**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. 3mpp1901

Иванова Е.В.

Проверил: к.т.н., доцент каф. МКиИТ

Махров С.С.

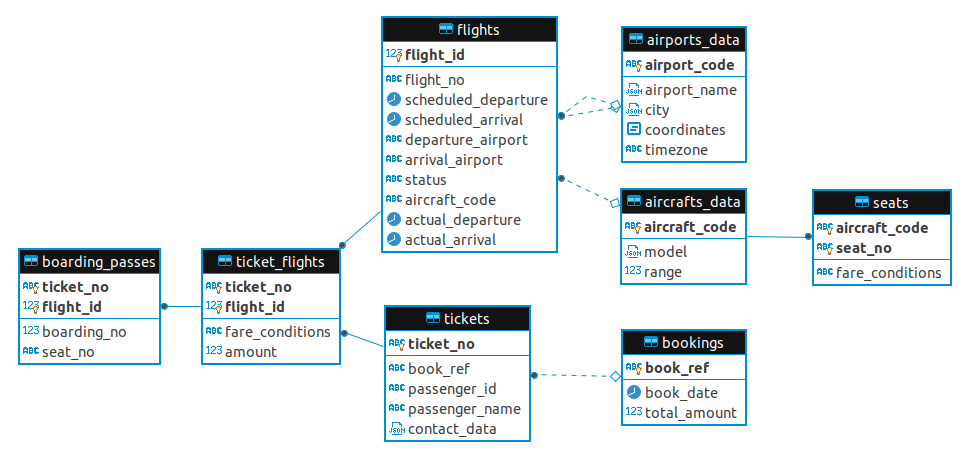
**Москва 2020**

Цель работы:

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

Отчет по лабораторной работе № 2

1. ER-диаграмма



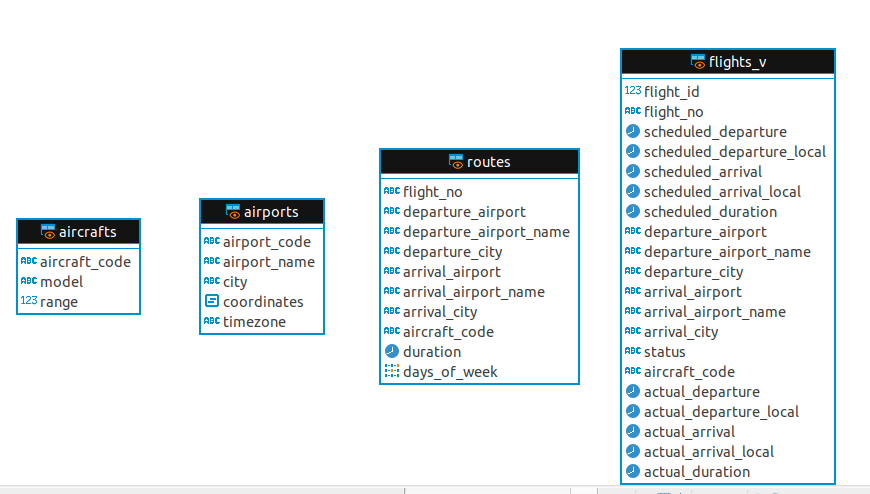


Рис. 1 - ER диаграмма

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

**select**

f.flight\_no №\_рейса

,f.departure\_airport А\_о

,f.arrival\_airport А\_п

,t.book\_ref №\_брони

,**date**(b.book\_date) **as** Дата\_брони

,t.passenger\_name ФИ\_пассажира

,tfl.ticket\_no №\_билета

**from** demo.bookings.tickets t

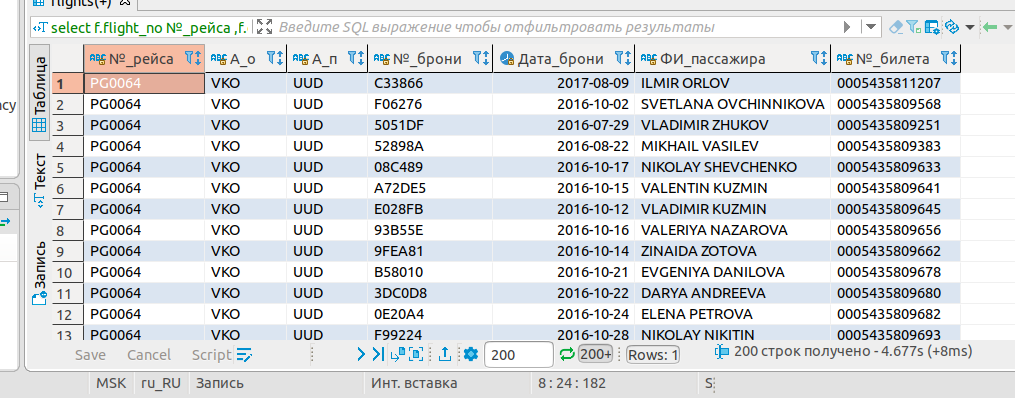
**inner** **join** demo.bookings.bookings b **on** b.book\_ref = t.book\_ref

**inner** **join** demo.bookings.ticket\_flights tfl **on** tfl.ticket\_no = t.ticket\_no

**inner** **join** demo.bookings.flights f **on** f.flight\_id = tfl.flight\_id

**where** tfl.amount > 90000

**order** **by** f.flight\_no **asc**



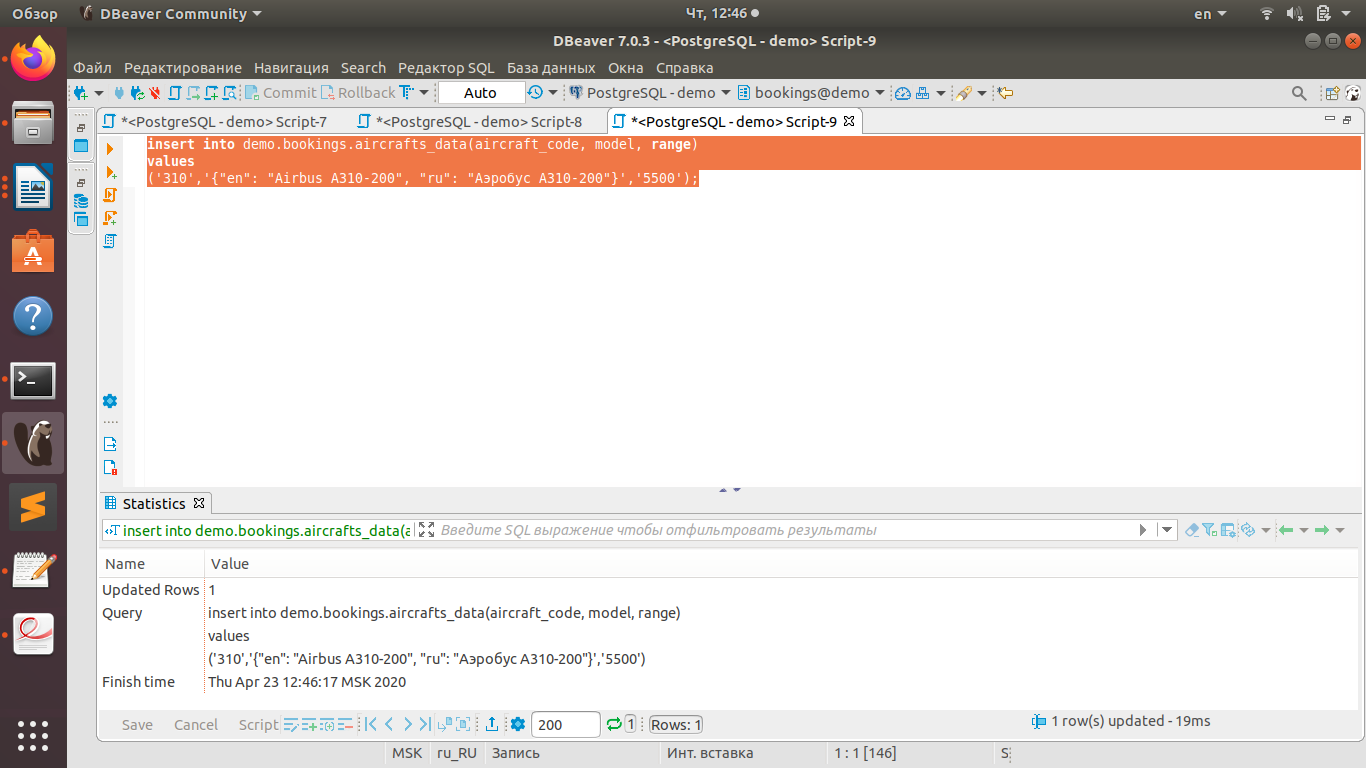
1. Тестовые запросы CRUD и результат выполнения каждого запроса

Добавим новую строку: код самолета, модель, дальность полета в таблицу самолёты:

**insert** **into** demo.bookings.aircrafts\_data(aircraft\_code, model, **range**)

**values**

('310','{"en": "Airbus A310-200", "ru": "Аэробус A310-200"}','5500');

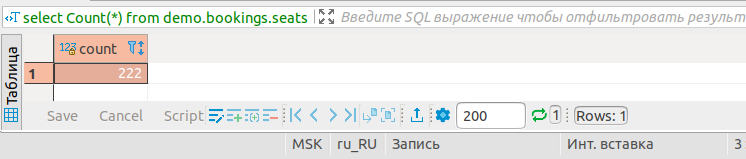


Найдём количество записей в таблице места, где код самолета = 763:

**select** **Count**(\*)

**from** demo.bookings.seats

**where** aircraft\_code = '763';



Покажем номер рейса, аэропорт отправления и прибытия из таблицы рейсы, и номер места и номер билета из таблицы посадочные талоны, где номер рейса в таблице посадочные талоны не пустой, после этого результат отсортируем по возрастанию номера рейса таблицы рейсы:

**select**

f.flight\_no

,f.departure\_airport

,f.arrival\_airport

,bp.seat\_no

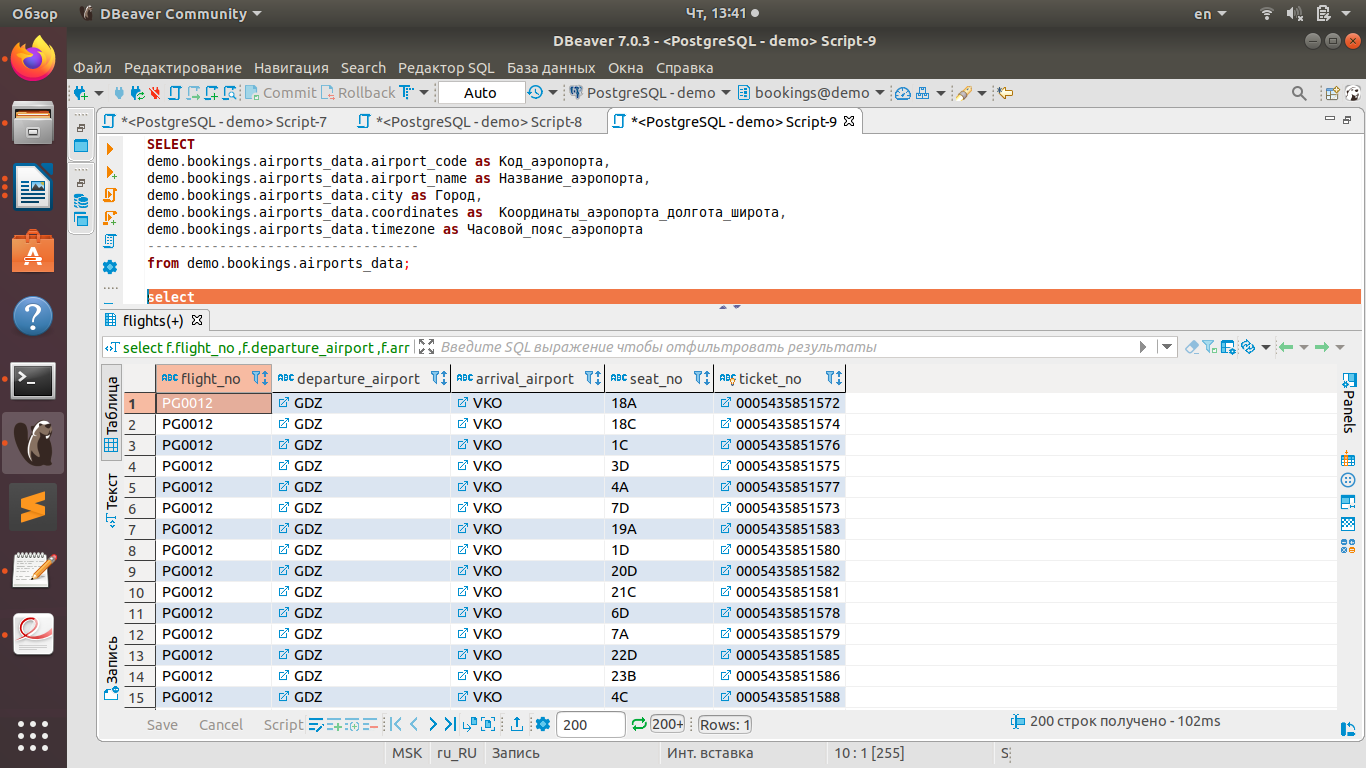
,bp.ticket\_no

**from** demo.bookings.flights f

**left** **outer** **join** demo.bookings.boarding\_passes bp **on** bp.flight\_id = f.flight\_id

**where** bp.ticket\_no **is** **not** **null**

**order** **by** f.flight\_no **asc**

****

Обновим таблицу бронирования записью полной суммы бронирования 5000, где дата бронирования '2017-01-01' и полная сумма бронирования между 6000 и 10000:

**update** demo.bookings.bookings

**set** total\_amount = 5000

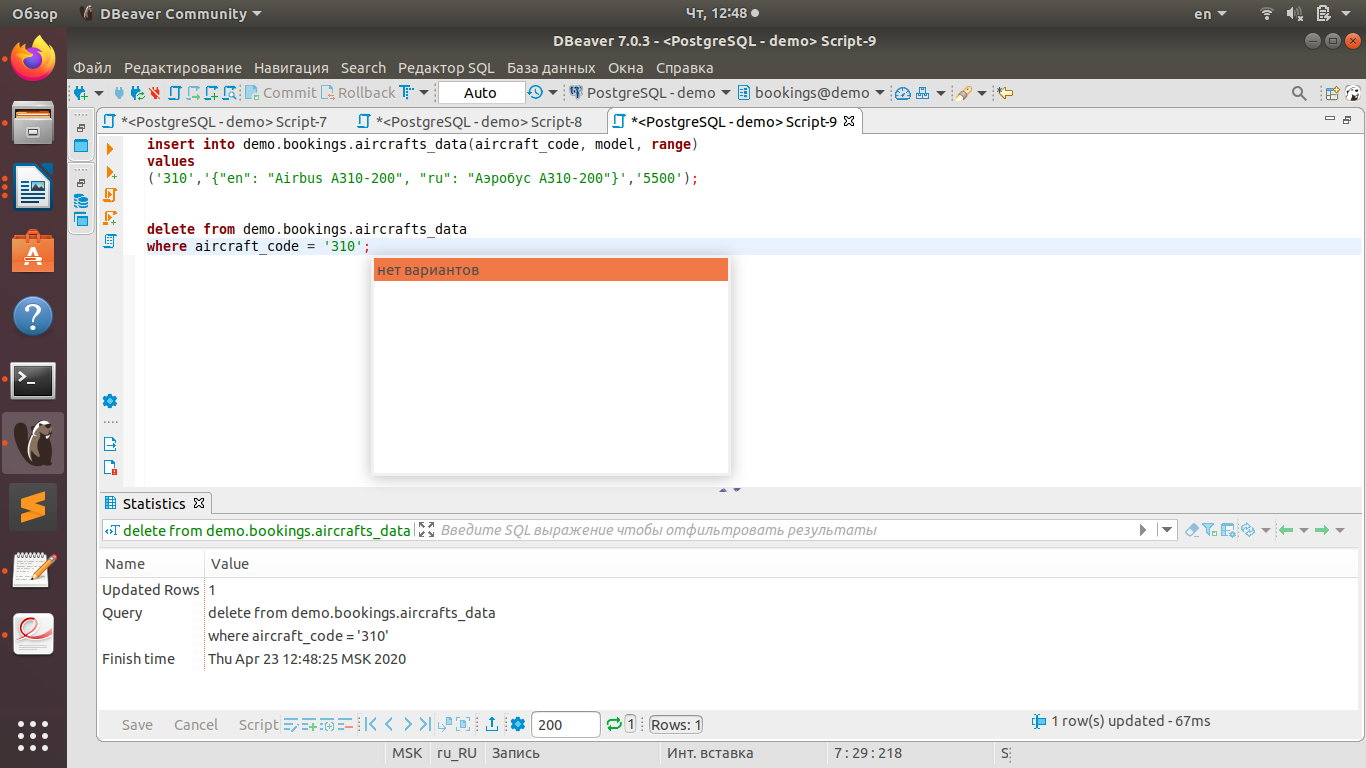
**where** **date**(book\_date) = '2017-01-01' **and** total\_amount **between** '6000' **and** '10000';



В таблице самолеты удалим строку, в которой код самолета = 310:

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

**where** aircraft\_code = '310';



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

**explain** **select**

f.flight\_no №\_рейса

,f.departure\_airport А\_о

,f.arrival\_airport А\_п

,t.book\_ref №\_брони

,**date**(b.book\_date) **as** Дата\_брони

,t.passenger\_name ФИ\_пассажира

,tfl.ticket\_no №\_билета

**from** demo.bookings.tickets t

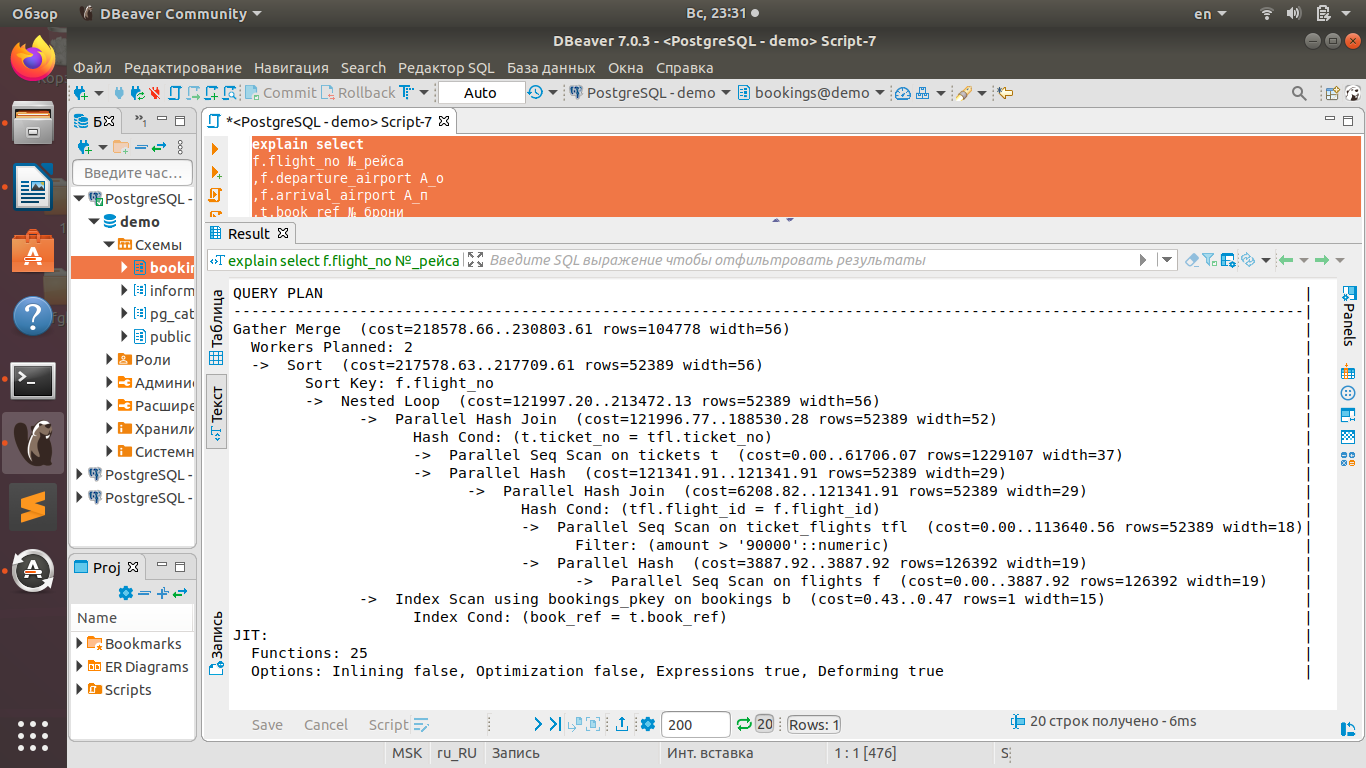
**inner** **join** demo.bookings.bookings b **on** b.book\_ref = t.book\_ref

**inner** **join** demo.bookings.ticket\_flights tfl **on** tfl.ticket\_no = t.ticket\_no

**inner** **join** demo.bookings.flights f **on** f.flight\_id = tfl.flight\_id

**where** tfl.amount > 90000

**order** **by** f.flight\_no **asc**



**explain** (**analyze**) **select**

f.flight\_no №\_рейса

,f.departure\_airport А\_о

,f.arrival\_airport А\_п

,t.book\_ref №\_брони

,**date**(b.book\_date) **as** Дата\_брони

,t.passenger\_name ФИ\_пассажира

,tfl.ticket\_no №\_билета

**from** demo.bookings.tickets t

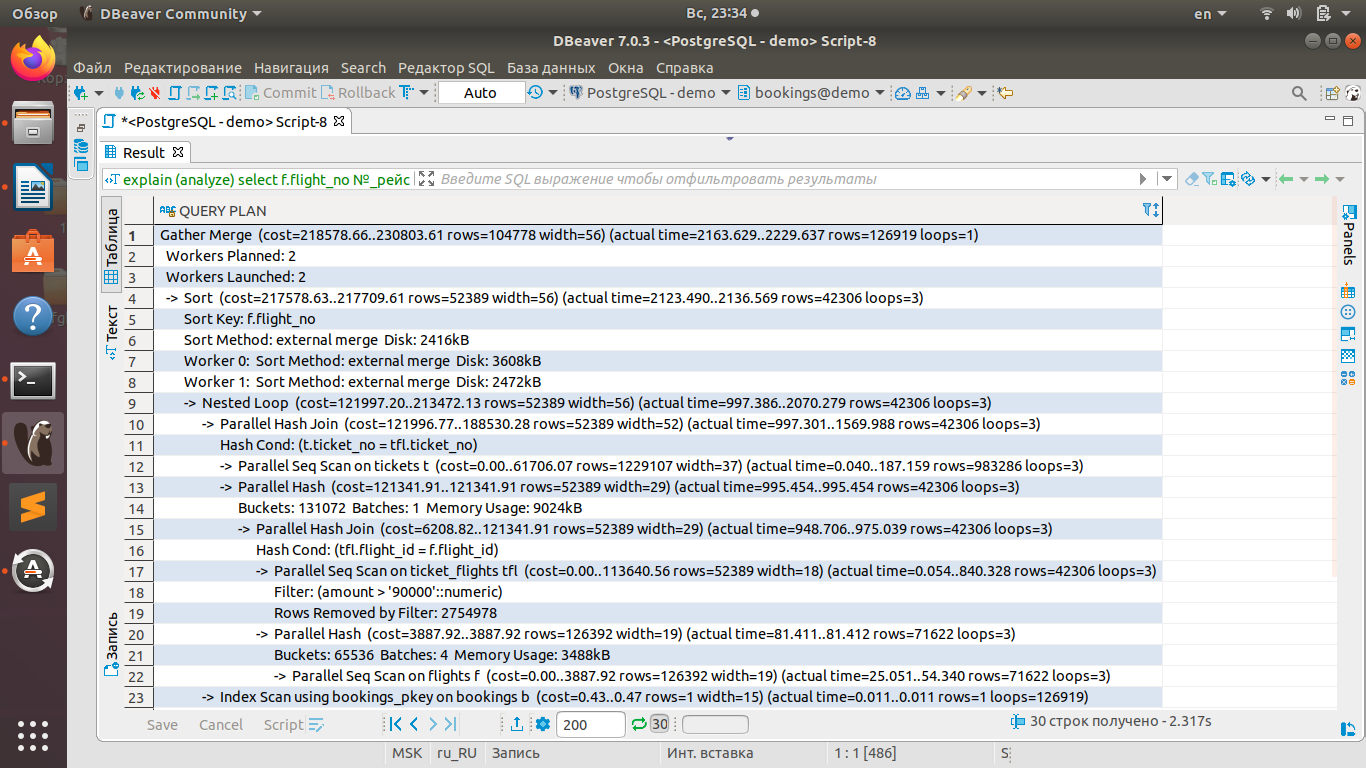
**inner** **join** demo.bookings.bookings b **on** b.book\_ref = t.book\_ref

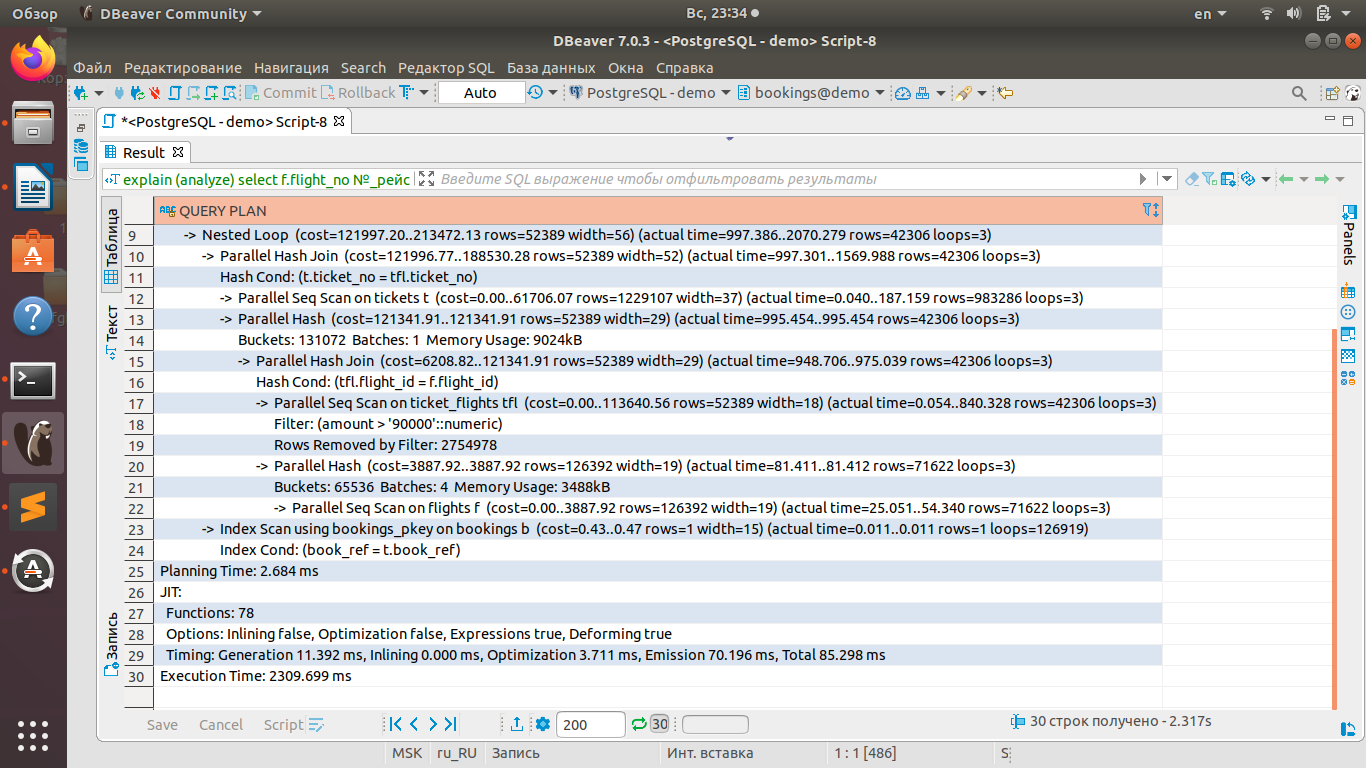
**inner** **join** demo.bookings.ticket\_flights tfl **on** tfl.ticket\_no = t.ticket\_no

**inner** **join** demo.bookings.flights f **on** f.flight\_id = tfl.flight\_id

**where** tfl.amount > 90000

**order** **by** f.flight\_no **asc**





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

Для таблицы tickets был создан индекс indx\_book\_ref по полю book\_ref для оптимизации join с таблицей bookings, где поле book\_ref является первичным ключом. Для таблицы ticket\_flights был создан индекс indx\_ticket\_no по полю ticket\_no для оптимизации join с таблицей tickets, где поле ticket\_no является первичным ключом. Для таблицы ticket\_flights был создан индекс indx\_flight\_id по полю flight\_id для оптимизации join с таблицей flights, где поле flight\_id является первичным ключом.

**create index** indx\_book\_ref **on** bookings.tickets(book\_ref);

**create index** indx\_ticket\_no **on** bookings.ticket\_flights(ticket\_no);

**create index** indx\_flight\_id **on** bookings.ticket\_flights(flight\_id);

Индекс indx\_amount был создан для оптимизации запросов к таблице ticket\_flights.

**create index** indx\_amount **on** bookings.ticket\_flights(amount);

Для select были созданы следующие индексы:

**create index** indx\_flight\_no **on** bookings.flights(flight\_no);

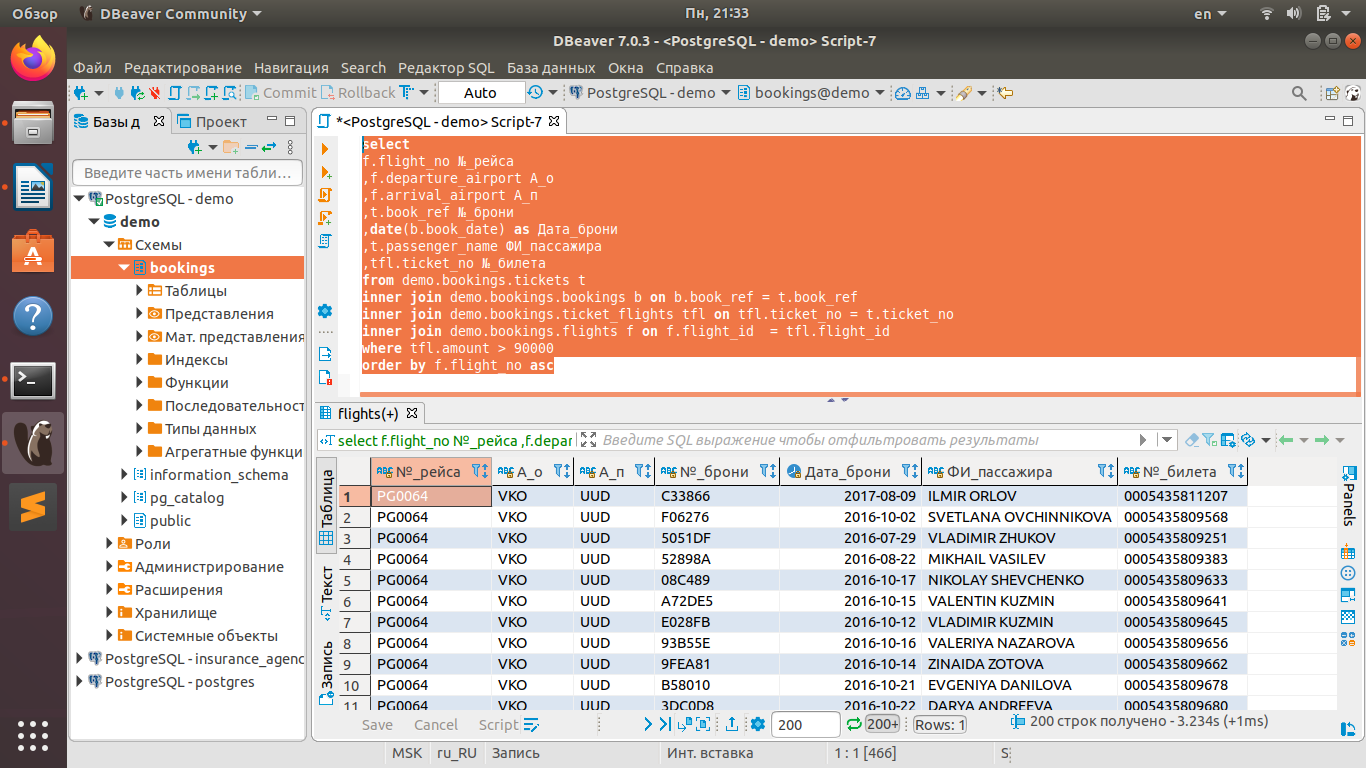
**create index** indx\_departure\_airport **on** bookings.flights(departure\_airport);

**create index** indx\_arrival\_airport **on** bookings.flights(arrival\_airport);

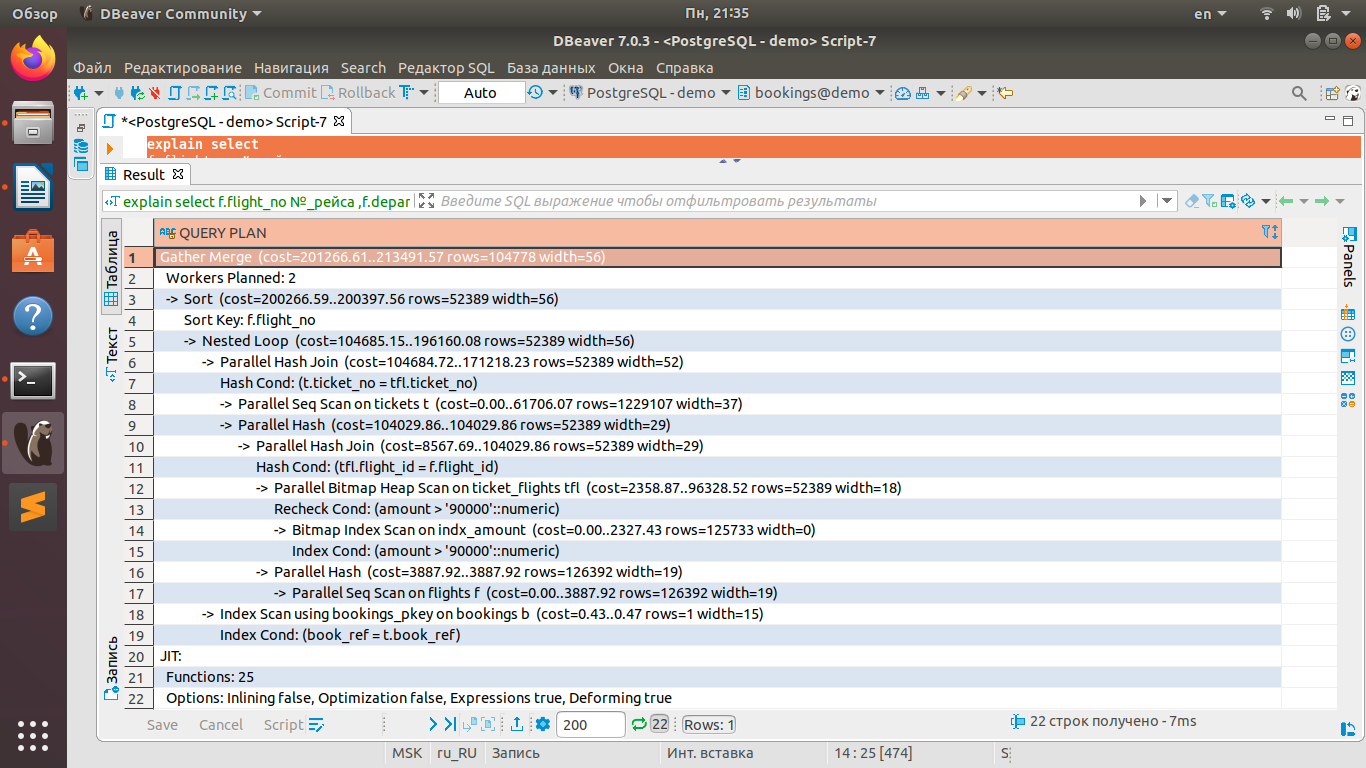
**create index** indx\_book\_date **on** bookings.bookings(book\_date);

**create index** indx\_passenger\_name **on** bookings.tickets(passenger\_name);

Результат выполнения запроса с созданными индексами:



Результат выполнения запроса с помощью команды explain с созданными индексами:



Результат выполнения запроса с помощью команды explain (analyze) с созданными индексами:





Таким образом, использование индексов привело к увеличению производительности запроса (скорости выдачи данных).

Согласно заданию необходимо применить последовательно следующие стратегии оптимизации: Index Scan, Bitmap Index Scan, Index Only Scan.

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

**explain** (**analyze**) **select**

\*

**from** demo.bookings.tickets t

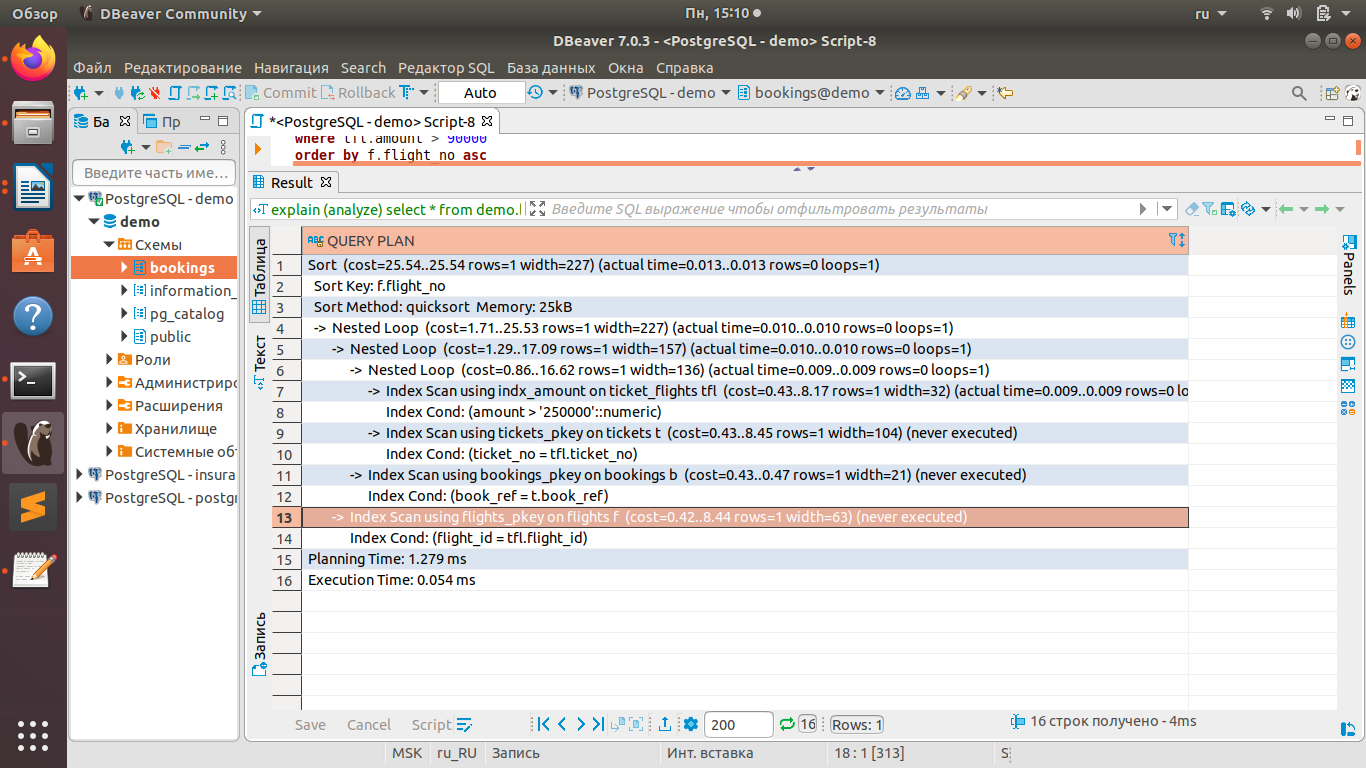
**inner** **join** demo.bookings.bookings b **on** b.book\_ref = t.book\_ref

**inner** **join** demo.bookings.ticket\_flights tfl **on** tfl.ticket\_no = t.ticket\_no

**inner** **join** demo.bookings.flights f **on** f.flight\_id = tfl.flight\_id

**where** tfl.amount > 250000

**order** **by** f.flight\_no **asc**



Index Only Scan - самая быстрая стратегия оптимизации. Читается только индекс.

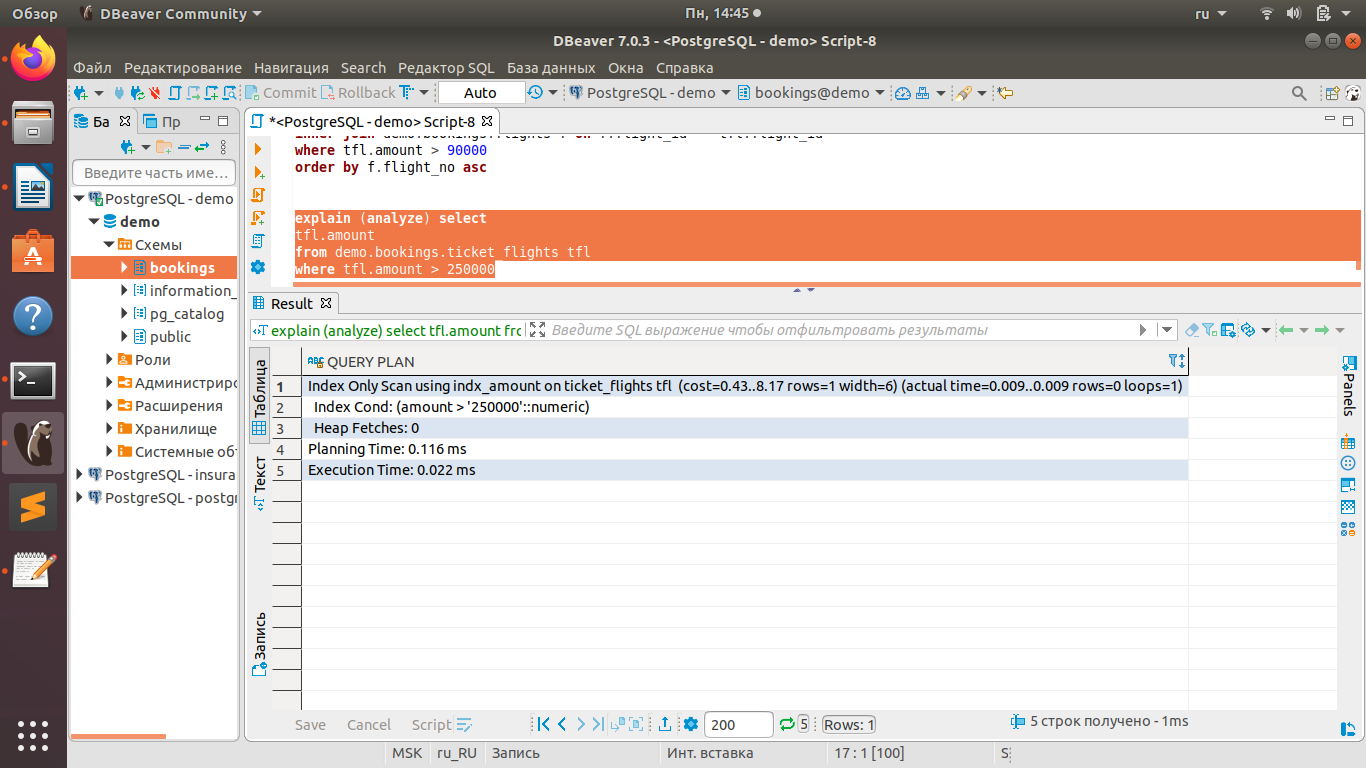
Как правило, основная задача метода доступа— вернуть идентификаторы подходящих строк таблицы, чтобы механизм индексирования мог прочитать из них необходимые данные.

**explain** (**analyze**) **select**

tfl.amount

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

**where** tfl.amount > 250000



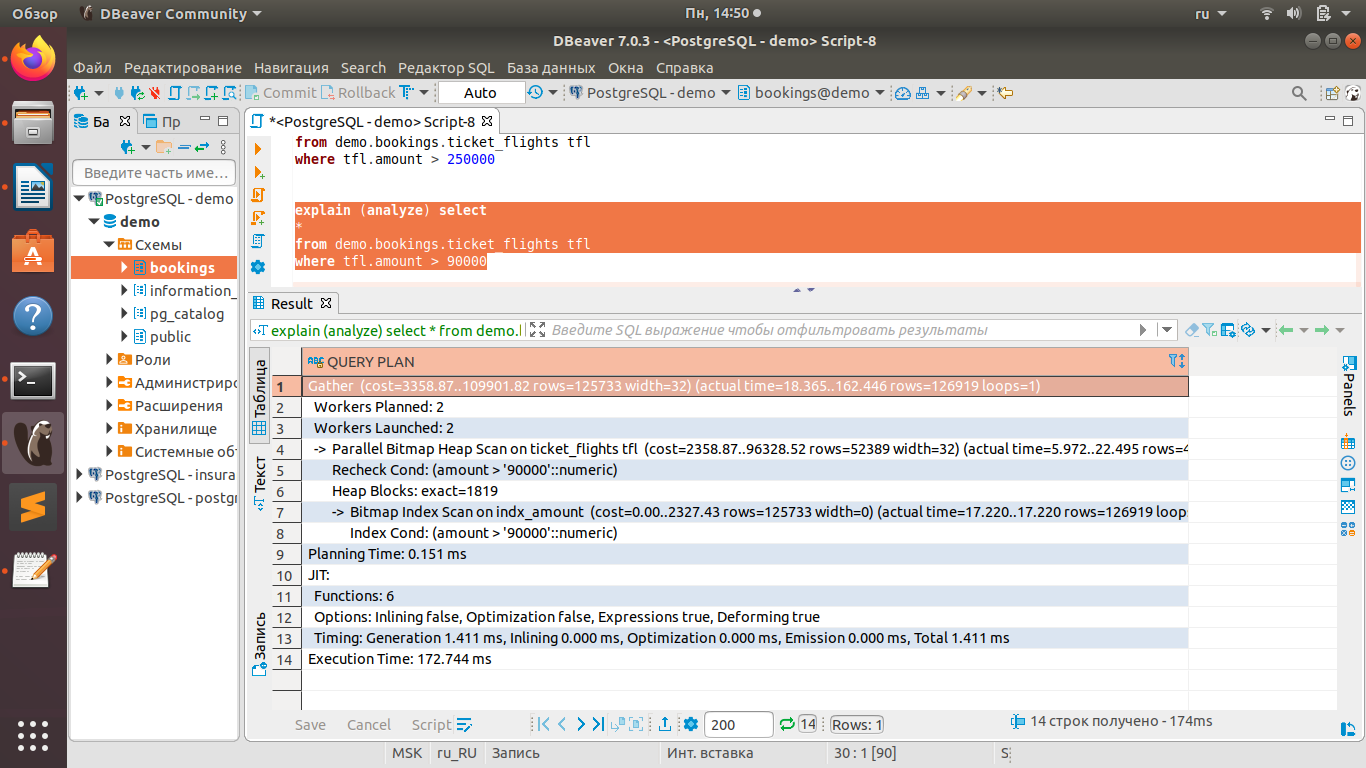
Bitmap Index Scan –используется сначала Index Scan, затем контроль выборки по таблице. Эффективно для большого количества строк.

**explain** (**analyze**) **select**

\*

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

**where** tfl.amount > 90000



1. Результаты исследования использования кэша

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

**select**

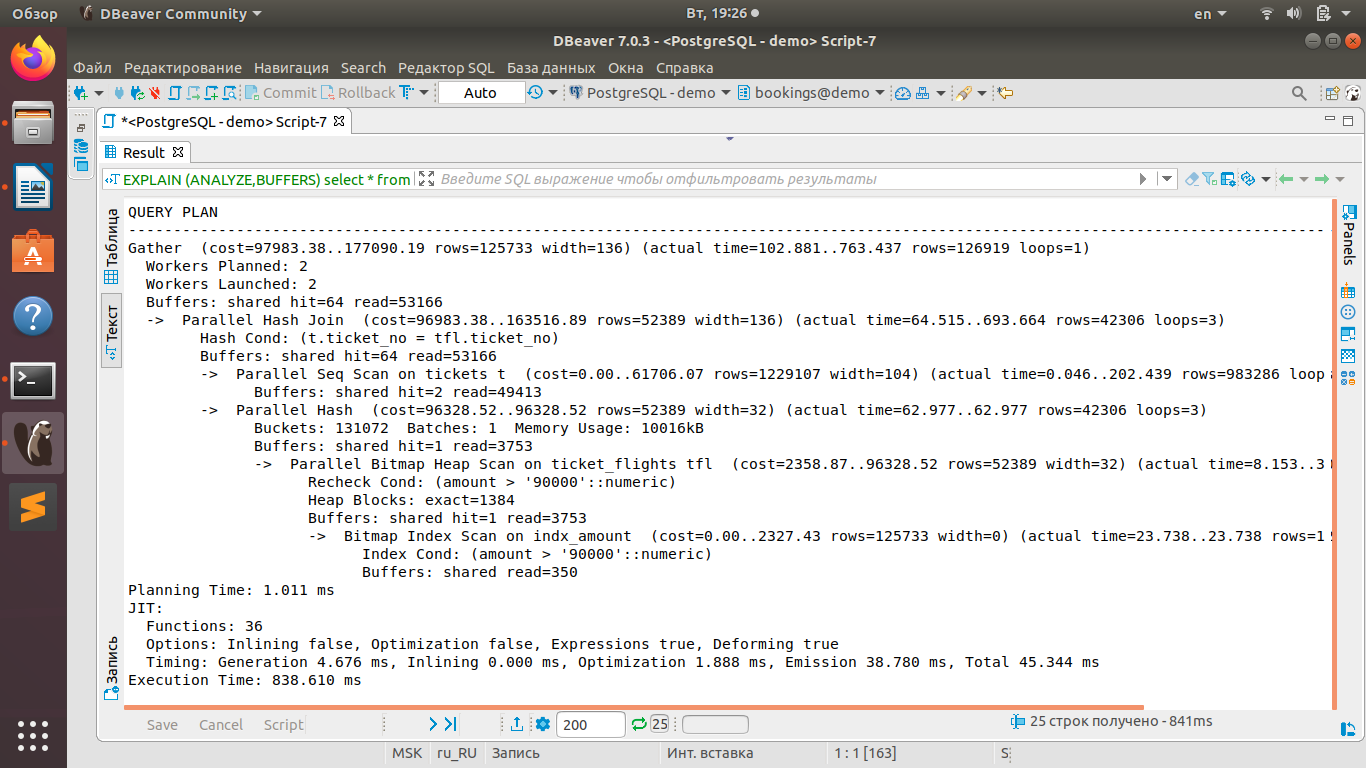
\*

**from** demo.bookings.tickets t

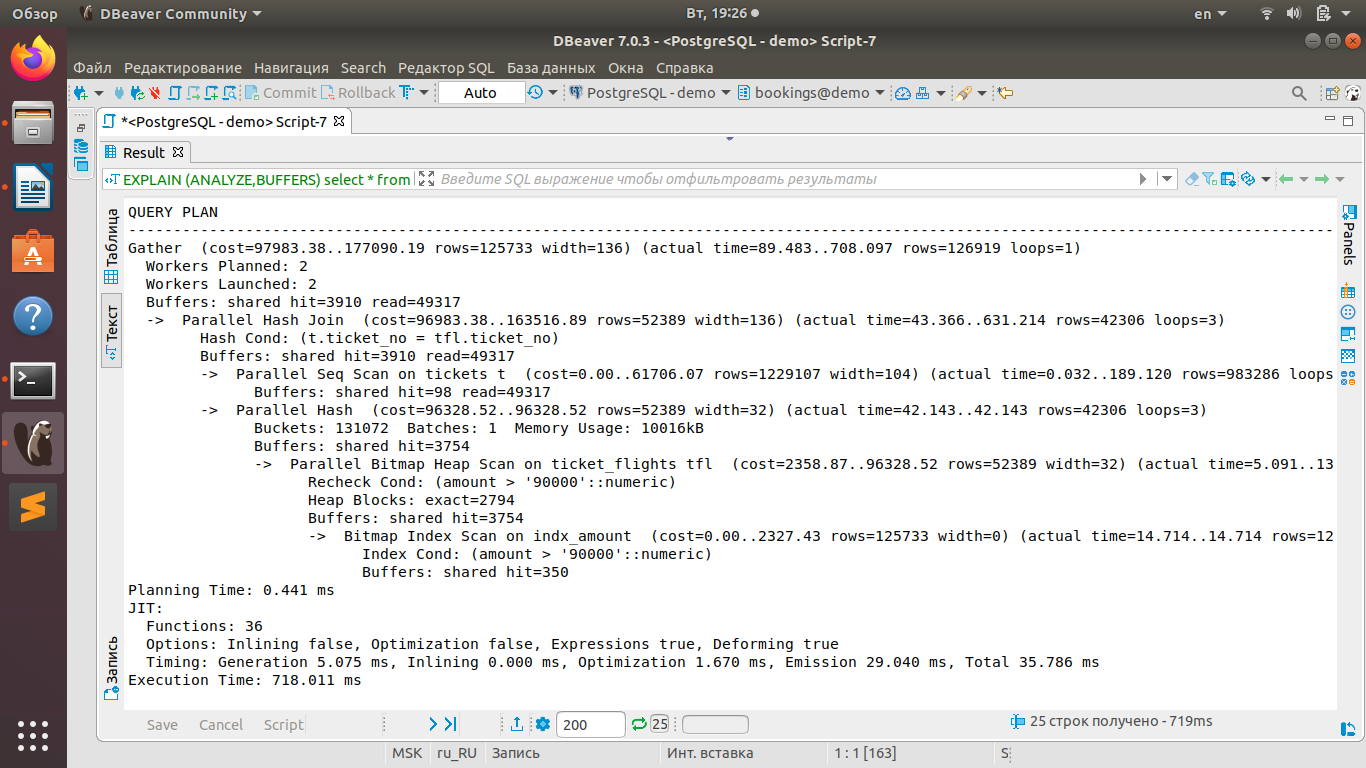
**inner** **join** demo.bookings.ticket\_flights tfl **on** tfl.ticket\_no = t.ticket\_no

**where** tfl.amount > 90000

Результат выполнения запроса первый раз:



Результат выполнения запроса второй раз:



Результат выполнения запроса третий раз:



Таким образом, в первом запросе из кэша Postres было прочитано 64 блоков, о чем говорит запись планировщика: Buffers: shared hit=64. С каждым последующим 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.

Вывод: В результате выполнения лабораторной работы, были изучены возможности профилирования и оптимизации запросов в реляционных базах данных. Появились практические навыки работы с командой EXPLAIN и ее вариациями, а также созданием и использованием индексов Postgres.

Ответы на вопросы:

1. Что такое EXPLAIN PLAN?

EXPLAIN PLAN — это план ядра Postgres на выполнение запроса.

1. Зачем нужны операторы EXPLAIN и ANALYZE? Можно ли их применять отдельно?

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

1. Зачем нужен BUFFERS?

Команда BUFFERS позволят получить информацию об использовании кэша Postgres.

1. Зачем нужны индексы?

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.

1. Какие типы индексов есть в PostgreSQL?

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.

1. Как работает кэш в PostgreSQL?

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.