

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

**«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО»**

институт компьютерных наук и кибербезопасности

высшая школа технологий искусственного интеллекта

направление 02.03.01 математика и компьютерные науки

Отчет по курсовой работе

по дисциплине:

«Теоретические основы баз данных»

Тема работы:

«Разработка и программирование базы данных предметной области
«Пассажирские авиаперевозки»»

Обучающийся: _____ Черепанов Михаил Дмитриевич

Руководитель: _____ Попов Сергей Геннадиевич

«_____» _____ 20__ г.

Санкт-Петербург 2024

Содержание

1	Аналитика	3
1.1	Описание предметной области	3
1.2	Формулировка целей создания базы данных	6
1.3	Основные сущности	6
1.4	ER Диаграмма	7
1.5	Чтение ER диаграммы	9
1.6	Диаграмма объектов	9
2	Проектирование базы данных	11
2.1	Таблицы базы данных	11
2.2	Схема базы данных	15
2.3	Создание базы данных	18
2.4	Заполнение базы данных	18
3	Запросы к базе данных	19
3.1	Запрос№1	20
3.1.1	Запрос№1.a	20
3.1.2	Запрос№1.b	22
3.2	Запрос№2	24
3.3	Запрос№3	26
3.3.1	Запрос№3.a	26
3.3.2	Запрос№3.b	28
3.4	Запрос№4	30
3.4.1	Запрос№4.a	30
3.4.2	Запрос№4.b	32
3.5	Запрос№5	34
3.6	Запрос№6	36
3.7	Запрос№7	37
3.8	Запрос№8	40
4	Заключение	45
5	Приложение №2. Программа заполнения базы данных	52

1 Аналитика

1.1 Описание предметной области

Пассажирские авиаперевозки – транспортная отрасль, занимающаяся перевозками людей (пассажиров) и их багажа между начальным и конечным пунктами (аэропортами).

Аэропорт — это комплекс сооружений, предназначенный для приёма, отправки, базирования воздушных судов.

Авиакомпания - организация, производящая пассажирские перевозки. Авиакомпания заключает устанавливает расписание рейсов и заключает договоры с клиентами (пассажирами) о перелетах.

Пассажирский транспортный самолет - средство для перевозки пассажиров.

Пассажирский транспортный самолет – это воздушное судно, способное перевозить определенное кол-во пассажиров и их багажа из одной точки, в другую.

Каждый экземпляр самолета характеризуется набором параметров:

- Модель;
- Регистрационный номер;
- Наличие классов обслуживания(эконом, премиум-эконом, бизнес, первый класс).
Мест каждого из классов обслуживания в самолете - определенное количество.
- Общий вес багажа
- Экипаж, необходимый для обслуживания данного самолета и пассажиров.

Экипаж самолета.

Экипаж самолета состоит из нескольких пилотов и нескольких бортпроводников.

Пилот – квалифицированный специалист, управляющий воздушным судном. Среди пилотов есть один – главный пилот/ капитан воздушного судна.

Бортпроводник – квалифицированный специалист, выполняющий работы по обслуживанию пассажиров во время полета. Среди бортпроводников есть старший бортпроводник, руководящий командой и принимающий решения в исключительных ситуациях.

Кол-во бортпроводников и пилотов зависит от конкретного экземпляра самолета.

Требуется специальное образование, чтобы стать бортпроводником или пилотом.

Рейс.

Рейс - это коммерческий перелет самолета из точки отправления в точку прибытия.

Возможны многосегментные(с промежуточными остановками для дозаправки или тех. обслуживания самолета) и односегментные рейсы(без промежуточных остановок). Во время промежуточных остановок посадка и высадка пассажиров невозможна.

Сегмент - перелет самолета между двумя аэропортами без промежуточных остановок.

Каждый рейс определяется набором параметров:

- Аэропорт отправления;
- Аэропорт назначения;
- Промежуточные остановки;
- Дата и время вылета;
- Время в пути;

Рейсы выполняются самолетами в соответствии с установленным авиакомпанией расписанием.

Пассажир.

Пассажир — человек, заключивший договор о перевозке с авиакомпанией(купивший билет).

Пассажиром самолета может стать любой человек, не занесенный в черный список(список лиц, которым по какой-либо причине запрещено летать рейсами данной авиакомпании). То есть у каждого человека, хоть раз обращавшегося в авиакомпанию есть статус, который обозначает, разрешено ли ему летать рейсами данной авиакомпании.

Билет на самолет(Договор).

Билет на самолет - договор между пассажиром и авиакомпанией, разрешающий пассажиру перелет по определенному маршруту рейсами данной авиакомпании.

Договор определяет маршрут пассажира.

Маршрут - это непустое множество пар: { Пункт отправления, Пункт назначения }.

Описание процесса перелета пассажирского самолета

Пассажир заключает договор на перелет с авиакомпанией.

При заключении договора пассажир выбирает подходящий ему маршрут.

Авиакомпания предлагает пассажиру возможные варианты построения этого маршрута(комбинацию рейсов) в соответствии с расписанием или сообщает пассажиру, что перелет по данному маршруту невозможен. Маршрут между начальной и конечной точкой пары, указанной в маршруте может быть составлен из нескольких рейсов.

У пассажира есть возможность купить билет с открытой или установленной датой.

В случае билета с установленной датой, пассажир сразу выбирает удобный ему вариант маршрута.

В случае билета с открытой датой пассажир выбирает граничные даты. Авиакомпания обязуется по его требованию предоставить возможность перелета любым из предложенных вариантов в промежутке между этими датами.

После выбора варианта маршрута, авиакомпания соотносит множество пар { Пункт отправления, Пункт назначения} с конкретными рейсами.

Затем пассажир выбирает подходящий ему класс обслуживания из тех, что присутствует в назначенном самолете и количество необходимого ему багажа. Номер места пассажир выбирает сам или предоставляет это права авиакомпании. Информация о классе обслуживания, номере места и количестве багажа так же вносится в билет.

После выбора даты пассажир в назначенный день прибывает в указанный в договоре аэропорт, сдает багаж, занимает указанное в договоре место.

В назначенное время вылета самолет с обслуживающим его экипажем и пассажирами, вылетает назначенным ему рейсом.

Если рейс является многосегментным, самолет выполняет остановки в соответствии с расписанием рейса.

По прилете в конечную точку рейса, пассажиры покидают самолет, забирают багаж.

Если маршрут состоит из нескольких рейсов, то по окончании каждого из них пассажир прибывает в указанный аэропорт отправления следующего рейса в соответствии с датой, указанной в билете(или сам выбирает дату полета из предложенных, если билет с открытой датой).

Затем повторяет вышеописанную процедуру.

1.2 Формулировка целей создания базы данных

1. Учёт количества перелетов и суммарного времени в полете каждого самолета.
2. Учёт перелётов каждого пассажира.
3. Предоставление сведений о доступных билетах на конкретные даты по определенным маршрутам.

1.3 Основные сущности

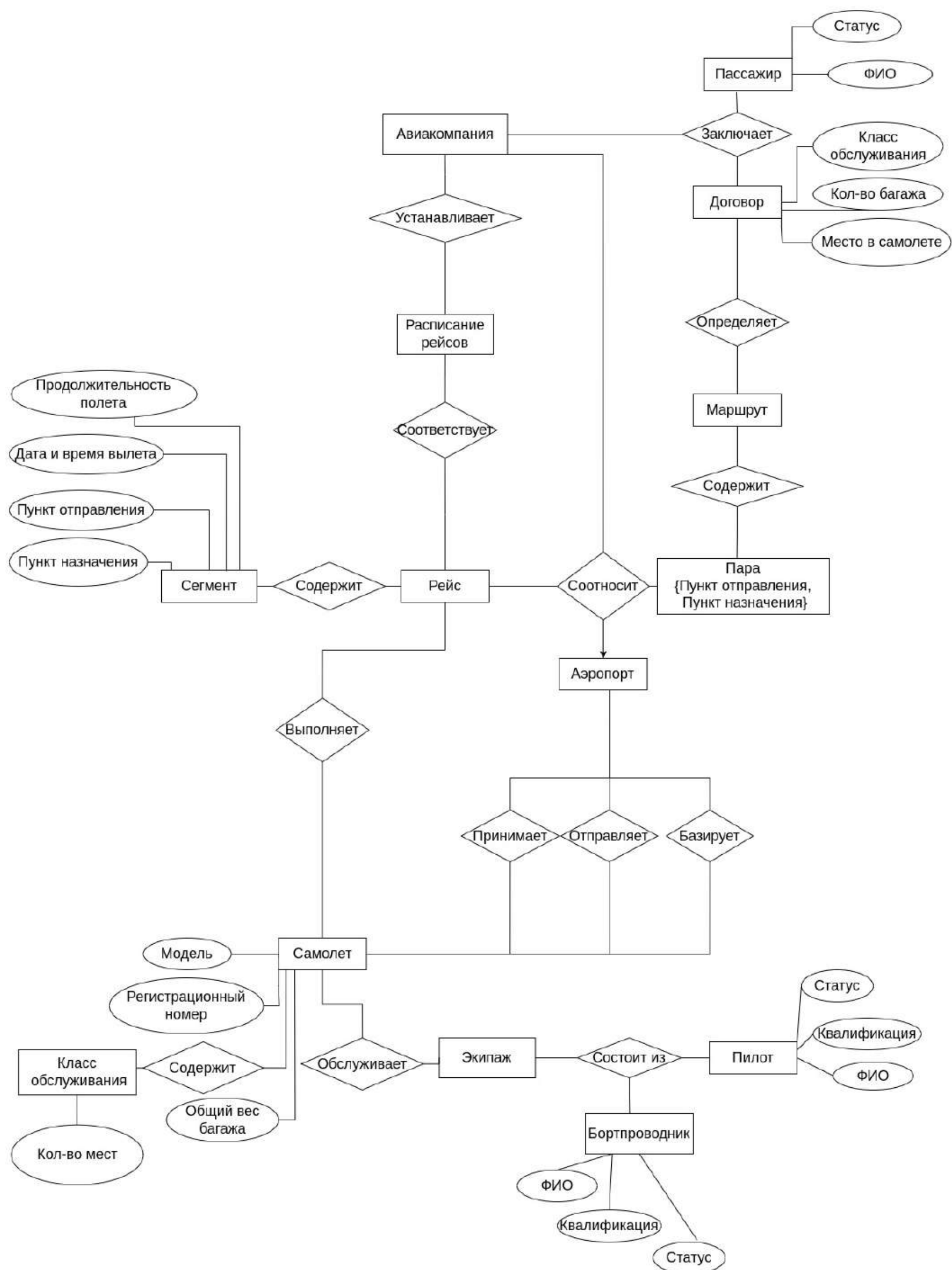
В рамках данной курсовой работы введены следующие сущности:

- Авиакомпания - организация, производящая пассажирские перевозки.
- Аэропорт - комплекс сооружений, предназначенный для приёма, отправки, базирования воздушных судов.
- Самолет - воздушное транспортное судно.
 - Модель
 - Регистрационный номер
 - Общий вес багажа
- Класс обслуживания - соглашения, определяющие уровень оказываемых услуг.
 - Кол-во мест
- Экипаж - команда людей, управляющих и обслуживающих самолет и пассажиров.
- Бортпроводник - член экипажа, обслуживающий пассажиров.
 - ФИО
 - Квалификация
 - Статус
- Пилот - член экипажа, управляющий самолетом.
 - ФИО
 - Квалификация
 - Статус
- Расписание рейсов - упорядоченные по времени рейсы.
- Рейс - это коммерческий перелет самолета из точки отправления в точку прибытия.

- Сегмент - ерелет самолета между двумя аэропортами без промежуточных остановок.
 - Пункт отправления
 - Пункт назначения
 - Дата и время вылета
 - Время в пути
- Пассажир - человек, заключивший договор о перевозке с авикомпанией.
 - ФИО
 - Статус
- Договор - документ, разрешающий пассажиру перелет по определенному маршруту
 - Место в самолете.
 - Кол-во багажа.
 - Класс обслуживания.
- Маршрут то непустое множество пар: { Пункт отправления, Пункт назначения }.
- Пара(Пункт отправления, пункт назначения).

1.4 ER Диаграмма

ER диаграмма представлена на рис. 1.



1.5 Чтение ER диаграммы

ER диаграмму следует читать следующим образом:

1. Авиакомпания заключает договор с пассажиром;
2. Договор определяет маршрут пассажира;
3. Маршрут содержит множество пар { Пункт отправления - пункт назначения};
4. Авиакомпания устанавливает расписание рейсов;
5. Рейсы соответствуют расписанию рейсов;
6. Авиакомпания соотносит рейс и пару { Пункт отправления - пункт назначения};
7. Авиакомпания соотносит пару { Пункт отправления - пункт назначения} и аэропорты отправления и прибытия.
8. Аэропорт принимает, отправляет и базирует самолеты;
9. Рейс содержит сегменты рейса;
10. Самолет выполняет рейс;
11. Самолет содержит классы обслуживания;
12. Экипаж обслуживает самолет;
13. Экипаж состоит из пилотов и бортпроводников;

1.6 Диаграмма объектов

Диаграмма объектов представлена на рис.№2.

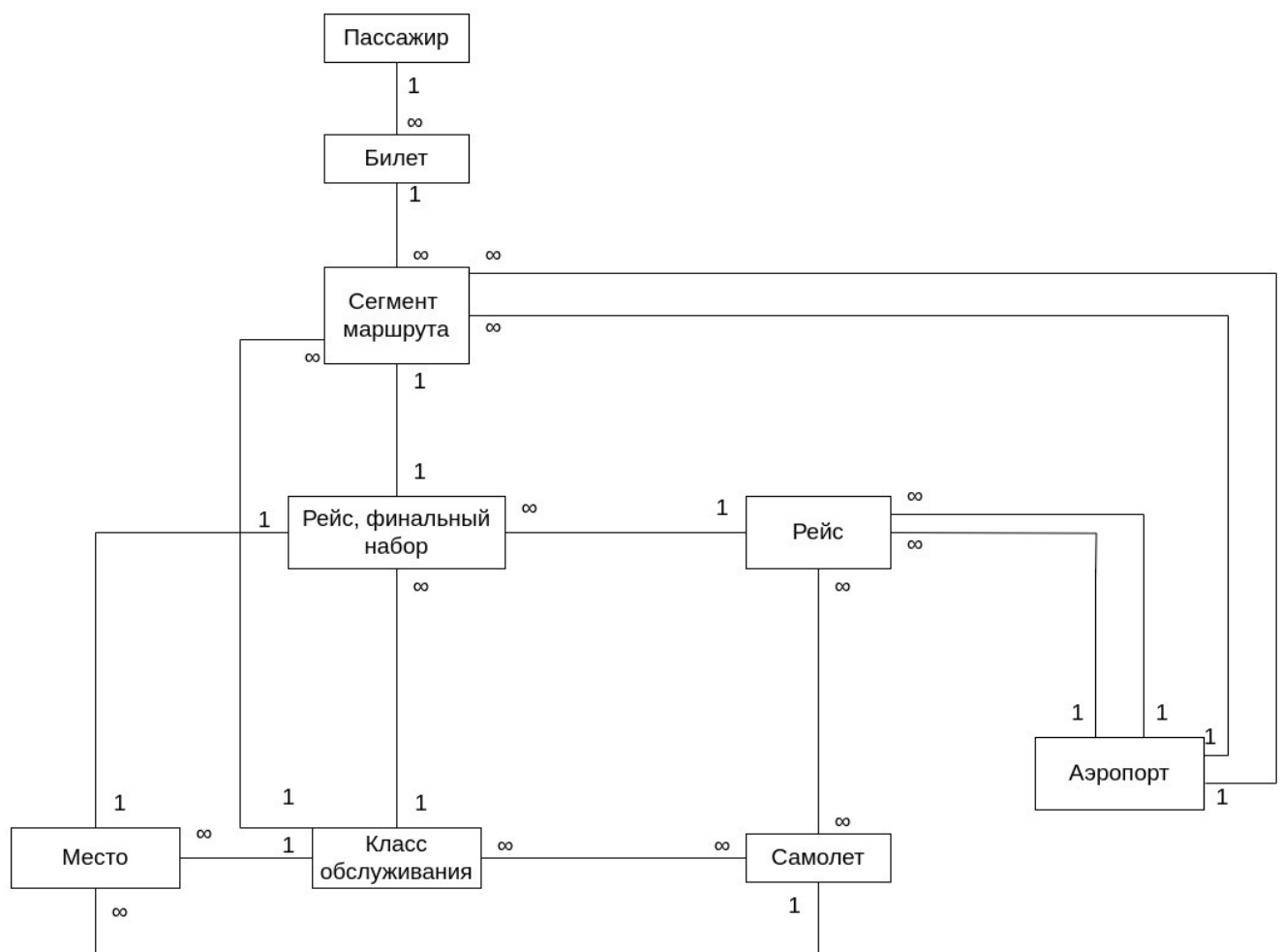


Рис.№2. Диаграмма объектов.

2 Проектирование базы данных

Для реализации базы данных были созданы 11 таблиц, приведенные на рис. 1 - рис.11.

2.1 Таблицы базы данных

Пассажир

Название	Название на английском	Тип	Тип ключа	Ссылка
Пассажир_id	Passenger_id	INTEGER	PK	
Фамилия	Last name	VARCHAR(50)		
Имя	First name	VARCHAR(30)		
Отчество	Middle name	VARCHAR(30)		
Серия паспорта	Passport series	VARCHAR(4)		
Номер паспорта	Passport number	VARCHAR(7)		
Дата рождения	Date of birth	DATE (01.01.1900 - CURR_DATE)		

Рис. 1: Пассажир (Passenger)

Аргументация выбранных ограничений:

Самая длинная фамилия в России: Колобкова-Крупчатникова – 23 символа, возьмем число, в два раза большее этого, округлим до десятков.

Самое длинное имя в России: Абдурахмангаджи – 15 символов, возьмем число, в два раза большее этого, округлим до десятков. Самое длинное отчество, очевидно, образовано от самого длинного имени → подставим такое же ограничение как и на имя.

Билет

Название	Название на английском	Тип	Тип ключа	Ссылка
Билет_id	Ticket_id	INTEGER	PK	
Пассажир_id	Passenger_id	INTEGER	FK	Passenger.passenger_id

Рис. 2: Билет (Ticket)

Сегмент маршрута

Название	Название на английском	Тип	Тип ключа	Ссылка
Сегмент маршрута_id	Route_segment_id	INTEGER	PK	
Билет_id	Ticket_id	INTEGER	FK	Билет(билет_id)
Аэропорт_вылета_id	Airport_flight_id	INTEGER	FK	Airport.airport_id
Аэропорт_прилета_id	Airport_of_landing_id	INTEGER	FK	Airport.airport_id
Класс_обслуживания_id	Class_of_service_id	INTEGER	FK	Class_of_service. Class_of_service_id
Рейс_финальный набор_id	Flight_final_set_id	INTEGER	FK	Flight_final_set. Flight_final_set_id
Количество багажа	Number of baggage	INTEGER(0 - 100)		

Рис. 3: Сегмент маршрута (Route segment)

Аргументация выбранных ограничений:

В компании “Аэрофлот” – самой известной на Российском рынке – максимальный тариф провоза багажа – 32 кг – для бизнес класса. Однако, перевес возможен и за него нужно доплачивать. В открытом доступе не удалось найти явную границу ограничения на количество багажа сверху, поэтому ограничимся соображениями здравого смысла и поставим границу – 100 кг.

Рейс финальный набор

Название	Название на английском	Тип	Тип ключа	Ссылка
Сегмент маршрута_id	Route_segment_id	INTEGER	PK	
Билет_id	Ticket_id	INTEGER	FK	Билет(билет_id)
Аэропорт_вылета_id	Airport_flight_id	INTEGER	FK	Airport.airport_id
Аэропорт_прилета_id	Airport_of_landing_id	INTEGER	FK	Airport.airport_id
Класс_обслуживания_id	Class_of_service_id	INTEGER	FK	Class_of_service. Class_of_service_id
Рейс_финальный набор_id	Flight_final_set_id	INTEGER	FK	Flight_final_set. Flight_final_set_id
Количество багажа	Number of baggage	INTEGER(0 - 100)		

Рис. 4: Рейс финальный набор(Flight final set)

Класс обслуживания

Название	Название на английском	Тип	Тип ключа	Ссылка
Класс_обслуживания_id	Class_of_service_id	INTEGER	PK	
Название	Name	VARCHAR(50)		

Рис. 5: Класс обслуживания(Class of service)

Аргументация выбранных ограничений:

Самое длинное название класса обслуживания в самолете – премиум эконом – 14 символов. Умножим это число на два и округлим до десятков.

Место

Название	Название на английском	Тип	Тип ключа	Ссылка
Место_id	Seat_id	INTEGER	PK	
Класс_обслуживания_id	Class_of_service_id	INTEGER	FK	Class_of_service.Class_of_service_id
Самолет_id	Airplane_id	INTEGER	FK	Airplane.Airplane_id
Номер места	Number of seat	VARCHAR(5)		

Рис. 6: Место (Seat)

Аргументация выбранных ограничений:

Места в самолете обычно обозначаются числом(до трехзначного) и одной буквой – получим 4 символа, округлим до 5.

Класс обслуживания - самолет

Название	Название на английском	Тип	Тип ключа	Ссылка
Самолет_id	Airplane_id	INTEGER	PK,FK	Самолет(Самолет_id)
Класс_обслуживания_id	Class_of_service_id	INTEGER	PK,FK	Класс_обслуживания(Класс_обслуживания_id)

Рис. 7: Класс обслуживания - самолет (Class of service - airplane)

Аргументация выбранных ограничений:

Самое длинное слово в русском языке имеет длину 35 символов. Названия моделей самолетов, как правило, состоят из 1-2 слов и, возможно, набора цифр. Ограничим длину циферной цепочки 10 символами – получим итоговое ограничение: 80 символов.

Самолет

Название	Название на английском	Тип	Тип ключа	Ссылка
Самолет_id	Airplane_id	INTEGER	PK	
Модель	Model	VARCHAR(80)		
Общий вес багажа	number_of_baggage	INTEGER (0-250 000)		

Рис. 8: Самолет (Airplane)

Аргументация выбранных ограничений:

Самый большой грузовой самолет Ан-255 вмещает 250 тонн груза, данная база данных разрабатывается для пассажирских самолетов, но в качестве исключения поставим ограничение 250 тонн.

Самолет - рейс

Название	Название на английском	Тип	Тип ключа	Ссылка
Самолет_id	Airplane_id	INTEGER	PK,FK	Самолет(Самолет_id)
Рейс_id	Flight_id	INTEGER	PK,FK	Рейс(Рейс_id)

Рис. 9: Самолет - рейс (Airplane - Flight)

Рейс

Название	Название на английском	Тип	Тип ключа	Ссылка
Рейс_id	Flight_id	INTEGER	PK	
Аэропорт_вылета_id	Airport_flight_id	INTEGER	FK	Airport.airport_id
Аэропорт_прилета_id	Airport_of_landing_id	INTEGER	FK	Airport.airport_id
Дата и время вылета	Date and time flight	DATESTAMP (1903-2124)		
Дата и время посадки	Date and time landing	DATESTAMP (1903-2124)		

Рис. 10: Рейс (Flight)

Аргументация выбранных ограничений:

В качестве нижней границы даты установим дату первого в мире полета на самолете. Предположим, что через 100 лет, наша база данных уже устареет, поэтому 2124г – верхняя граница.

Аэропорт

Название	Название на английском	Тип	Тип ключа	Ссылка
Аэропорт_id	Airport_id	INTEGER	PK	
Название	Name_airport	VARCHAR(70)		
Страна	Country	VARCHAR(60)		

Рис. 11: Аэропорт (Airport)

Аргументация выбранных ограничений:

Самое длинное официальное название страны - United Kingdom of Great Britain and Northern Ireland, но оно редко используется в официальных документах. Его длина – 52 символа. Округлим это значение вверх до десятков

Все выбранные ограничения взяты на основе данных, предоставленных Yandex browser, одного из самых востребованных поисковиков в России, исходя из того, что база данных предназначена для Российского использования.

2.2 Схема базы данных

Схема базы данных приведена на рис. 3(На русском языке) и на рис. 4 (На английском языке).

Около каждой таблицы отмечен уровень ее заполнения.

Рис. №3: Схема базы данных(на русском)

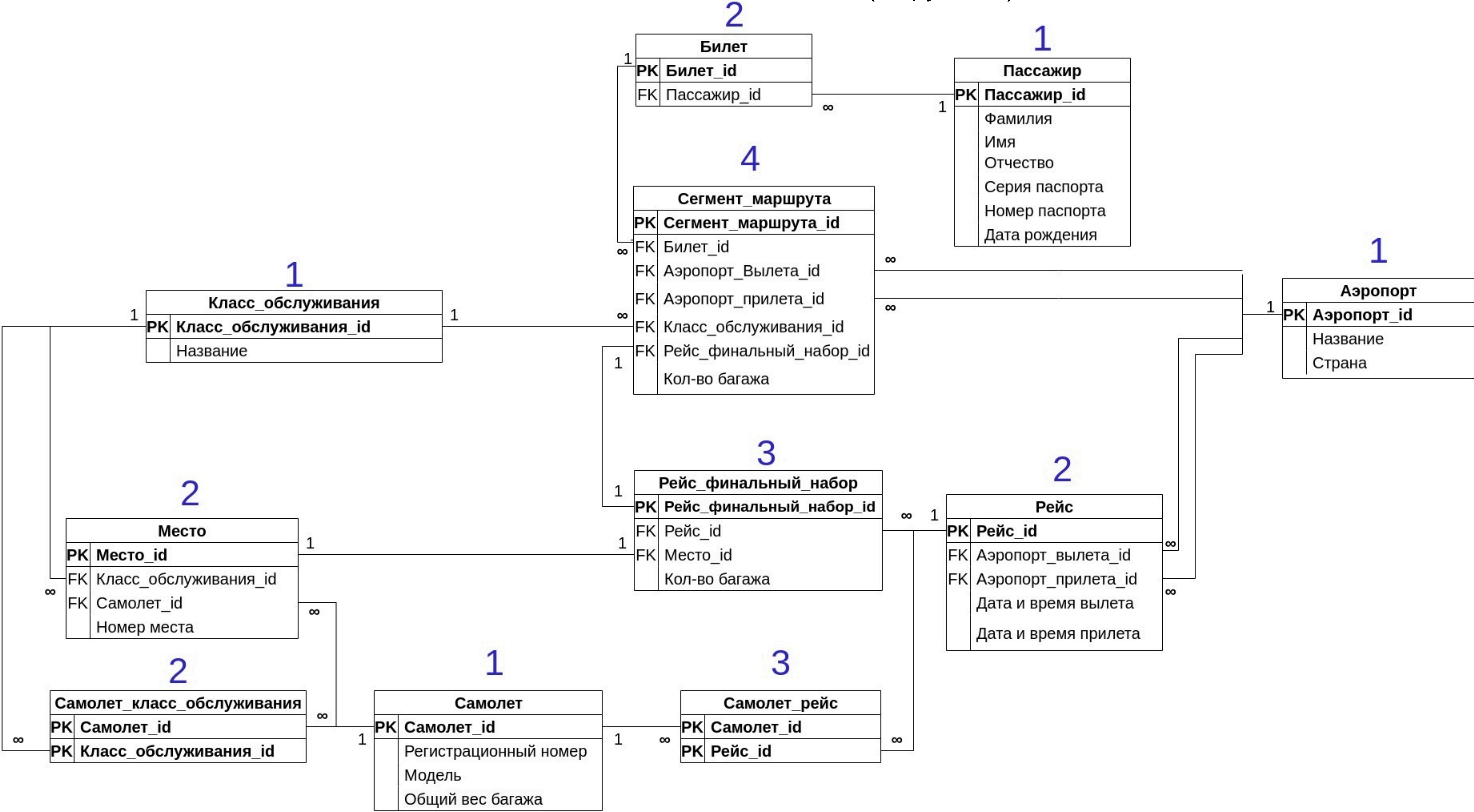
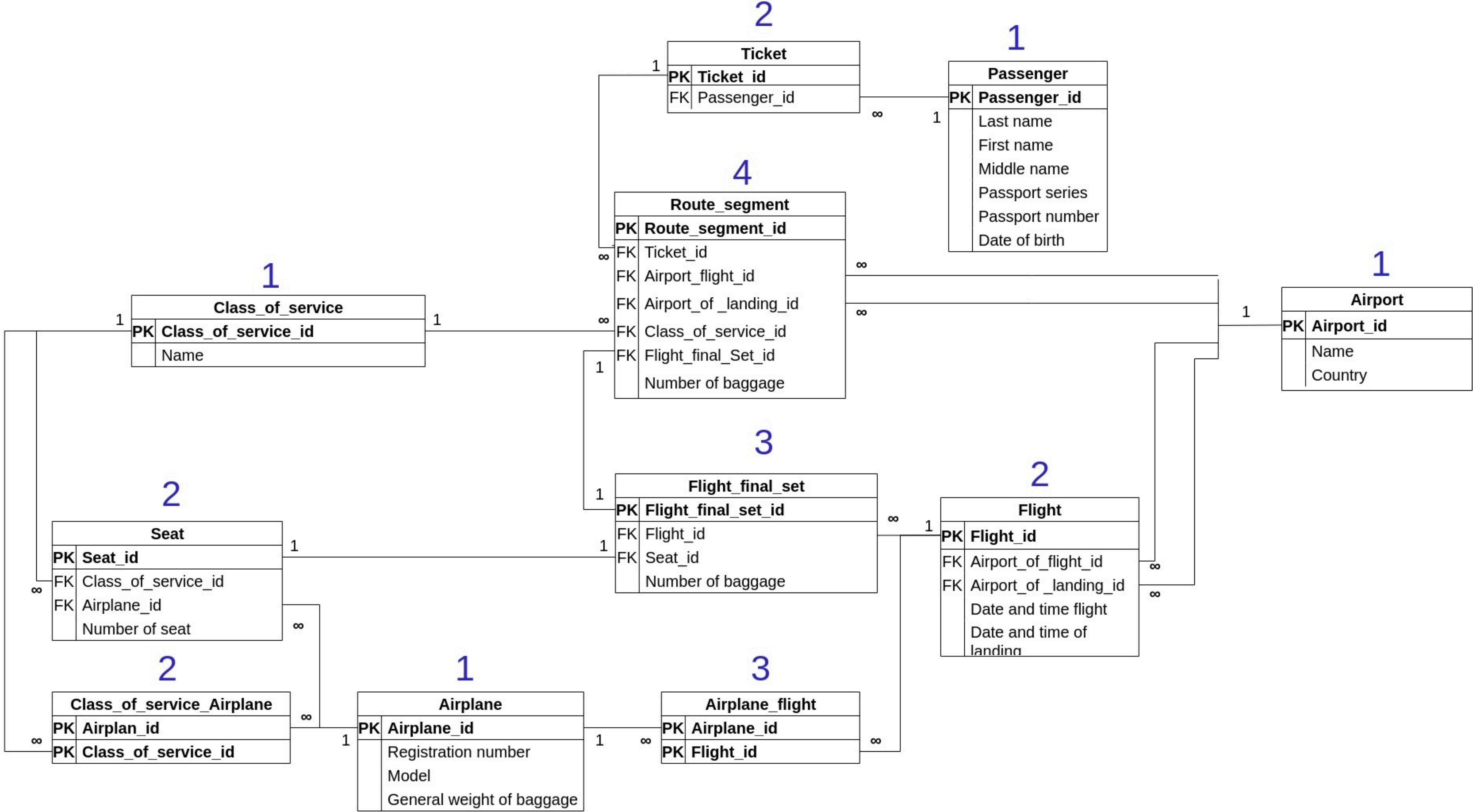


Рис. №4: Схема базы данных(на английском)



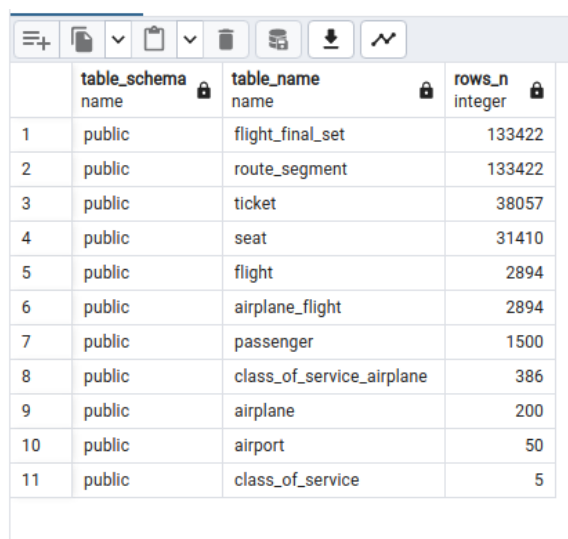
2.3 Создание базы данных

Создание базы данных было реализовано на языке SQL в субд Posgresql. Исходный код создания базы данных приведен в Приложении №1.

2.4 Заполнение базы данных

Заполнение базы данных было реализовано на языке Python,используя api Posgresql, исходный код программы приведен в Приложении №2.

На рис. 12 показана информация о количестве записей в каждой из таблиц.



	table_schema name	table_name name	rows_n integer
1	public	flight_final_set	133422
2	public	route_segment	133422
3	public	ticket	38057
4	public	seat	31410
5	public	flight	2894
6	public	airplane_flight	2894
7	public	passenger	1500
8	public	class_of_service_airplane	386
9	public	airplane	200
10	public	airport	50
11	public	class_of_service	5

Рис. 12: Количество записей в каждой таблице БД

3 Запросы к базе данных

1. Найти пассажиров которые:
 - а. Улетали из аэропорта А с классом обслуживания В без багажа;
 - б. Улетали из аэропорта А с классом обслуживания В без багажа, но фактически летели другим классом;
2. Посчитать количество самолетов, летавших из аэропорта А в аэропорт В;
3. Для каждого аэропорта посчитать:
 - а. Число проданных билетов (из этого аэропорта) за 2024 год;
 - б. Число мест в рейсах (из этого аэропорта) за 2024 год;
4. Найти число самолетов, выполнивших:
 - а. Максимальное число рейсов;
 - б. Минимальное число рейсов;
5. Посчитать число рейсов с одинаковым количеством мест;
6. Найти число пассажиров, купивших больше билетов, чем пассажир А;
7. Найти число самолетов, ни разу не бывавших в Аэропорту А;
8. а. Для каждого класса обслуживания посчитать число мест, которые были заняты;
- б. Для каждого самолета посчитать число мест, которые были заняты;

Время работы и план запросов определялось по результатам работы команды explain analyze.

Обозначения в выводе explain

План выполнения запроса представляется в виде дерева, узлы которого содержат физические операции над данными. Для каждого узла дерева указаны:

- тип операции;
- предполагаемая стоимость в условных единицах (затраты на подготовку к началу выполнения узла; полные затраты на получение всех результирующих строк);
- предполагаемое число обрабатываемых строк;
- предполагаемая ширина строки в байтах.

Операции классифицируются следующим образом:

- операции чтения таблиц

- Seq Scan on tbl — последовательный просмотр всех строк таблицы.
- Index Scan using ix on tbl — просмотр индекса на предмет строк, удовлетворяющих условию.
- Index Only Scan using ix on tbl — просмотр только индекса.
- операции соединения (двух наборов строк)
 - Nested Loop — первый дочерний узел запускается один раз, для каждой создаваемой строки соответствующая ищется во втором узле.
 - Hash Join — один из наборов соединяемых строк хешируется по ключу (узел Hash).
 - Merge Join — соединение отсортированных наборов.
- операции группировки
 - Aggregate — объединение строк (это может быть GROUP BY, UNION или SELECT DISTINCT, и/или функций типа COUNT, MAX или SUM).
 - GroupAggregate — объединение предварительно отсортированных строк (требует хранения в памяти только текущую группу).
 - HashAggregate — объединение несортированных строк (требует место для каждой записи).
- операции сортировки
 - Sort — сортировка строк, обычно результат ORDER BY.
 - Incremental Sort — сортировка строк по одному столбцу за раз.
- операции запоминания
 - Materialize — сохранение результата дочернего узла в памяти для ускорения доступа родительскими узлами.
 - Hash — хеширование набора строк для использования родительскими узлами, чаще всего JOIN.
 - Memoize — кэширование результатов поиска.

3.1 Запрос№1

3.1.1 Запрос№1.а

Формулировка запроса на естественном языке:

Найти пассажиров, которые улетали из аэропорта А, с классом обслуживания В, без багажа.

Для запроса были выбраны:

Аэропорт - Лос-Анжелес,

Класс - бизнес.

Текст запроса на языке SQL:

```

1
2 SELECT ps.* FROM passenger ps
3 JOIN ticket tick ON tick.passenger_id = ps.passenger_id
4 JOIN route_segment rs ON rs.ticket_id = tick.ticket_id
5 JOIN airport ap ON ap.airport_id = rs.airport_flight_id
6 JOIN class_of_service cs_planned ON
  ↳ cs_planned.class_of_service_id = rs.class_of_service_id
7 WHERE ap.name_airport='Лос-Анджелес'
8 AND cs_planned.name_class='Бизнес'
9 AND rs.number_of_baggage = 0;

```

Объяснение запроса:

В запросе соединяются следующие таблицы: passenger, ticket, route_segment, airport, class_of_service;

Затем устанавливаются ограничения: на название аэропорта - «Лос-Анджелес», на название класса обслуживания - «Бизнес».

Выборка происходит по всем полям из таблицы passenger.

Время выполнения запроса: 3.69 ms;

Результат выполнения запроса:

	passenger_id [PK] integer	last_name character varying (50)	first_name character varying (30)	middle_name character varying (30)	passport_series character	passport_number character	date_of_birth date
1	7	Джонс	Алексей	Николаевна	2917	7997595	1940-11-25
2	9	Кузнецов	Елена	Станиславович	2398	8506453	1923-07-13
3	23	Шмидт	Виктор	Олеговна	4154	8302216	2012-05-21
4	24	Кузнецов	Галина	Анатольевич	1580	9792807	1967-05-04
5	26	Шмидт	Валерий	Алексеевич	6891	8427851	1989-09-06
6	37	Кузнецов	Евгений	Евгеньевич	0499	3005448	1976-04-07
7	47	Богданов	Иван	Анатольевич	4625	2638843	1987-03-05
8	48	Thompson	Алексей	Сергеевич	6461	8506453	1954-04-22
9	68	Иванов	Сергей	Григорьевич	4154	9792807	1928-08-11
10	87	Богданов	Евгений	Валерьевич	4183	8302216	1988-09-06
11	98	Гарсия	Мария	Иванович	1067	8302216	1948-05-15
12	99	Taylor	Ирина	Денисович	2398	9019725	1987-03-05
13	121	Иванов	Мария	Викторович	6461	0844848	1989-09-06
14	155	Иванов	Наталья	Евгеньевич	2398	2261128	1942-06-18
15	196	Виноградов	Лариса	Максимович	6891	3817061	1915-04-22
..	...	-	-

Рис. 13: Первые 15 строк запроса №1.а

Всего было выделено 186 строк.

План запроса:

```

Nested Loop (cost=33.20..1082.92 rows=4 width=67) (actual time=1.039..6.565 rows=186
↳ loops=1)
-> Nested Loop (cost=32.92..1081.73 rows=4 width=4) (actual time=1.025..5.952
↳ rows=186 loops=1)
-> Nested Loop (cost=32.63..1080.37 rows=4 width=4) (actual time
↳ =1.017..4.989 rows=186 loops=1)
-> Nested Loop (cost=32.47..1062.76 rows=664 width=8) (actual time
↳ =0.973..4.380 rows=759 loops=1)
-> Seq Scan on airport ap (cost=0.00..1.62 rows=1 width=4) (
↳ actual time=0.016..0.024 rows=1 loops=1)

```

```

Filter: ((name_airport)::text = '          ' -          '::text)
      ↳ text)
Rows Removed by Filter: 49
-> Bitmap Heap Scan on route_segment rs (cost=32.47..1054.49
      ↳ rows=664 width=12) (actual time=0.950..4.207 rows=759 loops
      ↳ =1)
Recheck Cond: (airport_flight_id = ap.airport_id)
Filter: (number_of_baggage = 0)
Rows Removed by Filter: 2169
Heap Blocks: exact=930
-> Bitmap Index Scan on
      ↳ route_segment_airport_flight_id_idx (cost
      ↳ =0.00..32.30 rows=2668 width=0) (actual time
      ↳ =0.599..0.599 rows=2928 loops=1)
Index Cond: (airport_flight_id = ap.airport_id)
-> Memoize (cost=0.16..0.18 rows=1 width=4) (actual time=0.000..0.000
      ↳ rows=0 loops=759)
Cache Key: rs.class_of_service_id
Cache Mode: logical
Hits: 754 Misses: 5 Evictions: 0 Overflows: 0 Memory Usage: 1
      ↳ kB
-> Index Scan using class_of_service_pkey on class_of_service
      ↳ cs_planned (cost=0.15..0.17 rows=1 width=4) (actual time
      ↳ =0.007..0.008 rows=0 loops=5)
Index Cond: (class_of_service_id = rs.class_of_service_id)
Filter: ((name_class)::text = '          ' '::text)
Rows Removed by Filter: 1
-> Index Scan using ticket_pkey on ticket tick (cost=0.29..0.34 rows=1
      ↳ width=8) (actual time=0.004..0.004 rows=1 loops=186)
Index Cond: (ticket_id = rs.ticket_id)
-> Index Scan using passenger_passenger_id_idx on passenger ps (cost=0.28..0.30
      ↳ rows=1 width=67) (actual time=0.002..0.002 rows=1 loops=186)
Index Cond: (passenger_id = tick.passenger_id)

```

3.1.2 Запрос№1.b

Формулировка запроса на естественном языке:

Найти пассажиров которые, улетали из аэропорта А с классом обслуживания В, без багажа, но фактически летели другим классом;

Для запроса были так же выбраны:

Аэропорт - Лос-Анжелес,

Класс - бизнес.

Текст запроса на языке SQL:

```

1
2 SELECT ps.* , ap.name_airport, cs_planned.name_class AS PLANNED_CLASS
   ↳ , cs_final.name_class AS FINAL_CLASS FROM passenger ps
3     JOIN ticket tick ON tick.passenger_id = ps.passenger_id
4     JOIN route_segment rs ON rs.ticket_id = tick.ticket_id
5     JOIN airport ap ON ap.airport_id = rs.airport_flight_id
6     JOIN class_of_service cs_planned ON
   ↳ cs_planned.class_of_service_id = rs.class_of_service_id
7     JOIN flight_final_set ffs ON ffs.flight_final_set_id =
   ↳ rs.flight_final_set_id
8     JOIN seat s ON s.seat_id = ffs.seat_id
9     JOIN class_of_service cs_final ON cs_final.class_of_service_id
   ↳ = s.class_of_service_id

```

```

10 WHERE ap.name_airport='Лос-Анджелес'
11 AND rs.number_of_baggage = 0
12 AND cs_planned.name_class <> cs_final.name_class;

```

Объяснение запроса:

В запросе соединяются следующие таблицы: passenger, ticket, route_segment, flight_final_set и airport. Затем таблица class_of_service соединяется с таблицей route_segment как cs_planned и с таблицей flight_final_set как cs_final;

После этого устанавливаются ограничение на название аэропорта: «Лос-Анджелес» и название класса обслуживания: «бизнес». Так же ставиться условие, что cs_planned.name_class не равно cs_final.name_class (фактически летели другим классом).

Время выполнения запроса: 8.727 ms;

Результат выполнения запроса:



passenger_id	last_name	first_name	middle_name	passport_series	passport_number	date_of_birth	final_class
integer	character varying (50)	character varying (30)	character varying (30)	character	character	date	character varying (30)

Рис. 14: Результат запроса №1.b

Всего было выделено 0 строк (Все планируемые и фактические классы обслуживания совпадают).

План запроса:

```

Hash Join (cost=112.90..1890.66 rows=661 width=145) (actual time=9.546..9.553 rows=0
↳ loops=1)
  Hash Cond: (tick.passenger_id = ps.passenger_id)
  ↳ Nested Loop (cost=60.15..1836.18 rows=661 width=82) (actual time=8.792..8.798
  ↳ rows=0 loops=1)
    ↳ Hash Join (cost=59.86..1611.02 rows=661 width=82) (actual time
    ↳ =8.791..8.796 rows=0 loops=1)
      Hash Cond: (s.class_of_service_id = cs_final.class_of_service_id)
      Join Filter: ((cs_planned.name_class)::text <> (cs_final.name_class)::
      ↳ text)
      Rows Removed by Join Filter: 759
      ↳ Nested Loop (cost=33.21..1582.61 rows=664 width=86) (actual time
      ↳ =0.844..8.449 rows=759 loops=1)
        ↳ Nested Loop (cost=32.93..1374.96 rows=664 width=86) (actual
        ↳ time=0.834..6.077 rows=759 loops=1)
          ↳ Nested Loop (cost=32.63..1080.35 rows=664 width=86) (
          ↳ actual time=0.822..3.576 rows=759 loops=1)
            ↳ Nested Loop (cost=32.47..1062.76 rows=664 width
            ↳ =12) (actual time=0.804..2.951 rows=759 loops
            ↳ =1)
              ↳ Seq Scan on airport ap (cost=0.00..1.62
              ↳ rows=1 width=4) (actual time=0.013..0.019
              ↳ rows=1 loops=1)
                Filter: ((name_airport)::text = '
                ↳ '::text)
                Rows Removed by Filter: 49
              ↳ Bitmap Heap Scan on route_segment rs (cost
              ↳ =32.47..1054.49 rows=664 width=16) (
              ↳ actual time=0.783..2.800 rows=759 loops
              ↳ =1)

```

```

Recheck Cond: (airport_flight_id = ap.
    ↳ airport_id)
Filter: (number_of_baggage = 0)
Rows Removed by Filter: 2169
Heap Blocks: exact=930
-> Bitmap Index Scan on
    ↳ route_segment_airport_flight_id_idx
    ↳ (cost=0.00..32.30 rows=2668 width
    ↳ =0) (actual time=0.440..0.441 rows
    ↳ =2928 loops=1)
    Index Cond: (airport_flight_id = ap
        ↳ .airport_id)
-> Memoize (cost=0.16..0.18 rows=1 width=82) (
    ↳ actual time=0.000..0.000 rows=1 loops=759)
    Cache Key: rs.class_of_service_id
    Cache Mode: logical
    Hits: 754 Misses: 5 Evictions: 0 Overflows:
    ↳ 0 Memory Usage: 1kB
-> Index Scan using class_of_service_pkey on
    ↳ class_of_service cs_planned (cost
    ↳ =0.15..0.17 rows=1 width=82) (actual time
    ↳ =0.003..0.003 rows=1 loops=5)
    Index Cond: (class_of_service_id = rs.
        ↳ class_of_service_id)
-> Index Scan using flight_final_set_pkey on
    ↳ flight_final_set ffs (cost=0.29..0.44 rows=1 width
    ↳ =8) (actual time=0.003..0.003 rows=1 loops=759)
    Index Cond: (flight_final_set_id = rs.
        ↳ flight_final_set_id)
-> Index Scan using seat_pkey on seat s (cost=0.29..0.31 rows=1
    ↳ width=8) (actual time=0.003..0.003 rows=1 loops=759)
    Index Cond: (seat_id = ffs.seat_id)
-> Hash (cost=17.40..17.40 rows=740 width=82) (actual time
    ↳ =0.023..0.024 rows=5 loops=1)
    Buckets: 1024 Batches: 1 Memory Usage: 9kB
    ↳ Seq Scan on class_of_service cs_final (cost=0.00..17.40 rows
    ↳ =740 width=82) (actual time=0.014..0.016 rows=5 loops=1)
-> Index Scan using ticket_pkey on ticket tick (cost=0.29..0.34 rows=1
    ↳ width=8) (never executed)
    Index Cond: (ticket_id = rs.ticket_id)
-> Hash (cost=34.00..34.00 rows=1500 width=67) (actual time=0.747..0.747 rows
    ↳ =1500 loops=1)
    Buckets: 2048 Batches: 1 Memory Usage: 165kB
    ↳ Seq Scan on passenger ps (cost=0.00..34.00 rows=1500 width=67) (actual
    ↳ time=0.010..0.282 rows=1500 loops=1)

```

3.2 Запрос№2

Формулировка запроса на естественном языке:

Посчитать количество самолетов, летавших из аэропорта А в аэропорт В;

Для запроса были выбраны:

Аэропорт А - Торонто,

Аэропорт В - Пулково.

Текст запроса на языке SQL:

```

1
2 SELECT COUNT(*) from airplane
3     JOIN airplane_flight af ON af.airplane_id =
   ↳ airplane.airplane_id
4     JOIN flight f ON f.flight_id = af.flight_id

```



```

5      JOIN airport airport_flight ON airport_flight.airport_id =
      ↪ f.airport_flight_id
6      JOIN airport airport_land ON airport_land.airport_id =
      ↪ f.airport_of_landing_id
7      WHERE airport_flight.name_airport = 'Торонто' AND
      ↪ airport_land.name_airport = 'Пулково'

```

Объяснение запроса:

В запросе соединяются таблицы `airplane`, `airplane_flight`, `flight` по идентификаторам каждой из таблиц. Затем таблица `airport` как `airport_flight` соединяется с таблицей `flight` по полю `airport_flight_id` и как `airport_land` с таблицей `flight` по полю `airport_of_landing_id`

После этого устанавливаются ограничение на название аэропортов из таблиц `airport_flight` и `airport_land`: «Торонто» и «Пулково».

Для подсчета общего количества самолетов считается количество всех строк в получившейся таблице.

Время выполнения запроса: 0.158 ms;

Результат выполнения запроса:

	count	
	bigint	
1		2

Рис. 15: Результат запроса №2

План запроса:

```

=Aggregate (cost=31.97..31.98 rows=1 width=8) (actual time=0.112..0.113 rows=1 loops
↪ =1)
-> Nested Loop (cost=6.79..31.97 rows=1 width=0) (actual time=0.075..0.109 rows=2
↪ loops=1)
-> Nested Loop (cost=6.65..31.80 rows=1 width=4) (actual time=0.070..0.103
↪ rows=2 loops=1)
-> Hash Join (cost=6.37..31.46 rows=1 width=4) (actual time
↪ =0.064..0.092 rows=2 loops=1)
    Hash Cond: (f.airport_of_landing_id = airport_land.airport_id)
-> Nested Loop (cost=4.73..29.66 rows=58 width=8) (actual time
↪ =0.022..0.065 rows=55 loops=1)
-> Seq Scan on airport airport_flight (cost=0.00..1.62
↪ rows=1 width=4) (actual time=0.002..0.004 rows=1 loops
↪ =1)
    Filter: ((name_airport)::text = '                '::text
↪ )
    Rows Removed by Filter: 49
-> Bitmap Heap Scan on flight f (cost=4.73..27.45 rows=58
↪ width=12) (actual time=0.018..0.054 rows=55 loops=1)
    Recheck Cond: (airport_flight_id = airport_flight.
↪ airport_id)
    Heap Blocks: exact=21

```

```

-> Bitmap Index Scan on flight_airport_flight_id_idx
    ↳ (cost=0.00..4.71 rows=58 width=0) (actual time
    ↳ =0.010..0.010 rows=55 loops=1)
    Index Cond: (airport_flight_id = airport_flight.
    ↳ airport_id)
-> Hash (cost=1.62..1.62 rows=1 width=4) (actual time
    ↳ =0.016..0.016 rows=1 loops=1)
    Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Seq Scan on airport_airport_land (cost=0.00..1.62 rows
    ↳ =1 width=4) (actual time=0.009..0.012 rows=1 loops=1)
    Filter: ((name_airport)::text = ' '::text
    ↳ )
    Rows Removed by Filter: 49
-> Index Scan using airplane_flight_flight_id_idx on airplane_flight af
    ↳ (cost=0.28..0.33 rows=1 width=8) (actual time=0.004..0.004 rows
    ↳ =1 loops=2)
    Index Cond: (flight_id = f.flight_id)
-> Index Only Scan using airplane_pkey on airplane (cost=0.14..0.17 rows=1
    ↳ width=4) (actual time=0.003..0.003 rows=1 loops=2)
    Index Cond: (airplane_id = af.airplane_id)
    Heap Fetches: 2
Planning Time: 0.672 ms
Execution Time: 0.185 ms

```

3.3 Запрос№3

3.3.1 Запрос№3.a

Формулировка запроса на естественном языке:

Для каждого аэропорта посчитать число проданных билетов (из этого аэропорта) за 2024 год.

Текст запроса на языке SQL:

```

1
2 SELECT airport_flight.name_airport , count(*) FROM ticket tick
3 JOIN route_segment rs ON rs.ticket_id = tick.ticket_id
4 JOIN flight_final_set ffs ON ffs.flight_final_set_id =
   ↳ rs.flight_final_set_id
5 JOIN flight f ON f.flight_id = ffs.flight_id
6 JOIN airport_airport_flight ON airport_flight.airport_id =
   ↳ rs.airport_flight_id
7 WHERE EXTRACT(YEAR FROM f.date_and_time_flight) = 2024
8 GROUP BY airport_flight.name_airport

```

Объяснение запроса:

В данном запросе соединяются таблицы `ticket`, `route_segment`, `flight_final_set`, `flight` и `airport` по соответствующим идентификаторам каждой из таблиц.

После этого устанавливается ограничение на год выполнения рейса: выбираются только те рейсы, которые были выполнены в 2024 году. Это достигается с помощью функции `EXTRACT`, которая извлекает год из поля `date_and_time_flight` таблицы `flight`.

Затем выполняется группировка результатов по названиям аэропортов отправления (`airport_flight.name_airport`), и для каждой группы подсчитывается количество записей.

Запрос возвращает названия аэропортов отправления и количество рейсов, отправленных из каждого аэропорта в 2024 году.

Время выполнения запроса: 138.286 ms;

Результат выполнения запроса:

	name_airport character varying (70)	count bigint
1	Макао	41
2	Амстердам Схипхол	117
3	Париж Шарль-де-Голль	34
4	Даллас/Форт-Уэрт	20
5	Лиссабон Портела	67
6	Жуляны	20
7	Минск 2	19
8	Венеция Марко Поло	38
9	Нарита	19

Рис. 16: Результат запроса №3.а

План запроса:

```
HashAggregate (cost=2115.72..2116.21 rows=49 width=28) (actual time=134.879..134.886
  ↳ rows=9 loops=1)
  Group Key: airport_flight.name_airport
  Batches: 1 Memory Usage: 24kB
  ↳ Nested Loop (cost=5.39..2112.50 rows=645 width=20) (actual time=3.176..134.387
    ↳ rows=375 loops=1)
    ↳ Nested Loop (cost=5.24..2088.56 rows=645 width=4) (actual time
      ↳ =2.981..133.369 rows=375 loops=1)
      ↳ Nested Loop (cost=4.95..1888.17 rows=645 width=8) (actual time
        ↳ =2.131..112.483 rows=375 loops=1)
        ↳ Nested Loop (cost=4.65..1654.07 rows=645 width=4) (actual
          ↳ time=1.430..43.952 rows=375 loops=1)
          ↳ Seq Scan on flight f (cost=0.00..65.41 rows=14 width=4)
            ↳ (actual time=0.719..2.639 rows=10 loops=1)
            Filter: (EXTRACT(year FROM date_and_time_flight) =
              ↳ '2024'::numeric)
            Rows Removed by Filter: 2884
          ↳ Bitmap Heap Scan on flight_final_set ffs (cost
            ↳ =4.65..113.02 rows=46 width=8) (actual time
              ↳ =0.219..4.109 rows=38 loops=10)
            Recheck Cond: (flight_id = f.flight_id)
            Heap Blocks: exact=361
          ↳ Bitmap Index Scan on
            ↳ flight_final_set_flight_id_idx (cost=0.00..4.64
              ↳ rows=46 width=0) (actual time=0.109..0.109 rows
                ↳ =38 loops=10)
            Index Cond: (flight_id = f.flight_id)
        ↳ Index Scan using route_segment_flight_final_set_id_idx on
          ↳ route_segment rs (cost=0.29..0.35 rows=1 width=12) (actual
            ↳ time=0.181..0.181 rows=1 loops=375)
```

```

Index Cond: (flight_final_set_id = ffs.flight_final_set_id)
-> Index Only Scan using ticket_pkey on ticket tick (cost=0.29..0.31
    ↪ rows=1 width=4) (actual time=0.054..0.054 rows=1 loops=375)
    Index Cond: (ticket_id = rs.ticket_id)
    Heap Fetches: 0
-> Memoize (cost=0.15..0.17 rows=1 width=24) (actual time=0.002..0.002 rows
    ↪ =1 loops=375)
    Cache Key: rs.airport_flight_id
    Cache Mode: logical
    Hits: 365 Misses: 10 Evictions: 0 Overflows: 0 Memory Usage: 2kB
-> Index Scan using airport_pkey on airport airport_flight (cost
    ↪ =0.14..0.16 rows=1 width=24) (actual time=0.021..0.022 rows=1
    ↪ loops=10)
    Index Cond: (airport_id = rs.airport_flight_id)
Planning Time: 14.091 ms
Execution Time: 138.286 ms

```

3.3.2 Запрос№3.b

Формулировка запроса на естественном языке:

Для каждого аэропорта посчитать число мест в рейсах (из этого аэропорта) за 2024 год.

Текст запроса на языке SQL:

```

1
2 SELECT airport.name_airport, COUNT(*)
3 FROM airport
4 JOIN flight f ON f.airport_flight_id = airport.airport_id
5 JOIN airplane_flight af ON af.flight_id = f.flight_id
6 JOIN airplane ON airplane.airplane_id = af.airplane_id
7 JOIN seat s ON s.airplane_id = airplane.airplane_id
8 WHERE EXTRACT(YEAR FROM f.date_and_time_flight) = 2024
9 GROUP BY airport.name_airport

```

Объяснение запроса:

В данном запросе соединяются таблицы `airport`, `flight`, `airplane_flight`, `airplane` и `seat` по соответствующим идентификаторам каждой из таблиц.

После этого устанавливается ограничение на год выполнения рейса: выбираются только те рейсы, которые были выполнены в 2024 году. Это достигается с помощью функции `EXTRACT`, которая извлекает год из поля `date_and_time_flight` таблицы `flight`.

Затем выполняется группировка результатов по названиям аэропортов (`airport.name_airport`) и для каждой группы подсчитывается количество записей.

Запрос возвращает названия аэропортов и количество рейсов, отправленных из каждого аэропорта в 2024 году.

Время выполнения запроса: 5.624 ms;

Результат выполнения запроса:

	name_airport character varying (70) 🔒	count bigint 🔒
1	Макао	50
2	Амстердам Схипхол	450
3	Париж Шарль-де-Голль	50
4	Даллас/Форт-Уэрт	20
5	Лиссабон Портела	400
6	Жуляны	20
7	Минск 2	20
8	Венеция Марко Поло	50
9	Нарита	20

Рис. 17: Результат запроса №3.b

План запроса:

```

HashAggregate (cost=201.19..201.68 rows=49 width=28) (actual time=4.659..4.663 rows=9
↳ loops=1)
  Group Key: airport.name_airport
  Batches: 1 Memory Usage: 24kB
  ↳ Nested Loop (cost=68.14..190.19 rows=2199 width=20) (actual time=2.814..4.502
    ↳ rows=1080 loops=1)
      Join Filter: (airplane.airplane_id = s.airplane_id)
      ↳ Nested Loop (cost=67.85..119.65 rows=14 width=28) (actual time
        ↳ =2.219..3.281 rows=10 loops=1)
          ↳ Hash Join (cost=67.71..117.29 rows=14 width=24) (actual time
            ↳ =1.650..2.692 rows=10 loops=1)
              Hash Cond: (f.airport_flight_id = airport.airport_id)
              ↳ Hash Join (cost=65.58..115.13 rows=14 width=8) (actual time
                ↳ =1.614..2.651 rows=10 loops=1)
                  Hash Cond: (af.flight_id = f.flight_id)
                  ↳ Seq Scan on airplane_flight af (cost=0.00..41.94 rows
                    ↳ =2894 width=8) (actual time=0.284..0.878 rows=2894
                      ↳ loops=1)
                  ↳ Hash (cost=65.41..65.41 rows=14 width=8) (actual time
                    ↳ =0.467..0.467 rows=10 loops=1)
                      Buckets: 1024 Batches: 1 Memory Usage: 9kB
                      ↳ Seq Scan on flight f (cost=0.00..65.41 rows=14
                        ↳ width=8) (actual time=0.085..0.464 rows=10 loops
                          ↳ =1)
                          Filter: (EXTRACT(year FROM date_and_time_flight)
                            ↳ = '2024'::numeric)
                          Rows Removed by Filter: 2884
                      ↳ Hash (cost=1.50..1.50 rows=50 width=24) (actual time
                        ↳ =0.024..0.024 rows=50 loops=1)
                          Buckets: 1024 Batches: 1 Memory Usage: 11kB
                          ↳ Seq Scan on airport (cost=0.00..1.50 rows=50 width=24)
                            ↳ (actual time=0.010..0.014 rows=50 loops=1)
                      ↳ Index Only Scan using airplane_pkey on airplane (cost=0.14..0.17
                        ↳ rows=1 width=4) (actual time=0.058..0.058 rows=1 loops=10)
                          Index Cond: (airplane_id = af.airplane_id)
                          Heap Fetches: 10
                      ↳ Index Only Scan using seat_airplane_id on seat s (cost=0.29..3.08 rows
                        ↳ =157 width=4) (actual time=0.107..0.113 rows=108 loops=10)
                          Index Cond: (airplane_id = af.airplane_id)

```

Heap Fetches: 0
Planning Time: 6.442 ms
Execution Time: 5.624 ms

Результат запросов 3.a и 3.b представлен в виде диаграммы на рис. 18

Результат выполнения запроса:

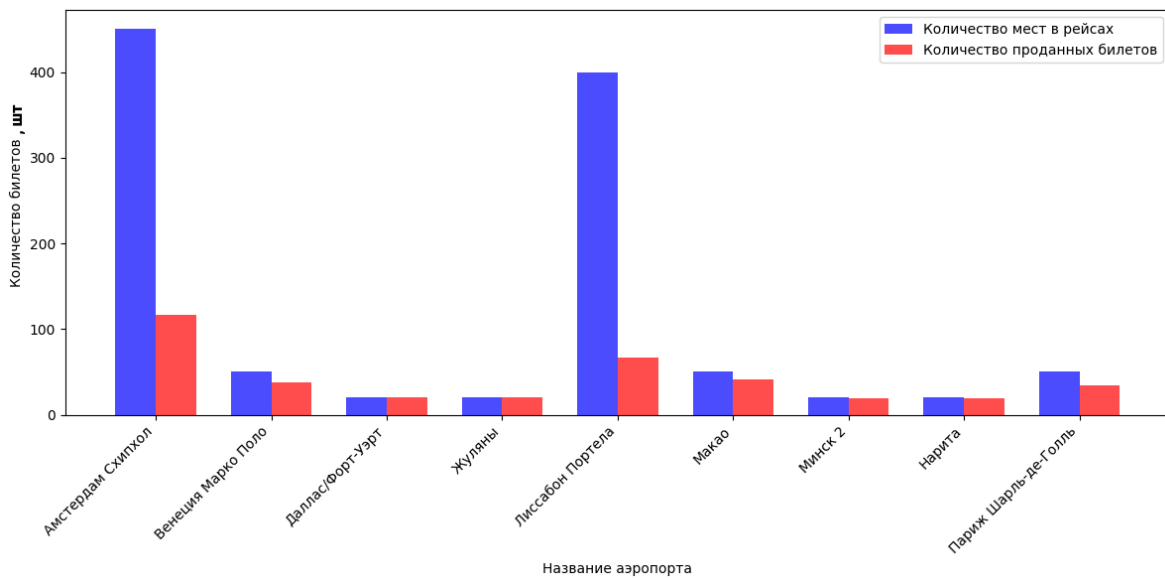


Рис. 18: Диаграмма распределения проданных билетов и общего количества мест

3.4 Запрос№4

3.4.1 Запрос№4.a

Формулировка запроса на естественном языке:

Найти число самолетов, выполнивших максимальное число рейсов.

Текст запроса на языке SQL:

```
1
2 SELECT a.airplane_id, COUNT(af.flight_id)
3 FROM airplane a
4 JOIN airplane_flight af ON a.airplane_id = af.airplane_id
5 GROUP BY a.airplane_id, a.registration_number, a.model,
6         ↪ a.number_of_baggage
7         HAVING COUNT(af.flight_id) = (SELECT count_flight
8         FROM (
9             SELECT COUNT(af.flight_id) AS count_flight FROM
10            ↪ airplane a
11            JOIN airplane_flight af ON a.airplane_id =
12            ↪ af.airplane_id
```

```

10         GROUP BY a.airplane_id
11         ORDER BY count_flight DESC
12     LIMIT 1
13 ) AS counts)
14

```

Объяснение запроса:

1. Происходит соединение таблиц airplane, airplane_flight , flight по соответствующим идентификаторам. Затем выполняется группировка полей из таблицы airplane для выполнения подсчета.

2. С помощью функции HAVING происходит сортировка сгруппированных данных по убыванию, затем возвращается максимальный элемент.

Время выполнения запроса: 2.999 ms;

Результат выполнения запроса:

	airplane_id [PK] integer	count bigint
1	146	25
2	190	25
3	98	25
4	82	25
5	40	25
6	147	25
7	79	25
8	85	25
9	72	25
10	81	25
11	19	25
12	56	25
13	29	25
14	4	25
15	138	25

Рис. 19: Первые 15 строк запроса №4.а

Всего было выделено 30 строк.

План запроса:

```

HashAggregate  (cost=144.34..146.84 rows=1 width=36) (actual time=2.886..2.908 rows
    ↳ =30 loops=1)
  Group Key: a.airplane_id
  Filter: (count(af.flight_id) = $0)
  Batches: 1  Memory Usage: 64kB
  Rows Removed by Filter: 170
  InitPlan 1 (returns $0)

```

```

-> Subquery Scan on counts (cost=73.66..73.67 rows=1 width=8) (actual time
    ↳ =1.361..1.363 rows=1 loops=1)
    -> Limit (cost=73.66..73.66 rows=1 width=12) (actual time=1.361..1.362
        ↳ rows=1 loops=1)
        -> Sort (cost=73.66..74.16 rows=200 width=12) (actual time
            ↳ =1.360..1.361 rows=1 loops=1)
            Sort Key: (count(af_1.flight_id)) DESC
            Sort Method: top-N heapsort Memory: 25kB
        -> HashAggregate (cost=70.66..72.66 rows=200 width=12) (
            ↳ actual time=1.312..1.335 rows=200 loops=1)
            Group Key: a_1.airplane_id
            Batches: 1 Memory Usage: 48kB
            -> Hash Join (cost=6.50..56.19 rows=2894 width=8) (
                ↳ actual time=0.045..0.806 rows=2894 loops=1)
                Hash Cond: (af_1.airplane_id = a_1.airplane_id)
                -> Seq Scan on airplane_flight af_1 (cost
                    ↳ =0.00..41.94 rows=2894 width=8) (actual time
                        ↳ =0.002..0.185 rows=2894 loops=1)
                -> Hash (cost=4.00..4.00 rows=200 width=4) (
                    ↳ actual time=0.039..0.039 rows=200 loops=1)
                    Buckets: 1024 Batches: 1 Memory Usage: 16kB
                    -> Seq Scan on airplane a_1 (cost
                        ↳ =0.00..4.00 rows=200 width=4) (actual
                            ↳ time=0.003..0.019 rows=200 loops=1)
            -> Hash Join (cost=6.50..56.19 rows=2894 width=32) (actual time=0.117..0.971 rows
                ↳ =2894 loops=1)
                Hash Cond: (af.airplane_id = a.airplane_id)
                -> Seq Scan on airplane_flight af (cost=0.00..41.94 rows=2894 width=8) (
                    ↳ actual time=0.005..0.199 rows=2894 loops=1)
                -> Hash (cost=4.00..4.00 rows=200 width=28) (actual time=0.105..0.106 rows
                    ↳ =200 loops=1)
                    Buckets: 1024 Batches: 1 Memory Usage: 21kB
                    -> Seq Scan on airplane a (cost=0.00..4.00 rows=200 width=28) (actual
                        ↳ time=0.005..0.029 rows=200 loops=1)
Planning Time: 0.415 ms
Execution Time: 2.962 ms

```

3.4.2 Запрос№4.b

Формулировка запроса на естественном языке:

Найти число самолетов, выполнивших минимальное количество рейсов.

Текст запроса на языке SQL:

```

1
2
3 SELECT a.airplane_id, COUNT(af.flight_id)
4 FROM airplane a
5 JOIN airplane_flight af ON a.airplane_id = af.airplane_id
6 GROUP BY a.airplane_id, a.registration_number, a.model,
    ↳ a.number_of_baggage
7 HAVING COUNT(af.flight_id) =(SELECT count_flight
8 FROM (
9 SELECT COUNT(af.flight_id)AS count_flight FROM
    ↳ airplane a
10 JOIN airplane_flight af ON a.airplane_id =
    ↳ af.airplane_id
11 GROUP BY a.airplane_id
12 ORDER BY count_flight ASC

```



```

13         LIMIT 1
14     ) AS counts)
15
16
17
18

```

Объяснение запроса: Запрос выполняется так же, как в пункте Запрос №4.a, с той разницей, что фильтрация происходит по минимальному количеству самолетов.

Время выполнения запроса: 2.276 ms;

Результат выполнения запроса:

	airplane_id [PK] integer	count bigint
1	52	5
2	92	5
3	101	5
4	60	5
5	156	5
6	59	5
7	197	5
8	65	5
9	173	5
10	188	5
11	26	5
12	187	5
13	77	5
14	3	5
15	198	5

Рис. 20: Первые 15 строк запроса №4.b

Всего было выделено 41 строка. **План запроса:**

```

HashAggregate (cost=145.83..148.33 rows=1 width=36) (actual time=2.876..2.896 rows
    ↳ =30 loops=1)
  Group Key: a.airplane_id
  Filter: (count(af.flight_id) = $0)
  Batches: 1 Memory Usage: 64kB
  Rows Removed by Filter: 170
  InitPlan 1 (returns $0)
    ↳ Aggregate (cost=75.16..75.17 rows=1 width=8) (actual time=1.237..1.239 rows
        ↳ =1 loops=1)
      ↳ HashAggregate (cost=70.66..72.66 rows=200 width=12) (actual time
          ↳ =1.205..1.225 rows=200 loops=1)
        Group Key: a_1.airplane_id
        Batches: 1 Memory Usage: 48kB

```

```

-> Hash Join (cost=6.50..56.19 rows=2894 width=8) (actual time
    ↳ =0.050..0.743 rows=2894 loops=1)
    Hash Cond: (af_1.airplane_id = a_1.airplane_id)
    -> Seq Scan on airplane_flight af_1 (cost=0.00..41.94 rows
        ↳ =2894 width=8) (actual time=0.004..0.212 rows=2894 loops
        ↳ =1)
    -> Hash (cost=4.00..4.00 rows=200 width=4) (actual time
        ↳ =0.040..0.041 rows=200 loops=1)
        Buckets: 1024 Batches: 1 Memory Usage: 16kB
        -> Seq Scan on airplane a_1 (cost=0.00..4.00 rows=200
            ↳ width=4) (actual time=0.003..0.020 rows=200 loops
            ↳ =1)
-> Hash Join (cost=6.50..56.19 rows=2894 width=32) (actual time=0.048..1.076 rows
    ↳ =2894 loops=1)
    Hash Cond: (af.airplane_id = a.airplane_id)
    -> Seq Scan on airplane_flight af (cost=0.00..41.94 rows=2894 width=8) (
        ↳ actual time=0.005..0.250 rows=2894 loops=1)
    -> Hash (cost=4.00..4.00 rows=200 width=28) (actual time=0.038..0.039 rows
        ↳ =200 loops=1)
        Buckets: 1024 Batches: 1 Memory Usage: 21kB
        -> Seq Scan on airplane a (cost=0.00..4.00 rows=200 width=28) (actual
            ↳ time=0.003..0.016 rows=200 loops=1)
Planning Time: 0.431 ms
Execution Time: 2.945 ms

```

3.5 Запрос№5

Формулировка запроса на естественном языке:

Посчитать число рейсов с одинаковым количеством мест;

Текст запроса на языке SQL:

```

1
2 SELECT amount_seat ,count(*) FROM(
3 SELECT f.flight_id, count(*)AS amount_seat from flight f
4 JOIN airplane_flight af ON af.flight_id = f.flight_id
5 JOIN airplane a ON a.airplane_id = af.airplane_id
6 JOIN seat s ON s.airplane_id = a.airplane_id
7 GROUP BY f.flight_id
8 ) AS airplane_seat
9 GROUP BY amount_seat
10

```

Объяснение запроса:

1. Используется подзапрос для подсчета количества мест для каждого самолета. Для этого объединяются таблицы flight, airplane_flight, airplane, seat и происходит группировка по количеству мест.

2. Затем происходит группировка и подсчет строк по одинаковому количеству мест.

Время выполнения запроса: 100.289 ms;

Результат выполнения запроса:

	amount_seat bigint	count bigint
1	400	765
2	50	766
3	150	508
4	20	855

Рис. 21: Результат запроса №5

На рис. 22 представлен результат запроса в виде диаграммы:

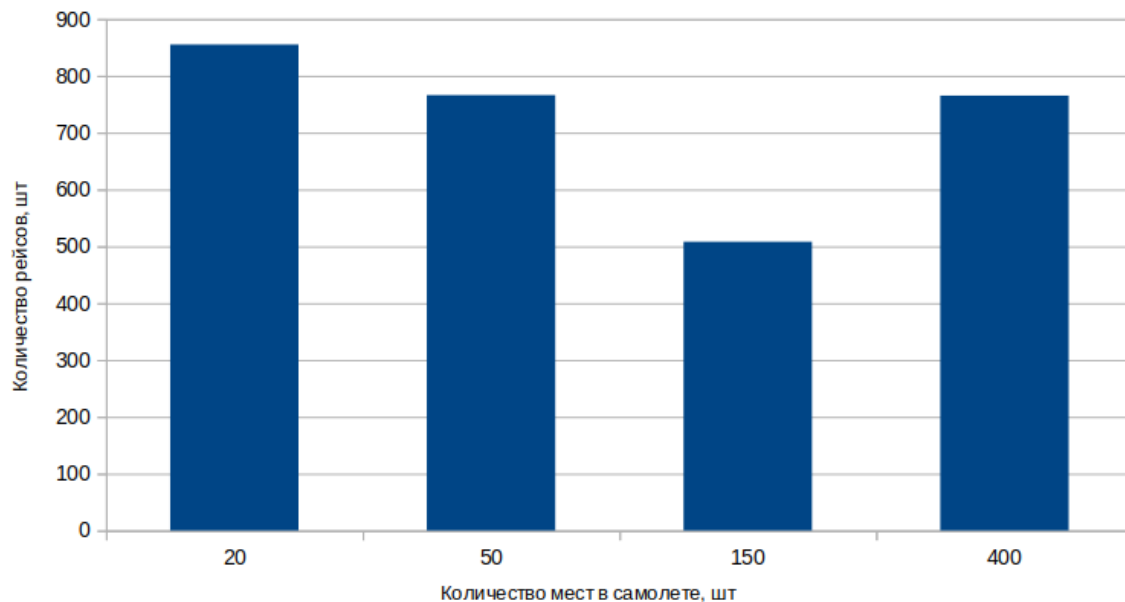


Рис. 22: Результат запроса №5. Диаграмма.

План запроса:

```

HashAggregate (cost=8189.29..8191.29 rows=200 width=16) (actual time=100.230..100.234
  ↳ rows=4 loops=1)
  Group Key: count(*)
  Batches: 1 Memory Usage: 40kB
  ↳ HashAggregate (cost=8116.94..8145.88 rows=2894 width=12) (actual time
    ↳ =99.671..99.956 rows=2894 loops=1)
    Group Key: f.flight_id
    Batches: 1 Memory Usage: 369kB
    ↳ Hash Join (cost=187.09..5844.42 rows=454503 width=4) (actual time
      ↳ =2.875..45.432 rows=437600 loops=1)

```

```

Hash Cond: (s.airplane_id = a.airplane_id)
-> Seq Scan on seat s (cost=0.00..484.10 rows=31410 width=4) (actual
    ↪ time=0.006..2.553 rows=31410 loops=1)
-> Hash (cost=150.92..150.92 rows=2894 width=12) (actual time
    ↪ =2.864..2.867 rows=2894 loops=1)
    Buckets: 4096 Batches: 1 Memory Usage: 157kB
    -> Hash Join (cost=93.62..150.92 rows=2894 width=12) (actual
        ↪ time=0.718..2.309 rows=2894 loops=1)
        Hash Cond: (af.airplane_id = a.airplane_id)
        -> Hash Join (cost=87.12..136.66 rows=2894 width=8) (
            ↪ actual time=0.658..1.648 rows=2894 loops=1)
            Hash Cond: (af.flight_id = f.flight_id)
            -> Seq Scan on airplane_flight af (cost=0.00..41.94
                ↪ rows=2894 width=8) (actual time=0.002..0.225
                ↪ rows=2894 loops=1)
            -> Hash (cost=50.94..50.94 rows=2894 width=4) (
                ↪ actual time=0.650..0.650 rows=2894 loops=1)
                Buckets: 4096 Batches: 1 Memory Usage: 134kB
                -> Seq Scan on flight f (cost=0.00..50.94 rows
                    ↪ =2894 width=4) (actual time=0.002..0.287
                    ↪ rows=2894 loops=1)
        -> Hash (cost=4.00..4.00 rows=200 width=4) (actual time
            ↪ =0.057..0.057 rows=200 loops=1)
            Buckets: 1024 Batches: 1 Memory Usage: 16kB
            -> Seq Scan on airplane a (cost=0.00..4.00 rows=200
                ↪ width=4) (actual time=0.004..0.027 rows=200
                ↪ loops=1)

```

Planning Time: 1.768 ms
Execution Time: 100.289 ms

3.6 Запрос№6

Формулировка запроса на естественном языке:

Найти число пассажиров, купивших больше билетов, чем пассажир А;

В качестве пассажира А был выбран пассажир с индексом 1.

Текст запроса на языке SQL:

```

1
2 SELECT p.passenger_id, p.first_name, p.last_name,
3     ↪ COUNT(t.ticket_id) AS ticket_count FROM passenger p
4     JOIN ticket t ON t.passenger_id = p.passenger_id
5     GROUP BY p.passenger_id, p.first_name, p.last_name
6     HAVING COUNT(t.ticket_id) > (SELECT COUNT(t.ticket_id) FROM
    ↪ ticket t WHERE t.passenger_id = 1)

```

Объяснение запроса:

1. Соединение таблицы passenger с таблицей ticket.

2. Результаты группируются по идентификатору пассажира (passenger_id), имени (first_name) и фамилии (last_name). Для каждого пассажира подсчитывается количество билетов (ticket_count), купленных им.

3. После группировки происходит фильтрация результатов с помощью HAVING. Для каждого пассажира проверяется, что количество купленных им билетов больше чем у пассажира с id = 1.

Время выполнения запроса: 60.121 ms;

Результат выполнения запроса:

	passenger_id [PK] integer	first_name character varying (30)	last_name character varying (50)	ticket_count bigint
1	652	Елена	Орлов	35
2	951	Михаил	Волков	40
3	839	Ольга	Джонс	35
4	350	Евгений	Смирнов	39
5	758	Олег	Тарасов	49
6	1108	Валерий	Мартинес	37
7	1128	Александра	Смирнов	48
8	1284	Ольга	Lee	42
9	278	Владимир	Джонс	35
10	946	Олег	Мартинес	40
11	292	Иван	Богданов	47
12	1331	Галина	Lee	39
13	929	Сергей	Виноградов	41
14	1493	Максим	Богданов	44
15	764	Ольга	Беляев	37

Рис. 23: Первые 15 строк результата выполнения запроса №6

Всего было выделено 498 строк.

План запроса:

```
HashAggregate (cost=1537.57..1556.32 rows=500 width=37) (actual time=55.594..55.815
  ↳ rows=498 loops=1)
  Group Key: p.passenger_id
  Filter: (count(t.ticket_id) > $0)
  Batches: 1 Memory Usage: 321kB
  Rows Removed by Filter: 1002
  InitPlan 1 (returns $0)
    ↳ Aggregate (cost=644.77..644.78 rows=1 width=8) (actual time=2.928..2.929 rows
      ↳ =1 loops=1)
      ↳ Seq Scan on ticket t_1 (cost=0.00..644.71 rows=24 width=4) (actual time
        ↳ =0.006..2.922 rows=33 loops=1)
        Filter: (passenger_id = 1)
        Rows Removed by Filter: 38024
    ↳ Hash Join (cost=52.75..702.50 rows=38057 width=33) (actual time=7.223..44.882
      ↳ rows=38057 loops=1)
      Hash Cond: (t.passenger_id = p.passenger_id)
      ↳ Seq Scan on ticket t (cost=0.00..549.57 rows=38057 width=8) (actual time
        ↳ =0.375..16.904 rows=38057 loops=1)
      ↳ Hash (cost=34.00..34.00 rows=1500 width=29) (actual time=6.279..6.280
        ↳ rows=1500 loops=1)
        Buckets: 2048 Batches: 1 Memory Usage: 108kB
        ↳ Seq Scan on passenger p (cost=0.00..34.00 rows=1500 width=29) (
          ↳ actual time=0.751..4.119 rows=1500 loops=1)
Planning Time: 12.774 ms
Execution Time: 60.121 ms
```

3.7 Запрос №7

Формулировка запроса на естественном языке:

Найти самолеты, ни разу не бывавшие в аэропорту А.
В качестве аэропорта А был выбран аэропорт «Сингапур Чанги»
Текст запроса на языке SQL:

```
1
2 SELECT plan.registration_number
3 FROM airplane plan
4 WHERE plan.airplane_id NOT IN(
5 SELECT DISTINCT plan.airplane_id FROM airplane plan
6     JOIN airplane_flight af ON af.airplane_id = plan.airplane_id
7     JOIN flight f ON f.flight_id = af.flight_id
8     JOIN airport port_flight ON port_flight.airport_id =
9     ↪ f.airport_flight_id
10    JOIN airport port_landing ON port_landing.airport_id =
11    ↪ f.airport_of_landing_id
12    WHERE port_landing.name_airport = 'Сингапур Чанги'
13    OR port_flight.name_airport = 'Сингапур Чанги'
14    ORDER BY plan.airplane_id
15 )
```

Объяснение запроса:

Для выполнения данного запроса:

1. Создается подзапрос, соединяющий таблицы airplane, airplane_flight, flight, airport (два раза, как аэропорт прибытия и как аэропорт отправления). Выбираются те самолеты, у которых аэропорт прибытия или аэропорт отправления был равен «Сингапур Чанги».

2. В основном запросе с помощью NOT IN исключаются записи, присутствующие в подзапросе.

Время выполнения запроса: 12.809 ms;

Результат выполнения запроса:

	registration_number character varying (15) 🔒
1	A20A4U7X
2	TCCQJPH
3	RAKJVEH8M
4	NS1CJP63
5	CNGMR21
6	BVTOXWZE
7	B9UCM3
8	BN1NI7
9	URWTXDUB
10	G3NFCPK
11	A2AYLQ
12	A0RZFZ1
13	RA7X0CRE
14	RAUZCB8TZ
15	ABB00T

Рис. 24: Первые 15 строк результата выполнения запроса №7

Всего было выделено 111 строк.

План запроса:

```

Filter: (NOT (hashed SubPlan 1))
Rows Removed by Filter: 89
SubPlan 1
-> Unique (cost=120.83..121.40 rows=115 width=4) (actual time=12.555..12.573
    ↪ rows=89 loops=1)
    -> Sort (cost=120.83..121.12 rows=115 width=4) (actual time=12.554..12.562
        ↪ rows=117 loops=1)
        Sort Key: plan_1.airplane_id
        Sort Method: quicksort Memory: 30kB
    -> Hash Join (cost=11.03..116.89 rows=115 width=4) (actual time
        ↪ =0.873..12.519 rows=117 loops=1)
        Hash Cond: (af.airplane_id = plan_1.airplane_id)
    -> Nested Loop (cost=4.53..110.08 rows=115 width=4) (actual
        ↪ time=0.667..12.271 rows=117 loops=1)
        -> Hash Join (cost=4.25..71.06 rows=115 width=4) (actual
            ↪ time=0.444..5.850 rows=117 loops=1)
            Hash Cond: (f.airport_of_landing_id = port_landing.
                ↪ airport_id)
            Join Filter: (((port_landing.name_airport)::text = '
                ↪ ')::text) OR ((
                ↪ port_flight.name_airport)::text = '
                ↪ ')::text))

```

```

Rows Removed by Join Filter: 2777
-> Hash Join (cost=2.12..61.31 rows=2894 width=28)
    ↪ (actual time=0.429..5.382 rows=2894 loops=1)
    Hash Cond: (f.airport_flight_id = port_flight.
        ↪ airport_id)
    -> Seq Scan on flight f (cost=0.00..50.94
        ↪ rows=2894 width=12) (actual time
        ↪ =0.313..2.581 rows=2894 loops=1)
    -> Hash (cost=1.50..1.50 rows=50 width=24) (
        ↪ actual time=0.010..0.011 rows=50 loops
        ↪ =1)
        Buckets: 1024 Batches: 1 Memory Usage:
        ↪ 11kB
        -> Seq Scan on airport port_flight (
            ↪ cost=0.00..1.50 rows=50 width=24)
            ↪ (actual time=0.002..0.005 rows=50
            ↪ loops=1)
    -> Hash (cost=1.50..1.50 rows=50 width=24) (actual
        ↪ time=0.011..0.011 rows=50 loops=1)
        Buckets: 1024 Batches: 1 Memory Usage: 11kB
        -> Seq Scan on airport port_landing (cost
            ↪ =0.00..1.50 rows=50 width=24) (actual
            ↪ time=0.002..0.006 rows=50 loops=1)
    -> Index Scan using airplane_flight_flight_id_idx on
        ↪ airplane_flight af (cost=0.28..0.33 rows=1 width=8)
        ↪ (actual time=0.054..0.054 rows=1 loops=117)
        Index Cond: (flight_id = f.flight_id)
-> Hash (cost=4.00..4.00 rows=200 width=4) (actual time
    ↪ =0.030..0.031 rows=200 loops=1)
    Buckets: 1024 Batches: 1 Memory Usage: 16kB
    -> Seq Scan on airplane plan_1 (cost=0.00..4.00 rows=200
        ↪ width=4) (actual time=0.002..0.014 rows=200 loops
        ↪ =1)

```

Planning Time: 6.222 ms
Execution Time: 12.809 ms

3.8 Запрос№8

Формулировка запроса на естественном языке:

Для каждого класса обслуживания и каждого самолета посчитать число мест, которые были заняты.

Текст запроса на языке SQL:

```

1
2 SELECT a.airplane_id, cs.name_class,
3     (
4         SELECT COUNT(*)
5         FROM airplane_flight af
6         JOIN flight f ON f.flight_id = af.flight_id
7         JOIN flight_final_set ffs ON ffs.flight_id =
8             ↪ f.flight_id
9         JOIN route_segment rs ON rs.flight_final_set_id =
10             ↪ ffs.flight_final_set_id
11         JOIN class_of_service cs1 ON cs1.class_of_service_id =
12             ↪ rs.class_of_service_id
13     WHERE af.airplane_id = a.airplane_id
14     AND cs1.class_of_service_id = cs.class_of_service_id

```



```

12         ) AS count_occupied
13 FROM airplane a
14 CROSS JOIN class_of_service cs
15 ORDER BY a.airplane_id;
16
17

```

Объяснение запроса:

Для выполнения данного запроса:

1. Создается подзапрос внутри SELECT, соединяющий таблицы flight, airplane_flight, flight_final_set, route_segment, class_of_service.
2. В основном запросе соединяются результаты подзапроса с таблицей class_of_service с помощью CROSS JOIN.

Время выполнения запроса: 2349.247 ms ms;

Результат выполнения запроса:

	airplane_id integer	name_class character varying (30)	count_occupied bigint
1	1	Эконом	0
2	1	Премиум-эконом	0
3	1	Комфорт	0
4	1	Бизнес	233
5	1	Первый	0
6	2	Эконом	0
7	2	Премиум-эконом	0
8	2	Комфорт	1421
9	2	Бизнес	364
10	2	Первый	0
11	3	Эконом	223
12	3	Премиум-эконом	0
13	3	Комфорт	43
14	3	Бизнес	25
15	3	Первый	0

Рис. 25: Первые 15 строк результата выполнения запроса №8

Всего было выделено 1000(200 самолетов,50 классов) строк.

На рис. 26 - На рис. 29 представленные результаты запроса в виде диаграммы.

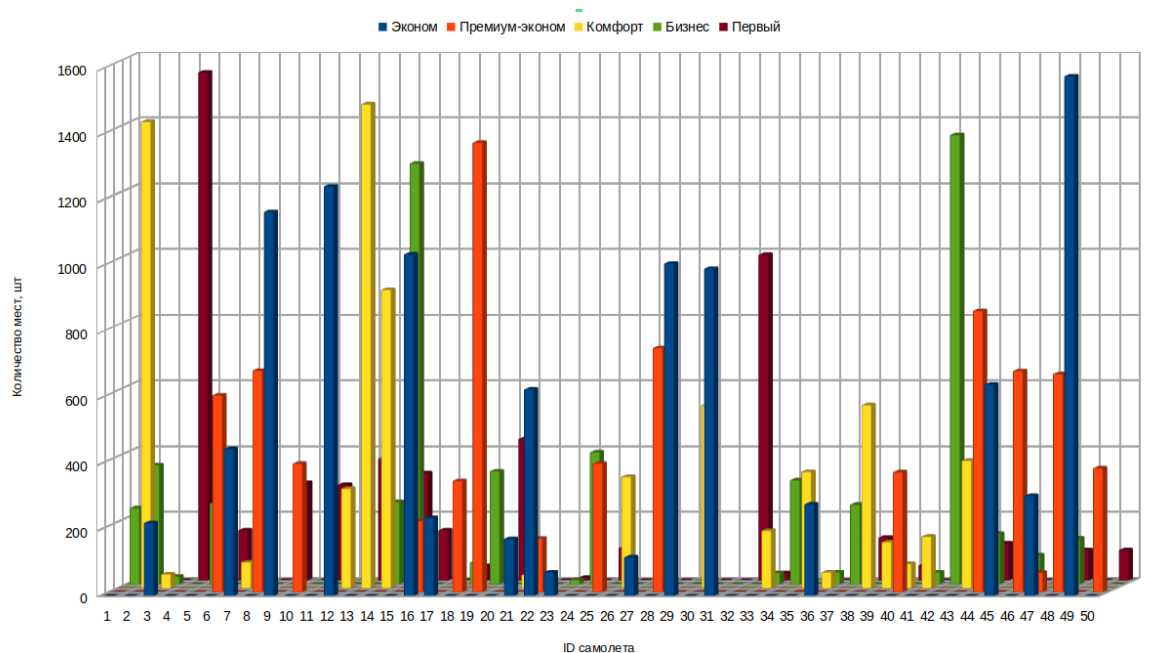


Рис. 26: Результат запроса №8. Диаграмма. ID №1 - №50.

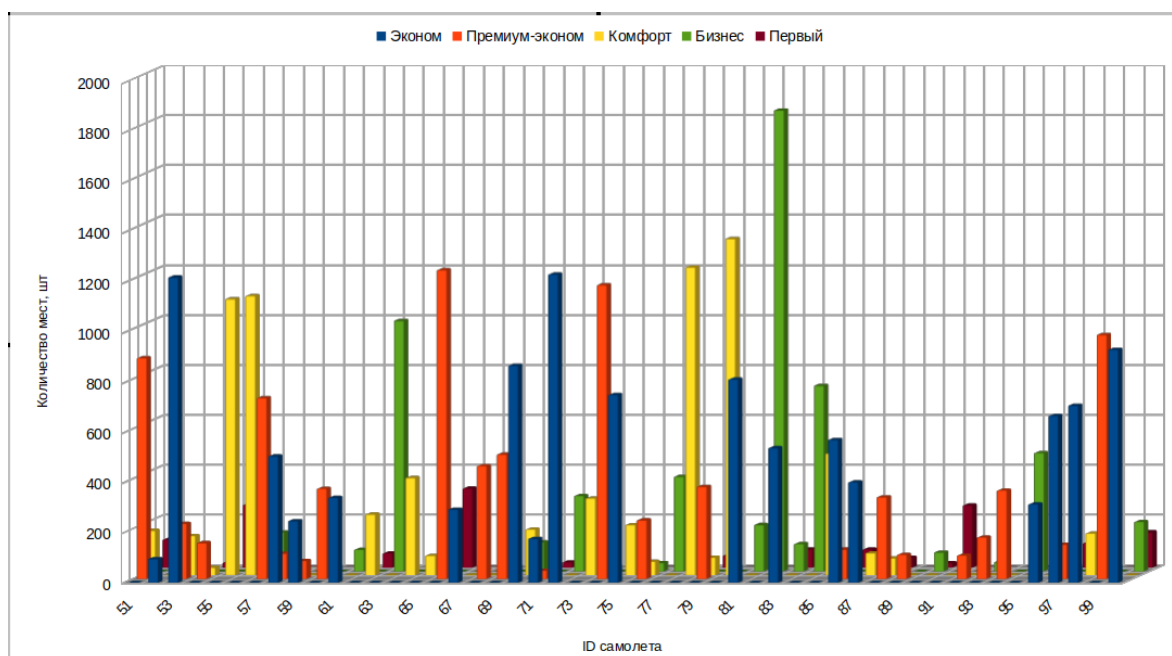


Рис. 27: Результат запроса №8. Диаграмма. ID №51 - №100.

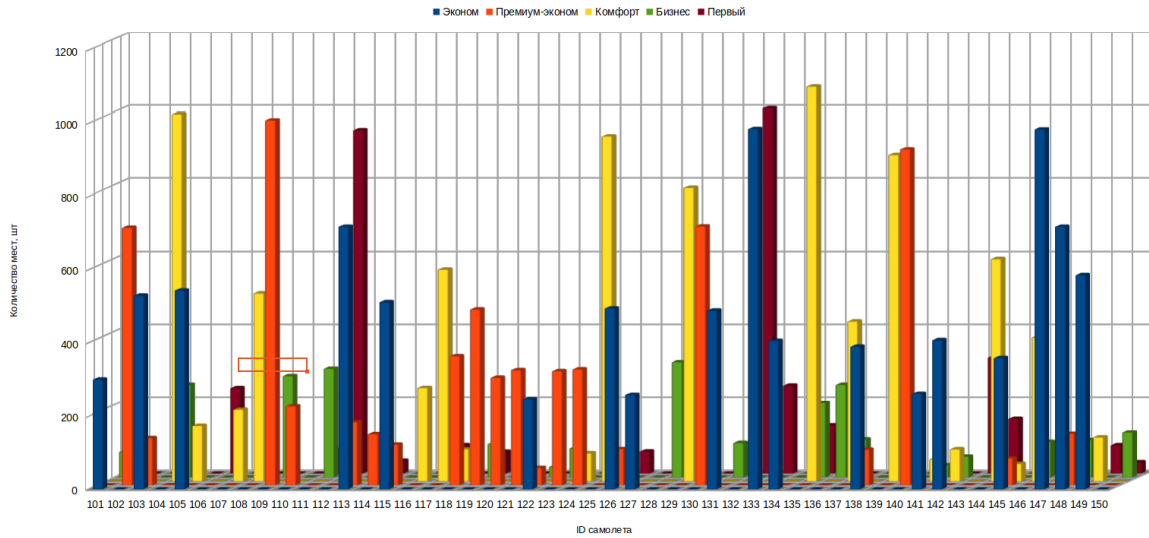


Рис. 28: Результат запроса №8. Диаграмма. ID №101 - №150.

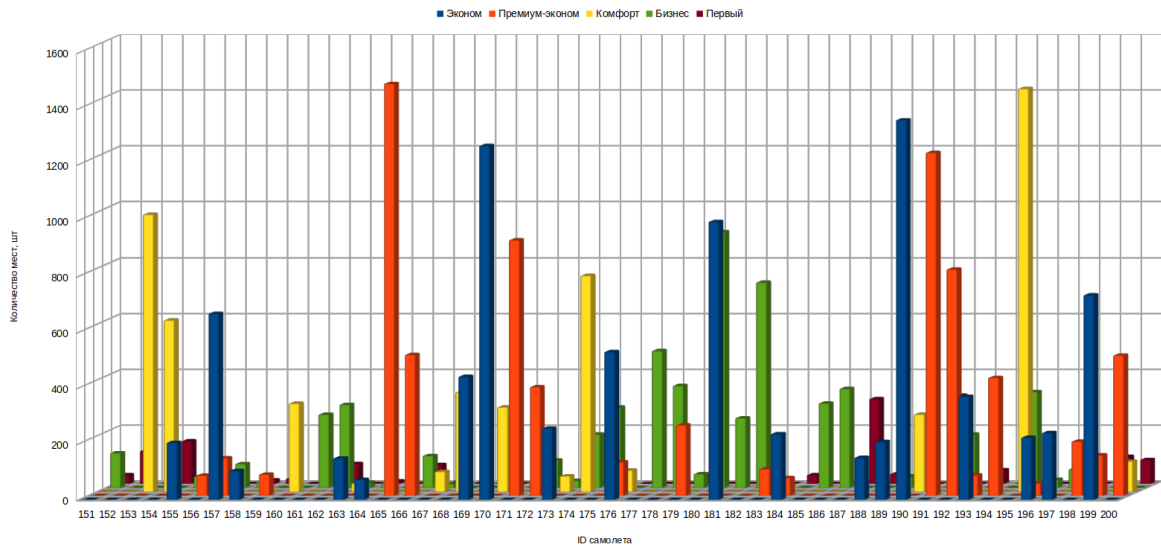


Рис. 29: Результат запроса №8. Диаграмма. ID №151 - №200.

План запроса:

```

Nested Loop (cost=0.14..48688583.16 rows=148000 width=90) (actual time
  ↳ =217.432..2346.958 rows=1000 loops=1)
  ↳ Index Only Scan using airplane_pkey on airplane a (cost=0.14..16.14 rows=200
    ↳ width=4) (actual time=0.022..0.310 rows=200 loops=1)
    Heap Fetches: 200
  ↳ Materialize (cost=0.00..21.10 rows=740 width=82) (actual time=1.081..1.085
    ↳ rows=5 loops=200)
    ↳ Seq Scan on class_of_service cs (cost=0.00..17.40 rows=740 width=82) (
      ↳ actual time=216.176..216.182 rows=5 loops=1)
SubPlan 1
  ↳ Aggregate (cost=328.95..328.96 rows=1 width=8) (actual time=2.126..2.126
    ↳ rows=1 loops=1000)

```

```

-> Nested Loop (cost=1.30..328.63 rows=129 width=0) (actual time
    ↳ =1.264..2.114 rows=133 loops=1000)
    -> Index Only Scan using class_of_service_pkey on class_of_service
        ↳ cs1 (cost=0.15..8.17 rows=1 width=4) (actual time=0.003..0.003
        ↳ rows=1 loops=1000)
        Index Cond: (class_of_service_id = cs.class_of_service_id)
        Heap Fetches: 1000
    -> Nested Loop (cost=1.15..319.17 rows=129 width=4) (actual time
        ↳ =1.260..2.099 rows=133 loops=1000)
        -> Nested Loop (cost=0.85..83.46 rows=645 width=4) (actual
            ↳ time=0.008..0.530 rows=667 loops=1000)
            -> Nested Loop (cost=0.56..44.69 rows=14 width=8) (
                ↳ actual time=0.005..0.033 rows=14 loops=1000)
                -> Index Only Scan using airplane_flight_pkey on
                    ↳ airplane_flight af (cost=0.28..4.53 rows=14
                    ↳ width=4) (actual time=0.003..0.007 rows=14
                    ↳ loops=1000)
                    Index Cond: (airplane_id = a.airplane_id)
                    Heap Fetches: 0
                -> Index Only Scan using flight_pkey on flight f
                    ↳ (cost=0.28..2.87 rows=1 width=4) (actual time
                    ↳ =0.001..0.001 rows=1 loops=14470)
                    Index Cond: (flight_id = af.flight_id)
                    Heap Fetches: 0
            -> Index Scan using flight_final_set_flight_id_idx on
                ↳ flight_final_set ffs (cost=0.29..2.31 rows=46
                ↳ width=8) (actual time=0.002..0.031 rows=46 loops
                ↳ =14470)
                Index Cond: (flight_id = f.flight_id)
        -> Index Scan using route_segment_flight_final_set_id_idx on
            ↳ route_segment rs (cost=0.29..0.36 rows=1 width=8) (
            ↳ actual time=0.002..0.002 rows=0 loops=667110)
            Index Cond: (flight_final_set_id = ffs.
            ↳ flight_final_set_id)
            Filter: (class_of_service_id = cs.class_of_service_id)
            Rows Removed by Filter: 1

```

Planning Time: 1.338 ms

JIT:

Functions: 29

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

Timing: Generation 1.622 ms, Inlining 10.035 ms, Optimization 122.764 ms, Emission

↳ 83.423 ms, Total 217.843 ms

Execution Time: 2349.247 ms

4 Заключение

В ходе данной работы были получены навыки работы с PostgreSQL 16.2 , pgAdmin4, изучен синтаксис и особенности языков определения данных и манипулирования данными. Также были получены аналитические навыки, построение ER-диаграмм, схем базы данных и проектирование базы данных.

В рамках разработки базы данных пассажирских авиаперевозок было сделано:

1. Разработана ER-диаграмма(14 сущностей,14 связей, 19 атрибутов), описывающая процессы, протекающие в предметной области, а также схема объектов;
2. На основе построенной диаграммы и поставленных целей создания базы данных - была спроектирована и реализована база данных, содержащая 11 таблиц и 42 столбца;
3. База данных была заполнена случайными данными, сгенерированными программой на языке программирования Python (Приложение №2);
4. Реализованы 11 запросов базе данных. К каждому из запросов приложен вывод EXPLAIN, указано время выполнения и план запроса на естественном языке.

Список источников

1. Рогов Е.В. PostgreSQL 16 изнутри. - М.: ДМК Пресс, 2024. - С.664. [Электронный ресурс]. - URL: https://edu.postgrespro.ru/postgresql_internals-16.pdf (дата обращения: 20.05.2024)
2. Документация pgMustard — визуализатора результатов EXPLAIN [Электронный ресурс]. <https://www.pgmustard.com/docs/explain> (дата обращения: 29.05.2024).

Приложение №1. Программа создания базы данных

```
1
2
3 DROP TABLE IF EXISTS passenger CASCADE;
4 DROP TABLE IF EXISTS ticket CASCADE;
5 DROP TABLE IF EXISTS route_segment CASCADE;
6 DROP TABLE IF EXISTS class_of_service CASCADE;
7 DROP TABLE IF EXISTS flight_final_set CASCADE;
8 DROP TABLE IF EXISTS seat CASCADE;
9 DROP TABLE IF EXISTS class_of_service_airplane CASCADE;
10 DROP TABLE IF EXISTS airplane CASCADE;
11 DROP TABLE IF EXISTS airplane_flight CASCADE;
12 DROP TABLE IF EXISTS flight CASCADE ;
13 DROP TABLE IF EXISTS airport CASCADE;
14
15 CREATE TABLE passenger(
16     passenger_id SERIAL PRIMARY KEY,
17     last_name varchar(50),
18     first_name varchar(30),
19     middle_name varchar(30),
20     passport_series char(4) CHECK(passport_series ~ '[0-9]{4}'),
21     passport_number char(7) CHECK(passport_number ~ '[0-9]{7}'),
22     date_of_birth DATE CHECK (date_of_birth >= '1890-01-01' AND
        ↪ date_of_birth <= CURRENT_DATE)
23 );
24
25 CREATE TABLE ticket(
26     ticket_id SERIAL PRIMARY KEY,
27     passenger_id INT
28 );
29
30 CREATE TABLE route_segment(
31     route_segment_id SERIAL PRIMARY KEY,
32     ticket_id INT,
33     airport_flight_id INT,
34
35     airport_of_landing_id INT,
36     class_of_service_id INT,
37     flight_final_set_id INT,
38     number_of_baggage INT CHECK (number_of_baggage>=0 AND
        ↪ number_of_baggage<100)
39 );
40
41 CREATE TABLE flight_final_set(
```

```

42     flight_final_set_id SERIAL PRIMARY KEY,
43     flight_id INT,
44     seat_id INT,
45     number_of_baggage INT CHECK (number_of_baggage>=0 AND
    ↪ number_of_baggage<100)
46 );
47
48
49
50 CREATE TABLE class_of_service(
51     class_of_service_id SERIAL PRIMARY KEY,
52     name_class varchar(30)
53 );
54
55 CREATE TABLE seat(
56     seat_id SERIAL PRIMARY KEY,
57     class_of_service_id INT,
58     airplane_id INT,
59     number_of_seat varchar(5)
60 );
61
62 CREATE TABLE class_of_service_airplane(
63     airplane_id INT,
64     class_of_service_id INT,
65     PRIMARY KEY (airplane_id, class_of_service_id)
66 );
67
68 CREATE TABLE airplane(
69     airplane_id SERIAL PRIMARY KEY,
70     model varchar(80),
71     registration_number varchar(15),
72     number_of_baggage INT CHECK (number_of_baggage>=0 AND
    ↪ number_of_baggage<250000)
73 );
74
75 CREATE TABLE airplane_flight(
76     airplane_id INT,
77     flight_id INT,
78     PRIMARY KEY (airplane_id, flight_id)
79 );
80
81 CREATE TABLE flight(
82     flight_id SERIAL PRIMARY KEY,
83     airport_flight_id INT,
84     airport_of_landing_id INT,

```



```

85     date_and_time_flight TIMESTAMP CHECK (date_and_time_flight >=
      ↳ '1903-12-17 00:00:00' AND date_and_time_flight <= '2124-04-29
      ↳ 00:00:00'),
86     date_and_time_landing TIMESTAMP CHECK (date_and_time_landing >=
      ↳ '1903-12-17 00:00:00' AND date_and_time_landing <= '2124-04-29
      ↳ 00:00:00')
87 );
88
89 CREATE TABLE airport(
90     airport_id SERIAL PRIMARY KEY,
91     name_airport varchar(70),
92     country varchar(60)
93 );
94
95 CREATE TABLE seat(
96     seat_id SERIAL PRIMARY KEY,
97     class_of_service_id INT,
98     airplane_id INT,
99     number_of_seat varchar(5)
100 );
101
102 ALTER TABLE route_segment
103     ADD FOREIGN KEY (ticket_id) REFERENCES ticket(ticket_id)
104     ON UPDATE CASCADE ON DELETE NO ACTION,
105     ADD FOREIGN KEY (airport_flight_id) REFERENCES
      ↳ airport(airport_id)
106     ON UPDATE CASCADE ON DELETE NO ACTION,
107     ADD FOREIGN KEY (airport_of_landing_id) REFERENCES
      ↳ airport(airport_id)
108     ON UPDATE CASCADE ON DELETE NO ACTION,
109     ADD FOREIGN KEY (class_of_service_id) REFERENCES
      ↳ class_of_service(class_of_service_id)
110     ON UPDATE CASCADE ON DELETE NO ACTION,
111     ADD FOREIGN KEY (flight_final_set_id) REFERENCES
      ↳ flight_final_set(flight_final_set_id)
112     ON UPDATE CASCADE ON DELETE NO ACTION;
113
114
115
116 ALTER TABLE ticket
117     ADD FOREIGN KEY (passenger_id) REFERENCES passenger(passenger_id)
118     ON UPDATE CASCADE ON DELETE NO ACTION;
119
120
121

```

```

122 ALTER TABLE flight_final_set
123     ADD FOREIGN KEY (flight_id) REFERENCES flight(flight_id)
124     ON UPDATE CASCADE ON DELETE NO ACTION,
125     ADD FOREIGN KEY (seat_id) REFERENCES seat(seat_id)
126     ON UPDATE CASCADE ON DELETE NO ACTION;
127
128 ALTER TABLE seat
129     ADD FOREIGN KEY (class_of_service_id) REFERENCES
130     ↪ class_of_service(class_of_service_id)
131     ON UPDATE CASCADE ON DELETE NO ACTION,
132     ADD FOREIGN KEY (airplane_id) REFERENCES airplane(airplane_id)
133     ON UPDATE CASCADE ON DELETE NO ACTION;
134
135 ALTER TABLE class_of_service_airplane
136     ADD FOREIGN KEY (airplane_id) REFERENCES airplane(airplane_id)
137     ON UPDATE CASCADE ON DELETE NO ACTION,
138     ADD FOREIGN KEY (class_of_service_id) REFERENCES
139     ↪ class_of_service(class_of_service_id)
140     ON UPDATE CASCADE ON DELETE NO ACTION;
141
142 ALTER TABLE airplane_flight
143     ADD FOREIGN KEY (airplane_id) REFERENCES airplane(airplane_id)
144     ON UPDATE CASCADE ON DELETE NO ACTION,
145     ADD FOREIGN KEY (flight_id) REFERENCES flight(flight_id)
146     ON UPDATE CASCADE ON DELETE NO ACTION;
147
148
149
150 ALTER TABLE flight
151     ADD FOREIGN KEY (airport_flight_id) REFERENCES airport(airport_id)
152     ON UPDATE CASCADE ON DELETE NO ACTION,
153     ADD FOREIGN KEY (airport_of_landing_id) REFERENCES
154     ↪ airport(airport_id)
155     ON UPDATE CASCADE ON DELETE NO ACTION;
156
157 CREATE INDEX passenger_last_name_idx ON passenger (last_name);
158 CREATE INDEX passenger_passport_number_idx ON passenger
159     ↪ (passport_number);
160 CREATE INDEX passenger_passport_series_idx ON passenger
161     ↪ (passport_series);
162 CREATE INDEX passenger_passenger_id_idx ON passenger (passenger_id);

```

```

162
163 CREATE INDEX route_segment_ticket_id_idx ON route_segment (ticket_id);
164 CREATE INDEX route_segment_airport_flight_id_idx ON route_segment
    ↪ (airport_flight_id);
165 CREATE INDEX route_segment_airport_of_landing_id_idx ON route_segment
    ↪ (airport_of_landing_id);
166 CREATE INDEX route_segment_class_of_service_id_idx ON route_segment
    ↪ (class_of_service_id);
167 CREATE INDEX route_segment_flight_final_set_id_idx ON route_segment
    ↪ (flight_final_set_id);
168
169
170 CREATE INDEX flight_final_set_flight_id_idx ON flight_final_set
    ↪ (flight_id);
171 CREATE INDEX flight_final_set_seat_id_idx ON flight_final_set
    ↪ (seat_id);
172
173 CREATE INDEX flight_airport_flight_id_idx ON flight
    ↪ (airport_flight_id);
174 CREATE INDEX flight_airport_of_landing_id_idx ON flight
    ↪ (airport_of_landing_id);
175
176 CREATE INDEX airport_name_airport_idx ON airport (name_airport);
177
178 CREATE INDEX class_of_service_name_class_idx ON class_of_service
    ↪ (name_class);
179
180
181 CREATE INDEX airplane_flight_airplane_id_idx ON airplane_flight
    ↪ (airplane_id);
182 CREATE INDEX airplane_flight_flight_id_idx ON airplane_flight
    ↪ (flight_id);
183
184
185
186 CREATE INDEX class_of_service_airplane_airplane_id_idx ON
    ↪ class_of_service_airplane (airplane_id);
187 CREATE INDEX class_of_service_airplane_class_of_service_id_idx ON
    ↪ class_of_service_airplane (class_of_service_id);
188
189
190 CREATE INDEX seat_class_of_service_id_idx ON seat
    ↪ (class_of_service_id);
191 CREATE INDEX seat_airplane_id ON seat (airplane_id);

```

5 Приложение №2. Программа заполнения базы данных

```
1  import psycpg2
2  import random
3  import numpy as np
4  from datetime import datetime, timedelta
5
6  from numpy import number
7  from psycpg2 import Error
8  from psycpg2.extensions import ISOLATION_LEVEL_AUTOCOMMIT
9
10 AMOUNT_AIRPLANES=200
11 AMOUNT_PASSENGERES=500
12 AMOUNT_AIRPORTS=50
13 AMOUNT_FLIGHTS=0 # заполняется в InsertFlight
14 AMOUNT_TICKETS=0 # заполняется в InsertFlight
15 amount_of_baggage=[0,10,20,30]
16
17
18 def equal_distribution(general, scale):
19     # Используем нормальное распределение для генерации случайных чисел
20     amount = np.random.choice(range(1,general))
21     return int(amount)
22
23
24 def normal_distribution_for_airplane():
25     # Используем нормальное распределение для генерации случайных чисел
26     amount = np.random.normal(loc=15, scale=10)
27     amount = max(5, min(25, int(round(amount))))
28     return amount
29
30 def generate_aircraft_registration():
31     # Префиксы для различных стран
32     country_prefixes = ['RA', 'TC', 'UR', 'N', 'G', 'F', 'C', 'B', 'A']
33     # Буквы и цифры, которые могут использоваться в номере
34     characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
35
36     # Выбор случайного префикса для страны
37     prefix = random.choice(country_prefixes)
38
39     # Генерация случайного номера самолета (5-7 символов)
40     aircraft_number = ''.join(random.choices(characters,
```

```

41     # Сборка итогового номера
42     registration_number = prefix + aircraft_number
43
44     return registration_number
45
46
47 last_names = ["Иванов", "Garcia", "Смирнов", "Lee", "Попов",
48 ↪ "Thompson", "Кузнецов", "Hall",
49 ↪ "Соколов", "Brown", "Петров", "Мартинес", "Волков",
50 ↪ "Lewis", "Новиков", "Тарасов",
51 ↪ "Миллер", "Семёнов", "Комаров", "Гарсия", "Виноградов",
52 ↪ "Родригес", "Богданов",
53 ↪ "Макаров", "Климов", "Taylor", "Орлов", "Шмидт",
54 ↪ "Беляев", "Джонс"]
55
56 # Самые популярные имена
57 first_names = [
58     'Александр', 'Сергей', 'Андрей', 'Дмитрий', 'Анна', 'Елена',
59     ↪ 'Ольга', 'Наталья',
60     'Михаил', 'Иван', 'Евгений', 'Татьяна', 'Мария', 'Ирина',
61     ↪ 'Владимир', 'Олег',
62     'Павел', 'Максим', 'Екатерина', 'Алексей', 'Николай', 'Виктор',
63     ↪ 'Юлия', 'Галина',
64     'Валентин', 'Валерий', 'Александра', 'Евгения', 'Григорий',
65     ↪ 'Лариса', 'Августина'
66 ]
67
68 # Самые популярные отчества
69 middle_names = [
70     'Александрович', 'Сергеевич', 'Андреевич', 'Дмитриевич',
71     ↪ 'Алексеевич', 'Михайлович',
72     'Иванович', 'Евгеньевич', 'Павлович', 'Максимович', 'Николаевич',
73     ↪ 'Владимирович',
74     'Георгиевич', 'Денисович', 'Анатолевич', 'Олегович',
75     ↪ 'Станиславович', 'Романович',
76     'Валерьевич', 'Викторович', 'Григорьевич', 'Семенович',
77     ↪ 'Николаевна', 'Владимировна',
78     'Александровна', 'Андреевна', 'Михайловна', 'Сергеевна',
79     ↪ 'Евгеньевна', 'Олеговна'
80 ]
81
82 passport_series = ['{:04d}'.format(random.randint(0, 9999)) for _ in
83 ↪ range(30)]
84
85
86 passport_numbers = ['{:07d}'.format(random.randint(0, 9999999)) for _
87 ↪ in range(30)]

```

```

71
72 airplane_models = ['Boeing 747' , 'Airbus A380', 'Airbus A350', 'Airbus
    ↳ A220', 'Embraer E175', 'Airbus A320neo']
73
74 general_wieght_baggage = ['500', '1000', '2000', '5000', '10000', '20000']
75
76
77 date_of_birth = ['{:02d}-{:02d}-{:04d}'].format(
78     random.randint(1, 28), # День
79     random.randint(1, 12), # Месяц
80     random.randint(1900, 2023) # Год
81 ) for _ in range(30)]
82
83
84 airport_country_pairs = [
85     ("Шереметьево", "Россия"), ("Домодедово", "Россия"), ("Внуково",
    ↳ "Россия"), ("Пулково", "Россия"),
86     ("Толмачёво", "Россия"), ("Кольцово", "Россия"), ("Жуковский",
    ↳ "Россия"), ("Курумоч", "Россия"),
87     ("Борисполь", "Украина"), ("Жуляны", "Украина"), ("Торонто",
    ↳ "Канада"), ("Куала-Лумпур", "Малайзия"),
88     ("Дубай", "ОАЭ"), ("Хитроу", "Великобритания"), ("Франкфурт",
    ↳ "Германия"), ("Нарита", "Япония"),
89     ("Бен-Гурион", "Израиль"), ("Шанхай Пудонг", "Китай"), ("Инчхон",
    ↳ "Южная Корея"),
90     ("Сингапур Чанги", "Сингапур"), ("Лос-Анджелес", "США"),
    ↳ ("Ататюрк", "Турция"), ("Делли", "Индия"),
91     ("Сидней", "Австралия"), ("Орlando", "США"), ("Гонконг",
    ↳ "Гонконг"), ("Фукуока", "Япония"),
92     ("Даллас/Форт-Уэрт", "США"), ("Амстердам Схипхол", "Нидерланды"),
    ↳ ("Бангкок Суварнабхуми", "Таиланд"),
93     ("Мюнхен", "Германия"), ("Грибочки", "Россия"), ("Ягодки", "Папуа
    ↳ Новая Гвинея"), ("Сеул Инчхон", "Южная Корея"), ("Макао",
    ↳ "Макао"), ("Париж Шарль-де-Голль", "Франция"),
94     ("Феникс Скай-Харбор", "США"), ("Джон Ф. Кеннеди", "США"),
    ↳ ("Хонолулу", "США"), ("Минск 2", "Беларусь"),
95     ("Лиссабон Портела", "Португалия"), ("Хьюстон Джорджа Буша",
    ↳ "США"), ("Барселона Эль Прат", "Испания"),
96     ("Стамбул", "Турция"), ("Каир", "Египет"), ("Анталья", "Турция"),
    ↳ ("Берлин Шёнеберг", "Германия"),
97     ("Женева", "Швейцария"), ("Венеция Марко Поло", "Италия"),
    ↳ ("Амстердам Схипхол", "Нидерланды")
98 ]
99

```

```

100 class_of_service =
    ↪ ['Эконом', 'Премиум-эконом', 'Комфорт', 'Бизнес', 'Первый']
101
102
103 def generate_datetime_pair():
104     # Генерируем случайную дату и время для первого момента
105     start_date = datetime(1904, 1, 1)
106     end_date = datetime(2123, 12, 31)
107     delta = end_date - start_date
108     random_date1 = start_date + timedelta(days=random.randint(0,
    ↪ delta.days),
109
110
111                                     hours=random.randint(0, 23),
112                                     minutes=random.randint(0,
    ↪ 59),
113                                     seconds=random.randint(0,
    ↪ 59))
114
115     # Генерируем случайное время от 5 минут до 24 часов
116     delta_time = timedelta(minutes=random.randint(5, 24 * 60),
    ↪ seconds=random.randint(0, 59))
117
118     # Вычисляем второй момент времени
119     random_date2 = random_date1 + delta_time
120
121     # Преобразуем дату и время в строки с нужным форматом
122     date_time_format = "%Y-%m-%d %H:%M:%S"
123     date_time_str1 = random_date1.strftime(date_time_format)
124     date_time_str2 = random_date2.strftime(date_time_format)
125
126     return (date_time_str1, date_time_str2)
127
128
129
130 def Connect():
131     # Подключение к существующей базе данных
132     connection = psycopg2.connect(user="postgres",
133
134                                     # пароль, который указали при
    ↪ установке PostgreSQL
135                                     password="7052018",
136                                     host="127.0.0.1",
137                                     port="5432",
138                                     database="airtransport") #
    ↪ подключились к бд

```

```

138 cursor = connection.cursor() # возвращает объект подключения
139 print("Информация о сервере PostgreSQL")
140 print(connection.get_dsn_parameters(), "\n")
141 # Выполнение SQL-запроса
142 cursor.execute("SELECT version();")
143 # Получить результат
144 record = cursor.fetchone()
145 print("Вы подключены к - ", record, "\n")
146
147 cursor.execute("TRUNCATE TABLE passenger CASCADE; "
148               "TRUNCATE TABLE ticket CASCADE; "
149               "TRUNCATE TABLE route_segment CASCADE;"
150               "TRUNCATE TABLE class_of_service CASCADE;"
151               "TRUNCATE TABLE flight_final_set CASCADE;"
152               "TRUNCATE TABLE seat CASCADE;"
153               "TRUNCATE TABLE class_of_service_airplane CASCADE;"
154               "TRUNCATE TABLE airplane CASCADE;"
155               "TRUNCATE TABLE airplane_flight CASCADE;"
156               "TRUNCATE TABLE flight CASCADE ;"
157               "TRUNCATE TABLE airport CASCADE;"
158               "TRUNCATE TABLE passenger RESTART IDENTITY CASCADE;"
159               "TRUNCATE TABLE ticket RESTART IDENTITY CASCADE;"
160               "TRUNCATE TABLE route_segment RESTART IDENTITY
161               ↪ CASCADE;"
162               "TRUNCATE TABLE class_of_service RESTART IDENTITY
163               ↪ CASCADE;"
164               "TRUNCATE TABLE flight_final_set RESTART IDENTITY
165               ↪ CASCADE;"
166               "TRUNCATE TABLE seat RESTART IDENTITY CASCADE;"
167               "TRUNCATE TABLE class_of_service_airplane RESTART
168               ↪ IDENTITY CASCADE;"
169               "TRUNCATE TABLE airplane RESTART IDENTITY CASCADE;"
170               "TRUNCATE TABLE airplane_flight RESTART IDENTITY
171               ↪ CASCADE;"
172               "TRUNCATE TABLE flight RESTART IDENTITY CASCADE;"
173               "TRUNCATE TABLE airport RESTART IDENTITY CASCADE;"
174               )
175 connection.commit()
176
177 for i in range(AMOUNT_PASSENGERES): # заполняем пассажиров
178     cursor.execute("INSERT INTO passenger "
179                   "(last_name, first_name, middle_name,
180                    ↪ passport_series, passport_number,
181                    ↪ date_of_birth) ")

```



```

176         "VALUES (%s, %s, %s, %s, %s, %s)",
177         (
178             last_names[random.randint(0, len(last_names)
179                 ↪ - 1)], # Фамилия
180             first_names[random.randint(0,
181                 ↪ len(first_names) - 1)], # Имя
182             middle_names[random.randint(0,
183                 ↪ len(middle_names) - 1)], # Отчество
184             passport_series[random.randint(0,
185                 ↪ len(passport_series) - 1)], # Серия
186                 ↪ паспорта
187             passport_numbers[random.randint(0,
188                 ↪ len(passport_numbers) - 1)], # Номер
189                 ↪ паспорта
190             date_of_birth[random.randint(0,
191                 ↪ len(date_of_birth) - 1)] # Дата
192                 ↪ рождения
193         )
194     )
195
196     for i in range(AMOUNT_AIRPLANES): # самолеты
197         cursor.execute("INSERT INTO airplane "
198             " (model, registration_number, number_of_baggage)
199             ↪ "
200             "VALUES (%s, %s, %s)",
201             (
202                 airplane_models[random.randint(0,
203                     ↪ len(airplane_models) - 1)], # модель
204                 generate_aircraft_registration(), # номер
205                 general_wieght_baggage[random.randint(0,
206                     ↪ len(general_wieght_baggage) - 1)], #
207                     ↪ багаж
208             )
209         )
210     connection.commit()
211
212     for airport, country in airport_country_pairs: # аэропорты
213         cursor.execute("INSERT INTO airport (name_airport, country)
214             ↪ VALUES (%s, %s)", (airport, country))
215
216     for class_ in class_of_service: # классы обслуживания
217         cursor.execute("INSERT INTO class_of_service (name_class)
218             ↪ VALUES (%s)", (class_,))
219
220

```

```

206
207     #2
208
209
210
211     #булеты
212     for i in range(AMOUNT_PASSENGERES):
213         for _ in range(equal_distribution(50,10)):
214             global AMOUNT_TICKETS
215             AMOUNT_TICKETS+=1
216             cursor.execute(
217                 "INSERT INTO ticket (passenger_id) "
218                 "VALUES (%s)",
219                 (i+1,)
220             )
221
222     #места и классы
223     def insertIntoSeat(n,i,j):
224
225         cursor.execute(
226             "INSERT INTO seat
227             ↪ (class_of_service_id,airplane_id,number_of_seat) "
228             "VALUES (%s,%s,%s)",
229             (random_indices_classe_sorted[n]+1, i+1, seat_names[j])
230         )
231
232     def insertIntoClasses_Airplanes(air, cls):
233         cursor.execute(
234             "INSERT INTO class_of_service_airplane
235             ↪ (airplane_id,class_of_service_id) "
236             "VALUES (%s,%s)",
237             (air+1,cls+1)
238         )
239
240     amount_seats_list=[20,AMOUNT_AIRPORTS,150,400]
241     amount_classes_list=[1,2,3]
242     seat_names=[str(i)+j for i in range(1, 100) for j in
243         ↪ ('a','b','c','d','e','f')]
244     for i in range(AMOUNT_AIRPLANES):
245         amount_classes=amount_classes_list[random.randint
246             (0,len(amount_classes_list)-1)]
247         amount_seats = amount_seats_list[random.randint
248             (0, len(amount_seats_list) - 1)]
249         random_indices_classe_sorted =
250             ↪ random.sample(range(len(class_of_service)), amount_classes)
251         random_indices_classe_sorted.sort()

```

```

247     names_classes_in_airplane = [class_of_service[i] for i in
    ↪ random_indices_classe_sorted]
248     if amount_classes==1:
249         for j in range(0,amount_seats):
250             insertIntoSeat(0, i, j)
251
    ↪ insertIntoClasses_Airplanes(i,random_indices_classe_sorted[0])
252     if amount_classes == 2:
253         for j in range(0,int(0.8*amount_seats)):
254             insertIntoSeat(0, i, j)
255         for j in range(int(0.8*amount_seats),amount_seats):
256             insertIntoSeat(1, i, j)
257         insertIntoClasses_Airplanes(i,
    ↪ random_indices_classe_sorted[0])
258         insertIntoClasses_Airplanes(i,
    ↪ random_indices_classe_sorted[1])
259     if amount_classes == 3:
260         for j in range(0, int(0.75 * amount_seats)):
261             insertIntoSeat(0, i, j)
262         for j in range(int(0.75 * amount_seats), int(0.9 *
    ↪ amount_seats)):
263             insertIntoSeat(1, i, j)
264         for j in range(int(0.9 * amount_seats) , amount_seats):
265             insertIntoSeat(2, i, j)
266         insertIntoClasses_Airplanes(i,
    ↪ random_indices_classe_sorted[0])
267         insertIntoClasses_Airplanes(i,
    ↪ random_indices_classe_sorted[1])
268         insertIntoClasses_Airplanes(i,
    ↪ random_indices_classe_sorted[2])
269
270
271
272
273     #3
274     # Самолет_рейс                u рейс
275
276     # Рейсы
277     def insertFlight():
278         global AMOUNT_FLIGHTS
279         AMOUNT_FLIGHTS += 1
280         i1 = random.randint(1, AMOUNT_AIRPORTS)
281         i_ = random.randint(1, AMOUNT_AIRPORTS)
282         i2 = i1 + 1 if i1 == i_ else i_
283         if i2 == 51:

```

```

284         i2 = i1 - 1;
285         date1, date2 = generate_datetime_pair()
286         cursor.execute(
287             "INSERT INTO flight (airport_flight_id,
288                 ↪ airport_of_landing_id, date_and_time_flight,
289                 ↪ date_and_time_landing) "
290             "VALUES (%s, %s, %s, %s)",
291             (i1, i2, date1, date2)
292         )
293
294     current_flight=0
295     for i in range(AMOUNT_AIRPLANES):
296         amount_flight=normal_distribution_for_airplane()
297
298         for _ in range(amount_flight):
299             insertFlight()
300             current_flight+=1
301             cursor.execute(
302                 "INSERT INTO airplane_flight (airplane_id,flight_id) "
303                 "VALUES (%s,%s)",
304                 (i + 1, current_flight)
305             )
306
307
308     counter_flight_final_set=0
309     # рейс финальный набор и сегмент маршрута:
310     for i in range(AMOUNT_TICKETS):
311         amount_segment_for_ticket= random.randint(1,6)
312         for _ in range(amount_segment_for_ticket):
313             random_flight=0
314             while 1==1:
315                 random_flight = random.randint(1, AMOUNT_FLIGHTS)
316                 cursor.execute("""
317
318                     SELECT s.seat_id,
319                         ↪ s.class_of_service_id
320                     FROM seat s
321                     JOIN airplane a ON a.airplane_id =
322                         ↪ s.airplane_id
323                     JOIN airplane_flight af ON
324                         ↪ af.airplane_id = a.airplane_id
325                     JOIN flight f ON f.flight_id =
326                         ↪ af.flight_id
327                     WHERE f.flight_id = {}

```

```

323
324         """.format(random_flight))
           ↳ # получили места и классы
           ↳ обслуживания рейса № random_flight
325 list_of_pair_seat_class = cursor.fetchall()
326 random_pair_seat_class =
           ↳ random.choice(list_of_pair_seat_class)
           ↳ # получили случайное место на рейсе
327
328 cursor.execute("""SELECT * FROM flight_final_set WHERE
           ↳ seat_id={} AND flight_id={}
329
330
331         """.format(random_pair_seat_class[0]
332
333 ,random_flight))
334
335     if not cursor.fetchall():
336         break
337
338 cursor.execute("""
339         SELECT
           ↳ airport_flight_id,airport_of_landing_id
           ↳ FROM flight
           ↳ WHERE flight.flight_id={}
340         """.format(random_flight))
341 random_pair_flight_landing = cursor.fetchall()
342
343
344 random_amount_of_baggage=random.choice(amount_of_baggage)
345 cursor.execute("INSERT INTO flight_final_set
           ↳ (flight_id,seat_id,number_of_baggage) "
346 "VALUES (%s,%s,%s)",
347 (random_flight,random_pair_seat_class[0],
           ↳ random_amount_of_baggage))
348 counter_flight_final_set+=1
349 cursor.execute("INSERT INTO
           ↳ route_segment(ticket_id,airport_flight_id,airport_of_landing_id,"
350 "class_of_service_id,flight_final_set_id
           ↳ ,number_of_baggage) "
351 "VALUES (%s,%s,%s,%s,%s,%s)",
352 (i+1, random_pair_flight_landing[0][0],
           ↳ random_pair_flight_landing[0][1],
353 random_pair_seat_class[1],
354
           ↳ counter_flight_final_set,random_amount_of_baggage))

```

```

355
356     print("index of ticket: ", i, "max index ", AMOUNT_TICKETS,
    ↪     "index of flight: ", counter_flight_final_set)
357     # Вывод результатов
358     # for row in list_of_pair_seat_class:
359     #     print("Seat ID:", row[0], "Class of Service ID:",
    ↪     row[1])
360
361     connection.commit()
362 def main():
363
364
365     Connect()
366     list =[i for i in range(0,3)]
367     print (list)
368     list = [i for i in range(4, 6)]
369     print(list)
370 main()

```