Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчёт по лабораторным работам

Дисциплина: Базы данных

Выполнил студент гр.3530901/70203 _____Черникова А.С.
(подпись)

Преподаватель _____Мяснов А. В.
(подпись)

“___”_____2020 г.

Санкт-Петербург
2020

Оглавление

1 Лабораторная работа №1

1.1 Цель работы

Познакомиться с основами проектирования схемы БД, способами организации данных в SQL-БД, а также изучить язык SQL

1.2 Программа работы

- Создание проекта для работы в GitLab
- Выбор задания (предметной области), описание набора данных
- Формирование в свободном формате схемы БД, соответствующей заданию
- Создание скрипта, генерирующего БД согласно схеме
- Изменение скрипта создания БД согласно требованиям преподавателя
- Окончательный вид схемы БД

1.3 Выполнение работы

1.3.1 Ссылка на созданный на GitLab проект:

http://gitlab.icc.spbstu.ru/chernrina/db-task

1.3.2 Предметная область:

Создание базы данных для агрегатора авиабилетов

1.3.3 Описание таблиц:

1. human
    - human_document – документ, удостоверяющий личность (первичный ключ)
    - full_name – полное имя
    - gender – пол человека
    - email – почта
2. rate
    - id_ratr – порядковый номер в таблице
    - baggage – допустимый вес багажа
    - select_seat – возможность выбрать место
    - class - класс
3. airline
    - id_airline – название авиакомпании (первичный ключ)
    - supervisor – имя руководителя
    - create_in – год создания
    - mark_sum – средняя оценка по отзывам

4. review
    - id_review – порядковый номер в таблице
    - human_name – имя человека, оставившего отзыв
    - airline – название авиакомпании
    - mark – оценка
5. airplane
    - id_airplane – название самолета (первичный ключ)
    - places – количество мест
    - free_places – свободные места
6. airport
    - id_airport – идентификатор аэропорта (первичный ключ)
    - city – название города
    - full_name – название аэропорта
    - address – адрес
7. voyage
    - id_voyage – название рейса (первичный ключ)
    - airport_from – начальный аэропорт
    - airport_to – конечный аэропорт
    - airline – авиакомпания, выполняющая рейс
8. flight
    - id_flight – порядковый номер в таблице
    - voyage – рейс
    - date_from – время начала полета
    - date_to – время окончания полета
    - airplane – название самолета
9. timetable
    - id_timetable – порядковый номер в таблице
    - date – дата
    - voyage – рейс
    - flight - полет
10. ticket
    - id_ticket – порядковый номер в таблице
    - human_document – название компании доставки
    - flight – полет
    - place – место
    - status – статус билета
    - cost – стоимость
    - rate - тариф

1.3.4 Схема БД

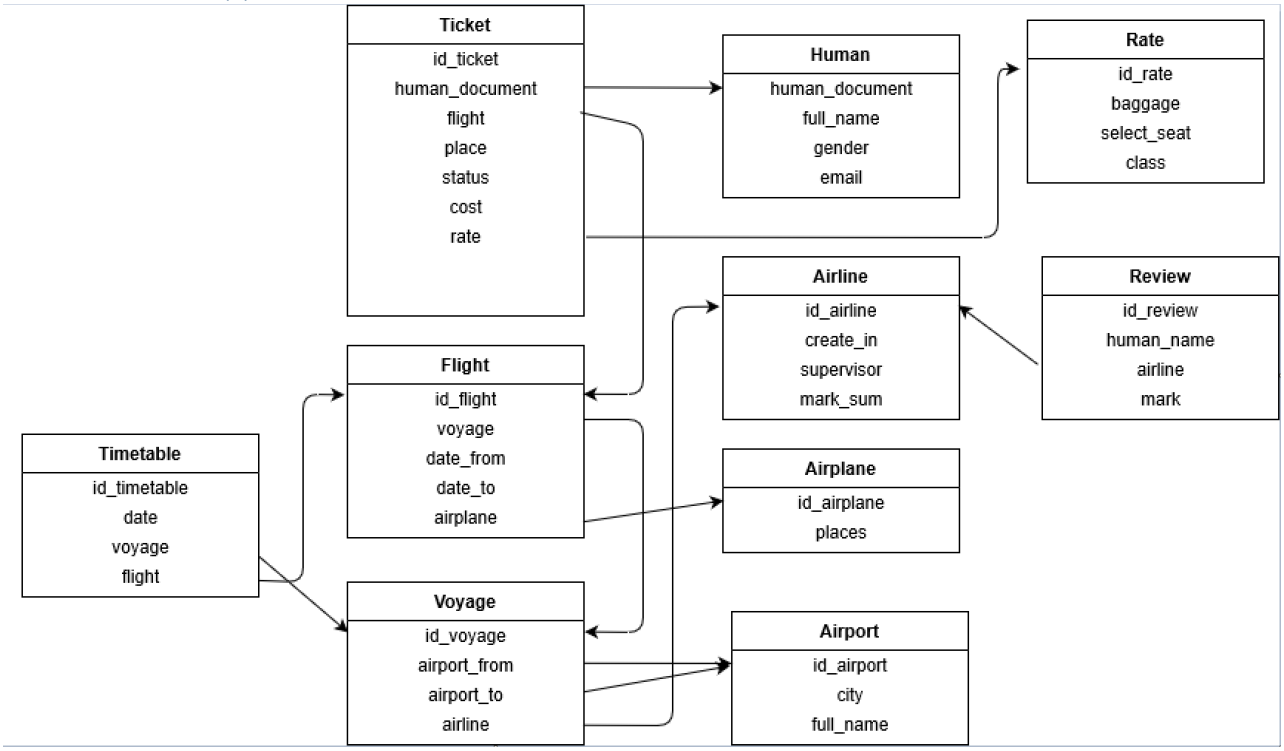


Рис. 1. Схема БД

1.3.5 Скрипт создания БД

Листинг 1.

```
DROP DATABASE IF EXISTS flight_db;

CREATE DATABASE flight_db;
\c flight_db;

START TRANSACTION;

CREATE TABLE IF NOT EXISTS public.human (
 human_document VARCHAR(25) NOT NULL,
 full_name VARCHAR(100) NOT NULL,
 gender CHAR NOT NULL,
 email VARCHAR(50) NULL,
 PRIMARY KEY (human_document));

CREATE TABLE IF NOT EXISTS public.rate (
 id_rate INT NOT NULL,
 baggage VARCHAR(50) NOT NULL,
 select_seat VARCHAR(10) NOT NULL,
 class CHAR NOT NULL,
 PRIMARY KEY (id_rate));

CREATE TABLE IF NOT EXISTS public.airline (
 id_airline VARCHAR(30) NOT NULL,
 create_in INT NOT NULL,
 supervisor VARCHAR(100) NOT NULL,
 mark_sum NUMERIC NOT NULL,
 PRIMARY KEY (id_airline));

CREATE TABLE IF NOT EXISTS public.review (
 id_review INT NOT NULL,
 human_name VARCHAR(100) NOT NULL,
```

```
 airline VARCHAR(30) NOT NULL,
 mark NUMERIC NOT NULL,
 PRIMARY KEY (id_review),
 CONSTRAINT airline_with_name
  FOREIGN KEY (airline)
  REFERENCES public.airline (id_airline)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION) ;

CREATE TABLE IF NOT EXISTS public.airplane (
 id_airplane VARCHAR(30) NOT NULL,
 places INT NOT NULL,
 free_places INT NOT NULL,
 PRIMARY KEY(id_airplane));

CREATE TABLE IF NOT EXISTS public.airport (
 id_airport VARCHAR(10) NOT NULL,
 full_name VARCHAR(50) NOT NULL,
 city VARCHAR(50) NOT NULL,
 address VARCHAR(100) NOT NULL,
 PRIMARY KEY(id_airport));

CREATE TABLE IF NOT EXISTS public.voyage (
 id_voyage VARCHAR(10) NOT NULL,
 airport_from VARCHAR(10) NOT NULL,
 airport_to VARCHAR(10) NOT NULL,
 airline VARCHAR(30) NOT NULL,
 PRIMARY KEY (id_voyage),
 CONSTRAINT from_with_airport
  FOREIGN KEY(airport_from)
  REFERENCES public.airport (id_airport)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
 CONSTRAINT to_with_airport
  FOREIGN KEY(airport_to)
  REFERENCES public.airport (id_airport)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
 CONSTRAINT airline_with_name
  FOREIGN KEY(airline)
  REFERENCES public.airline(id_airline)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION);

CREATE TABLE IF NOT EXISTS public.flight (
 id_flight VARCHAR(10) NOT NULL,
 voyage VARCHAR(10) NOT NULL,
 date_from TIMESTAMP NOT NULL,
 date_to TIMESTAMP NOT NULL,
 airplane VARCHAR(30) NOT NULL,
 PRIMARY KEY (id_flight),
 CONSTRAINT airplane_with_name
  FOREIGN KEY(airplane)
  REFERENCES public.airplane (id_airplane)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
 CONSTRAINT voyage_with_name
  FOREIGN KEY(voyage)
  REFERENCES public.voyage (id_voyage)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION);

CREATE TABLE IF NOT EXISTS public.ticket (
 id_ticket INT NOT NULL,
 human_document VARCHAR(25) NOT NULL,
 flight VARCHAR(15) NOT NULL,
 place VARCHAR(5) NOT NULL,
 status CHAR(15) NULL,
```

```
 cost NUMERIC NULL,
 rate INT NULL,
 PRIMARY KEY (id_ticket),
 CONSTRAINT document_with_human
  FOREIGN KEY(human_document)
  REFERENCES public.human (human_document)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
 CONSTRAINT rate_with_name
  FOREIGN KEY(rate)
  REFERENCES public.rate (id_rate)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
 CONSTRAINT flight_with_id
  FOREIGN KEY(flight)
  REFERENCES public.flight (id_flight)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION);

CREATE TABLE IF NOT EXISTS public.timetable (
 id_timetable INT NOT NULL,
 date TIMESTAMP NOT NULL,
 voyage VARCHAR(10) NOT NULL,
 flight VARCHAR(10) NOT NULL,
 PRIMARY KEY (id_timetable),
 CONSTRAINT voyage_with_name
  FOREIGN KEY(voyage)
  REFERENCES public.voyage (id_voyage)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
 CONSTRAINT flight_with_name
  FOREIGN KEY(flight)
  REFERENCES public.flight (id_flight)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION);

COMMIT;
```

1.3.6 Изменение скрипта БД

Листинг 2.

```
\c flight_db;

START TRANSACTION;

DROP TABLE IF EXISTS public.order;

CREATE TABLE IF NOT EXISTS public.order (
  id_order INT NOT NULL,
  quantity INT NOT NULL,
  customer VARCHAR(100) NOT NULL,
  PRIMARY KEY (id_order)
);

DROP TABLE IF EXISTS public.trip;

CREATE TABLE IF NOT EXISTS public.trip (
  id_trip INT NOT NULL,
  order_id INT NOT NULL,
  city_from VARCHAR(30) NOT NULL,
  city_to VARCHAR(30) NOT NULL,
  PRIMARY KEY (id_trip),
  CONSTRAINT order_with_order_id
  FOREIGN KEY(order_id)
  REFERENCES public.order (id_order)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION
);

ALTER TABLE public.ticket ADD
  trip_id INT NOT NULL;

ALTER TABLE public.ticket ADD
  seq_num INT NOT NULL;

ALTER TABLE public.ticket ADD
  CONSTRAINT trip_with_trip_id
  FOREIGN KEY(trip_id)
  REFERENCES public.trip (id_trip)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;

COMMIT;
```

1.3.7 Окончательная схема БД

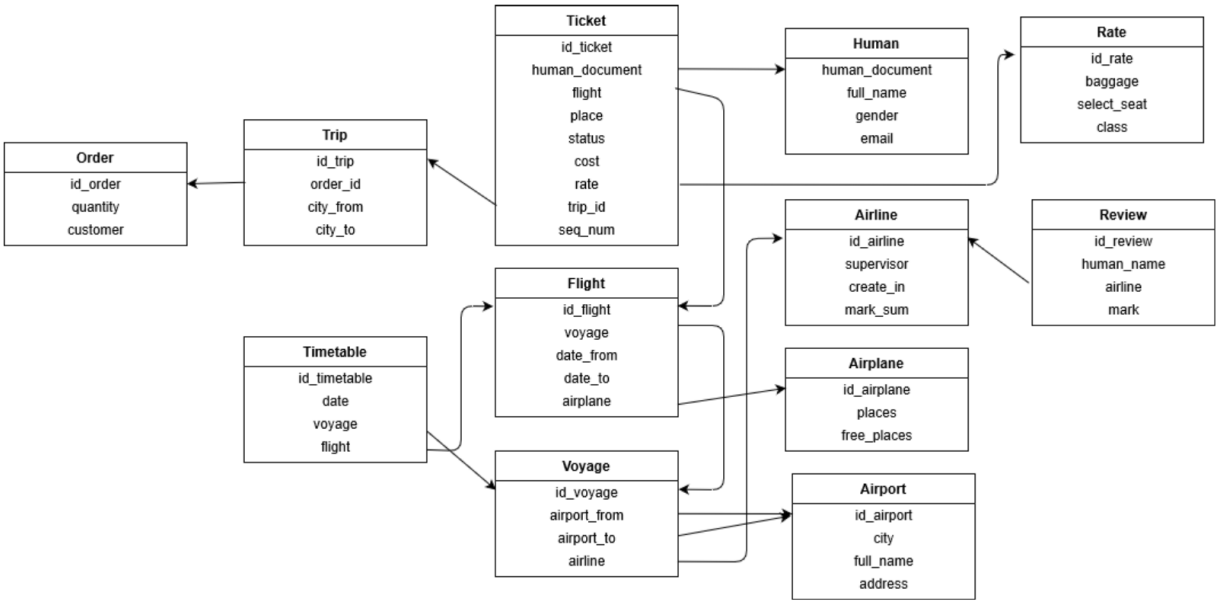


Рис. 2. Окончательный вид схемы БД

Добавлены таблицы:

1. order
    - id_order – порядковый номер в таблице
    - quantity – количество билетов
    - customer – покупатель
2. trip
    - id_trip – порядковый номер в таблице
    - order_id – порядковый номер заказа
    - city_from – начальный город путешествия
    - city_to – конечный город путешествия

В таблицу ticket добавлены поля:

- trip_id – порядковый номер путешествия
- seq_num – порядковый номер билета для реализации стыковки

1.4 Вывод

В данной лабораторной работе был создан проект на GitLab, выбрана предметная область базы данных, описан набор данных, сформирована схема БД в графическом формате, а также изучен язык SQL, с помощью которого разработаны скрипты для создания таблиц согласно схеме, а также скрипт изменения таблиц согласно заданию преподавателя. Сложно было понять логику связей между таблицами, но удалось разобраться.

2 Лабораторная работа № 2

2.1 Цель работы

Сформировать набор данных, позволяющий производить операции на реальных объемах данных.

2.2 Программа работы

- Реализация в виде программы параметризируемого генератора, который позволит сформировать набор связанных данных в каждой таблице
- Количество записей в справочных таблицах должно соответствовать ограничениям предметной области
- Количество записей в таблицах, хранящих информацию об объектах или субъектах должно быть параметром генерации
- Значения для внешних ключей необходимо брать из связанных таблиц

2.3 Выполнение работы

2.3.1 Подключение к БД

Для выполнения лабораторной работы выбран язык Python. Подключение к базе данных осуществляется с помощью библиотеки psycopg2. Все данные необходимые для подключения находятся в строке подключения.

Пример подключения к БД:
connection = psycopg2.connect(dbname='flight_db', user='user_flight', password='user', host='127.0.0.1')
cursor=connection.cursor()

2.3.2 Заполнение БД

2.3.2.1 Заполнение таблицы human

Листинг 3.

```
def human_generation(amount=0, surname=0, name=0, fathers_name=0, gender=0):

    flag = surname != 0 and name != 0 and fathers_name != 0 and gender != 0

    cursor.execute('SELECT human_document FROM public.human')
    human_doc_all = cursor.fetchall()

    for i in range(len(human_doc_all)):
        human_doc.append(human_doc_all[i][0])

    human = ''

    for count in range(int(amount)):

        newDoc = str(random.randrange(1000,10000,1)) + ' ' + str(random.randrange(100000,1000000,1))
        while (newDoc in human_doc):
            doc = str(random.randrange(1000,10000,1)) + ' ' + str(random,randrange(100000,1000000,1))

        human_doc.append(newDoc)

        newEmail = ''
        for j in range(random.randrange(1,15,1)):
            newEmail = newEmail+random.choice(letters)
        newEmail = newEmail+random.choice(emails)

        if (flag):
            human = human + '( \'' + str(newDoc) + '\', \'' + surname + ' ' + name + ' ' + fathers_name + '\', \'' + gender + '\',\'' + newEmail + '\'),'
            print('Successful adding')
        else:
            if (random.randrange(0,2,1)):
                human = human + '( \'' + str(newDoc) + '\', \'' + random.choice(names_man) + '\', \'' + 'M' + '\',\'' + newEmail + '\'),'

            else:
                human = human + '( \'' + str(newDoc) + '\', \'' + random.choice(names_man) + '\', \'' + 'Ж' + '\',\'' + newEmail + '\'),'

    exec_str = 'INSERT INTO public.human (human_document, full_name, gender, email) VALUES {}'.format(human)
    cursor.execute(exec_str[:len(exec_str)-1])
```

human_document	full_name	gender	email
4145 960563	Князев Власий Геласьевич	М	IXqozLXC@yandex.com
2915 805289	Кулагин Аввакум Кириллович	Ж	SkC@yandex.com
8163 725152	Кириллов Панкрат Леонидович	Ж	RTxizE@yandex.com
6924 276670	Игнатьев Александр Борисович	М	OCRAvcnrp@yandex.com
6076 495863	Степанов Фрол Кириллович	М	RidHQuvbjxhcyA@yandex.com
8386 186946	Кириллов Панкрат Леонидович	Ж	Hmwffk@mail.ru

Рис. 3 Пример заполнения таблицы human

2.3.2.2 Заполнение таблицы rate

Листинг 4.

```
def rate_generation():

    rate = ''

    for i in range(len(classes)):
        for j in range(len(baggage)):
            for k in range(len(select_seat)):
                rate = rate + '( \'' + baggage[j] + '\', \'' + select_seat[k] + '\', \'' + classes[i] +'\'),'

    exec_str = 'INSERT INTO public.rate (baggage, select_seat, class) VALUES {}'.format(rate)
    cursor.execute(exec_str[:len(exec_str)-1])
```

id_rate	baggage	select_seat	class
1	До 10 кг	Можно выбрать место	Эконом
2	До 10 кг	Выбрать место нельзя	Эконом
3	До 20 кг	Можно выбрать место	Эконом
4	До 20 кг	Выбрать место нельзя	Эконом
5	До 23 кг	Можно выбрать место	Эконом
6	До 23 кг	Выбрать место нельзя	Эконом
7	Только ручная кладь до 5 кг	Можно выбрать место	Эконом
8	Только ручная кладь до 5 кг	Выбрать место нельзя	Эконом
9	До 10 кг	Можно выбрать место	Бизнес

Рис. 4. Пример заполнения таблицы rate

2.3.2.3 Заполнение таблицы airline

Листинг 5.

```
def airline_generation():

    airline = ''

    for i in range(len(airlines)):
        element = airlines[i].split(' \n')
        airline = airline + '( \'' + element[0] + '\',\'' + element[1] + '\',\'' + element[2] + '\',\'' + str(0) + '\'),'

    exec_str = 'INSERT INTO public.airline (id_airline, supervisor, create_in, mark_sum) VALUES {}'.format(airline)
    cursor.execute(exec_str[:len(exec_str)-1])
```

id_airline	create_in	supervisor	mark_sum
Aurora Airlines	2013	Константин Сухоребрик	3.642857142857142857142857143
Azimuth Airlines	2017	Екжанов Павел	3.625
Azur Air	2014	Юрий Стогний	3.090909090909090909090909091
I-Fly	2009	Кирилл Романовский	4.555555555555555555555555556
Алсора	1995		4.266666666666666666666666667
Ангара	2000	Анатолий Федорович Юртаев	4.285714285714285714285714286
Aeroflot	1932	Виталий Савельев	4.818181818181818181818181818
VIM-Avia	2002	Лайко Андрей Константинович	4.222222222222222222222222222
Владивосток Авиа	1932	Дмитрий Иванович Тыщук	4.166666666666666666666666667
Дон Авиа	1925	Денис Изюмский	4.705882352941176470588235294

Рис. 5. Пример заполнения таблицы airline

2.3.2.4 Заполнение таблицы review

Листинг 6.

```
def review_generation(amount, airline=0, mark=0):

    flag = airline != 0 and mark != 0

    review = ''

    for count in range(int(amount)):
        if (flag):
            review = review + '( \'' + random.choice(names_man) + '\',\'' + airline + '\',\'' + mark + '\'),'
            print('Successful adding')

        else:
            review = review + '( \'' + random.choice(names_man) + '\',\'' + random.choice(airlines).split(' \'')[0] + '\',\'' + str(random.randrange(0,10,1)) + '\'),'

    exec_str = 'INSERT INTO public.review (human_name, airline, mark) VALUES {}'.format(review)
    cursor.execute(exec_str[:len(exec_str)-1])

    connection.commit()

    for i in range(len(airlines)):

        elem = airlines[i].split(' \'')[0]

        cursor.execute("""SELECT mark FROM public.review WHERE airline=%s""",(elem,))
        results = cursor.fetchall()

        if(len(results)!=0):
            sums = 0
            for j in range(len(results)):
                sums = sums + results[j][0]
            sums = sums / len(results)
            cursor.execute('UPDATE airline SET mark_sum=%s WHERE id_airline=%s', (sums,elem,))
```

id_review	human_name	airline	mark
1	Пахомов Нинель Серапионович	Azimuth Airlines	3
2	Давыдов Пантелей Протасьевич	Дон Авиа	7
3	Чернов Макар Макарович	Austrian Airlines	0
4	Доронин Эрнест Матвеевич	Azimuth Airlines	2
5	Калашников Агафон Львович	Победа	3
6	Иванков Андрей Рубенович	Якутия	5

Рис. 6. Пример заполнения таблицы review

2.3.2.5 Заполнение таблицы airplane

Листинг 7.

```
def airplane_generation():

    airplane = ''
    for i in range(len(airplanes)):
        element = airplanes[i].split(' \ ')
        airplane = airplane + '( \'' + element[0] + '\',\'' + element[1] + '\'),'

    exec_str = 'INSERT INTO public.airplane (id_airplane, places) VALUES {}'.format(airplane)
    cursor.execute(exec_str[:len(exec_str)-1])
```

id_airplane	places
Ty-134	80
Ty-154	152
Airbus A310	183
Airbus A320	90
Airbus A330	335
Boeing-737-600	122
Boeing-737-700	140

Рис. 7. Пример заполнения таблицы airline

2.3.2.6 Заполнение таблицы airport

Листинг 8.

```
def airport_generation():

    airport = ''
    for i in range(len(airports)):
        element = airports[i].split('|')
        airport = airport + '( \'' + element[0] + '\',\'' + element[2] + '\',\'' + element[1] + '\'),'

    exec_str = 'INSERT INTO public.airport (id_airport, city, full_name) VALUES {}'.format(airport)
    cursor.execute(exec_str[:len(exec_str)-1])
```

id_airport	full_name	city
AAA	Анаа	Анаа
AAC	Аль-Ариш	Аль-Ариш
AAH	Аахен	Аахен
AAJ	Аварадам	Аварадам
AAK	Аранука	Аранука
AAL	Аальборг	Аальбург
AAM	Мала-Мала	Мала-Мала
AAN	Эль-Аин	Аль-Айн
AAO	Анако	Анако
AAQ	Витязево	Анапа
AAR	Аарус	Аарус
AAT	Алтай	Алтай
AAU	Асау	Асау
AAY	Аль-Гайда	Аль-Гайда

Рис. 8. Пример заполнения таблицы airport

2.3.2.7 Заполнение таблицы voyage

Листинг 9.

```
def voyage_generation(amount, airp_from=0, airp_to=0,intro=0):

    flag = airp_from != 0 and airp_to != 0

    cursor.execute('SELECT id_voyage FROM public.voyage')
    voyages_all = cursor.fetchall()
    for i in range(len(voyages_all)):
        voyages.append(voyages_all[i][0])

    for count in range(int(amount)):

        elem = random.choice(letters) + str(random.randrange(0,100,1)) + random.choice(letters) + str(random.randrange(0,10,1))
        while (elem in voyages):
            elem = random.choice(letters) + str(random.randrange(0,100,1)) + random.choice(letters) + str(random.randrange(0,10,1))
        voyages.append(elem)

        if (flag):
            if (airp_from == airp_to):
                print("Two airports are the same")
                sys.exit(1)
            cursor.execute('INSERT INTO public.voyage (id_voyage, airport_from, airport_to, airline) VALUES (%s,%s,%s,%s)',
                (elem,airp_from,airp_to,airlines[random.randrange(1,len(airlines),1)].split(' \n')[0]))
            flight_generation(elem)
            if (intro != 0): return elem
            else: print('Successful adding')

        else:
            airport_from = airports[random.randint(0,cities-1)].split('|')[0]
            airport_to = airports[random.randint(0,cities-1)].split('|')[0]
            while (airport_to == airport_from):
                airport_to = airports[random.randrange(1,cities,1)].split('|')[0]
            cursor.execute('INSERT INTO public.voyage (id_voyage, airport_from, airport_to, airline) VALUES (%s,%s,%s,%s)',
                (elem,airport_from,airport_to,airlines[random.randrange(1,len(airlines),1)].split(' \n')[0]))
            flight_generation(elem)
```

id_voyage	airport_from	airport_to	airline
F24X2	TUC	YXE	Austrian Airlines
k67F7	AKO	TES	Air Berlin
L75d4	MPI	OLO	Победа
i81E7	RMQ	URS	Ural Airlines
g85P2	MEP	VGA	РусЛайн
X17w4	AOR	STU	Саратовские авиалинии
g60n2	UAQ	TIU	Red Wings Airlines
X61O3	LKC	TJV	Air Europa
A3s0	PIU	PFR	Якутия
x18m2	LGX	SAY	Azimuth Airlines
C47l0	HYA	WET	American Airlines

Рис. 9. Пример заполнения таблицы voyage

2.3.2.8 Заполнение таблицы flight

Листинг 10.

```
def flight_generation(voyage):

    flight = ''
    for i in range(const.month):
        random_date = date.today() + timedelta(days=i,seconds=random.randint(0,3600))
        hour_from = random.randint(0,23)
        hour_to = random.randint(0,23)
        if (hour_to < hour_from or hour_from == 0):
            random_date_to = random_date + timedelta(days=1)
        else: random_date_to = random_date
        date_time_from = " " + str(hour_from) + ":" + str(random.randint(0,59)) + ":00"
        date_time_to = " " + str(hour_to) + ":" + str(random.randint(0,59)) + ":00"
        flight = flight + '( \'' + voyage + '\',\'' + str(random_date) + date_time_from + '\',\'' + str(random_date_to) + date_time_to
         + '\',\'' + airplanes[random.randrange(1,len(airplanes),1)].split(' \n')[0] + '\'),'

    exec_str = 'INSERT INTO public.flight (voyage, date_from, date_to, airplane) VALUES {}'.format(flight)
    cursor.execute(exec_str[:len(exec_str)-1])
```

id_flight	voyage	date_from	date_to	airplane
1	F24X2	2020-05-16 12:02:00	2020-05-17 09:55:00	Airbus A310
2	F24X2	2020-05-17 11:57:00	2020-05-18 00:01:00	Boeing-737-800
3	F24X2	2020-05-18 03:42:00	2020-05-18 20:36:00	Ty-154
4	F24X2	2020-05-19 10:08:00	2020-05-19 21:02:00	Airbus A320
5	F24X2	2020-05-20 15:48:00	2020-05-20 21:22:00	Airbus A320
6	F24X2	2020-05-21 22:26:00	2020-05-21 23:16:00	Boeing-737-700
7	F24X2	2020-05-22 16:37:00	2020-05-22 23:33:00	Ty-154
8	F24X2	2020-05-23 18:28:00	2020-05-23 20:20:00	Boeing-737-900ER
9	F24X2	2020-05-24 22:46:00	2020-05-25 13:45:00	Boeing-737-700
10	F24X2	2020-05-25 03:29:00	2020-05-25 13:04:00	Airbus A310
11	F24X2	2020-05-26 14:02:00	2020-05-27 09:53:00	Boeing-737-700
12	F24X2	2020-05-27 00:44:00	2020-05-28 17:27:00	Airbus A310
13	F24X2	2020-05-28 04:04:00	2020-05-28 16:34:00	Boeing-737-600

Рис.10. Пример заполнения таблицы flight

2.3.2.9 Заполнение таблицы timetable

Листинг 11.

```
def timetable_generation(amount):

    cursor.execute('SELECT count(id_timetable) FROM public.timetable')
    last_date = cursor.fetchone()
    cursor.execute('SELECT date_timetable FROM public.timetable WHERE id_timetable=%s',(last_date,))
    last_date = cursor.fetchone()
    if (last_date is not None):
        last_date = last_date[0] + timedelta(days=1)
    else:
        last_date = date.today()

    timetable = ''
    for count in range(int(amount)):
        date_timetable = last_date + timedelta(days=count)
        date_flight = str(date_timetable) + " 23:59:59"
        cursor.execute("""SELECT id_flight,voyage FROM public.flight WHERE date_from < %s """,(date_flight,))
        results = cursor.fetchall()
        for i in range(len(results)):
            timetable = timetable + '( \'' + str(date_timetable) + '\',\'' + str(results[i][1]) + '\',\'' + str(results[i][0]) + '\'),'

    exec_str = 'INSERT INTO public.timetable (date_timetable, voyage, flight) VALUES {}'.format(timetable)
    cursor.execute(exec_str[:len(exec_str)-1])
```

id_timetable	date_timetable	voyage	flight
1	2020-05-16	F24X2	1
2	2020-05-16	k67F7	91
3	2020-05-16	L75d4	181
4	2020-05-16	i81E7	271
5	2020-05-16	g85P2	361
6	2020-05-16	X17w4	451
7	2020-05-16	g60n2	541
8	2020-05-16	X61O3	631
9	2020-05-16	A3s0	721
10	2020-05-16	x18m2	811
11	2020-05-16	C47l0	901

Рис. 11. Пример заполнения таблицы timetable

2.3.2.10 Заполнение таблицы order

Листинг 12.

```
def order_generation(amount,quantity=0,name=0,city_from=0,city_to=0,transit=0):
    flag = quantity != 0 and name != 0

    for count in range(len(airports)):
        airports_all.append(airports[count].split('|')[0])

    for count in range(int(amount)):
        if (not flag):
            if (random.randrange(0,2,1)): name = random.choice(names_woman)
            else: name = random.choice(names_man)
            quantity = random.randint(1,3)
        cursor.execute('INSERT INTO public.order (quantity, customer) VALUES (%s,%s)',
            (quantity, name))
        if (not flag):
            if (random.randint(0,2)): transit = 1
        trip_generation(quantity,flag,city_from,city_to,transit)
```

id_order	quantity	customer
1	2	Степанов Тарас Артёмович
2	3	Никонова Эльвира Феликсовна
3	1	Исаков Андрей Степанович
4	1	Казакова Эльвина Платоновна
5	3	Авдеева Софья Макаровна
6	2	Самсонов Тарас Денисович
7	2	Коновалов Тимур Александрович
8	2	Зимин Альфред Ефимович
9	1	Петров Харитон Геласьевич

Рис. 12. Пример заполнения таблицы order

2.3.2.11 Заполнение таблицы trip

Листинг 13.

```
def trip_generation(amount,flag,city_from=0,city_to=0,transit=0):
    flag = city_from != 0 and city_to != 0

    cursor.execute("SELECT count(id_order) FROM public.order")
    order_id = cursor.fetchone()[0]

    if (flag):
        cursor.execute('SELECT id_airport FROM public.airport WHERE city=%s',(city_from,))
        airport_from = cursor.fetchone()
        cursor.execute('SELECT id_airport FROM public.airport WHERE city=%s',(city_to,))
        airport_to = cursor.fetchone()
        if (airport_to is None or airport_from is None):
            print('Such city does not exists')
            sys.exit(1)
        cursor.execute('SELECT id_voyage FROM public.voyage WHERE airport_from=%s AND airport_to=%s LIMIT 10', (airport_from,airport_to))
        voyage_id = cursor.fetchone()
        if (voyage_id is None): voyage_id = voyage_generation(1,airport_from,airport_to,1)

    else:
        cursor.execute('SELECT id_voyage FROM public.voyage LIMIT 50')
        voyages_all = cursor.fetchall()
        voyage_id = random.choice(voyages_all)[0]

    if (not flag):
        cursor.execute('SELECT airport_from FROM public.voyage WHERE id_voyage=%s',(voyage_id,))
        airport_from = cursor.fetchone()[0]
        cursor.execute('SELECT airport_to FROM public.voyage WHERE id_voyage=%s',(voyage_id,))
        airport_to = cursor.fetchone()[0]
        cursor.execute('SELECT city FROM public.airport WHERE id_airport=%s',(airport_from,))
        city_from = cursor.fetchone()[0]
        cursor.execute('SELECT city FROM public.airport WHERE id_airport=%s',(airport_to,))
        city_to = cursor.fetchone()[0]
    cursor.execute('INSERT INTO public.trip (order_id, city_from, city_to) VALUES (%s,%s,%s)',
        (order_id,city_from, city_to))
    if (transit!=0): flag = True
    else: flag = False
    ticket_generation(int(amount), voyage_id,flag, airport_from, airport_to)
```

id_trip	order_id	city_from	city_to
1	1	Айова Сити	Хельсинки
2	2	Бахиа-Ангелес	Урай
3	3	Гренобль	Вангренг
4	4	Айова Сити	Хельсинки
5	5	Каниама	Бахоне
6	6	Мамитупо	Оломоуц
7	7	Таракбиц	Дурбан
8	8	Булса	Килагуни
9	9	Сан Хуан	Тимару

Рис. 13. Пример заполнения таблицы trip

2.3.2.12 Заполнение таблицы ticket

Листинг 14.

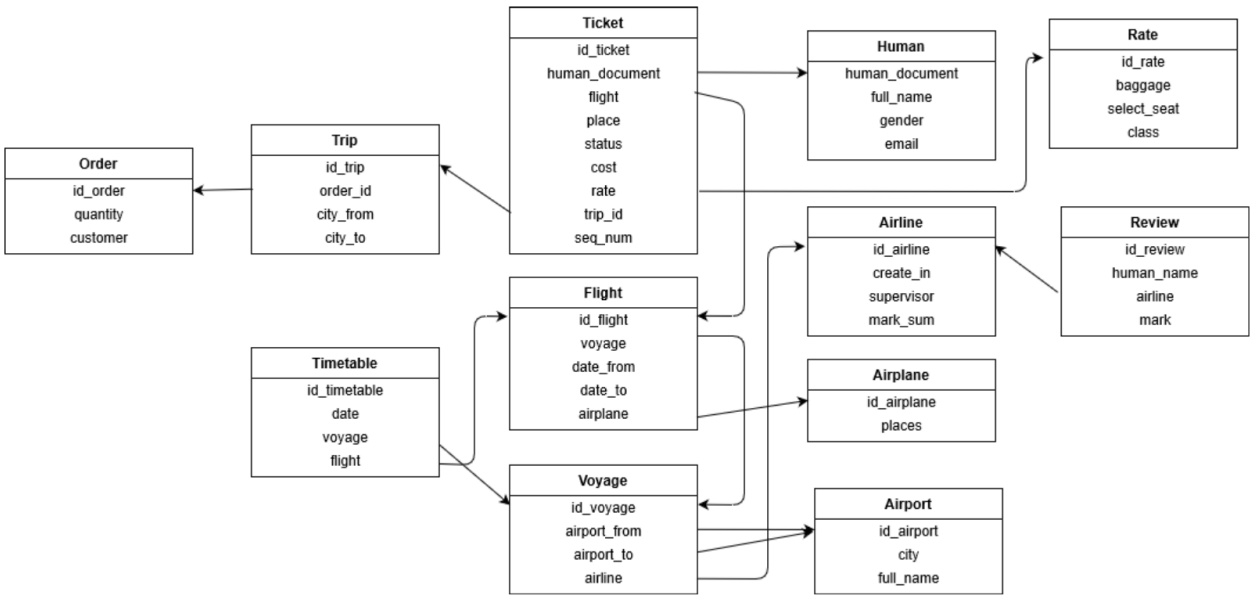
```
def ticket_generation(amount,voyage_id, flag,transit_airport,airport_to):
    seat = const.seat
    status = const.status
    cursor.execute("SELECT id_flight FROM public.flight WHERE voyage=%s AND date_from>%s AND date_from<%s ORDER BY date_from ASC LIMIT 5",
        (voyage_id, date.today(), date.today() + timedelta(days=2)))
    flight_id = cursor.fetchone()[0]
    cursor.execute("SELECT count(id_rate) FROM public.rate")
    rate_amount = cursor.fetchone()[0]
    cursor.execute("SELECT count(id_trip) FROM public.trip")
    trip_id = str(cursor.fetchone()[0])
    cursor.execute('SELECT human_document FROM public.human')
    human_doc = cursor.fetchall()

    ticket=''
    if (flag and amount != 1):
        transit = random.randint(2,int(amount))
        amount = amount-transit

    for count in range(amount):
        place = str(random.randint(1,30)) + random.choice(seat)
        ticket = ticket + '( \'' + random.choice(human_doc)[0] + '\',\'' + str(flight_id) + '\',\'' + place + '\',\'' + random.choice(status) +
         '\',\'' + str(random.randrange(1,100000,50)) + '\',\'' + str(random.randint(1,rate_amount)) + '\',\'' + trip_id + '\',\'' + str(1) + '\'),'

    if (flag and amount != 1):
        document = random.choice(human_doc)[0]
        for count in range(transit):

            place = str(random.randint(1,30)) + random.choice(seat)
            if (count == transit-1):
                cursor.execute('SELECT id_voyage FROM public.voyage WHERE airport_from=%s AND airport_to=%s LIMIT 10',(transit_airport,airport_to,))
            else:
                cursor.execute('SELECT id_voyage FROM public.voyage WHERE airport_from=%s AND airport_to!=%s LIMIT 10',(transit_airport, airport_to,))

            listOfVoyages = cursor.fetchall()

            airport_for_voyage = random.choice(airports_all)
            while (airport_for_voyage == transit_airport or airport_for_voyage == airport_to):
                airport_for_voyage = random.choice(airports_all)
            if (count == transit-1): airport_for_voyage = airport_to

            if (len(listOfVoyages) < 2):
                new_voyage_id = voyage_generation(1,transit_airport,airport_for_voyage,1)
            else:
                new_voyage_id = random.choice(listOfVoyages)[0]
                while new_voyage_id == voyage_id:
                    new_voyage_id = random.choice(listOfVoyages)[0]

            if (count == 0):
                cursor.execute('SELECT id_flight FROM public.flight WHERE voyage=%s AND date_from>%s AND date_from<%s ORDER BY date_from ASC LIMIT 5',
                    (new_voyage_id, date.today(), date.today() + timedelta(days=2)))
                flight_id = cursor.fetchone()[0]
                cursor.execute('SELECT date_to FROM public.flight WHERE id_flight=%s',(flight_id,))
                date_from = cursor.fetchone()[0]
            else:
                date_check = date_from + timedelta(days=2)
                flight_id = None
                while (flight_id is None):
                    cursor.execute('SELECT id_flight FROM public.flight WHERE voyage=%s AND date_from<%s AND date_from>%s LIMIT 5',
                        (new_voyage_id, date_check,date_from))
                    flight_id = cursor.fetchone()
                    if (flight_id is None):
                        flight_generation(new_voyage_id)
                flight_id = flight_id[0]
                cursor.execute('SELECT date_to FROM public.flight WHERE id_flight=%s',(flight_id,))
                date_from = cursor.fetchone()[0]

            transit_airport = airport_for_voyage
            ticket = ticket + '( \'' + document + '\',\'' + str(flight_id) + '\',\'' + place + '\',\'' + random.choice(status) + '\',\'' +
            str(random.randrange(1,100000,50)) + '\',\'' + str(random.randint(1,rate_amount)) + '\',\'' + trip_id + '\',\'' + str(count+1) + '\'),'
            print(trip_id) #

    exec_str = 'INSERT INTO public.ticket (human_document,flight, place, status, cost, rate, trip_id, seq_num) VALUES {}'.format(ticket)
    cursor.execute(exec_str[:len(exec_str)-1])
```

id_ticket	human_document	flight	place	status	cost	rate	trip_id	seq_num
1	3871 521933	9001	27B	Оплачено	87701	11	1	1
2	3871 521933	9092	28E	Бронь	21401	16	1	2
3	2191 137561	9181	29B	Бронь	43601	13	2	1
4	2191 137561	9272	24A	Бронь	91601	2	2	2
5	2191 137561	9363	25A	Оплачено	92851	23	2	3
6	2717 161429	3421	13E	Оплачено	90951	20	3	1
7	2371 719743	2521	4F	Бронь	46451	19	4	1
8	4890 188675	4141	20D	Бронь	16201	18	5	1
9	5598 818400	9451	3E	Оплачено	62801	24	6	1
10	5598 818400	9542	21E	Бронь	91401	24	6	2
11	7778 167510	9631	5D	Оплачено	66351	13	7	1

Рис. 14. Пример заполнения таблицы ticket

При написании генерации данных, написан еще один скрипт, меняющий структуру таблиц (скрипт flight-db-next-changed.sql).

Финальная схема базы данных показана ниже:



Удалены поле free_places в таблице airplane и address в таблице airport, изменены типы данных для некоторых полей нескольких таблиц.

2.4 Вывод

В данной лабораторной работе написан генератор тестовых данных для базы данных. Сложно было реализовывать стыковку полетов, а также нужно было привыкнуть работать с внешними ключами. Много времени ушло на поиск мелких ошибок, которые тормозили всю работу при большем значении данных для генерации (когда нужно было генерировать >5000 записей в таблицу).

3 Лабораторная работа №3

3.1 Цель работы

Познакомиться с языком создания запросов управления данными SQL-DML.

3.2 Программа работы

- Изучение SQL-DML.
- Выполнение всех запросов из списка стандартных запросов.
- Получение у преподавателя и реализация SQL-запросов в соответствии с индивидуальным заданием.

3.3 Выполнение работы

3.3.1 Стандартные запросы

3.3.1.1 Сделайте выборку всех данных из каждой таблицы

Листинг 15.

```
SELECT * FROM human;
SELECT * FROM rate;
SELECT * FROM airline;
SELECT * FROM airplane;
SELECT * FROM airport;
SELECT * FROM review;
SELECT * FROM voyage;
SELECT * FROM flight;
SELECT * FROM timetable;
SELECT * FROM public.order;
SELECT * FROM trip;
SELECT * FROM ticket;
```

Результаты работы (часть):

human_document	full_name	gender	email
8153 226123	Орлов Кассиан Лукьянович	Ж	NCZgSAvPsGEQeH@gmail.com
2322 183713	Миронов Корнелий Иосифович	М	ZeO@yandex.com
9514 930110	Дементьев Мартин Митрофанович	М	shOpHbJinBbomW@gmail.com
7315 158034	Артемьев Елисей Владимирович	Ж	qQtdKdhaqS@gmail.com
9447 279702	Ермаков Ярослав Богуславович	Ж	lvSZZKdfSQI@yandex.com
6531 973241	Федотов Александр Антонинович	Ж	yhrDZaziGcp@mail.ru

Рис. 15. Таблица human

id_rate	baggage	select_seat	class
1	До 10 кг	Можно выбрать место	Эконом
2	До 10 кг	Выбрать место нельзя	Эконом
3	До 20 кг	Можно выбрать место	Эконом
4	До 20 кг	Выбрать место нельзя	Эконом

Рис. 16. Таблица rate

id_airline	create_in	supervisor	mark_sum
Aurora Airlines	2013	Константин Сухоребрик	4.4
Azimuth Airlines	2017	Екжанов Павел	5.25806451612903225806451612 9
Azur Air	2014	Юрий Стогний	4.25
I-Fly	2009	Кирилл Романовский	5.19047619047619047619047619 0
Алсора	1995		5.38888888888888888888888889
Ангара	2000	Анатолий Федорович Юртаев	4.47368421052631578947368421 1

Рис. 17. Таблица airline

id_airplane	places
Ту-134	80
Ту-154	152
Airbus A310	183
Airbus A320	90

Рис. 18. Таблица airplane

id_airport	full_name	city
AAA	Анаа	Анаа
AAC	Аль-Ариш	Аль-Ариш
AAH	Аахен	Аахен
AAJ	Аварадам	Аварадам
AAK	Аранука	Аранука
AAL	Аальборг	Аальбург
AAM	Мала-Мала	Мала-Мала
AAN	Эль-Аин	Аль-Айн

Рис. 19. Таблица airport

id_review	human_name	airline	mark
1	Брагин Игнат Эдуардович	S7 Airlines	8
2	Авдеев Ярослав Степанович	Korean air	3
3	Дьячков Ермолай Федосеевич	Delta Air Lines	4
4	Дементьев Рудольф Святославович	Aeroflot	5

Рис. 20. Таблица review

id_voyage	airport_from	airport_to	airline
Z70q3	CAS	MMB	Pegas Fly
J88b1	ALC	MVL	American Airlines
s40S8	LSO	TNI	Air Astana
k27G1	SFA	SXT	VIM-Avia
o64F4	HEA	VID	Pegas Fly
X14L3	RCH	BHW	Ангара
a59z6	OSD	CMT	Победа
u10Q7	MOM	BBC	Austrian Airlines

Рис. 21. Таблица voyage

id_flight	voyage	date_from	date_to	airplane
1	Z70q3	2020-04-16 02:52:00	2020-04-16 11:03:00	Boeing-737-900ER
2	Z70q3	2020-04-17 05:18:00	2020-04-17 18:36:00	Boeing-737-800
3	Z70q3	2020-04-18 18:05:00	2020-04-19 13:23:00	Boeing-737-900ER
4	Z70q3	2020-04-19 02:35:00	2020-04-19 12:25:00	Airbus A330
5	Z70q3	2020-04-20 13:59:00	2020-04-20 14:04:00	Boeing-737-600
6	Z70q3	2020-04-21 14:54:00	2020-04-22 02:10:00	Airbus A310
7	Z70q3	2020-04-22 23:06:00	2020-04-23 22:25:00	Boeing-737-900ER

Рис. 22. Таблица flight

id_timetable	date_timetable	voyage	flight
1	2020-04-16	Z70q3	1
2	2020-04-16	J88b1	91
3	2020-04-16	s40S8	181
4	2020-04-16	k27G1	271
5	2020-04-16	o64F4	361
6	2020-04-16	X14L3	451
7	2020-04-16	a59z6	541
8	2020-04-16	u10Q7	631

Рис. 23. Таблица timetable

id_order	quantity	customer
1	1	Нестерова Доля Константиновна
2	2	Романова Октябрина Агафоновна
3	2	Ефимова Хельга Якововна
4	2	Большакова Аза Демьяновна
5	2	Соболева Виктория Антониновна

Рис. 24. Таблица order

id_trip	order_id	city_from	city_to
1	1	Иокеа	Гранд-Цесс
2	2	Каваджиа	Нью-Йорк
3	3	Никаро	Фоллз-Крик
4	4	Остерсанд	Камета
5	5	Вичи	Париж
6	6	Лес-Саблес	Сатна

Рис. 25. Таблица trip

id_ticket	human_document	flight	place	status	cost	rate	trip_id	seq_num
1	8153 226123	1171	17B	Бронь	75501	2	1	1
2	5123 672601	9001	14C	Бронь	66451	1	2	1
3	5123 672601	9091	3F	Бронь	98101	12	2	2
4	1648 580313	9181	13E	Бронь	39551	19	3	1
5	1648 580313	9272	8B	Бронь	19251	15	3	2
6	3890 965559	9361	9C	Оплачено	45101	1	4	1
7	3890 965559	9452	11F	Оплачено	52401	6	4	2
8	9534 945705	9541	23F	Бронь	19251	1	5	1

Рис. 26. Таблица ticket

3.3.1.2 Сделайте выборку данных из одной таблицы при нескольких условиях, с использованием логических операций, LIKE, BETWEEN, IN (не менее 3-х разных примеров)

Листинг 16.

```
SELECT id_airline,create_in
FROM airline
WHERE id_airline LIKE '%S%';

SELECT human_document,flight,cost
FROM ticket
WHERE cost BETWEEN 10000 AND 20000;

SELECT id_airline,mark_sum
FROM airline
WHERE mark_sum IN (4.4,4.5,4.6,4.7,4.8,4.9,5.0);

SELECT id_flight,voyage,airplane
FROM flight
WHERE airplane LIKE 'A%' AND voyage IN ('d13J4','I84c5');
```

Результат (часть):

```
     id_airline     | create_in
--------------------+-----------
 NordStar           |      2008
 S7 Airlines        |      1957
 SriLankan Airlines |      1979
(3 строки)
```

Рис. 27. Like

```
 human_document | flight | cost
----------------+--------+-------
 1648 580313    |   9272 | 19251
 9534 945705    |   9541 | 19251
 9534 945705    |   9632 | 13001
 1582 590997    |  10083 | 19651
 2247 452021    |  10261 | 10351
 2571 564031    |  10712 | 17401
```

Рис. 28. Between

```
   id_airline    | mark_sum
-----------------+----------
 Aurora Airlines |      4.4
 NordStar        |      4.5
 Победа          |      4.4
 Utair           |        5
(4 строки)
```

Рис. 29. In

id_flight	voyage	airplane
54811	d13J4	Airbus A330
54812	d13J4	Airbus A330
54814	d13J4	Airbus A310
54815	d13J4	Airbus A330
54818	d13J4	Airbus A320
54819	d13J4	Airbus A330
54821	d13J4	Airbus A310
54824	d13J4	Airbus A320
54826	d13J4	Airbus A320
54829	d13J4	Airbus A320
54832	d13J4	Airbus A320

Рис. 30. Like, in

3.3.1.3 Создайте в запросе вычисляемое поле

Листинг 17.

```
SELECT id_order, quantity, (id_order*quantity) AS mult
FROM public.order;
```

Результат (часть):

id_order	quantity	mult
1	1	1
2	2	4
3	2	6
4	2	8
5	2	10
6	3	18
7	3	21
8	2	16
9	1	9
10	3	30
11	2	22
12	2	24
13	3	39
14	1	14

Рис. 31. Результаты запроса 3

3.3.1.4 Сделайте выборку всех данных с сортировкой по нескольким полям

Листинг 18.

```
SELECT * FROM ticket
ORDER BY cost, trip_id ASC;
```

Результат (часть):

id_ticket	human_document	flight	place	status	cost	rate	trip_id	seq_num
646	9808 534917	39782	6A	Бронь	1	9	390	2
791	7081 732944	45182	26E	Бронь	1	5	480	2
1080	3741 561635	56792	19E	Бронь	1	14	641	2
697	8841 623409	41493	28B	Оплачено	101	20	423	2
1689	9621 136719	3061	1B	Бронь	151	10	985	1
1280	2598 430348	28621	28B	Бронь	251	8	754	1
1527	7785 542392	11521	29E	Бронь	251	13	891	1
946	7787 260500	27451	15C	Оплачено	351	16	570	1
230	6928 521329	3331	18C	Бронь	401	21	133	1

Рис. 32. Результат запроса 4

3.3.1.5 Создайте запрос, вычисляющий несколько совокупных характеристик таблиц

Листинг 19.

```
SELECT AVG(cost) AS cost_avg, MAX(flight) AS max_flight
FROM ticket;
```

Результат:

```
      cost_avg       | max_flight
---------------------+------------
 49900.137854363535 |     174511
(1 строка)
```

Рис. 33. Результат запроса 5

3.3.1.6 Сделайте выборку данных из связанных таблиц (не менее двух примеров)

Листинг 20.

```
SELECT id_flight, voyage.airport_from AS from, voyage.airport_to AS to
FROM flight
INNER JOIN voyage ON voyage.id_voyage = flight.voyage;

SELECT ticket.human_document, human.full_name AS name, rate.baggage AS baggage
FROM ticket
LEFT JOIN rate ON rate.id_rate = ticket.rate
LEFT JOIN human ON human.human_document = ticket.human_document;
```

Результат (часть):

```
 id_flight | from | to
-----------+------+-----
         1 | CAS  | MMB
         2 | CAS  | MMB
         3 | CAS  | MMB
         4 | CAS  | MMB
         5 | CAS  | MMB
         6 | CAS  | MMB
         7 | CAS  | MMB
         8 | CAS  | MMB
         9 | CAS  | MMB
```

Рис. 34. Результат запроса 6_1

```
human_document |                name                |          baggage
---------------+------------------------------------+---------------------------
8153 226123    | Орлов Кассиан Лукьянович           +| До 10 кг
               |                                    |
5123 672601    | Петров Влас Натанович              +| До 10 кг
               |                                    |
5123 672601    | Петров Влас Натанович              +| До 20 кг
               |                                    |
1648 580313    | Исаков Людвиг Сергеевич            +| До 20 кг
               |                                    |
```

Рис. 35. Результат запроса 6_2

3.3.1.7 Создайте запрос, рассчитывающий совокупную характеристику с использованием группировки, наложите ограничение на результат группировки

Листинг 21.

```
SELECT id_voyage, airport.city AS city_from, airline.mark_sum
AS markOfAirline
FROM voyage
INNER JOIN airport ON airport.id_airport = voyage.airport_from
INNER JOIN airline ON airline.id_airline = voyage.airline
GROUP BY id_voyage, airport.city, airline.mark_sum
HAVING airline.mark_sum>5;
```

Результат (часть):

id_voyage	city_from	markofairline
d59L1	Ванганui	5.31428571428571428571428571
b52e3	Потоси	5.18181818181818181818181818
s92l1	Мулеж	5.18181818181818181818181818
X47M7	Кьяуктау	5.31428571428571428571428571
a11J7	Актау	5.25806451612903225806451613
S87I0	Мандера	5.22580645161290322580645161
y61A3	Монтеррей	5.25806451612903225806451613
y27Z8	Бостон	5.19354838709677419354838709

Рис. 36. Результат запроса 7

3.3.1.8 Придумайте и реализуйте пример использования вложенного запроса

Листинг 22.

```
SELECT id_ticket, flight FROM ticket
WHERE flight IN
(SELECT id_flight FROM flight WHERE voyage LIKE '%70%');
```

Результат (часть):

id_ticket	flight
642	1
592	1
162	1
1132	1
776	1
787	1
44	1
360	1
1733	1
615	1
474	2161
1114	2161
703	2161

Рис. 37. Результат запроса 8

3.3.1.9 С помощью оператора INSERT добавьте в каждую таблицу по одной записи

Листинг 23.

```
INSERT INTO public.human (human_document, full_name, gender, email) VALUES ('1234 567891',
'Ivanov Ivan Ivanovich', 'M', 'ivan@mail.ru');
INSERT INTO public.rate (baggage, select_seat, class) VALUES ('Up to 1 kg', 'Can choose', 'Second');
INSERT INTO public.airline (id_airline, supervisor, create_in, mark_sum) VALUES ('Russia', 'Ivanov
Ivan Ivanovich', '2020', '0');
INSERT INTO public.review (human_name, airline, mark) VALUES ('Ivanov Ivan Ivanovich', 'Russia',
'10');
INSERT INTO public.airplane (id_airplane, places) VALUES ('Airbus A319', '150');
INSERT INTO public.airport (id_airport, city, full_name) VALUES ('AAB','City','Airport');
INSERT INTO public.voyage (id_voyage, airport_from, airport_to, airline) VALUES
('s1s00','LED','KZN','Russia');
INSERT INTO public.flight (voyage, date_from, date_to, airplane) VALUES ('s1s00','2020-04-16
05:25:00','2020-04-16 12:48:00','Boeing-737-900ER');
INSERT INTO public.timetable (date_timetable, voyage, flight) VALUES ('2020-04-16','s1s00',(SELECT
COUNT(id_flight) FROM flight));
INSERT INTO public.order (quantity, customer) VALUES ('1','Ivanov Ivan Ivanovich');
INSERT INTO public.trip (order_id, city_from, city_to) VALUES ((SELECT COUNT(id_order) FROM
public.order), (SELECT city FROM airport WHERE id_airport='LED'), (SELECT city FROM airport
WHERE id_airport='KZN'));
INSERT INTO public.ticket (human_document,flight, place, status, cost, rate, trip_id, seq_num) VALUES
('1234 567891', (SELECT COUNT(id_flight) FROM flight), '1A', 'Paid', '7500', (SELECT
COUNT(id_rate) from rate), (SELECT COUNT(id_trip) FROM trip), '1');
```

3.3.1.10 С помощью оператора UPDATE измените значения нескольких полей у всех записей, отвечающих заданному условию

Листинг 24.

```
UPDATE ticket SET cost=15000 WHERE cost>70000 AND cost<90000;
```

3.3.1.11 С помощью оператора DELETE удалите запись, имеющую максимальное (минимальное) значение некоторой совокупной характеристики

Листинг 25.

```
DELETE FROM ticket WHERE cost= ( SELECT MAX(cost) FROM ticket);
```

3.3.1.12 С помощью оператора DELETE удалите записи в главной таблице, на которые не ссылается подчиненная таблица (используя вложенный запрос)

Листинг 26

```
DELETE FROM human WHERE human_document NOT IN ( SELECT human_document FROM ticket GROUP
BY human_document);
```

3.3.2 Индивидуальные задания

3.3.2.1 Вывести людей, которые летают не менее 2-х раз в месяц за последний год

Алгоритм работы: сгруппировать данные о людях, которые летали более двух раз в месяц за последний год. Затем выбрать из них тех, которые летали каждый месяц.

Листинг 27.

```
select human_document, full_name from
(select human.human_document, full_name, extract(month from
flight.date_from) as mon, count(*) as cnt
from human
inner join ticket on ticket.human_document = human.human_document
inner join flight on flight.id_flight = ticket.flight
where flight.date_from >= (current_date - integer '365')
group by human.human_document, full_name, extract(month from
flight.date_from)
having count(*)>=2) as query
group by human_document, full_name
having count(*) = 12
\g 'dml_ind_res/1.txt'
```

Результат:

```
 human_document | full_name
----------------+-----------
(0 строк)
```

Рис. 38. Результат запроса 1

Такие условия тяжело реализовать в силу особенностей написанной генерации в предыдущей лабораторной.

3.3.2.2 Вывести маршруты, на которых средняя заполняемость полетов менее 50%

Алгоритм работы: выбрать имя товара и артикль товара, а затем посчитать в скольких заказах он участвует. После этого отсортировать полученные данные по убыванию и вывести первые 5 позиций.

Листинг 28.

```
select id_voyage from voyage
inner join flight on flight.voyage=voyage.id_voyage
inner join airplane on airplane.id_airplane=flight.airplane
group by id_voyage
having sum((select count(*) from ticket where flight=flight.id_flight)
/airplane.places)
/(select count(*) from flight where voyage=id_voyage)<0.5
\g 'dml_ind_res/2.txt'
```

Результат (часть):

id_voyage
A20q1
a30h9
a31X1
A35q8
A35w1
a47Z1
a51D4
a51d9
A56C2
a59O5
A60S8
a63R5
a64a9
A70d4
A73W0
a76R7
A81E1
a82O3

Рис. 39. Результат запроса 2

3.4 Вывод

В данной лабораторной работе изучены возможности создания запросов, и получения различных данных из таблиц. Сложным показался первый запрос в индивидуальном задании, т.к. сначала было непонятно, как правильнее и компактнее выбрать данные.

4 Лабораторная работа №4

4.1 Цель работы

Знакомство с проблемами, возникающими при высокой нагрузке на базу данных, и методами их решения, путем оптимизации запросов.

4.2 Программа работы

- Написание параметризированных типовых запросов пользователей
- Моделирование нагрузки базы данных
- Снятие показателей работы сервиса и построение соответствующих графиков
- Применение возможных оптимизаций запросов и повторное снятие показателей
- Сравнительный анализ результатов

4.3 Выполнение работы

4.3.1 Типовые запросы пользователей

Листинг 29.

```
query_1="SELECT human_document from human WHERE full_name = %(name)s;"
query_2="SELECT id_flight, voyage.airport_from AS airport_from, voyage.airport_to AS airport_to FROM flight"\
" INNER JOIN voyage ON voyage.id_voyage = %(voyage)s LIMIT 20;"
query_3="SELECT human_document,cost FROM ticket WHERE cost BETWEEN 10000 AND 25000 AND seq_num>2;"
query_4="INSERT INTO public.rate (baggage, select_seat, class) VALUES (%(baggage)s, %(select_seat)s, %(class)s);"
query_5="INSERT INTO public.order (quantity, customer) VALUES (5,%(customer)s);"
query_6="SELECT id_voyage, airline.mark_sum AS markOfAirline FROM voyage INNER JOIN airline ON airline.id_airline = %(airline)s"\
" WHERE id_voyage LIKE 'A%%'"
query_7="SELECT id_flight,voyage,airplane FROM flight WHERE voyage=%(voyage)s;"
query_8="SELECT id_ticket from ticket WHERE place = %(place)s AND trip_id<100;"
query_9="SELECT human_document, full_name FROM"\
" (SELECT human.human_document, full_name, extract(month from flight.date_from) as mon, count(*) as cnt"\
" FROM human INNER JOIN ticket on ticket.human_document = human.human_document"\
" INNER JOIN flight on flight.id_flight = ticket.flight"\
" WHERE flight.date_from >= (current_date - integer '365')"\
" GROUP BY human.human_document, full_name, extract(month from flight.date_from)"\
" HAVING count(*)>=2) as query"\
" GROUP BY human_document, full_name"\
" HAVING human_document LIKE %(docum)s"
```

4.3.2 Моделирование нагрузки на БД и снятие показателей

Листинг 30.

```
def constant_threads(num_threads):
    curr_res_cosntant_threads.clear()
    for t in range(num_threads):
        dbt = DBThread_constant_threads(t)
        dbt.start()
    while threading.activeCount() > 1:
        time.sleep(1)
    plot_x=[k for k in range(401,10000,200)]
    plot_y=[]
    for i in range(len(curr_res_cosntant_threads[0])):
        curr_sum=0
        for j in range(num_threads):
            curr_sum+=curr_res_cosntant_threads[j][i]
        plot_y.append(curr_sum/num_threads)
    plt.plot(plot_x,plot_y,linewidth=2.0)
    plt.xlabel('Запросов в секунду')
    plt.ylabel('Время ответа на один запрос, мс')
    if before:
        plt.title("Before. Num threads: {}".format(num_threads))
    else:
        plt.title("After. Num threads: {}".format(num_threads))
    plt.show()

def dinamic_threads(num_querys):
    plot_x=[k for k in range(1,31)]
    plot_y=[]
    for num_threads in range(1,31):
        curr_res_dinamic_threads.clear()
        for t in range(num_threads):
            dbt = DBThread_dinamic_threads(num_querys)
            dbt.start()

        while threading.activeCount() > 1:
            time.sleep(1)

        plot_y.append(sum(curr_res_dinamic_threads)/len(curr_res_dinamic_threads))


    plt.plot(plot_x,plot_y,linewidth=2.0)
    plt.xlabel('Количество потоков')
    plt.ylabel('Время ответа на один запрос, мс')
    if before:
        plt.title("Before. Dinamic thread num")
    else:
        plt.title("After. Dinamic thread num")
    plt.show()
```

Функция constant_threads и класс DBThread_constant_threads работают с постоянным количеством потоков. В constant_threads создаётся нужное количество потоков, которые вызывают переменное количество раз (от 401 до 10000 с шагом 200) рандомные пользовательские запросы в функции класса DBThread_constant_threads.

Аналогично функция dinamic_threads и класс DBThread_dinamic_threads работают для изменяющегося количества потоков, только в этом случае работает постоянное количество вызовов запросов (2000).

Также в функциях constant_threads и dinamic_threads строятся графики для отображения результатов. В первом случае на них показывается зависимость времени, потраченного на 1 запрос относительно количества запросов в секунду. Во втором случае – относительно числа потоков, так как число запросов постоянное.

4.3.3 Оптимизация

Для оптимизации запросов были выбраны 2 метода: CREATE INDEX ON И PREPARE. CREATE INDEX ON работает сразу для всех клиентов, поэтому этот вид оптимизации

можно выделить в отдельную функцию, в то время как PREPARE нужно создавать отдельный для каждого клиента, поэтому он реализуется в классе потока.

Листинг 31.

```
def optimize():
    cursor.execute("CREATE INDEX human_name ON human(full_name);")
    cursor.execute("CREATE INDEX ticket_place ON ticket(place) WHERE trip_id<100;")
    cursor.execute("CREATE INDEX voyage_id_voyage ON voyage(id_voyage)")
    cursor.execute("CREATE INDEX flight_voyage ON flight(voyage)")
    cursor.execute("CREATE INDEX airline_id_airline ON airline(id_airline)")

    cursor.execute("CREATE INDEX human_document ON human(human_document)")

    global before
    before = False

def optimize_9():

    cursor.execute("CREATE INDEX hd ON human(human_document)")
    cursor.execute("CREATE INDEX hf ON human(full_name)")
    cursor.execute("CREATE INDEX td ON ticket(human_document)")
    cursor.execute("CREATE INDEX tf ON ticket(flight)")
    cursor.execute("CREATE INDEX fi ON flight(id_flight)")
    cursor.execute("CREATE INDEX hum ON human(human_document, full_name)")

    global before
    before = False
```

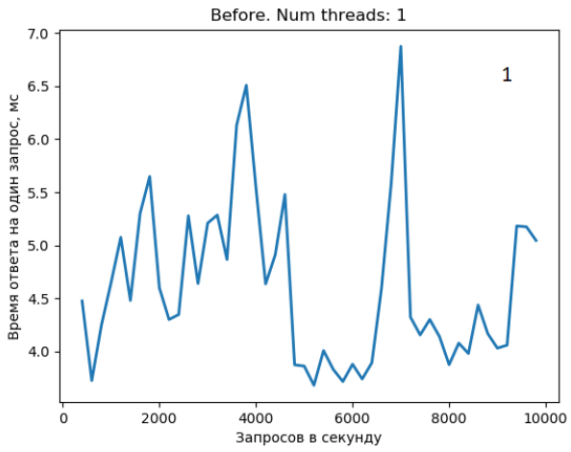
Функция optimize_9 создана отдельно для оптимизации query_9.

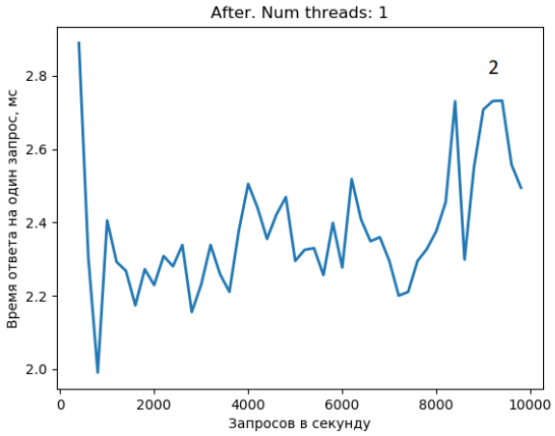
Листинг 32.

```
        if not before:
            self.cur.execute("PREPARE query1 (varchar(100)) AS SELECT human_document from human WHERE full_name = $1;")
            self.cur.execute("PREPARE query2 (varchar(10)) AS SELECT id_flight, voyage.airport_from AS airport_from, voyage.airport_to AS airport_to FROM flight"\
" INNER JOIN voyage ON voyage.id_voyage = $1 LIMIT 20;")
            self.cur.execute("PREPARE query3 AS SELECT human_document,cost FROM ticket WHERE cost BETWEEN 10000 AND 20000 AND seq_num>2;")
            self.cur.execute("PREPARE query4 (varchar(50),varchar(20),varchar(30)) AS INSERT INTO public.rate (baggage, select_seat, class) VALUES ($1,$2,$3);")
            self.cur.execute("PREPARE query5 (varchar(100)) AS INSERT INTO public.order (quantity, customer) VALUES (5,$1);")
            self.cur.execute("PREPARE query6 (varchar(30)) AS SELECT id_voyage, airline.mark_sum AS markOfAirline FROM voyage INNER JOIN airline ON airline.id_airline = $1"\
" WHERE id_voyage LIKE 'A%%';")
            self.cur.execute("PREPARE query7 (varchar(10)) AS SELECT id_flight,voyage,airplane FROM flight WHERE voyage= $1;")
            self.cur.execute("PREPARE query8 (varchar(5)) AS SELECT id_ticket from ticket WHERE place = $1 AND trip_id<100;")
            self.cur.execute("PREPARE query9 (varchar) AS SELECT human_document, full_name FROM"\
" (SELECT human.human_document, full_name, extract(month from flight.date_from) as mon, count(*) as cnt"\
" FROM human INNER JOIN ticket on ticket.human_document = human.human_document"\
" INNER JOIN flight on flight.id_flight = ticket.flight"\
" WHERE flight.date_from >= (current_date - integer '365')"\
" GROUP BY human.human_document, full_name, extract(month from flight.date_from)"\
" HAVING count(*)>=2) as query"\
" GROUP BY human_document, full_name"\
" HAVING human_document LIKE $1 ;")
```

4.3.4 Сравнительный анализ результатов

4.3.4.1 Анализ графических результатов





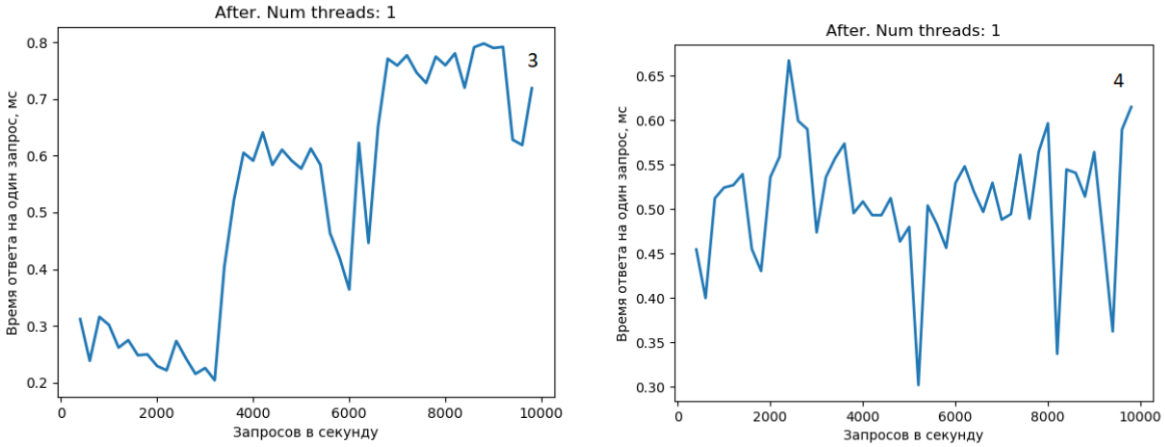


Рис. 40. Результат работы запросов при 1 потоке: 1 – до оптимизации, 2 – только после prepare, 3 – только после index, 4 – после всех видов оптимизации

При работе 1 потока с увеличением количества запросов в секунду видно, что результат улучшается при добавлении каждого вида оптимизации. После применения индексов результат улучшается более чем в 10 раз.

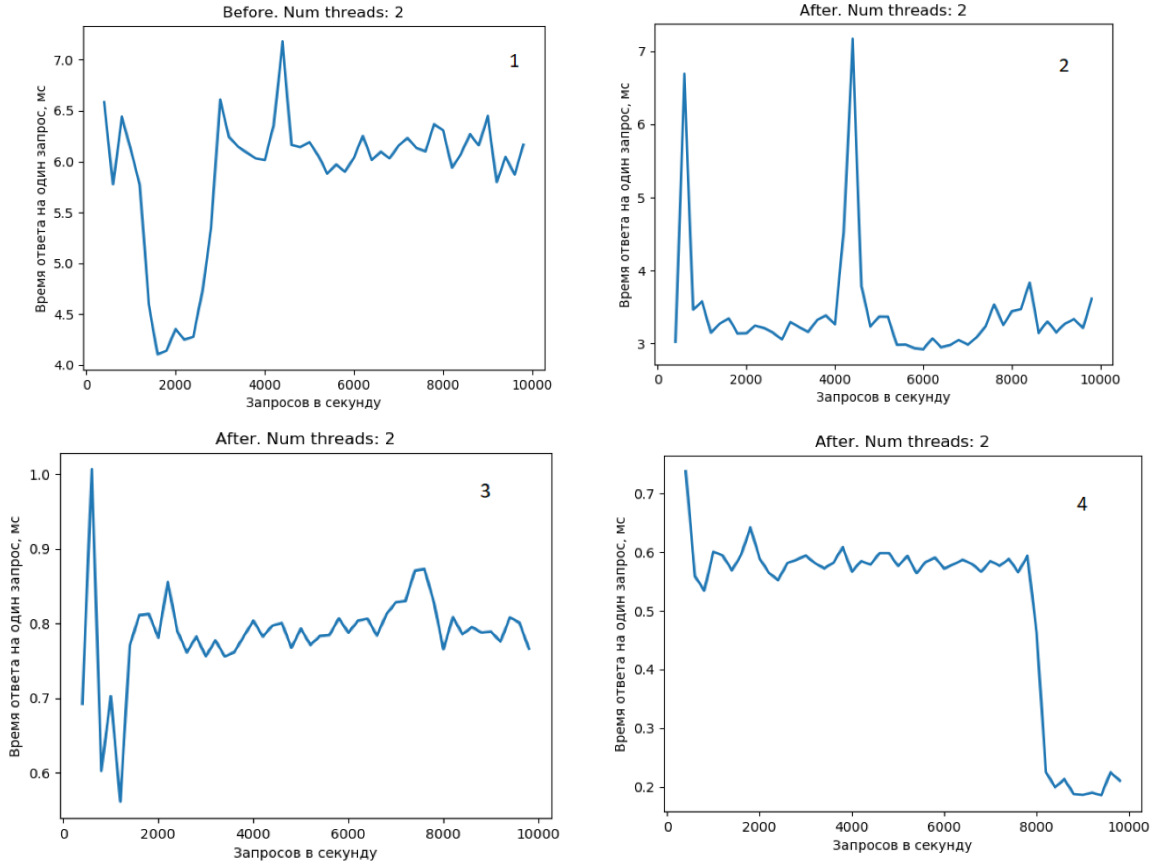


Рис. 41. Результат работы запросов при 2 потоках: 1 – до оптимизации, 2 – только после prepare, 3 – только после index, 4 – после всех видов оптимизации

При работе 2 потоков с увеличением количества запросов в секунду видно, что результат улучшается при добавлении каждого вида оптимизации. После применения индексов результат улучшается почти в 10 раз.

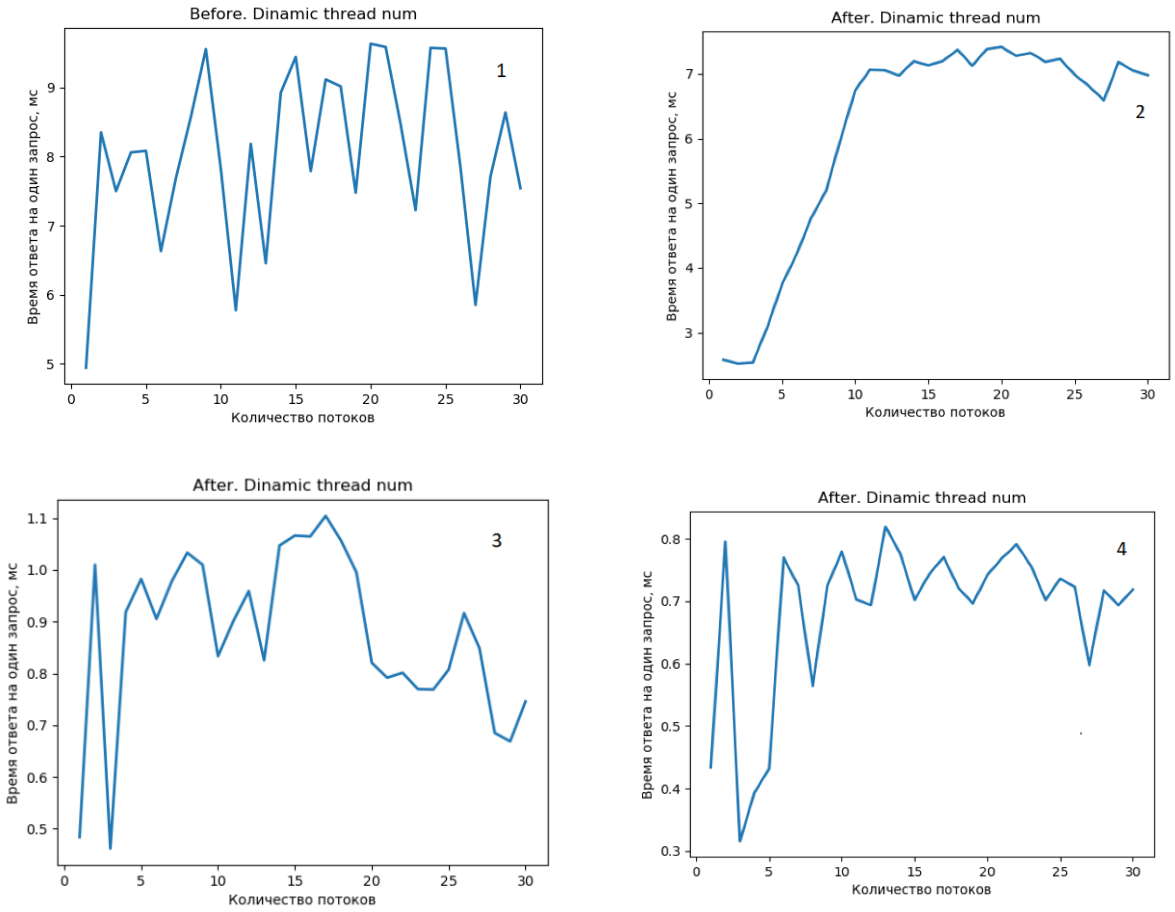


Рис. 42. Результат работы запросов при изменяющемся количестве потоков: 1 – до оптимизации, 2 – только после prepare, 3 – только после index, 4 – после всех видов оптимизации

По результатам можно сделать вывод, что все виды оптимизации работают. Лучший вариант можно получить при использовании обоих видов оптимизации. Как можно увидеть, при изменяющемся количестве потоков результат оптимизации виден сильнее.

Оптимизация query_9 заняла много времени, т.к. запрос сложный и результаты получились не совсем ожидаемые.

Лучшие результаты показаны на рис.43.

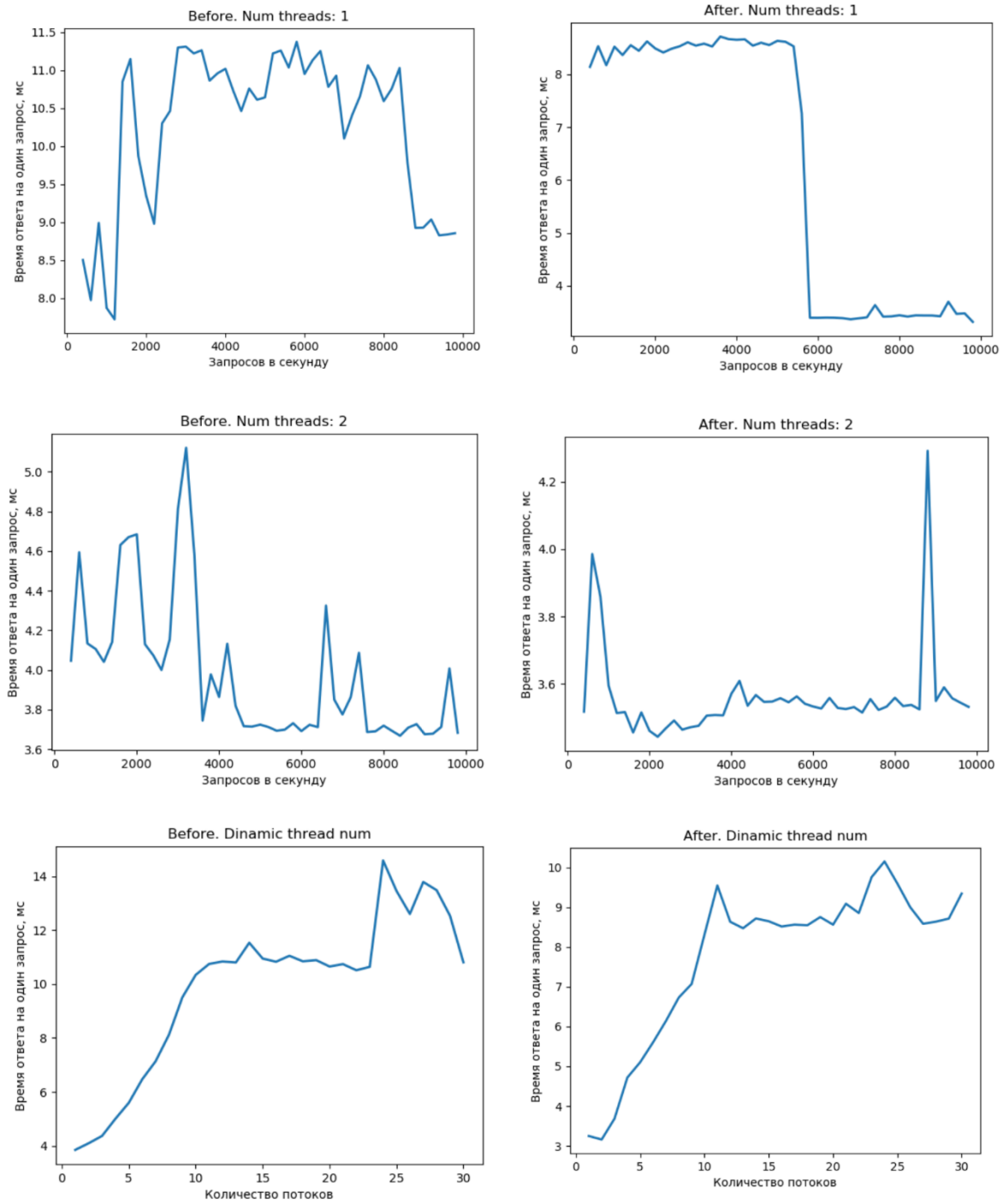


Рис.43.

На рис.43. два верхний графика - результат работы запросов при 1 потоке до оптимизации и после, следующие два графика - результат работы запросов при 2 потоках до оптимизации и после, нижние два графика - результат работы запросов при изменяющемся количестве потоков до оптимизации и после.

4.3.4.2 Анализ результатов explain

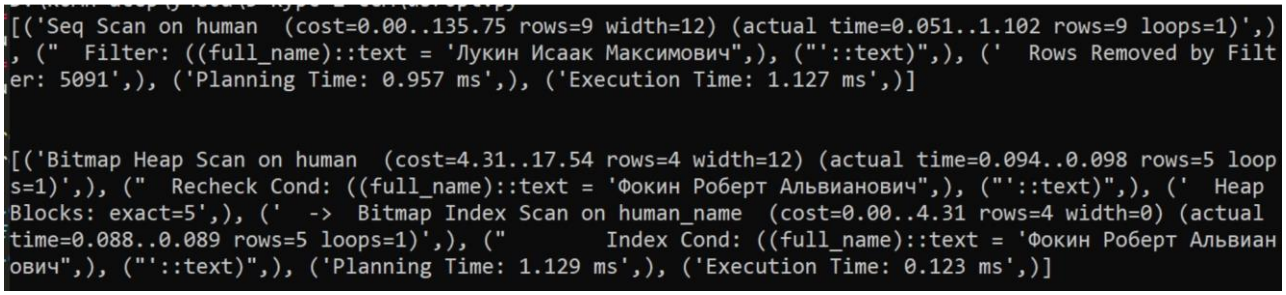


Рис. 44. 1 запрос

До оптимизации выполняется последовательное сканирование, выбираются записи, удовлетворяющие условию. После оптимизации выбран тип сканирования таблицы – bitmap index scan. Bitmap Index Scan создает битовую карту, где каждой странице таблицы будет соответствовать один бит. Все эти биты будут установлены в 0. Затем, Bitmap Index Scan устанавливает некоторые биты в 1, в зависимости от того, на какой странице таблицы может находиться строка, которую нужно вернуть. Затем Bitmap Heap Scan считывает «помеченные» строки. Время выполнения запроса улучшилось.

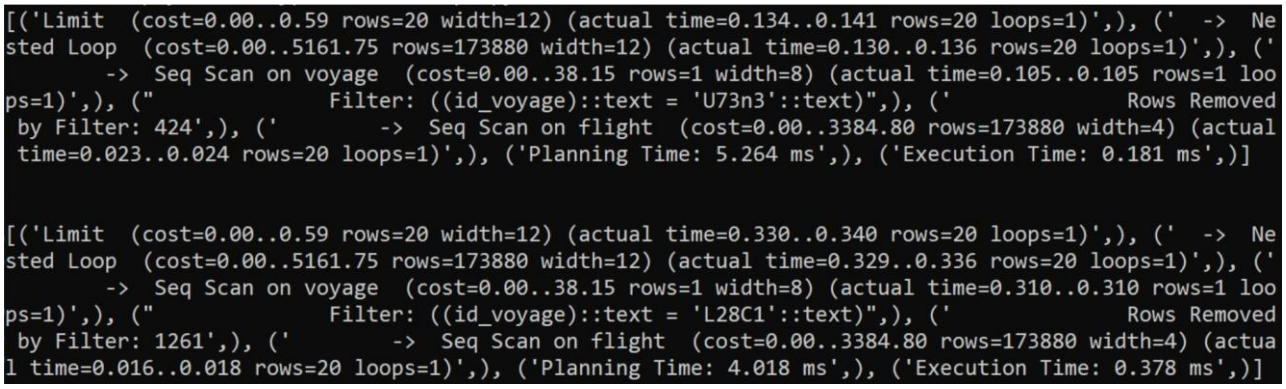


Рис. 45. 2 запрос

В данном случае план выполнения запроса не изменился после оптимизации. Внутри узла limit выполняется внутренний цикл, куда поступают данные от одного потомка – последовательного сканирования, затем проводится последовательное сканирование непосредственно из нужной таблицы.

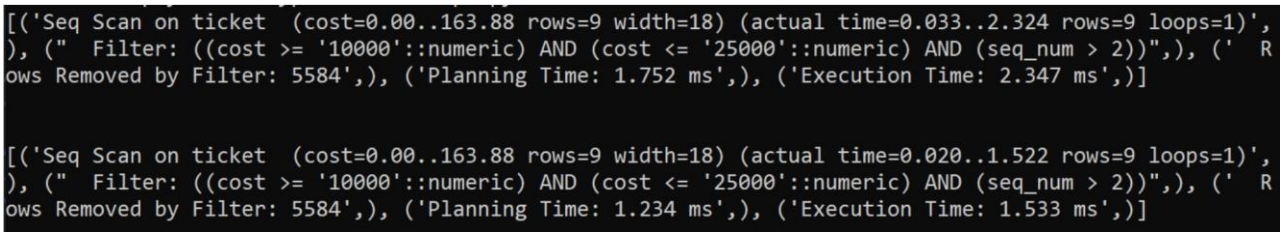


Рис. 46. 3 запрос

В данном случае планировщик также не меняет плана после оптимизации, т.к. условия запроса не используют аргументов и создание индексов не затрагивают поля, указанные в условии. Выполняется последовательное сканирование строк с выбором тех, которые удовлетворяют условию.

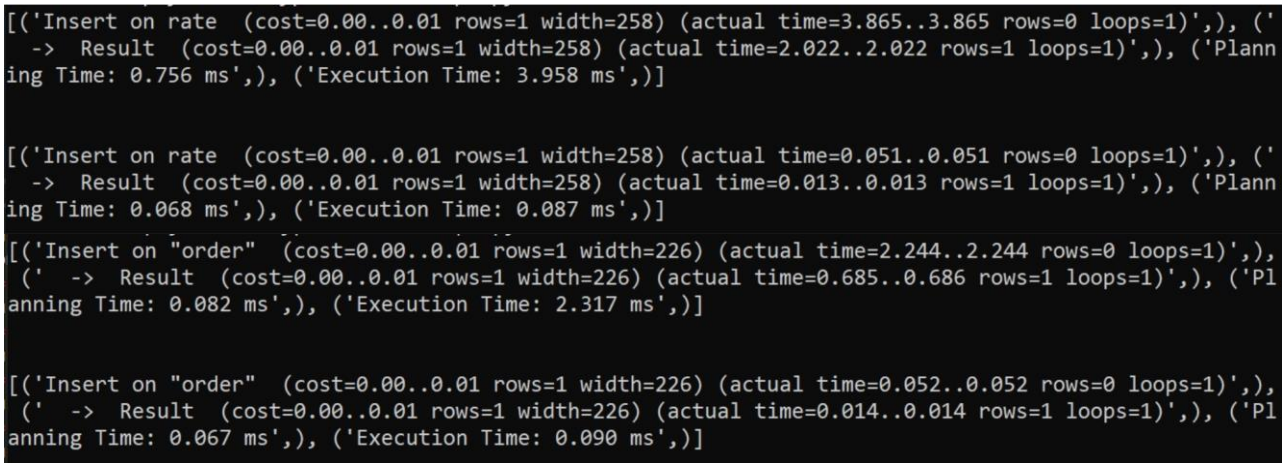


Рис. 47. 4,5 запрос

Когда выполняется команда Insert, собственно изменение данных в таблице происходит в узле верхнего уровня Insert. Узлы плана более низких уровней выполняют работу по нахождению старых строк или вычислению новых данных. Узел, изменяющий данные, может выполняться значительное время, но планировщик не учитывает эту работу в оценке общей стоимости. Это связано с тем, что эта работа будет одинаковой при любом правильном плане запроса, и поэтому на выбор плана она не влияет.

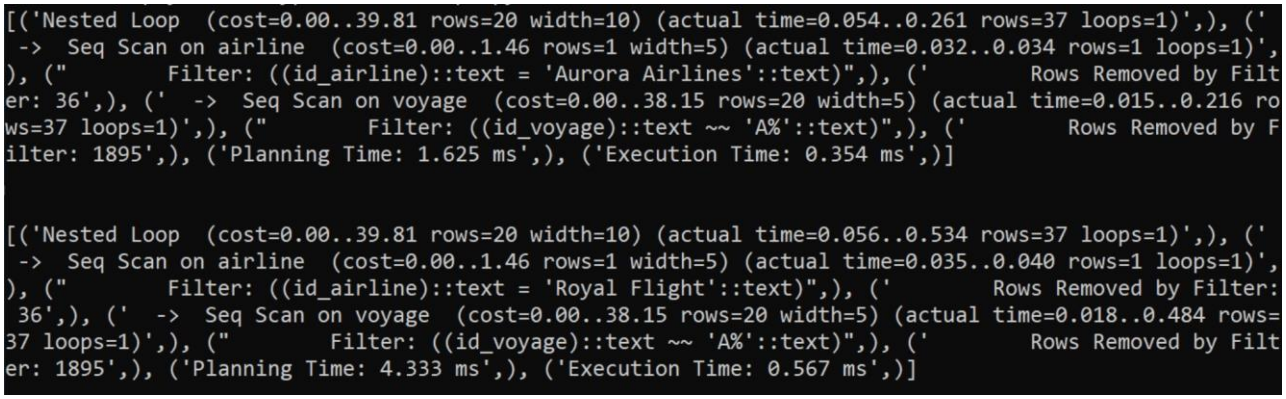


Рис.48. 6 запрос

Этот план аналогичен плану запроса 2 без узла Limit.

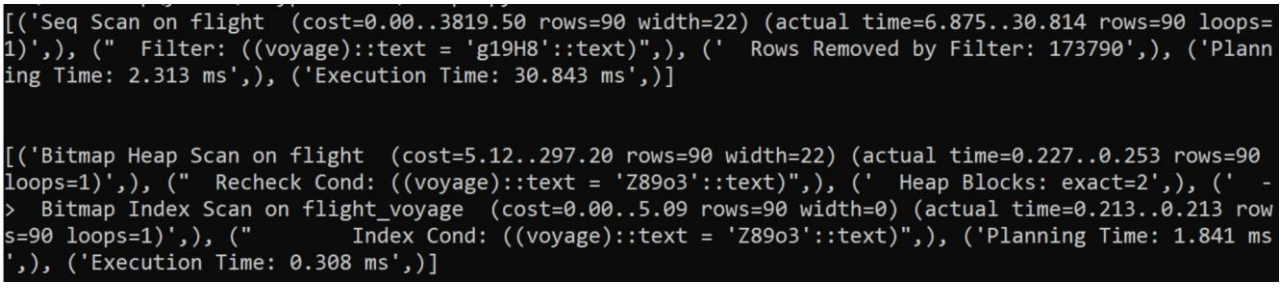


Рис. 49. 7 запрос

Этот план аналогичен плану запроса 1. Время выполнения запроса после оптимизации значительно уменьшилось.

```
[('Seq Scan on ticket  (cost=0.00..149.89 rows=1 width=4) (actual time=0.031..1.155 rows=3 loops=1)',)
, ("  Filter: ((trip_id < 100) AND ((place)::text = '13F'::text))",), ('  Rows Removed by Filter: 5590
',), ('Planning Time: 0.981 ms',), ('Execution Time: 1.181 ms',)]

[('Index Scan using ticket_place on ticket  (cost=0.14..8.16 rows=1 width=4) (actual time=0.066..0.066
 rows=0 loops=1)',), ("  Index Cond: ((place)::text = '28E'::text)",), ('Planning Time: 1.926 ms',), (
'Execution Time: 0.098 ms',)]
```

Рис.50. 8 запрос

В плане до оптимизации используется последовательное сканирование, после оптимизации – индексное сканирование. Время выполнения запроса также уменьшилось.

```
                                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Group  (cost=1399.90..1400.67 rows=103 width=62) (actual time=11.444..11.475 rows=69 loops=1)
   Group Key: query.human_document, query.full_name
   ->  Sort  (cost=1399.90..1400.16 rows=103 width=62) (actual time=11.442..11.448 rows=69 loops=1)
         Sort Key: query.human_document, query.full_name
         Sort Method: quicksort  Memory: 34kB
         ->  Subquery Scan on query  (cost=1388.20..1396.46 rows=103 width=62) (actual time=11.077..11.320 rows=69 loops=1)
               ->  GroupAggregate  (cost=1388.20..1395.43 rows=103 width=78) (actual time=11.075..11.307 rows=69 loops=1)
                     Group Key: human.human_document, (date_part('month'::text, flight.date_from))
                     Filter: (count(*) >= 2)
                     Rows Removed by Filter: 142
                     ->  Sort  (cost=1388.20..1388.97 rows=310 width=70) (actual time=11.041..11.071 rows=330 loops=1)
                           Sort Key: human.human_document, (date_part('month'::text, flight.date_from))
                           Sort Method: quicksort  Memory: 71kB
                           ->  Nested Loop  (cost=283.69..1375.37 rows=310 width=70) (actual time=4.439..9.228 rows=330 loops=1)
                                 ->  Hash Join  (cost=283.27..351.56 rows=310 width=66) (actual time=4.372..5.881 rows=330 loops=1)
                                       Hash Cond: ((ticket.human_document)::text = (human.human_document)::text)
                                       ->  Seq Scan on ticket  (cost=0.00..60.95 rows=2795 width=16) (actual time=0.032..0.651 rows=2795 loops=1)
                                       ->  Hash  (cost=269.25..269.25 rows=1122 width=62) (actual time=4.296..4.300 rows=1102 loops=1)
                                             Buckets: 2048  Batches: 1  Memory Usage: 118kB
                                             ->  Seq Scan on human  (cost=0.00..269.25 rows=1122 width=62) (actual time=0.021..3.741 rows=1102 loops=1)
                                                   Filter: ((human_document)::text ~~ '2%'::text)
                                                   Rows Removed by Filter: 8998
                                 ->  Index Scan using flight_pkey on flight  (cost=0.42..3.30 rows=1 width=12) (actual time=0.009..0.009 rows=1 loops=330)
                                       Index Cond: (id_flight = ticket.flight)
                                       Filter: (date_from >= (CURRENT_DATE - 365))
 Planning Time: 10.238 ms
 Execution Time: 11.785 ms
```

```
                                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Group  (cost=432.84..432.91 rows=9 width=62) (actual time=5.693..5.696 rows=5 loops=1)
   Group Key: query.human_document, query.full_name
   ->  Sort  (cost=432.84..432.87 rows=9 width=62) (actual time=5.692..5.692 rows=5 loops=1)
         Sort Key: query.human_document, query.full_name
         Sort Method: quicksort  Memory: 25kB
         ->  Subquery Scan on query  (cost=431.96..432.70 rows=9 width=62) (actual time=5.651..5.665 rows=5 loops=1)
               ->  GroupAggregate  (cost=431.96..432.61 rows=9 width=78) (actual time=5.650..5.662 rows=5 loops=1)
                     Group Key: human.human_document, (date_part('month'::text, flight.date_from))
                     Filter: (count(*) >= 2)
                     Rows Removed by Filter: 7
                     ->  Sort  (cost=431.96..432.03 rows=28 width=70) (actual time=5.623..5.625 rows=19 loops=1)
                           Sort Key: human.human_document, (date_part('month'::text, flight.date_from))
                           Sort Method: quicksort  Memory: 27kB
                           ->  Nested Loop  (cost=270.94..431.28 rows=28 width=70) (actual time=3.975..5.528 rows=19 loops=1)
                                 ->  Hash Join  (cost=270.52..338.81 rows=28 width=66) (actual time=3.865..4.913 rows=19 loops=1)
                                       Hash Cond: ((ticket.human_document)::text = (human.human_document)::text)
                                       ->  Seq Scan on ticket  (cost=0.00..60.95 rows=2795 width=16) (actual time=0.021..0.505 rows=2795 loops=1)
                                       ->  Hash  (cost=269.25..269.25 rows=102 width=62) (actual time=3.720..3.723 rows=89 loops=1)
                                             Buckets: 1024  Batches: 1  Memory Usage: 17kB
                                             ->  Seq Scan on human  (cost=0.00..269.25 rows=102 width=62) (actual time=0.020..3.590 rows=89 loops=1)
                                                   Filter: ((human_document)::text ~~ '45%'::text)
                                                   Rows Removed by Filter: 10011
                                 ->  Index Scan using fi on flight  (cost=0.42..3.30 rows=1 width=12) (actual time=0.029..0.030 rows=1 loops=19)
                                       Index Cond: (id_flight = ticket.flight)
                                       Filter: (date_from >= (CURRENT_DATE - 365))
 Planning Time: 13.878 ms
 Execution Time: 6.024 ms
```

Рис.51. 9 запрос

В этом плане для выбора людей с указанным документом используется последовательное сканирование, для выборки полетов за последний год – индексное сканирование, для выбора билетов, соответствующих выбранным документам – последовательное сканирование. Затем выполняется два узла группировки данных с соответствующими условиями.

Данный запрос оказался сложным в оптимизации.

4.4 Вывод

В данной лабораторной работе проведена работа по оптимизации запросов. Самым сложным было анализировать результаты explain. Лучшим видом оптимизации оказался index, так как при его применении результат виден сильнее, и он применим к конкретным столбцам таблицы, что помогает ускорить поиск по ней. Примеры до и после оптимизации с index различны на порядок, в то время как с prepare разница менее заметна.