrafts\_data.**range** **as** Максимальная\_дальность\_полета\_км

**from** demo.bookings.aircrafts\_data

**where** demo.bookings.aircrafts\_data.**range**>11000;



-----рейсы, найдём всё рейсы которые вылетели с задержкой

**SELECT**

demo.bookings.flights.flight\_id **as** Идентификатор\_рейса,

demo.bookings.flights.flight\_no **as** Номер\_рейса,

demo.bookings.flights.scheduled\_departure **as** Время\_вылета\_по\_расписанию,

demo.bookings.flights.scheduled\_arrival **as** Время\_прилёта\_по\_расписанию,

demo.bookings.flights.departure\_airport **as** Аэропорт\_отправления,

demo.bookings.flights.arrival\_airport **as** Аэропорт\_прибытия,

demo.bookings.flights.status **as** Статус\_рейса,

demo.bookings.flights.aircraft\_code **as** Код\_самолета\_IATA,

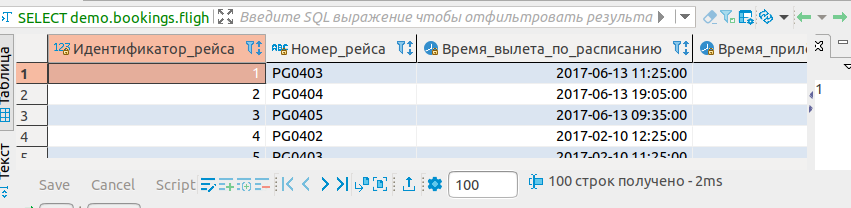
demo.bookings.flights.actual\_departure **as** Фактическое\_время\_вылета,

demo.bookings.flights.actual\_arrival **as** Фактическое\_время\_прилёта

----------------------------------

**from** demo.bookings.flights

**where** demo.bookings.flights.actual\_departure!=demo.bookings.flights.scheduled\_departure;

------найдём пасажиров, которые имеют билеты с перелётами и самолёты на которых они должны лететь, которые при этом вывелетели позднее намеченого время вылета

**SELECT**

demo.bookings.tickets.passenger\_id **as** Идентификатор\_пассажира,

demo.bookings.tickets.book\_ref **as** Номер\_бронирования,

demo.bookings.tickets.contact\_data **as** Контактные\_данные\_пассажира,

demo.bookings.tickets.passenger\_name **as** Имя\_пассажира,

demo.bookings.tickets.ticket\_no **as** Номер\_билета,

----------------------------------

demo.bookings.ticket\_flights.ticket\_no **as** Номер\_билета,

demo.bookings.ticket\_flights.flight\_id **as** Идентификатор\_рейса,

demo.bookings.ticket\_flights.fare\_conditions **as** Номер\_посадочного\_талона,

demo.bookings.ticket\_flights.amount **as** Номер\_места,

-----------------------------------

demo.bookings.flights.scheduled\_departure **as** Время\_вылета\_по\_расписанию,

demo.bookings.flights.scheduled\_arrival **as** Время\_прилёта\_по\_расписанию,

demo.bookings.flights.actual\_departure **as** Фактическое\_время\_вылета,

demo.bookings.flights.actual\_arrival **as** Фактическое\_время\_прилёта

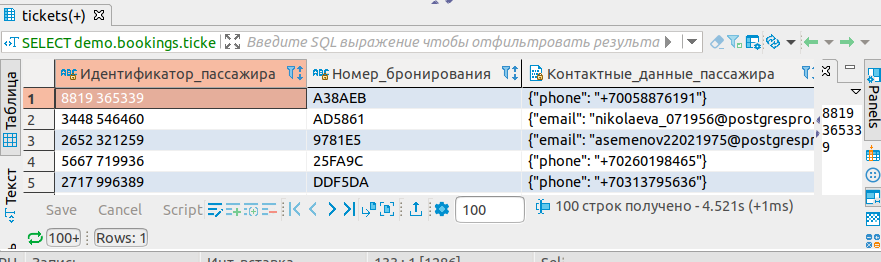
----------------------------------

**from** demo.bookings.tickets

**join** demo.bookings.ticket\_flights **on** demo.bookings.tickets.ticket\_no =demo.bookings.ticket\_flights.ticket\_no

**join** demo.bookings.flights **on** demo.bookings.flights.flight\_id =demo.bookings.ticket\_flights.flight\_id

**where** demo.bookings.flights.actual\_departure > demo.bookings.flights.scheduled\_departure;

---найдём пассажиров которые заплатил больше 1 000 000 за перелёт

**SELECT**

demo.bookings.bookings.book\_ref **as** Номер\_бронирования,

demo.bookings.bookings.book\_date **as** Дата\_бронирования,

demo.bookings.bookings.total\_amount **as** Полная\_сумма\_бронирования,

-----------------------------------

demo.bookings.tickets.passenger\_id **as** Идентификатор\_пассажира,

demo.bookings.tickets.book\_ref **as** Номер\_бронирования,

demo.bookings.tickets.contact\_data **as** Контактные\_данные\_пассажира,

demo.bookings.tickets.passenger\_name **as** Имя\_пассажира,

demo.bookings.tickets.ticket\_no **as** Номер\_билета

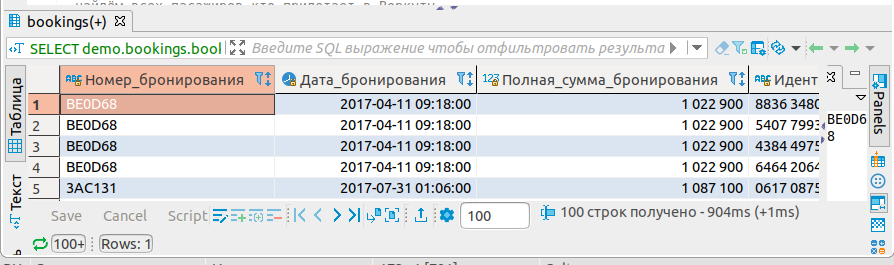
-----------------------

**from** demo.bookings.bookings

**join** demo.bookings.tickets

**on** demo.bookings.bookings.book\_ref =demo.bookings.tickets.book\_ref

**where** demo.bookings.bookings.total\_amount>1000000;



--найдём всех пасажиров кто прилетает в Воркуту

**SELECT**

demo.bookings.boarding\_passes.flight\_id **as** Идентификатор\_рейса,

demo.bookings.boarding\_passes.boarding\_no **as** Номер\_билета,

demo.bookings.boarding\_passes.seat\_no **as** Номер\_места,

demo.bookings.boarding\_passes.ticket\_no **as** Номер\_посадочного\_талона,

-----------------------------------

demo.bookings.ticket\_flights.ticket\_no **as** Номер\_билета,

demo.bookings.ticket\_flights.flight\_id **as** Идентификатор\_рейса,

demo.bookings.ticket\_flights.fare\_conditions **as** Номер\_посадочного\_талона,

demo.bookings.ticket\_flights.amount **as** Номер\_места,

-----------------------

demo.bookings.flights.departure\_airport **as** Аэропорт\_отправления,

demo.bookings.flights.arrival\_airport **as** Аэропорт\_прибытия,

----------------------------------------

demo.bookings.airports\_data.airport\_name **as** Название\_аэропорта,

demo.bookings.airports\_data.city **as** Город

----------------------------------------

**from** demo.bookings.boarding\_passes

**join** demo.bookings.ticket\_flights

**on** demo.bookings.boarding\_passes.ticket\_no =demo.bookings.ticket\_flights.ticket\_no

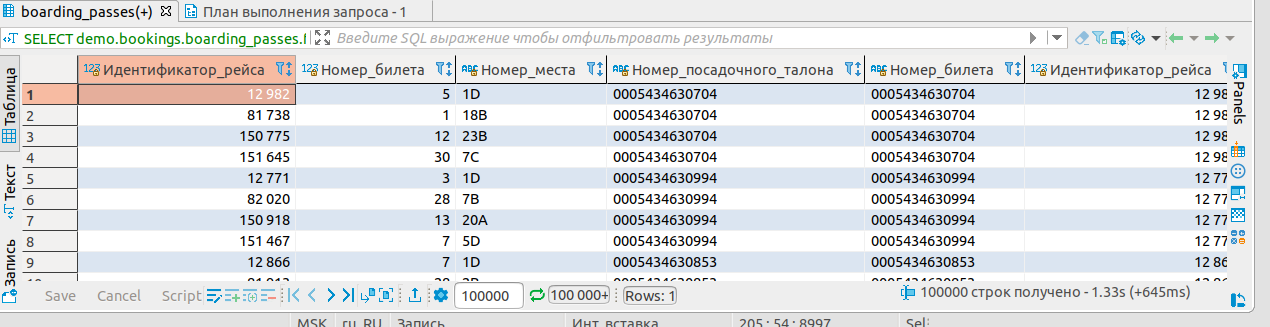
**join** demo.bookings.flights

**on** demo.bookings.flights.flight\_id =demo.bookings.ticket\_flights.flight\_id

**join** demo.bookings.airports\_data

**on** demo.bookings.airports\_data.airport\_code =demo.bookings.flights.arrival\_airport

**where** demo.bookings.airports\_data.airport\_code='VKT';

4. Для запроса показать результат EXPLAIN, а затем EXPLAIN (ANALYZE).

**SELECT**

demo.bookings.ticket\_flights.ticket\_no **as** Номер\_билета,

demo.bookings.ticket\_flights.flight\_id **as** Идентификатор\_рейса,

demo.bookings.ticket\_flights.fare\_conditions **as** Класс\_обслуживания,

demo.bookings.ticket\_flights.amount **as** Стоимость\_перелета,

-----------------------

demo.bookings.flights.departure\_airport **as** Аэропорт\_отправления,

demo.bookings.flights.actual\_arrival **as** дата\_прилётa,

----------------------------------------

demo.bookings.airports\_data.airport\_code **as** Код\_аэропорта,

demo.bookings.airports\_data.airport\_name **as** Название\_аэропорта,

demo.bookings.airports\_data.city **as** Город,

demo.bookings.airports\_data.timezone **as** Часовой\_пояс\_аэропорта

----------------------------------------

**from** demo.bookings.ticket\_flights

**join** demo.bookings.flights

**on** demo.bookings.flights.flight\_id =demo.bookings.ticket\_flights.flight\_id

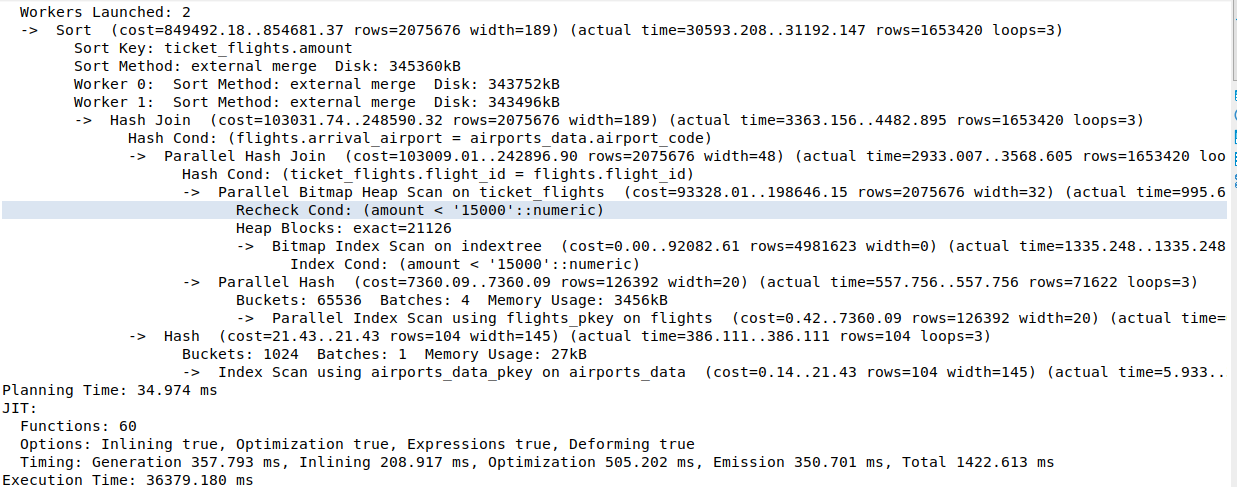
**join** demo.bookings.airports\_data

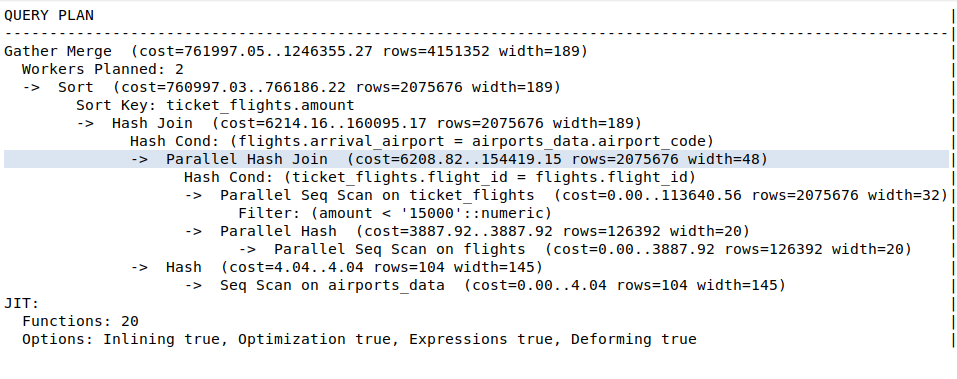
**on** demo.bookings.airports\_data.airport\_code =demo.bookings.flights.arrival\_airport

--------------------------------------------------------------------

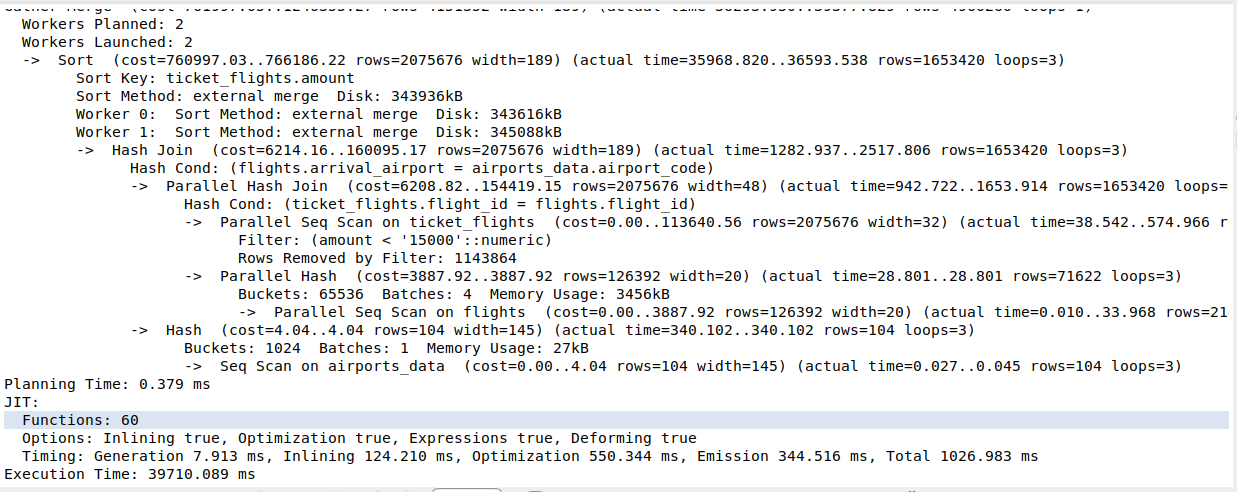
**and** demo.bookings.ticket\_flights.amount < 15000

**order** **by** demo.bookings.ticket\_flights.amount **asc**;

запрос без использования индексов **explain**:



запрос без использования индексов **explain (analyze**):



5. Описание того, на какие поля создавали индексы и как применяли различные стартегии оптимизации? Какой получался результат?

Используем следующий запрос:

**SELECT**

demo.bookings.ticket\_flights.ticket\_no **as** Номер\_билета,

demo.bookings.ticket\_flights.flight\_id **as** Идентификатор\_рейса,

demo.bookings.ticket\_flights.fare\_conditions **as** Класс\_обслуживания,

demo.bookings.ticket\_flights.amount **as** Стоимость\_перелета,

-----------------------

demo.bookings.flights.departure\_airport **as** Аэропорт\_отправления,

demo.bookings.flights.actual\_arrival **as** дата\_прилёте,

----------------------------------------

demo.bookings.airports\_data.airport\_code **as** Код\_аэропорта,

demo.bookings.airports\_data.airport\_name **as** Название\_аэропорта,

demo.bookings.airports\_data.city **as** Город,

demo.bookings.airports\_data.timezone **as** Часовой\_пояс\_аэропорта

----------------------------------------

**from** demo.bookings.ticket\_flights

**join** demo.bookings.flights

**on** demo.bookings.flights.flight\_id =demo.bookings.ticket\_flights.flight\_id

**join** demo.bookings.airports\_data

**on** demo.bookings.airports\_data.airport\_code =demo.bookings.flights.arrival\_airport

--------------------------------------------------------------------

**and** demo.bookings.ticket\_flights.amount < 15000

**order** **by** demo.bookings.ticket\_flights.amount **asc**;

--индексы

**create** **index** indexOne **On** demo.bookings.ticket\_flights (flight\_id);

**create** **index** indexTwo **On** demo.bookings.flights (arrival\_airport);

**create** **index** indexTree **On** demo.bookings.ticket\_flights (amount);

Для таблицы ticket\_flights был создан индекс indexOne по полю flight\_id для оптимизации join с таблицей flights, где поле light\_id является первичным ключом.

Для таблицы flights был создан индекс indexOne по полю arrival\_airport для оптимизации join с таблицей airports\_data, где поле airport\_code является первичным ключом.

Индекс indexTree был создан для демонстрации оптимизации запросов к таблице ticket\_flights.

Результат назначение индексов узнаём с помощью команд **explain** и **explain (analyze**):

**explain**:

QUERY PLAN |

Gather Merge (cost=822114.20..1306472.42 rows=4151352 width=173)

Workers Planned: 2 |

-> Sort (cost=821114.18..826303.37 rows=2075676 width=173)

Sort Key: ticket\_flights.amount |

-> Hash Join (cost=103031.74..248590.32 rows=2075676 width=173)

Hash Cond: (flights.arrival\_airport = airports\_data.airport\_code) |

-> Parallel Hash Join (cost=103009.01..242896.90 rows=2075676 width=48)

Hash Cond: (ticket\_flights.flight\_id = flights.flight\_id) |

-> Parallel Bitmap Heap Scan on ticket\_flights (cost=93328.01..198646.15 rows=2075676 width=32) |

Recheck Cond: (amount < '15000'::numeric) |

-> Bitmap Index Scan on indextree (cost=0.00..92082.61 rows=4981623 width=0)

Index Cond: (amount < '15000'::numeric) |

-> Parallel Hash (cost=7360.09..7360.09 rows=126392 width=20)

-> Parallel Index Scan using flights\_pkey on flights (cost=0.42..7360.09 rows=126392 width=20)|

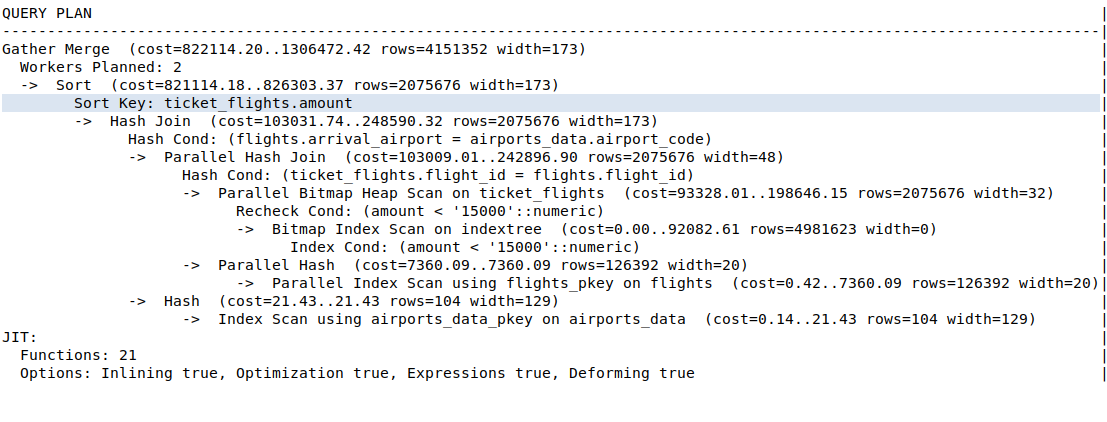
-> Hash (cost=21.43..21.43 rows=104 width=129) |

-> Index Scan using airports\_data\_pkey on airports\_data (cost=0.14..21.43 rows=104 width=129)

JIT: |

Functions: 21 |

Options: Inlining true, Optimization true, Expressions true, Deforming true |

****

**explain (analyze**)

QUERY PLAN

Gather Merge (cost=822114.20..1306472.42 rows=4151352 width=173) (actual time=24539.307..27482.328 rows=4960260 loops=1) |

Workers Planned: 2

Workers Launched: 2

-> Sort (cost=821114.18..826303.37 rows=2075676 width=173) (actual time=24084.285..24668.828 rows=1653420 loops=3) |

Sort Key: ticket\_flights.amount

Sort Method: external merge Disk: 313872kB

Worker 0: Sort Method: external merge Disk: 313968kB

Worker 1: Sort Method: external merge Disk: 311592kB

-> Hash Join (cost=103031.74..248590.32 rows=2075676 width=173) (actual time=1505.958..2663.142 rows=1653420 loops=3) |

Hash Cond: (flights.arrival\_airport = airports\_data.airport\_code)

-> Parallel Hash Join (cost=103009.01..242896.90 rows=2075676 width=48) (actual time=1167.992..1827.397 rows=1653420 loops=3) |

Hash Cond: (ticket\_flights.flight\_id = flights.flight\_id)

-> Parallel Bitmap Heap Scan on ticket\_flights (cost=93328.01..198646.15 rows=2075676 width=32) (actual time=495.077..790.309 rows=1653420 loops=3)|

Recheck Cond: (amount < '15000'::numeric)

Heap Blocks: exact=21164

-> Bitmap Index Scan on indextree (cost=0.00..92082.61 rows=4981623 width=0) (actual time=481.763..481.763 rows=4960260 loops=1) |

Index Cond: (amount < '15000'::numeric)

-> Parallel Hash (cost=7360.09..7360.09 rows=126392 width=20) (actual time=35.855..35.856 rows=71622 loops=3) |

Buckets: 65536 Batches: 4 Memory Usage: 3456kB

-> Parallel Index Scan using flights\_pkey on flights (cost=0.42..7360.09 rows=126392 width=20) (actual time=0.043..18.246 rows=71622 loops=3)|

-> Hash (cost=21.43..21.43 rows=104 width=129) (actual time=337.897..337.897 rows=104 loops=3) |

Buckets: 1024 Batches: 1 Memory Usage: 25kB

-> Index Scan using airports\_data\_pkey on airports\_data (cost=0.14..21.43 rows=104 width=129) (actual time=337.802..337.857 rows=104 loops=3) |

Planning Time: 93.394 ms

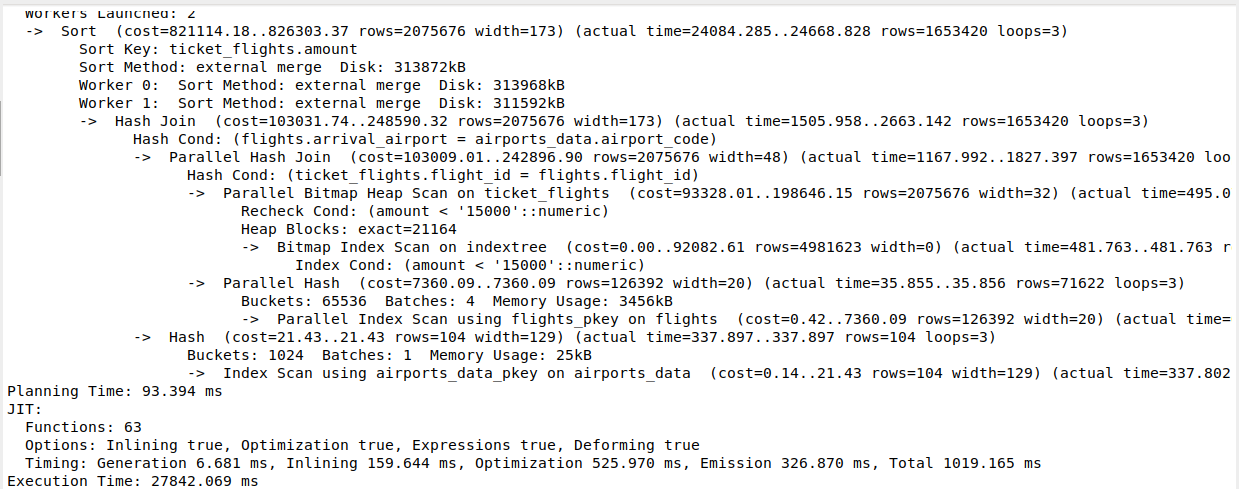
JIT:

Functions: 63

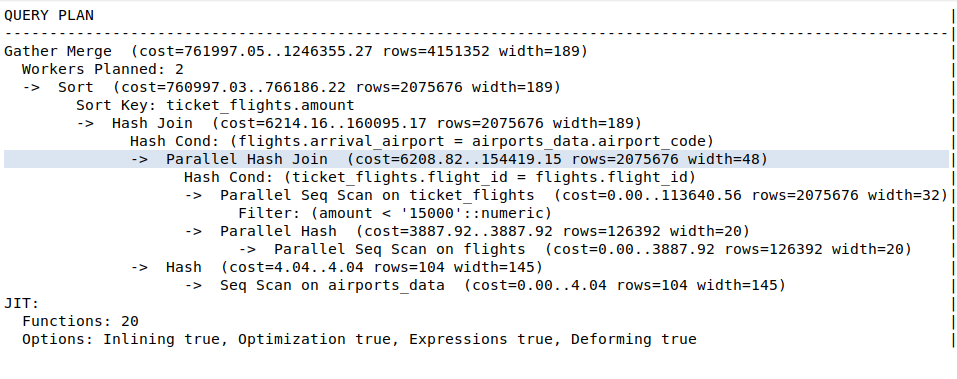
Options: Inlining true, Optimization true, Expressions true, Deforming true

Timing: Generation 6.681 ms, Inlining 159.644 ms, Optimization 525.970 ms, Emission 326.870 ms, Total 1019.165 ms |

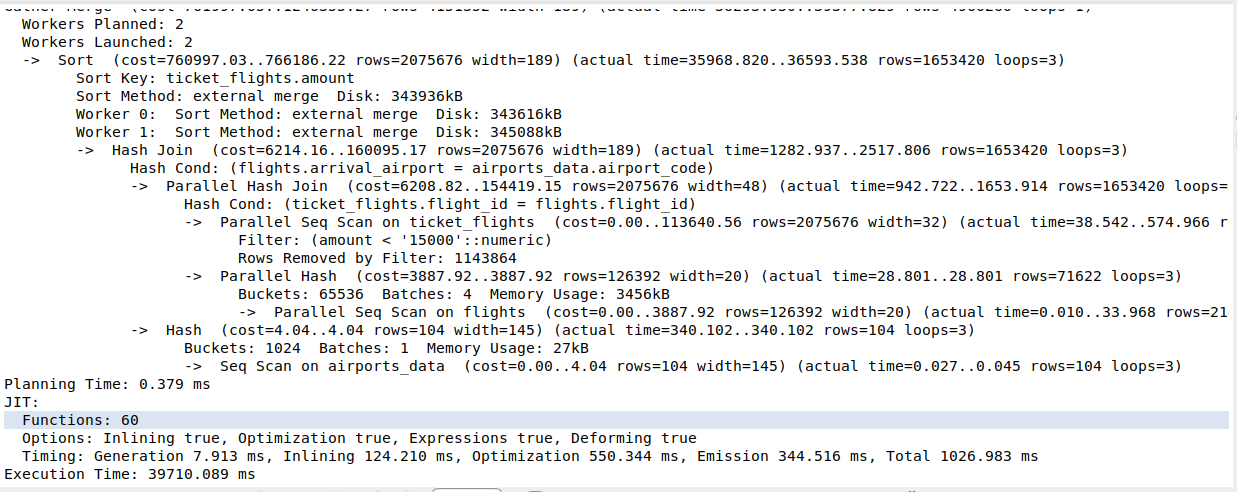
Execution Time: 27842.069 ms |



запрос без использования индексов **explain**:



запрос без использования индексов **explain (analyze**):



При поиске значения из таблицы БД сама решает каким образом лучше это сделать.

Согласно заданию надо использовать

**Index Scan**- используется индекс для условий WHERE, читает таблицу при отборе строк.

При индексном просмотре метод доступа возвращает значения TID по одному, до тех пор, пока подходящие строки не закончатся. Механизм индексирования по очереди обращается к тем страницам таблицы, на которые указывают TID, получает версию строки, проверяет ее видимость в соответствии с правилами многоверсионности, и возвращает полученные данные

Для использования данного метода, надо выбрать всё поля, даже те на которые индексы не были назначены, а выводимый результат **не должен** содержать большое кол-во строк, в противном случае будет использован другой метод выдачи запроса пользователю(Bitmap Index Scan )

**explain** **analyze** -----Index Scan.

**SELECT** \*

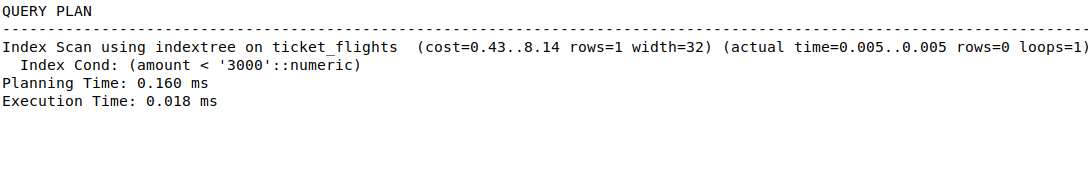
----------------------------------------

**from** demo.bookings.ticket\_flights

--------------------------------------------------------------------

**where** demo.bookings.ticket\_flights.amount < 3000

**order** **by** demo.bookings.ticket\_flights.amount **asc**;

**Bitmap Index Scan** -сначала Index Scan, затем контроль выборки по таблице. Эффективно для большого количества строк.

Сначала метод доступа возвращает все TID, соответствующие условию (узел Bitmap Index Scan), и по ним строится битовая карта версий строк. Затем версии строк читаются из таблицы (Bitmap Heap Scan)— при этом каждая страница будет прочитана только один раз.

Для использования данного метода, надо выбрать всё поля, даже те на которые индексы не были назначены, а выводимый результат **должен** содержать большое кол-во строк, в противном случае будет использован другой метод выдачи запроса пользователю(Bitmap Index Scan )

**explain** **analyze** -----Bitmap Index Scan on indextree

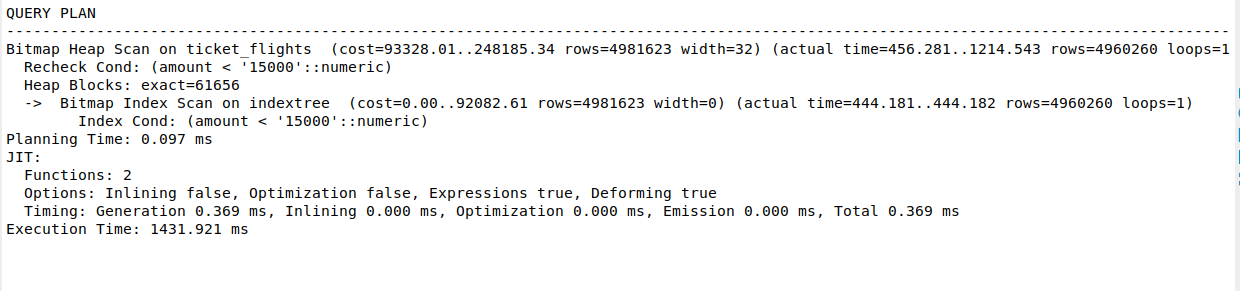
**SELECT**

\*

----------------------------------------

**from** demo.bookings.ticket\_flights

--------------------------------------------------------------------

**where** demo.bookings.ticket\_flights.amount < 15000

**Index Only Scan** -самый быстрый. Читается только индекс.

Как правило, основная задача метода доступа— вернуть идентификаторы подходящих строк таблицы, чтобы механизм индексирования мог прочитать из них необходимые данные. Но что, если индекс уже содержит все необходимые для запроса данные? Такой индекс называетсяпокрывающим(covering), и в этом случае оптимизатор может применитьисключительно индексное сканирование(Index Only Scan):

Для использования данного метода, надо выбрать те поля на которые индексы были назначены, а выводимый результат **не должен** содержать большое кол-во строк, в противном случае будет использован другой метод выдачи запроса пользователю(Bitmap Index Scan )

**explain** **analyze** -----Index Only Scan using

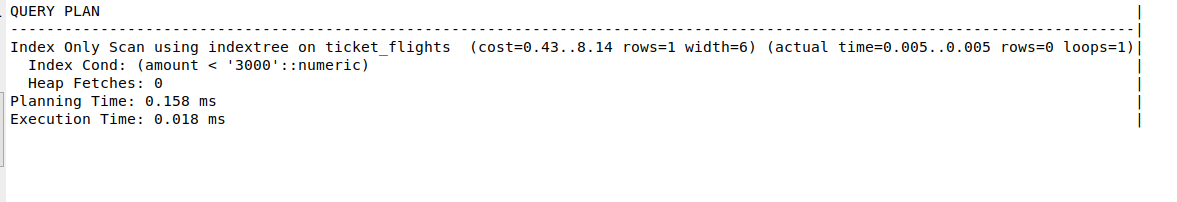
**SELECT** demo.bookings.ticket\_flights.amount

----------------------------------------

**from** demo.bookings.ticket\_flights

--------------------------------------------------------------------

**where** demo.bookings.ticket\_flights.amount < 3000

**вывод**: благодаря использованию индексов, получилось увеличить производительность запроса(скорость выдачи данных)

6. Результаты исследования использования кэша.

**EXPLAIN** (**ANALYZE**,BUFFERS)

**select** \*

**from** demo.bookings.ticket\_flights

**Результат выполнения 1 запроса:**

QUERY PLAN

Seq Scan on ticket\_flights (cost=0.00..153851.52 rows=8391852 width=32) (actual time=0.051..896.942 rows=8391852 loops=1)|

Buffers: shared hit=32 read=69901

Planning Time: 0.058 ms

Execution Time: 1350.187 ms

**Результат выполнения 2 запроса:**

QUERY PLAN

Seq Scan on ticket\_flights (cost=0.00..153851.52 rows=8391852 width=32) (actual time=0.040..953.955 rows=8391852 loops=1)|

Buffers: shared hit=64 read=69869

Planning Time: 0.041 ms

Execution Time: 1424.993 ms

**Результат выполнения 3 запроса:**

QUERY PLAN

Seq Scan on ticket\_flights (cost=0.00..153851.52 rows=8391852 width=32) (actual time=0.059..887.738 rows=8391852 loops=1)|

Buffers: shared hit=96 read=69837

Planning Time: 0.047 ms

Execution Time: 1337.109 ms

**4 запрос**

QUERY PLAN

Seq Scan on ticket\_flights (cost=0.00..153851.52 rows=8391852 width=32) (actual time=0.048..929.875 rows=8391852 loops=1)|

Buffers: shared hit=128 read=69805

Planning Time: 0.043 ms

Execution Time: 1393.798 ms

**Вывод(6 пункт):** в первом запросе из кэша Postres было прочитано 32 блоков, о чем говорит запись планировщика:

Buffers: shared hit=32 . С каждым последующем запросом количество блоков в кэше(оперативной памяти) увеличивается и соответственно планировщик ядра Postgres прочитывает все больше данных из оперативной памяти(кэша), так как это собственно увеличивает производительность.

В четвёртом запросе количество считываемых блоков продолжалось увеличиваться. Это говорит о том, что ещё не вся таблица находится в оперативной памяти.Таблица будет загружена вся в ОЗУ компьютера, когда количество считываемых блоков не будет увеличиваться, тогда это будет означать, что эта таблица будет в хранится в ОЗУ. К сожалению мне не удалось загрузить таблицу в ОЗУ.

# Вывод по лабораторной работе:

В результате выполнения лабораторной работы, были изучены механизмы планировщика ядра Postgres. Появились практические навыки работы с командой EXPLAIN и ее вариациями, а также созданием и использованием индексов Postgres. Так в ходе лабораторной работы были дополнены знания об использовании контейнеров, загрузки в него sql скрипта с помощью dockerFile и разворачивание базы данных внутри контейнера.

Ответы на вопросы.

1. EXPLAIN PLAN — это план ядра Postgres на выполнение запроса.

2. Команда EXPLAIN выводит «ожидания» планировщика для выполнения запроса. Вариация EXPLAIN(ANALYZE) считывает определённое количество строк таблицы, выбранных случайным образом и таким орбазом собирается статистика значений по каждой из колонок таблицы. Применять отдельно можно, ANALYZE используется для обновления плана запроса таблицы.

3. Команда BUFFERS позволят получить информацию об использовании кэша Postgres.

4. Индексы - это специальные структуры в базах данных, которые позволяют ускорить поиск и сортировку по определенному полю или набору полей в таблице, а также используются для обеспечения уникальности данных.

5. Типы индексов Postgres: B-дерево, хеш, GiST, SP-GiST, GIN и BRIN. Для разных типов индексов применяются разные алгоритмы, ориентированные на определённые типы запросов. По умолчанию команда CREATE INDEX создаёт индексы типа B-дерево, эффективные в большинстве случаев.

6. Кэш по сути является выделенным местом в оперативной памяти. После каждого запроса ядро Postgres постепенно помещает некоторое количество данных(блоков определенной размерности) в кэш(оперативную память) откуда чтение происходит гораздо быстрее.