Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра 307 «Информационные системы и технологии»

ОТЧЁТ

О выполнении задания по дисциплине

«Базы данных»

Отчет по работе

“Создание базы данных и клиентского приложения”

Выполнили:

Студенты группы М3О-212Б-22:

Барденков Т. Д.

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Абдуллин Т.И.

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель проекта:

к. техн. наук, доцент кафедры 307:

Склеймин Ю.Б.

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2023 г.

Цель работы

Разработать базу данных, реализующую функцию телефонного справочника, на СУБД PostgreSQL и приложение по архитектуре клиент-сервер для работы с ней. Интерфейс должен включать в себя основные функции для работы с базой данных:

1. Добавление записи
2. Изменение записи
3. Удаление записи
4. Вывод всей таблицы

Структура базы данных

Для полей таблицы справочника было решено использовать столбцы u\_id, fam, nam, otc, street, building, phone (уникальный идентификатор, фамилия, имя, отчество, улица, строение, телефон). Их типы соответственно: serial, integer, integer, integer, integer, char 10, char 20.

Для полей фамилии, имени, отчества и улицы используются родительские таблицы, то есть в полях справочника хранятся уникальные идентификаторы фамилий, имен, отчеств и улиц в соответствующих им родительских таблицах. Сделан данный функционал для оптимальной работы БД. Хранящиеся значения в родительских таблицах обеспечивают уникальность значений в дочерней таблице main, что позволяет уменьшить расход памяти для хранения БД и оптимизировать ее работу.

Родительская таблица fam содержит 2 столбца: fam\_id и fam, хранящие уникальные ключи и соответствующие им фамилии. Соответственно для остальных таблиц nam, otc и street существуют столбцы с уникальным ключом и столбцом данных.

Добавлены ограничения в виде ссылочной целостности для столбцов fam, nam, otc и street. При попытке удаления записи из родительской таблицы, на которую ссылается запись из дочерней, этот запрос на удаление не выполняется, т.к. для операции удаления выставлено действие RESTRICT (запрет удаления). При попытке же обновления записи из родительской таблицы, на которую ссылается запись из дочерней, этот запрос на обновление выполняется не только для родительской, но и для дочерней таблицы, т.к. для операции обновления выставлено действие CASCADE (Каскадное обновление данных).

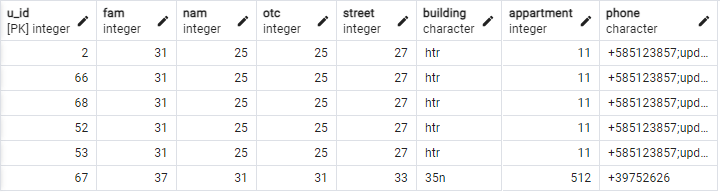


Рисунок 1 — структура таблицы main



Рисунок 2 — структура одной из родительских таблиц

Библиотека psycopg2

Данная библиотека позволяет управлять БД с помощью программ на языке программирования Python.

Для подключения к БД используется команда conn = psycopg2.connect(“Адрес БД”). Для установки указателя среду выполнения запросов выполняется команда cur = conn.cursor(). Далее для выполнения SQL запросов используется метод cur.execute(“SQL запрос”).

Программа

Программа при открытии показывает пользователю графический интерфейс, имеющий поля ввода данных и кнопки действий.

Пользователь должен иметь возможность:

1. Ввод данных (Добавить данные): ввести в текстовые поля ФИО, улицу, дом и телефон новой записи. Ввод всех данных обязателен. При нажатии кнопки “Добавить данные” производится проверка – существуют ли указанные ФИО в соответствующих родительских таблицах. Если нет, то новые данные добавляются в родительские таблицы. Иначе в таблицу main вставляются идентификаторы уже существующих записей.
2. Обновление данных (Изменить данные): производятся те же действия, что и при операции “Добавить данные”, однако вместо добавления новой записи в таблицу main изменяется уже существующая запись. Помимо всех данных изменяемой записи необходимо также ввести идентификатор записи в соответствующее поле.
3. Удаление данных (Удалить данные): выбирается запись таблицы main, имеющая совпадающий с введённым в поле “ID для удаления” идентификатор записи. Далее запись удаляется.
4. Вывод данных (Показать данные): производится вывод отдельного окна таблицы main.

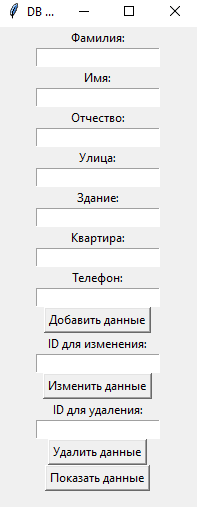


Рисунок 3 — Интерфейс клиента

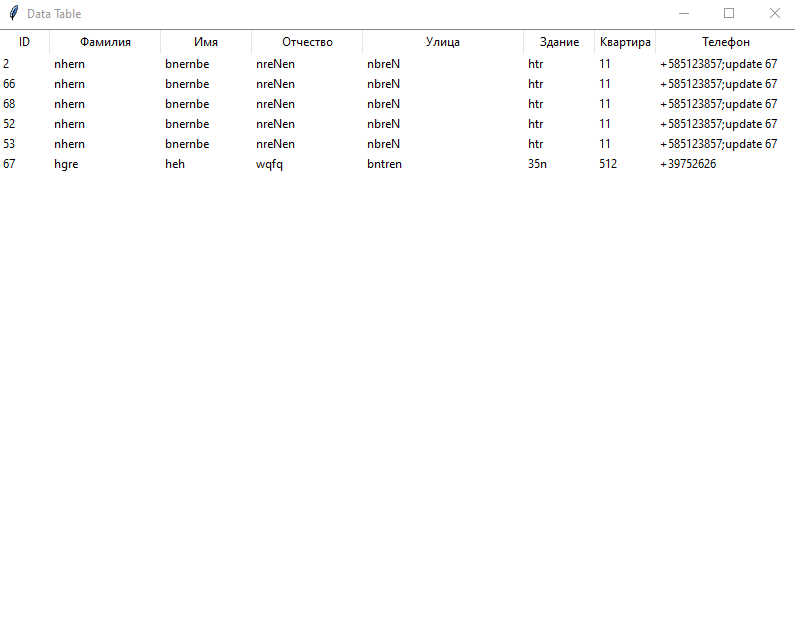


Рисунок 4 — Интерфейс таблицы

Тестирование

На этапе тестирования мы протестируем все 4 возможные функции клиентского приложения: добавление, изменение, удаление записи и вывод всей таблицы.

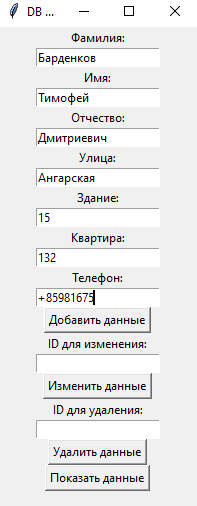


Рисунок 5 — Добавление записи

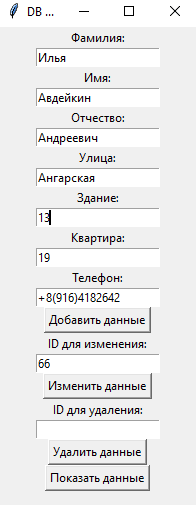


Рисунок 6 — Изменение записи

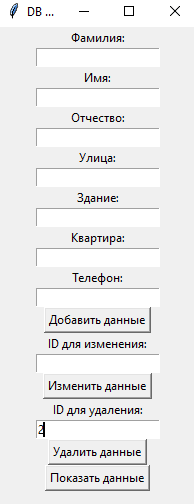


Рисунок 7 — Удаление записи

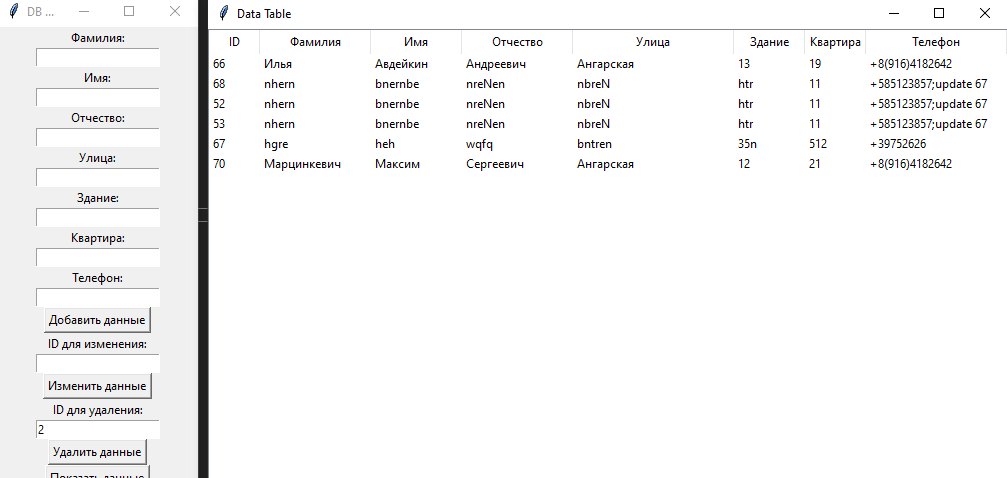


Рисунок 8 — Вывод таблицы

Как видно из приведенных рисунков 5-8, все функции работают исправно.

Вывод

Разработана структура базы данных, создана сама база данных, реализующая функции телефонного справочника. Настроены ограничения по ссылочной целостности для корректной работы. Разработан пользовательский интерфейс с архитектурой клиент-сервер на языке программирования Python 3.12 с использованием графической библиотеки tkinter и библиотекой для работы с СУБД psycopg2. Проведены тесты программы, которые показали работоспособность разработанной программы и исправность работы данных функций.

Приложение 1: код программы

import psycopg2

import tkinter as tk

from tkinter import ttk

def main():

attributes = ('fam', 'nam', 'otc', 'street')

fams = []

nams = []

otcs = []

streets = []

# DB connect

try:

conn = psycopg2.connect(

dbname = "phonebook",

user = "postgres",

password = "12345",

host = "localhost"

)

if not conn.closed:

print("Успешное подключение к базе данных PostgreSQL!")

cur = conn.cursor()

conn.autocommit = True

for i in range(4):

cur.execute(f"""select {attributes[i]} from {attributes[i]}""")

elements = cur.fetchall()

cur.execute(f"""select {attributes[i]}\_id from {attributes[i]}""")

ids = cur.fetchall()

for elem in range(len(elements)):

elements[elem] = (elements[elem][0]).strip()

for id in range(len(ids)):

ids[id] = ids[id][0]

match i:

case 0:

for m in range(len(elements)):

fams.append([elements[m], ids[m]])

case 1:

for m in range(len(elements)):

nams.append([elements[m], ids[m]])

case 2:

for m in range(len(elements)):

otcs.append([elements[m], ids[m]])

case 3:

for m in range(len(elements)):

streets.append([elements[m], ids[m]])

else:

print("Не удалось установить соединение с базой данных.")

except psycopg2.Error as e:

print("Ошибка при подключении к базе данных PostgreSQL:", e)

# Initialize window

def init():

window = tk.Tk()

window.geometry("200x480")

window.title("DB Client")

# Get data from TKinter entry poles

def get\_entry\_data():

data = []

data.append(entry\_fam.get())

data.append(entry\_nam.get())

data.append(entry\_otc.get())

data.append(entry\_street.get())

data.append(entry\_building.get())

data.append(entry\_apartment.get())

data.append(entry\_phone.get())

return data

# Insert function

def insert\_data():

data = get\_entry\_data()

flag = True

print()

for elem in data:

if not elem: flag = False

if not flag:

print("Not all data inserted, enter all data!")

else:

if data[0] not in fams:

cur.execute(f"""insert into fam values(default, '{data[0]}')""")

cur.execute(f"""select fam\_id from fam where fam = '{data[0]}'""")

fam\_id = cur.fetchone()[0]

fams.append([data[0], fam\_id])

else:

cur.execute(f"""select fam\_id from fam where fam = '{data[0]}'""")

fam\_id = cur.fetchone()[0]

print(fam\_id)

if data[1] not in nams:

cur.execute(f"""insert into nam values(default, '{data[1]}')""")

cur.execute(f"""select nam\_id from nam where nam = '{data[1]}'""")

nam\_id = cur.fetchone()[0]

nams.append([data[1], nam\_id])

else:

cur.execute(f"""select nam\_id from nam where nam = '{data[1]}'""")

nam\_id = cur.fetchone()[0]

if data[2] not in otcs:

cur.execute(f"""insert into otc values(default, '{data[2]}')""")

cur.execute(f"""select otc\_id from otc where otc = '{data[2]}'""")

otc\_id = cur.fetchone()[0]

otcs.append([data[2], otc\_id])

else:

cur.execute(f"""select otc\_id from otc where otc = '{data[2]}'""")

otc\_id = cur.fetchone()[0]

if data[3] not in streets:

cur.execute(f"""insert into street values(default, '{data[3]}')""")

cur.execute(f"""select street\_id from street where street = '{data[3]}'""")

street\_id = cur.fetchone()[0]

streets.append([data[3], street\_id])

else:

cur.execute(f"""select street\_id from street where street = '{data[3]}'""")

street\_id = cur.fetchone()[0]

cur.execute(f"""insert into main values(default, {fam\_id}, {nam\_id}, {otc\_id}, {street\_id}, '{data[4]}', {data[5]}, '{data[6]}')""")

print('Succesful insert')

# Change function

def change\_data():

change\_id = int(entry\_change\_id.get())

if not change\_id:

print("Insert ID for string you want to change!")

data = get\_entry\_data()

flag = True

print()

for elem in data:

if not elem: flag = False

if not flag:

print("Not all data inserted, enter all data!")

else:

if data[0] not in fams:

cur.execute(f"""insert into fam values(default, '{data[0]}')""")

cur.execute(f"""select fam\_id from fam where fam = '{data[0]}'""")

fam\_id = cur.fetchone()[0]

fams.append([data[0], fam\_id])

else:

cur.execute(f"""select fam\_id from fam where fam = '{data[0]}'""")

fam\_id = cur.fetchone()[0]

print(fam\_id)

if data[1] not in nams:

cur.execute(f"""insert into nam values(default, '{data[1]}')""")

cur.execute(f"""select nam\_id from nam where nam = '{data[1]}'""")

nam\_id = cur.fetchone()[0]

nams.append([data[1], nam\_id])

else:

cur.execute(f"""select nam\_id from nam where nam = '{data[1]}'""")

nam\_id = cur.fetchone()[0]

if data[2] not in otcs:

cur.execute(f"""insert into otc values(default, '{data[2]}')""")

cur.execute(f"""select otc\_id from otc where otc = '{data[2]}'""")

otc\_id = cur.fetchone()[0]

otcs.append([data[2], otc\_id])

else:

cur.execute(f"""select otc\_id from otc where otc = '{data[2]}'""")

otc\_id = cur.fetchone()[0]

if data[3] not in streets:

cur.execute(f"""insert into street values(default, '{data[3]}')""")

cur.execute(f"""select street\_id from street where street = '{data[3]}'""")

street\_id = cur.fetchone()[0]

streets.append([data[3], street\_id])

else:

cur.execute(f"""select street\_id from street where street = '{data[3]}'""")

street\_id = cur.fetchone()[0]

cur.execute(f"""update main set fam = {fam\_id}, nam = {nam\_id}, otc = {otc\_id}, street = {street\_id}, building = '{data[4]}', appartment = {data[5]}, phone = '{data[6]}' where u\_id = {change\_id}""")

print("Succesful data change")

# Delete function

def delete\_data():

delete\_id = int(entry\_delete\_id.get())

if not delete\_id:

print("Insert ID for string you want to delete!")

cur.execute(f"""delete from main where u\_id = {delete\_id}""")

print("Succesful deletion")

# Show data function

def get\_table\_data():

data = []

cur.execute("""select u\_id, fam.fam, nam.nam, otc.otc,

street.street, building, appartment, phone from main

join fam on main.fam = fam\_id

join nam on main.nam = nam\_id

join otc on main.otc = otc\_id

join street on main.street = street\_id;""")

rows = cur.fetchall()

for row in rows:

cleaned\_row = [str(item).strip() for item in row]

data.append(cleaned\_row)

print("Succesful show")

return data

# Show table

def show\_table():

data = get\_table\_data()

table\_window = tk.Toplevel()

table\_window.geometry("800x600")

table\_window.title("Data Table")

columns = ('id', 'fam', 'nam', 'otc', 'street', 'building', 'appartment', 'phone')

tree = ttk.Treeview(table\_window, columns=columns, show='headings')

#tree["columns"] = ("ID", "Фамилия", "Имя", "Отчество", "Улица", "Здание", "Квартира", "Телефон")

tree.heading("id", text="ID")

tree.heading("fam", text="Фамилия")

tree.heading("nam", text="Имя")

tree.heading("otc", text="Отчество")

tree.heading("street", text="Улица")

tree.heading("building", text="Здание")

tree.heading("appartment", text="Квартира")

tree.heading("phone", text="Телефон")

i = 0

for wid in (10, 70, 50, 70, 120, 30, 20, 100):

tree.column(tree["columns"][i], width=wid)

i += 1

for row in data:

tree.insert("", 'end', values=row)

tree.pack(expand=True, fill="both")

# Функциональные поля для ввода данных

label\_fam = tk.Label(window, text="Фамилия:")

label\_fam.pack()

entry\_fam = tk.Entry(window)

entry\_fam.pack()

label\_nam = tk.Label(window, text="Имя:")

label\_nam.pack()

entry\_nam = tk.Entry(window)

entry\_nam.pack()

label\_otc = tk.Label(window, text="Отчество:")

label\_otc.pack()

entry\_otc = tk.Entry(window)

entry\_otc.pack()

label\_street = tk.Label(window, text="Улица:")

label\_street.pack()

entry\_street = tk.Entry(window)

entry\_street.pack()

label\_building = tk.Label(window, text="Здание:")

label\_building.pack()

entry\_building = tk.Entry(window)

entry\_building.pack()

label\_apartment = tk.Label(window, text="Квартира:")

label\_apartment.pack()

entry\_apartment = tk.Entry(window)

entry\_apartment.pack()

label\_phone = tk.Label(window, text="Телефон:")

label\_phone.pack()

entry\_phone = tk.Entry(window)

entry\_phone.pack()

# Кнопка для добавления данных в БД

button\_insert = tk.Button(window, text="Добавить данные", command=insert\_data)

button\_insert.pack()

# Кнопка для обновления данных в БД

# Entry for ID to change

label\_change\_id = tk.Label(window, text="ID для изменения:")

label\_change\_id.pack()

entry\_change\_id = tk.Entry(window)

entry\_change\_id.pack()

button\_change = tk.Button(window, text="Изменить данные", command=change\_data)

button\_change.pack()

# Кнопка для удаления данных БД

label\_delete\_id = tk.Label(window, text="ID для удаления:")

label\_delete\_id.pack()

entry\_delete\_id = tk.Entry(window)

entry\_delete\_id.pack()

button\_delete = tk.Button(window, text="Удалить данные", command=delete\_data)

button\_delete.pack()

# Кнопка для показа данных БД

button\_show = tk.Button(window, text="Показать данные", command=show\_table)

button\_show.pack()

window.mainloop()

init()

if \_\_name\_\_ == "\_\_main\_\_":

main()