

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

Институт информатики, математики и электроники
Факультет информатики
Кафедра суперкомпьютеров и общей информатики

Отчет по лабораторной работе № 1

Дисциплина: “Databases in Enterprise Systems”

(«Корпоративные базы данных»)

Выполнил: Мелешенко И. С.

Группа: 6133-010402D

Самара 2022

СОДЕРЖАНИЕ

Задание на лабораторную работу №1	4
1. Выбор и описание ER-модели	5
2. Разработать ER модель, включающую минимум 5-6 сущностей и типы связей: 1-N, N:M, 1-1	6
2.1 Описание сущностей.	6
2.1.1 Сущность ActiveAircraft.....	6
2.1.2 Сущность InformationAircraft	7
2.1.3 Сущность Airport.....	8
2.1.4 Сущность Runway	9
2.1.5 Сущность Aircompany	10
2.1.6 Сущность Passenger	10
2.1.7 Сущность ClientAircompany	11
2.2 Описание связей между сущностями.....	12
2.2.1 Связь типа «Один к одному»	12
2.2.2 Связь типа «Один к многим».....	13
2.2.3 Связь типа «Многие ко многим».....	13
3 - 4. Создать базу данных по модели в СУБД PostgreSQL. Определить индексы, уникальные индексы.	14
3.1 Создание базы данных.....	14
3.2 Создание схемы ER-модели данных	14
4.1 Установка индексов	15
5. Разработать типовые запросы к СУБД на языке SQL. Получение списков данных. Агрегация. Поиск.	16
5.1 Типовые запросы к базе данных.....	16
5.2 Запросы получения списков данных и поиска.....	18
5.3 Запрос агрегации	19

6-7. Разработайте хранимые процедуры на языке PL/pgSQL для генерации случайных данных для базы данных. Сгенерируйте тестовые данные при помощи разработанных процедур.	21
6.1 Характеристика классов, описывающих сущности ER-модели	21
6.2 Характеристика класса, содержащего данные для случайной генерации объектов	24
6.3 Характеристика класса, содержащего точку входа в программу	24
7.1 Генерация тестовых данных.	24
8 – 9. Протестируйте работу запросов на больших объёмах данных (Порядка 1 миллиона записей в основных таблицах). Измените конфигурацию сервера PostgreSQL для достижения лучшей производительности на самых медленных запросах. Оптимизируйте схему БД и запросы для достижения лучшей производительности.	26
Заключение	32

Задание на лабораторную работу №1

1. Выбрать предметную область
2. Разработать ER модель, включающую минимум 5-6 сущностей и типы связей: 1-N, N:M, 1-1.
3. Создать базу данных по модели в СУБД PostgreSQL.
4. Определить индексы, уникальные индексы.
5. Разработать типовые запросы к СУБД на языке SQL. Получение списков данных. Агрегация. Поиск.
6. Разработайте хранимые процедуры на языке PL/pgSQL для генерации случайных данных для базы данных.
7. Сгенерируйте тестовые данные при помощи разработанных процедур.
8. Протестируйте работу запросов на больших объёмах данных (Порядка 1 миллиона записей в основных таблицах).
9. Измените конфигурацию сервера PostgreSQL для достижения лучшей производительности на самых медленных запросах. Оптимизируйте схему БД и запросы для достижения лучшей производительности.

Пункты 6-7 допустимо реализовывать другими способами без PL/pgSQL

1. Выбор и описание ER-модели

Для работы с базой данных была придумана ER-модель на основе бизнес-модели авиационных перевозок. Данная модель состоит из двух видов сущностей и трех типов связей. Схема модели представлена на рисунке 1.

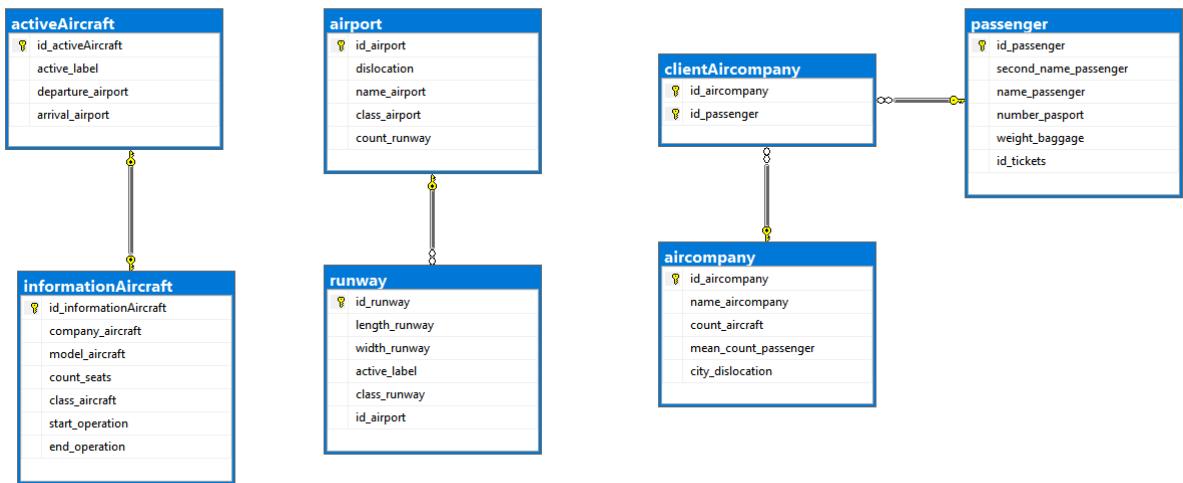


Рисунок 1 – Схема ER-модели бизнес-процессса

2. Разработать ER модель, включающую минимум 5-6 сущностей и типы связей: 1-N, N:M, 1-1.

В моей модели реализовано 7 сущностей, которые описывают те или иные объекты/субъекты авиационных перевозок. Рассмотрим их по подробнее.

2.1 Описание сущностей.

В качестве сущностей ER-модели выбраны те или иные реально существующие объекты модели авиационных перевозок. Описываемая модель включает следующие сущности:

- ActiveAircraft
- InformationAircraft
- Airport
- Runway
- Aircompany
- Passenger
- ClientAircompany

Далее рассмотрим каждую более подробно.

2.1.1 Сущность ActiveAircraft

Данная сущность необходима для описания действующий самолет. Сущность модели представлена на рисунке 2.

activeAircraft	
	id_activeAircraft
	active_label
	departure_airport
	arrival_airport

Рисунок 2 – Сущность ActiveAircraft

Рассмотрим их описание и характеристику.

Таблица 1 – Описание полей сущности ActiveAircraft

Название поля	Тип данных	Описание	Ключ
id_activeAircraft	int	Является уникальным идентификатором, который используется для однозначной маркировки записей в таблице.	Primary Key
active_label	bool	Является маркером действительности самолета. Если выбрано значение true, то самолет действителен может летать, вследствие чего должен иметь место вылета и место прилета. Если выбрано значение false самолет считается не действительным и летать не может, а в места вылета и прилета имеют значения NULL.	—
departure_airport	varchar() string	Является полем, содержащим информацию о месте вылета (город) самолета, в случае если он действителен.	—
arrival_airport	varchar() string	Является полем, содержащим информацию о месте прилета (город) самолета, в случае если он действителен.	—

2.1.2 Сущность InformationAircraft

Данная сущность необходима для более полного описания действующего самолета и представлена на рисунке 3.

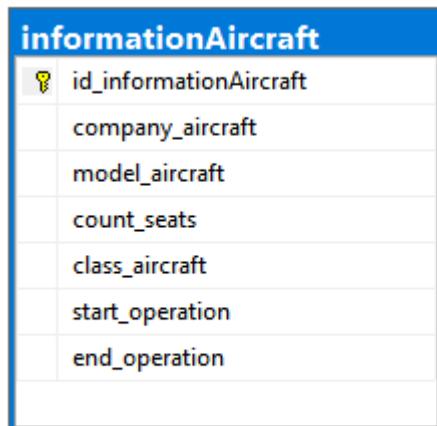


Рисунок 3 – Сущность InformationAircraft

Рассмотрим их описание и характеристику.

Таблица 2 – Описание полей сущности InformationAircraft

Название поля	Тип данных	Описание поля	Ключ
id_informationAircraft	int	Является уникальным идентификатором, который используется для однозначной маркировки записей в таблице.	Primary Key/Foreign Key

Название поля	Тип данных	Описание поля	Ключ
company_aircraft	VARCHAR() String	Является полем, содержащим информацию о производителе самолета.	—
model_aircraft	VARCHAR() String	Является полем, содержащим информацию о модели самолета.	—
count_seats	int	Является полем, содержащим информацию о количестве мест в самолете.	—
class_aircraft	VARCHAR() String	Является полем, содержащим информацию о классе самолета.	—
start_operation	Date	Является полем, содержащим информацию о дате начала эксплуатации самолета.	—
end_operation	Date	Является полем, содержащим информацию о дате окончания эксплуатации самолета.	—

2.1.3 Сущность Airport

Данная сущность необходима для описания аэропорта и представлена на рисунке 4.

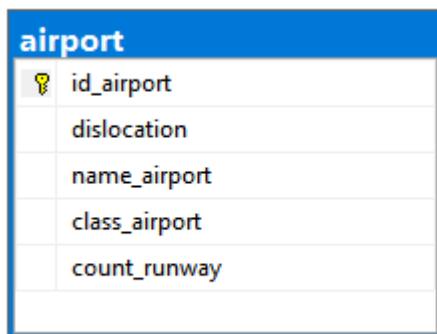


Рисунок 4 – Сущность Airport

Рассмотрим их описание и характеристику.

Таблица 3 – Описание полей сущности Airport

Название поля	Тип данных	Описание	Ключ
id_airport	int	Является уникальным идентификатором, который используется для однозначной маркировки записей в таблице.	Primary Key
dislocation	VARCHAR() string	Является полем, содержащим информацию о городе дислокации аэропорта.	—
name_airport	VARCHAR() string	Является полем, содержащим информацию	—

Название поля	Тип данных	Описание	Ключ
		о имени аэропорта.	
class_airport	VARCHAR() string	Является полем, содержащим информацию о типе/классе аэропорта.	—
count_runway	int	Является полем, содержащим информацию о количестве взлетных полос в аэропорту.	—

2.1.4 Сущность Runway

Данная сущность необходима для описания летной полосы аэропорта и представлена на рисунке 5.

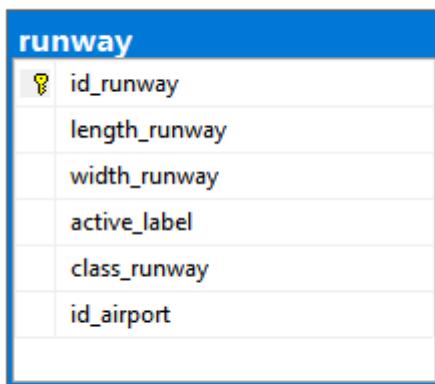


Рисунок 5 – Сущность Runway

Рассмотрим их описание и характеристику.

Таблица 4 – Описание полей сущности Runway

Название поля	Тип данных	Описание	Ключ
id_runway	int	Является уникальным идентификатором, который используется для однозначной маркировки записей в таблице.	Primary Key
length_runway	int	Является полем, содержащим информацию о длине взлетной полосы.	—
width_runway	int	Является полем, содержащим информацию о ширине взлетной полосы.	—
active_label	bool	Является маркером действительности взлетной полосы. Если выбрано значение true, то полоса доступна для работы, иначе не доступна.	—
class_runway	int	Является полем, содержащим информацию о классе взлетной полосы.	—

Название поля	Тип данных	Описание	Ключ
id_airport	int	Является полем, содержащим информацию об аэропорте, в котором находится взлетная полоса.	Foreign Key

2.1.5 Сущность Aircompany

Данная сущность необходима для описания авиакомпаний, осуществляющие авиаперевозки и представлена на рисунке 6.



Рисунок 6 – Сущность Aircompany

Рассмотрим их описание и характеристику.

Таблица 5 – Описание полей сущности Aircompany

Название поля	Тип данных	Описание	Ключ
id_aircompany	int	Является уникальным идентификатором, который используется для однозначной маркировки записей в таблице.	Primary Key
name_aircompany	VARCHAR() string	Является полем, содержащим информацию о имени авиакомпании.	—
count_aircraft	int	Является полем, содержащим информацию о количестве самолетов в компании	—
mean_count_passenger	int	Является полем, содержащим информацию о среднем количестве пассажиров, перевозимых компанией.	—
city_dislocation	VARCHAR() string	Является полем, содержащим информацию о городе дислокации авиакомпании.	—

2.1.6 Сущность Passenger

Данная сущность необходима для описания клиентов авиакомпаний – пассажиров, и представлена на рисунке 7.

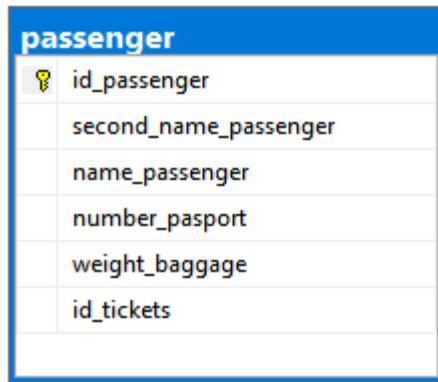


Рисунок 7 – Сущность Passenger

Рассмотрим их описание и характеристику.

Таблица 6 – Описание полей сущности Passenger

Название поля	Тип данных	Описание	Ключ
id_passenger	int	Является уникальным идентификатором, который используется для однозначной маркировки записей в таблице.	Primary Key
second_name_passenger	VARCHAR() String	Является полем, содержащим информацию о фамилии пассажира.	—
name_passenger	VARCHAR() String	Является полем, содержащим информацию о имени пассажира.	—
number_pasport	int	Является полем, содержащим информацию о номере паспорта пассажира.	—
weight_baggage	int	Является полем, содержащим информацию о весе багажа пассажира.	—
id_tickets	int	Является полем, содержащим информацию о номере билета пассажира.	—

2.1.7 Сущность ClientAircompany

Данная сущность необходима для описания связи между пассажиром и авиакомпанией, и представлена на рисунке 8.

passenger	
	id_passenger
	second_name_passenger
	name_passenger
	number_pasport
	weight_baggage
	id_tickets

Рисунок 8 – Сущность Passenger

Рассмотрим их описание и характеристику.

Таблица 7 – Описание полей сущности Passenger

Название поля	Тип данных	Описание	Ключ
id_aircompany	int	Является уникальным идентификатором, который используется для однозначной маркировки записей в таблице.	Primary Key Foreign Key
id_passenger	int	Является уникальным идентификатором, который используется для однозначной маркировки записей в таблице.	Primary Key Foreign Key

2.2 Описание связей между сущностями

В моей ER-модели реализовано 3 типа связей:

- *1-1 – связь один к одному,*
- *1-N – связь один ко многим,*
- *N:M – связь многие ко многим.*

Далее рассмотрим каждый тип связи, и посмотрим какие сущности каким типом связи связаны.

2.2.1 Связь типа «Один к одному»

Тип связи	Описание
Один к одному	<p>Данным типом связи, связаны 2 сущности: "действующий" самолет и информация об этом самолете, поскольку не может быть 2-х абсолютно одинаковых самолетов.</p>

2.2.2 Связь типа «Один к многим»

Тип связи	Описание
Один к многим	<p>Данным типом связи, связаны 2 сущности:</p> <p style="text-align: center;"><u>аэропорт</u> и <u>взлетные полосы</u></p> <p>поскольку аэропорт может иметь как одну так и несколько взлетных полос.</p>

2.2.3 Связь типа «Многие ко многим»

Тип связи	Описание
Многие ко многим	<p>Данным типом связи, связаны 2 сущности, через промежуточную третью:</p> <p style="text-align: center;"><u>авиакомпания и пассажир</u> связаны через <u>клиентов авиакомпаний.</u></p> <p>Поскольку, пассажир может быть зарегистрирован не в одной авиакомпании, а авиакомпании явно имеют более одного клиента(пассажира).</p>

3 - 4. Создать базу данных по модели в СУБД PostgreSQL.

Определить индексы, уникальные индексы.

Для последующего выполнения лабораторной работы нам потребуются следующие инструменты:

MS SQL Server 2019 Developer – SQL сервер, на котором мы и будем работать.

SQL Server Management Studio (SSMS) – утилита из Microsoft SQL Server 2019 для конфигурирования, управления и администрирования всех компонентов Microsoft SQL Server.

Далее будет показана, работа с базой данных. Скрипты, упоминаемые в данном отчете, расположены в следующем репозитории на сайте GitHub по ссылке: <https://github.com/Black-Viking-63/EnterpriseDataBase>.

3.1 Создание базы данных

Для создания базы данных в утилите SSMS выполним скрипт со следующей командой, как показано на рисунке 9.

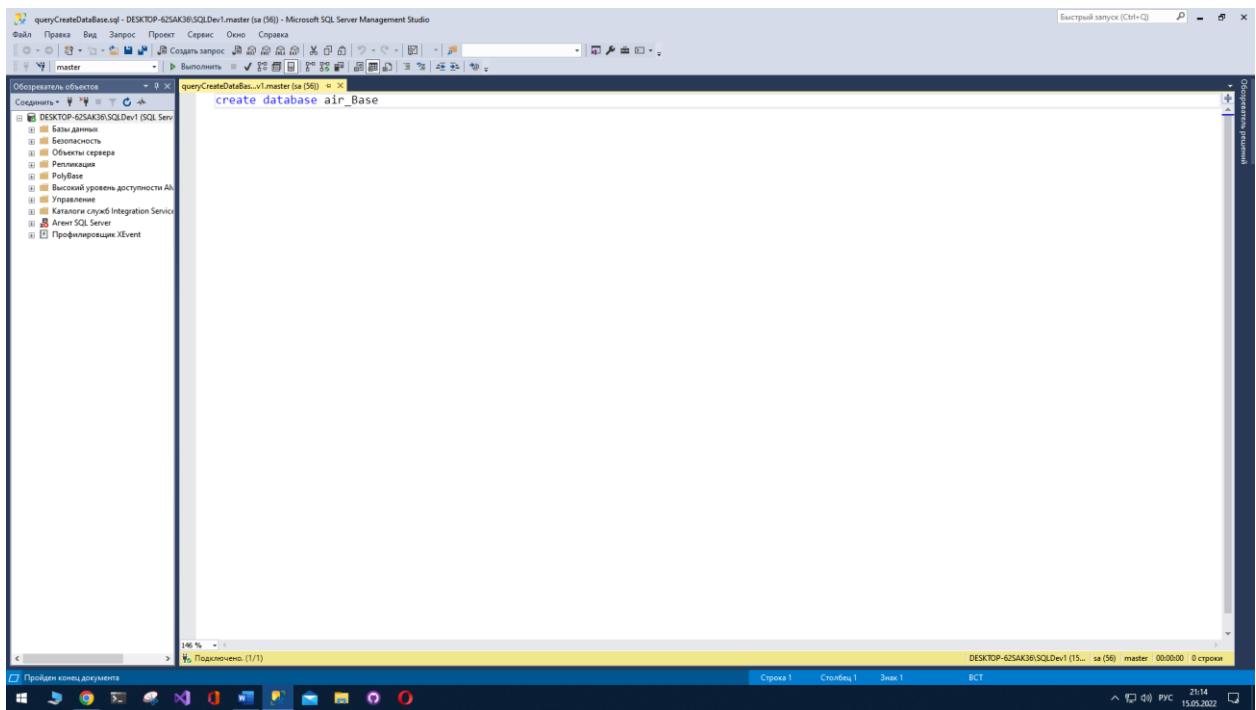


Рисунок 9 – Создание базы данных

3.2 Создание схемы ER-модели данных

Для создания схемы ER-модели данных в утилите SSMS выполним скрипт со следующими командами, как показано на рисунке 10.

```

queryCreateScheme.sql - DESKTOP-62SAK3R\SQLDev1\air_Base (sa (52)) - Microsoft SQL Server Management Studio
Файл Правка Вид Запрос Проект Сервис Окно Справка
Создание > Выполнить > Создать запрос | Быстрый запуск (Ctrl+Q) | Помощник по созданию схемы
queryCreateScheme.sql
-- air_Base
CREATE SCHEMA [air_Base]
GO
CREATE TABLE air_base.dbo.activeAircraft(
    id_activeaircraft int primary key,
    active_label varchar(5) not null,
    departure_airport varchar(50),
    arrival_airport varchar(50)
)
CREATE TABLE air_base.dbo.informationAircraft(
    id_informationaircraft int primary key,
    id_activeaircraft int not null,
    model_aircraft varchar(50) not null,
    count_seats int check(count_seats > 0 and count_seats < 800) not null,
    class_aircraft int not null,
    start_operation Date,
    end_operation Date,
    constraint fk_aircraft_num foreign key (id_informationaircraft) references activeAircraft(id_activeaircraft) on delete cascade on update cascade
)
CREATE TABLE air_base.dbo.airport(
    id_airport int primary key,
    dislocation varchar(50) not null,
    name_airport varchar(50) not null,
    count_passenger int check(count_passenger > 0) not null,
    count_runway int check(count_runway > 0 and count_runway < 10) not null
)
CREATE TABLE air_base.dbo.runway(
    id_runway int primary key,
    length_runway int check(length_runway>300 and length_runway<5500) not null,
    width_runway int check(width_runway>10 and width_runway<105) not null,
    active_label varchar(5) not null,
    class_runway int check(class_runway<0 and class_runway>7) not null,
    id_airport int foreign key references airport(id_airport) on delete cascade on update cascade
)
CREATE TABLE air_base.dbo.aircompany(
    id_aircompany int primary key,
    name_aircompany varchar(50) not null,
    count_aircraft int check(count_aircraft>0 and count_aircraft<1000) not null,
    seat_count_passenger int check(seat_count_passenger>0) not null,
    seat_dif_passenger int check(seat_dif_passenger>0) not null
)
CREATE TABLE air_base.dbo.passenger(
    id_passenger int primary key,
    second_name_passenger varchar(50) not null,
    name_passenger varchar(50) not null,
    number_passport int unique check(number_passport>100000 and number_passport<999999) not null,
    weight_tickets int check(weight_tickets>25 and weight_tickets<50) not null,
    id_tickets int unique check(id_tickets > 10000000 and id_tickets < 9999999)
)
CREATE TABLE air_base.dbo.clientsAircompany(
    id_aircompany int foreign key references aircompany(id_aircompany) on delete cascade on update cascade,
    id_passenger int foreign key references passenger(id_passenger) on delete cascade on update cascade,
    Primary key (id_aircompany, id_passenger)
)

```

Рисунок 10 – Создание схемы ER-модели данных

4.1 Установка индексов

В качестве уникального индекса для каждой сущности был выбран его уникальный идентификационный номер – id.

5. Разработать типовые запросы к СУБД на языке SQL. Получение списков данных. Агрегация. Поиск.

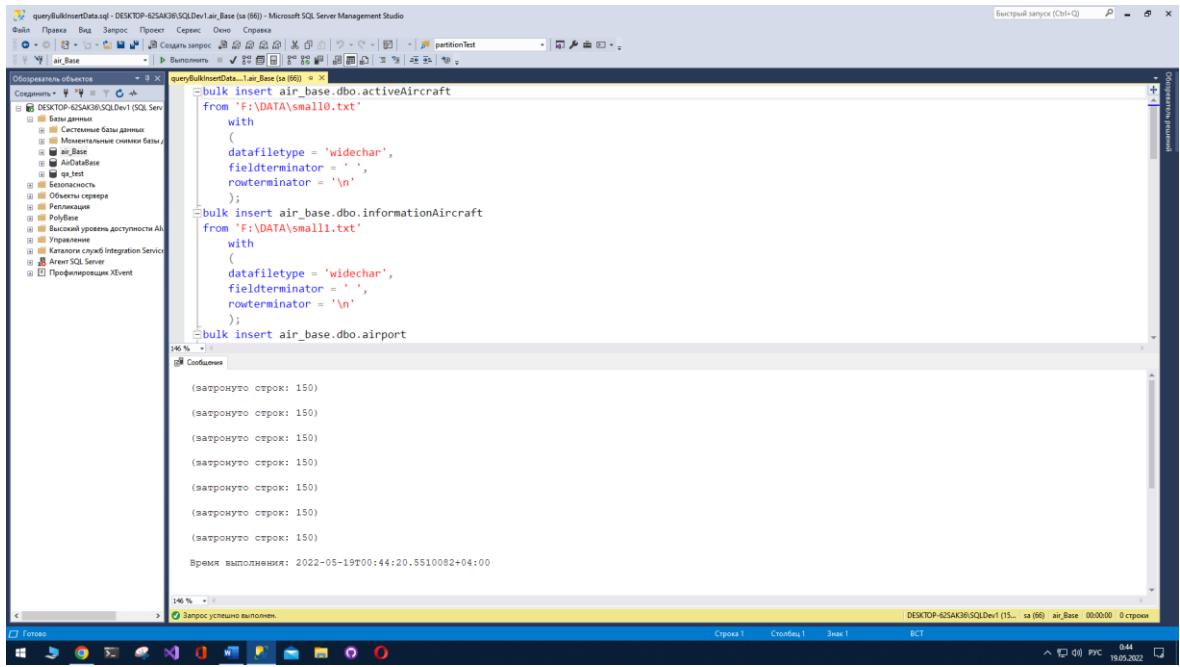
5.1 Типовые запросы к базе данных

К типовым запросам относят следующие запросы:

- insert

Данный тип запроса производит добавление данных в таблицу посредством добавления или вставки новой строки в конец таблицы. Например, запрос вставки новых данных в таблицу activeAircraft
insert into activeAircraft values(@id, 'True', 'Москва', 'Самара'). Однако в своей работе я использовал модифицированный запрос insert: bulk insert. Данный запрос производит массовую вставку данных из файла, например, из текстового файла, формата txt. Однако в этом случае данные должны быть форматированы, например, разделителем слов является пробел, а разделителем строк является символ переноса строки, и файл указанного формата сохранен с использованием кодировки utf-16.

Выполнение данного запроса представлено на рисунке 11.



The screenshot shows the Microsoft SQL Server Management Studio interface. A query window titled 'queryBulkInsertData.sql - DESKTOP-625A40B\SQLDev1_air_Base (sa (86)) - Microsoft SQL Server Management Studio' displays the following T-SQL code:

```
bulk insert air_base.dbo.activeAircraft
from 'F:\DATA\small10.txt'
with
(
    datafiletype = 'widechar',
    fieldterminator = ' ',
    rowterminator = '\n'
);
bulk insert air_base.dbo.informationAircraft
from 'F:\DATA\small11.txt'
with
(
    datafiletype = 'widechar',
    fieldterminator = ' ',
    rowterminator = '\n'
);
bulk insert air_base.dbo.airport
```

Below the code, the status bar indicates: 'Запрос успешно выполнен.' (Query successfully executed). The bottom right corner shows the system tray with icons for network, battery, and date/time.

Рисунок 11 – Выполнение запроса bulk insert.

- update

Данный тип запроса производит обновление записей в таблице, посредством изменения значений в строке или столбце таблицы. Например, запрос обновления данных в таблице airport

```
update airport set airport.class_airport = @class_airport  
where airport.dislocation like @dislocation and airport.name_airport like @name
```

Выполнение данного запроса представлено на рисунке 12.

The screenshot shows the Microsoft SQL Server Management Studio interface. A query window titled 'queryUpdateRowAirport.sql' is open, displaying the following T-SQL code:

```
--DECLARE @count_runway int = 0,  
--@class_airport varchar(50) = 'Военный',  
--@dislocation varchar(50) = 'Омск',  
--@name varchar(50) = 'Гумрак',  
--@id int = 0;  
-- обновление класса самолета по имени и дислокации самолета  
UPDATE air_base.dbo.airport  
SET air_base.dbo.airport.class_airport = @class_airport  
WHERE air_base.dbo.airport.dislocation like @dislocation AND air_base.dbo.airport.name_airport like @name;  
-- обновление числа полос по классу самолета  
UPDATE air_base.dbo.airport  
SET air_base.dbo.airport.count_runway = @count_runway  
WHERE air_base.dbo.airport.class_airport like @class_airport  
-- обновление класса самолета по id  
UPDATE air_base.dbo.airport  
SET air_base.dbo.airport.class_airport = @class_airport  
WHERE air_base.dbo.airport.id_airport = @id
```

Below the code, a note states '(затронута одна строка)'. At the bottom of the window, it says 'Время выполнения: 2022-05-19T00:56:14.9314753+04:00'.

The status bar at the bottom right shows: DESKTOP-625AK36\SQLDev1\15... sa (66) air_Base | 00:00:00 | 0 строк | Строки | Столбцы | Йоды | ВСТ | 0:56 | 18.05.2022 | РУС |

Рисунок 12 – Выполнение типового запроса update.

- delete

Данный тип запроса производит удаление данных из таблицы. Удаление может производиться несколькими путями: удаление конкретной строки, нескольких строк объединенных определенным условием, так и полное удаление всех данных из таблицы. Например,

```
delete from informationAircraft where informationAircraft.id_informationAircraft = @id;
```

Выполнение данного запроса представлено на рисунке 13.

The screenshot shows the Microsoft SQL Server Management Studio interface. A query window titled 'queryDeleteRowInformationAircraft.sql' is open, displaying a DELETE statement. The code is as follows:

```
--Полное удаление  
--delete from air_base.dbo.informationAircraft  
--@id_informationAircraft  
declare @id int = 8;  
delete from air_base.dbo.informationAircraft where air_base.dbo.informationAircraft.id_informationAircraft = @id;  
delete from air_base.dbo.informationAircraft where air_base.dbo.informationAircraft.id_informationAircraft > @id;  
delete from air_base.dbo.informationAircraft where air_base.dbo.informationAircraft.id_informationAircraft < @id;  
delete from air_base.dbo.informationAircraft where air_base.dbo.informationAircraft.id_informationAircraft >= @id;  
delete from air_base.dbo.informationAircraft where air_base.dbo.informationAircraft.id_informationAircraft <= @id;  
--company_aircraft  
declare @company_aircraft varchar(50) = '';  
delete from air_base.dbo.informationAircraft where air_base.dbo.informationAircraft.company_aircraft like @company_aircraft;  
delete from air_base.dbo.informationAircraft where air_base.dbo.informationAircraft.company_aircraft not like @company_aircraft;  
--count_seats  
declare @count_seats int = 0;  
delete from air_base.dbo.informationAircraft where air_base.dbo.informationAircraft.count_seats= @count_seats;  
delete from air_base.dbo.informationAircraft where air_base.dbo.informationAircraft.count_seats > @count_seats;
```

The status bar at the bottom indicates the query was successfully executed. The database object tree on the left shows the 'air_Base' database structure.

Рисунок 13 – Выполнение типового запроса delete.

5.2 Запросы получения списков данных и поиска

Данные запросы имеют единый тип – запросы поиска, только различие в накладываемых ограничениях. Для получения списка данных происходит наложение менее строгих ограничений, потому что необходимо получить несколько строк данных. Для поиска конкретного элемента (строки), необходимо производить наложение более строгих ограничений. Приведем примеры подобных запросов.

Например, для получения списка данных мы можем использовать следующий запрос, который осуществляет поиск пассажиров, у которых вес багажа соответствует указанному:

```
select * from passenger where passenger.weight_baggage = @weight;
```

в результате выполнения данного запроса, явно будет выведена не одна строка, а некоторый список.

Выполнение данного запроса представлено на рисунке 14.

The screenshot shows the Microsoft SQL Server Management Studio interface. A query named 'querySearchRowPassenger.sql' is being run against the 'air_Base' database. The code is a cursor-based search for passengers with a specific weight. The results are displayed in a table:

	id_passenger	Фамилия_пассажира	Имя_пассажира	Номер_паспорта	weight_baggage	id_ticks
1	57	Сергей	Яков	1000057	15	1000057
2	60	Борис	Светлана	1000060	15	1000060
3	82	Владимир	Петрович	1000082	15	1000082
4	106	Миниева	Рекс	1000106	15	1000106
5	109	Роджер	Григорий	1000109	15	1000109
6	135	Владик	Софья	1000135	15	1000135

Рисунок 14 – Выполнение запроса получения списка данных.

В это же время для более конкретного и детального поиска элемента таблицы (строки), необходимо накладывать более серьезные ограничения, например, производить поиск по уникальному индексу который даст однозначный ответ:

```
select * from airport where airport.id_airport = @id_airport.
```

Выполнение данного запроса представлено на рисунке 15.

The screenshot shows the Microsoft SQL Server Management Studio interface. A query named 'querySearchRowAirport.sql' is being run against the 'air_Base' database. The code is a cursor-based search for an airport by its ID. The results are displayed in a table:

	id_airport	дисклокация	название_аэропорта	класс_аэропорта	count_跑道
1	19	Изюм	Сокол	Международный	1

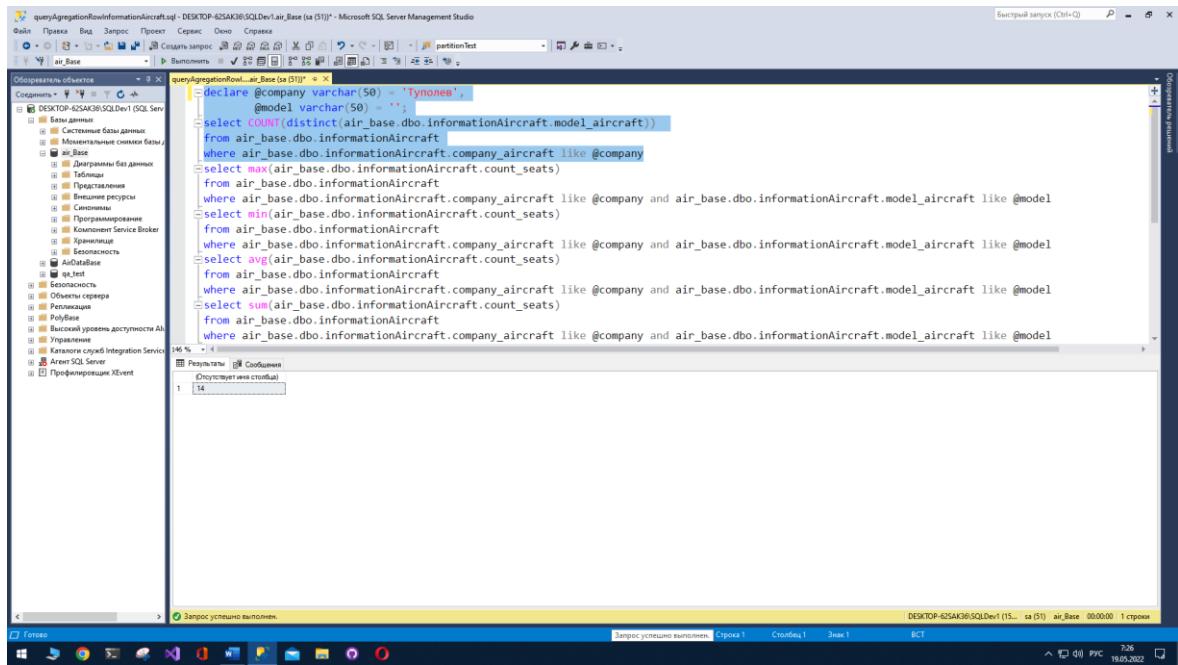
Рисунок 15 – Выполнение запроса поиска.

5.3 Запрос агрегации

Данный тип запроса позволяет производить агрегирование данных, посредством поисков минимального, среднего, максимального числовых значений, а также суммирования числовых данных, в рамках заданного условия.

```
select COUNT(distinct(informationAircraft.model_aircraft))
from informationAircraft
where informationAircraft.company_aircraft like @company
```

Выполнение данного запроса представлено на рисунке 16.



```
queryAggregationRow_air_Base (sa (5)) - DESKTOP-G2SAK36\SQLDev1\air_Base (sa (5)) - Microsoft SQL Server Management Studio

declare @company varchar(50) = 'Туполев';
        @model varchar(50) = '';
select COUNT(distinct air_base.dbo.informationAircraft.model_aircraft)
from air_base.dbo.informationAircraft
where air_base.dbo.informationAircraft.company_aircraft like @company
select max(air_base.dbo.informationAircraft.count_seats)
from air_base.dbo.informationAircraft
where air_base.dbo.informationAircraft.company_aircraft like @company and air_base.dbo.informationAircraft.model_aircraft like @model
select min(air_base.dbo.informationAircraft.count_seats)
from air_base.dbo.informationAircraft
where air_base.dbo.informationAircraft.company_aircraft like @company and air_base.dbo.informationAircraft.model_aircraft like @model
select avg(air_base.dbo.informationAircraft.count_seats)
from air_base.dbo.informationAircraft
where air_base.dbo.informationAircraft.company_aircraft like @company and air_base.dbo.informationAircraft.model_aircraft like @model
select sum(air_base.dbo.informationAircraft.count_seats)
from air_base.dbo.informationAircraft
where air_base.dbo.informationAircraft.company_aircraft like @company and air_base.dbo.informationAircraft.model_aircraft like @model
```

1	14
Результаты	[Сообщения]

Рисунок 16 – Выполнение запроса поиска.

6-7. Разработайте хранимые процедуры на языке PL/pgSQL для генерации случайных данных для базы данных. Сгенерируйте тестовые данные при помощи разработанных процедур.

Поскольку в задание сказано, что выполнение данного задания допустимо производить без использования PL/pgSQL, то мы так и поступим. Для генерации большого числа данных было разработано консольное приложение на высокоуровневом языке программирования C#.

Для разработки указанного приложения использовалась среда разработки Visual Studio 2019 с языком C#.

В результате разработки приложения было описано несколько классов:

- классы, описывающие сущности ER-модели;
- классы, содержащие необходимые данные для случайного генерирования объектов;
- класс содержащий точку входа в программу.

6.1 Характеристика классов, описывающих сущности ER-модели

Каждый класс, описывающий сущность, имеет в своей структуре следующие элементы:

- поля;
- конструктор и методы доступа;
- метод случайной генерации объекта;
- метод вывода объекта, согласно заданному формату.

Поля классов, совпадают со столбцами сущностей, описанными в схеме базы данных, и дают некоторые характеристики объекту, которые они описывают, как на рисунке 17.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace AirBase.Classes.Models
8  {
9      class clsInfoAircraft
10     {
11         private int id_info;
12         private string company_aircraft;
13         private string model_aircraft;
14         private int count_seats;
15         private string class_aircraft;
16         private DateTime start_operation;
17         private DateTime end_operation;
18
19         public int Id_info { get => id_info; set => id_info = value; }
20         public string Company_aircraft { get => company_aircraft; set => company_aircraft = value; }
21         public string Model_aircraft { get => model_aircraft; set => model_aircraft = value; }
22         public int Count_seats { get => count_seats; set => count_seats = value; }
23         public DateTime Start_operation { get => start_operation; set => start_operation = value; }
24         public DateTime End_operation { get => end_operation; set => end_operation = value; }
25         public string Class_aircraft { get => class_aircraft; set => class_aircraft = value; }
26
27         public clsInfoAircraft(int id_info, string company_aircraft, string model_aircraft, int count_seats, string class_aircraft)
28         {
29             this.id_info = id_info;
30             this.company_aircraft = company_aircraft;
31             this.model_aircraft = model_aircraft;
32         }
33     }
34 }
```

Рисунок 17 – Поля класса clsInfoAircraft

Конструктор, как и методы доступа необходимы, для работы с объектами по канонам объектно-ориентированного программирования, как это показано на рисунке 18.

```
13     private string name_airport;
14     private string class_airport;
15     private int count_runway;
16
17     public clsAirport(int id_airport, string dislocation, string name_airport, string class_airport, int count_runway)
18     {
19         this.id_airport = id_airport;
20         this.dislocation = dislocation;
21         this.name_airport = name_airport;
22         this.class_airport = class_airport;
23         this.count_runway = count_runway;
24     }
25
26     public int Id_airport { get => id_airport; set => id_airport = value; }
27     public string Dislocation { get => dislocation; set => dislocation = value; }
28     public string Name_airport { get => name_airport; set => name_airport = value; }
29     public string Class_airport { get => class_airport; set => class_airport = value; }
30     public int Count_runway { get => count_runway; set => count_runway = value; }
31
32     public static clsAirport RandomAirport(int id, Random random)
33     {
34         string dislocation = clsData.dislocations[random.Next(0, clsData.dislocations.Length)];
35         string name = clsData.names_airports[random.Next(0, clsData.names_airports.Length)];
36         int count = random.Next(1, 10);
37         string type = clsData.types_airport[random.Next(0, clsData.types_airport.Length)];
38         return new clsAirport(id, dislocation, name, type, count);
39     }
40
41     public String ToString()
42     {
43         return Id_airport.ToString() +
44             " " + Dislocation.ToString() +
45             " " + Name_airport.ToString() +
46             " " + Class_airport.ToString() +
47             " " + Count_runway.ToString();
48     }
49 }
```

Рисунок 18 – Конструктор и методы доступа класса clsAirport

Метод случайной генерации объекта производит генерацию объекта при каждом ее вызове, для того чтобы не производить генерацию объектов в ручном режиме. Данные для генерации случайного объекта берутся из

специально описанного класса, который содержит в себе наборы данных, из которых и производится выборка данных для генерации.

The screenshot shows the Microsoft Visual Studio IDE interface. The code editor displays the `clsAircraft.cs` file with the following code:

```
22     }
23
24     public int Id_activeAircraft { get => id_activeAircraft; set => id_activeAircraft = value; }
25     public bool Active_label { get => active_label; set => active_label = value; }
26     public string Departure_airport { get => departure_airport; set => departure_airport = value; }
27     public string Arrival_airport { get => arrival_airport; set => arrival_airport = value; }
28
29     public static clsAircraft randomAircraft(int id, Random random)
30     {
31         bool label = clsData.labels[random.Next(0, clsData.labels.Length)];
32         if (label == true)
33         {
34             string d_airport = clsData.airports[random.Next(0, clsData.airports.Length)];
35             string a_airport = clsData.airports[random.Next(0, clsData.airports.Length)];
36             while (a_airport.Equals(d_airport))
37             {
38                 a_airport = clsData.airports[random.Next(0, clsData.airports.Length)];
39             }
40             return new clsAircraft(id, label, d_airport, a_airport);
41         }
42         else
43         {
44             string d_airport = "NULL";
45             string a_airport = "NULL";
46             return new clsAircraft(id, label, d_airport, a_airport);
47         }
48     }
49
50     public String ...toString()
51     {
52         return Id_activeAircraft.ToString() + ...
53     }
54 }
```

The solution explorer on the right shows the project structure for "ConsoleApp2".

Рисунок 19 – Метод случайной генерации объекта `clsAircraft`

Метод вывода объекта, согласно заданного формата, производит сохранение объекта в формате строки заданного формата, который необходим для последующей массовой вставки в таблицу базы данных.

The screenshot shows the Microsoft Visual Studio IDE interface. The code editor displays the `clsRunway.cs` file with the following code:

```
37     int length = clsData.lengths[random.Next(0, clsData.lengths.Length)];
38     int width = clsData.widths[random.Next(0, clsData.widths.Length)];
39     bool active = clsData.labels[random.Next(0, clsData.labels.Length)];
40     int classy = clsData.classes_airport[random.Next(0, clsData.classes_airport.Length)];
41     int id = random.Next(1, countStart);
42     return new clsRunway(id_runway, length, width, active, classy, id);
43 }
44
45     public String ...toString()
46     {
47         return Id_run.ToString() +
48             " " + Length_runway.ToString() +
49             " " + Width_runway.ToString() +
50             " " + Active_label.ToString() +
51             " " + Class_runway.ToString() +
52             " " + Id_airport.ToString();
53     }
54 }
```

The solution explorer on the right shows the project structure for "ConsoleApp2".

Рисунок 20 – Метод вывода объекта, согласно заданного формата класса `clsRunway`

6.2 Характеристика класса, содержащего данные для случайной генерации объектов

Данный класс является статическим и содержит в себе массивы различных типов данных, которые используются методом случайной генерации объекта. Часть массивов данного класса показана на рисунке 21.

The screenshot shows the Microsoft Visual Studio IDE interface with the code editor open. The file is named 'clsData.cs'. The code defines a static class 'clsData' with several static arrays containing strings. The arrays include 'labels', 'airports', 'companies', 'models', and 'classes_aircraft'. The 'airports' array contains numerous Russian city names. The 'models' array contains various aircraft model names from different manufacturers. The 'classes_aircraft' array contains three descriptive strings related to aircraft classes.

```
7  namespace AirBase.Classes.Models
8  {
9      public static class clsData
10     {
11         public static bool[] labels = { false, true };
12         public static string[] airports =
13             {
14                 "Абакан", "Анадырь", "Анапа", "Архангельск", "Астрахань", "Барнаул", "Белгород", "Благовещенск", "Братск", "Брянск",
15                 "Варандей", "Владивосток", "Владикавказ", "Волгоград", "Грозный", "Екатеринбург", "Жуковский", "Иркутск",
16                 "Казань", "Калининград", "Калуга", "Кемерово", "Краснодар", "Красноярск", "Курган", "Курск", "Липецк", "Магадан", "Магнитогорск",
17                 "Махачкала", "Минеральные Воды", "Москва", "Мурманск", "Нальчик", "Нижегородск", "Нижнемаксимск", "Нижний Новгород",
18                 "Новокузнецк", "Новосибирск", "Омск", "Оренбург", "Орск", "Осташево", "Пермь", "Петрозаводск", "Петропавловск-Камчатский",
19                 "Псков", "Ростов-на-Дону", "Саветогорск", "Самара", "Санкт-Петербург", "Саранск", "Саратов", "Симферополь", "Сочи", "Ставрополь",
20                 "Сургут", "Сыктывкар", "Томск", "Тимань", "Улан-Удэ", "Ульяновск", "Уфа", "Хабаровск", "Ханты-Мансийск", "Чебоксары", "Челябинск",
21                 "Череповец", "Чита", "Элиста", "Южно-Сахалинск", "Якутск", "Ярославль"
22             };
23         public static string[] companies =
24             {
25                 "Airbus", "ATR", "Saab_AB", "Антонов", "ОАК",
26                 "Сухой", "Иркут", "Туполев", "Ильинин", "Boeing",
27                 "Douglas", "Bombardier", "Embraer"
28             };
29         public static string[] models =
30             {
31                 "A320", "A300", "A310", "A318", "A319", "A320", "A321", "A320", "A340", "A350", "A380", "ATR_42", "ATR_72", "Saab_2000", "Saab_340",
32                 "Ан-2", "Ан-3", "Ан-4", "Ан-6", "Ан-8", "Ан-10", "Ан-12", "Ан-14", "Ан-22", "Ан-24", "Ан-26", "Ан-28", "Ан-30", "Ан-32", "Ан-34", "Ан-38",
33                 "Ан-50", "Ан-70", "Ан-71", "Ан-72", "Ан-74", "Ан-124", "Ан-132", "Ан-140", "Ан-148", "Ан-158", "Ан-168", "Ан-178", "Ан-180", "Ан-188", "Ан-218",
34                 "Ан-225", "Ан-318", "Ан-418", "Ил-96-300", "Ил-62М", "Ил-114", "Ту-160", "Ту-22М3", "Ту-95МС", "Ту-204", "Ту-214", "707", "717", "727",
35                 "737", "747", "757", "767", "777", "787", "Business_Jet", "B-52", "Chinook", "EA-18_Growler", "E-17", "VAL-1", "175", "EMB_110",
36                 "EMB_120", "EMB_121", "ERJ_135", "ERJ_140", "ERJ_145", "E170", "E175", "E190", "E195", "E175-E2", "E190-E2", "E195-E2",
37                 "Legacy_600", "Lineage_1000", "Legacy_450", "Legacy_500", "Legacy_650", "Phenom_100", "Phenom_300", "Praetor_500", "Praetor_600"
38             };
39         public static string[] classes_aircraft = { "Ближнемагистральный", "Среднемагистральный", "Дальнемагистральный" };
40     }
41 }
```

Рисунок 20 – Массивы данных, необходимые для генерации объектов

6.3 Характеристика класса, содержащего точку входа в программу

Данный класс содержит в себе точку входа в программу, а так же метод который и выполняет всю работу приложения, по генерации объектов.

7.1 Генерация тестовых данных.

Генерация тестовых данных производится в результате работы программы. Во время запуска Пользователю предоставляется два варианта работы программы:

- режим малой генерации данных (по 150 элементов каждого класса)
- режим большой генерации данных (по 1 000 000 элементов каждого класса)

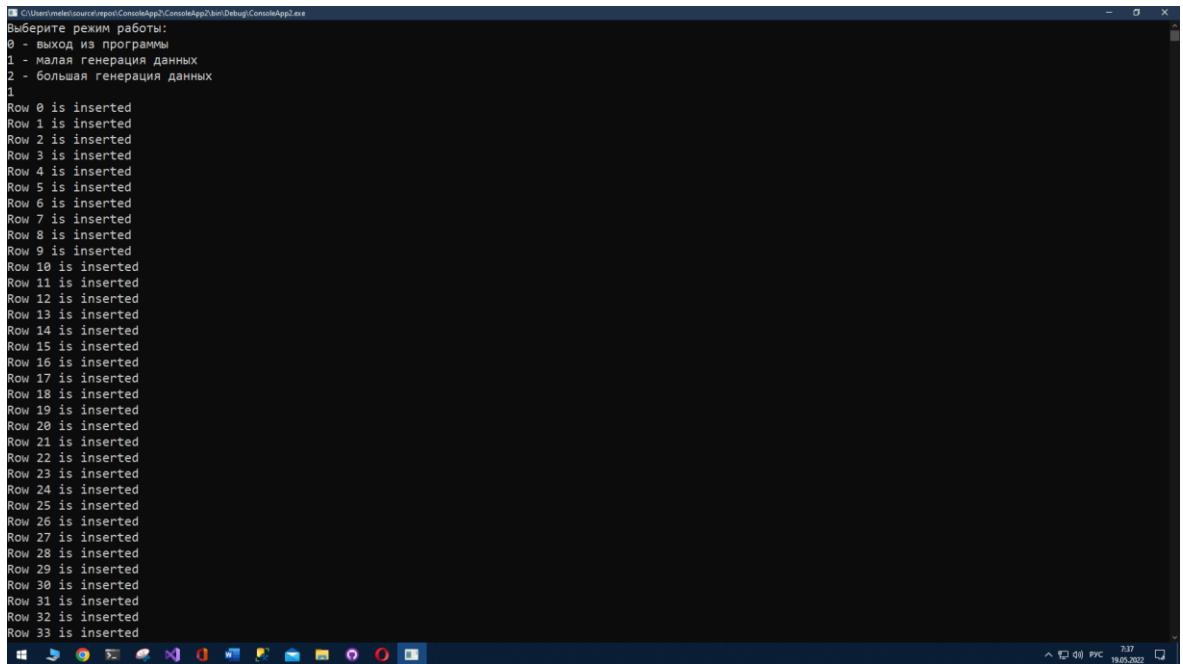
Независимо от режима работы приложения, программа выполняет циклическую последовательность действий:

Генерация объекта;

Сохранение его в строковом формате в список.

По завершении работы цикла, полученный список объектов сохраняется в файл.

Пример работы программы представлен на рисунке 21.



```
C:\Users\metin\source\repos\ConsoleApp2\ConsoleApp2\bin\Debug\ConsoleApp2.exe

Выберите режим работы:
0 - выход из программы
1 - малая генерация данных
2 - большая генерация данных
1
Row 0 is inserted
Row 1 is inserted
Row 2 is inserted
Row 3 is inserted
Row 4 is inserted
Row 5 is inserted
Row 6 is inserted
Row 7 is inserted
Row 8 is inserted
Row 9 is inserted
Row 10 is inserted
Row 11 is inserted
Row 12 is inserted
Row 13 is inserted
Row 14 is inserted
Row 15 is inserted
Row 16 is inserted
Row 17 is inserted
Row 18 is inserted
Row 19 is inserted
Row 20 is inserted
Row 21 is inserted
Row 22 is inserted
Row 23 is inserted
Row 24 is inserted
Row 25 is inserted
Row 26 is inserted
Row 27 is inserted
Row 28 is inserted
Row 29 is inserted
Row 30 is inserted
Row 31 is inserted
Row 32 is inserted
Row 33 is inserted
```

Рисунок 21 – Генерация тестовых данных

8–9. Протестируйте работу запросов на больших объемах данных (Порядка 1 миллиона записей в основных таблицах). Измените конфигурацию сервера PostgreSQL для достижения лучшей производительности на самых медленных запросах. Оптимизируйте схему БД и запросы для достижения лучшей производительности.

Для выполнения данного задания нам потребуется база данных с большим объемом таблиц. При помощи, описанной выше программы произведем генерацию данных, необходимого объема. После чего подготовленные данные перенесем в таблицы баз данных, с использованием команды массовой вставки, поскольку данные сохранены в необходимом формате.

Совершенно очевидным становится тот факт, что наиболее медленными запросами будут запросы выбора. Поскольку именно эти запросы производят наибольшую обработку строк таблиц базы данных, соответственно и времени потребуется больше. Именно с этим типом запросов мы и будем работать.

Для контроля скорости выполнения запросов воспользуемся штатными средствами SQL Server Management Studio (SSMS):

- План выполнения
- Статистика активных запросов
- Статистика клиента

План выполнения запроса, как и статистика активных запросов помогает нам увидеть слабые или уязвимые места в наших запросах, которые отнимают не мало времени, и очень сильно тормозят процесс получения информации из базы данных, поскольку скорость получения информации в современном мире имеет огромное значение для человека. Работа данных средства показана на рисунках 22 – 24.

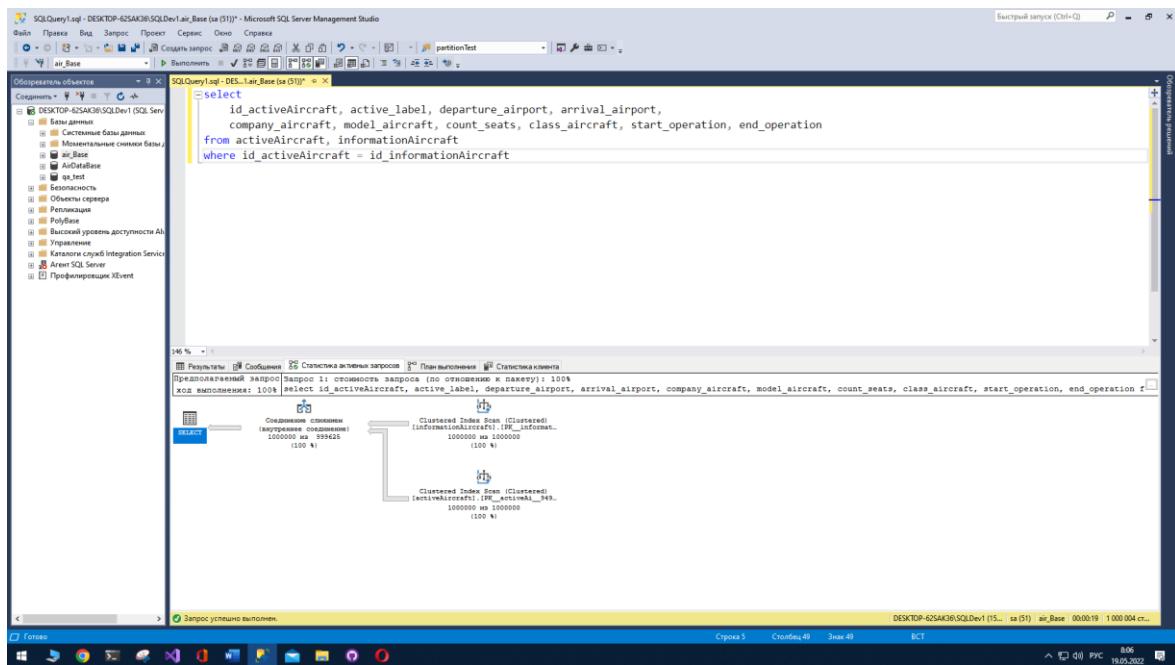


Рисунок 22 – План выполнения запроса

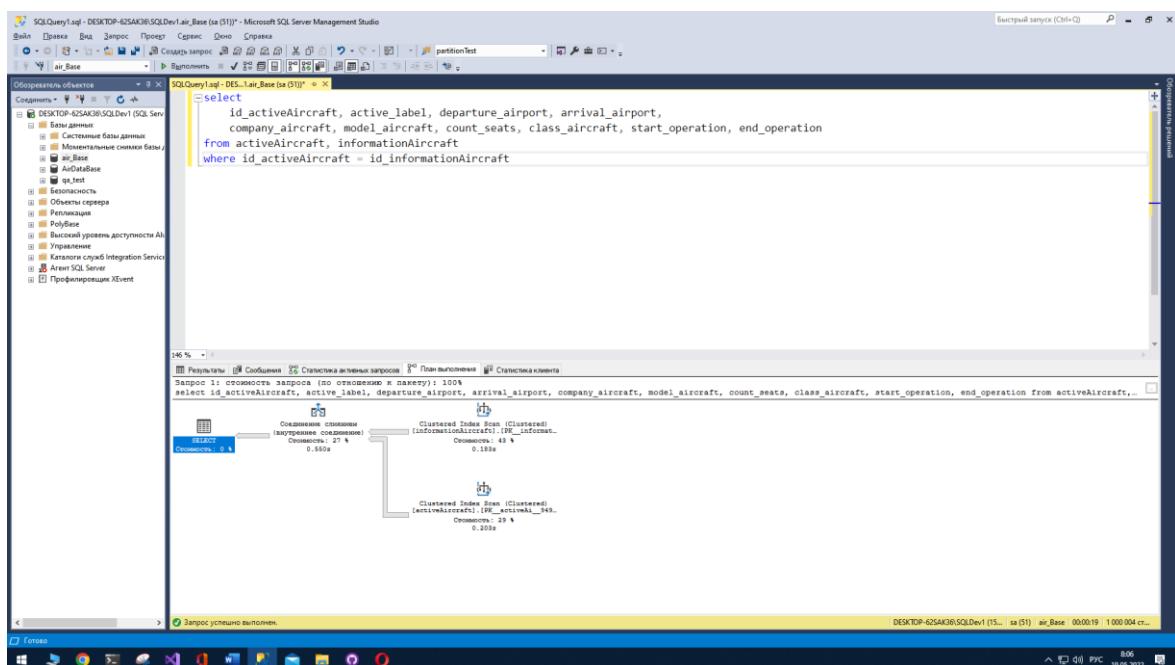


Рисунок 23 – Статистика активных запросов

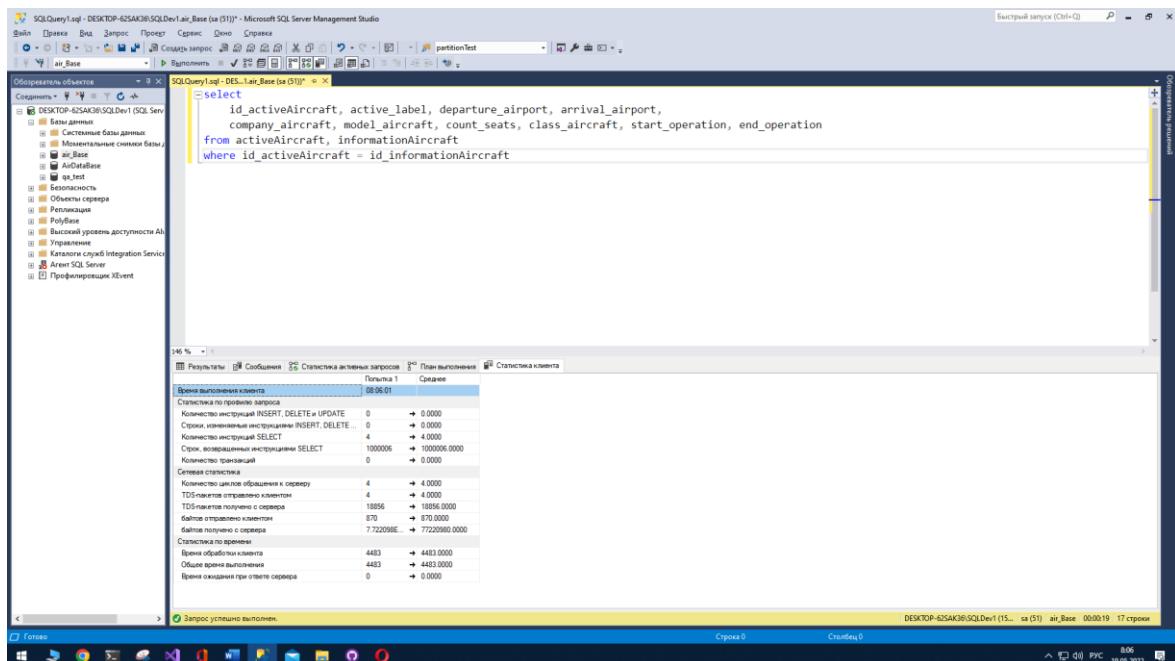


Рисунок 24 – Статистика клиента

Из статистики клиента не трудно заметить, что на выполнение данного запроса требуется около 4,5 секунд. Очевидным становится тот факт, что это долго, однако стоит учитывать объем данных. Как можно сократить время выполнения данного запроса? Это сделать практически не возможно. Однако в выборках такого объема данных нет необходимости, поскольку они становятся не читаемыми и не воспринимаемыми. Отсюда можно сделать вывод, что первой возможностью повысить скорость выполнения запросов – правильно формировать запросы, использовать условия, знать, что конкретно необходимо найти.

Рассмотрим тот же запрос вывода списка данных, однако он будет ограничен по одному условию:

```
select *
from activeAircraft, informationAircraft
where id_activeAircraft = id_informationAircraft and class_aircraft like
'Dальнемагистральный'
```

SQLQueryLog - DESKTOP-625A03\SQLDev1.air_Base (sa (52)) - Microsoft SQL Server Management Studio

```
select *
from activeAircraft, informationAircraft
where id_activeAircraft = id_informationAircraft and class_aircraft = 'дальнемагистральный'
```

	Ползунок 10	Ползунок 9	Ползунок 8	Ползунок 7	Ползунок 6	Ползунок 5	Ползунок 4	Ползунок 3	Ползунок 2	Ползунок 1	Среднее
Время выполнения клиента	13:27:28	13:27:26	13:27:12	13:27:00	13:26:52	13:26:44	13:26:36	13:26:27	13:26:17	13:26:09	
Статистика на профиль запроса											
Количество инструкций INSERT, DELETE и UPDATE	0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0.0000
Строк измененных инструкций INSERT, DELETE	0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0.0000
Количество инструкций SELECT	5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 4	→ 4.9000
Строк возвращенных инструкции SELECT	332867	→ 332867	→ 332867	→ 332867	→ 332867	→ 332867	→ 332867	→ 332867	→ 332867	→ 332866	→ 332866.9000
Количество транзакций	0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0.0000
Статистика											
Количество циклов обращения к серверу	5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 4	→ 4.9000
TDS-пакетов отправлено клиентом	5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 4	→ 4.9000
TDS-пакетов получено с сервера	6610	→ 6610	→ 6610	→ 6610	→ 6610	→ 6610	→ 6610	→ 6610	→ 6610	→ 6609	→ 6609.0000
Байты отправлено клиентом	686	→ 686	→ 686	→ 686	→ 686	→ 686	→ 686	→ 686	→ 686	→ 628	→ 682.2000
Байты получено с сервера	2 7057952	→ 2 7057952	→ 2 7057944	→ 2 7057952	→ 2 7057956	→ 2 7057956	→ 2 7057956	→ 2 7057956	→ 2 7057956	→ 2 7057956	→ 2 7057956.0000
Статистика по времени											
Время обработки клиента	1494	↓ 1723	↓ 6062	↑ 1498	↑ 1341	↓ 1533	↑ 1467	↑ 1464	↓ 1748	↓ 2067	→ 2039.7000
Общее время выполнения	1495	↓ 1760	↓ 6062	↑ 1498	↑ 1341	↓ 1534	↑ 1468	↑ 1464	↓ 1749	↓ 2697	→ 2116.8000
Время ожидания при ответе сервера	1	↓ 137	↑ 0	→ 0	→ 0	↓ 1	↑ 1	↓ 0	↓ 1	↓ 630	→ 77.1000

Рисунок 25 – Временная статистика выполнения запроса

Как можно заметить из статистики клиента, на выполнение запроса требуется в среднем 2 секунды. Попробуем улучшить данный временной показатель. Модифицируем запрос с применением join.

SQLQueryLog - DESKTOP-625A03\SQLDev1.air_Base (sa (52)) - Microsoft SQL Server Management Studio

```
select *
from activeAircraft
join informationAircraft on id_activeAircraft = id_informationAircraft
where class_aircraft = 'дальнемагистральный'
```

	Ползунок 10	Ползунок 9	Ползунок 8	Ползунок 7	Ползунок 6	Ползунок 5	Ползунок 4	Ползунок 3	Ползунок 2	Ползунок 1	Среднее
Время выполнения клиента	13:30:12	13:30:05	13:29:57	13:29:49	13:29:39	13:29:32	13:29:24	13:29:11	13:29:04	13:28:55	
Статистика на профиль запроса											
Количество инструкций INSERT, DELETE и UPDATE	0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0.0000
Строк измененных инструкций INSERT, DELETE	0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0.0000
Количество инструкций SELECT	5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 4	→ 4.9000
Строк возвращенных инструкции SELECT	332867	→ 332867	→ 332867	→ 332867	→ 332867	→ 332867	→ 332867	→ 332867	→ 332867	→ 332866	→ 332866.9000
Количество транзакций	0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0.0000
Статистика											
Количество циклов обращения к серверу	5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 4	→ 4.9000
TDS-пакетов отправлено клиентом	5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 5	→ 4	→ 4.9000
TDS-пакетов получено с сервера	6610	→ 6610	→ 6610	→ 6610	→ 6610	→ 6610	→ 6610	→ 6610	→ 6610	→ 6609	→ 6609.0000
Байты отправлено клиентом	694	→ 694	→ 694	→ 694	→ 694	→ 694	→ 694	→ 694	→ 694	→ 636	→ 682.2000
Байты получено с сервера	2 7057962	→ 2 7057962	→ 2 7057968	→ 2 7057968	→ 2 7057968	→ 2 7057968	→ 2 7057968	→ 2 7057968	→ 2 7057968	→ 2 7057968	→ 2 7057968.0000
Статистика по времени											
Время обработки клиента	1608	↑ 1519	↓ 1806	↑ 1424	↓ 1607	↓ 1604	↑ 1603	↑ 1599	↑ 1477	↓ 2021	→ 2048.0000
Общее время выполнения	1610	↑ 1519	↓ 1806	↑ 1425	↓ 1425	↓ 1723	↑ 1608	↑ 1561	↑ 1478	↓ 2669	→ 2148.3000
Время ожидания при ответе сервера	2	↑ 0	→ 0	↓ 2	↑ 1	↓ 39	↑ 0	↓ 2	↑ 1	↓ 588	→ 63.9000

Рисунок 26 – Временная статистика выполнения запроса

Нетрудно увидеть, что модификация запроса не принесла особого успеха и на выполнение запроса все так же требуется в среднем 2 секунды. Попробуем улучшить данный временной показатель, посредством изменения структуры таблицы и данных в таблице: произведем замену строковых

значений в столбце class_aircraft таблицы informationAircraft на числовые, например, классу ближнемагистральных самолётов, будет соответствовать класс самолетов – 1, для классу среднемагистральных соответствует класс 2, и т.д. С учетом данной корректировки попробуем выполнить вышеуказанный запрос.

The screenshot shows the Microsoft SQL Server Management Studio interface. In the center, there is a query window with the following SQL code:

```
select *
from activeAircraft, informationAircraft
where id_activeAircraft = id_informationAircraft and class_aircraft = 3
```

Below the query window, the results pane displays temporary statistics for the execution plan. The table is titled "Статистика клиента" (Client Statistics) and includes columns for various metrics across 10 partitions. The last row shows the average values.

	Политка 10	Политка 9	Политка 8	Политка 7	Политка 6	Политка 5	Политка 4	Политка 3	Политка 2	Политка 1	Среднее
Время выполнения запроса	13:59:43	13:59:37	13:59:36	13:59:24	13:59:16	13:59:09	13:59:02	13:58:58	13:58:42	13:58:35	
Статистика по профилью запроса											
Количество инструкций INSERT, DELETE и UPDATE	0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0.0000
Строки, измененные инструкциями INSERT, DELETE	0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0.0000
Количество инструкций SELECT	2	→ 2	→ 2	→ 2	→ 2	→ 2	→ 2	→ 2	→ 2	→ 1	→ 1.9000
Количество строк, возвращенных инструкциями SELECT	332861	→ 332861	→ 332861	→ 332861	→ 332861	→ 332861	→ 332861	→ 332861	→ 332861	→ 332860	→ 332860.9000
Количество транзакций	0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0.0000
Сетевая статистика											
Количество циклов обращения к серверу	2	→ 2	→ 2	→ 2	→ 2	→ 2	→ 2	→ 2	→ 2	→ 1	→ 1.9000
TDS-пакетов отправлено клиентом	2	→ 2	→ 2	→ 2	→ 2	→ 2	→ 2	→ 2	→ 2	→ 1	→ 1.9000
TDS-пакетов получено сервером	5215	→ 5215	→ 5215	→ 5215	→ 5215	→ 5215	→ 5215	→ 5215	→ 5215	→ 5215	→ 5215.9000
Быстро отправлены клиентом	338	→ 338	→ 338	→ 338	→ 338	→ 338	→ 338	→ 338	→ 338	→ 338	→ 332.2000
Быстро получено с сервера	2 135702...	→ 2 135702...	→ 2 135702...	→ 2 135702...	→ 2 135702...	→ 2 135702...	→ 2 135702...	→ 2 135702...	→ 2 135702...	→ 2 135699...	→ 2 1357020.0000
Статистика по времени											
Время обработки клиента	1393	↓ 1417	↑ 1392	↓ 1626	↑ 1607	↑ 1510	↓ 6456	↑ 1396	↓ 1646	↑ 1526	→ 1996.9000
Общее время выполнения	1394	↓ 1417	↑ 1392	↓ 1628	↑ 1609	↑ 1527	↓ 6456	↑ 1398	↓ 1648	↑ 1526	→ 1999.5000
Время ожидания при ответе сервера	1	↑ 0	→ 0	↓ 2	→ 2	↓ 17	↑ 0	↓ 2	→ 2	↑ 0	→ 2.6000

At the bottom of the results pane, it says "Запрос успешно выполнен." (Query executed successfully).

Рисунок 27 – Временная статистика выполнения запроса

После осуществления реорганизации таблицы базы данных, выполнение запроса заняло чуть меньшее время. Однако его уменьшение не столь велико чтобы можно было говорить о каким-либо существенном ускорении. Попробуем в добавок к реорганизации базы данных изменить запрос, и повторим его выполнение.

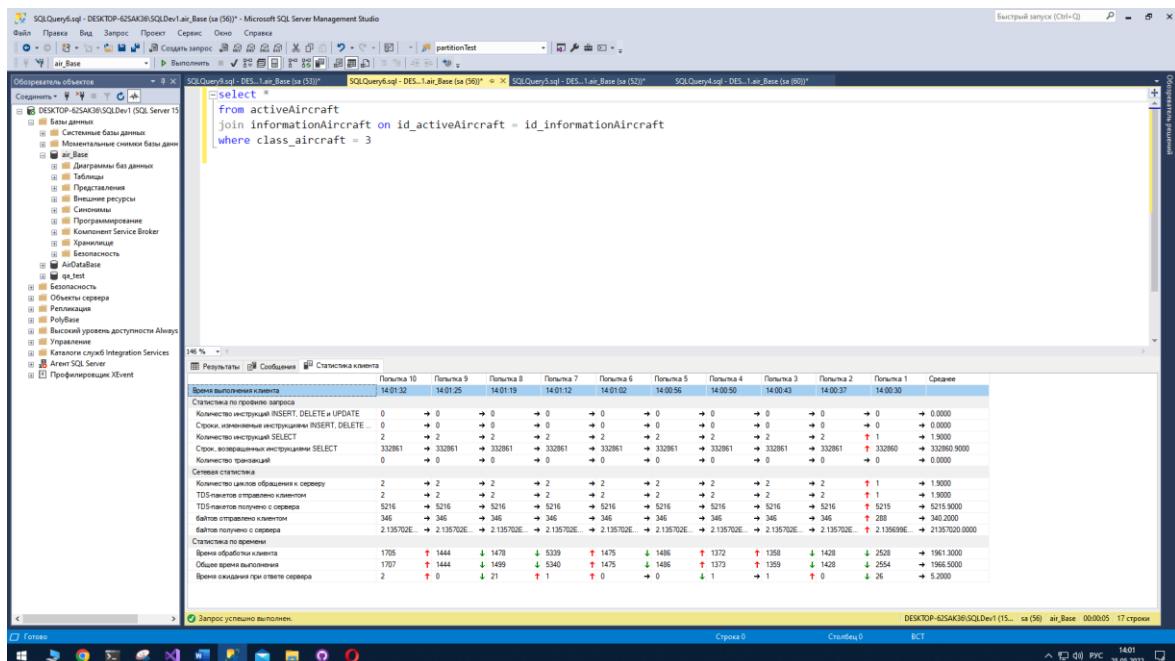


Рисунок 28 – Временная статистика выполнения запроса

После проведения описанных манипуляций можно заметить незначительное сокращение времени, относительно предыдущей модификации. Однако, теперь сравним время выполнения данного запроса, до начала всех модификаций (2116), со временем выполнения запроса со всеми модификациями(1966). Не трудно заметить, что произошло уменьшение времени на 150мс, что составляет порядка 7-8%. Все таки данное значение, не является столь существенным, чтобы можно было говорить, о каком либо значительном ускорении.

В результате оптимизации, получить значительного ускорения работы запросов не удалось, в виду определенных причин:

- достаточно большие размеры таблицы базы данных (порядка 1 миллиона строк);
- отсутствие возможности изменять конфигурацию СУБД, поскольку MSSQL – проприетарный продукт, в котором такие возможности не предусмотрены;
- относительно малая вычислительная мощность CPU на используемом ПК (в качестве CPU используется Intel Xeon E5-2620v3 с тактовой частотой 2.8 ГГц).

ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы были получены и отточены навыки по работе с СУБД, рассмотрены особенности корпоративных систем, а так особенности работы с базами данных больших объемов, порядка 1 миллиона строк.

Все скрипты, для работы с СУБД MS SQL 2019 Developer, а так же код программы для генерации данных расположены на сайте GitHub по ссылке:
<https://github.com/Black-Viking-63/EnterpriseDataBase>