

Установка и настройка СУБД PostgreSQL

Загрузить пакет ПО, необходимого для работы с PostgreSQL можно [здесь](#).

При возникновении вопросов по работе с PostgreSQL следует обратиться к [документации](#) ([на русском](#)).

Во время установки следует оставить все параметры по умолчанию.

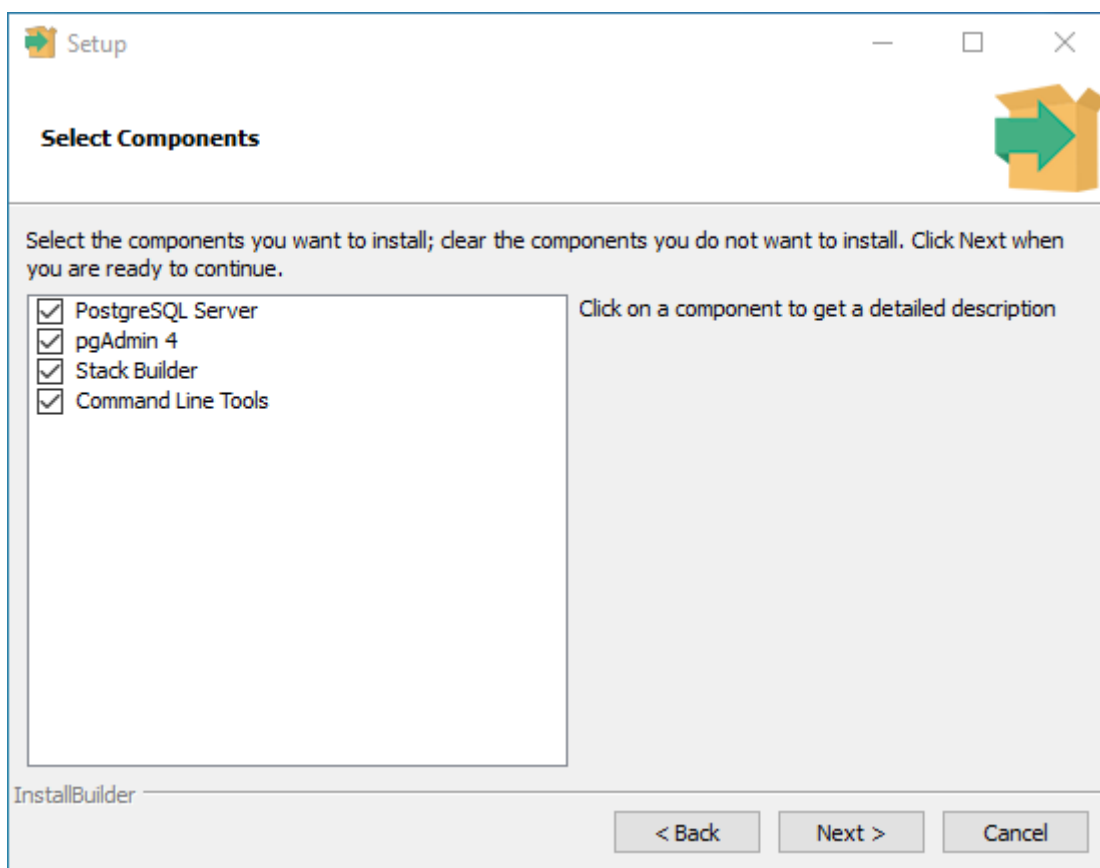


Рис. 1 Установка PostgreSQL

Далее необходимо задать пароль для суперпользователя postgres.

Пароль суперпользователя крайне не рекомендуется забывать. Во время выполнения практики рекомендуется использовать пароль «postgres».

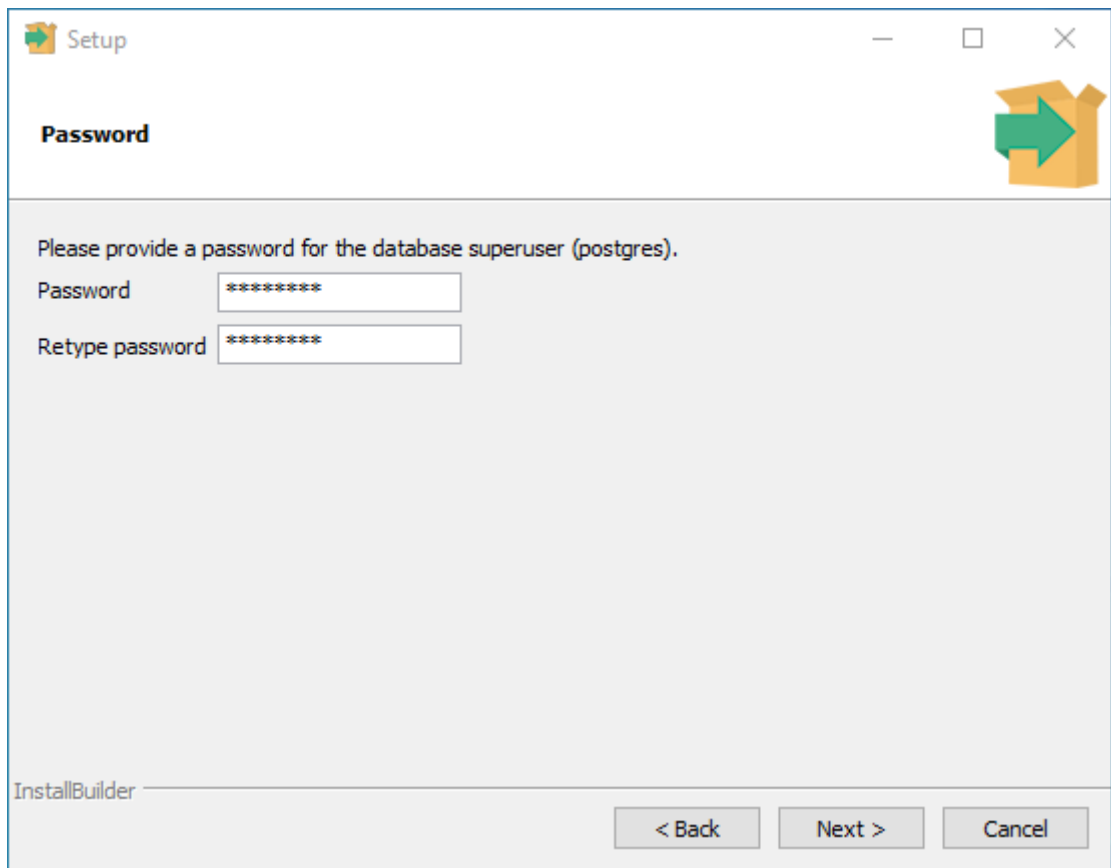


Рис. 2 Установка пароля суперпользователя PostgreSQL

После завершения установки следует предотвратить запуск Stack Builder.

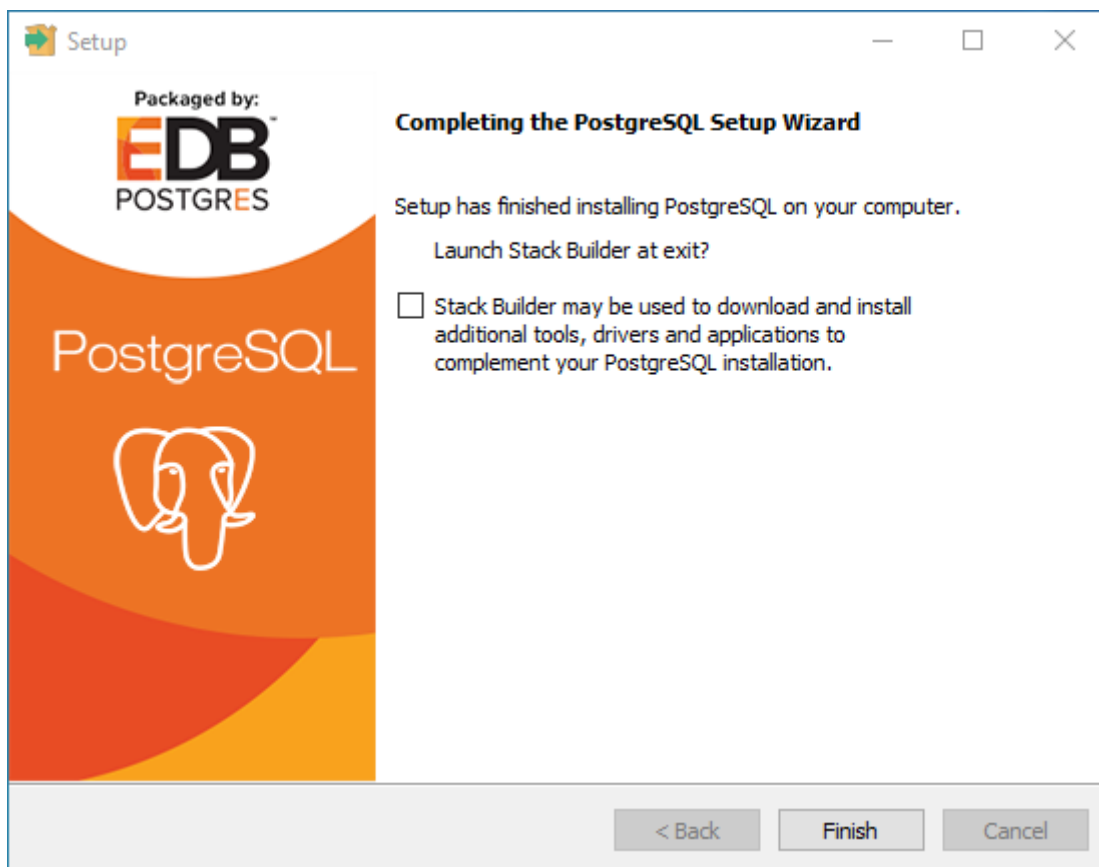


Рис. 3 Завершение установки PostgreSQL

Работа с системой управления базами данных PostgreSQL осуществляется с помощью программы psql.

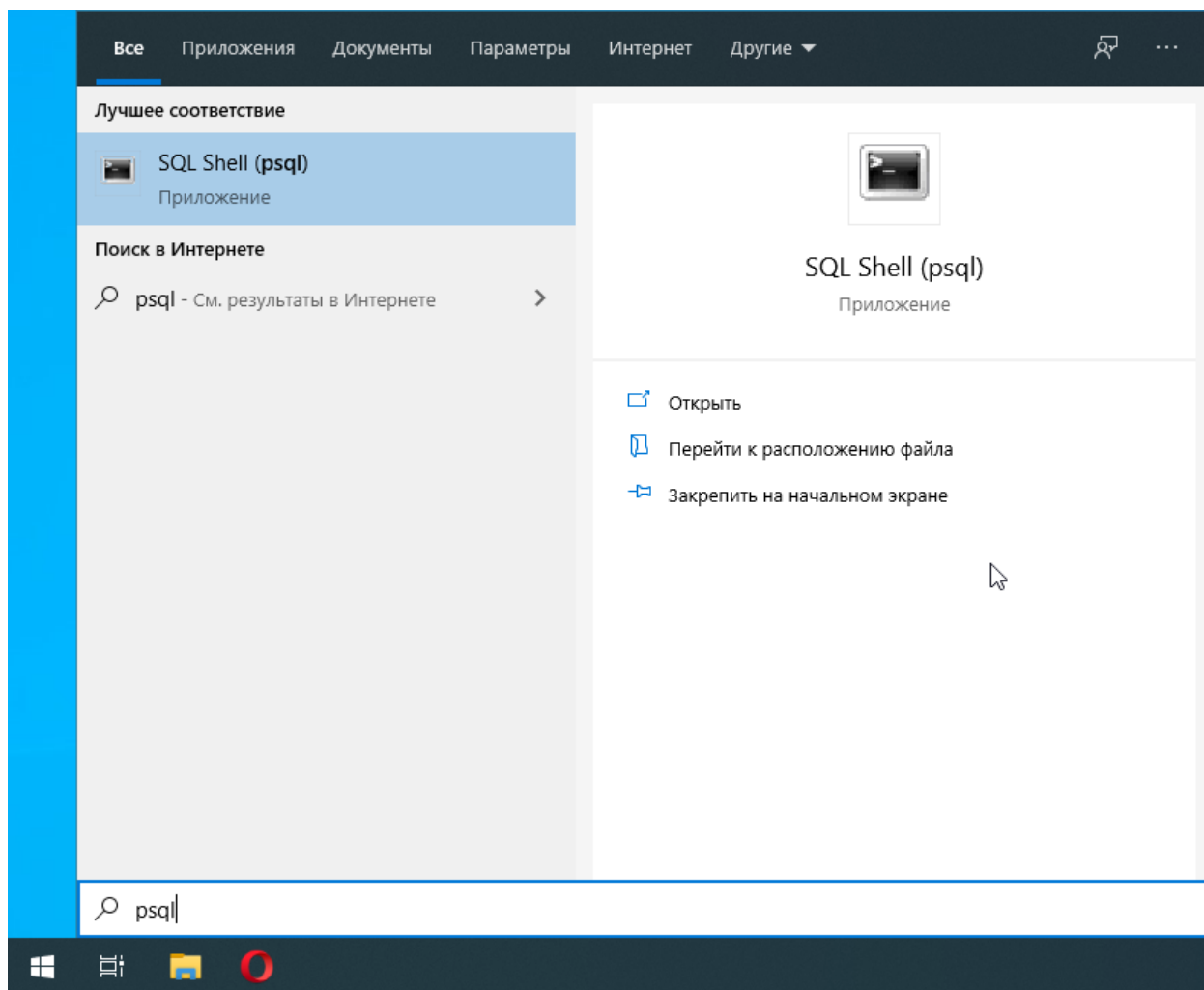
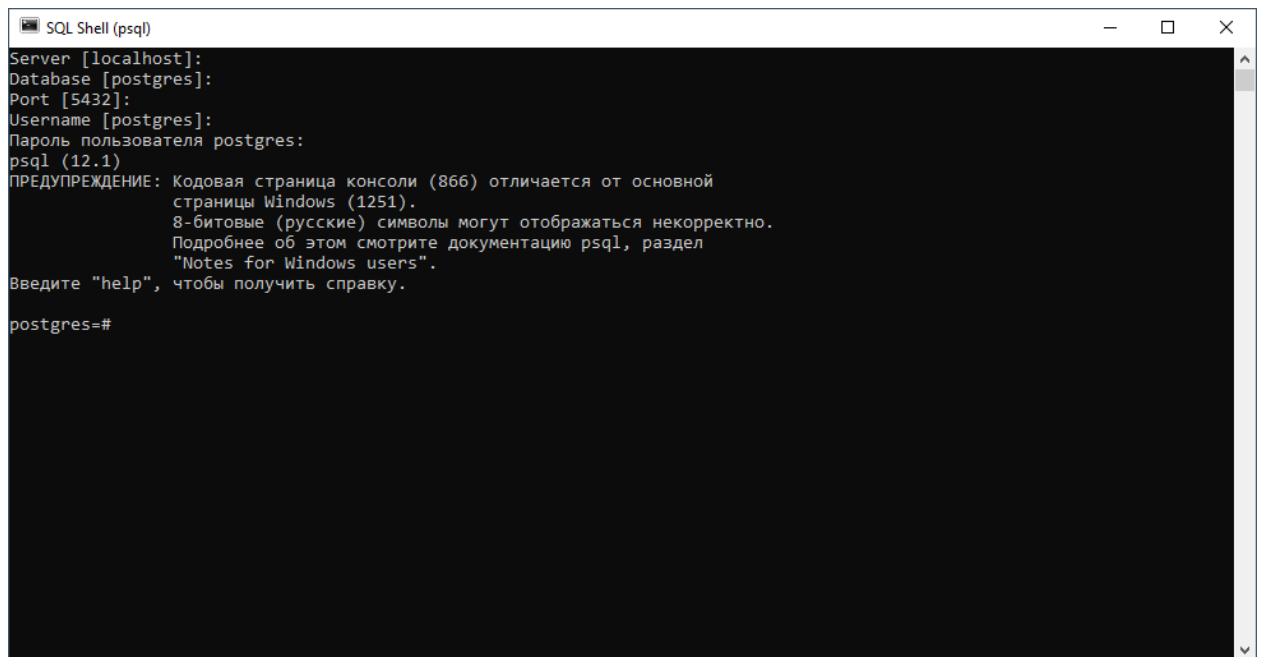


Рис. 4 Запуск psql

При первом запуске psql предупредит о несоответствии кодировок, что может вызвать неудобства при работе с ней.



```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Пароль пользователя postgres:
psql (12.1)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
                  страницы Windows (1251).
                  8-битовые (русские) символы могут отображаться некорректно.
                  Подробнее об этом смотрите документацию psql, раздел
                  "Notes for Windows users".
Введите "help", чтобы получить справку.
postgres=#
```

Рис. 5 Первый запуск psql

Для устранения неудобства необходимо исправить скрипт запуска psql.
Для этого запустим любой текстовый редактор **с правами администратора**.

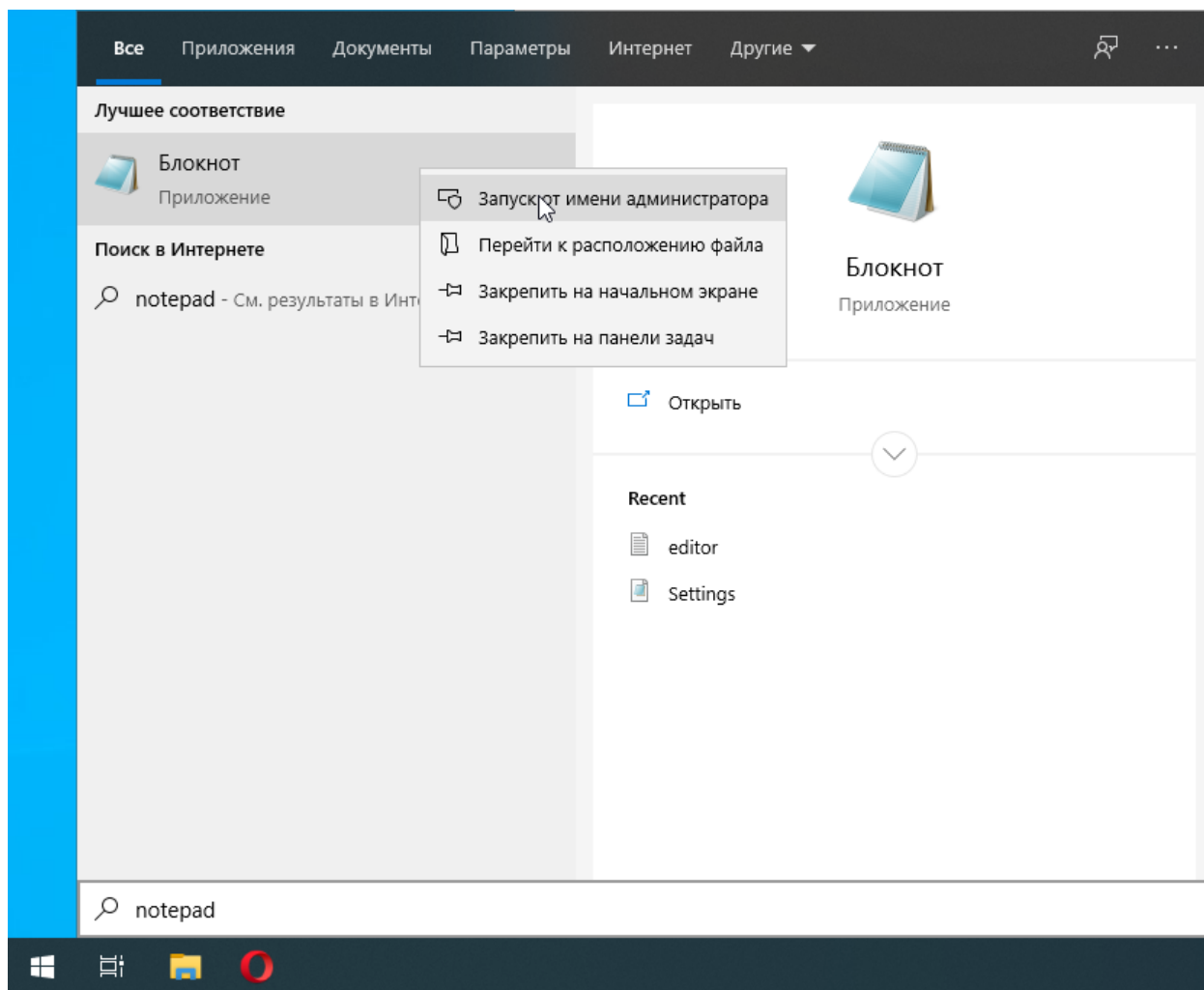


Рис. 6 Запуск стандартного текстового редактора MS Windows

Затем откроем для редактирования скрипт запуска `psql`, расположение которого по умолчанию отображено на рисунке 7 ниже.

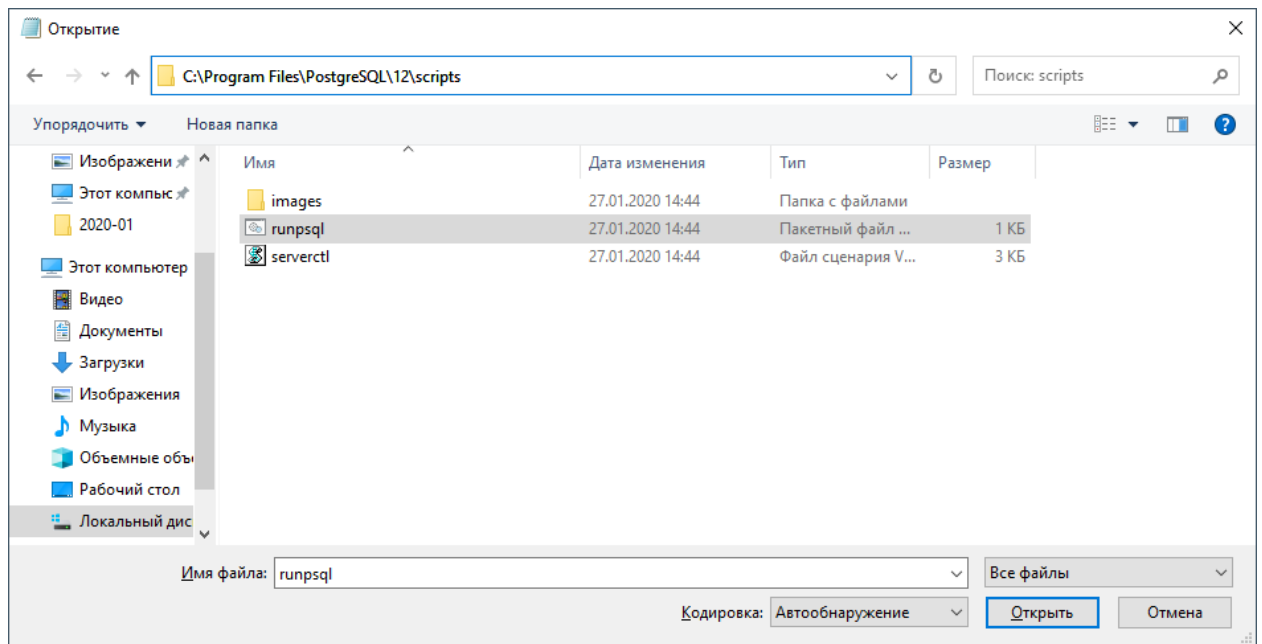


Рис. 7 Открытие скрипта запуска psql для редактирования

Для исправления проблемы необходимо добавить строчку «CHCP 1251», которая будет при запуске psql устанавливать кодировку 1251, которая соответствует системной. Затем файл следует сохранить.

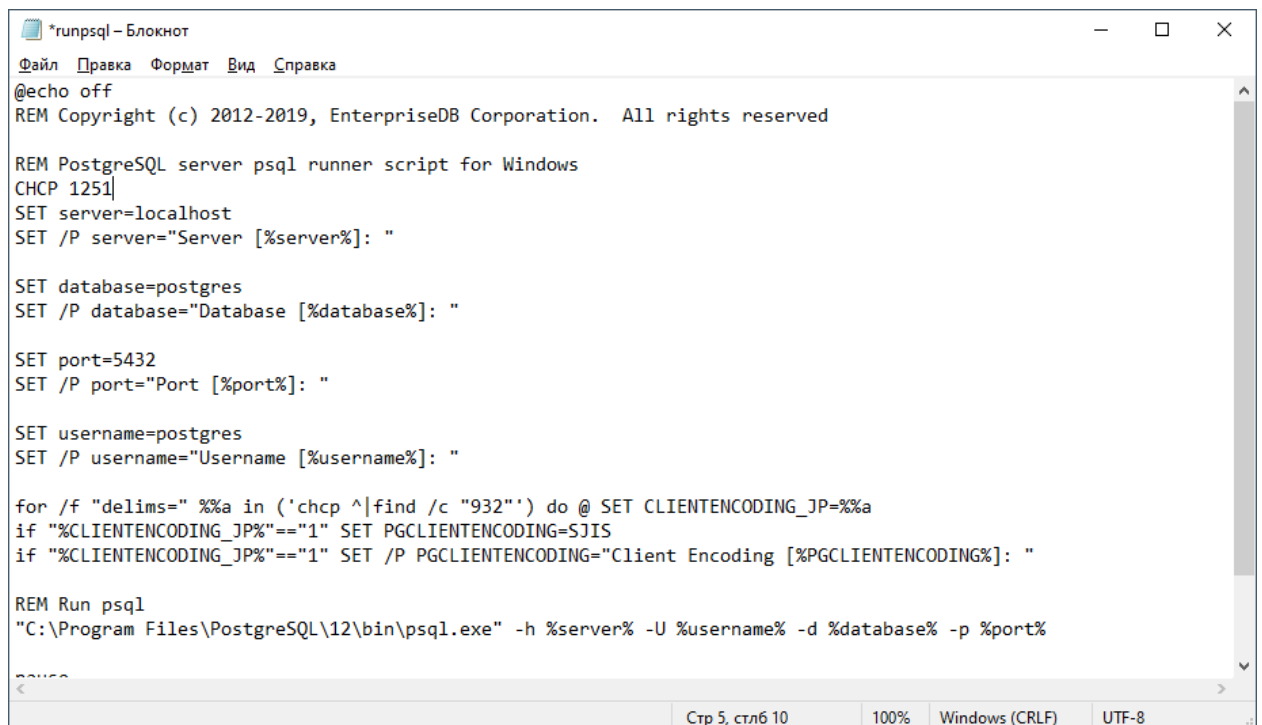
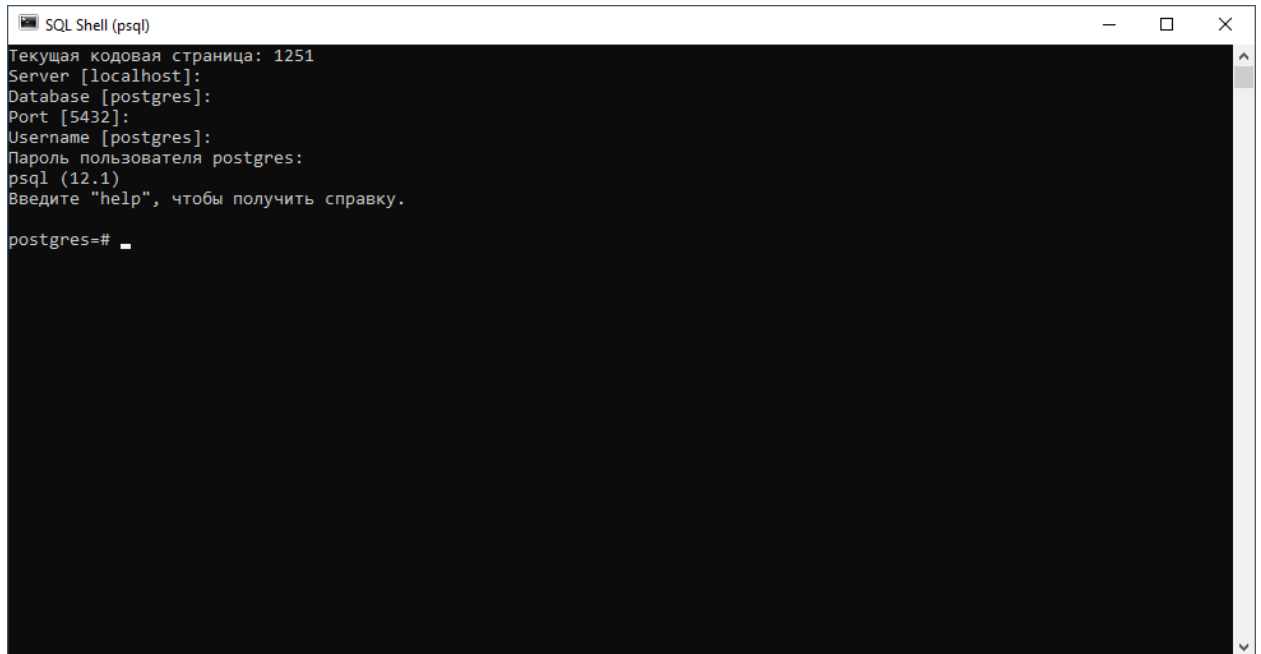


Рис. 8 Исправление скрипта запуска psql

Теперь при запуске psql предупреждение и соответствующие проблемы более не будут возникать. Обратите внимание, что psql предлагает параметры по умолчанию, приводя их в квадратных скобках. В данный момент мы хотим подключиться со всеми параметрами по умолчанию, для этого можно просто не вводить их, а пропускать нажатием Enter.



```
SQL Shell (psql)
Текущая кодовая страница: 1251
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Пароль пользователя postgres:
psql (12.1)
Введите "help", чтобы получить справку.
postgres=#
```

Рис. 9 Запуск psql

Создание и настройка базы данных

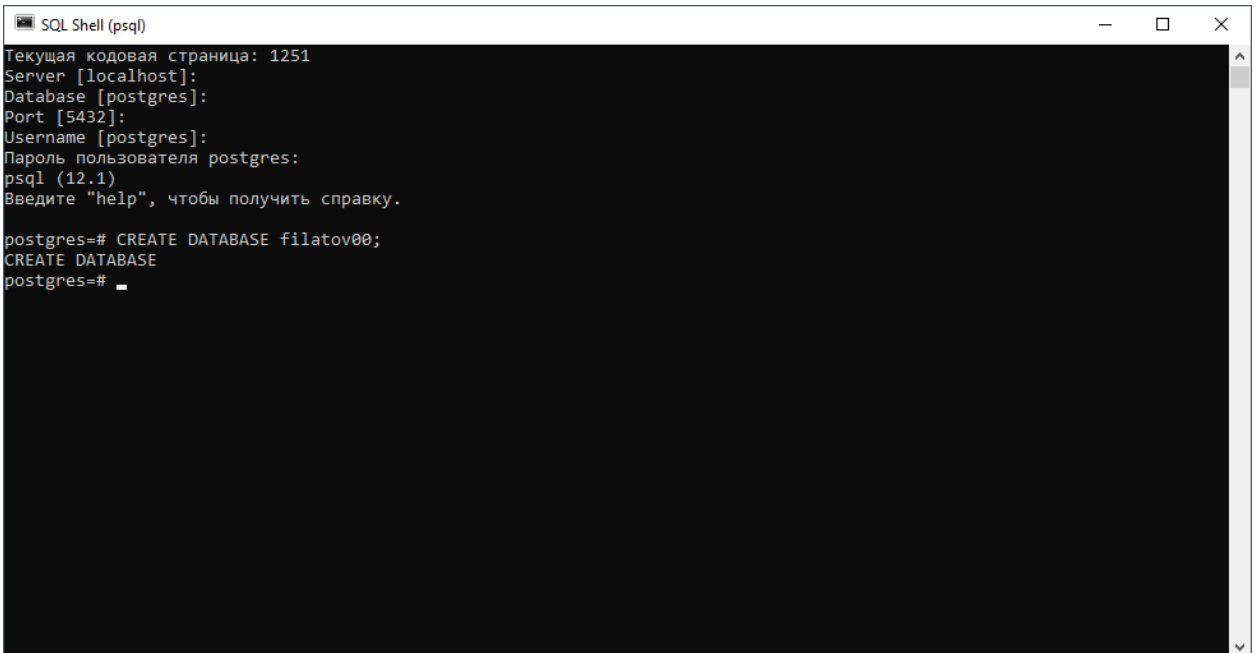
Создайте базу данных с помощью команды `CREATE DATABASE`, которую назовите в формате `ivanovXX`, где `ivanov` – фамилия студента, а `XX` – двузначный номер студента в списке группы.

Работа непосредственно с сущностями базы данных осуществляется с помощью SQL-команд на [языке SQL](#).

SQL-запросы всегда оканчиваются символом `;`

Пример SQL-запроса для создания базы данных:


```
CREATE DATABASE filatov00;
```



```
SQL Shell (psql)
Текущая кодовая страница: 1251
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Пароль пользователя postgres:
psql (12.1)
Введите "help", чтобы получить справку.

postgres=# CREATE DATABASE filatov00;
CREATE DATABASE
postgres=#
```

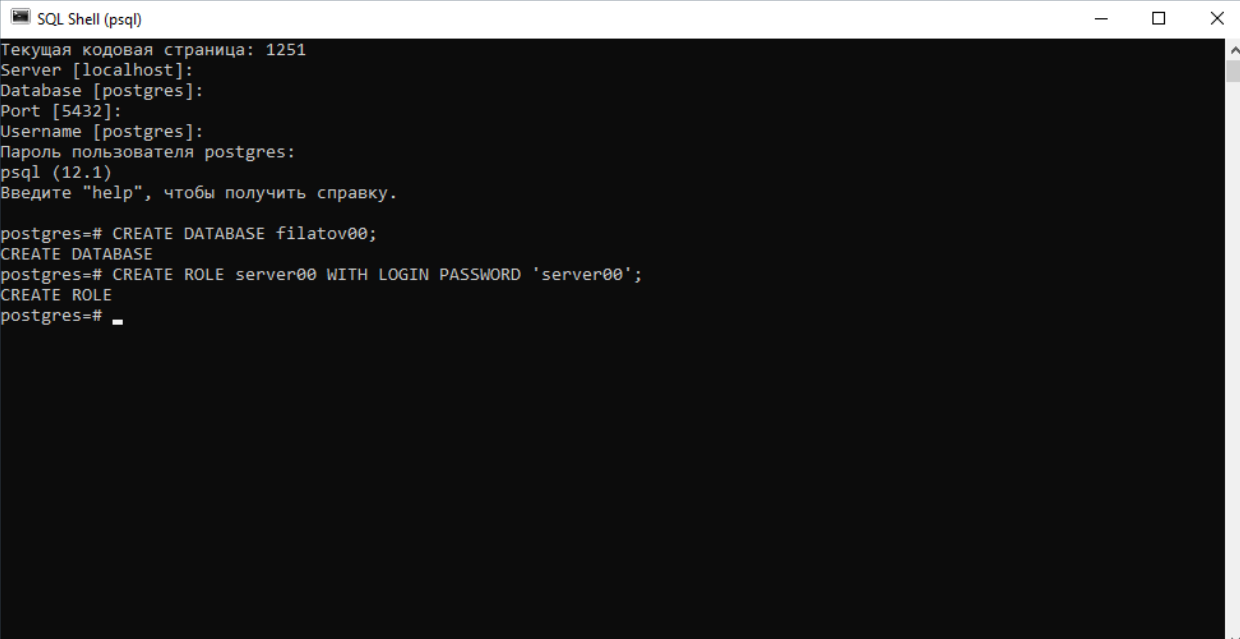
Рис. 10 Выполнение SQL-запроса на создание базы данных

Создайте роль с помощью команды `CREATE ROLE`, которая будет использоваться для доступа и работы с создаваемой базой данных. Назовите ее в формате `serverXX`, где `XX` – двузначный номер студента в списке группы. Для простоты работы укажите атрибут `LOGIN`, который позволит выполнять вход от имени этой роли.

Также при создании роли задается пароль, который на момент обучения для простоты рекомендуется сделать равным имени роли.

Пример SQL-запроса для создания роли:

```
CREATE ROLE server00 WITH LOGIN PASSWORD 'server00';
```



```
SQL Shell (psql)
Текущая кодовая страница: 1251
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Пароль пользователя postgres:
psql (12.1)
Введите "help", чтобы получить справку.

postgres=# CREATE DATABASE filatov00;
CREATE DATABASE
postgres=# CREATE ROLE server00 WITH LOGIN PASSWORD 'server00';
CREATE ROLE
postgres=#
```

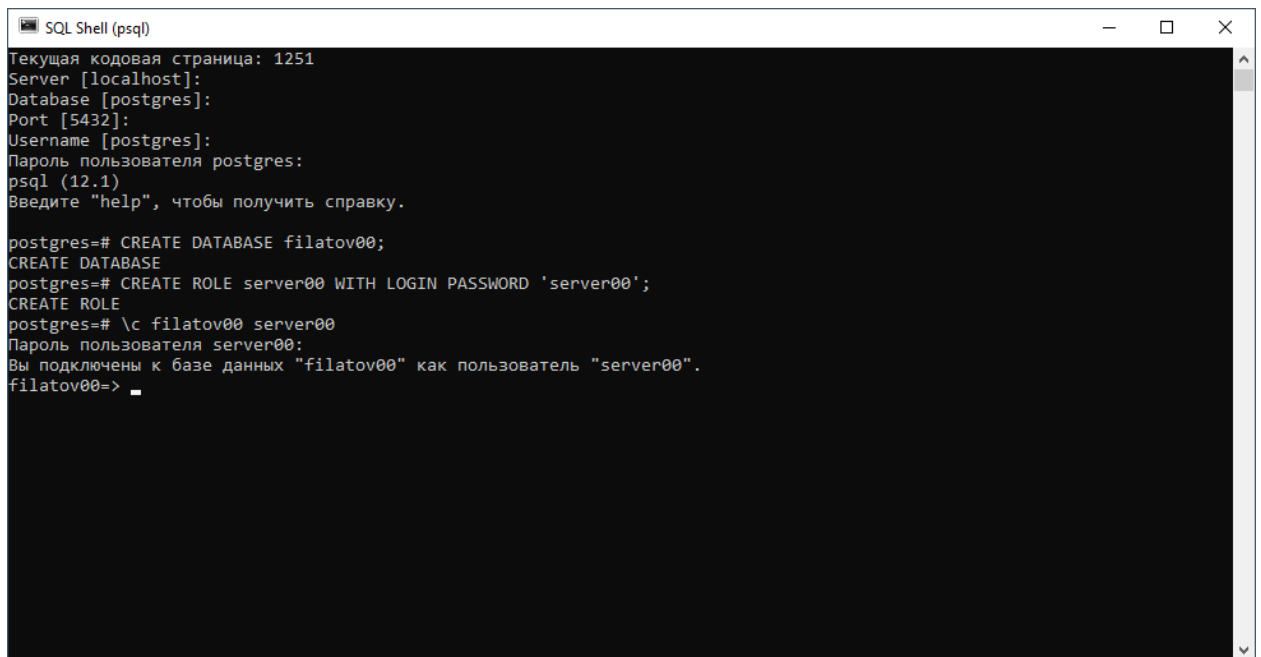
Рис. 11 Выполнение SQL-запроса на создание роли

Далее выполните подключение к созданной базе данных от имени созданной роли с помощью мета-команды \с.

Мета-команда не является SQL-запросом, поэтому оканчивать ее знаком ; не нужно.

Пример команды для переподключения к серверу:

```
\с filatov00 server00
```

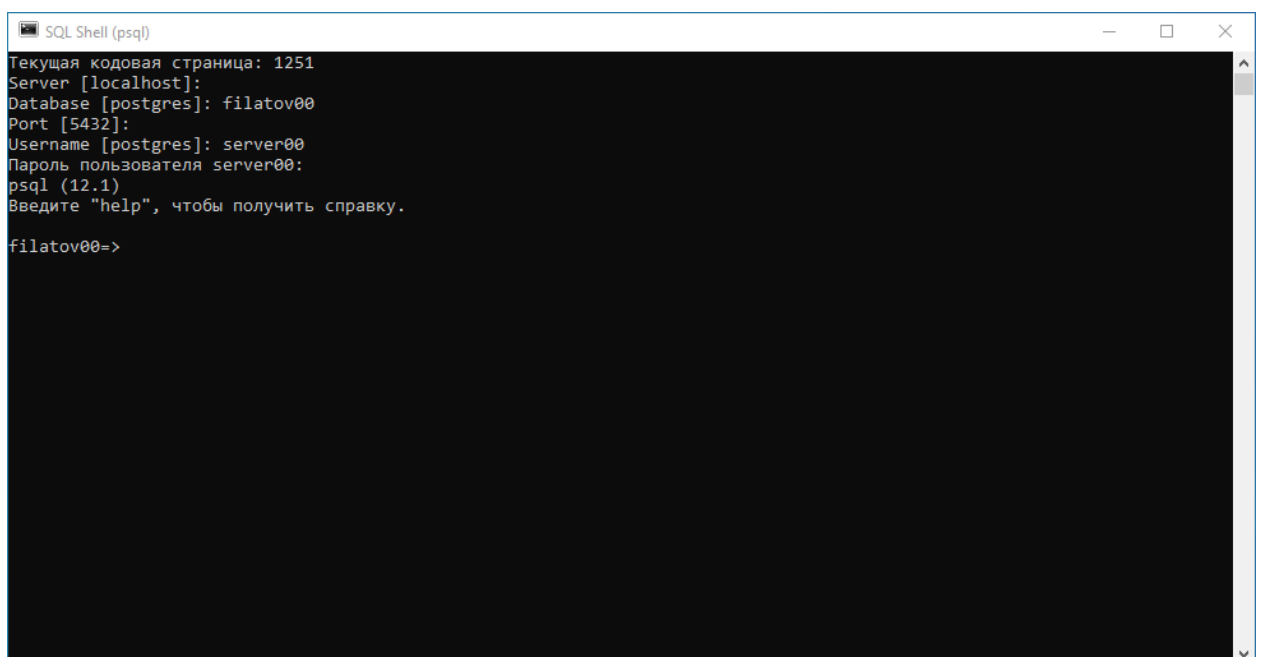


```
SQL Shell (psql)
Текущая кодовая страница: 1251
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Пароль пользователя postgres:
psql (12.1)
Введите "help", чтобы получить справку.

postgres=# CREATE DATABASE filatov00;
CREATE DATABASE
postgres=# CREATE ROLE server00 WITH LOGIN PASSWORD 'server00';
CREATE ROLE
postgres=# \c filatov00 server00
Пароль пользователя server00:
Вы подключены к базе данных "filatov00" как пользователь "server00".
filatov00=>
```

Рис. 12 Выполнение команды для переподключения к базе данных

Теперь при запуске psql можно сразу выполнять подключение к новой базе данных от имени новой роли.



```
SQL Shell (psql)
Текущая кодовая страница: 1251
Server [localhost]:
Database [postgres]: filatov00
Port [5432]:
Username [postgres]: server00
Пароль пользователя server00:
psql (12.1)
Введите "help", чтобы получить справку.

filatov00=>
```

Рис. 13 Выполнение подключения к базе данных

Определение сущностей базы данных

Базы данных в основном состоят из таблиц, соответствующих сущностям в реальном мире (в вашей задаче).

Пример: «Таблица автомобилей», «Таблица пользователей», «Таблица книг», «Таблица заказов».

В таблицах определены «столбцы» – атрибуты, имеющие определенный тип, отражающие свойства, которыми обладает сущность. Обычно таблицы имеют специальное свойство – **ключ**, значения которого уникальны и позволяют использовать его как идентификатор конкретного экземпляра сущности (строки).

Пример: «**Номер**, Марка, Цвет, Владелец», «**Номер**, Имя, Дата регистрации», «**Номер**, Наименование, Автор, Год издания», «**Номер**, Номер заказчика, Сумма».

Таблицы содержат в себе строки, определяющие конкретные экземпляры сущности.

Пример: «9012, "Tesla", "Черный", "Петров И. И."», «77674, "Иван", 4.01.2006», «1984, "1984", "Джордж Оруэлл", 2017», «66, 425, 999.99»

Как и сущности в реальном мире (в вашей задаче), таблицы могут быть связаны между собой. Реляционная база данных определяет 3 типа связей:

- Один к одному 1–1

«У кошки может быть не более одного хвоста (или не быть вовсе)». Данный тип связи редко встречается и должен быть использован только если нам действительно нужен «хвост» как отдельная сущность. Данная связь реализуется с помощью внешнего ключа, который у каждой строки будет равен ключу соответствующей записи в основной таблице.

Пример: «Хвост 11432 принадлежит коту 11432» – проще говоря, у кота и хвоста будет одинаковый идентификационный номер (ключ).

- Один к многим 1–N

«У пользователя может быть много заказов (или не быть вовсе)». Самый часто встречаемый тип связи, при которой некоторые атрибуты (поля) одной сущности (таблицы) зависят от конкретного экземпляра другой сущности (строки). Данная связь реализуется путем определения у дочерней сущности (таблицы) атрибута (поля), указывающего на конкретный экземпляр родительской другой сущности (строки). Такой атрибут называется внешним ключом.

Пример: «Заказ 1142 совершен пользователем 33; Заказ 1143 совершен пользователем 33». В реляционных базах данных данный тип связи реализуется именно так, а не путем определения в родительской таблице **массива** заказов – такой подход имеет свои преимущества.

- Многие ко многим N–M

«У пользователя может быть много любимых фильмов (или не быть вовсе). Фильм может быть любимым у многих пользователей (или ни у кого)». При данном типе связи конкретный экземпляр одной сущности (строка) связан с некоторыми конкретными экземплярами другой сущности (строками) и наоборот. Данная связь реализуется путем создания третьей таблицы, которая будет содержать в себе лишь 2 внешних ключа, указывающих на соответствующие строки в обеих таблицах.

Пример: «Пользователю 11 нравится фильм 45688; Пользователю 15 нравится фильм 77509; Фильм 4095 нравится пользователю 22».

Теперь нужно придумать, что же конкретно будет храниться в вашей базе данных и как. Проектирование диаграммы базы данных рекомендуется производить в draw.io. В среде draw.io следует использовать элементы из

группы «Entity Relation»: «Table» в качестве основы таблицы и элементы «Row 3» и «Row 6» для добавления первичного ключа и обычного поля соответственно.

Задача, рассматриваемая в примере: «В компании работают сотрудники и выполняются задачи. Каждый сотрудник может работать над несколькими задачами. Несколько сотрудников могут работать над одной задачей. О сотрудниках должно быть известно: имя, рост, вес, возраст. О задачах должно быть известно: название, сложность, приоритет, отметка о выполнении».

Проанализировав поставленную задачу и выделив сущности можно определить, что база данных в примере будет состоять из двух таблиц: персоны (person) и задачи (task). Обратите внимание, что таблица отображает конкретную сущность, так что принято давать им имена в единственном числе, несмотря на то, что в таблице задач будет храниться множество задач. Также необходима вспомогательная таблица для хранения фактов прикрепления задач к сотрудникам (persontasks). С помощью этой таблицы будет реализована связь «многие ко многим».

Обычно вспомогательная таблица опускается и заменяется связью «многие ко многим» в соответствии с нотацией диаграммы, поскольку не отображает какую-либо сущность, однако в данной работе она должна быть отображена на диаграмме схемы базы данных.

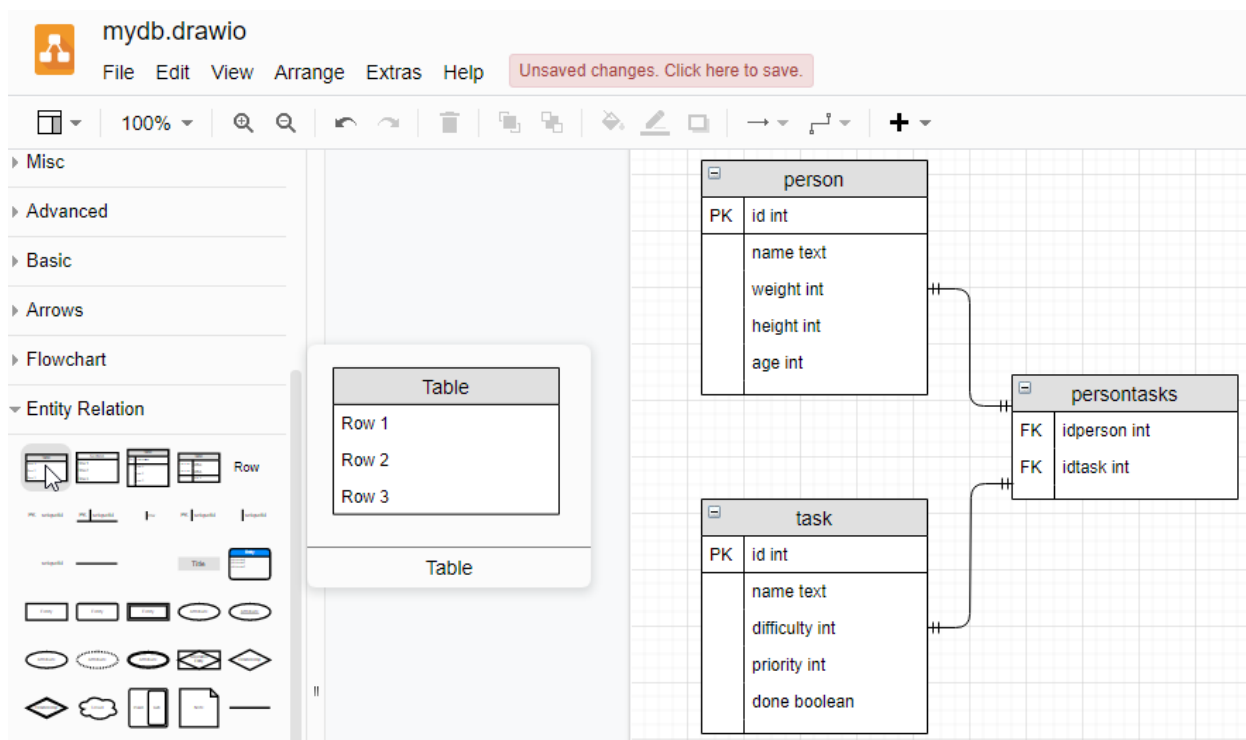


Рис. 14 Проектирование базы данных

Теперь опишите свою задачу **аналогичной сложности** и спроектируйте свою базу данных, используя draw.io. В вашей базе данных должно быть не менее трех таблиц, одна из которых должна реализовывать связь «один ко многим» или «многие ко многим». Словесное описание задачи и схематическое изображение базы данных должны быть добавлены в отчет.

База данных спроектирована, теперь нужно написать SQL-запросы для создания необходимых сущностей. В данном примере таблицы Person и Task содержат первичный ключ – уникальный идентификационный номер. Для более удобного добавления записей в эти таблицы, следует [создать последовательность](#) – генератор чисел, который мы будем использовать лишь для того, чтобы он отсчитывал номера за нас, каждый раз выдавая новое число. Для такой цели никаких дополнительных параметров при создании последовательности указывать не нужно.

Последовательность – самостоятельная сущность, никак не зависящая от какой-либо таблицы. Последовательность может быть использована для любых целей.

Пример SQL-запросов для создания последовательностей:

```
CREATE SEQUENCE seq_personid;  
CREATE SEQUENCE seq_taskid;
```

Далее необходимо [создать таблицы](#), которые будут хранить данные в соответствии с задачей (в примере – таблицы person и task). Рекомендуется сразу создавать таблицу со всеми столбцами так, чтобы позже её не пришлось редактировать, что должно получиться, так как мы все спроектировали. После названия таблицы в скобках указываются имена атрибутов, их тип и прочие параметры атрибута. Уникальный ключ обозначается параметром PRIMARY KEY, а с помощью параметра DEFAULT и [функции последовательности](#) nextval укажем, что по умолчанию при добавлении новой строки в эту таблицу это поле будет равно следующему значению из созданной ранее последовательности. Остальные поля вводим в соответствии с диаграммой и [типами данных](#), соответствующими задаче.

Все зарезервированные слова принято писать строчными буквами, а ключевые слова – прописными.

Psql считывает SQL-запрос до тех пор, пока не будет введен символ его окончания – ;. Таким образом, SQL-запрос можно написать в удобной форме, как в примере ниже, а затем ввести полученный текст разом.

Пример SQL-запросов для создания таблиц:

```
CREATE TABLE person  
(
```



```
id INT PRIMARY KEY DEFAULT nextval('seq_personid'),
name TEXT,
weight INT2,
height INT2,
age INT2
);

CREATE TABLE task
(
id INT PRIMARY KEY DEFAULT nextval('seq_taskid'),
name TEXT,
difficulty TEXT,
priority TEXT,
done BOOLEAN NOT NULL DEFAULT false
);
```

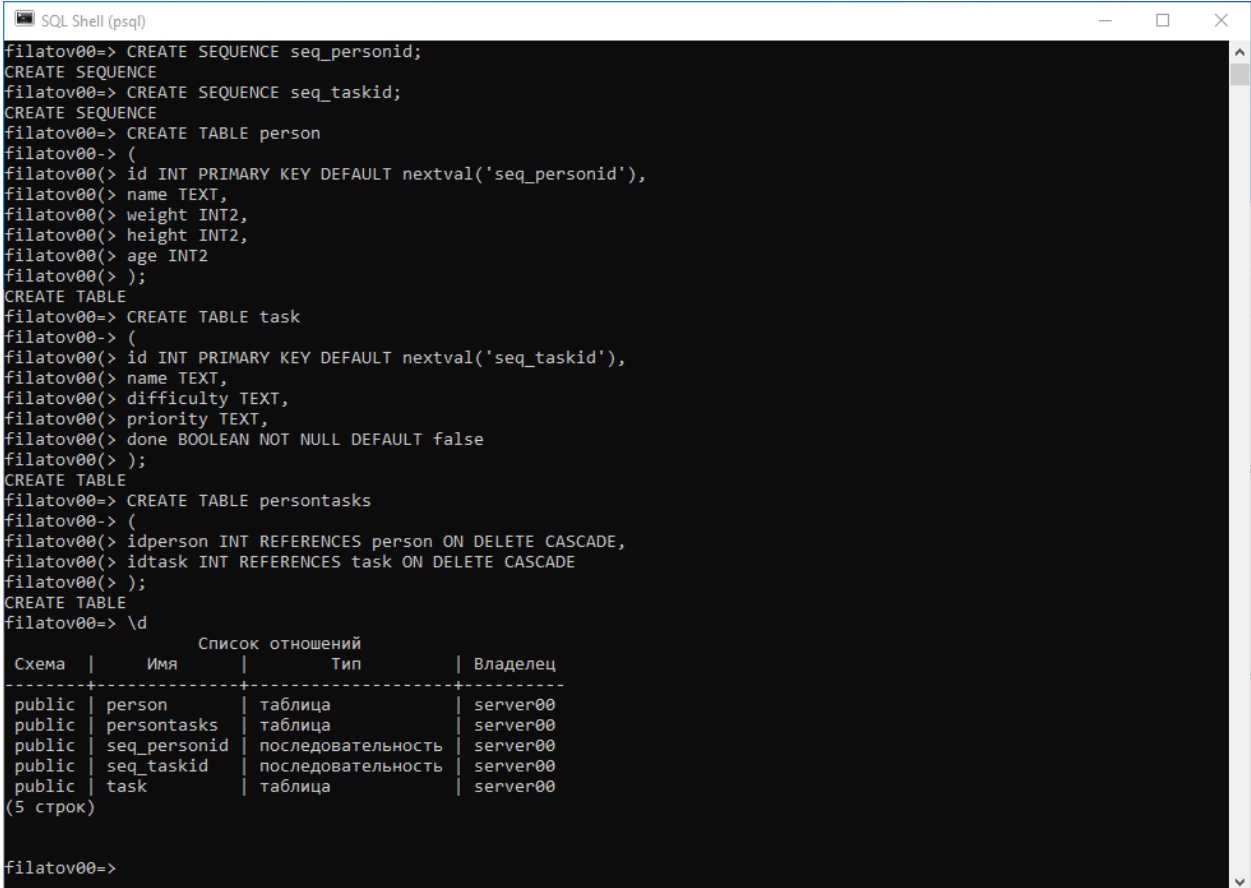
Также необходимо создать вспомогательную таблицу, реализующую связь «многие ко многим». Таблица будет содержать только [внешние ключи](#) на другие таблицы, что объявляется с помощью ключевого слова REFERENCES и указания таблицы, на которую этот внешний ключ будет указывать. Также добавим параметр ON DELETE RESTRICT, который будет требовать удалить строку, содержащую внешний ключ, ссылающийся на удаляемую строку, и параметр ON DELETE CASCADE, который сделает удаление за нас. В примере ниже, если удаляется задача, то база данных потребует удалить все её прикрепления к сотрудникам, а если удаляется сотрудник, то СУБД автоматически проведет удаление всех прикрепленных к нему задач.

Пример SQL-запроса для создания таблицы:

```
CREATE TABLE persontasks
(
```

```
idperson INT REFERENCES person ON DELETE CASCADE,  
idtask INT REFERENCES task ON DELETE RESTRICT  
);
```

Проверить создание сущностей можно с помощью мета-команды \d – эта команда выводит список таблиц, представлений и последовательностей.



```
SQL Shell (psql)  
filatov00=> CREATE SEQUENCE seq_personid;  
CREATE SEQUENCE  
filatov00=> CREATE SEQUENCE seq_taskid;  
CREATE SEQUENCE  
filatov00=> CREATE TABLE person  
filatov00-> (  
filatov00(> id INT PRIMARY KEY DEFAULT nextval('seq_personid'),  
filatov00(> name TEXT,  
filatov00(> weight INT2,  
filatov00(> height INT2,  
filatov00(> age INT2  
filatov00(> );  
CREATE TABLE  
filatov00=> CREATE TABLE task  
filatov00-> (  
filatov00(> id INT PRIMARY KEY DEFAULT nextval('seq_taskid'),  
filatov00(> name TEXT,  
filatov00(> difficulty TEXT,  
filatov00(> priority TEXT,  
filatov00(> done BOOLEAN NOT NULL DEFAULT false  
filatov00(> );  
CREATE TABLE  
filatov00=> CREATE TABLE persontasks  
filatov00-> (  
filatov00(> idperson INT REFERENCES person ON DELETE CASCADE,  
filatov00(> idtask INT REFERENCES task ON DELETE CASCADE  
filatov00(> );  
CREATE TABLE  
filatov00=> \d  
Список отношений  
-----  
Схема | Имя | Тип | Владелец  
-----  
public | person | таблица | server00  
public | persontasks | таблица | server00  
public | seq_personid | последовательность | server00  
public | seq_taskid | последовательность | server00  
public | task | таблица | server00  
(5 строк)  
filatov00=>
```

Рис. 15 Создание сущностей базы данных и последующая проверка

Проверить правильность создания конкретной таблицы можно с помощью той же мета-команды, в таком случае она выведет описание заданного элемента.

```

filatov00=> \d person
                                Таблица "public.person"
Столбец | Тип | Правило сортировки | Допустимость NULL | По умолчанию
-----|-----|-----|-----|-----
id       | integer | | not null | nextval('seq_personid'::regclass)
name     | text    | | | 
weight   | smallint | | | 
height   | smallint | | | 
age       | smallint | | | 
Индексы:
    "person_pkey" PRIMARY KEY, btree (id)
Ссылки извне:
    TABLE "persontasks" CONSTRAINT "persontasks_idperson_fkey" FOREIGN KEY (idperson) REFERENCES person(id) ON DELETE CASCADE
filatov00=> _

```

Рис. 16 Проверка правильности создания таблицы

Теперь напишите текстовый файл, содержащий команды для создания сущностей, соответствующих вашей задаче (не менее двух последовательностей и не менее трех таблиц) и выполните эти команды в psql. При возникновении ошибок, сущности можно удалить из базы данных с помощью команды [DROP](#).

Содержание текстового файла и скриншот с результатами выполнения команд \d для базы данных и каждой таблицы должны быть добавлены в отчет.

Выполнение операций над данными

Как было описано ранее, база данных в основном состоит из таблиц, а таблицы хранят в себе данные в виде строк (записей). Наполним созданные таблицы данными. Для удобства создадим файл sql, в котором напишем все необходимые запросы, а затем укажем программе psql на исполнение запросов из этого файла, вместо того, чтобы вводить все команды в консоль. Этот файл будет выглядеть так же, как и файл с командами для создания сущностей, написанный ранее.

Язык запросов SQL — очень мощный инструмент в руках программиста, позволяющий выполнять [сложные операции](#) над данными, однако в рамках данной работы будут использоваться лишь простейшие запросы для ознакомления с основными принципами работы баз данных.

Прежде чем создавать файл для заполнения базы данных, стоит создать файл для ее очистки для быстрого исправления ошибок. Очистку таблицы можно осуществить с помощью простой sql-команды [TRUNCATE](#). Обратите внимание, что очистка таблиц производится параллельно с очисткой таблицы связей, так как [мы запретили удаление записей](#), на которые ссылаются записи в этой таблице. Также необходимо сбросить счетчик последовательностей, чтобы номера новых записей начинались с единицы, с помощью команды [ALTER SEQUENCE](#).

Файл назовите clear.sql. Пример содержания файла:

```
ALTER SEQUENCE seq_personid RESTART WITH 1;  
ALTER SEQUENCE seq_taskid RESTART WITH 1;  
  
TRUNCATE persontasks, person, task;
```

Вставка данных в таблицу производится sql-командой [INSERT](#). Далее в скобках указывается имя таблицы, в которую будут вставлены данные, перечень параметров, а затем в скобках их значения в соответствующем порядке.

Обратите внимание, что поля, для которых были установлены значения по умолчанию с помощью атрибута DEFAULT, можно не указывать при вставке данных, однако их можно и указать, если это требуется. В своей работе сделайте пару примеров указания необязательных полей. Следует быть осторожным с ручным указанием значения поля, являющимся ключом, для определения которого мы используем последовательность. Не следует самому придумывать значения для этого поля, потому что, когда счетчик последовательности дойдет до придуманного вами значения, запись не сможет быть вставлена, потому что значение ключа должно быть уникальным. В примере ниже значение

вводится вручную равное тому, что было бы введено самой последовательностью. Обратите внимание, что после этого следует запросить следующее число последовательности «впустую», чтобы не получить описанную выше ошибку при следующей вставке.

Далее требуется написать следующие запросы:

- Добавление не менее 10 записей в одну таблицу, 2 из с переопределением значений по умолчанию;
- Добавление не менее 5 записей во вторую таблицу;
- Добавление не менее 5 записей в таблицу, хранящую связи.

Файл назовите insert.sql. Пример содержания файла:

```
INSERT INTO person (name, weight, height, age) VALUES ('Carl', 70,  
170, 21);
```

```
INSERT INTO person (name, weight, height, age) VALUES ('Alex', 68,  
173, 23);
```

```
INSERT INTO person (name, weight, height, age) VALUES ('Barry', 85,  
181, 30);
```

```
INSERT INTO person (name, weight, height, age) VALUES ('Janny', 66,  
170, 20);
```

```
INSERT INTO person (id, name, weight, height, age) VALUES (5, 'Philip',  
82, 180, 45);
```

```
SELECT nextval('seq_personid');
```

```
INSERT INTO person (name, weight, height, age) VALUES ('Tabitha', 60,  
165, 19);
```

```
INSERT INTO person (name, weight, height, age) VALUES ('Velma', 66,  
171, 28);
```

```
INSERT INTO person (name, weight, height, age) VALUES ('Mindy', 55,  
165, 27);
```

```
INSERT INTO person (name, weight, height, age) VALUES ('Gwen', 50, 160, 25);
```

```
INSERT INTO person (name, weight, height, age) VALUES ('Rakim', 70, 180, 25);
```

```
INSERT INTO task (id, name, difficulty, priority) VALUES (1, 'Cleaning', 10, 8);
```

```
SELECT nextval('seq_taskid');
```

```
INSERT INTO task (name, difficulty, priority, done) VALUES ('Fix the computer', 8, 10, false);
```

```
INSERT INTO task (name, difficulty, priority) VALUES ('Prepare the halloween party', 7, 2);
```

```
INSERT INTO task (name, difficulty, priority) VALUES ('Test the employees', 5, 8);
```

```
INSERT INTO task (name, difficulty, priority) VALUES ('Encourage the employees', 7, 6);
```

```
INSERT INTO persontasks (idperson, idtask) VALUES (1,1);
```

```
INSERT INTO persontasks (idperson, idtask) VALUES (2,1);
```

```
INSERT INTO persontasks (idperson, idtask) VALUES (3,1);
```

```
INSERT INTO persontasks (idperson, idtask) VALUES (4,1);
```

```
INSERT INTO persontasks (idperson, idtask) VALUES (5,1);
```

```
INSERT INTO persontasks (idperson, idtask) VALUES (6,1);
```

```
INSERT INTO persontasks (idperson, idtask) VALUES (7,1);
```

```
INSERT INTO persontasks (idperson, idtask) VALUES (8,1);
```

```
INSERT INTO persontasks (idperson, idtask) VALUES (9,1);
```

```
INSERT INTO persontasks (idperson, idtask) VALUES (10,1);
```

```
INSERT INTO persontasks (idperson, idtask) VALUES (3,2);
```

```
INSERT INTO persontasks (idperson, idtask) VALUES (2,2);
```

```
INSERT INTO persontasks (idperson, idtask) VALUES (4,3);
```

```
INSERT INTO persontasks (idperson, idtask) VALUES (6,3);  
INSERT INTO persontasks (idperson, idtask) VALUES (7,3);  
INSERT INTO persontasks (idperson, idtask) VALUES (8,3);  
INSERT INTO persontasks (idperson, idtask) VALUES (9,3);  
INSERT INTO persontasks (idperson, idtask) VALUES (5,4);  
INSERT INTO persontasks (idperson, idtask) VALUES (5,4);
```

Данные нужно не только хранить в базе данных, но и извлекать для чтения. Данные извлекаются из таблиц с помощью SQL-запроса [SELECT](#). Далее в скобках указывается что требуется извлечь (какие столбцы) и в каком порядке. Если нужно извлечь все данные из таблицы, можно поставить символ *, вместо того, чтобы перечислять все столбцы. Далее пунктом FROM указывается, откуда эти данные требуется извлечь (название таблицы), после чего указываются дополнительные пункты в зависимости от задачи. Например, пунктом [ORDER BY](#) указывается столбец, по которому полученные строки будут отсортированы, а пунктом [WHERE](#) определяется условие, по которому полученные строки будут отфильтрованы.

Далее требуется написать следующие запросы:

- Чтение всех данных из одной таблицы так, как они записаны в таблице;
- Чтение и сортировка определенных столбцов данных из второй таблицы;
- Чтение данных с условием из третьей таблицы. Условие должно быть составным – содержать несколько логических операторов (аналогично примеру).

Файл назовите select.sql. Пример содержания файла:

```
SELECT * FROM task;  
SELECT (name, height, weight) FROM person ORDER BY height;
```

```
SELECT * FROM persontasks WHERE idperson BETWEEN 1 AND 5 OR  
idtask = 1;
```

Данные, хранящиеся в таблицах, также можно изменять или удалять.

Изменение (обновление) данных производится с помощью SQL-запроса [UPDATE](#). В команде указывается таблица, в которой требуется произвести изменения (обновления), затем пунктом SET определяются необходимые изменения, после чего пунктом WHERE указываются условия, определяющие какие записи требуется изменить.

Удаление записей производится с помощью SQL-запроса [DELETE](#). Пунктом FROM указывается, в какой таблице требуется удалить записи. затем командой WHERE указываются условия, определяющие какие записи требуется удалить.

Далее требуется написать следующие запросы:

- Изменение не менее 2 записей одной таблицы так, чтобы изменился результат второго запроса из файла выше (запрос с сортировкой строк);
- Удаление другой таблицы записи, на которую ссылается какая-либо запись из таблицы связей с параметром ON DELETE RESTRICT. Этот запрос должен провалиться, так как мы [запретили подобное удаление](#);
- Удаление записи из таблицы связей, которая ссылается на ту, что мы пытались удалить;
- Повторение второго пункта, теперь он должен выполняться без ошибок.
- Удаление другой таблицы записи, на которую ссылается какая-либо запись из таблицы связей с параметром ON DELETE CASCADE.

Файл назовите update.sql. Пример содержания файла:

```
UPDATE person SET weight = 65, height = 175 WHERE name = 'Alex';  
UPDATE person SET weight = 88, height = 179 WHERE name = 'Barry';
```

```
DELETE FROM task WHERE id = 4;
```

```
DELETE FROM persontasks WHERE idtask = 4;
```

```
DELETE FROM task WHERE id = 4;
```

```
DELETE FROM person WHERE id = 4;
```

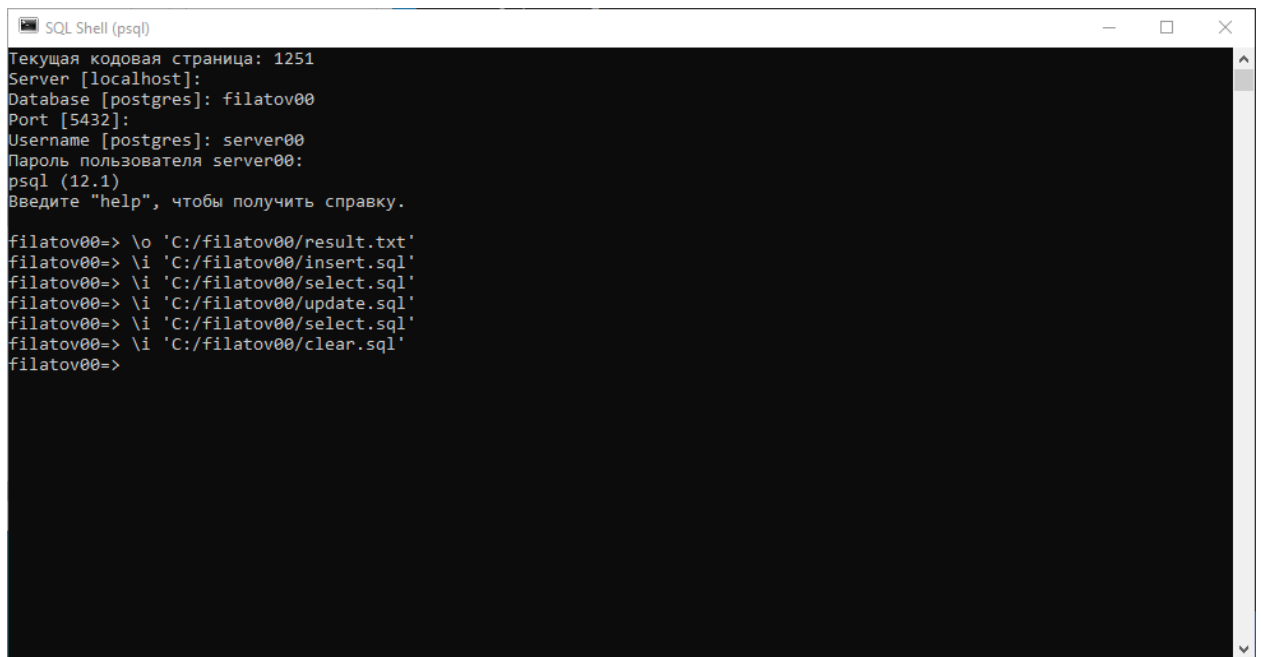
Были написаны 4 файла:

- Очистка базы данных – clear.sql;
- Добавление данных – insert.sql;
- Получение данных – select.sql;
- Изменение данных – update.sql;

Теперь необходимо выполнить написанные запросы. Для этого воспользуемся мета-командой `psql \i` или `\include`, которая считывает и исполняет весь файл. После команды должен быть указан путь к файлу в одинарных кавычках. Вместо того, чтобы вводить его вручную, можно перетащить его мышью в окно терминала, однако после этого нужно будет исправить все обратные косые черты (`\`) на косые черты (`/`).

Исполните файлы в следующем порядке: добавление данных, получение данных, изменение данных, получение данных, очистка базы данных. Исправьте запросы, если возникли ошибки.

Теперь, когда все запросы выполняются без ошибок, для удобства составления отчета следует перенаправить запись результатов выполнения команд в текстовый файл с помощью мета-команды `\o` или `\out` с указанием пути файла, создавать его предварительно не нужно.



```
SQL Shell (psql)
Текущая кодовая страница: 1251
Server [localhost]:
Database [postgres]: filatov00
Port [5432]:
Username [postgres]: server00
Пароль пользователя server00:
psql (12.1)
Введите "help", чтобы получить справку.

filatov00=> \o 'C:/filatov00/result.txt'
filatov00=> \i 'C:/filatov00/insert.sql'
filatov00=> \i 'C:/filatov00/select.sql'
filatov00=> \i 'C:/filatov00/update.sql'
filatov00=> \i 'C:/filatov00/select.sql'
filatov00=> \i 'C:/filatov00/clear.sql'
filatov00=>
```

Рис. 17 Выполнение SQL-команд из файлов

Исполните команды в таком же порядке после настройки вывода результата команд в файл. Проверьте корректность записанных в файл данных. Обратите внимание, что там только результаты запросов, сами запросы в файл не записывались. Содержание файла должно быть добавлено в отчет.

Выполнение работы окончено, приступайте к оформлению отчета. Отчет должен состоять из теоретической и практической частей. Практическая часть должна содержать описание всех шагов выполнения работы, выделенных зеленым цветом.