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

Аэропорт

Студент: И.И. Божко

Преподаватель: Д.В. Куприянова

МИНСК 2024

1 План выполнения работы

Создание прикладной программы для работы с базой данных и выполняющей заданные транзакции, а также реализовать механизм работы с базой данных (добавление новых данных в таблицу, удаление, обновление). Можно использовать любую среду и язык программирования.

Писать запрос в приложении нельзя! Нужно реализовать интерфейс вывода запросов из 4 и 5 лабораторной работы.

В отчете отразить фактически руководство пользователя и лист кода.

2 Описание приложения

Приложение предназначено для работы с базой данных “Аэропорт”. Приложение выполняет следующие функции: добавление записи, удаление записи, просмотр таблицы, удаление записи, выполнение SQL запросов. Для разработки приложения был выбран язык программирования Python, для создания графического интерфейса – библиотека Tkinter. Данные технологии были выбраны в связи с наличием большого количества методов и функций для работы с базой данных и текстовыми данными, а также с простотой создания графического интерфейса.

Требования к запуску приложения:

Операционная система:

1. Windows 7 или более поздняя
2. macOS 10.10 или более поздняя
3. Linux (любой дистрибутив)

Библиотеки:

1. tkinter: входит в стандартную библиотеку Python 3
2. psycopg2: 2.8.6 или более поздняя

Объём занимаемого места в ОЗУ: 15 МБ.

Внешний вид приложения представлен на рисунке 2.1. Приложение имеет 3 текстовых окна: Ввод данных (1), Вывод данных (2), Статус (3). Также приложение имеет 5 кнопок: Вывести таблицу (4), Выполнить запрос (5), Добавить (6), Удалить (7), Обновить (8). Также приложение имеет 2 выпадающих меню: выбор таблицы (9), Выбор запроса (10). Внешний вид приложения с обозначением элементов представлен на рисунке 2.2.

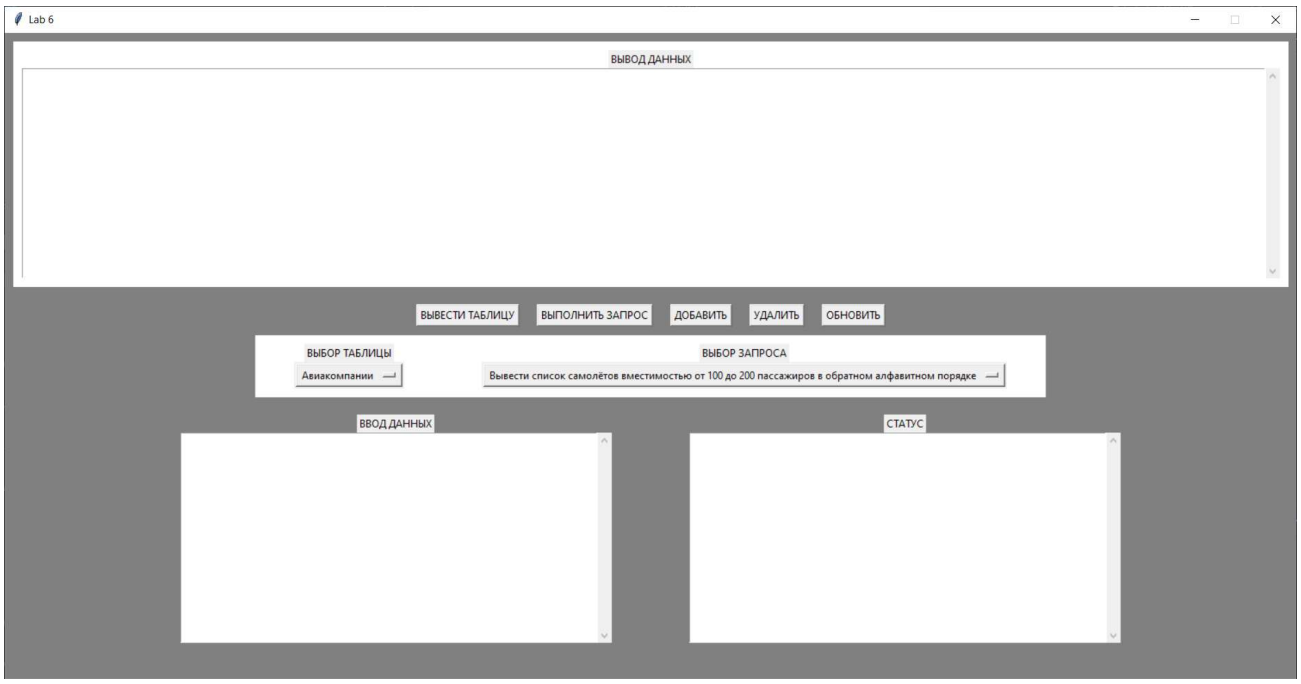

Рисунок 2.1 – Вид приложения при запуске

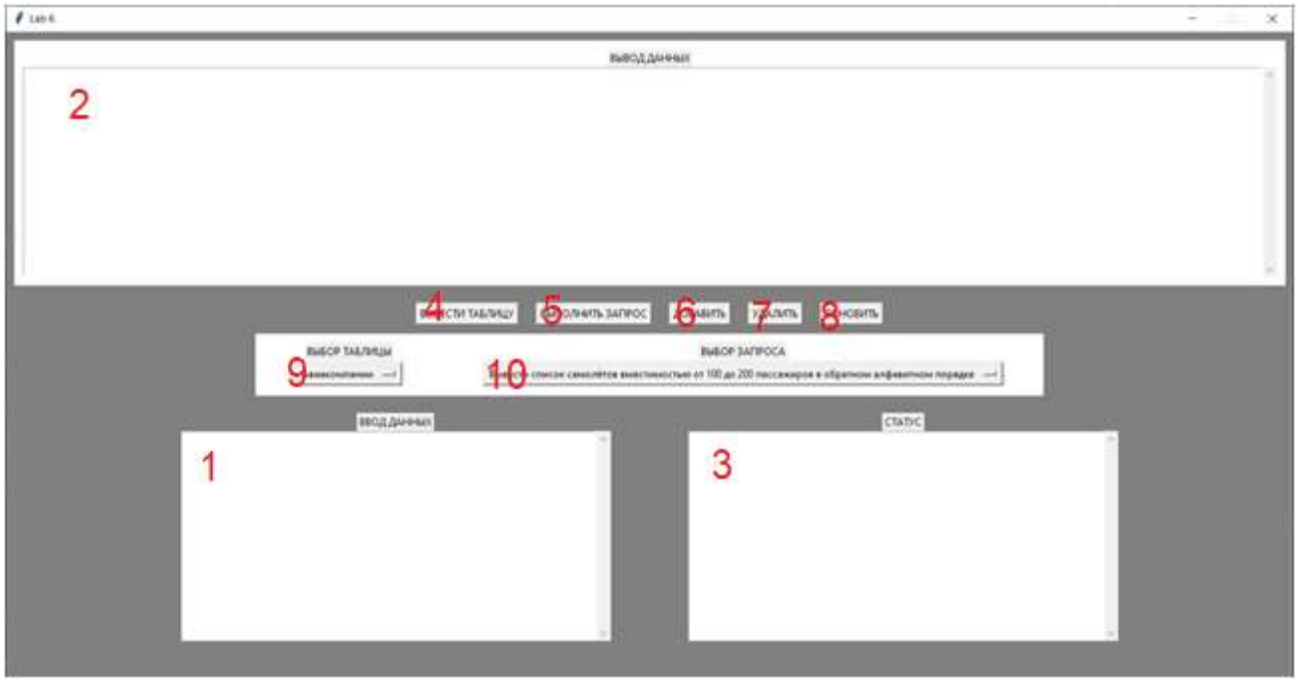

Рисунок 2.2 – Вид приложения при запуске с обозначением элементов

3 Руководство пользователя

3.1 Просмотр таблицы

Для просмотра таблицы необходимо выбрать таблицу в меню “Выбор таблицы” (9), затем нажать кнопку “Вывести таблицу” (4). Данные будут выведены в окно “Вывод данных” (2). Операция просмотра таблицы показана на рисунке 3.1.

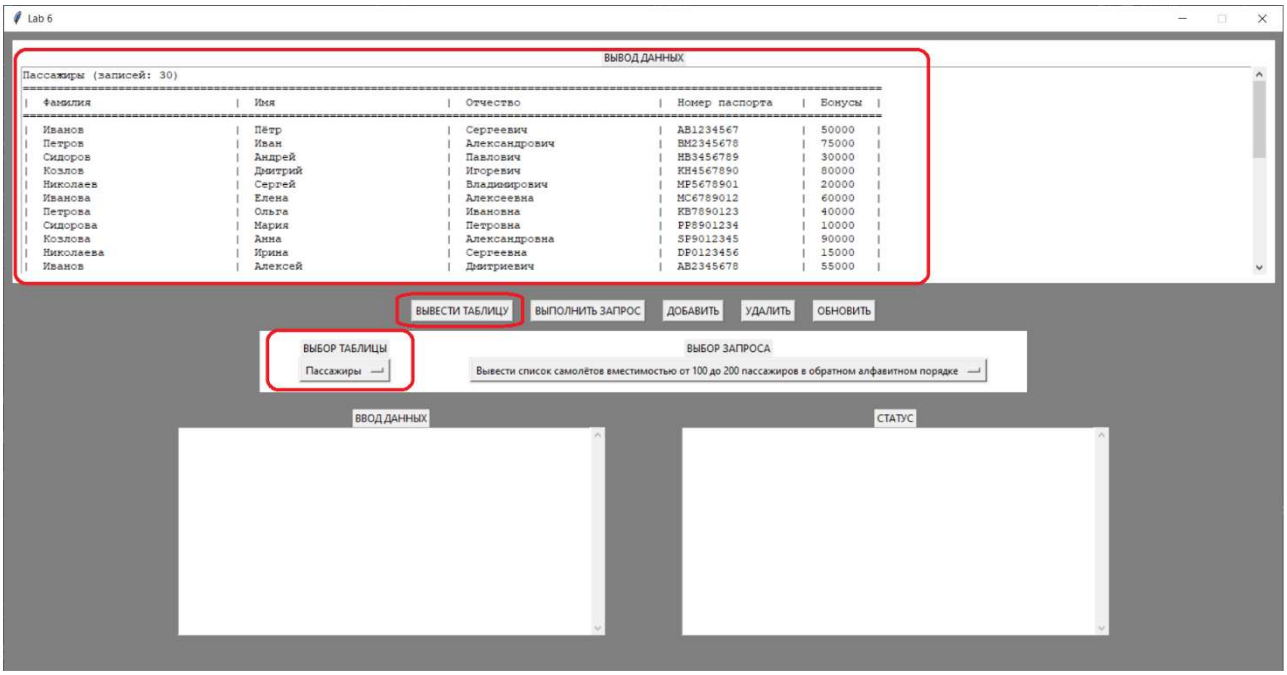

Рисунок 3.1 – Просмотр таблицы “Пассажиры”

3.2 Добавление записи в таблицу

Для добавления записи в таблицу необходимо выбрать таблицу в меню “Выбор таблицы” (9), затем ввести данные в окно “Ввод данных” (1). Данные необходимо вводить в порядке их вывода из таблицы в окне 1, значение каждого столбца на новой строке. Затем необходимо нажать кнопку “Добавить” (6). При успешном добавлении данных в окне “Статус” (3) будет выведена информация об успешном добавлении. При ошибке добавления в окне “Статус” (3) будет выведена информация о причине ошибки. Операция добавления записи показана на рисунках 3.2 – 3.4.

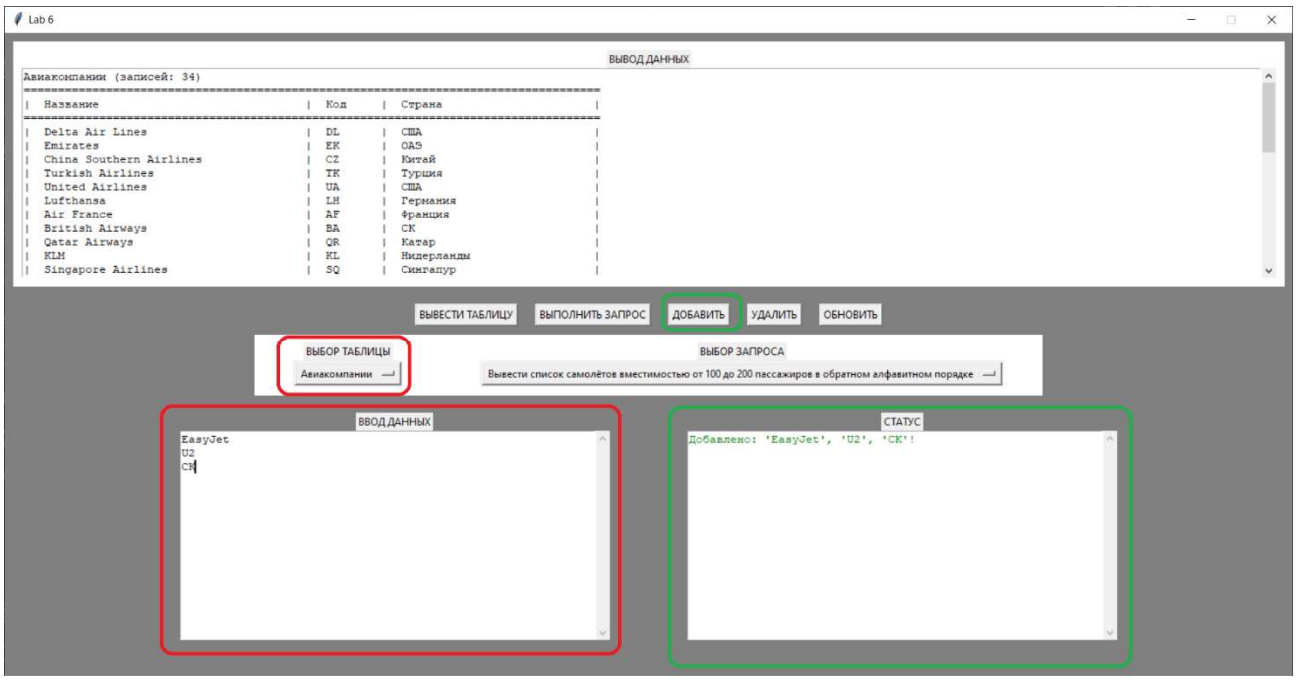

Рисунок 3.2 – Успешное добавление записи в таблицу “Авиакомпании”

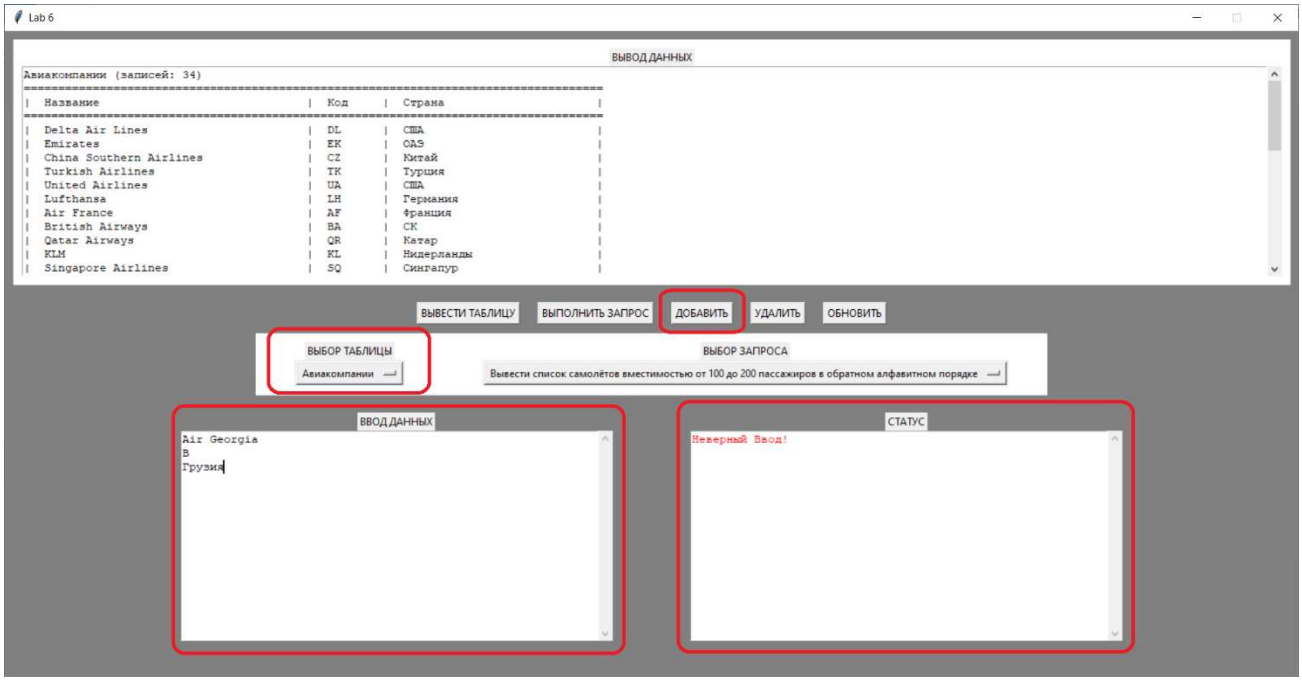

Рисунок 3.3 – Ошибка добавления записи в таблицу “Авиакомпании”: Неверный ввод

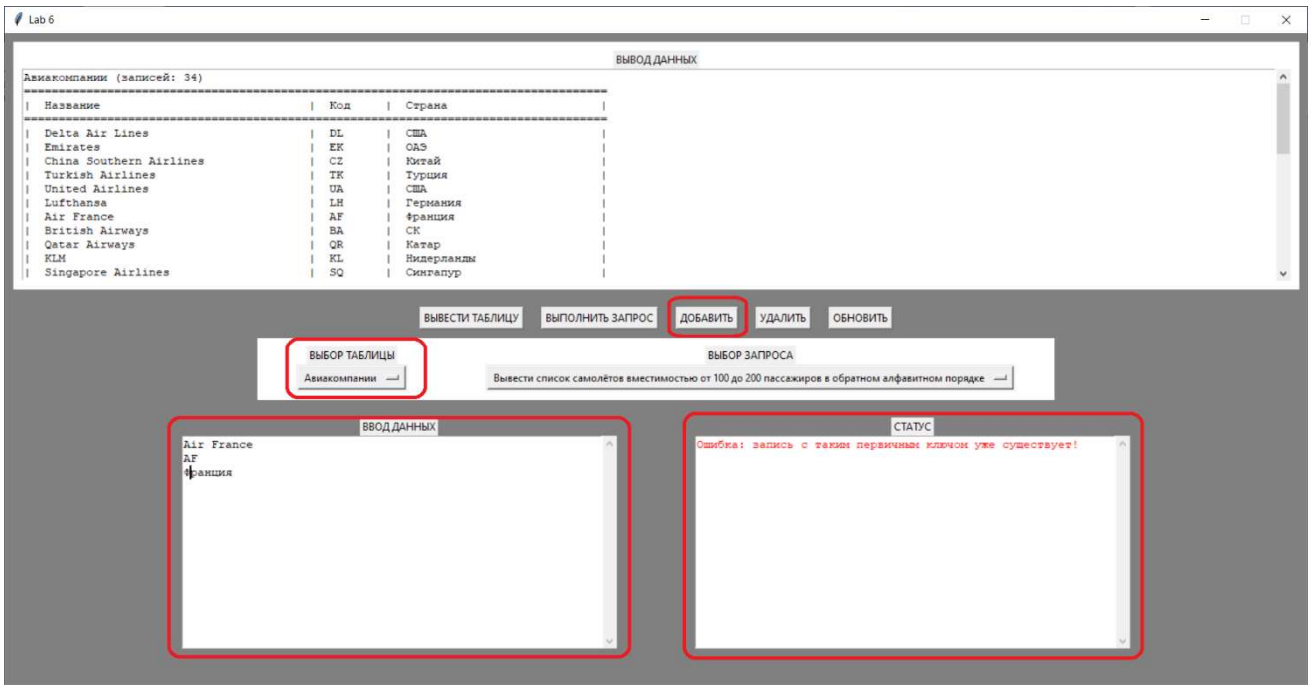

Рисунок 3.4 – Ошибка добавления записи в таблицу “Авиакомпании”: нарушение уникальности первичного ключа

3.3 Удаление записи из таблицы

Для удаления записи из таблицы необходимо выбрать таблицу в меню “Выбор таблицы” (9), затем ввести данные в окно “Ввод данных” (1). Необходимо вводить значение уникального столбца (столбцов). Затем необходимо нажать кнопку “Удалить” (7). При успешном удалении данных в окне “Статус” (3) будет выведена информация об успешном удалении. При ошибке удаления в окне “Статус” (3) будет выведена информация о причине ошибки. Операция удаления записи показана на рисунках 3.5 – 3.7.

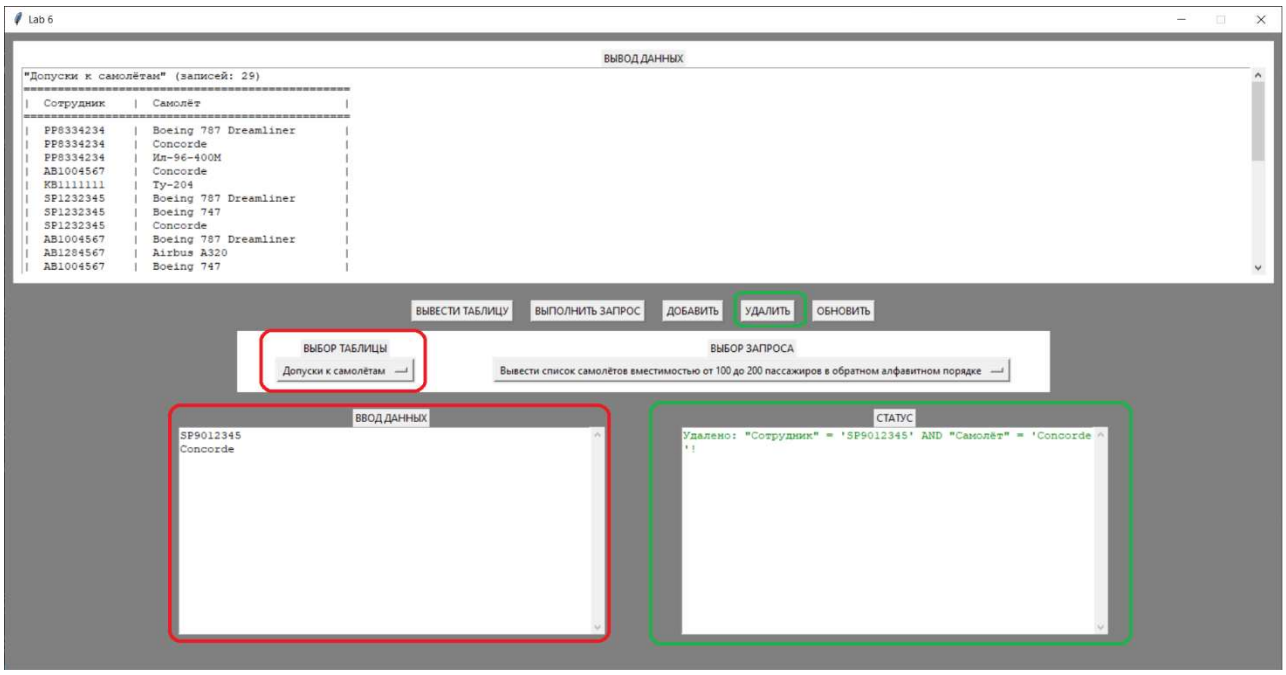

Рисунок 3.5 – Успешное удаление записи из таблицы “Допуски к самолётам”

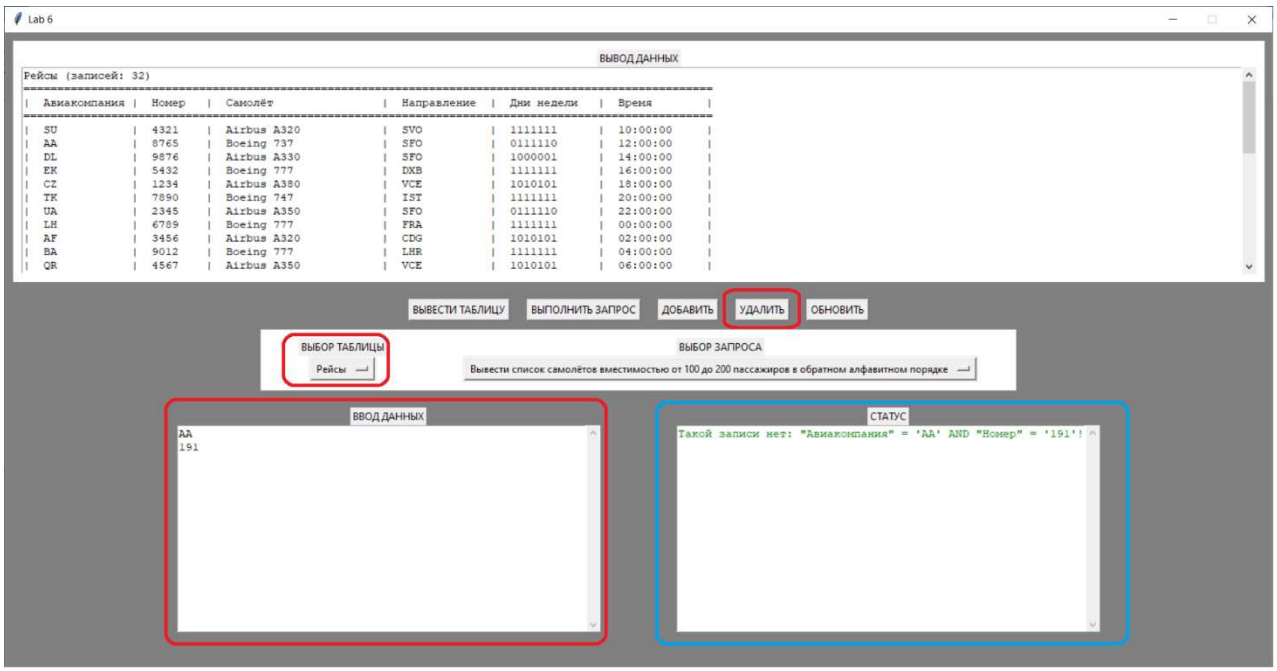

Рисунок 3.6 – Ошибка удаления записи из таблицы “Рейсы”: нет такой записи

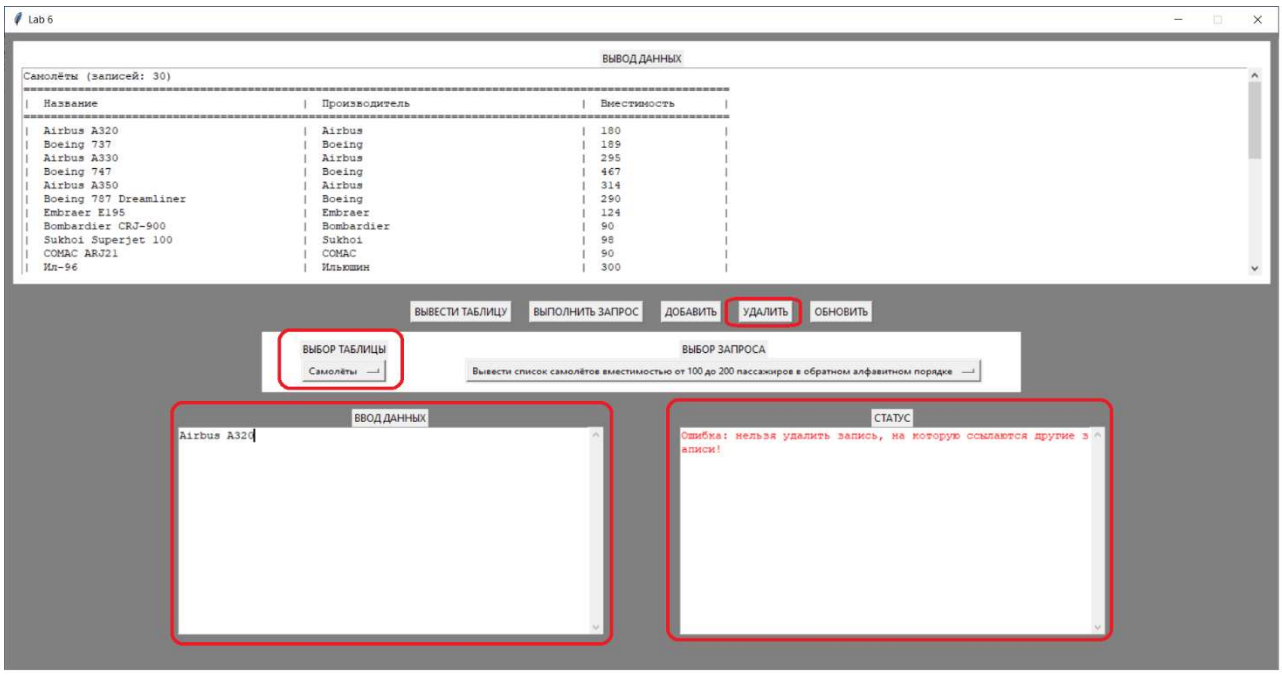

Рисунок 3.7 – Ошибка удаления записи из таблицы “Самолёты”: на запись ссылаются другие записи

3.4 Обновление записи в таблице

Для обновления записи в таблице необходимо выбрать таблицу в меню “Выбор таблицы” (9), затем ввести данные в окно “Ввод данных” (1). Данные необходимо вводить в порядке: Название столбца, в котором необходимо изменить данные; Новые данные; Значения уникальных столбцов. Затем необходимо нажать кнопку “Обновить” (8). При успешном обновлении данных в окне “Статус” (3) будет выведена информация об успешном обновлении. При ошибке добавления в окне “Статус” (3) будет выведена информация о причине ошибки. Операция обновления записи показана на рисунках 3.8 – 3.10.

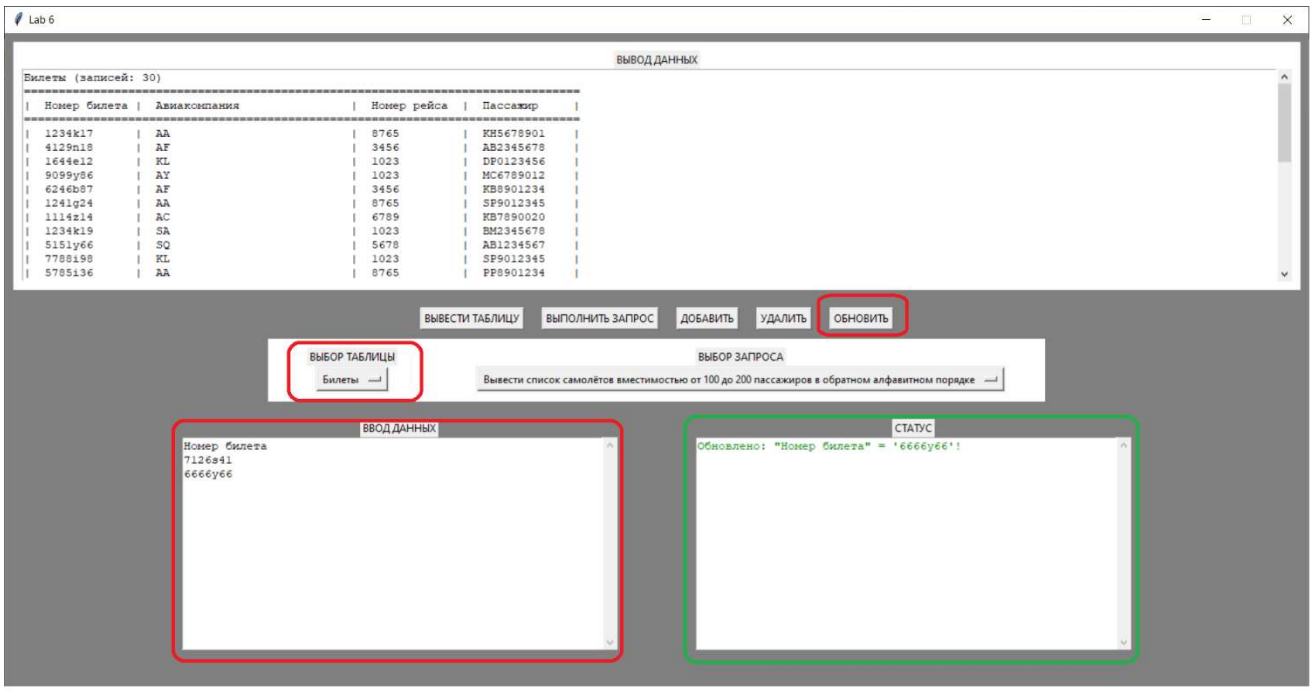

Рисунок 3.8 – Успешное обновление записи в таблице “Билеты”

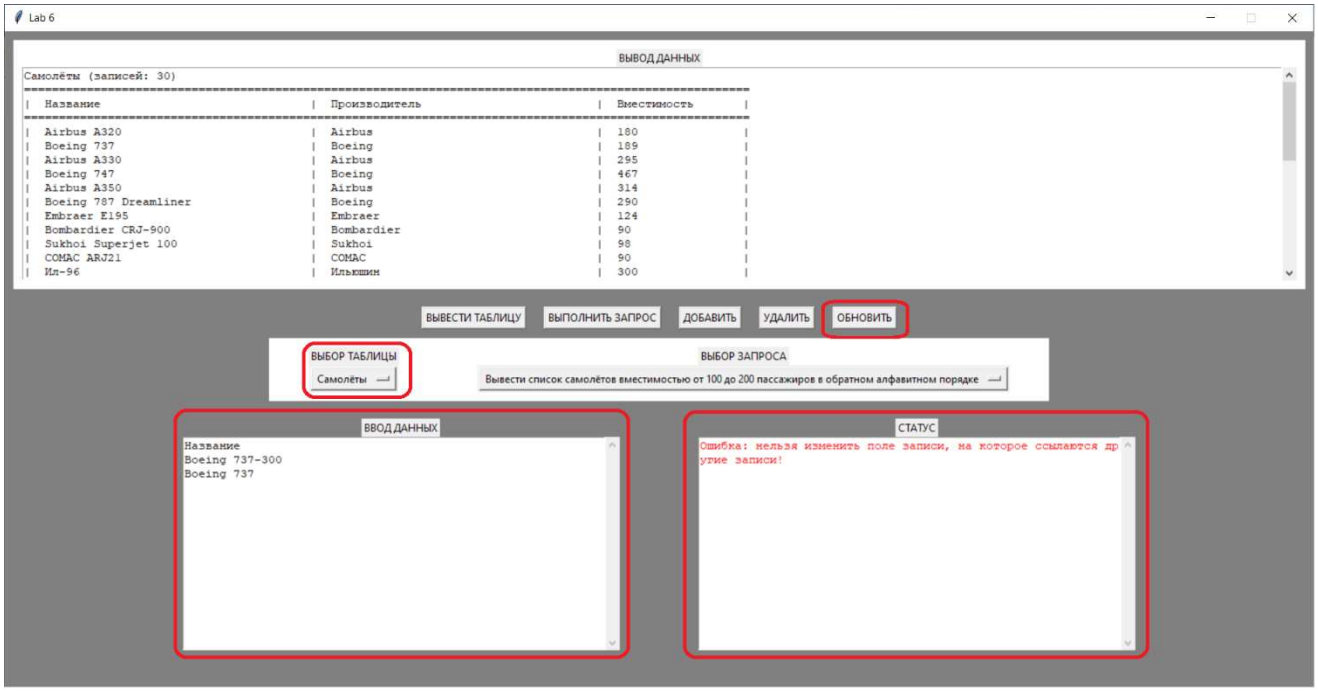

Рисунок 3.9 – Ошибка обновления записи в таблице “Самолёты”: на поле ссылаются другие записи

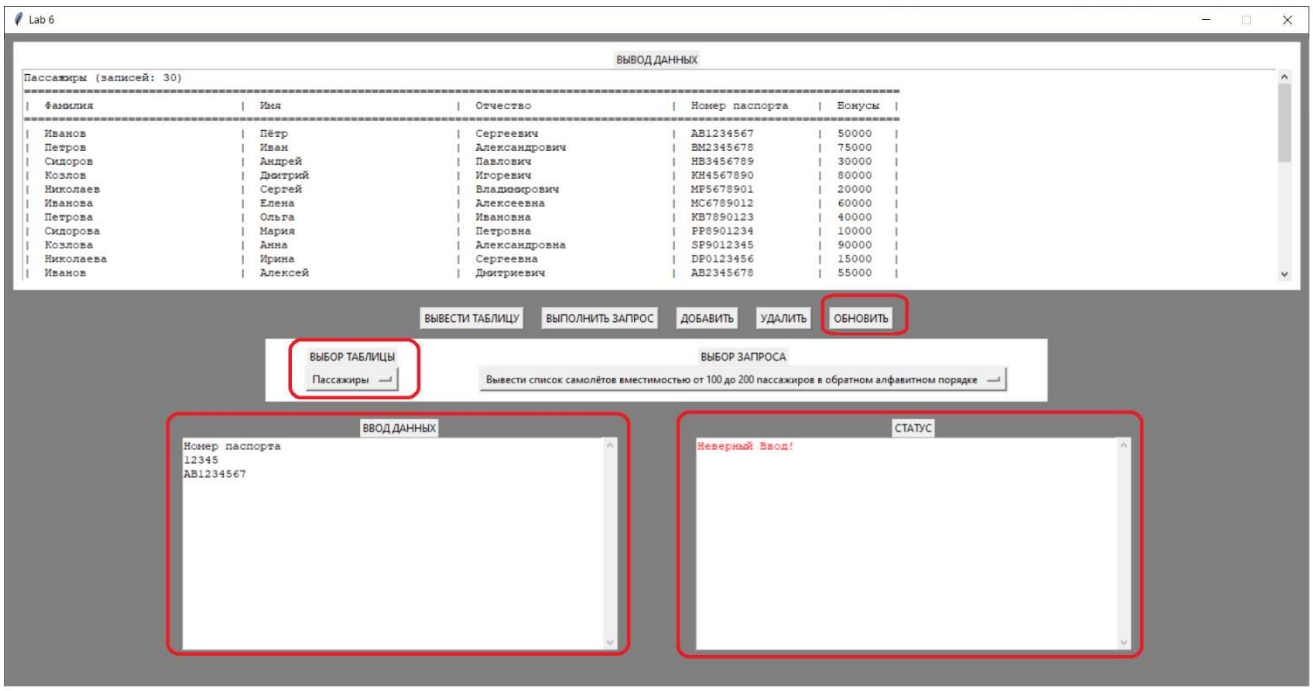

Рисунок 3.10 – Ошибка обновления записи в таблице “Пассажиры”: неверный ввод

3.5 Выполнение SQL запроса

Для выполнения SQL запроса необходимо выбрать запрос в меню “Выбор запроса” (10), затем нажать кнопку “Выполнить запрос” (5). Результат выполнения запроса и текст запроса будут выведены в окно “Вывод данных” (2). Примеры выполнения запросов приведены на рисунках 3.11 – 3.13.

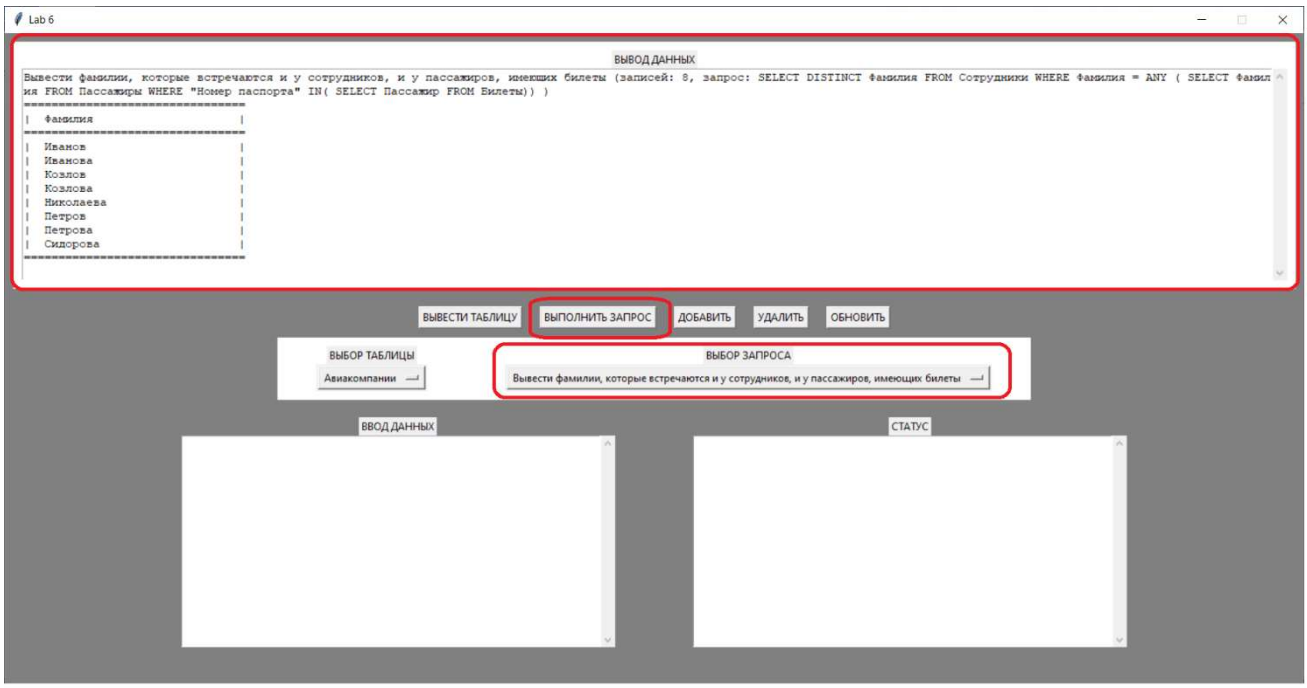

Рисунок 3.11 – Выполнение запроса “Вывести фамилии, которые встречаются и у сотрудников, и у пассажиров, имеющих билеты”

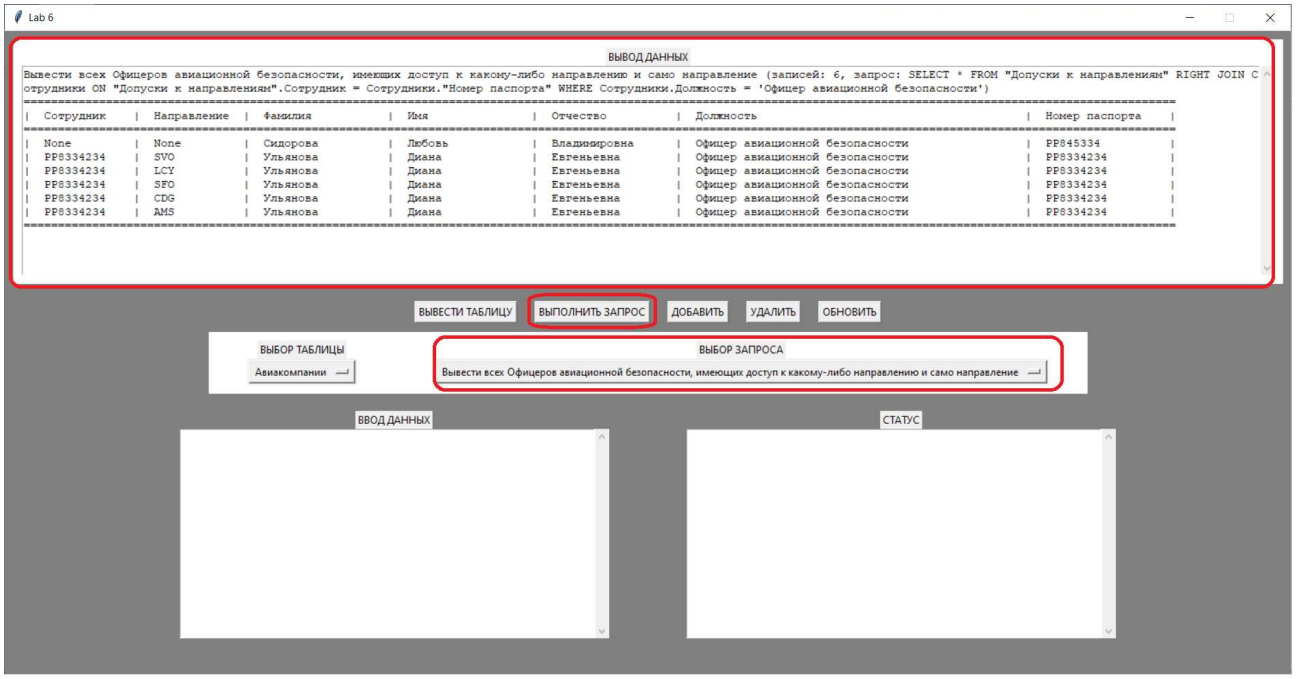

Рисунок 3.12 – Выполнение запроса “Вывести всех Офицеров авиационной безопасности, имеющих доступ к какому-либо направлению и само направление”

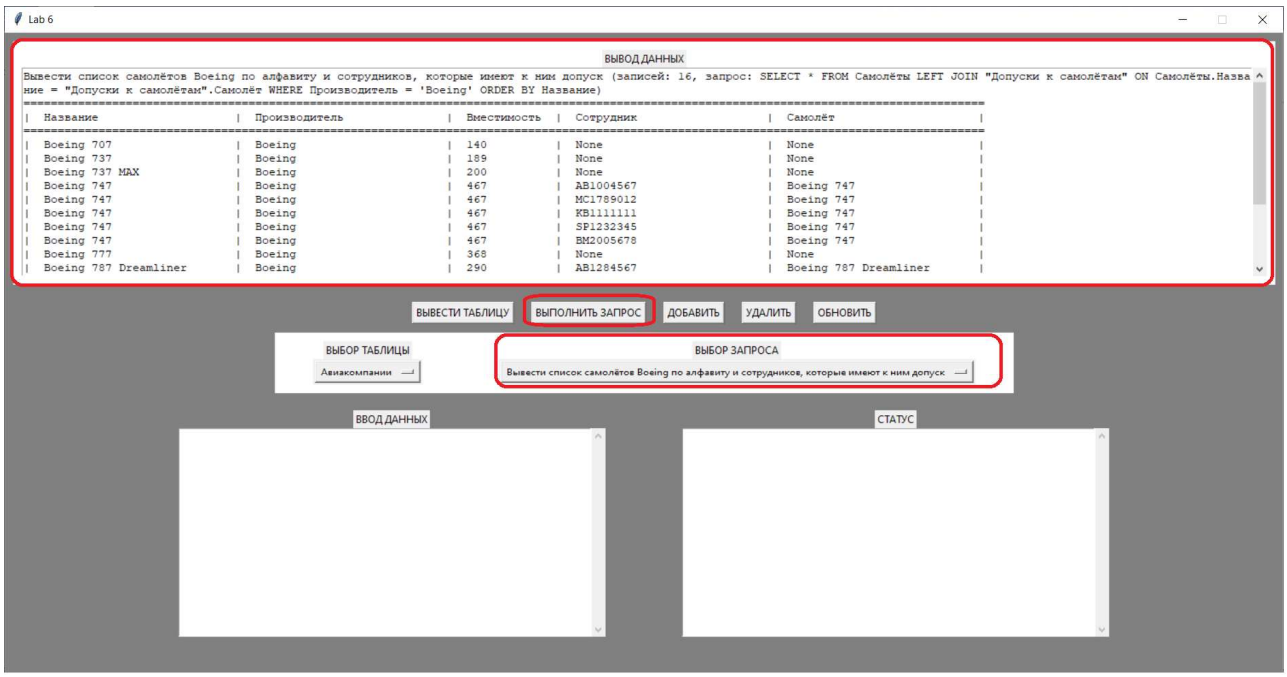

Рисунок 3.13 – Выполнение запроса “Вывести список самолётов Boeing по алфавиту и сотрудников, которые имеют к ним допуск”

4 Листинг кода

Листинг файла application.py (создаёт графический интерфейс и выполняет основные функции):

```
import psycopg2
import string
from tkinter import *
from tables_params import *
from queries_params import *
from validators import *

conn = ''
cursor = ''

def db_connect():
  global conn
  global cursor
  try:
```

```
        conn = psycopg2.connect(
            database="airport",
            user="postgres",
            password="123",
            host="localhost",
            port="5432"
        )
        cursor = conn.cursor()
        print("Successfully connected to PostgreSQL")
    except (Exception, psycopg2.Error) as error:
        print("Error while connecting to PostgreSQL", error)

def on_closing():
    global conn
    finish = 0
    try:
        conn.close()
        print("Successfully disconnected from PostgreSQL")
    except Exception:
        pass
    root.destroy()

def add_data():
    global conn
    global cursor
    goodparam = 1
    lines = field1.get(1.0, END)
    data = [line for line in lines.split('\n') if line]
    table_data = getTableParams(selected_value1.get())
    if not len(data) == table_data[-1]:
        goodparam = 0
    q = 0
    if goodparam == 1:
        for z in range(0, table_data[-1]):
```

```
if table_data[z + 1][3] == 'st':
  q += 1
elif table_data[z + 1][3] == 's2':
  res = s2_val(data[q])
  if res:
    q += 1
  else:
    goodparam = 0
    break
elif table_data[z + 1][3] == 's3':
  res = s3_val(data[q])
  if res:
    q += 1
  else:
    goodparam = 0
    break
elif table_data[z + 1][3] == 'nu':
  res = nu_val(data[q])
  if res:
    q += 1
  else:
    goodparam = 0
    break
elif table_data[z + 1][3] == 'n4':
  res = n4_val(data[q])
  if res:
    q += 1
  else:
    goodparam = 0
    break
elif table_data[z + 1][3] == 'fl':
  res = fl_val(data[q])
  if res:
    q += 1
```

```
      else:
        goodparam = 0
        break
    elif table_data[z + 1][3] == 'tm':
      res = tm_val(data[q])
      if res:
        q += 1
      else:
        goodparam = 0
        break
    elif table_data[z + 1][3] == 'np':
      res = np_val(data[q])
      if res:
        q += 1
      else:
        goodparam = 0
        break
    elif table_data[z + 1][3] == 'nt':
      res = nt_val(data[q])
      if res:
        q += 1
      else:
        goodparam = 0
        break
    elif table_data[z + 1][3] == 'wd':
      res = wd_val(data[q])
      if res:
        q += 1
      else:
        goodparam = 0
        break
if goodparam == 1:
  stringtoadd = ''
  for datum in data:
```

```
      stringtoadd += '\'' + str(datum) + '\', '
      last_comma_index = stringtoadd.rfind(",")
    if last_comma_index != -1:
      stringtoadd = stringtoadd[:last_comma_index]
    try:
      cursor.execute(f"INSERT INTO {table_data[0]} VALUES
({stringtoadd});")
      conn.commit()
      show_data()
      field2.config(fg="green")
      field2.config(state=NORMAL)
      field2.delete("1.0", "end")
      field2.insert("1.0", f"Добавлено: {stringtoadd}!")
      field2.config(state=DISABLED)
    except (psycopg2.errors.UniqueViolation):
      field2.config(fg="red")
      field2.config(state=NORMAL)
      field2.delete("1.0", "end")
      field2.insert("1.0", "Ошибка: запись с таким первичным
ключом уже существует!")
      field2.config(state=DISABLED)
      try:
        conn.close()
        db_connect()
      except Exception:
        pass
    except (psycopg2.errors.ForeignKeyViolation):
      field2.config(fg="red")
      field2.config(state=NORMAL)
      field2.delete("1.0", "end")
      field2.insert("1.0", "Ошибка: внешний ключ не найден!")
      field2.config(state=DISABLED)
      try:
        conn.close()
        db_connect()
```

```
        except Exception:
          pass
    else:
      field2.config(fg="red")
      field2.config(state=NORMAL)
      field2.delete("1.0", "end")
      field2.insert("1.0", "Неверный Ввод!")
      field2.config(state=DISABLED)

def upd_data():
    global conn
    global cursor
    goodparam = 1
    keyscount = 0
    keystuple = ()
    lines = field1.get(1.0, END)
    data = [line for line in lines.split('\n') if line]
    table_data = getTableParams(selected_value1.get())
    for d in range (0, table_data[-1]):
      if table_data[d + 1][0] == data[0]:
        keyscount += 1
        keystuple = keystuple + (table_data[d + 1],)
    if not keyscount > 0:
      goodparam = 0
    if goodparam == 1:
      for z in range(0, keyscount):
        if keystuple[z][3] == 'st':
          pass
        elif keystuple[z][3] == 's2':
          res = s2_val(data[1])
          if res:
            pass
          else:
            goodparam = 0
```

```
    break
elif keystuple[z][3] == 's3':
  res = s3_val(data[1])
  if res:
    pass
  else:
    goodparam = 0
    break
elif keystuple[z][3] == 'nu':
  res = nu_val(data[1])
  if res:
    pass
  else:
    goodparam = 0
    break
elif keystuple[z][3] == 'n4':
  res = n4_val(data[1])
  if res:
    pass
  else:
    goodparam = 0
    break
elif keystuple[z][3] == 'fl':
  res = fl_val(data[1])
  if res:
    pass
  else:
    goodparam = 0
    break
elif keystuple[z][3] == 'tm':
  res = tm_val(data[1])
  if res:
    pass
  else:
```

```
        goodparam = 0
        break
    elif keystuple[z][3] == 'np':
      res = np_val(data[1])
      if res:
        pass
      else:
        goodparam = 0
        break
    elif keystuple[z][3] == 'nt':
      res = nt_val(data[1])
      if res:
        pass
      else:
        goodparam = 0
        break
    elif keystuple[z][3] == 'wd':
      res = wd_val(data[1])
      if res:
        pass
      else:
        goodparam = 0
        break
if goodparam == 1:
  keyscount = 0
  keystuple = ()
  for d in range (0, table_data[-1]):
    if table_data[d + 1][2] == 1:
      keyscount += 1
      keystuple = keystuple + (table_data[d + 1],)
  if not len(data) == (keyscount + 2):
    goodparam = 0
  q = 2
  if goodparam == 1:
```

```
for z in range(0, keyscount):
  if keystuple[z][3] == 'st':
    q += 1
  elif keystuple[z][3] == 's2':
    res = s2_val(data[q])
    if res:
      q += 1
    else:
      goodparam = 0
      break
  elif keystuple[z][3] == 's3':
    res = s3_val(data[q])
    if res:
      q += 1
    else:
      goodparam = 0
      break
  elif keystuple[z][3] == 'nu':
    res = nu_val(data[q])
    if res:
      q += 1
    else:
      goodparam = 0
      break
  elif keystuple[z][3] == 'n4':
    res = n4_val(data[q])
    if res:
      q += 1
    else:
      goodparam = 0
      break
  elif keystuple[z][3] == 'fl':
    res = fl_val(data[q])
    if res:
```

```
            q += 1
          else:
            goodparam = 0
            break
        elif keystuple[z][3] == 'tm':
          res = tm_val(data[q])
          if res:
            q += 1
          else:
            goodparam = 0
            break
        elif keystuple[z][3] == 'np':
          res = np_val(data[q])
          if res:
            q += 1
          else:
            goodparam = 0
            break
        elif keystuple[z][3] == 'nt':
          res = nt_val(data[q])
          if res:
            q += 1
          else:
            goodparam = 0
            break
        elif keystuple[z][3] == 'wd':
          res = wd_val(data[q])
          if res:
            q += 1
          else:
            goodparam = 0
            break
    if goodparam == 1:
      stringtodel = ''
```

```
    w = 0
    for datum in data:
      if w >= 2:
        stringtodel += '\"' + str(keystuple[w - 2][0]) + '\" =
\'' + str(datum) + '\' AND '
        last_and_index = stringtodel.rfind(" AND ")
      w += 1
    if last_and_index != -1:
      stringtodel = stringtodel[:last_and_index]
    stringtomod = '\"' + data[0] + '\" = \'' + data[1] + '\''
    try:
      cursor.execute(f"UPDATE {table_data[0]} SET {stringtomod}
WHERE {stringtodel}")
      del_count = cursor.rowcount
      conn.commit()
      show_data()
      field2.config(fg="green")
      field2.config(state=NORMAL)
      field2.delete("1.0", "end")
      if del_count > 0:
        field2.insert("1.0", f"Обновлено: {stringtodel}!")
      else:
        field2.insert("1.0", f"Такой записи нет:
{stringtodel}!")
      field2.config(state=DISABLED)
    except (psycopg2.errors.UniqueViolation):
      field2.config(fg="red")
      field2.config(state=NORMAL)
      field2.delete("1.0", "end")
      field2.insert("1.0", "Ошибка: запись с таким первичным
ключом уже существует!")
      field2.config(state=DISABLED)
      try:
        conn.close()
        db_connect()
```

```
      except Exception:
        pass
    except (psycopg2.errors.ForeignKeyViolation):
      field2.config(fg="red")
      field2.config(state=NORMAL)
      field2.delete("1.0", "end")
      field2.insert("1.0", "Ошибка: нельзя изменить поле записи,
на которое ссылаются другие записи!")
      field2.config(state=DISABLED)
      try:
        conn.close()
        db_connect()
      except Exception:
        pass
  else:
    field2.config(fg="red")
    field2.config(state=NORMAL)
    field2.delete("1.0", "end")
    field2.insert("1.0", "Неверный Ввод!")
    field2.config(state=DISABLED)


def del_data():
  global conn
  global cursor
  goodparam = 1
  keyscount = 0
  keystuple = ()
  lines = field1.get(1.0, END)
  data = [line for line in lines.split('\n') if line]
  table_data = getTableParams(selected_value1.get())
  for d in range (0, table_data[-1]):
    if table_data[d + 1][2] == 1:
      keyscount += 1
      keystuple = keystuple + (table_data[d + 1],)
```

```
if not len(data) == keyscount:
  goodparam = 0
q = 0
if goodparam == 1:
  for z in range(0, keyscount):
    if keystuple[z][3] == 'st':
      q += 1
    elif keystuple[z][3] == 's2':
      res = s2_val(data[q])
      if res:
        q += 1
      else:
        goodparam = 0
        break
    elif keystuple[z][3] == 's3':
      res = s3_val(data[q])
      if res:
        q += 1
      else:
        goodparam = 0
        break
    elif keystuple[z][3] == 'nu':
      res = nu_val(data[q])
      if res:
        q += 1
      else:
        goodparam = 0
        break
    elif keystuple[z][3] == 'n4':
      res = n4_val(data[q])
      if res:
        q += 1
      else:
        goodparam = 0
```

```
    break
elif keystuple[z][3] == 'fl':
  res = fl_val(data[q])
  if res:
    q += 1
  else:
    goodparam = 0
    break
elif keystuple[z][3] == 'tm':
  res = tm_val(data[q])
  if res:
    q += 1
  else:
    goodparam = 0
    break
elif keystuple[z][3] == 'np':
  res = np_val(data[q])
  if res:
    q += 1
  else:
    goodparam = 0
    break
elif keystuple[z][3] == 'nt':
  res = nt_val(data[q])
  if res:
    q += 1
  else:
    goodparam = 0
    break
elif keystuple[z][3] == 'wd':
  res = wd_val(data[q])
  if res:
    q += 1
  else:
```

```
            goodparam = 0
            break
    if goodparam == 1:
      stringtodel = ''
      w = 0
      for datum in data:
        stringtodel += '\"' + str(keystuple[w][0]) + '\" = \'' +
str(datum) + '\' AND '
        last_and_index = stringtodel.rfind(" AND ")
        w += 1
      if last_and_index != -1:
        stringtodel = stringtodel[:last_and_index]
      try:
        cursor.execute(f"DELETE FROM {table_data[0]} WHERE
{stringtodel};")
        del_count = cursor.rowcount
        conn.commit()
        show_data()
        field2.config(fg="green")
        field2.config(state=NORMAL)
        field2.delete("1.0", "end")
        if del_count > 0:
          field2.insert("1.0", f"Удалено: {stringtodel}!")
        else:
          field2.insert("1.0", f"Такой записи нет:
{stringtodel}!")
        field2.config(state=DISABLED)
      except (psycopg2.errors.ForeignKeyViolation):
        field2.config(fg="red")
        field2.config(state=NORMAL)
        field2.delete("1.0", "end")
        field2.insert("1.0", "Ошибка: нельзя удалить запись, на
которую ссылаются другие записи!")
        field2.config(state=DISABLED)
        try:
```

```
            conn.close()
            db_connect()
        except Exception:
            pass
    else:
        field2.config(fg="red")
        field2.config(state=NORMAL)
        field2.delete("1.0", "end")
        field2.insert("1.0", "Неверный Ввод!")
        field2.config(state=DISABLED)

def show_data():
            global conn
            global cursor
            table = getTableParams(selected_value1.get())
            cursor.execute(f"SELECT * FROM {table[0]};")
            rows = cursor.fetchall()
            i = 1
            mtext = ""
            j = table[-1]
            ftup = ()
            sumlen = 0
            counter = 0
            for m in range(j):
                ftup = ftup + (table[m + 1][0], table[m + 1][1],)
                sumlen += table[m + 1][1]
                counter += 1
            sumlen += counter + 1
            mtext += table[0] + ' (записей: ' + str(len(rows)) +
')\n'
            for x in range (sumlen):
                mtext += '='
            mtext += '\n'
            mtext += format_string_from_tuple(ftup)
```

```
        mtext += '\n'
        for x in range (sumlen):
            mtext += '='
        mtext += '\n'
        for row in rows:
            newtup = ()
            for k in range (j):
                newtup = newtup + (row[k], table[k + 1][1],)
            mtext += format_string_from_tuple(newtup)
            mtext += '\n'
            i += 1
        for x in range (sumlen):
            mtext += '='
        field.config(state=NORMAL)
        field.delete("1.0", "end")
        field.insert("1.0", mtext)
        field.config(state=DISABLED)

def ex_query():
        global conn
        global cursor
        query_to_ex = get_query_params(selected_value2.get())
        cursor.execute(query_to_ex[0][1])
        rows = cursor.fetchall()
        i = 1
        mtext = ""
        j = query_to_ex[-1]
        ftup = ()
        sumlen = 0
        counter = 0
        for m in range(j):
            ftup = ftup + (query_to_ex[m + 1][0], query_to_ex[m
+ 1][1],)
            sumlen += query_to_ex[m + 1][1]
```

```
            counter += 1
        sumlen += counter + 1
        mtext += query_to_ex[0][0] + ' (записей: ' +
str(len(rows)) + ', запрос: ' + query_to_ex[0][1] + ')\n'
        for x in range (sumlen):
            mtext += '='
        mtext += '\n'
        mtext += format_string_from_tuple(ftup)
        mtext += '\n'
        for x in range (sumlen):
            mtext += '='
        mtext += '\n'
        for row in rows:
            newtup = ()
            for k in range (j):
                newtup = newtup + (row[k], query_to_ex[k +
1][1],)
            mtext += format_string_from_tuple(newtup)
            mtext += '\n'
            i += 1
        for x in range (sumlen):
            mtext += '='
        field.config(state=NORMAL)
        field.delete("1.0", "end")
        field.insert("1.0", mtext)
        field.config(state=DISABLED)



########################################################################
####


root = Tk()
root.title("Lab 6")
root.geometry("1500x750")
root.resizable(False, False)
```

```
root.config(bg="gray")

root.protocol("WM_DELETE_WINDOW", on_closing)

label = Frame(root, bg="white", padx=10, pady=10)
label.grid(row=0, column=0, padx=10, pady=10)
Label(label, text="ВЫВОД ДАННЫХ").pack()
field = Text(label, width=180, height=15)
field.pack(side=LEFT)
scrollbar = Scrollbar(label)
scrollbar.pack(side=RIGHT, fill=Y)
field.config(yscrollcommand=scrollbar.set)
scrollbar.config(command=field.yview)
field.config(state=DISABLED)

buttonsstr = Frame(root, bg="gray", padx=45, pady=0)
buttonsstr.grid(row=1, column=0)

button1 = Button(buttonsstr, text="ВЫВЕСТИ ТАБЛИЦУ",
command=show_data)
button1.grid(row=0, column=0, padx=10, pady=10)

button2 = Button(buttonsstr, text="ВЫПОЛНИТЬ ЗАПРОС",
command=ex_query)
button2.grid(row=0, column=1, padx=10, pady=10)

button3 = Button(buttonsstr, text="ДОБАВИТЬ", command=add_data)
button3.grid(row=0, column=2, padx=10, pady=10)

button4 = Button(buttonsstr, text="УДАЛИТЬ", command=del_data)
button4.grid(row=0, column=3, padx=10, pady=10)

button5 = Button(buttonsstr, text="ОБНОВИТЬ", command=upd_data)
button5.grid(row=0, column=4, padx=10, pady=10)
```

```
selected_value1 = StringVar()

selected_value1.set("Авиакомпании")

selected_value2 = StringVar()

selected_value2.set("Вывести список самолётов вместимостью от
100 до 200 пассажиров в обратном алфавитном порядке")


options1 = ["Авиакомпании", "Билеты", "Должности", "Допуски к
направлениям", "Допуски к самолётам",

            "Направления", "Пассажиры", "Рейсы", "Самолёты",
"Сотрудники"]


options2 = ["Вывести список самолётов вместимостью от 100 до 200
пассажиров в обратном алфавитном порядке",

            "Вывести список билетов с указанием информации об
авиакомпании, направлении и паспорте пассажира. Сортировать по
названию авиакомпании",

            "Вывести названия авиакомпаний из США",

            "Вывести названия всех авиакомпаний по алфавиту,
которые имеют рейсы",

            "Вывести список сотрудников, которые имеют доступ к
самолёту Concorde",

            "Вывести все должности с окладом более 80000.
Сортировать по убыванию оклада",

            "Вывести всех пассажиров, у которых номер паспорта
начинается с AB",

            "Вывести номера рейсов, которые вылетают по пятницам
позже 13.00 но раньше 18.00",

            "Вывести список всех направлений в Швеции и рейсы на
это направление",

            "Вывести список самолётов Boeing по алфавиту и
сотрудников, которые имеют к ним допуск",

            "Вывести в верхнем регистре ФИО всех помощников
пилота",

            "Вывести список пассажиров, у которых от 10000 до
15000 бонусов и фамилией Сидоров/Сидорова",

            "Вывести список пассажиров, которые купили билет на
рейс авиакомпании Belavia (B2)",

            "Вывести всех Офицеров авиационной безопасности,
имеющих доступ к какому-либо направлению и само направление",

            "Вывести в верхнем регистре названия должностей, на
которых есть сотрудники с допуском к самолёту Airbus A320",
```

```
            "Вывести страны, у которых более 1 направления с
билетами и количество этих направлений",

            "Вывести самолёты вместимостью более 300 от
производителей, у которых есть самолёты вместимостью менее 150,
к которым имеет доступ хотя бы 1 сотрудник и у которых есть
рейсы",

            "Вывести фамилии, которые встречаются и у
сотрудников, и у пассажиров, имеющих билеты",

            "Вывести имена, которые встречаются у сотрудников с
должностью, содержащей слово техник, но не у пассажиров",

            "Вывести самолёты вместимостью более 400, их среднюю
вместимость и количество сотрудников, имеющих допуск к этим
самолётам и к любому направлению",

            "Вывести фамилии пассажиров, у которых (фамилий)
есть более 1 билета и у пассажиров с этой фамилией бонусов
больше, чем среднее расстояние до направлений",

            "Вывести всех сотрудников, у которых есть допуск и к
направлению, и к самолёту",

            "Вывести минимальный и максимальный номера билетов,
которые вылетают по пятницам от 08.00 до 18.00 по направлению,
которое находится дальше любого направления в России",

            "Вывести список всех авиакомпаний, в чьих странах
есть направления и есть рейс с номером меньше среднего номера",

            "Вывести сотрудников, имеющих допуски и к самолётам,
и к направлениям, и у которых количество допусков к направлениям
больше среднего количества допусков к самолётам",

            "Вывести названия должностей, которые начинаются с
‘A’ и не имеют сотрудников и названия направлений, которые
начинаются с ‘A’",

            "Вывести сумму расстояний до направлений, сумму
окладов и сумму количеств бонусов",

            "Вывести названия должностей, на которых нет
сотрудников, не имеющих допуска к самолёту",

            "Вывести рейсы, у которых совпадают страна
авиакомпании и страна направления",

            "Вывести время вылета рейсов, которые выполняет
авиакомпания, у которой есть билеты пассажиров с номером
паспорта на ‘AB’"]


str2 = Frame(root, bg="gray", padx=45, pady=0)

str2.grid(row=2, column=0)
```

```
str2_1 = Frame(str2, bg="white", padx=45, pady=10)
str2_1.grid(row=0, column=0)
Label(str2_1, text="ВЫБОР ТАБЛИЦЫ").pack()
dropdown1 = OptionMenu(str2_1, selected_value1, *options1)
dropdown1.pack()

str2_2 = Frame(str2, bg="white", padx=45, pady=10)
str2_2.grid(row=0, column=1)
Label(str2_2, text="ВЫБОР ЗАПРОСА").pack()
dropdown2 = OptionMenu(str2_2, selected_value2, *options2)
dropdown2.pack()

container3 = Frame(root, bg="gray", padx=45, pady=10)
container3.grid(row=3, column=0)

str3 = Frame(container3, bg="gray", padx=45, pady=10)
str3.grid(row=0, column=0)

lab1 = Label(str3, text="ВВОД ДАННЫХ").pack()
field1 = Text(str3, width=60, height=15)
field1.pack(side=LEFT)
scrollbar1 = Scrollbar(str3)
scrollbar1.pack(side=RIGHT, fill=Y)
field1.config(yscrollcommand=scrollbar1.set)
scrollbar1.config(command=field1.yview)

str4 = Frame(container3, bg="gray", padx=45, pady=10)
str4.grid(row=0, column=1)

lab2 = Label(str4, text="СТАТУС").pack()
field2 = Text(str4, width=60, height=15)
field2.pack(side=LEFT)
scrollbar2 = Scrollbar(str4)
scrollbar2.pack(side=RIGHT, fill=Y)
```

```
field2.config(yscrollcommand=scrollbar2.set)
scrollbar2.config(command=field2.yview)
field2.config(state=DISABLED)

db_connect()

root.mainloop()

##################################################################
####
```

Листинг файла tables_params.py (хранит параметры таблиц):

```
def getTableParams(name):
    if name == 'Авиакомпании':
        return airlines
    elif name == 'Билеты':
        return tickets
    elif name == 'Должности':
        return jobs
    elif name == 'Допуски к направлениям':
        return destinationaccess
    elif name == 'Допуски к самолётам':
        return airplaneaccess
    elif name == 'Направления':
        return destinations
    elif name == 'Пассажиры':
        return passengers
    elif name == 'Рейсы':
        return flights
    elif name == 'Самолёты':
        return airplanes
    elif name == 'Сотрудники':
        return employees
    else:
```

```
        return airlines

# st - string +
# s2 - string(2) +
# nt - ticket number +
# n4 - number < 10000 +
# np - passport number +
# fl - float +
# s3 - string(3) +
# nu - number +
# wd - days of week +
# tm - time +

airlines = (
    'Авиакомпании',
    ('Название', 40, 0, 'st'),
    ('Код', 10, 1, 's2'),
    ('Страна', 30, 0, 'st'),
    3
)

tickets = (
    'Билеты',
    ('Номер билета', 15, 1, 'nt'),
    ('Авиакомпания', 30, 0, 's2'),
    ('Номер рейса', 15, 0, 'n4'),
    ('Пассажир', 15, 0, 'np'),
    4
)

jobs = (
    'Должности',
    ('Название', 50, 1, 'st'),
    ('Оклад', 20, 0, 'fl'),
```

```
    ('Описание', 105, 0, 'st'),
    3
)

destinationaccess = (
    '"Допуски к направлениям"',
    ('Сотрудник', 15, 1, 'np'),
    ('Направление', 15, 1, 's3'),
    2
)

airplaneaccess = (
    '"Допуски к самолётам"',
    ('Сотрудник', 15, 1, 'np'),
    ('Самолёт', 30, 1, 'st'),
    2
)

destinations = (
    'Направления',
    ('Название', 30, 0, 'st'),
    ('Код', 10, 1, 's3'),
    ('Страна', 30, 0, 'st'),
    ('Расстояние', 20, 0, 'nu'),
    4
)

passengers = (
    'Пассажиры',
    ('Фамилия', 30, 0, 'st'),
    ('Имя', 30, 0, 'st'),
    ('Отчество', 30, 0, 'st'),
    ('Номер паспорта', 20, 1, 'np'),
    ('Бонусы', 10, 0, 'nu'),
```

```
    5
)

flights = (
    'Рейсы',
    ('Авиакомпания', 15, 1, 's2'),
    ('Номер', 10, 1, 'n4'),
    ('Самолёт', 25, 0, 'st'),
    ('Направление', 15, 0, 's3'),
    ('Дни недели', 15, 0, 'wd'),
    ('Время', 15, 0, 'tm'),
    6
)

airplanes = (
    'Самолёты',
    ('Название', 40, 1, 'st'),
    ('Производитель', 40, 0, 'st'),
    ('Вместимость', 20, 0, 'nu'),
    3
)

employees = (
    'Сотрудники',
    ('Фамилия', 30, 0, 'st'),
    ('Имя', 30, 0, 'st'),
    ('Отчество', 30, 0, 'st'),
    ('Должность', 50, 0, 'st'),
    ('Номер паспорта', 20, 1, 'np'),
    5
)
```

Листинг файла queries_params.py (хранит параметры запросов):

```
def get_query_params(name):
    for query in queries:
        if query[0][0] == name:
            return query
    return queries[0]


queries = (
    (("Вывести список самолётов вместимостью от 100 до 200 пассажиров в обратном алфавитном порядке",
    "SELECT * FROM Самолёты WHERE Вместимость >= 100 AND Вместимость <= 200 ORDER BY Название DESC"),
    ("Название", 30),
    ("Производитель", 30),
    ("Вместимость", 30),
    3),
    (("Вывести список билетов с указанием информации об авиакомпании, направлении и паспорте пассажира. Сортировать по названию авиакомпании",
    "SELECT Авиакомпании.Название, Авиакомпании.Код, Рейсы.Номер, Направления.Название, Рейсы.Направление, Билеты.Пассажир FROM Рейсы INNER JOIN Авиакомпании ON Авиакомпании.Код = Рейсы.Авиакомпания INNER JOIN Билеты ON Рейсы.\"Номер\" = Билеты.\"Номер рейса\" AND Рейсы.Авиакомпания = Билеты.Авиакомпания INNER JOIN Направления ON Рейсы.Направление = Направления.Код ORDER BY Авиакомпании.Название"),
    ("Название", 30),
    ("Код", 10),
    ("Номер", 10),
    ("Название", 30),
    ("Направление", 30),
    ("Пассажир", 30),
    6),
    (("Вывести названия авиакомпаний из США",
    "SELECT Название FROM Авиакомпании WHERE Страна = 'США'"),
    ("Название", 30),
```

```
    1),

    (("Вывести названия всех авиакомпаний по алфавиту, которые
имеют рейсы",

    "SELECT DISTINCT Название FROM Авиакомпании INNER JOIN Рейсы
ON Рейсы.Авиакомпания = Авиакомпании.Код ORDER BY Название"),

    ("Название", 30),

    1),

    (("Вывести список сотрудников, которые имеют доступ к
самолёту Concorde",

    "SELECT * FROM Сотрудники INNER JOIN \"Допуски к самолётам\"
ON Сотрудники.\"Номер паспорта\" = \"Допуски к
самолётам\".Сотрудник AND \"Допуски к самолётам\".Самолёт =
'Concorde'"),

    ("Фамилия", 20),

    ("Имя", 20),

    ("Отчество", 20),

    ("Должность", 50),

    ("Номер паспорта", 20),

    ("Сотрудник", 20),

    ("Самолёт", 20),

    7),

    (("Вывести все должности с окладом более 80000. Сортировать
по убыванию оклада",

    "SELECT * FROM Должности WHERE Оклад > 80000 ORDER BY Оклад
DESC"),

    ("Название", 50),

    ("Оклад", 20),

    ("Описание", 105),

    3),

    (("Вывести всех пассажиров, у которых номер паспорта
начинается с AB",

    "SELECT * FROM Пассажиры WHERE \"Номер паспорта\" LIKE
'AB%'"),

    ("Фамилия", 30),

    ("Имя", 30),

    ("Отчество", 30),

    ("Номер паспорта", 20),

    ("Бонусы", 10),
```

```
    5),
    (("Вывести номера рейсов, которые вылетают по пятницам позже
13.00 но раньше 18.00",
    "SELECT Авиакомпания, Номер FROM Рейсы WHERE Время >
'13:00:00' AND Время < '18:00:00' AND \"Дни недели\" LIKE
'____1__'"),
    ("Авиакомпания", 15),
    ("Номер", 10),
    2),
    (("Вывести список всех направлений в Швеции и рейсы на это
направление",
    "SELECT * FROM Направления LEFT JOIN Рейсы ON
Направления.Код = Рейсы.Направление WHERE Страна = 'Швеция'"),
    ("Название", 20),
    ("Код", 7),
    ("Страна", 12),
    ("Расстояние", 15),
    ("Авиакомпания", 15),
    ("Номер", 10),
    ("Самолёт", 20),
    ("Направление", 20),
    ("Дни недели", 20),
    ("Время", 15),
    10),
    (("Вывести список самолётов Boeing по алфавиту и
сотрудников, которые имеют к ним допуск",
    "SELECT * FROM Самолёты LEFT JOIN \"Допуски к самолётам\" ON
Самолёты.Название = \"Допуски к самолётам\".Самолёт WHERE
Производитель = 'Boeing' ORDER BY Название"),
    ("Название", 30),
    ("Производитель", 30),
    ("Вместимость", 15),
    ("Сотрудник", 30),
    ("Самолёт", 30),
    5),
    (("Вывести в верхнем регистре ФИО всех помощников пилота",
```

```
    "SELECT UPPER(Фамилия), UPPER(Имя), UPPER(Отчество) FROM
Сотрудники WHERE Должность = 'Помощник пилота'"),

    ("Фамилия", 30),

    ("Имя", 30),

    ("Отчество", 30),

    3),

    (("Вывести список пассажиров, у которых от 10000 до 15000
бонусов и фамилией Сидоров/Сидорова",

    "SELECT * FROM Пассажиры WHERE Бонусы >= 10000 AND Бонусы <=
15000 AND Фамилия LIKE 'Сидоров_'"),

    ("Фамилия", 30),

    ("Имя", 30),

    ("Отчество", 30),

    ("Номер паспорта", 30),

    ("Бонусы", 30),

    5),

    (("Вывести список пассажиров, которые купили билет на рейс
авиакомпании Belavia (B2)",

    "SELECT * FROM Пассажиры INNER JOIN Билеты ON
Пассажиры.\"Номер паспорта\" = Билеты.Пассажир WHERE
Билеты.Авиакомпания = 'B2'"),

    ("Фамилия", 20),

    ("Имя", 20),

    ("Отчество", 20),

    ("Номер паспорта", 20),

    ("Бонусы", 10),

    ("Номер билета", 15),

    ("Авиакомпания", 20),

    ("Номер рейса", 15),

    ("Пассажир", 20),

    9),

    (("Вывести всех Офицеров авиационной безопасности, имеющих
доступ к какому-либо направлению и само направление",

    "SELECT * FROM \"Допуски к направлениям\" RIGHT JOIN
Сотрудники ON \"Допуски к направлениям\".Сотрудник =
Сотрудники.\"Номер паспорта\" WHERE Сотрудники.Должность =
'Офицер авиационной безопасности'"),

    ("Сотрудник", 15),
```

```
    ("Направление", 15),
    ("Фамилия", 20),
    ("Имя", 20),
    ("Отчество", 20),
    ("Должность", 50),
    ("Номер паспорта", 20),
    7),
    (("Вывести в верхнем регистре названия должностей, на
которых есть сотрудники с допуском к самолёту Airbus A320",
    "SELECT UPPER(Название) FROM Должности INNER JOIN Сотрудники
ON Должности.Название = Сотрудники.Должность INNER JOIN
\"Допуски к самолётам\" ON \"Допуски к самолётам\".Сотрудник =
Сотрудники.\"Номер паспорта\" AND \"Допуски к
самолётам\".Самолёт = 'Airbus A320' "),
    ("Должность", 50),
    1),
    (("Вывести страны, у которых более 1 направления с билетами
и количество этих направлений",
    "SELECT Страна, COUNT(*) AS Количество FROM ( SELECT * FROM
Направления WHERE EXISTS ( SELECT * FROM Билеты INNER JOIN Рейсы
ON Билеты.\"Номер рейса\" = Рейсы.\"Номер\" AND
Билеты.Авиакомпания = Рейсы.Авиакомпания WHERE Код =
Направление) ) GROUP BY Страна HAVING COUNT(*) > 1 ORDER BY
COUNT(*) DESC"),
    ("Страна", 30),
    ("Количество", 30),
    2),
    (("Вывести самолёты вместимостью более 300 от
производителей, у которых есть самолёты вместимостью менее 150,
к которым имеет доступ хотя бы 1 сотрудник и у которых есть
рейсы",
    "SELECT * FROM Самолёты WHERE Производитель IN (SELECT
DISTINCT Производитель FROM Самолёты WHERE Вместимость < 150 )
AND Вместимость > 300 AND Название IN ( SELECT DISTINCT Самолёт
FROM \"Допуски к самолётам\" ) AND Название IN ( SELECT DISTINCT
Самолёт FROM Рейсы )"),
    ("Название", 30),
    ("Производитель", 30),
    ("Вместимость", 30),
    3),
```

```
    (("Вывести фамилии, которые встречаются и у сотрудников, и у
пассажиров, имеющих билеты",

    "SELECT DISTINCT Фамилия FROM Сотрудники WHERE Фамилия = ANY
( SELECT Фамилия FROM Пассажиры WHERE \"Номер паспорта\" IN(
SELECT Пассажир FROM Билеты)) "),

    ("Фамилия", 30),

    1),

    (("Вывести имена, которые встречаются у сотрудников с
должностью, содержащей слово техник, но не у пассажиров",

    "WITH Имена_1 AS (SELECT Имя FROM Пассажиры) SELECT DISTINCT
Имя FROM Сотрудники WHERE Должность IN (SELECT Название FROM
Должности WHERE Название LIKE '%техник%') EXCEPT SELECT * FROM
Имена_1 ORDER BY Имя"),

    ("Имя", 30),

    1),

    (("Вывести самолёты вместимостью более 400, их среднюю
вместимость и количество сотрудников, имеющих допуск к этим
самолётам и к любому направлению",

    "WITH Самолёты_1 AS (SELECT * FROM Самолёты WHERE
Вместимость > 400) SELECT Название, Вместимость FROM Самолёты_1
UNION SELECT 'Средняя вместимость:', AVG(Вместимость) FROM
Самолёты_1 UNION SELECT 'Сотрудники:', COUNT(*) FROM \"Допуски к
самолётам\" WHERE Самолёт IN ( SELECT Название FROM Самолёты_1)
AND Сотрудник IN ( SELECT Сотрудник FROM \"Допуски к
направлениям\")"),

    ("Название", 30),

    ("Вместимость", 30),

    2),

    (("Вывести фамилии пассажиров, у которых (фамилий) есть
более 1 билета и у пассажиров с этой фамилией бонусов больше,
чем среднее расстояние до направлений",

    "SELECT Фамилия, COUNT(*) as Количество FROM Пассажиры WHERE
EXISTS ( SELECT * FROM Билеты WHERE Пассажир = \"Номер
паспорта\") AND Бонусы > ( SELECT AVG(Расстояние) FROM
Направления) GROUP BY Фамилия HAVING COUNT(*) > 1"),

    ("Фамилия", 30),

    ("Количество", 30),

    2),

    (("Вывести всех сотрудников, у которых есть допуск и к
направлению, и к самолёту",
```

```
    "SELECT * FROM Сотрудники WHERE \"Номер паспорта\" IN
(SELECT Сотрудник FROM \"Допуски к направлениям\" INTERSECT
SELECT Сотрудник FROM \"Допуски к самолётам\")"),

    ("Фамилия", 20),

    ("Имя", 20),

    ("Отчество", 20),

    ("Должность", 50),

    ("Номер паспорта", 20),

    5),

    (("Вывести минимальный и максимальный номера билетов,
которые вылетают по пятницам от 08.00 до 18.00 по направлению,
которое находится дальше любого направления в России",

    "SELECT MIN(\"Номер рейса\"), MAX(\"Номер рейса\") FROM
Билеты WHERE Авиакомпания IN ( SELECT Авиакомпания FROM Рейсы
WHERE Время > '08:00:00' AND Время < '18:00:00' AND \"Дни
недели\" LIKE '____1__' AND Направление IN ( SELECT Код FROM
Направления WHERE Расстояние > ALL ( SELECT Расстояние FROM
Направления WHERE Страна = 'Россия') ))"),

    ("Min", 30),

    ("Max", 30),

    2),

    (("Вывести список всех авиакомпаний, в чьих странах есть
направления и есть рейс с номером меньше среднего номера",

    "SELECT * FROM Авиакомпании WHERE Страна IN ( SELECT Страна
FROM Направления) AND EXISTS (SELECT * FROM Рейсы WHERE
Авиакомпания = Код AND Номер < ( SELECT AVG(Номер) FROM
Рейсы))"),

    ("Название", 30),

    ("Код", 10),

    ("Страна", 30),

    3),

    (("Вывести сотрудников, имеющих допуски и к самолётам, и к
направлениям, и у которых количество допусков к направлениям
больше среднего количества допусков к самолётам",

    "WITH Avg_1 AS (SELECT AVG(Количество) FROM (SELECT
Сотрудник, COUNT(*) AS Количество FROM \"Допуски к самолётам\"
GROUP BY Сотрудник )), Tab_1 AS (SELECT Сотрудник, COUNT(*) AS
Количество FROM \"Допуски к направлениям\" GROUP BY Сотрудник )
SELECT * FROM Сотрудники WHERE \"Номер паспорта\" IN ( SELECT
Сотрудник FROM Tab_1) AND (SELECT * FROM Avg_1) < (SELECT
Количество FROM Tab_1 WHERE Tab_1.Сотрудник = \"Номер
паспорта\")"),
```

```
    ("Фамилия", 20),
    ("Имя", 20),
    ("Отчество", 20),
    ("Должность", 50),
    ("Номер паспорта", 20),
    5),
    (("Вывести названия должностей, которые начинаются с ‘A’ и
не имеют сотрудников и названия направлений, которые начинаются
с ‘A’",
    "SELECT Название FROM Должности WHERE Название LIKE 'A%' AND
Название NOT IN ( SELECT DISTINCT Должность FROM Сотрудники)
UNION SELECT Название FROM Направления WHERE Название LIKE
'A%'"),
    ("Название", 50),
    1),
    (("Вывести сумму расстояний до направлений, сумму окладов и
сумму количеств бонусов",
    "SELECT 'Расстояния', SUM(Расстояние) FROM Направления UNION
SELECT 'Оклады', SUM(Оклад) FROM Должности UNION SELECT
'Бонусы', SUM(Бонусы) FROM Пассажиры"),
    ("Параметр", 30),
    ("Сумма", 30),
    2),
    (("Вывести названия должностей, на которых нет сотрудников,
не имеющих допуска к самолёту",
    "SELECT Название FROM Должности EXCEPT SELECT Должность FROM
Сотрудники WHERE \"Номер паспорта\" NOT IN ( SELECT Сотрудник
FROM \"Допуски к самолётам\")"),
    ("Название", 50),
    1),
    (("Вывести рейсы, у которых совпадают страна авиакомпании и
страна направления",
    "SELECT * FROM Рейсы WHERE ( SELECT Страна FROM Авиакомпании
WHERE Авиакомпании.Код = Рейсы.Авиакомпания) = ( SELECT Страна
FROM Направления WHERE Направления.Код = Рейсы.Направление)"),
    ("Авиакомпания", 15),
    ("Номер", 10),
    ("Самолёт", 20),
    ("Направление", 20),
```

```
    ("Дни недели", 20),
    ("Время", 15),
    6),
    (("Вывести время вылета рейсов, которые выполняет
авиакомпания, у которой есть билеты пассажиров с номером
паспорта на ‘AB’",
    "SELECT Время FROM Рейсы WHERE Авиакомпания IN ( SELECT
Авиакомпания FROM Рейсы INTERSECT SELECT Авиакомпания FROM
Билеты WHERE EXISTS ( SELECT * FROM Пассажиры WHERE
Билеты.Пассажир = Пассажиры.\"Номер паспорта\" AND
Пассажиры.\"Номер паспорта\" LIKE 'AB%'))"),
    ("Время", 30),
    1),
)
```

Листинг файла validators.py (содержит функции для формирования строки таблицы и для проверки введённых данных):

```
import re

def format_string_from_tuple(data_tuple, filler=' '):
  formatted_string = "|  "
  for i in range(0, len(data_tuple), 2):
    if i + 1 < len(data_tuple):
      element = str(data_tuple[i])
      if len(element) < (data_tuple[i + 1] - 2):
        formatted_element = element.ljust(data_tuple[i + 1] - 3,
filler)
      else:
        formatted_element = element[:(data_tuple[i + 1] - 3)]
    else:
      formatted_element = str(data_tuple[i])
    formatted_string += formatted_element + filler + '|' + '  '
  return formatted_string[:-1]

def s2_val(str):
```

```
    return len(str) == 2 and all(char in
set("0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ") for char in str)


def s3_val(str):
    return len(str) == 3 and all(char in
set("ABCDEFGHIJKLMNOPQRSTUVWXYZ") for char in str)


def nt_val(str):
    pattern = r"\d{4}[a-z]{1}\d{2}"
    return bool(re.match(pattern, str))


def n4_val(str):
    try:
        a = int(str)
        if a > 0 and a <= 9999:
            return True
        else:
            return False
    except ValueError:
        return False


def np_val(str):
    pattern = r"[A-Z]{2}\d{7}"
    return bool(re.match(pattern, str))


def fl_val(str):
    try:
        float(str)
        return True
    except ValueError:
        return False


def nu_val(str):
    try:
        int(str)
```

```
        return True
    except ValueError:
        return False


def wd_val(str):
    return len(str) == 7 and all(char in "01" for char in str)


def tm_val(str):
    try:
        hours, minutes, seconds = map(int, str.split(":"))
        if 0 <= hours <= 23 and 0 <= minutes <= 59 and 0 <=
seconds <= 59:
            return True
        else:
            return False
    except (ValueError, IndexError):
        return False
```

5 Вывод

В ходе выполнения лабораторной работы было разработано приложение для работы с базой данных.