

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ОТЧЕТ  
по лабораторной работе № 6  
на тему  
СОЗДАНИЕ ПРИЛОЖЕНИЯ ДЛЯ БАЗЫ ДАННЫХ. «ШКОЛА»

Студент:

А.Н. Климович

Преподаватель:

Д.В. Куприянова

МИНСК 2024

## **1 ЦЕЛЬ РАБОТЫ**

Создание прикладной программы для работы с базой данных и выполняющей заданные транзакции.

## **2 ИСХОДНЫЕ ДАННЫЕ**

### **2.1 Системные требования**

Данная программа разрабатывалась под операционной системой Windows 10.

Для запуска программы необходимо иметь следующие минимальные системные требования:

- процессор: не менее 1 ГГц или SoC;
- ОЗУ: 1 ГБ для 32-разрядной системы или 2 ГБ для 64-разрядной системы;
- место на жестком диске: 16 ГБ для 32-разрядной ОС или 20 ГБ для 64-разрядной ОС;
- видеоадаптер: DirectX 9 или более поздняя версия с драйвером WDDM 1.0;
- экран: 800x600.

Исходные файлы разработанного приложения занимают не более 1 МБ на жестком диске.

### **2.2 Среда программирования**

Visual Studio Code – это легкая, гибкая и расширяемая среда разработки, разработанная компанией Microsoft.

Она предлагает широкий спектр функций, таких как подсветка синтаксиса, автодополнение кода, отладка, совместная работа и интеграция с Git.

Благодаря своей легкости и скорости работы, Visual Studio Code является отличным выбором для разработчиков, работающих с Python и другими языками программирования.

Именно по этим причинам эта среда была выбрана для разработки данного приложения.

### **2.3 Язык программирования**

Python – язык программирования общего назначения, который используют во многих областях IT-индустрии.

Синтаксическая особенность Python — выделение блоков кода отступами, что значительно упрощает зрительное восприятие программ, написанных на этом языке.

Программы, написанные на языке программирования Python, не переводятся в машинный код, а сразу выполняются программой-интерпретатором. Это позволяет запускать код на любой платформе с установленным заранее интерпретатором.

Python — это язык, созданный согласно парадигме объектно-ориентированного программирования (ООП). В ней основными являются понятия объекта и класса. Классы – это специальные типы данных, объекты – экземпляры классов. То есть любое значение является объектом конкретного класса. В Python вы можете не только использовать уже существующие классы, но и создавать свои собственные.

В отличие от С-подобных языков программирования, в Python переменные связываются с типом в момент присваивания в них конкретных значений.

По этим причинам именно этот язык был выбран для выполнения данной лабораторной работы.

## 2.4 Модуль GUI

Для разработки графического интерфейса программы было решено использовать готовый модуль (библиотеку).

Многие программы на сегодняшний день используют графический интерфейс, который более интуитивен и удобен для пользователя, чем консоль. И с помощью языка программирования Python также можно создавать графические программы. Для этого в Python по умолчанию применяется специальный тулkit – набор компонентов, который называется tkinter. Тулkit tkinter доступен в виде отдельного встроенного модуля, который содержит все необходимые графические компоненты - кнопки, текстовые поля и т.д.

По сути Tkinter представляет интерфейс в Python для графической библиотеки Tk (Собственно само название "Tkinter" является сокращением "Tk interface"). Первоначально данная библиотека разрабатывалась для языка Tcl - ее создал в 1988 году Джон Остерхаут (John Ousterhout), профессор computer science из Беркли для создания графических приложений для своего языка Tcl. Но впоследствии Tk была адаптирована для широкого ряда динамических языков, в частности, для Ruby, Perl и естественно для языка Python (в 1994 году). И на сегодняшний день и библиотека Tk, и сам тулkit tkinter доступны для большинства операционных систем, в том числе для Mac OS, Linux и Windows.

Преимущества Tkinter:

- Данный тулkit по умолчанию включен в стандартную библиотеку языка Python в виде отдельного модуля, поэтому не потребуется что-то дополнительно устанавливать;
- Tkinter – кроссплатформенный, один и тот же код будет работать одинаково на разных платформах (Mac OS, Linux и Windows);

- Tkinter легко изучать. Сам тулкит, хотя и содержит некоторый готовый код, виджеты и графические элементы, но при этом довольно лаконичен и прост.
- Tk распространяется по BSD-лицензии, поэтому библиотека может быть использована как в опенсорсных проектах, так и в коммерческих наработках.

## 2.5 Модуль PostgreSQL

Во время разработки приложений нужно подключить и использовать базу данных для хранения информации. Так как работа производится с базой данных PostgreSQL, нужно узнать, как работать в Python именно с ней. Для этого существует множество модулей, например:

- psycopg2;
- py-postgresql;
- pg8000.

Для выполнения данной лабораторной работы был выбран именно модуль psycopg2 по таким причинам:

- Распространенность. Psycopg2 использует большинство фреймворков Python;
- Поддержка. Psycopg2 активно развивается и поддерживает основные версии Python;
- Многопоточность. Psycopg2 позволяет нескольким потокам поддерживать одно и то же соединение.

Для начала работы с модулем достаточно установить пакет при помощи pip:

```
pip install psycopg2-binary.
```

## 3 РУКОВОДСТВО ПОЛЬЗОВАНИЯ

### 3.1 Запуск приложения

Ввести в терминале следующую команду:

```
python main.py
```

### 3.2 Подключение к серверу

После запуска откроется окно для подключения к серверу, изображенное на рисунке 3.1.



Рисунок 3.1 – Окно для подключения к серверу

Теперь необходимо ввести пароль для подключения к серверу (пароль по умолчанию: 1234), нажать ОК.

### 3.3 Работа в приложении

#### 3.3.1 Виджет Text

После подключения к серверу открывается главное окно программы, показанное на рисунке 3.2.

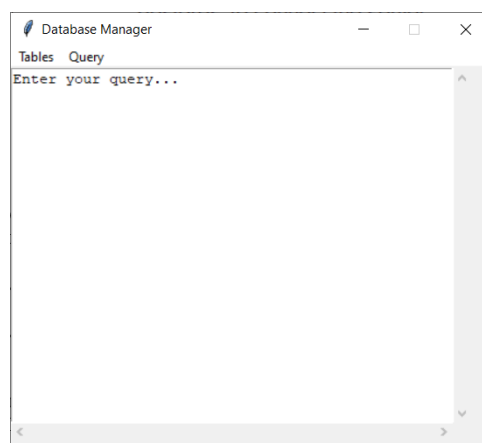


Рисунок 3.2 – Главное окно программы

В виджете **Text**, где по умолчанию выведена строка “*Enter your query...*” можно ввести свой SQL-запрос. После ввода необходимо в **Menubar** выбрать пункт **Query** и нажать на **Execute**.

В итоге, если запрос выполнится успешно, то вы получите таблицу результата в отдельном окне. Иначе появится окно об ошибке.

**Пример 3.1** – Вывести таблицу student с помощью виджета **Text**.

Ввод запроса для примера 3.1 показан на рисунке 3.3.

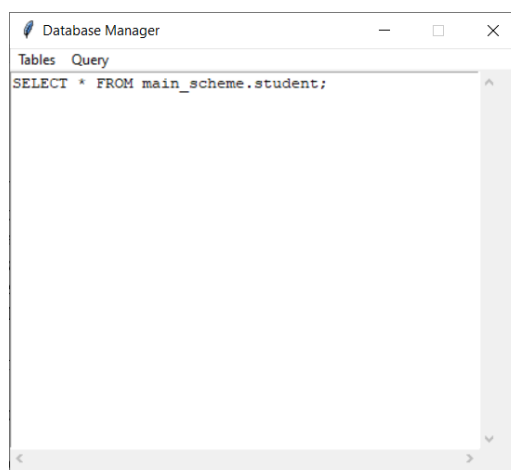


Рисунок 3.3 – Запрос для примера 3.3 в приложении

Результаты запроса для примера 3.1 показаны на рисунке 3.4.

The screenshot shows a window titled "Query Result" displaying a table with three columns: "id", "first\_name", and "last\_name". The table contains 17 rows of student data.

	id	first_name	last_name
1		Иван	Иванов
2		Петр	Петров
3		Анна	Сидорова
4		Мария	Козлова
5		Сергей	Васильев
6		Елена	Никитина
7		Алексей	Григорьев
8		Ольга	Павлова
9		Дмитрий	Федоров
10		Наталья	Иванова
11		Ирина	Петрова
12		Андрей	Сидоров
13		Виктория	Смирнова
14		Артем	Кузнецов
15		Татьяна	Морозова
16		Григорий	Новиков
17		Василиса	Зайцева

Рисунок 3.4 – Результаты выполнения запроса для примера 3.1

### 3.3.2 Меню. Вкладка Query

Для выполнения SQL-запроса из скрипта, подготовленного заранее, можно воспользоваться вкладкой **Query**. Для этого необходимо выбрать команду **Execute from file**, откроется проводник, где вы можете выбрать SQL-скрипт, который можно выполнить.

**Пример 3.2** – Выполнить скрипт create.sql для создания таблиц.

Выбор команды выполнения SQL-запроса из файла для представлен на рисунке 3.5.

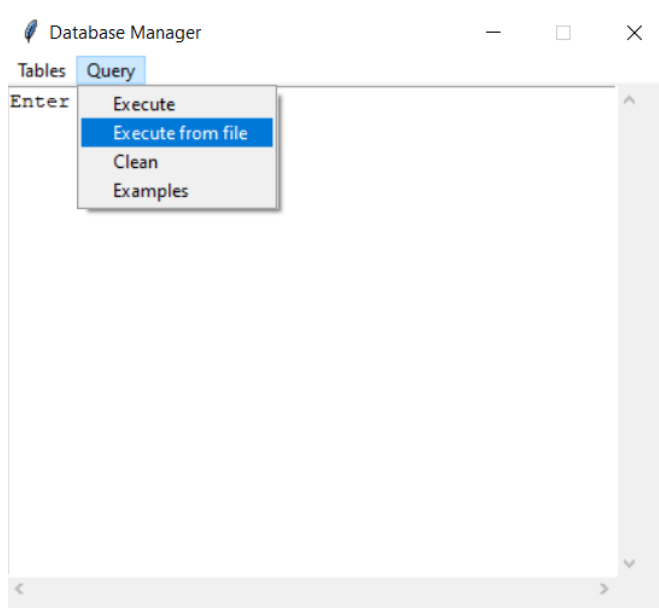


Рисунок 3.5 – Выбор команды выполнения SQL-запроса из файла

Выбор файла, который содержит SQL-запрос, представлен на рисунке 3.6.

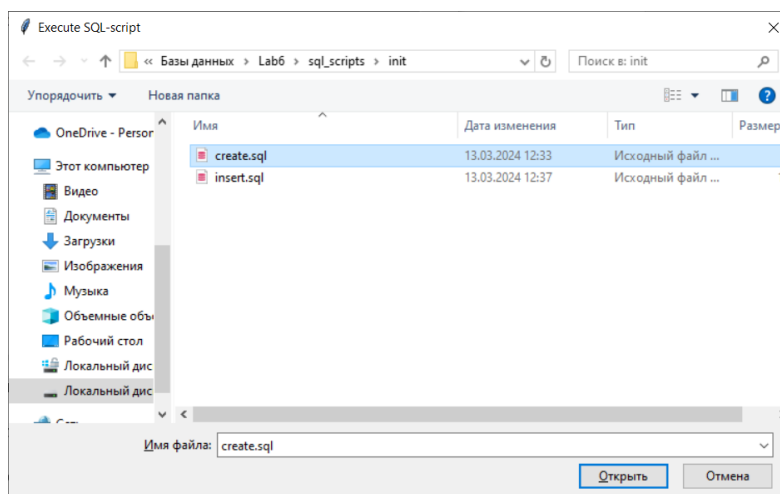


Рисунок 3.6 – Выбор SQL-скрипта для выполнения

После выполнения запроса появится информационное окно, показанное на рисунке 3.7. Далее можно воспользоваться, например, pgAdmin4, чтобы удостовериться, что таблицы действительно были созданы.

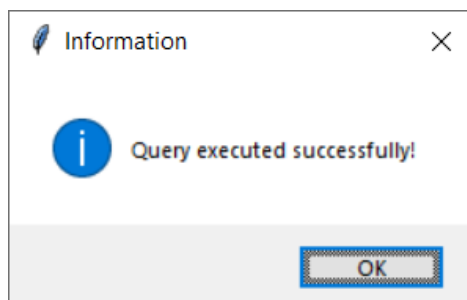


Рисунок 3.7 – Информационное окно о успешном выполнении запроса

Команда **Clean** позволяет очистить введенный запрос в виджете **Text**.

Команда **Examples** открывает новое окно со списком готовых SQL-скриптов (см. рисунок 3.8).

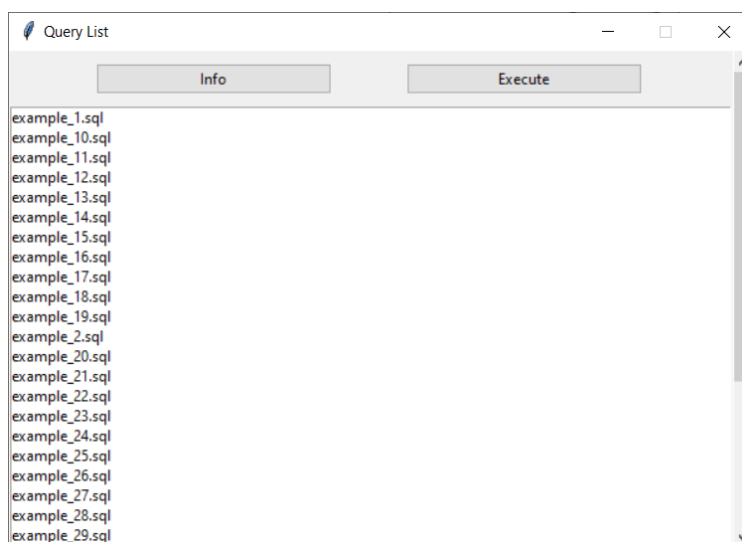


Рисунок 3.8 – Окно Examples с готовыми SQL-запросами для выполнения

Чтобы получить информацию о скрипте, нужно выделить его и нажать на кнопку **Info**.

Для выполнения одного из готовых скриптов, нужно выделить его и нажать на кнопку **Execute**.

**Пример 3.3** – Вывести информацию о скрипте example\_1.sql и выполнить его.

Информация о скрипте example\_1.sql представлена на рисунке 3.9, результаты выполнения этого скрипта – на рисунке 3.10.



```

-- Вывести ФИО учеников, у которых имя начинается
SELECT
    main_scheme.student.first_name,
    main_scheme.student.last_name,
    main_scheme.student.middle_name
FROM main_scheme.student
WHERE first_name LIKE 'А%';

```

Рисунок 3.9 – Вывод информации о запросе из файла example\_1.sql

first_name	last_name	middle_name
Анна	Сидорова	Александровна
Алексей	Григорьев	Андреевич
Андрей	Сидоров	Петрович
Артем	Кузнецов	Иванович
Александра	Иванова	Александровна
Антон	Козлов	Павлович
Анастасия	Козлова	Павловна

Рисунок 3.10 – Результаты выполнения скрипта example\_1.sql

### 3.3.3 Меню. Вкладка Tables

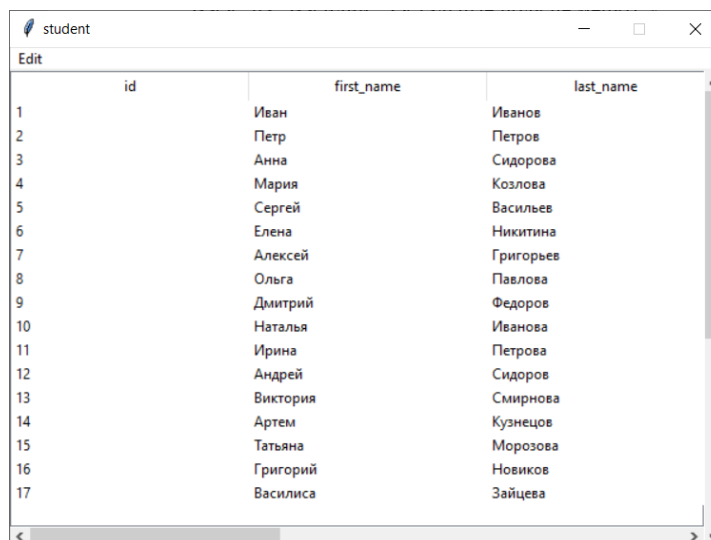
Через вкладку **Tables** можно выбрать таблицу, с которой дальше работать. Откроется новое окно, в котором выведется выбранная таблица.

В открывшемся окне есть вкладка **Edit**, где вы можете выбрать пункт **Insert, Delete, Update Truncate** для вставки, удаления, обновления строки или каскадной очистки всей таблицы соответственно.

**Пример 3.4** – Выбрать таблицу student.

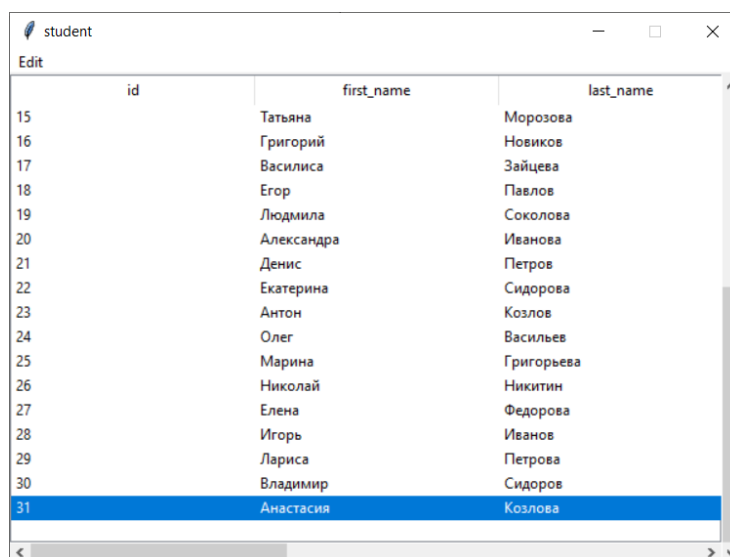
1. Выполнить запрос **INSERT** и добавить ученика со следующими параметрами: first\_name – “Вася”, last\_name – “Пупкин”, middle\_name – “Иванович”, passport\_id – “AC9876543DC21”, date\_of\_birth – “15.04.2004”, class\_id – “1”, residential\_address\_id – “1”.

2. Выполнить запрос UPDATE и изменить имя добавленного ученика с “Вася” на “Василий”. Остальные поля не менять.
  3. Выполнить запрос DELETE и удалить добавленного ученика.
  4. Выполнить запрос TRUNCATE для очистки таблицы student.
- Исходная таблица student представлена на рисунках 3.11 – 3.12.



	id	first_name	last_name
1		Иван	Иванов
2		Петр	Петров
3		Анна	Сидорова
4		Мария	Козлова
5		Сергей	Васильев
6		Елена	Никитина
7		Алексей	Григорьев
8		Ольга	Павлова
9		Дмитрий	Федоров
10		Наталья	Иванова
11		Ирина	Петрова
12		Андрей	Сидоров
13		Виктория	Смирнова
14		Артем	Кузнецов
15		Татьяна	Морозова
16		Григорий	Новиков
17		Василиса	Зайцева

Рисунок 3.11 – Первая часть исходной таблицы student



	id	first_name	last_name
15		Татьяна	Морозова
16		Григорий	Новиков
17		Василиса	Зайцева
18		Егор	Павлов
19		Людмила	Соколова
20		Александра	Иванова
21		Денис	Петров
22		Екатерина	Сидорова
23		Антон	Козлов
24		Олег	Васильев
25		Марина	Григорьева
26		Николай	Никитин
27		Елена	Федорова
28		Игорь	Иванов
29		Лариса	Петрова
30		Владимир	Сидоров
31		Анастасия	Козлова

Рисунок 3.12 – Вторая часть исходной таблицы student

Заполнение полей для добавления “Васи Пупкина” в таблицу student представлено на рисунке 3.13.

INSERT student

first\_name: Вася

last\_name: Пупкин

middle\_name: Иванович

passport\_id: AC9876543DC21

date\_of\_birth: 15.04.2004

class\_id: 1

residential\_address\_id: 1

INSERT

Рисунок 3.13 – Заполнение полей для добавления “Васи Пупкина” в таблицу student

Результаты выполнения запроса INSERT для примера 3.4 представлены на рисунке 3.14.

id	first_name	last_name
16	Григорий	Новиков
17	Василиса	Зайцева
18	Егор	Павлов
19	Людмила	Соколова
20	Александра	Иванова
21	Денис	Петров
22	Екатерина	Сидорова
23	Антон	Козлов
24	Олег	Васильев
25	Марина	Григорьева
26	Николай	Никитин
27	Елена	Федорова
28	Игорь	Иванов
29	Лариса	Петрова
30	Владимир	Сидоров
31	Анастасия	Козлова
32	Вася	Пупкин

Рисунок 3.14 – Результаты выполнения запроса INSERT для примера 3.4

Заполнение полей для обновления полей “Васи Пупкина” в таблице student представлено на рисунке 3.15.

UPDATE student

id: 32

first\_name: Василий

last\_name:

middle\_name:

passport\_id:

date\_of\_birth:

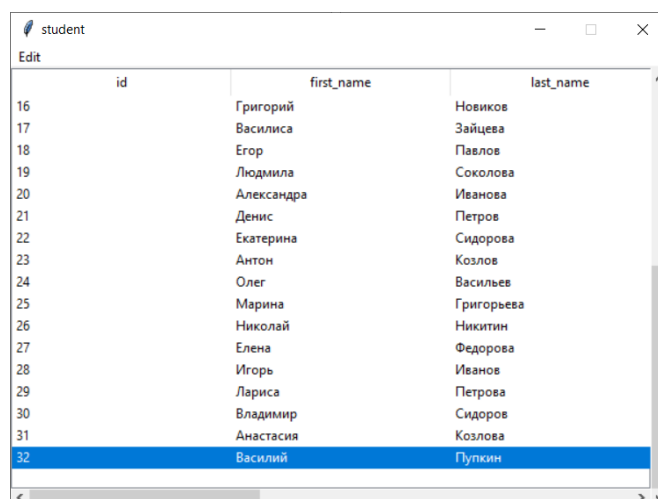
class\_id:

residential\_address\_id:

UPDATE

Рисунок 3.15 – Заполнение полей для обновления полей “Васи Пупкина” в таблице student

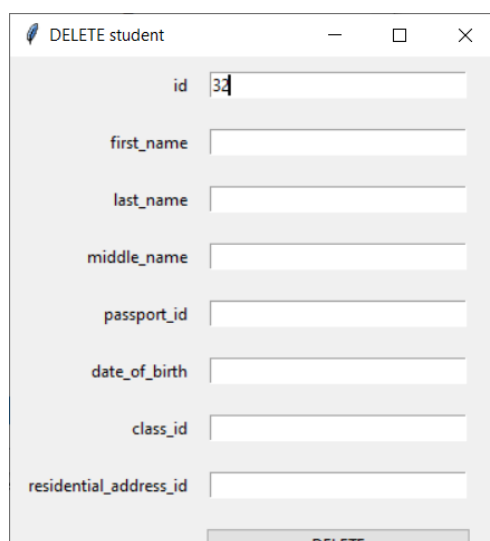
Результаты выполнения запроса UPDATE для примера 3.4 представлены на рисунке 3.16.



id	first_name	last_name
16	Григорий	Новиков
17	Василиса	Зайцева
18	Егор	Павлов
19	Людмила	Соколова
20	Александра	Иванова
21	Денис	Петров
22	Екатерина	Сидорова
23	Антон	Козлов
24	Олег	Васильев
25	Марина	Григорьева
26	Николай	Никитин
27	Елена	Федорова
28	Игорь	Иванов
29	Лариса	Петрова
30	Владимир	Сидоров
31	Анастасия	Козлова
32	Василий	Пупкин

Рисунок 3.16 – Результаты выполнения запроса UPDATE для примера 3.4

Заполнение полей для удаления ученика “Василия Пупкина” в таблице student представлено на рисунке 3.17.

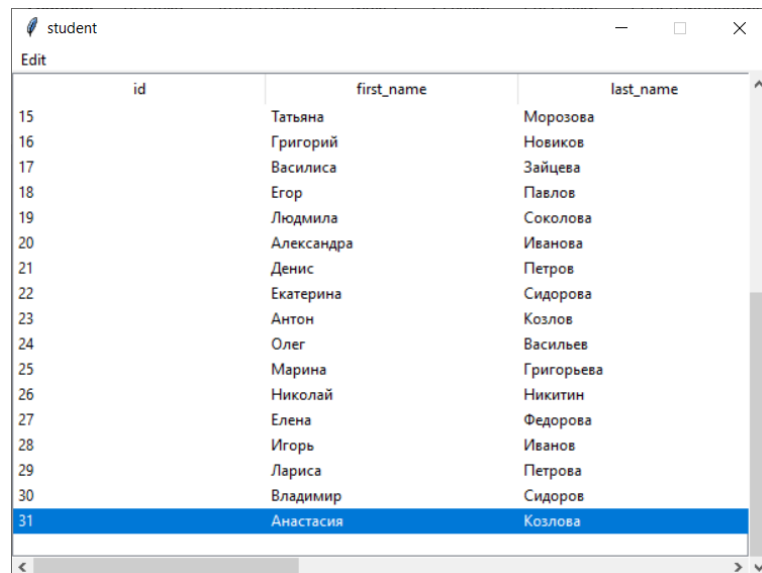


id	32
first_name	
last_name	
middle_name	
passport_id	
date_of_birth	
class_id	
residential_address_id	

DELETE

Рисунок 3.17 – Заполнение полей для удаления ученика “Василия Пупкина” в таблице student

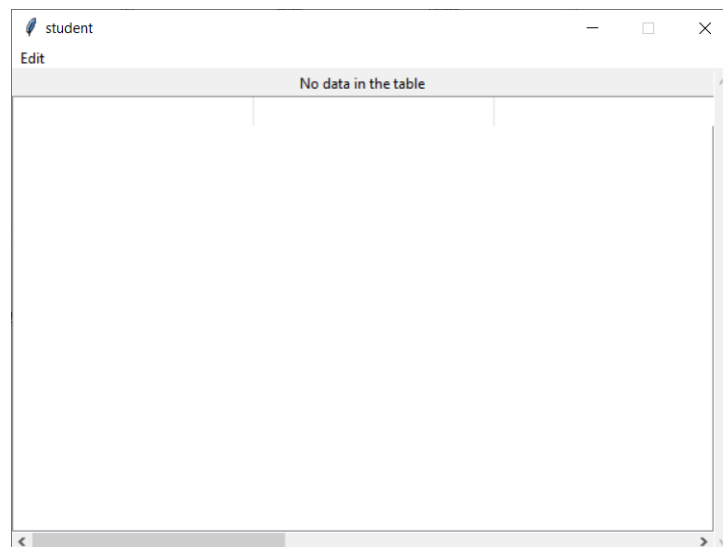
Результаты выполнения запроса DELETE для примера 3.4 представлены на рисунке 3.18.



id	first_name	last_name
15	Татьяна	Морозова
16	Григорий	Новиков
17	Василиса	Зайцева
18	Егор	Павлов
19	Людмила	Соколова
20	Александра	Иванова
21	Денис	Петров
22	Екатерина	Сидорова
23	Антон	Козлов
24	Олег	Васильев
25	Марина	Григорьева
26	Николай	Никитин
27	Елена	Федорова
28	Игорь	Иванов
29	Лариса	Петрова
30	Владимир	Сидоров
31	Анастасия	Козлова

Рисунок 3.18 – Результаты выполнения запроса DELETE для примера 3.4

Результаты выполнения запроса TRUNCATE для примера 3.4 представлены на рисунке 3.19.



id	first_name	last_name
No data in the table		

Рисунок 3.18 – Результаты выполнения запроса TRUNCATE для примера 3.4

### 3.3.4 Выход из приложения

Чтобы выйти из приложения, необходимо закрыть все открытые окна. Соединение с базой данных закроется автоматически.

## 4 ЛИСТИНГ КОДА

### 4.1 Файл main.py

```
import database as db
import mainwindow as mw
from tkinter import messagebox

try:
    postgresql = db.Database()
    postgresql.mainloop()
    if db.connection == None:
        exit()

    app = mw.MainWindow(postgresql)
    app.mainloop()

except Exception:
    messagebox.showerror(title="Error", message="Error while
working with PostgreSQL")
```

### 4.2 Файл database.py

```
import psycopg2
from config import HOST, USER, DB_NAME
from tkinter import *
from tkinter import ttk
from tkinter import messagebox

connection = None

class Database(Tk):
    def __init__(self):
        super().__init__()

        self.title("Connect to Server")
        self.geometry("400x150+540+320")
        self.resizable(False, False)
        self.option_add("*tearOff", FALSE)

        password_label = Label(text=
            "Please enter the password for the user
            'postgres' to connect the server - PostgreSQL
            16",
                                font=("Arial", 12),
                                wraplength=300)
        password_label.pack()

        password_entry = ttk.Entry(show='•')
        password_entry.pack()
```

```

        password_button = ttk.Button(text="OK",
command=lambda: self.on_password_button_click(password_entry))
        password_button.pack()

def __del__(self):
    if connection != None:
        connection.close()
        print "[" + "\033[32m{}".format("INFO") +
"\033[37m{}".format("] PostgreSQL connection closed"))

def on_password_button_click(self, password_entry):
    global connection
    try:
        # connect to database
        connection = psycopg2.connect(
            host=HOST,
            user=USER,
            password=password_entry.get(),
            database=DB_NAME
        )
        connection.autocommit = True

        # the cursor for performing database operations
        with connection.cursor() as cursor:
            cursor.execute("SELECT version();")
            messagebox.showinfo(title="Information",
            message="Successful connection to:\n" +
            "".join(cursor.fetchone()))

        self.destroy()

    except Exception:
        messagebox.showerror(title="Error",
            message="Wrong password!")

def get_table(self, table_name):
    rows = []
    column_names = []
    with connection.cursor() as cursor:
        query = f"SELECT * FROM
            main_scheme.{table_name};"
        cursor.execute(query)
        rows = cursor.fetchall()
        column_names = [desc[0] for desc in
            cursor.description]
    return rows, column_names

```

### 4.3 Файл config.py

```
HOST = '127.0.0.1'
USER = 'postgres'
PASSWORD = '1234'
DB_NAME = 'school'
TABLE_NAMES = [
    "class",
    "employee",
    "employee_position",
    "gradebook",
    "job_position",
    "knowledge_of_subject",
    "residential_address",
    "schedule",
    "student",
    "subject"
]

# В запросах нужно указывать схему, из которой берем таблицы
# для запросов
# SCHEME_NAME = "main_scheme"
```

### 4.4 Файл mainwindow.py

```
from config import TABLE_NAMES
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from tkinter import filedialog
import tablewindow as tw
import database as db
import os

EXAMPLES_PATH = "./sql_scripts/examples/"

class MainWindow(Tk):
    def __init__(self, database):
        super().__init__()
        self.my_db = database
        self.title("Database Manager")
        self.geometry("440x350+540+220")
        self.resizable(False, False)
        self.option_add("*tearOff", FALSE)

        editor = Text(wrap = "none", height=20, width=50)
        editor.grid(column = 0, row = 0, sticky = NSEW)
        ys = ttk.Scrollbar(orient = "vertical",
                           command = editor.yview)
        ys.grid(column = 1, row = 0, sticky = NS)
        xs = ttk.Scrollbar(orient = "horizontal",
                           command = editor.xview)
```



```

xs.grid(column = 0, row = 1, sticky = EW)
editor["yscrollcommand"] = ys.set
editor["xscrollcommand"] = xs.set
editor.insert("1.0", "Enter your query...")

main_menu = Menu()
table_menu = Menu()
query_menu = Menu()

for table_name in TABLE_NAMES:
    table_menu.add_command(label=table_name,
command=lambda name=table_name: self.show_table(name))
    main_menu.add_cascade(label="Tables",
                           menu=table_menu)

    query_menu.add_command(label="Execute",
command=lambda: self.execute_query(editor.get('1.0',
'end')))
    query_menu.add_command(label="Execute from file",
command=self.execute_query_from_file)
    query_menu.add_command(label="Clean",
command=lambda: editor.delete('1.0', 'end'))
    query_menu.add_command(label="Examples",
command=self.show_query_examples)
    main_menu.add_cascade(label="Query",
                           menu=query_menu)

self.config(menu=main_menu)

def show_table(self, table_name):
    table_window = tw.TableWindow(database=self.my_db,
                                   table_name=table_name)
    table_window.mainloop()

def execute_query(self, query):
    try:
        cursor = db.connection.cursor()

        cursor.execute(query)
        rows = cursor.fetchall()
        column_names = [desc[0] for desc in
                        cursor.description]

        result_window = Tk()
        result_window.option_add("*tearOff", FALSE)
        result_window.title("Query Result")
        result_window.geometry("600x400")
        result_window.resizable(False, False)

        result_table = ttk.Treeview(result_window,

```

```

        columns=column_names, show="headings", height=20)

        table_yscrollbar = ttk.Scrollbar(result_window,
orient="vertical", command=result_table.yview)
        table_xscrollbar = ttk.Scrollbar(result_window,
orient="horizontal", command=result_table.xview)
        table_yscrollbar.pack(side=RIGHT, fill=Y)
        table_xscrollbar.pack(side=BOTTOM, fill=X)

result_table["yscrollcommand"]=table_yscrollbar.set

result_table["xscrollcommand"]=table_xscrollbar.set

        if not rows:
            label = Label(result_window,
                           text="No data in the table")
            label.pack()
        else:
            for col in column_names:
                result_table.heading(col, text=col)
                result_table.column(col, stretch=True)

            for row in rows:
                result_table.insert('', 'end',
                                    values=row)

        result_table.pack(fill="both")
        messagebox.showinfo(title="Information",
                             message="Query executed successfully!")
        result_window.mainloop()
        return True

    except Exception as _ex:
        messagebox.showerror(title="Error",
                             message=f"Wrong query: {_ex}")

    finally:
        cursor.close()

def execute_query_from_file(self):
    filepath = filedialog.askopenfile(
        title="Execute SQL-script")
    if filepath != "":
        with open(filepath, "r") as file:
            text = file.read()
            self.execute_query(text)

def show_query_examples(self):
    examples_window = Tk()
    examples_window.option_add("*tearOff", FALSE)

```

```

examples_window.title("Query List")
examples_window.geometry("600x400")
examples_window.resizable(False, False)

query_list = Listbox(examples_window,
                     selectmode=SINGLE, height=22)

info_button = ttk.Button(examples_window,
                        text="Info", width=30, command=lambda:
self.show_script_info(query_list))
exec_button = ttk.Button(examples_window,
                        text="Execute", width=30, command=lambda:
self.execute_example(query_list))
info_button.place(x=70, y=10)
exec_button.place(x=320, y=10)

list_yscrollbar = ttk.Scrollbar(examples_window,
                                orient="vertical", command=query_list.yview)
list_yscrollbar.pack(side='right', fill='y')
query_list["yscrollcommand"]=list_yscrollbar.set

file_names = []
with os.scandir(EXAMPLES_PATH) as entries:
    for entry in entries:
        if entry.is_file():
            file_names.append(entry.name)

for name in file_names:
    query_list.insert("end", name)

query_list.pack(fill='both', side="bottom")
examples_window.mainloop()

def show_script_info(self, query_list):
    selected_query_index = query_list.curselection()
    if len(selected_query_index) != 0:
        selected_query_name =
            query_list.get(selected_query_index[0])

        info_window = Tk()
        info_window.option_add("*tearOff", FALSE)
        info_window.title(
            f"Info about {selected_query_name}")
        info_window.geometry("440x350")
        info_window.resizable(False, False)

        info = Text(info_window, wrap = "none",
                    height=20, width=50)
        info.grid(column = 0, row = 0, sticky = NSEW)
        ys = ttk.Scrollbar(info_window,
                            orient = "vertical", command = info.yview)

```

```

        xs = ttk.Scrollbar(info_window,
            orient = "horizontal", command = info.xview)
        ys.grid(column = 1, row = 0, sticky = NS)
        xs.grid(column = 0, row = 1, sticky = EW)
        info["yscrollcommand"] = ys.set
        info["xscrollcommand"] = xs.set

        with open(EXAMPLES_PATH + selected_query_name,
            "r", encoding="utf-8") as file:
            text = file.read()
            info.insert('1.0', text)
            info['state']='disabled'

def execute_example(self, query_list):
    selected_query_index = query_list.curselection()
    if len(selected_query_index) != 0:
        selected_query_name =
            query_list.get(selected_query_index[0])
        with open(EXAMPLES_PATH + selected_query_name,
            "r", encoding="utf-8") as file:
            text = file.read()
            self.execute_query(text)

```

## 4.5 Файл tablewindow.py

```

from tkinter import *
from tkinter import ttk
from tkinter import messagebox
import database as db
import query_generator as qgen

class TableWindow(Tk):
    def __init__(self, database, table_name):
        super().__init__()
        self.option_add("*tearOff", FALSE)
        self.table_name = table_name
        self.database = database

        data, column_names = database.get_table(table_name)
        self.table_data = data
        self.column_names = column_names

        self.title(table_name)
        self.geometry("600x400")
        self.resizable(False, False)

        table_menu = Menu(self)
        edit_menu = Menu(self)
        edit_menu.add_command(label="Insert",
            command=lambda name=table_name: self.edit_table("INSERT"))

```

```

        edit_menu.add_command(label="Delete",
command=lambda name=table_name: self.edit_table("DELETE"))
        edit_menu.add_command(label="Update",
command=lambda name=table_name: self.edit_table("UPDATE"))
        edit_menu.add_command(label="Truncate",
command=lambda name=table_name: self.edit_table("TRUNCATE"))
        table_menu.add_cascade(label="Edit", menu=edit_menu)
        self.config(menu=table_menu)

        table = ttk.Treeview(self, columns=column_names,
                             show="headings", height=20)

        table_yscrollbar = ttk.Scrollbar(self,
                                         orient="vertical", command=table.yview)
        table_xscrollbar = ttk.Scrollbar(self,
                                         orient="horizontal", command=table.xview)
        table_yscrollbar.pack(side=RIGHT, fill=Y)
        table_xscrollbar.pack(side=BOTTOM, fill=X)
        table["yscrollcommand"]=table_yscrollbar.set
        table["xscrollcommand"]=table_xscrollbar.set

        if not data:
            label = Label(self, text="No data in the table")
            label.pack()
        else:
            for col in column_names:
                table.heading(col, text=col)
                table.column(col, stretch=True)

            for row in data:
                table.insert('', 'end', values=row)

        table.pack(fill="both")

    def edit_table(self, command):
        if command == "TRUNCATE":
            with db.connection.cursor() as cursor:
                query =
f"TRUNCATE TABLE main_scheme.{self.table_name} CASCADE;"
                cursor.execute(query)
                messagebox.showinfo(title="Information",
message="Query executed successfully!")
            return

        edit_window = Tk()
        edit_window.title(command + " " + self.table_name)
        edit_window.geometry("350x350")

        i = 0
        entries = []
        labels = []

```

```

for column_name in self.column_names:
    if column_name == 'id' and command == "INSERT":
        continue
    label = Label(edit_window, text=column_name)
    entry = Entry(edit_window, width=30)
    entries.append(entry)
    labels.append(label['text'])
    label.grid(row = i, column=0,
               sticky="e", padx=10, pady=10)
    entry.grid(row = i, column=1, pady=10)
    i += 1
button = ttk.Button(edit_window,
                    text=command,
                    width=30,
                    command=lambda: self.edit(command, labels,
                                              entries))
button.grid(row = i, column=1, pady=10)

edit_window.mainloop()

def edit(self, command, labels, entries):
    values = []
    for field in entries:
        values.append(field.get())

    print(f"Values: {values}")

    query_generator = qgen.QueryGenerator()

    try:
        if command == "INSERT":
            query = query_generator
            .generate_insert_query(
                table_name=self.table_name,
                column_names=labels,
                entries=values)
            print(query)
            with db.connection.cursor() as cursor:
                cursor.execute(query)

        elif command == "DELETE":
            query = query_generator
            .generate_delete_query(
                self.table_name,
                column_names=labels,
                entries=values)
            print(query)
            with db.connection.cursor() as cursor:
                cursor.execute(query)

        elif command == "UPDATE":

```

```

        query = query_generator
        .generate_update_query(self.table_name,
                               column_names=labels,
                               entries=values)
        print(query)
        with db.connection.cursor() as cursor:
            cursor.execute(query)

        messagebox.showinfo(title="Information",
                             message="Query completed successfully!")

    except Exception as _ex:
        messagebox.showerror("Error",
                              f"Wrong {command} query: {_ex}")

```

## 4.6 Файл query\_generator.py

```

class QueryGenerator:
    def generate_insert_query(self,
                              table_name,
                              column_names,
                              entries):
        # Формируем строку с названиями столбцов
        columns = ', '.join(column_names)

        # Формируем строку со значениями
        values = ', '.join(f"'{entry}'"
                            if entry else 'NULL' for entry in entries)

        # Собираем полный запрос
        query = f"INSERT INTO main_scheme.
                 {table_name} ({columns}) VALUES ({values});"
        return query

    def generate_delete_query(self,
                              table_name,
                              column_names,
                              entries):
        # Формируем условие для удаления
        conditions = ' AND '.join(f"{col} =
                                   '{entry}'" for col, entry in
                                   zip(column_names, entries) if entry)

        # Собираем полный запрос
        query = f"DELETE FROM main_scheme.{table_name}
                 WHERE {conditions};"
        return query

    def generate_update_query(self,
                              table_name,

```

```

        column_names,
        entries):
    if entries[0] == None:
        raise Exception("'id' cannot be NULL")

    update_query = f"UPDATE main_scheme.{table_name} SET "
    set_clauses = []

    for col_name, entry in zip(column_names, entries):
        if entry and col_name != 'id':
            # Если значение не пустое, добавляем его в запрос
            set_clauses.append(f"{col_name} =
                                '{entry}'")

    update_query += ", ".join(set_clauses) +
        f" WHERE {column_names[0]} = {entries[0]}"
    return update_query

```



## **5 ВЫВОД**

В ходе лабораторной работе было создана прикладная программа для работы с базой данных и выполняющая заданные транзакции. Также был реализован механизм работы с базой данных (добавление новых данных в таблицу, удаление, обновление).