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 C-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 тулкит – набор компонентов, который называется tkinter. Тулкит 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 тулкит tkinter доступны для большинства операционных систем, в том числе для Mac OS, Linux и Windows.

Преимущества Tkinter:

* Данный тулкит 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.

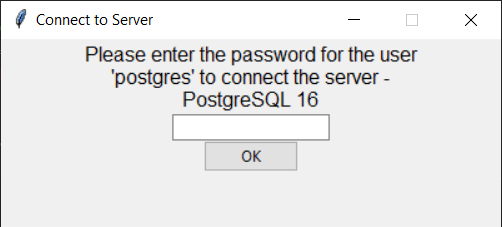


Рисунок 3.1 – Окно для подключения к серверу

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.

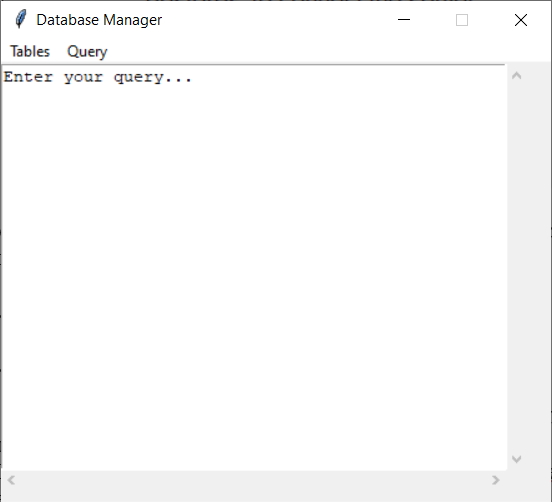


Рисунок 3.2 – Главное окно программы

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.

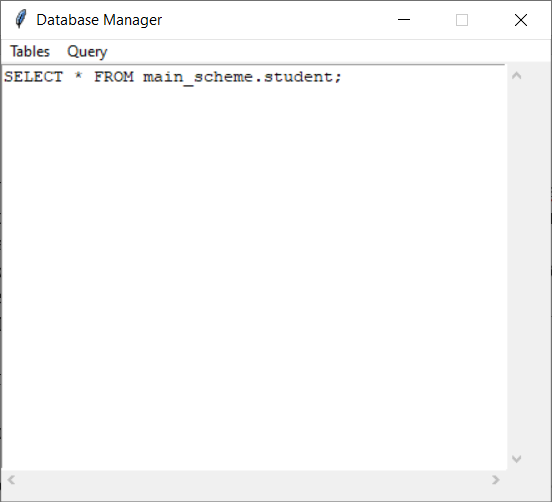


Рисунок 3.3 – Запрос для примера 3.3 в приложении

Результаты запроса для примера 3.1 показаны на рисунке 3.4.

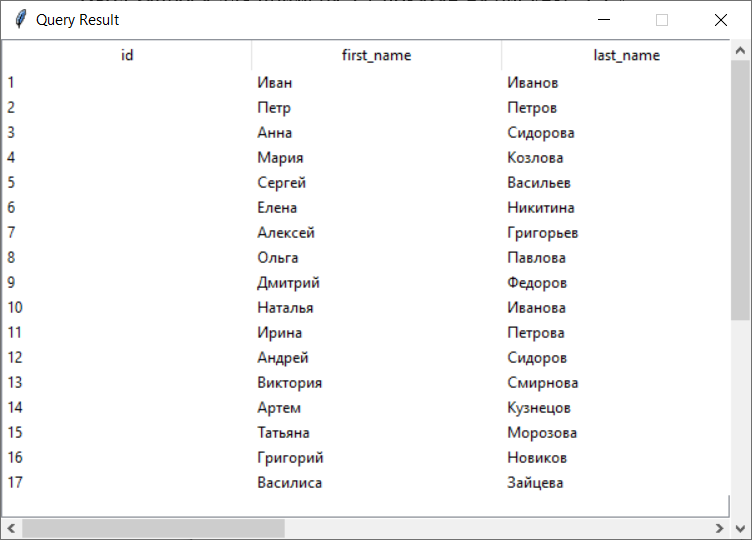


Рисунок 3.4 – Результаты выполнения запроса для примера 3.1

**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.

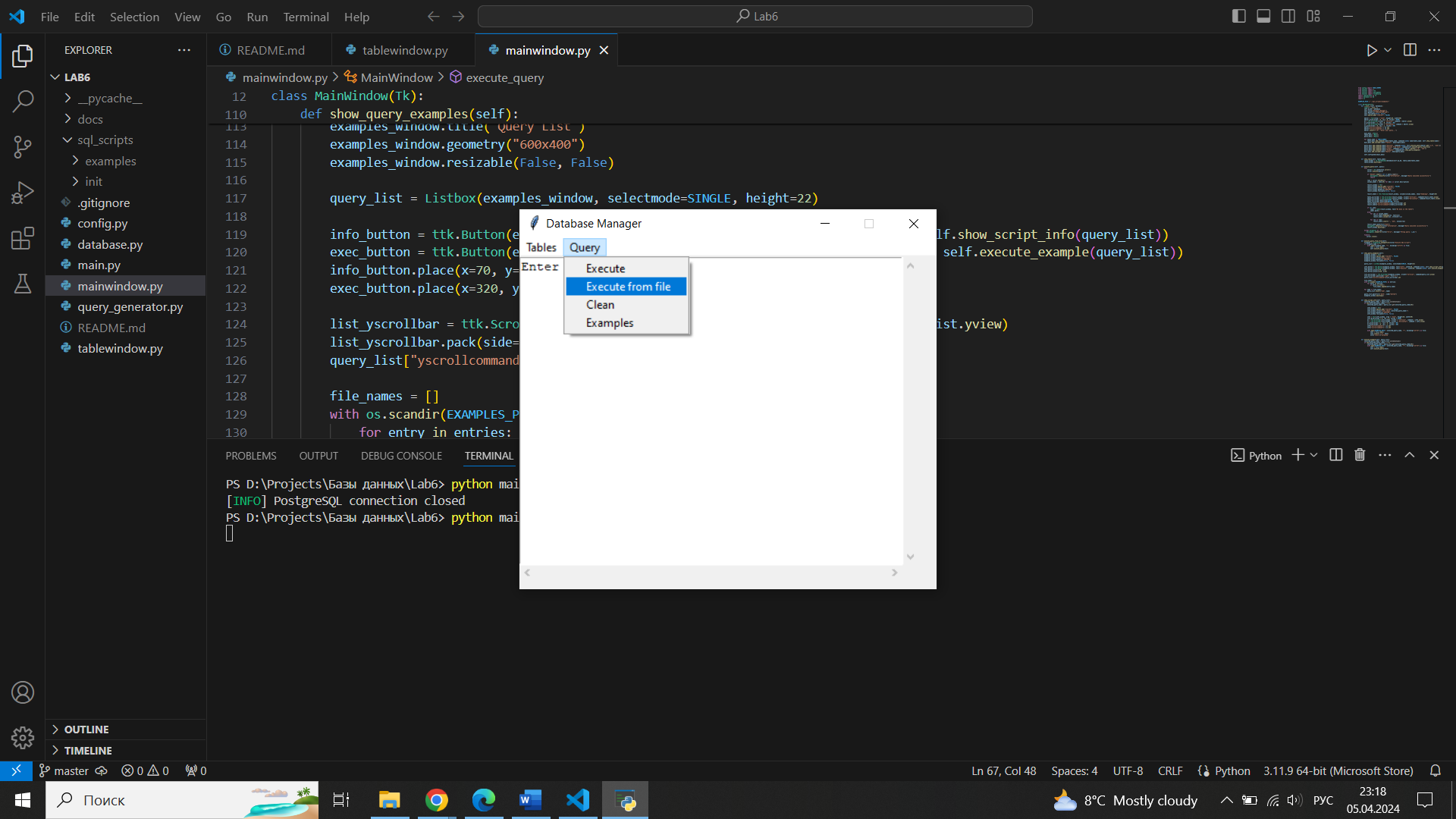


Рисунок 3.5 – Выбор команды выполнения SQL-запроса из файла

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

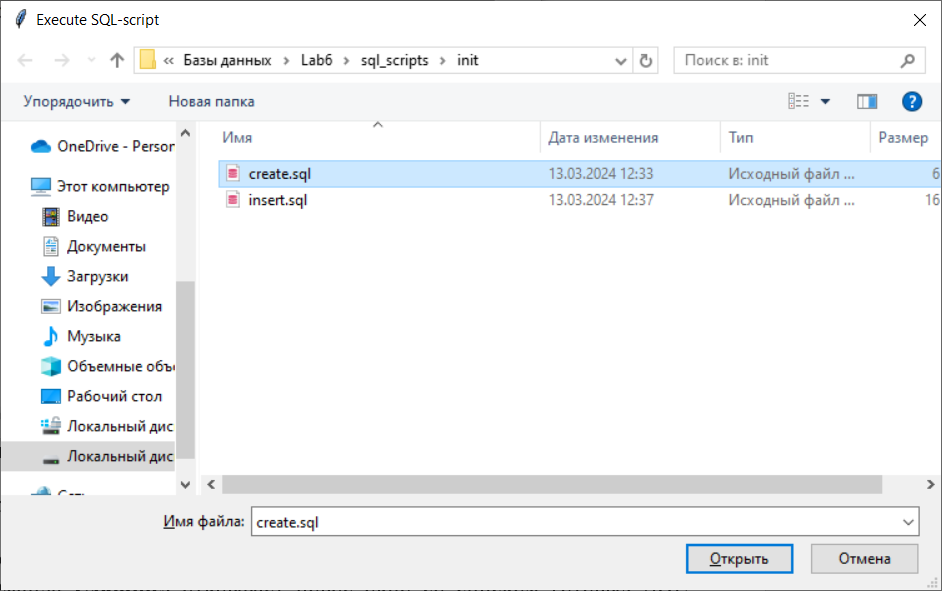


Рисунок 3.6 – Выбор SQL-скрипта для выполнения

После выполнения запроса появится информационное окно, показанное на рисунке 3.7. Далее можно воспользоваться, например, pgАdmin4, чтобы удостовериться, что таблицы действительно были созданы.

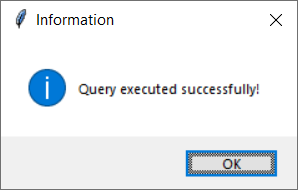


Рисунок 3.7 – Информационное окно о успешном выполнении запроса

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

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

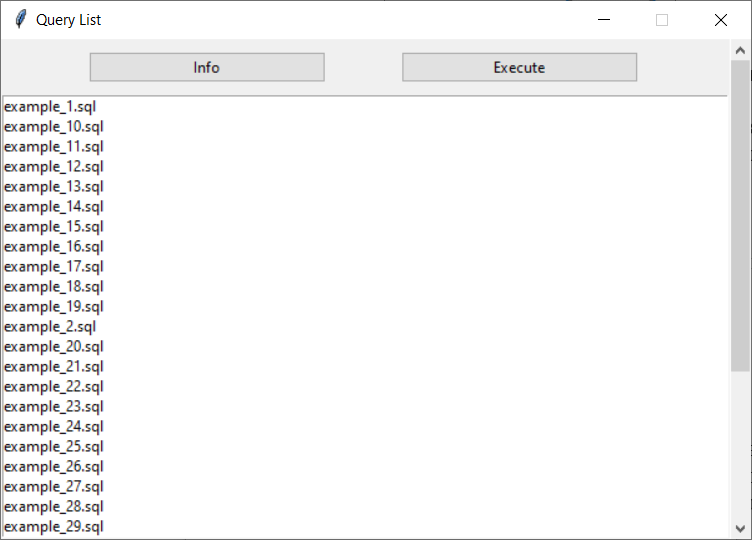


Рисунок 3.8 – Окно Examples с готовыми SQL-запросами для выполнения

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.

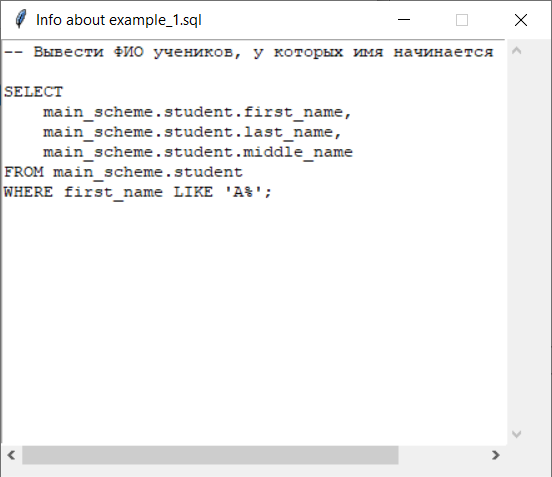


Рисунок 3.9 – Вывод информации о запросе из файла example\_1.sql

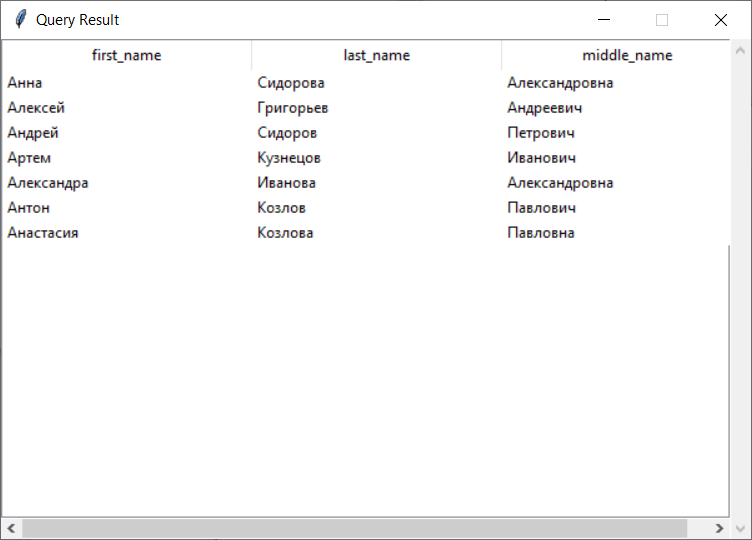


Рисунок 3.10 – Результаты выполнения скрипта example\_1.sql

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

**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.

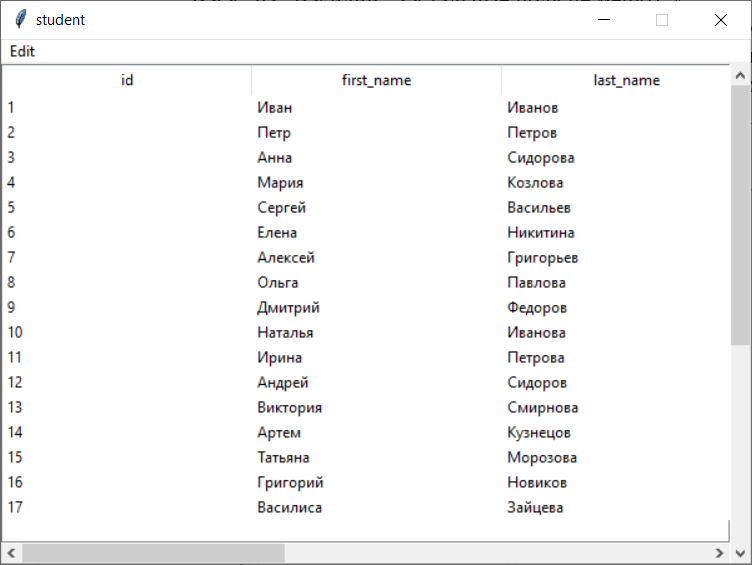


Рисунок 3.11 – Первая часть исходной таблицы student

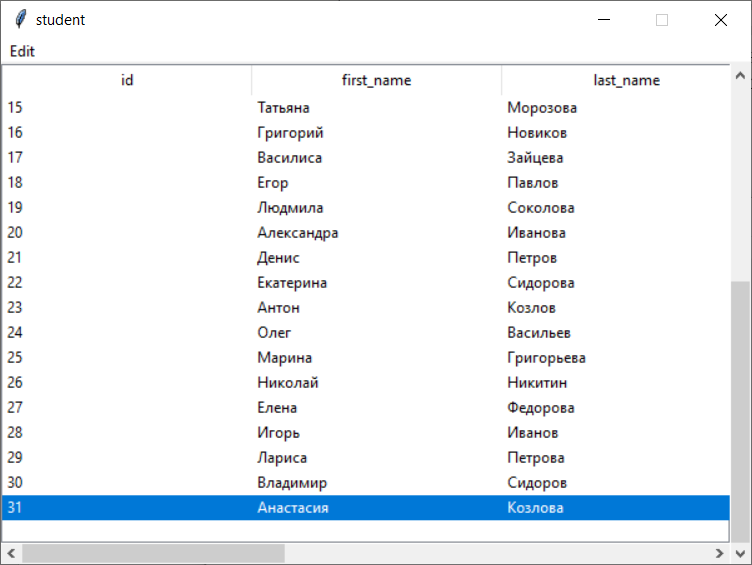


Рисунок 3.12 – Вторая часть исходной таблицы student

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

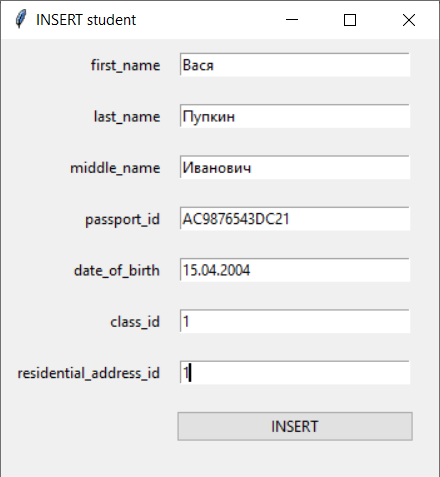


Рисунок 3.13 – Заполнение полей для добавления “Васи Пупкина” в таблицу student

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

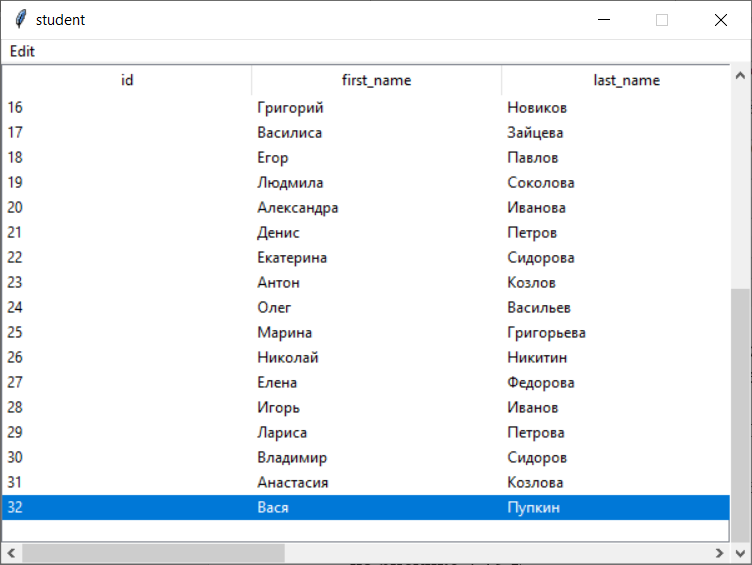


Рисунок 3.14 – Результаты выполнения запроса INSERT для примера 3.4

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

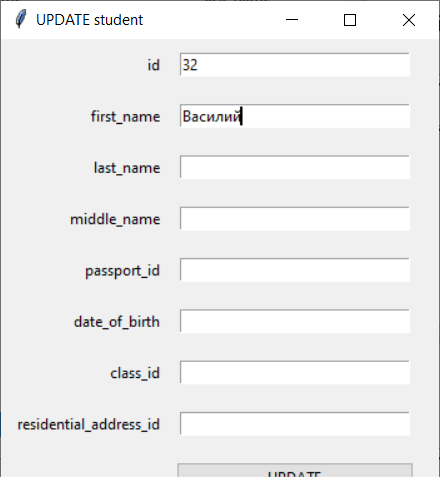


Рисунок 3.15 – Заполнение полей для обновления полей “Васи Пупкина” в таблице student

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

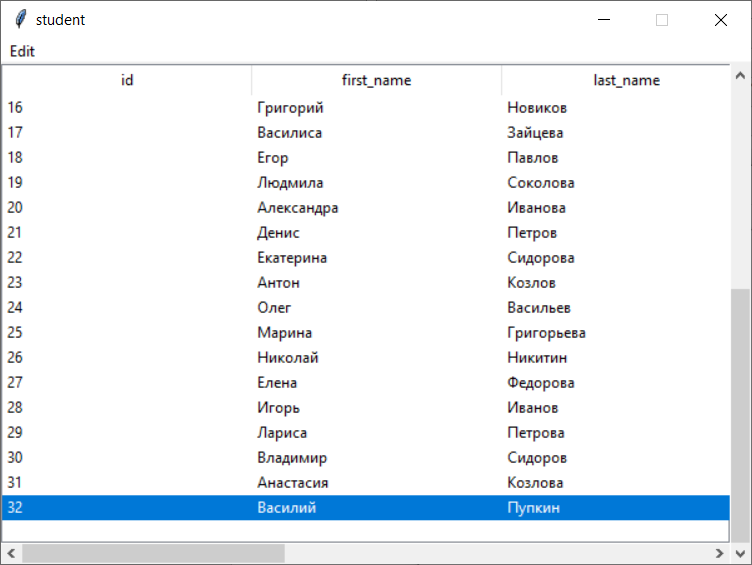


Рисунок 3.16 – Результаты выполнения запроса UPDATE для примера 3.4

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

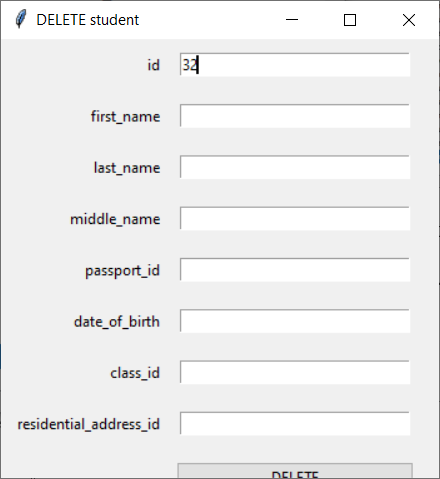


Рисунок 3.17 – Заполнение полей для удаления ученика “Василия Пупкина” в таблице student

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

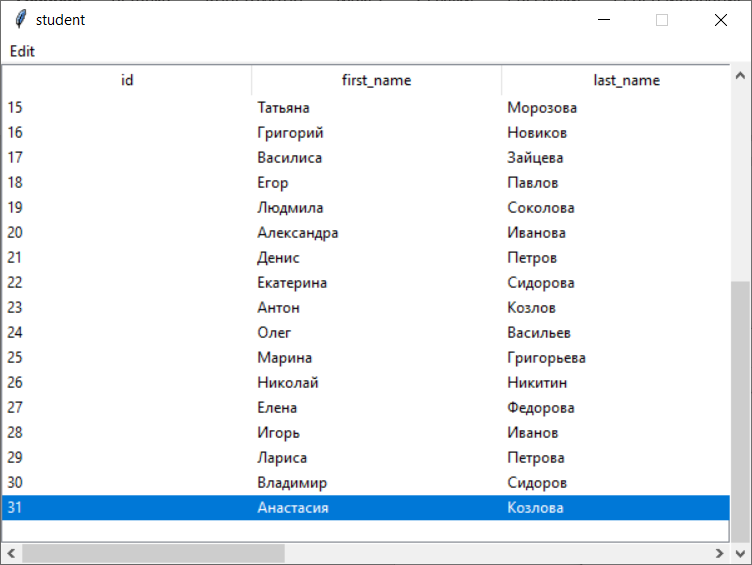


Рисунок 3.18 – Результаты выполнения запроса DELETE для примера 3.4

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

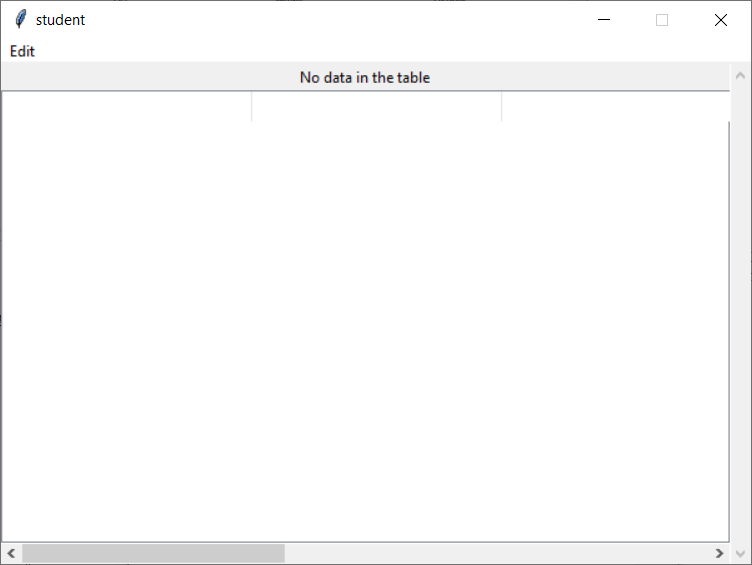


Рисунок 3.18 – Результаты выполнения запроса TRUNCATE для примера 3.4

**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 perfoming 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).