Санкт-Петербургское государственное бюджетное профессиональное образовательное учреждение

«Академия управления городской средой, градостроительства и печати»

УТВЕРЖДАЮ
Заместитель директора
по учебно-методической работе
О.В.Фомичева
«26» декабря 2025 г.

Контрольно-оценочные средства для текущего контроля и промежуточной аттестации

по учебной дисциплине

ОП.07 «ОСНОВЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ»

специальности 09.02.13 Интеграция решений с применением технологий искусственного интеллекта

Форма обучения -очная

Санкт-Петербург
2025

Разработаны на основе федерального государственного образовательного стандарта по специальностям среднего профессионального образования 09.02.13 Интеграция решений с применением технологий искусственного интеллекта, утвержденного приказом Министерства просвещения Российской Федерации № 1025 от 24.12.2024г., зарегистрировано Министерством юстиции (рег. № 81046 от 25.01.2025г.)

Разработчик: Ипатова С.В./Оболенская Е.Г., методисты СПб ГБПОУ АУГСГиП

Одобрены на заседании цикловой комиссии
Общетехнических дисциплин и компьютерных технологий
Протокол № 4
От 09.12.2025 г.
Председатель цикловой комиссии:
Шурухина И.Е.

1.Результаты освоения учебной дисциплины

В рамках программы учебной дисциплины обучающимися осваиваются умения и знания

формируемые ПК, ОК, ЛР	Умения	Знания
ОК 01-02, ОК 04-06 ПК 2.2 ПК 2.5 ЛР13-17	– проектировать реляционную базу данных; – использовать язык запросов для программного извлечения сведений из баз данных;	– основы теории баз данных; – модели данных; – особенности реляционной модели и проектирование баз данных; – изобразительные средства, используемые в ER- моделировании; – основы реляционной алгебры; – принципы проектирования баз данных; – обеспечение непротиворечивости и целостности данных; – средства проектирования структур баз данных; – язык запросов SQL.

ОК 01. Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02. Использовать современные средства поиска, анализа и интерпретации информации, и информационные технологии для выполнения задач профессиональной деятельности.

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях.

ОК 04. Эффективно взаимодействовать и работать в коллективе и команде.

ОК 05. Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста.

ОК 06. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных российских духовно-нравственных ценностей, в том числе с учетом гармонизации межнациональных и межрелигиозных отношений, применять стандарты антикоррупционного поведения.

ПК 2.2. Осуществлять процедуры администрирования баз данных.

ПК 2.5. Подготавливать данные для базы знаний

Критерии выставления оценок

- оценка «**отлично**» выставляется обучающемуся, если он выполнил 90-100% заданий без ошибок, умеет логически обосновать полученные результаты и сделать правильные выводы.

- оценка «**хорошо**» выставляется обучающемуся, если он выполнил 75-90% заданий без ошибок, допускает неточности в расчетах, владеет необходимыми навыками выполнения практических задач, но допуская недочеты в оформлении работы или формулировке выводов.

- оценка «**удовлетворительно**» выставляется обучающемуся, если он выполнил 50-75% заданий без ошибок, допускает неточности в расчетах, даёт недостаточно правильные формулировки, испытывает затруднения при решении задач и формулировке выводов.

- оценка «**неудовлетворительно**» выставляется обучающемуся, если он выполнил менее 50% задач, допускает существенные ошибки, не справляется с решением практических задач.

Критерии оценки при устном ответе

- оценка «**отлично**» выставляется студенту, если он правильно отвечает на все вопросы, владеет навыками и приемами решения задач, умеет логически обосновать полученные результаты и сделать правильные выводы.
- оценка «**хорошо**» выставляется студенту, если он твердо знает материал, грамотно и, по существу, излагает его, владеет необходимыми навыками решения задач, но допускает неточности в расчетах и в формулировке выводов.
- оценка «**удовлетворительно**» выставляется студенту, если он имеет знания основного материала, но отвечает не на все вопросы и испытывает затруднения в формулировке выводов.
- оценка «**неудовлетворительно**» выставляется студенту, если он не может ответить на поставленные теоретические вопросы, допускает существенные ошибки при расчетах, не справляется с решением задач.

КОМПЛЕКТ ОЦЕНОЧНЫХ СРЕДСТВ ДЛЯ ТЕКУЩЕГО КОНТРОЛЯ

1. Практическая контрольная работа № 1. Построение видов моделей данных

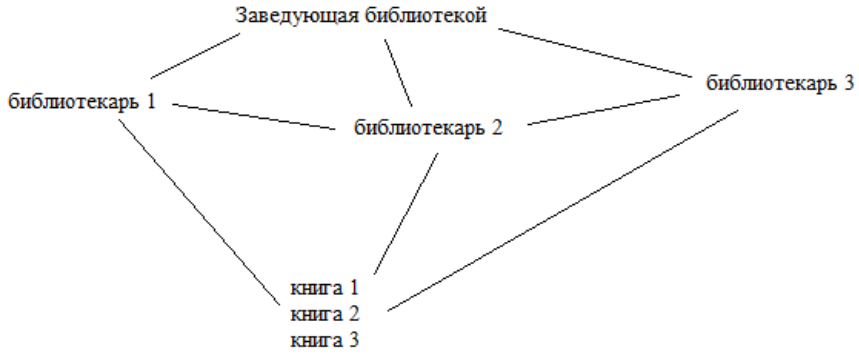
Задание: построить сетевую, иерархическую и реляционную модели для объекта по своему варианту.

Варианты заданий:

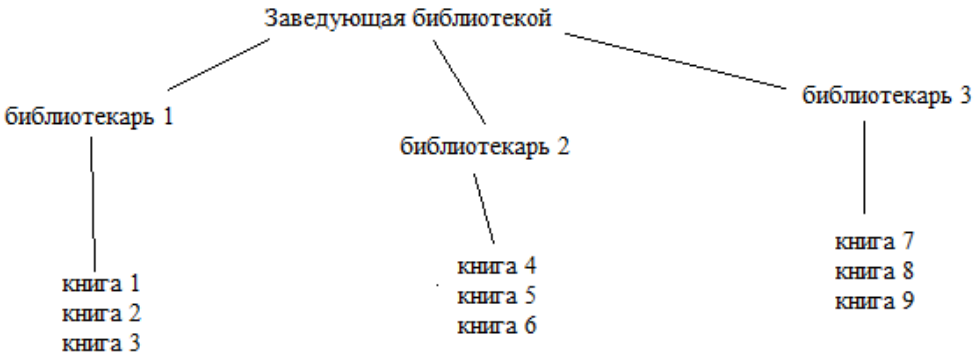
Вариант 1	Библиотека
Вариант 2	Авиаперевозки

Эталоны ответов

Вариант 1
Сетевая модель



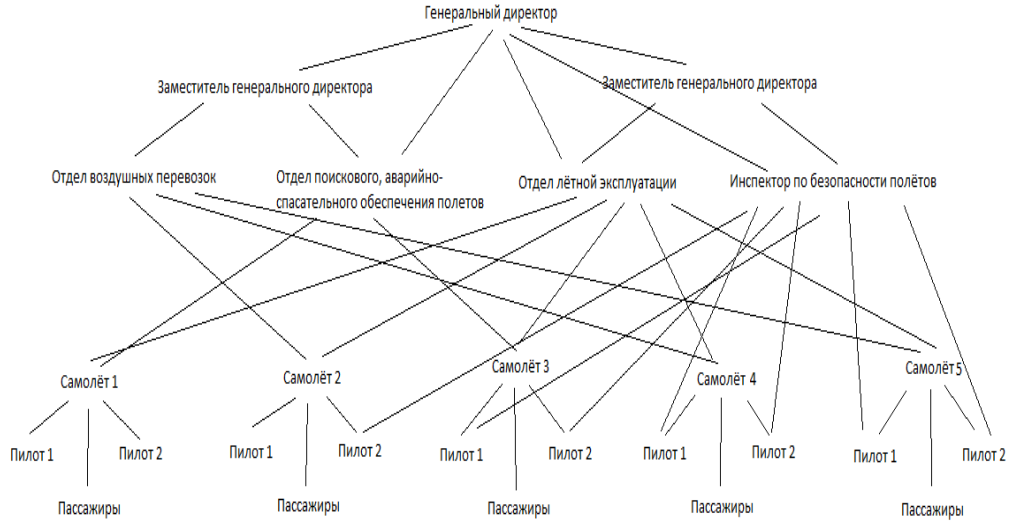
Иерархическая модель



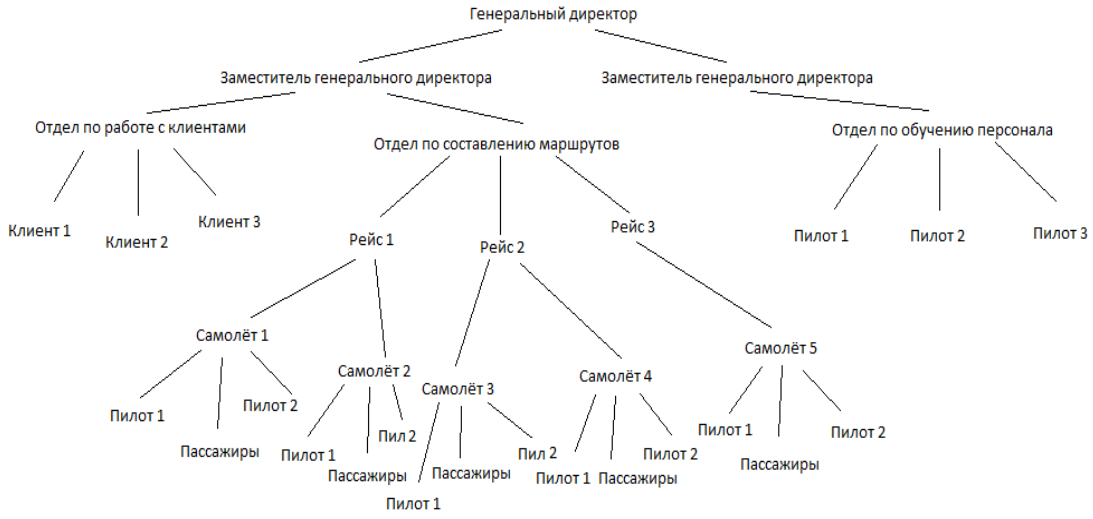
Реляционная модель

Вариант 2

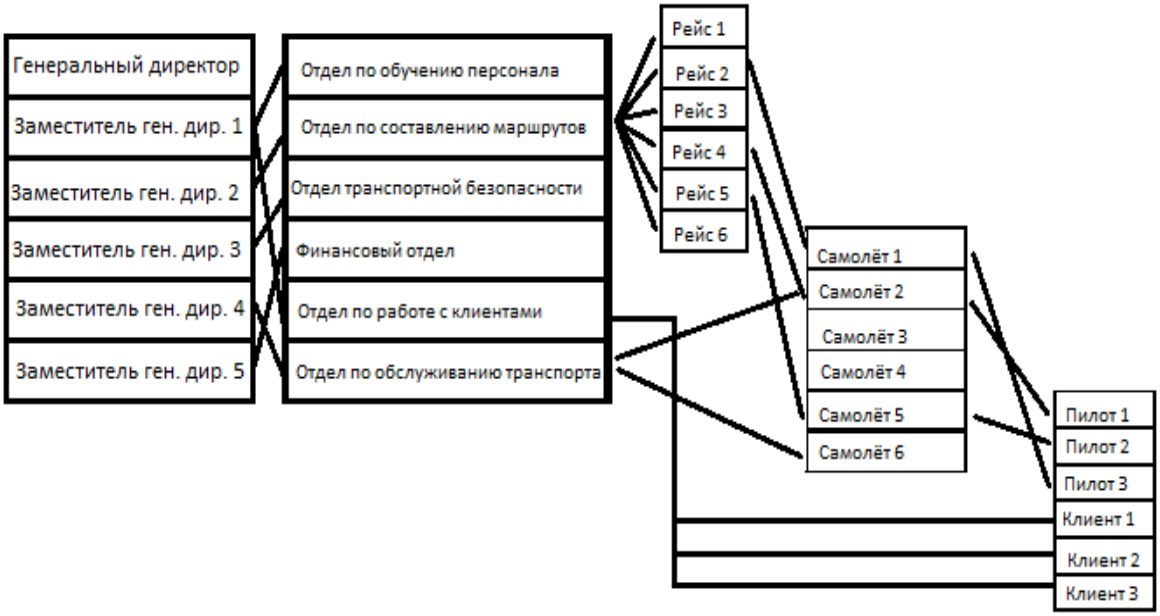
Сетевая модель



Иерархическая модель



Реляционная модель



2. Практическая контрольная работа № 2. Построение реляционной модели данных. Определение ключей и связей между объектами

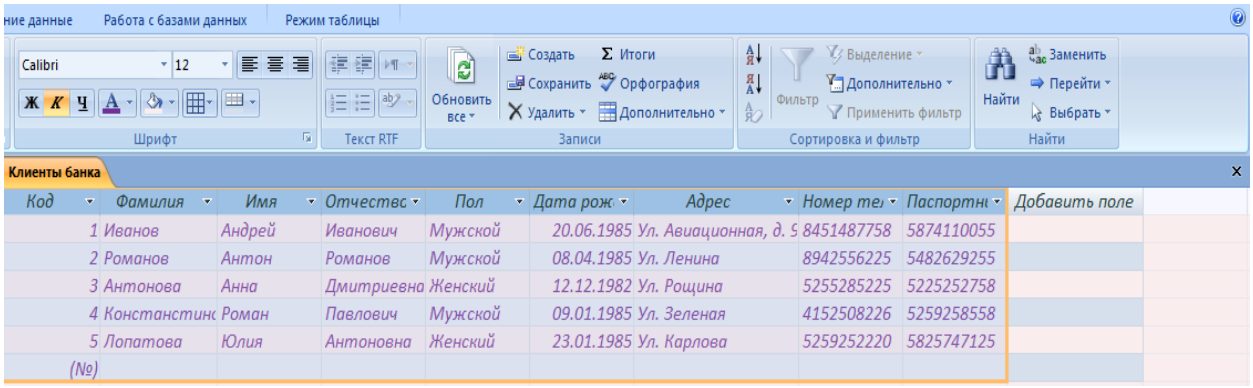
Задание:

1. Откройте программу Access и создайте новую базу данных по названию банка. Настройте сохранение базы данных в Вашу папку.
2. Требуется создать 1 таблицу под названием Клиенты банка.
3. В таблице должны быть следующие поля: Код клиента, Фамилия, Имя, Отчество, пол, дата рождения, адрес, номер телефона, паспортные данные. Подобрать правильный тип данных для каждого поля. Пол клиента должен выбираться из фиксированного набора значений (мужской, женский).
4. Заполните таблицу для 5 клиентов выбранного вами банка.
5. Для таблицы измените цвет фона, цвет текста, размер текста.
6. Создать вторую таблицу к вашей базе данных под названием Кредиты банка.
7. В таблице должны быть следующие поля: Код кредита, Название кредита Процентная ставка по кредиту, Срок кредита, Условия кредита. Подобрать правильный тип данных для каждого поля. Для Срока кредита сделать фиксированный набор значений.
8. Заполните таблицу для 5 кредитов вашего банка.
9. Для таблицы измените цвет фона, цвет текста, размер текста.
10. Создать третью таблицу к вашей базе данных под названием Вклады банка.
11. В таблице должны быть следующие поля: Код вклада, Название вклада Процентная ставка по вкладу, Срок вклада, Условия вклада. Подобрать правильный тип данных для каждого поля. Для Срока вклада сделать фиксированный набор значений.
12. Заполните таблицу для 5 вкладов по вашему банку.
13. Для таблицы измените цвет фона, цвет текста, размер текста.
14. Откройте таблицу Клиенты банка. Добавьте поля Вклад банка и Кредит банка, Сумма. Создайте с помощью мастера подстановок раскрывающиеся списки из таблиц Вклады банка и Кредиты банка.
15. Заполните три новых поля данными в таблице Клиенты банка.
16. В каждой таблице создайте ключевые поля (тип данных счётчик).

17. Создайте схему данных (отношения один ко многим).

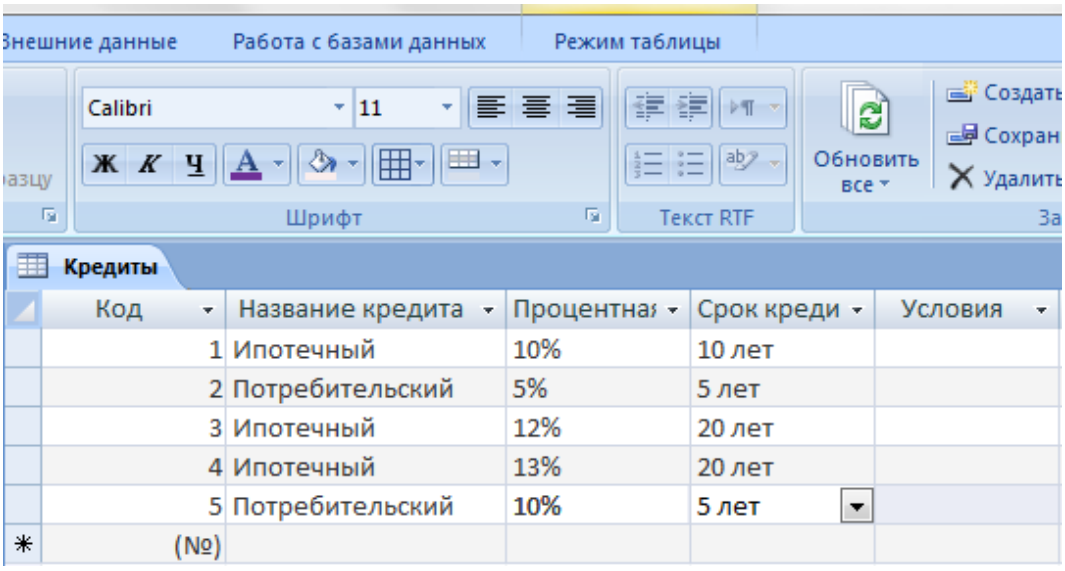
Эталон ответа

Таблица 1 – Клиенты банка



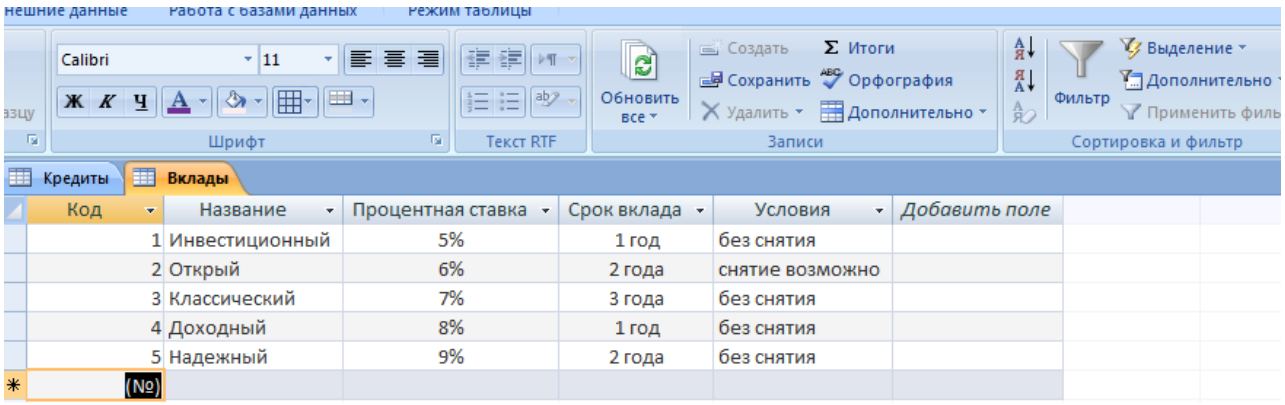
Код	Фамилия	Имя	Отчество	Пол	Дата рож	Адрес	Номер те	Паспортн	Добавить поле
1	Иванов	Андрей	Иванович	Мужской	20.06.1985	Ул. Авиационная, д. 5	8451487758	5874110055	
2	Романов	Антон	Романов	Мужской	08.04.1985	Ул. Ленина	8942556225	5482629255	
3	Антонова	Анна	Дмитриевна	Женский	12.12.1982	Ул. Рощина	5255285225	5225252758	
4	Константинс	Роман	Павлович	Мужской	09.01.1985	Ул. Зеленая	4152508226	5259258558	
5	Лопатова	Юлия	Антоновна	Женский	23.01.1985	Ул. Карлова	5259252220	5825747125	
(№)									

Таблица 2 – Кредиты банка



Код	Название кредита	Процентная	Срок креди	Условия
1	Ипотечный	10%	10 лет	
2	Потребительский	5%	5 лет	
3	Ипотечный	12%	20 лет	
4	Ипотечный	13%	20 лет	
5	Потребительский	10%	5 лет	
(№)				

Таблица 3 – Вклады банка



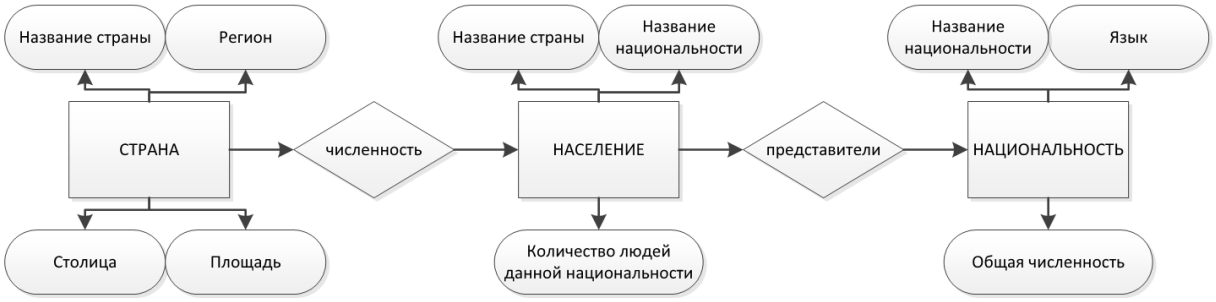
Код	Название	Процентная ставка	Срок вклада	Условия	Добавить поле
1	Инвестиционный	5%	1 год	без снятия	
2	Открый	6%	2 года	снятие возможно	
3	Классический	7%	3 года	без снятия	
4	Доходный	8%	1 год	без снятия	
5	Надежный	9%	2 года	без снятия	
(№)					

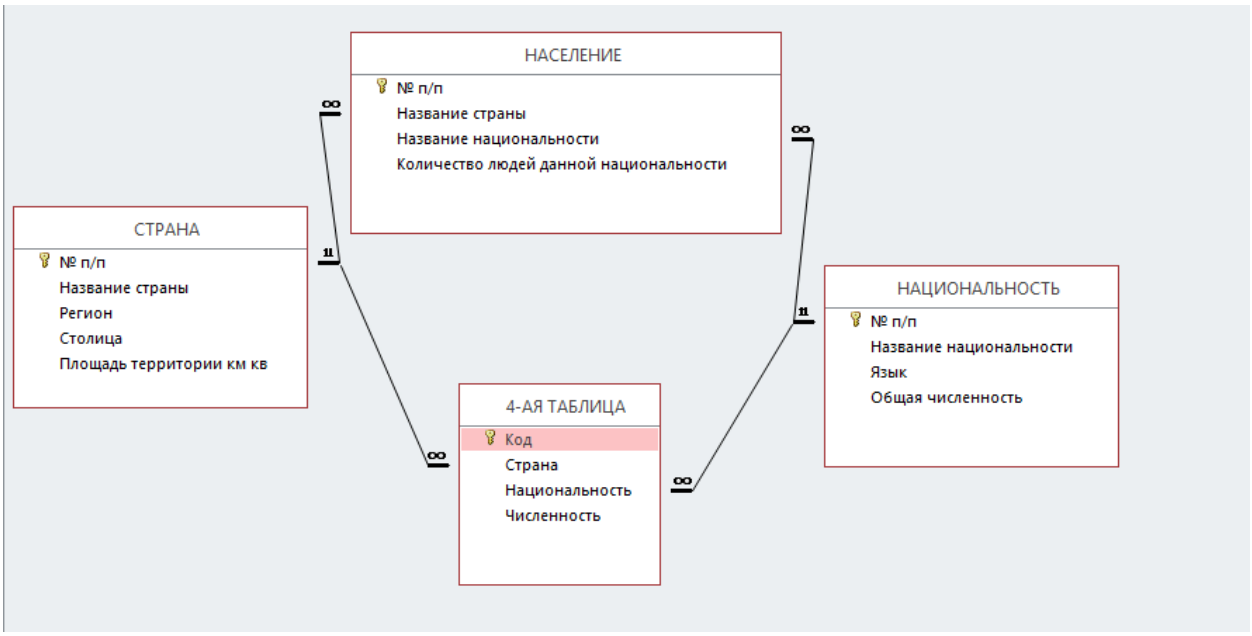
3. Практическая контрольная работа № 5. Приведение таблицы к нормальной форме. ER-диаграмма

Задание:

1. *Построить ER-диаграмму по СУБД, состоящей из трёх таблиц Страна, Население, Национальность.*
2. *Создать базу данных в MS Access.*
3. *Создать необходимые связи.*
4. *Добавить 4-ую таблицу к вашей базе данных.*
5. *Привести все таблицы к 3-ей нормальной форме.*
6. *Создать запросы по заданию к вашему варианту.*

Эталон ответа





4. Устный зачет по темам

Инструкция для обучающихся: Зачет сдается в рамках учебного занятия. Каждому студенту по выбору преподавателя дается два вопроса, на которые он отвечает в устной форме.
Выполнение задания: одному студенту на ответ выделяется 5 мин, группа сдает зачет за одно учебное занятие.

Вопросы к зачету:

1. Требования к системам управления базами данных;
2. Понятие кластера и понятие экземпляра базы данных
3. Функции и компоненты СУБД
4. Понятие модели данных и виды моделей
5. Понятие атрибута, домена, кортежа, ключа, связи, объекта
6. Понятие репликации базы данных
7. Понятие транзакции базы данных
8. Виды отношений в базе данных
9. Операции реляционной алгебры
10. Понятие модели «Сущность – связь»
11. Основные этапы создания ER-диаграммы
12. Понятие нормальной формы
13. Понятие первой, второй и третьей нормальной формы
14. Основные этапы проектирования базы данных

5. Практическая контрольная работа № 6. Установка системы управления базами данных PostgreSQL

Задание:

Установка на Windows

1. С официального сайта скачать PostgreSQL версии 11.5 для Windows 64x и установить на вашу виртуальную машину с Windows 10. (при необходимости можно использовать пояснения по ссылке: https://o7planning.org/ru/10713/installing-and-configuring-postgresql-database)

2. Запомнить/записать пароль и порт, по которому работает PostgreSQL.

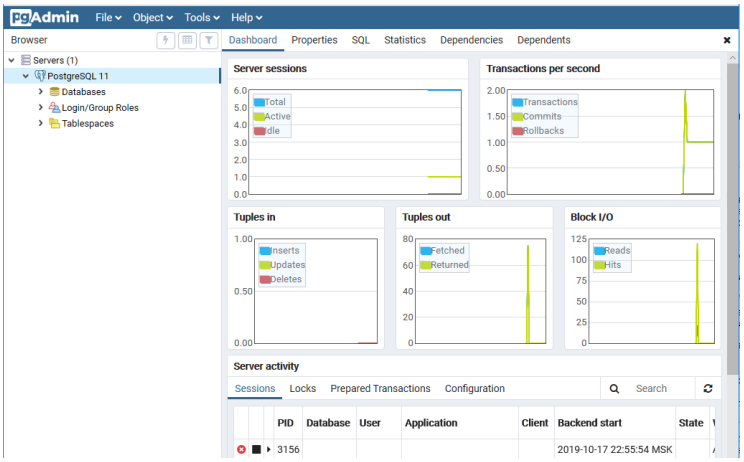
3. За время установки и по завершении установки для отчёта сделайте скриншот.

4. После завершения установки перезагрузите вашу виртуальную машину.

5. По ссылке https://edu.postgrespro.ru/demo_small. zip скачайте тестовую базу данных для развёртывания.

6. Запустите из папки с PostgreSQL pgAdmin4. Убедитесь, что он запускается и вы в него заходите. Сделайте для отчёта скриншот.

При правильной работе вы должны увидеть:



Установка на Linux

7. Import the repository key from https://www.postgresql.org/media/keys/ACCC4CF8.asc:

```
sudo apt-get install curl ca-certificates gnupg
curl https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
```

8. Create /etc/apt/sources.list.d/pgdg.list. The distributions are called codename-pgdg. In the example, replace buster with the actual distribution you are using:

```
deb http://apt.postgresql.org/pub/repos/apt/ buster-pgdg main
```

9. (You may determine the codename of your distribution by running lsb_release -c). For a shorthand version of the above, presuming you are using a supported release:

```
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt/ $(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'
```

10. Finally, update the package lists, and start installing packages:

```
sudo apt-get update
sudo apt-get install postgresql-11 pgadmin4
```

11. Alternately, this shell script will automate the repository setup. The script is included in the postgresql-common package in Debian and Ubuntu, so you can also run it straight from there:

```
sudo apt install postgresql-common
sudo sh /usr/share/postgresql-common/pgdg/apt.postgresql.org.sh
```

12. Сброс пароля:

sudo -u postgres psql postgres

\password postgres

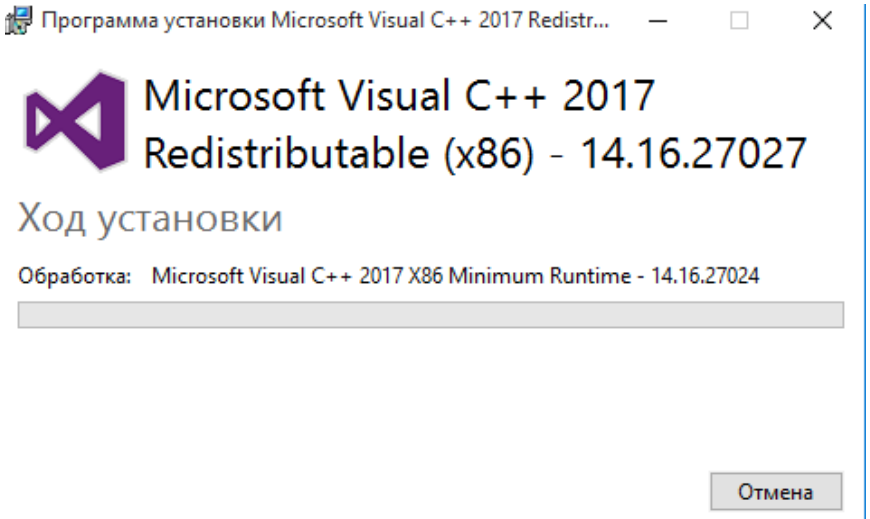
Enter new password:

13. Запустите из папки с PostgreSQL pgAdmin4 или терминала и убедитесь, что он запускается и вы в него заходите. Сделайте для отчёта скриншот. При правильной работе вы должны увидеть такое же окно, как при установке в Windows.

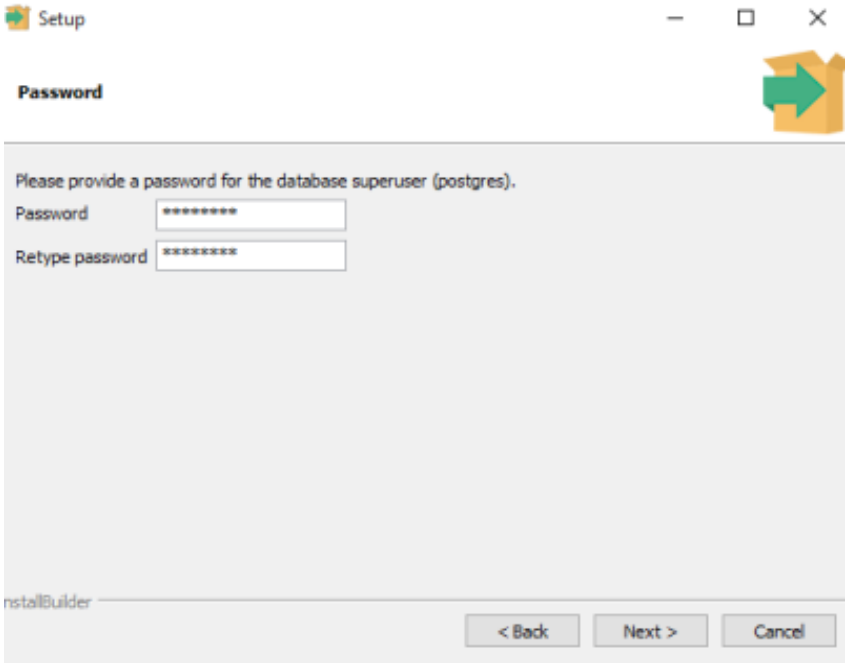
Эталон ответа:

PostgreSQL на Windows

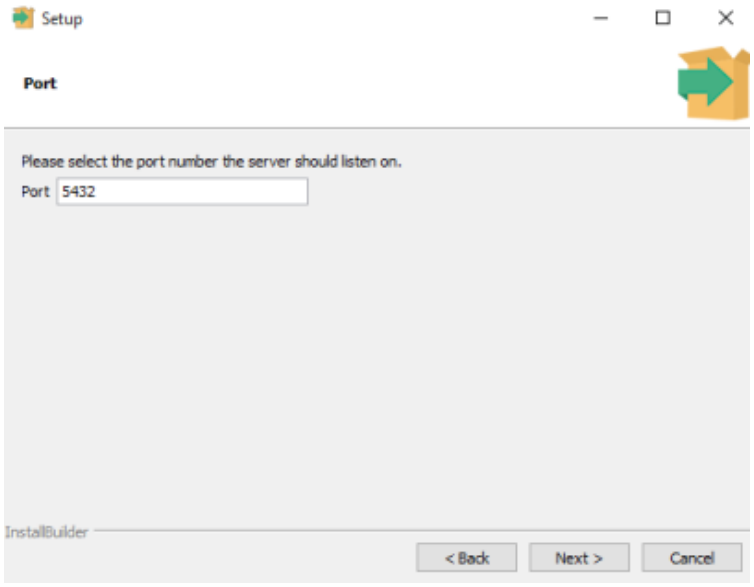
Старт установки PostgreSQL:



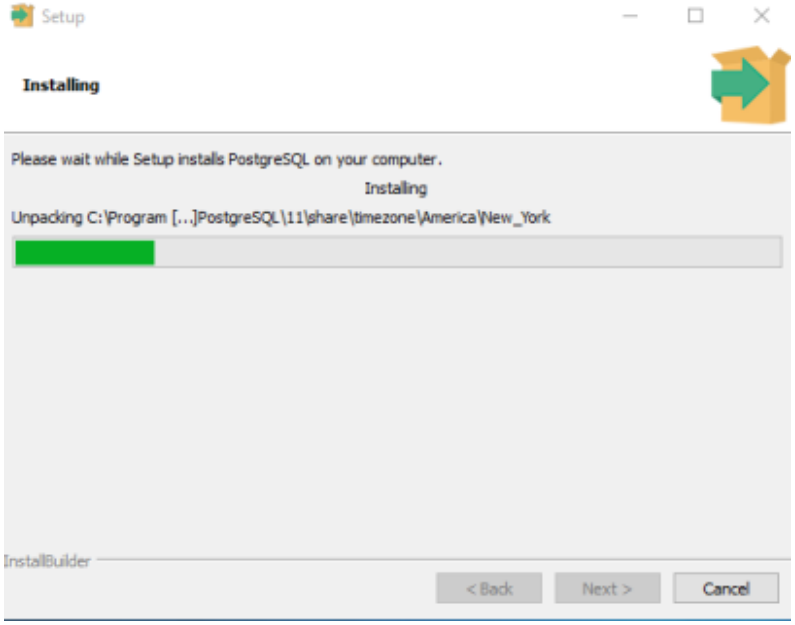
Был введен пароль: admin679



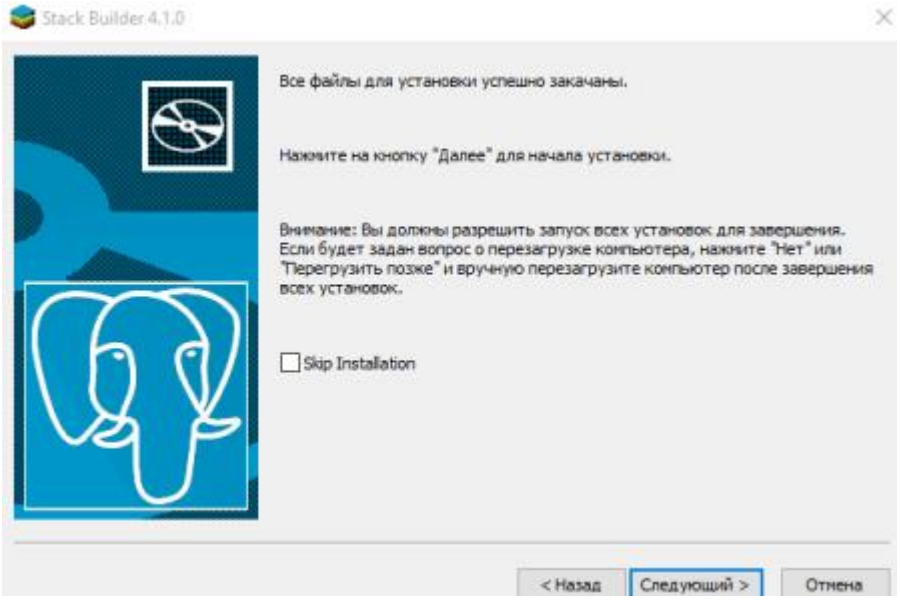
Port number:



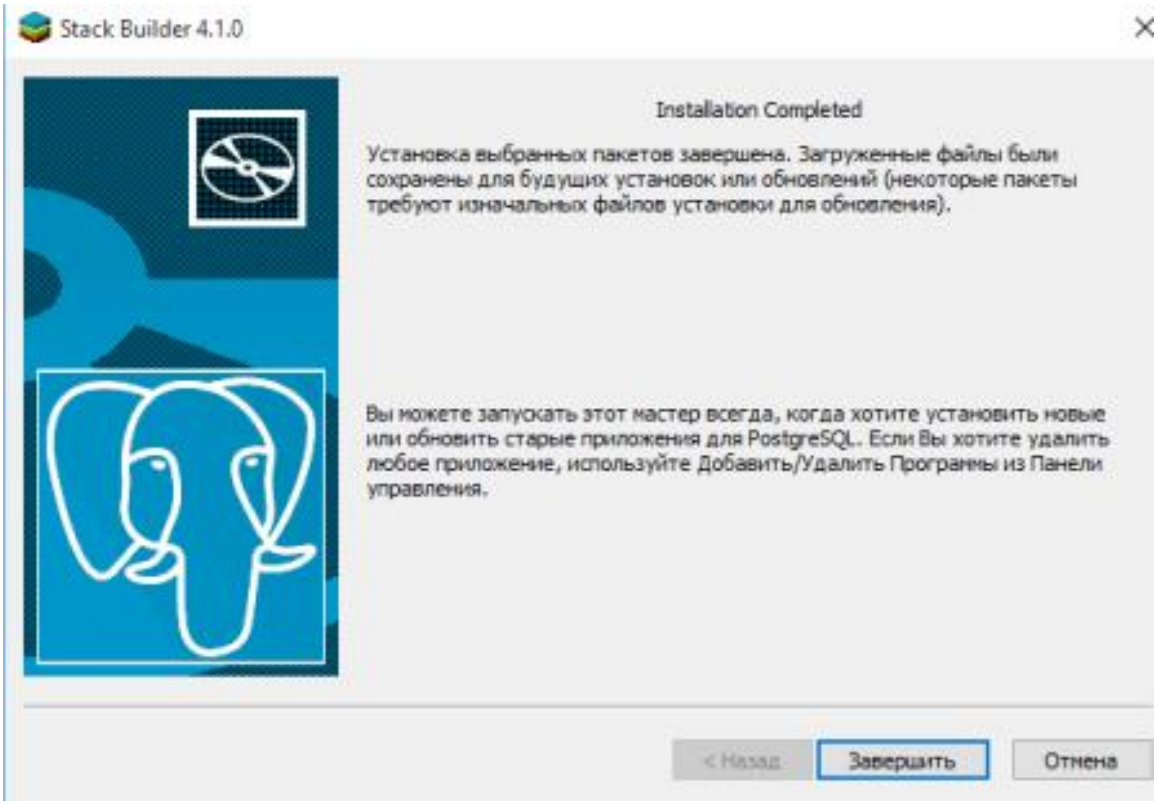
Установка 25%



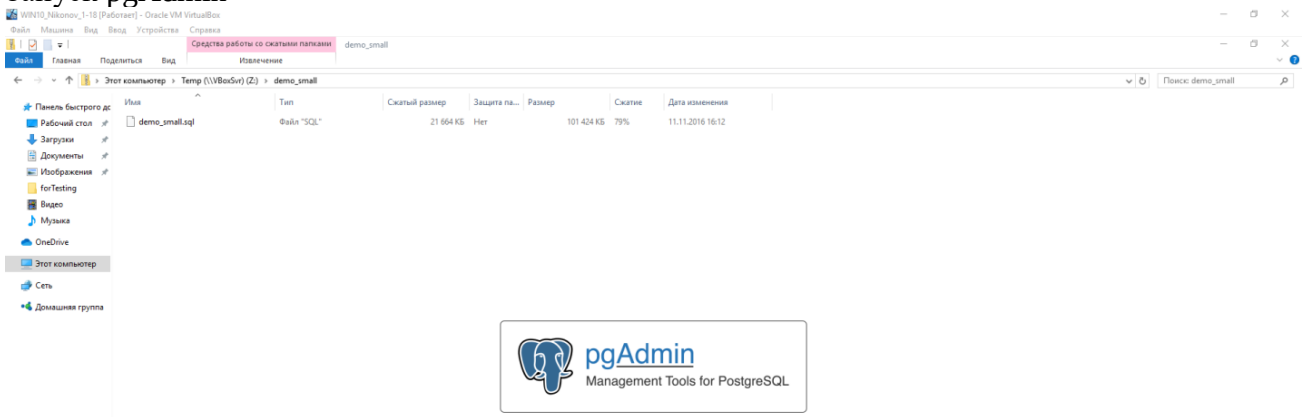
Установка Stack Builder 4.1.0



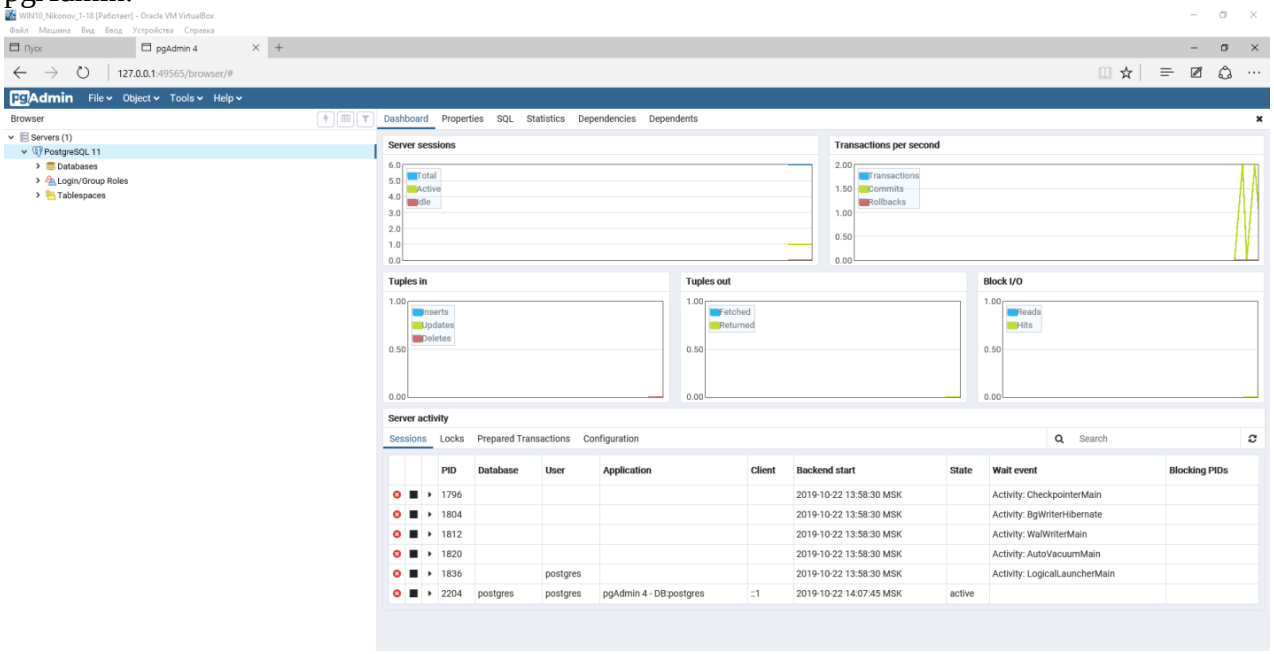
Установка Stack Builder завершена успешно



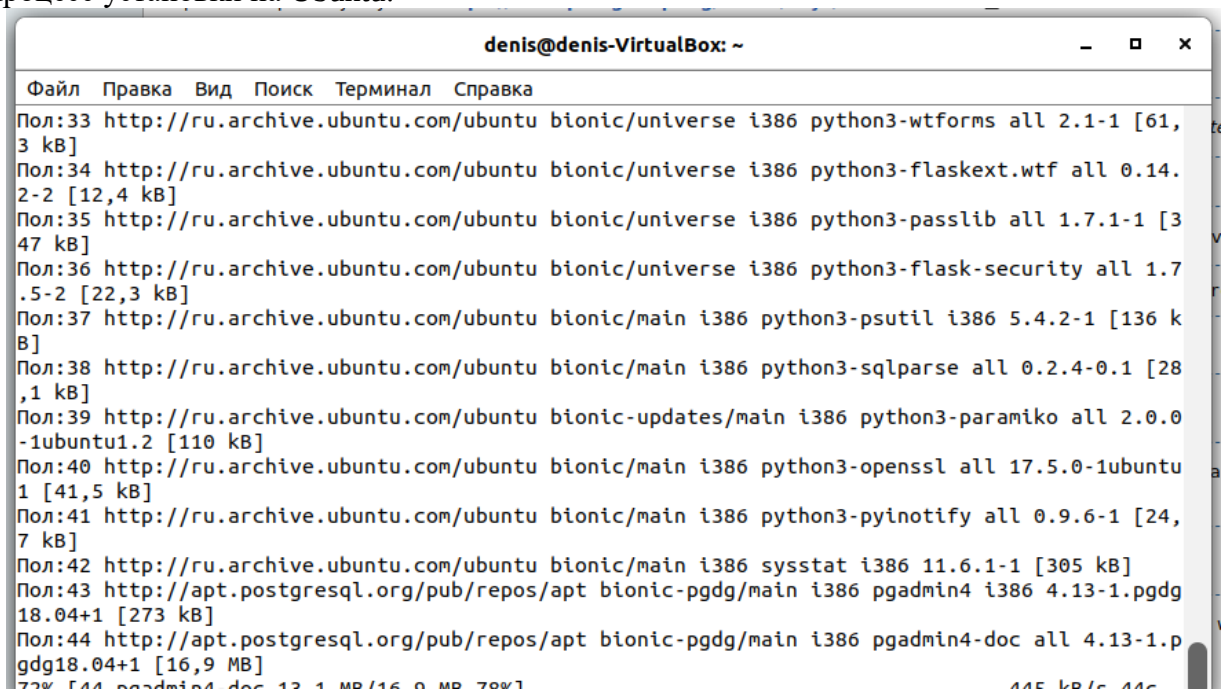
Запуск pgAdmin



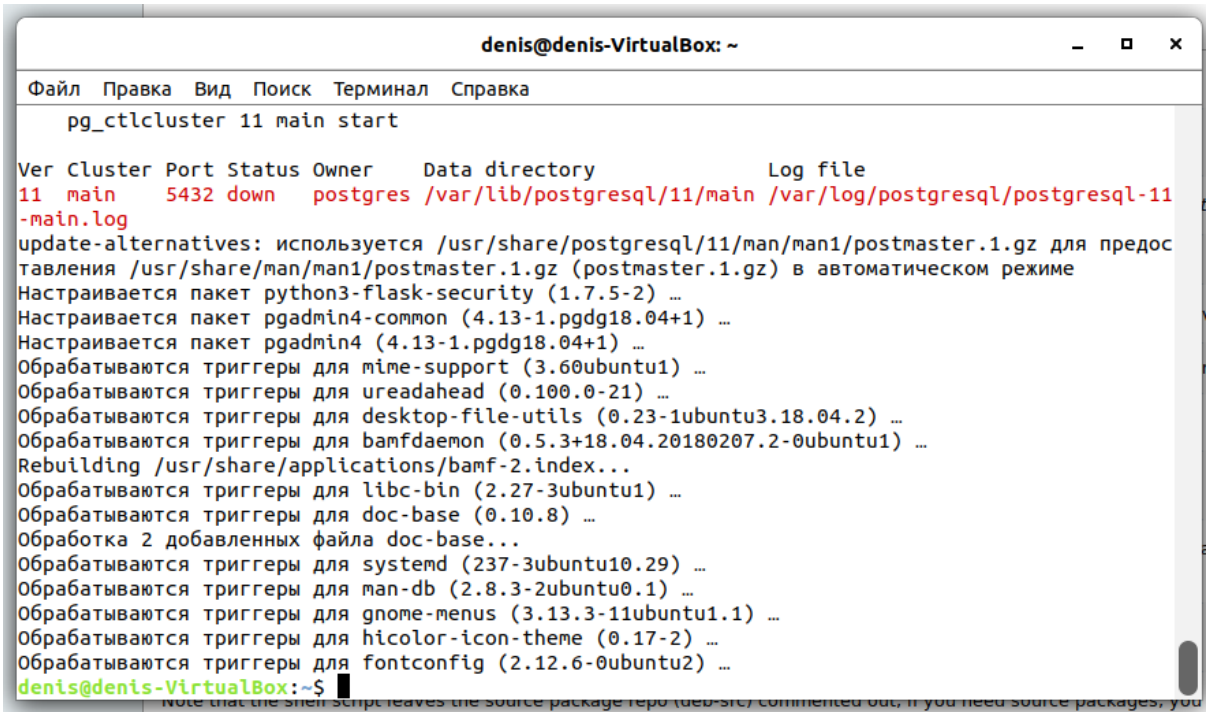
Окно
pgAdmin:



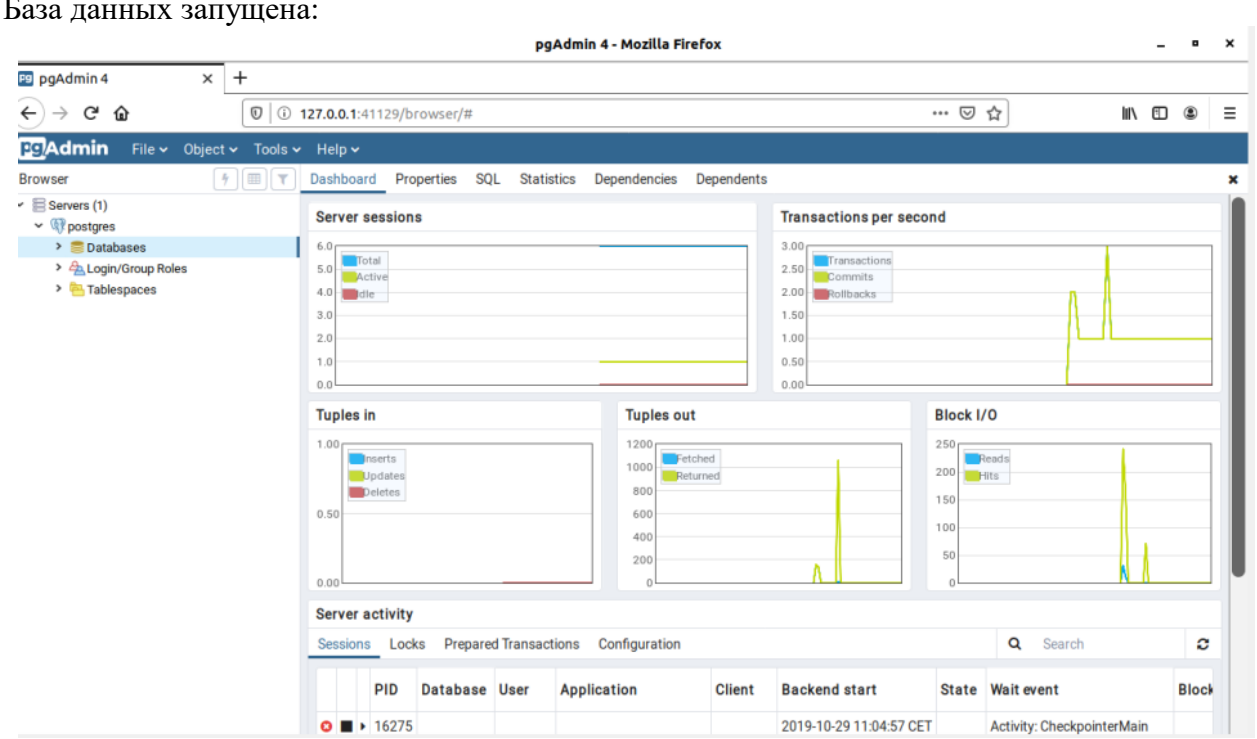
PostgreSQL на Ubuntu
Процесс установки на Ubuntu:



Установка завершена успешно!



База данных запущена:



6. Практическая контрольная работа № 7. Создание запросов на минимальные и максимальные значения

Задание:

1. Подключиться к базе данных demo.
В таблице «Самолеты» (aircrafts) есть столбец «Максимальная дальность полета» (range). Мы можем дополнить вывод данных из этой таблицы столбцом «Класс самолета», имея в виду принадлежность каждого самолета к классу дальнемагистральных, среднемагистральных или ближнемагистральных судов.
Команда для данных действий:

SELECT model, range,
CASE WHEN range < 2000 THEN 'Ближнемагистральный'
WHEN range < 5000 THEN 'Среднемагистральный'
ELSE 'Дальнемагистральный'
END AS type
FROM aircrafts
ORDER BY model;
В отчёт скриншот с результатом и с текстом пояснения конструкции команды.

2. В тех случаях, когда информации, содержащейся в одной таблице, недостаточно для получения требуемого результата, используют соединение (join) таблиц. С помощью команды:
SELECT a.aircraft_code, a.model, s.seat_no, s.fare_conditions
FROM seats AS s
JOIN aircrafts AS a
ON s.aircraft_code = a.aircraft_code
WHERE a.model ~ '^Cessna' ORDER BY s.seat_no;
объединим атрибуты двух таблиц Самолёты и Места. Обратите внимание на имена атрибутов — в выборке использованы псевдонимы атрибутов ,а объявлены они с помощью ключевого слова AS.
В отчёте написать комментарии к каждой строчке команды.

3. Напишите такие же запросы по местам для следующих самолётов:
Airbus A320-200
Bombardier CRJ-200
В отчёт скриншот с результатом.

4. В соединении одна и та же таблица может участвовать дважды, т. е. формировать соединение таблицы с самой собой. В качестве примера рассмотрим запрос для создания представления «Рейсы» (flights_v):
CREATE OR REPLACE VIEW flights_v AS
SELECT f.flight_id, f.flight_no,
f.scheduled_departure,
timezone(dep.timezone, f.scheduled_departure)
AS scheduled_departure_local,
f.scheduled_arrival,
timezone(arr.timezone, f.scheduled_arrival)
AS scheduled_arrival_local,
f.scheduled_arrival - f.scheduled_departure
AS scheduled_duration,
f.departure_airport, dep.airport_name AS
departure_airport_name, dep.city AS departure_city,
f.arrival_airport, arr.airport_name AS arrival_airport_name,
arr.city AS arrival_city, f.status,
f.aircraft_code,
f.actual_departure,
timezone(dep.timezone, f.actual_departure)
AS actual_departure_local,
f.actual_arrival,
timezone(arr.timezone, f.actual_arrival)
AS actual_arrival_local,
f.actual_arrival - f.actual_departure AS actual_duration
FROM flights f, airports dep, airports
arr

WHERE f.departure_airport = dep.airport_code AND f.arrival_airport = arr.airport_code;
В этом представлении используется не только таблица «Рейсы» (flights), но также и таблица «Аэропорты» (airports).
Выполнить данную команду
Вывести на экран содержимое данного представления.
В отчёт скриншот и пояснения по результату (что выводит данное представление?)

5. С помощью следующей команды выполняется запрос с подсчётом строк в соединённых двух одинаковых таблиц:
SELECT count(*)
FROM airports a1, airports a2
WHERE a1.city <> a2.city;
СУБД соединяет каждую строку первой таблицы с каждой строкой второй таблицы, т. е. формирует декартово произведение таблиц — все попарные комбинации строк из двух таблиц. Затем СУБД отбрасывает те комбинированные строки, которые не удовлетворяют условию, приведенному в предложении WHERE. В нашем примере условие как раз и отражает требование о том, что рейсов из одного города в тот же самый город быть не должно.
Выполните команду.

6. Во втором варианте запроса мы используем соединение таблиц на основе неравенства значений атрибутов. Тем самым мы перенесли условие отбора результирующих строк из предложения WHERE в предложение FROM.
SELECT count(*)
FROM airports a1
JOIN airports a2 ON a1.city <> a2.city;
Выполните команду.

7. Третий вариант предусматривает явное использование декартова произведения таблиц. Для этого служит предложение CROSS JOIN. Лишние строки, как и в первом варианте, отсеиваем с помощью предложения WHERE:
SELECT count(*)
FROM airports a1 CROSS JOIN airports a2 WHERE a1.city <> a2.city;
Выполните команду и вставьте скриншот с результатами.

8. Создайте такие же три запроса с таблицей Рейсы.
Скриншот с результатами в отчёт.
9. Создайте дополнительную таблицу Бронирования:
CREATE TABLE bookings
(book_ref char(6) NOT NULL, -- номер бронирования
book_date timestamptz NOT NULL, -- дата бронирования
total_amount numeric(10, 2) NOT NULL, -- полная стоимость бронирования (возможные значения в 10 цифр с плавающей точкой и двумя цифрами после запятой)
PRIMARY KEY (book_ref));-- первичный ключ – номер бронирования
И таблицу Билеты:
CREATE TABLE tickets
(ticket_no char(13) NOT NULL, -- номер билета
book_ref char(6) NOT NULL, -- номер бронирования (должен соответствовать данным из таблицы Бронирования)
passenger_id varchar(20) NOT NULL, -- идентификатор пассажира
passenger_name text NOT NULL, -- имя пассажира
contact_data jsonb, -- контактные данные пассажира
PRIMARY KEY (ticket_no), -- первичный ключ – номер билета
FOREIGN KEY (book_ref) -- внешний ключ номер бронирования

REFERENCES bookings (book_ref)); -- ссылочный ключ на таблицу Бронирования по номеру бронирования.

И таблицу Перелёты:

```
CREATE TABLE ticket_flights
( ticket_no char( 13 ) NOT NULL, -- Номер билета
 flight_id integer NOT NULL, -- Идентификатор рейса
fare_conditions varchar( 10 ) NOT NULL, -- Класс обслуживания
 amount numeric( 10, 2 ) NOT NULL, -- Стоимость перелета
 CHECK ( amount >= 0 ), -- ограничение -- стоимость перелёта больше или равна 0
 CHECK ( fare_conditions IN ( 'Economy', 'Comfort', 'Business' ) ), -- ограничение – класс обслуживания экономный, комфорт, бизнес
 PRIMARY KEY ( ticket_no, flight_id ), -- первичные ключи номер билета и номер рейса
 FOREIGN KEY ( flight_id ) -- внешний ключ номер рейса
 REFERENCES flights ( flight_id ), -- ссылочный ключ на таблицу Рейсы по номеру рейса
 FOREIGN KEY ( ticket_no ) -- внешний ключ номер билета
 REFERENCES tickets ( ticket_no ) -- ссылочный ключ на таблицу Билеты по номеру билета
);
```

И таблицу Посадочные талоны:

```
CREATE TABLE boarding_passes
( ticket_no char( 13 ) NOT NULL, -- Номер билета
 flight_id integer NOT NULL, -- Идентификатор рейса
 boarding_no integer NOT NULL, -- Номер посадочного талона
 seat_no varchar( 4 ) NOT NULL, -- Номер места
 PRIMARY KEY ( ticket_no, flight_id ),
 UNIQUE ( flight_id, boarding_no ),
 UNIQUE ( flight_id, seat_no ),
 FOREIGN KEY ( ticket_no, flight_id )
 REFERENCES ticket_flights ( ticket_no, flight_id )
);
```

10. Вывести на экран по очереди таблицы.

В отчёт скриншот.

11. Добавить по три строки в каждую таблицу (в таблицу бронирования добавить строку с суммой в 1 204 500 рублей).

12. При выполнении выборок зачастую выполняются многотабличные запросы, включающие три таблицы и более. В качестве примера рассмотрим такую задачу: определить число пассажиров, не пришедших на регистрацию билетов и, следовательно, не вылетевших в пункт назначения. Будем учитывать только рейсы, у которых фактическое время вылета не пустое, т. е. рейсы, имеющие статус «Departed» или «Arrived».

```
SELECT count( * )
FROM ( ticket_flights t
JOIN flights f ON t.flight_id = f.flight_id)
LEFT OUTER JOIN boarding_passes b
ON t.ticket_no = b.ticket_no AND t.flight_id = b.flight_id
WHERE f.actual_departure IS NOT NULL AND b.flight_id IS NULL;
```

В отчёт скриншот с результатом выборки и ответ на следующий вопрос: какие таблицы были использованы в этом запросе?

13. Для выработки финансовой стратегии нашей авиакомпании требуется следующая информация: распределение количества бронирований по диапазонам сумм с шагом в сто тысяч рублей. Максимальная сумма в одном бронировании составляет 1 204 500 рублей. Учтем это при формировании диапазонов стоимостей.
Виртуальной таблице, создаваемой с помощью ключевого слова VALUES, присваивают имя с помощью ключевого слова AS. После имени в круглых скобках приводится список имен столбцов этой таблицы.

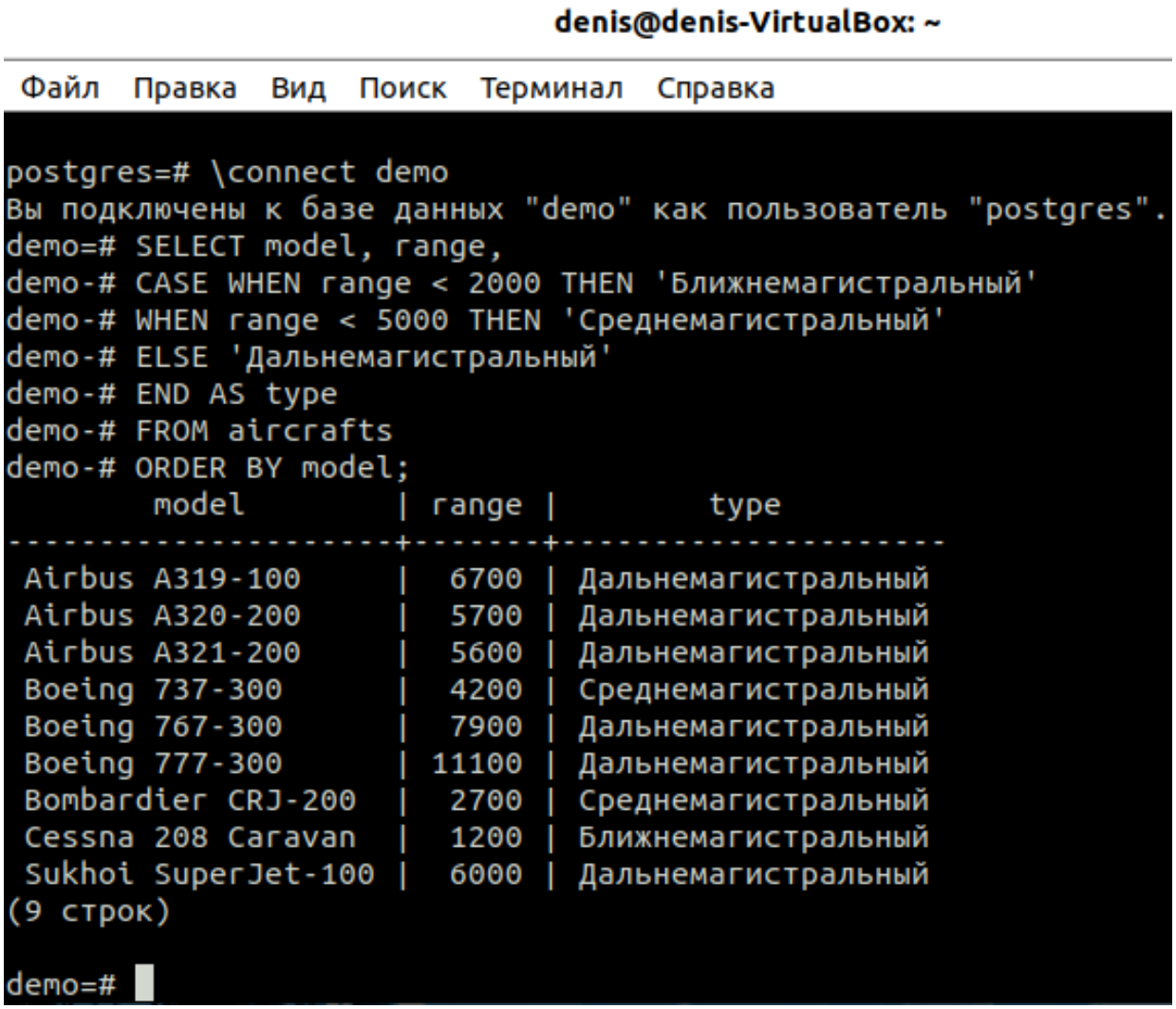
```
SELECT r.min_sum, r.max_sum, count( b.* )
FROM bookings b
RIGHT OUTER JOIN
        ( VALUES ( 0, 100000 ),          ( 100000, 200000 ),
        ( 200000, 300000 ),              ( 300000, 400000 ),
        ( 400000, 500000 ),              ( 500000, 600000 ),
        ( 600000, 700000 ),              ( 700000, 800000 ),
        ( 800000, 900000 ),              ( 900000, 1000000 ),
( 1000000, 1100000 ), ( 1100000, 1200000 ),
( 1200000, 1300000 )
) AS r ( min_sum, max_sum )
ON b.total_amount >= r.min_sum AND b.total_amount < r.max_sum
GROUP BY r.min_sum, r.max_sum
ORDER BY r.min_sum;
```

В этом запросе использовали внешнее соединение. Сделано это для того, чтобы в случаях, когда в каком-то диапазоне не окажется ни одного бронирования, результирующая строка выборки все же была бы сформирована. А правое соединение было выбрано только потому, что в качестве первой, базовой, таблицы мы выбрали таблицу «Бронирования» (bookings), но именно в ней может не оказаться ни одной строки для соединения с какой-либо строкой виртуальной таблицы. А все строки виртуальной таблицы, стоящей справа от предложения RIGHT OUTER JOIN, должны быть обязательно представлены в выборке: это позволит сразу увидеть «пустые» диапазоны, если они будут.
В этом запросе можно использовать и левое внешнее соединение, если поменять таблицы местами.

В отчёт скриншот с результатом запроса и комментарии по каждой строке запроса.

Эталон ответа:

1.



Пояснения:

SELECT model, range, “Выбираем столбцы model и range”
CASE WHEN range < 2000 THEN 'Ближнемагистральный' “Где дальность меньше 2000 там задаём значение ‘…’ ”
WHEN range < 5000 THEN 'Среднемагистральный' “Где дальность меньше 5000 там задаём значение ‘…’ ”
ELSE 'Дальнемагистральный' “Иначе значение ‘…’ ”
END AS type “Конец, задать как ‘type’ ”
FROM aircrafts “Выборка из таблицы самолёты”
ORDER BY model; “Сортировка по моделям”

2.

```
demo=# SELECT a.aircraft_code, a.model, s.seat_no, s.fare_conditions_code
FROM seats AS s
JOIN aircrafts AS a
ON s.aircraft_code = a.aircraft_code
WHERE a.model ~ '^Cessna' ORDER BY s.seat_no;
 aircraft_code |       model        | seat_no | fare_conditions_code
---------------+--------------------+---------+----------------------
 CN1           | Cessna 208 Caravan | 10A     |                    1
 CN1           | Cessna 208 Caravan | 10B     |                    1
 CN1           | Cessna 208 Caravan | 10F     |                    1
 CN1           | Cessna 208 Caravan | 1A      |                    2
 CN1           | Cessna 208 Caravan | 1B      |                    2
 CN1           | Cessna 208 Caravan | 2F      |                    1
 CN1           | Cessna 208 Caravan | 3A      |                    2
 CN1           | Cessna 208 Caravan | TA      |                    2
(8 строк)

demo=#
```

SELECT a.aircraft_code, a.model, s.seat_no, s.fare_conditions “Выбор кодовых имён, моделей самолётов, номеров сидений, классов”
FROM seats AS s “из мест как (места)”
JOIN aircrafts AS a “присоединить самолёты как (самолёты)”
ON s.aircraft_code = a.aircraft_code “сравнить код самолёта (места) и код самолёта (самолёты)”
WHERE a.model ~ '^Cessna' ORDER BY s.seat_no; “где модель (самолёты) подобно Цесна, сортировать по номерам мест”

3.

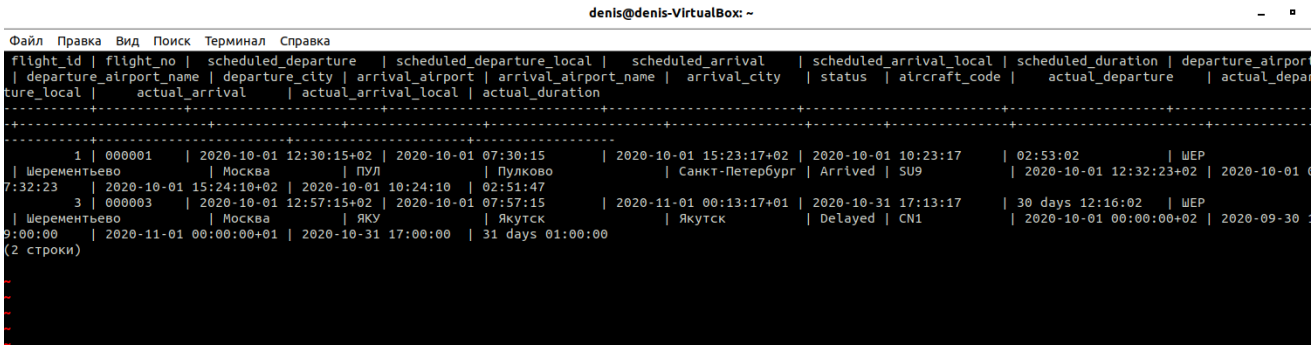
```
demo=# SELECT a.aircraft_code, a.model, s.seat_no, s.fare_conditions_code
FROM seats AS s
JOIN aircrafts AS a
ON s.aircraft_code = a.aircraft_code
WHERE a.model ~ '^Airbus' ORDER BY s.seat_no;
 aircraft_code |      model      | seat_no | fare_conditions_code
---------------+-----------------+---------+----------------------
 320           | Airbus A320-200 | 10A     |                    1
 320           | Airbus A320-200 | 10B     |                    1
 320           | Airbus A320-200 | 10F     |                    1
 320           | Airbus A320-200 | 1A      |                    2
 320           | Airbus A320-200 | 1B      |                    2
 320           | Airbus A320-200 | 2F      |                    1
 320           | Airbus A320-200 | 3A      |                    2
 320           | Airbus A320-200 | TA      |                    2
(8 строк)

demo=#
```

```
demo=# SELECT a.aircraft_code, a.model, s.seat_no, s.fare_conditions_code
FROM seats AS s
JOIN aircrafts AS a
ON s.aircraft_code = a.aircraft_code
WHERE a.model ~ '^Bombardier' ORDER BY s.seat_no;
 aircraft_code |       model         | seat_no | fare_conditions_code
---------------+---------------------+---------+----------------------
 CR2           | Bombardier CRJ-200 | 10A     |                    1
 CR2           | Bombardier CRJ-200 | 10B     |                    1
 CR2           | Bombardier CRJ-200 | 10F     |                    1
 CR2           | Bombardier CRJ-200 | 1A      |                    2
 CR2           | Bombardier CRJ-200 | 1B      |                    2
 CR2           | Bombardier CRJ-200 | 2F      |                    1
 CR2           | Bombardier CRJ-200 | 3A      |                    2
 CR2           | Bombardier CRJ-200 | TA      |                    2
(8 строк)

demo=#
```

4.



Выводятся сведения о рейсах, совместно с информацией о времени вылета и прибытия, а также моделью самолётов и продолжительностью полётов.

5.

6.

```
demo=# SELECT count(*)
demo-# FROM airports a1, airports a2
demo-# WHERE a1.city <> a2.city;
 count
-------
    30
(1 строка)

demo=#
```

```
demo=# SELECT count(*)
FROM airports a1
JOIN airports a2 ON a1.city <> a2.city;
 count
-------
    30
(1 строка)

demo=#
```

7.

```
demo=# SELECT count(*)
FROM airports a1 CROSS JOIN airports a2 WHERE a1.city <> a2.city;
 count
-------
    30
(1 строка)

demo=#
```

8.

```
demo=# SELECT count(*)
FROM flights a1 CROSS JOIN airports a2 WHERE a1.arrival_city <> a2.city;
 count
-------
    10
(1 строка)

demo=#
```

9.

10.

```
demo=# SELECT * FROM bookings;
 book_ref | book_date | total_amount
----------+-----------+--------------
(0 строк)

demo=# SELECT * FROM tickets;
 ticket_no | book_ref | passenger_id | passenger_name | contact_data
-----------+----------+--------------+----------------+--------------
(0 строк)

demo=# SELECT * FROM ticket_flights;
 ticket_no | flight_id | fare_conditions | amount
-----------+-----------+-----------------+--------
(0 строк)

demo=# SELECT * FROM boarding_passes;
 ticket_no | flight_id | boarding_no | seat_no
-----------+-----------+-------------+---------
(0 строк)

demo=#
```

11.

```
demo=# INSERT INTO bookings VALUES ( '1', '22.01.2020', '1204500');
INSERT 0 1
demo=# INSERT INTO bookings VALUES ( '1', '22.01.2020', '14500');
ОШИБКА:  повторяющееся значение ключа нарушает ограничение уникальности "booki
ngs_pkey"
ПОДРОБНОСТИ:  Ключ "(book_ref)=(1     )" уже существует.
demo=# INSERT INTO bookings VALUES ( '1', '22.01.2020', '14500');
ОШИБКА:  повторяющееся значение ключа нарушает ограничение уникальности "booki
ngs_pkey"
ПОДРОБНОСТИ:  Ключ "(book_ref)=(1     )" уже существует.
demo=# INSERT INTO bookings VALUES ( '1', '22.01.2020', '1204500');
ОШИБКА:  повторяющееся значение ключа нарушает ограничение уникальности "booki
ngs_pkey"
ПОДРОБНОСТИ:  Ключ "(book_ref)=(1     )" уже существует.
demo=# INSERT INTO bookings VALUES ( '2', '25.02.2020', '12500');
INSERT 0 1
demo=# INSERT INTO bookings VALUES ( '2', '25.02.2020', '454005');
ОШИБКА:  повторяющееся значение ключа нарушает ограничение уникальности "booki
ngs_pkey"
ПОДРОБНОСТИ:  Ключ "(book_ref)=(2     )" уже существует.
demo=# INSERT INTO bookings VALUES ( '3', '27.02.2020', '454005');
INSERT 0 1
demo=# SELECT * FROM bookings;
 book_ref |       book_date        | total_amount
----------+------------------------+--------------
 1        | 2020-01-22 00:00:00+01 |   1204500.00
 2        | 2020-02-25 00:00:00+01 |     12500.00
 3        | 2020-02-27 00:00:00+01 |    454005.00
(3 строки)
```

```
demo=# INSERT INTO tickets VALUES ( '01','1', '123456', 'Roman Ivanov', '89046
250712'),
( '02', '2', '234567', 'Maria Petrova', '89117540214'),
( '03', '3', '345678', 'Daria Menshikova', '89534560501');
INSERT 0 3
demo=# SELECT * FROM tickets;
   ticket_no   | book_ref | passenger_id |  passenger_name  | contact_data
---------------+----------+--------------+------------------+--------------
 01            | 1        | 123456       | Roman Ivanov     | 89046250712
 02            | 2        | 234567       | Maria Petrova    | 89117540214
 03            | 3        | 345678       | Daria Menshikova | 89534560501
(3 строки)

demo=#
```

```
demo=# INSERT INTO ticket_flights VALUES ( '01','1', 'Business', '1204500'),(
'02', '3', 'Economy', '12500'),
( '03', '1', 'Comfort', '454005');
INSERT 0 3
demo=# SELECT * FROM ticket_flights;
   ticket_no   | flight_id | fare_conditions |   amount
---------------+-----------+-----------------+------------
 01            |         1 | Business        | 1204500.00
 02            |         3 | Economy         |   12500.00
 03            |         1 | Comfort         |  454005.00
(3 строки)

demo=#
```

```
demo=# INSERT INTO boarding_passes VALUES ( '01','1', '0001', '1A'),( '02', '3
', '0002', 'TA'),( '03', '1', '0003', '2B');
INSERT 0 3
demo=# SELECT * FROM boarding_passes;
   ticket_no   | flight_id | boarding_no | seat_no
---------------+-----------+-------------+---------
 01            |         1 |           1 | 1A
 02            |         3 |           2 | TA
 03            |         1 |           3 | 2B
(3 строки)

demo=#
```

12.

```
demo=# SELECT count( * )
demo-# FROM ( ticket_flights t
demo(# JOIN flights f ON t.flight_id = f.flight_id)
demo-# LEFT OUTER JOIN boarding_passes b
demo-# ON t.ticket_no = b.ticket_no AND t.flight_id = b.flight_id
demo-# WHERE f.actual_departure IS NOT NULL AND b.flight_id IS NULL;
 count
-------
     0
(1 строка)

demo=#
```

Были использованы таблицы, созданные в 11 пункте, и таблица «Полёты»».

13. Д

```
demo=#      SELECT r.min_sum, r.max_sum, count( b.* )
demo-# FROM bookings b
demo-# RIGHT OUTER JOIN
demo-# ( VALUES ( 0, 100000 ),
demo(# ( 100000, 200000 ),
demo(# ( 200000, 300000 ),
demo(# ( 300000, 400000 ),
demo(# ( 400000, 500000 ),
demo(# ( 500000, 600000 ),
demo(# ( 600000, 700000 ),
demo(# ( 700000, 800000 ),
demo(# ( 800000, 900000 ),
demo(# ( 900000, 1000000 ),
demo(# ( 1000000, 1100000 ), ( 1100000, 1200000 ),
demo(# ( 1200000, 1300000 )
demo(# ) AS r ( min_sum, max_sum )
demo-# ON b.total_amount >= r.min_sum AND b.total_amount < r.max_sum
demo-# GROUP BY r.min_sum, r.max_sum
demo-# ORDER BY r.min_sum;
 min_sum | max_sum | count
---------+---------+-------
       0 |  100000 |     1
  100000 |  200000 |     0
  200000 |  300000 |     0
  300000 |  400000 |     0
  400000 |  500000 |     1
  500000 |  600000 |     0
  600000 |  700000 |     0
  700000 |  800000 |     0
  800000 |  900000 |     0
  900000 | 1000000 |     0
 1000000 | 1100000 |     0
 1100000 | 1200000 |     0
 1200000 | 1300000 |     1
(13 строк)

demo=#
```

9. Практическая контрольная работа № 8. Использование блокировок — встроенных механизмов защиты информации

Задание:
Подключиться к базе данных demo.
Включите секундомер на одном терминале.
1. На первом терминале организуйте транзакцию с уровнем изоляции READ COMMITTED и выполните следующую команду:
SELECT * FROM aircrafts_tmp WHERE model ~ '^Air' FOR UPDATE;
2. На втором терминале организуйте аналогичную транзакцию и выполните точно такую же команду. Вы увидите, что ее выполнение будет приостановлено.
SELECT * FROM aircrafts_tmp WHERE model ~ '^Air' FOR UPDATE;
3. На первом терминале обновите одну строку, а затем завершите транзакцию:
UPDATE aircrafts_tmp
SET range = 5800
WHERE aircraft_code = '320';
Перейдя на второй терминал, вы увидите, что там была, наконец, выполнена выборка, которая показала уже измененные данные.
4. Завершите и вторую транзакцию.
5. Аналогичным образом можно организовать блокировки на уровне таблиц. Также на первом терминале организуйте транзакцию с уровнем изоляции READ COMMITTED и выполните команду блокировки всей таблицы в самом строгом режиме, в котором другим транзакциям доступ к этой таблице запрещен полностью:
LOCK TABLE aircrafts_tmp IN ACCESS EXCLUSIVE MODE;
6. На втором терминале выполните совершенно «безобидную» команду:
SELECT * FROM aircrafts_tmp WHERE model ~ '^Air';
Вы увидите, что выполнение команды SELECT на втором терминале будет задержано.
7. Прервите транзакцию на первом терминале командой ROLLBACK. Вы увидите, что на втором терминале команда будет успешно выполнена.
8. С помощью Документации по PostgreSQL (раздел 13.3 «Явные блокировки») найдите и вставьте в отчёт информацию о режимах блокировок на уровне таблицы.
9. С помощью Документации по PostgreSQL выясните на каких ещё уровнях бывают блокировки. Перечислите их в отчёте.
10. Самостоятельно ознакомьтесь с предложением FOR SHARE команды SELECT и выполните необходимые эксперименты. Используйте документацию: раздел 13.3.2 «Блокировки на уровне строк» и описание команды SELECT.

Эталон ответа

1.

```
postgres=# \connect demo
Вы подключены к базе данных "demo" как пользователь "postgres".
demo=# SELECT * FROM aircrafts_tmp WHERE model ~ '^Air' FOR UPDATE;
 aircraft_code |      model      | range
---------------+-----------------+-------
 321           | Airbus A321-200 |  5600
 319           | Airbus A319-100 |  6700
 320           | Airbus A320-200 |  5900
(3 строки)

demo=#
```

2.

```
demo=# SELECT * FROM aircrafts_tmp WHERE model ~ '^Air' FOR UPDATE;
 aircraft_code |      model      | range
---------------+-----------------+-------
 321           | Airbus A321-200 |  5600
 319           | Airbus A319-100 |  6700
 320           | Airbus A320-200 |  5900
(3 строки)

demo=#
```

3.

```
demo=# UPDATE aircrafts_tmp
demo-# SET range = 5800
demo-# WHERE aircraft_code = '320';
UPDATE 1
demo=#
```

4.

```
                                    denis@denis-VirtualBox: ~
Файл   Правка   Вид   Поиск   Терминал   Справка
[sudo] пароль для denis:
postgres@denis-VirtualBox:/home/denis$ psql
psql (11.5 (Ubuntu 11.5-3.pgdg18.04+1))
Введите "help", чтобы получить справку.

postgres=# \connect demo
Вы подключены к базе данных "demo" как пользователь "postgres".
demo=# SELECT * FROM aircrafts_tmp WHERE model ~ '^Air' FOR UPDATE;
 aircraft_code |      model      | range
---------------+-----------------+-------
 321           | Airbus A321-200 |  5600
 319           | Airbus A319-100 |  6700
 320           | Airbus A320-200 |  5900
(3 строки)

demo=# SELECT * FROM aircrafts_tmp WHERE model ~ '^Air' FOR UPDATE;
 aircraft_code |      model      | range
---------------+-----------------+-------
 321           | Airbus A321-200 |  5600
 319           | Airbus A319-100 |  6700
 320           | Airbus A320-200 |  5800
(3 строки)

demo=#
```

5.

```
demo=# BEGIN;
BEGIN
demo=# LOCK TABLE aircrafts_tmp IN ACCESS EXCLUSIVE MODE;
LOCK TABLE
demo=#
```

6.

```
demo=# SELECT * FROM aircrafts_tmp WHERE model ~ '^Air';


```

7.

```
demo=# ROLLBACK;
ROLLBACK
demo=#
ПРЕДУПРЕЖДЕНИЕ:  НЕТ НЕЗАВЕРШЕННОЙ ТРАНЗАКЦИИ
COMMIT
demo=# SELECT * FROM aircrafts_tmp WHERE model ~ '^Air';


 aircraft_code |      model      | range
---------------+-----------------+-------
 321           | Airbus A321-200 |  5600
 319           | Airbus A319-100 |  6700
 320           | Airbus A320-200 |  5800
(3 строки)

demo=#
demo=#
demo=#
```

8.

Режимы блокировок на уровне таблицы:

ACCESS SHARE
Конфликтует только с режимом блокировки ACCESS EXCLUSIVE.

Команда SELECT получает такую блокировку для таблиц, на которые она ссылается. Вообще говоря, блокировку в этом режиме получает любой запрос, который только читает таблицу, но не меняет её данные.

ROW SHARE
Конфликтует с режимами блокировки EXCLUSIVE и ACCESS EXCLUSIVE.

Команды SELECT FOR UPDATE и SELECT FOR SHARE получают такую блокировку для своих целевых таблиц (помимо блокировок ACCESS SHARE для любых таблиц, которые используется в этих запросах, но не в предложении FOR UPDATE/FOR SHARE).

ROW EXCLUSIVE
Конфликтует с режимами блокировки SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE и ACCESS EXCLUSIVE.

Команды UPDATE, DELETE и INSERT получают такую блокировку для целевой таблицы (в дополнение к блокировкам ACCESS SHARE для всех других задействованных таблиц). Вообще говоря, блокировку в этом режиме получает любая команда, которая изменяет данные в таблице.

SHARE UPDATE EXCLUSIVE

Конфликтует с режимами блокировки SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE и ACCESS EXCLUSIVE. Этот режим защищает таблицу от параллельного изменения схемы и запуска процесса VACUUM.

Запрашивается командами VACUUM (без FULL), ANALYZE, CREATE INDEX CONCURRENTLY, CREATE STATISTICS, ALTER TABLE VALIDATE и другими видами ALTER TABLE (за подробностями обратитесь к ALTER TABLE).

SHARE
Конфликтует с режимами блокировки ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE ROW EXCLUSIVE, EXCLUSIVE и ACCESS EXCLUSIVE. Этот режим защищает таблицу от параллельного изменения данных.

Запрашивается командой CREATE INDEX (без параметра CONCURRENTLY).

SHARE ROW EXCLUSIVE
Конфликтует с режимами блокировки ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE и ACCESS EXCLUSIVE. Этот режим защищает таблицу от параллельных изменений данных и при этом он является самоисключающим, так что такую блокировку может получить только один сеанс.

Запрашивается командой CREATE COLLATION, CREATE TRIGGER и многими формами ALTER TABLE (см. ALTER TABLE).

EXCLUSIVE
Конфликтует с режимами блокировки ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE и ACCESS EXCLUSIVE. Этот режим совместим только с блокировкой ACCESS SHARE, то есть параллельно с транзакцией, получившей блокировку в этом режиме, допускается только чтение таблицы.

Запрашивается командой REFRESH MATERIALIZED VIEW CONCURRENTLY.

ACCESS EXCLUSIVE
Конфликтует со всеми режимами блокировки (ACCESS SHARE, ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE и ACCESS EXCLUSIVE). Этот режим гарантирует, что кроме транзакции, получившей эту блокировку, никакая другая транзакция не может обращаться к таблице каким-либо способом.

Запрашивается командами DROP TABLE, TRUNCATE, REINDEX, CLUSTER, VACUUM FULL и REFRESH MATERIALIZED VIEW (без CONCURRENTLY). Блокировку на этом уровне запрашивают также многие виды ALTER TABLE. В этом режиме по умолчанию запрашивают блокировку и операторы LOCK TABLE, если явно не выбран другой режим.

9.

- Блокировки на уровне таблицы;
- Блокировки на уровне строк;
- Блокировки на уровне страниц.

10. Устный зачет по темам

Инструкция для обучающихся: Зачет сдается в рамках учебного занятия. Каждому студенту по выбору преподавателя дается два вопроса, на которые он отвечает в устной форме.
Выполнение задания: одному студенту на ответ выделяется 5 мин, группа сдает зачет за одно учебное занятие.

Вопросы к зачету:

1. Понятие буферного кэша в PostgreSQL
2. Понятие блокировок в PostgreSQL
3. Понятие кластера баз данных
4. Понятие транзакции и её свойств
5. Понятие журналирования и виды уровней журнала
6. Понятие контрольной точки в PostgreSQL
7. Понятие многоверсионности и уровней изоляции
8. Понятие обработки запросов
9. Все виды ограничений в таблицах в СУБД PostgreSQL
10. Понятие первичного ключа
11. Понятие внешнего ключа

Эталоны ответов: приведены в Учебном пособии по дисциплине «Базы данных».

Контрольно-оценочные материалы для промежуточной аттестации по дисциплине

Формой промежуточной аттестации по учебной дисциплине является **дифференцированный зачет**

Перечень вопросов для дифференцированного зачета:

1. Основные понятия теории баз данных (определение базы данных, кластера, экземпляра БД, требования к СУБД, Функции СУБД, компоненты СУБД)
2. Основные понятия модели данных (определение модели данных, характеристики модели данных, классификация моделей данных, примеры для основных видов моделей данных)
3. Основные понятия реляционной базы данных (определение; структура; понятие атрибута, домена, кортежа; виды ключей; понятие связи)
4. Логическая и физическая модель данных (понятие объекта, репликации базы данных, транзакции; отличие логической модели данных от физической с примерами)
5. Принципы проектирования баз данных (понятие ER-модели; этапы создания ER-диаграммы; понятие сущности; понятие отношения; привести примеры)
6. Этапы проектирования баз данных (понятие нормальной формы; процесс нормализации; описание первых 3-х нормальных форм с примерами)
7. Реляционная алгебра (восемь операций реляционной алгебры с примерами)

8. Архитектура PostgreSQL (буферный кеш; общая память, конкурентный доступ, блокировки)
9. Организация данных в PostgreSQL (хранение данных, хранение объектов, транзакции)
10. Организация данных в PostgreSQL (журнал упреждающей записи, понятие контрольной точки, уровни журнала)
11. Организация данных в PostgreSQL (понятие многоверсионности, понятие очистки данных, снимок данных, уровни изоляции)
12. Обработка запросов в PostgreSQL (процессы, анализатор, планировщик, исполнитель)
13. Виды ограничений в PostgreSQL (значение по умолчанию, check, имена ограничений, not null, ограничение уникальности)
14. Использование ключей в PostgreSQL (первичный ключ, внешний ключ, работа со связанными строками при удалении строк из другой таблицы)
15. Модификация таблиц и представления в PostgreSQL
16. Схемы данных в PostgreSQL (понятие; просмотр списка в базе данных; команда доступа к схеме данных в базе данных; способы доступа к объектам схемы данных)
17. Построение запросов в СУБД PostgreSQL(возможности команды SELECT, соединения, агрегирование и группировка, подзапросы)
18. Изменение данных в PostgreSQL (вставка строк, обновление строк, удаление строк в таблицах)
19. Индексы в PostgreSQL (понятие индексов; индексы по нескольким столбцам; уникальные индексы; индексы на основе выражений; частичные индексы)
20. Транзакции в PostgreSQL (общая информация о транзакциях; уровень изоляции Read Uncommitted; уровень изоляции Read Committed; уровень изоляции Repeatable Read; уровень изоляции Serializable; блокировки)
21. Методы просмотра данных в PostgreSQL с целью повышения производительности (методы просмотра таблиц; методы формирования соединений наборов строк; управление планировщиком)
22. Оптимизация запросов в PostgreSQL (Изменение схемы данных для повышения производительности базы данных; модификация запросов для повышения производительности базы данных)