

Колледж космического машиностроения и технологий

КУРСОВОЙ ПРОЕКТ

По МДК.01.02 «Прикладное программирование»

Тема: «Разработка приложения с базой данных «Канцелярия» на Python»

Выполнил студент

Магомедов Керим Мурадович

Группа П2-19

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_ (Дата сдачи работы)

Проверил преподаватель

Гусятинер Леонид Борисович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Оценка)

Королёв 2022 г.

Оглавление

[**Введение** 3](#_Toc107355675)

[**1.** **Теоретическая часть** 4](#_Toc107355676)

[**1.1.** **Описание предметной области** 4](#_Toc107355677)

[**1.2.** **Описание существующей разработки** 5](#_Toc107355678)

[**2.** **Проектная часть** 9](#_Toc107355679)

[**2.1.** **Диаграмма прецедентов** 9](#_Toc107355680)

[**2.2.** **Выбор инструментов, среды разработки и языков.** 10](#_Toc107355681)

[**2.3.** **Проектирование сценария** 12](#_Toc107355682)

[**2.4.** **Построение диаграммы классов** 13](#_Toc107355683)

[**2.5.** **Описание главного модуля** 15](#_Toc107355684)

[**2.6.** **Описание спецификаций к модулям** 22](#_Toc107355685)

[**2.7.** **Описание модулей** 24](#_Toc107355686)

[**2.8.** **Описание тестовых наборов модулей** 25](#_Toc107355687)

[**2.9.** **Описание применения средств отладки** 29](#_Toc107355688)

[**3.** **Эксплуатационная часть** 30](#_Toc107355689)

[**3.1.** **Руководство оператора** 30](#_Toc107355690)

[**3.2.** **To-do лист** 38](#_Toc107355691)

[**Заключение** 39](#_Toc107355692)

[**Список литературы и интернет источников** 40](#_Toc107355693)

[**Приложения** 41](#_Toc107355694)

# **Введение**

Тема курсовой работы – «Разработка приложения с базой данных «Канцелярия».

Актуальность данного курсового проекта заключается в том, что она позволяет отсортировать необходимую информацию и автоматизировать процесс ведения документации.

Целью данной курсовой работы является разработка приложения, которое будет взаимодействовать с базой данных.

В первой части будет рассмотрена предметная область и существующие продукты по данной теме

Во второй части будут описаны инструменты и модули, которые были разработаны.

В третьей части будут описано руководство для пользователя.

В заключительной части будет приведён To-do лист и возможные дальнейшие доработки программы.

# **Теоретическая часть**

# **Описание предметной области**

Канцелярия - распространенная и прибыльная сфера бизнеса. Достаточно иметь свидетельство о регистрации в качестве индивидуального предпринимателя или юридического лица, найти помещение, в котором можно разместить оборудование и товары, закупить это самое оборудование, товары нанять продавца и можно приступать к работе.

Не оптимальное использование информации на предприятии существенно замедляет ее обработку и эффективность применения для управления предприятием. В настоящее время очень важную роль играет структурированная информация по различным предметным областям.

# **Описание существующей разработки**

В этом разделе рассмотрены некоторые уже имеющиеся программы по данной тематике.

Практически все из них предоставляют похожий функционал:

* Создание, регистрация и хранение электронных документов, создание карточки документа.
* Движение документа по заданному маршруту, возможность его отслеживания.
* Ведение истории изменений документа.

Примеры программ:

* МойОфис;

МойОфис – онлайн сервис, предназначенный для ведения документооборота, командного общения, совместного редактирования для организаций всех форм собственности (Рисунок 1). Есть версия для облачного использования, ПК и мобильного устройства. Сервис используют 300 крупных российских компаний, ВУЗы и другие организации.

Преимущества:

* Удобство
* Функциональность
* Есть стандартная домашняя бесплатная версия

Недостатки:

* Программа сложна в освоении
* Программа платная

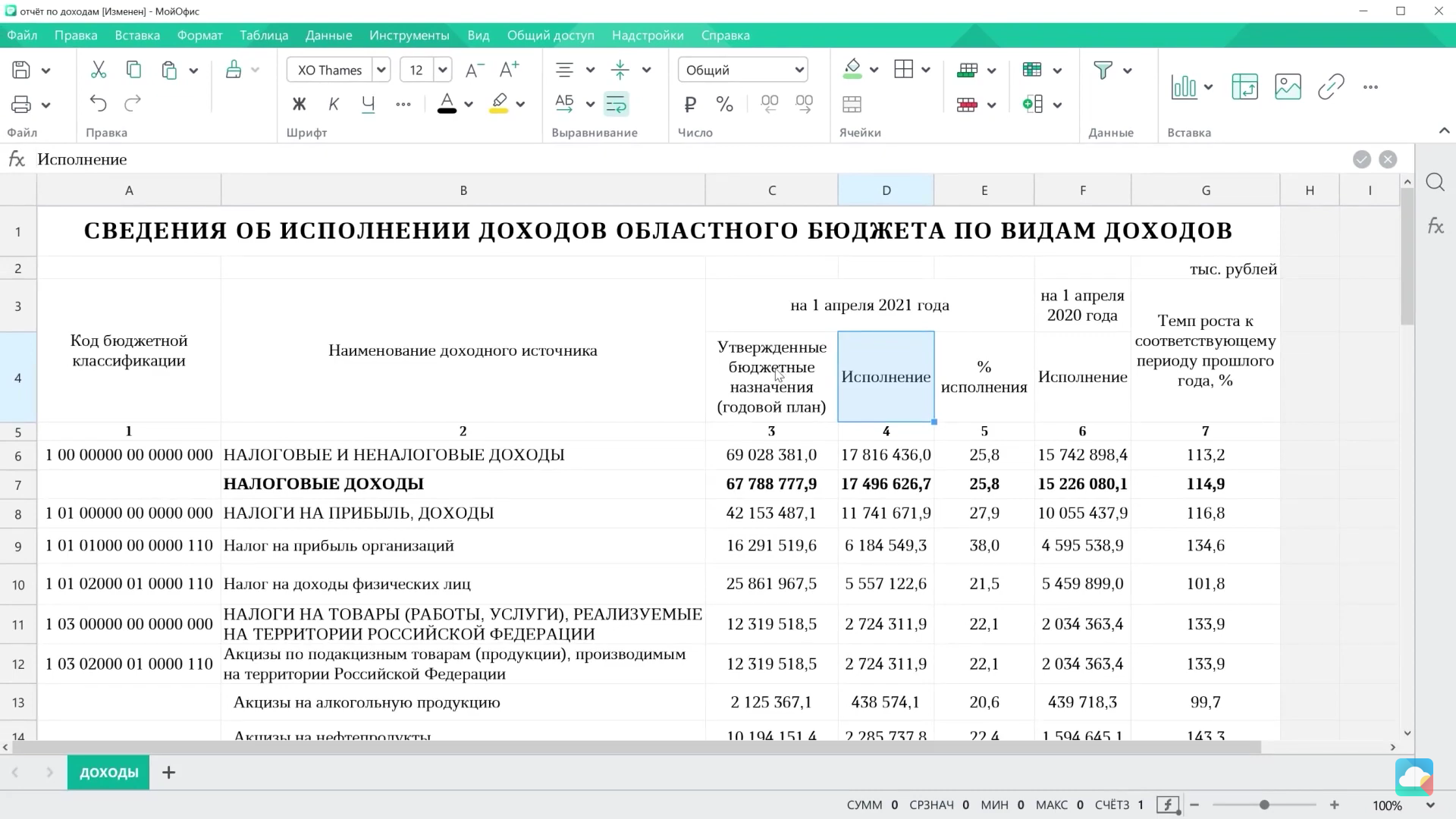


Рисунок 1 - Интерфейс программы «МойОфис»

* DocSpace;

DocSpace - это облачная платформа для ведения полноценного электронного документооборота (Рисунок 2).

Преимущества:

* Простота в использовании
* Функциональность

Недостатки:

* Программа платная
* Неудобный интерфейс

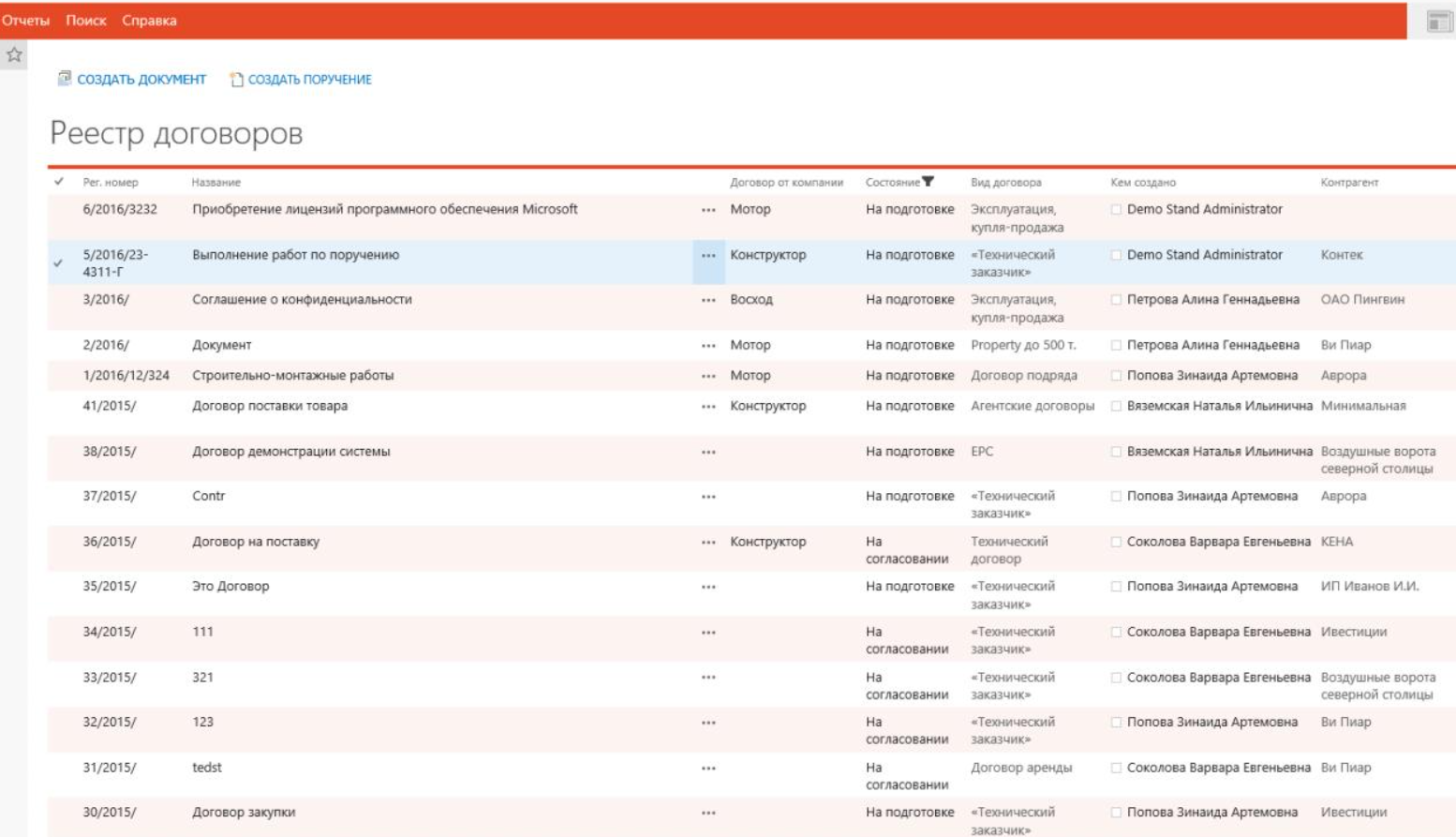


Рисунок 2. Интерфейс программы «DocSpace»

* Directrum RX;

Directrum Rx - Сервис автоматизации документооборота с готовыми бизнес – решениями по процессам: закуп, бухгалтерия, кадры и другое (Рисунок 3).

Преимущество:

* Удобный интерфейс
* Простота в использовании
* Мобильность (быстрая и лёгкая работа со смартфона или планшета через специальное приложение)

Недостатки:

* Программа платная
* Программа сложна в освоении

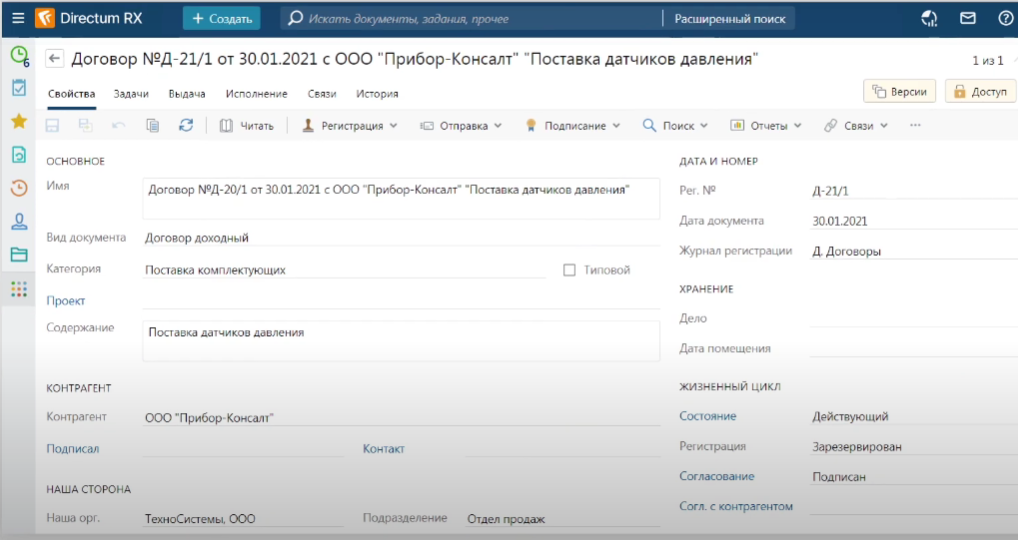


Рисунок 3. Интерфейс программы «Directrum RX»

# **Проектная часть**

# **Диаграмма прецедентов**

В этом разделе представлена диаграмма прецедентов. На которой показаны все возможные функциональные отношения (Рисунок 4).

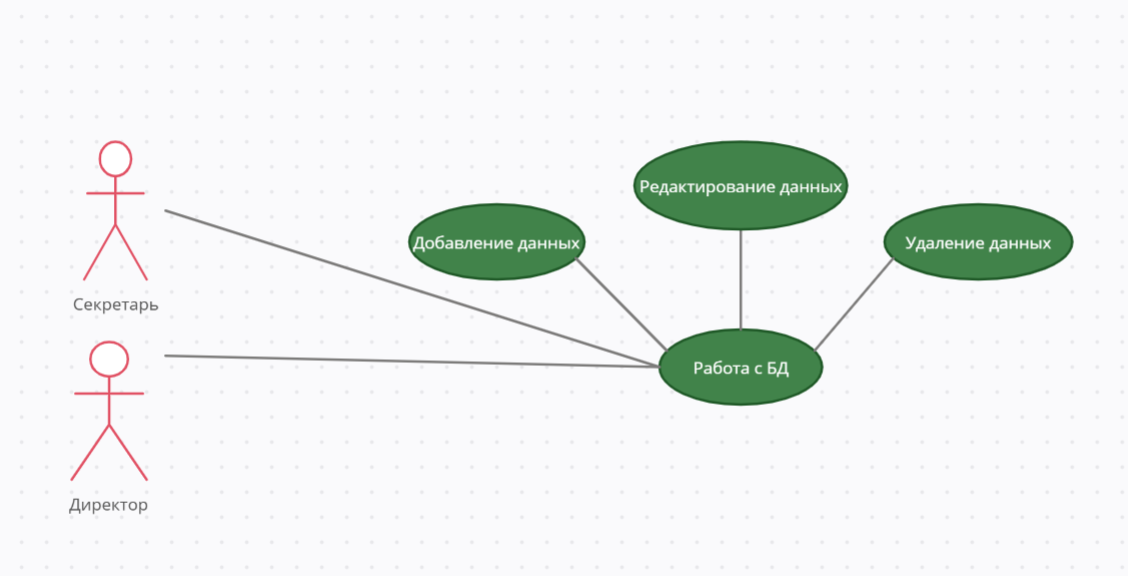


Рисунок 4. Диаграмма прецедентов программы

# **Выбор инструментов, среды разработки и языков.**

Для создания проекта, основным инструментом был выбран PyCharm, потому что он является одним из лучших инструментов для разработки на Python. В качестве СУБД был выбран SQLite [1].

Язык Python был выбран по следующим факторам:

* Средний уровень владения данным языком программирования у разработчика проекта
* Развитая поддержка
* Существует документация на русском языке

Сравнение Python с другими языками программирования представлено в таблице 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | **C** | **Python** | **Java** |
| Читабельность | 6 | 5 | 7 |
| Простота | 4 | 10 | 5 |
| GUI | 3 | 8 | 7 |
| Скорость разработки | 5 | 8 | 6 |
| Итого баллов | 18 | 31 | 25 |

Таблица 1. сравнения языков программирования

Причины, по которым был выбран дынный инструмент и СУБД:

**PyCharm**

PyCharm [2] - интегрированная среда разработки для языка программирования Python. Предоставляет средства для анализа кода, графический отладчик, инструмент для запуска юнит-тестов и поддерживает веб-разработку на Django.

PyCharm разработана компанией JetBrains на основе IntelliJ IDEA. PyCharm — это кроссплатформенная среда разработки, которая совместима с Windows, macOS, Linux.

PyCharm делает разработку максимально продуктивной благодаря функциям автодополнения и анализа кода, мгновенной подсветке ошибок и быстрым исправлениям. Автоматические рефакторинги помогают эффективно редактировать код, а удобная навигация позволяет мгновенно перемещаться по проекту.

Возможности форматирования и настройки стиля кода помогают писать понятный код, который легко поддерживать. В PyCharm есть встроенный форматер кода, отвечающий рекомендациям стандарта PEP-8 для Python, а также других стандартов для поддерживаемых языков.

**SQLite**

SQLite – это система управления базами данных, отличительной особенностью которой является ее встраиваемость в приложения. Это значит, что большинство СУБД являются самостоятельными приложениями, взаимодействие с которыми организовано по принципу клиент-сервер. Программа-клиент посылает запрос на языке SQL, СУБД, которая в том числе может находиться на удаленном компьютере, возвращает результат запроса.

Особенности SQLite:

* SQLite не требует отдельного процесса сервера или системы для работы (без сервера).
* SQLite поддерживает большинство функций языка запросов, найденных в стандарте SQL92 [3].

В данной программе будет использована консольная утилита sqlite3 в которую встроена библиотека SQLite

# **Проектирование сценария**

В данном разделе приведен сценарий использования программы пользователем приложения.

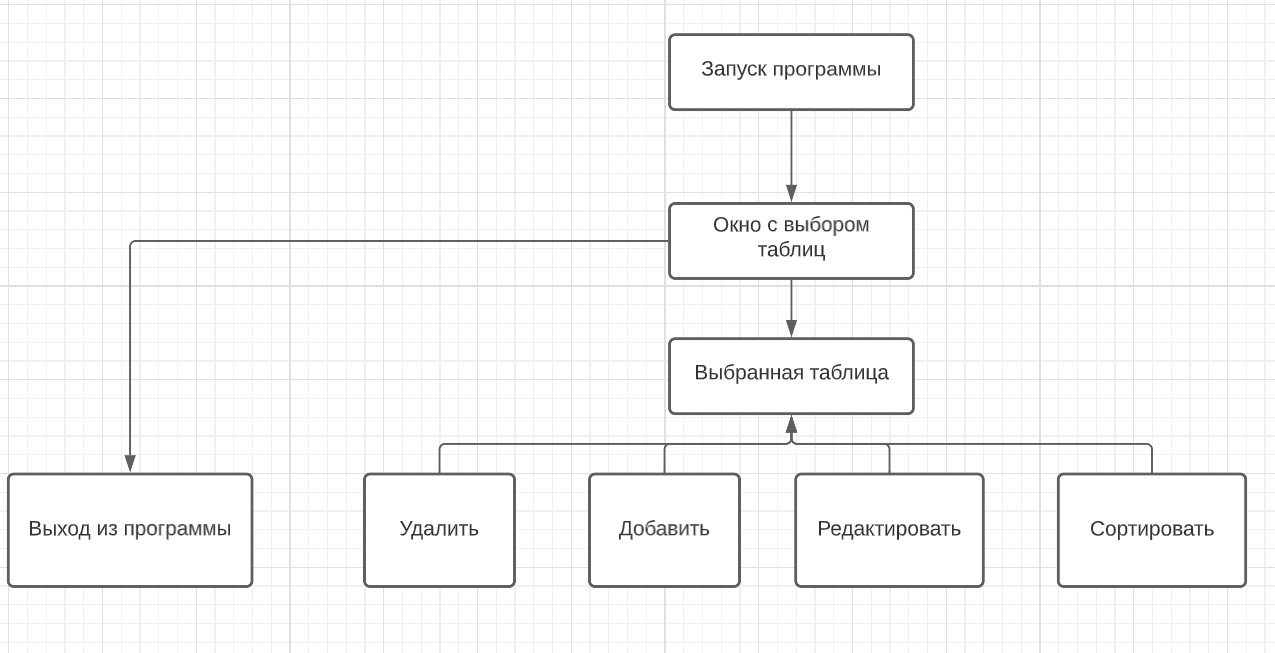


Рисунок 5. Сценарий использования

Пользователь после запуска программы может выбрать определённую таблицу или выйти из программы.

При выходе из программы, она завершает свою работу. Если пользователь выберет одну из возможных таблиц ему откроется сама таблица, и у него появится возможность добавить, удалить, редактировать, отсортировать значения.

# **Построение диаграммы классов**

В данном разделе представлены все классы, использующиеся в проекте, а также их отношения между собой.

Класс «DB1» содержит следующие методы:

* InsertDataDocuments\_types – Добавление данных для «Document\_types»
* insertDataDepartments – Добавление данных для «Departments»
* insertDataDocuments – Добавление данных для «Documents»
* records1 – Ввод новых данных «Document\_types»
* updateRecord1 – Редактирование данных «Document\_types»
* deleteRecords1 – Удаление результата «Document\_types»
* records2 – Ввод новых данных « Departments»
* updateRecord2 – Редактирование данных «Departments»
* deleteRecords2 – Удаление результата « Departments »
* records3 – Ввод новых данных «Documents»
* updateRecord3 – Редактирование данных «Documents»
* deleteRecords3 – Удаление результата «Documents»

Класс «Document\_types» содержит следующие методы:

* viewRecords – Вывод данных
* tableDocument\_types – Создание таблицу с помощью ttk.Treeview()
* editDocument\_types – Редактирование таблицы
* idgive – Получение строки по id из БД
* idDelete – Удаление строки по id из БД
* lineDelete – Удаление выбранной строки из таблицы tkinter

Класс «Departments» содержит следующие методы:

* viewRecords – Вывод данных
* tableDepartments – Создание таблицу с помощью ttk.Treeview()
* editDepartments – Редактирование таблицы
* idDelete – Удаление строки по id из БД
* lineDelete – Удаление выбранной строки из таблицы tkinter

Класс «Documents» содержит следующие методы:

* viewRecords – Вывод данных
* tableDocuments – Создание таблицу с помощью ttk.Treeview()
* editDocuments – Редактирование таблицы
* idDelete – Удаление строки по id из БД
* lineDelete – Удаление выбранной строки из таблицы tkinter

Класс «tkDocument\_types» содержит следующие методы:

* Document\_typesButton – Кнопка для открытия таблицы и её функций
* unionFunc – Компонует все кнопки функции выше
* addWindow – Создание нового окна для кнопки <Добавить>
* ChoosingTheRightOption – Выбор столбца для сортировки

Класс «tkDepartments» содержит следующие методы:

* DepartmentsButton – Кнопка для открытия таблицы и её функций
* unionFunc – Компонует все кнопки функции выше
* addWindow – Создание нового окна для кнопки <Добавить>

Класс «tkDocuments» содержит следующие методы:

* DocumentsButton – Кнопка для открытия таблицы и её функций
* unionFunc – Компонует все кнопки функции выше
* addWindow – Создание нового окна для кнопки <Добавить>
* ChoosingTheRightOption – Выбор столбца для сортировки

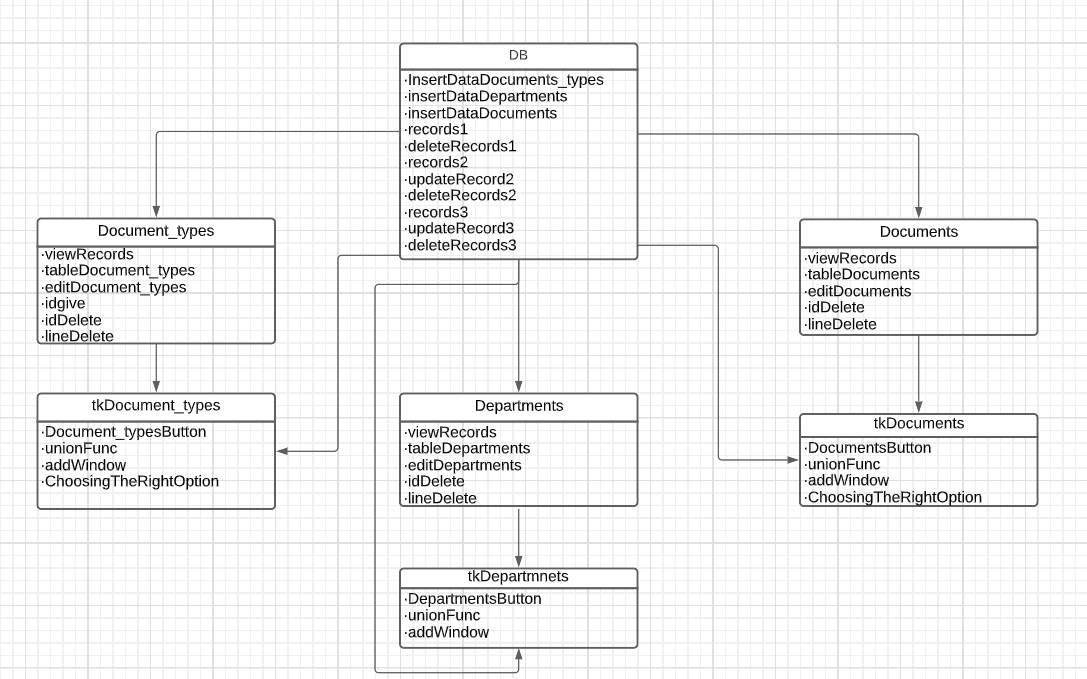


Рисунок 6. Диаграмма классов

# **Описание главного модуля**

В главный модуль входят классы «Assortment», «Shopping\_cart», «Workers», «tkAssortment», «tkShopping\_cart», «tkWorkers», которые отвечают за функционал программы.

Также в главный модуль входит код, который запускает саму программу при ее запуске. В листинге 1-3 представлена часть главного модуля. Полный код главного модуля находится в «Приложение 1».

**Листинг 1. Класс «tkDocument\_types» отвечающий за интерфейс таблицы «Document\_types»**

class tkDocument\_types():  
  
 def \_\_init\_\_(self):  
 self.sale = sale # экземпляр класса Document\_types()  
 self.db = db  
 def Document\_typesButton(self):  
 ''' Кнопка для открытия таблицы и её функций '''  
 def unionFunc():  
 ''' Компонует все кнопки функции выше '''  
 def addWindow():  
 ''' Создание нового окна для кнопки <Добавить> '''  
 def closeFunc():  
 s.destroy() # закрываем окно  
  
 # Создаем новое окно  
 s = Toplevel()  
 s.geometry('250x200')  
 s.title('Add')  
 s.resizable(False, False)  
  
 view\_code\_label = Label(s, text='Код вида').place(x=10, y=60, width=70, height=30)  
 name\_label = Label(s, text='Нaзвание').place(x=10, y=10, width=70, height=30)  
   
 name\_1 = StringVar()  
 name\_2 = StringVar()  
  
 view\_code\_entry = Entry(s, width=50, textvariable=name\_1).place(x=100, y=60, width=100, height=30)  
 name\_entry = Entry(s, width=50, textvariable=name\_2).place(x=100, y=10, width=100, height=30)  
  
   
 add\_button = Button(s, text='Добавить', command=lambda:  
 (self.db.records1(name\_1.get(), name\_2.get()), self.sale.viewRecords(), self.sale.tableDocument\_types()))  
 add\_button.place(x=10, y=160, width=100, height=30)  
  
 close\_button = Button(s, text='Закрыть', command=closeFunc)  
 close\_button.place(x=115, y=160, width=100, height=30)  
  
 self.add\_image = tk.PhotoImage(file='add.png')  
 Document\_types\_add\_button = Button(root, text='Добавить', image=self.add\_image, compound='top', command=addWindow)  
 Document\_types\_add\_button.place(x=10, y=50, width=100, height=80)  
  
 self.edit\_image = tk.PhotoImage(file='edit.png')  
 Document\_types\_edit\_button = Button(root, text='Редактировать', image=self.edit\_image, compound='top', command=lambda:  
 self.sale.editDocument\_types())  
 Document\_types\_edit\_button.place(x=120, y=50, width=100, height=80)  
   
 self.delete\_image = tk.PhotoImage(file='delete.png')   
 Document\_types\_edit\_button = Button(root, text='Удалить', image=self.delete\_image, compound='top', command=lambda:  
 (self.sale.idDelete(), self.sale.lineDelete()))  
 Document\_types\_edit\_button.place(x=230, y=50, width=100, height=80)  
  
 def ChoosingTheRightOption():  
 if column\_menu.get() == 'Нaзвание':  
 s = 'Нaзвание'  
 return s  
   
 column\_menu = ttk.Combobox(root,values=['Нaзвание'])  
 column\_menu.place(x=395, y=75, width=110, height=30)  
 column\_menu.current(0)  
  
 mb = Menubutton(root, text='Сортировка', relief=RAISED)  
 mb.place(x=537, y=75, width=100, height=30)  
   
 mb.menu = Menu(mb, tearoff=0)  
 mb['menu'] = mb.menu  
   
 mb.menu.add\_command(label='ВЫВОД В ЭКСЕЛЬ СЛУЖЕБОЙ ЗАПИСКИ ПЛАНОВОГО ОТДЕЛА', command=lambda:  
 (self.sale.generalIncreaseSort(ChoosingTheRightOption()), self.sale.tableDocument\_types()))  
 mb.menu.add\_command(label='ВЫВОД ТАБЛИЦЫ', command=lambda:  
 (self.sale.generalIncrease(ChoosingTheRightOption()), self.sale.tableDocument\_types()))  
   
 Document\_types\_button = Button(root, text='Document\_types', command=lambda:  
 (unionFunc(), self.sale.viewRecords(), self.sale.tableDocument\_types()))  
 Document\_types\_button.place(x=10, y=10, width=100, height=30)

**Листинг 2. Класс «tkDepartments» отвечающий за интерфейс таблицы «Departments»**

class tkDepartments():  
  
 def \_\_init\_\_(self):  
 self.cust = cust # экземпляр класса Departments()  
 self.db = db  
 def DepartmentsButton(self):  
 ''' Кнопка для открытия таблицы и её функций '''  
 def unionFunc():  
 ''' Компонует все кнопки функции выше '''  
 def addWindow():  
 ''' Создание нового окна для кнопки <Добавить> '''  
 def closeFunc():  
 s.destroy() # закрываем окно  
  
 # Создаем новое окно  
 s = Toplevel()  
 s.geometry('250x200')  
 s.title('Add')  
 s.resizable(False, False)  
  
 department\_code\_label = Label(s, text='Код отдела').place(x=10, y=10, width=80, height=30)  
 title\_label = Label(s, text='Название').place(x=10, y=60, width=80, height=30)  
   
 name\_1 = StringVar()  
 name\_3 = StringVar()  
  
  
 department\_code\_entry = Entry(s, width=50, textvariable=name\_1).place(x=130, y=10, width=100, height=30)  
 title\_entry = Entry(s, width=50, textvariable=name\_3).place(x=130, y=60, width=100, height=30)  
  
  
 add\_button = Button(s, text='Добавить', command=lambda:  
 (self.db.records2(name\_1.get(), name\_3.get()), self.cust.viewRecords(), self.cust.tableDepartments()))  
 add\_button.place(x=10, y=150, width=100, height=30)  
  
 close\_button = Button(s, text='Закрыть', command=closeFunc)  
 close\_button.place(x=115, y=150, width=100, height=30)  
  
 self.add\_image = tk.PhotoImage(file='add.png')  
 Document\_types\_add\_button = Button(root, text='Добавить', image=self.add\_image, compound='top', command=addWindow)  
 Document\_types\_add\_button.place(x=10, y=50, width=100, height=80)  
  
 self.edit\_image = tk.PhotoImage(file='edit.png')  
 Document\_types\_edit\_button = Button(root, text='Редактировать', image=self.edit\_image, compound='top', command=lambda:  
 self.cust.editDepartments())  
 Document\_types\_edit\_button.place(x=120, y=50, width=100, height=80)  
   
 self.delete\_image = tk.PhotoImage(file='delete.png')   
 Document\_types\_edit\_button = Button(root, text='Удалить', image=self.delete\_image, compound='top', command=lambda:  
 (self.cust.idDelete(), self.cust.lineDelete()))  
 Document\_types\_edit\_button.place(x=230, y=50, width=100, height=80)  
   
  
 Departments\_button = Button(root, text='Departments', command=lambda:  
 (unionFunc(), self.cust.viewRecords(), self.cust.tableDepartments()))  
 Departments\_button.place(x=120, y=10, width=100, height=30)

**Листинг 3. Класс «tkDocuments» отвечающий за интерфейс таблицы «Documents»**

class tkDocuments():  
  
 def \_\_init\_\_(self):  
 self.orde = orde # экземпляр класса Documents()  
 self.db = db  
 def DocumentsButton(self):  
 ''' Кнопка для открытия таблицы и её функций '''  
 def unionFunc():  
 ''' Компонует все кнопки функции выше '''  
 def addWindow():  
 ''' Создание нового окна для кнопки <Добавить> '''  
 def closeFunc():  
 s.destroy() # закрываем окно  
  
  
  
 # Создаем новое окно  
 s = Toplevel()  
 s.geometry('280x350')  
 s.title('Add')  
 s.resizable(False, False)  
  
  
 Document\_code\_label = Label(s, text='Код документа').place(x=20, y=10, width=100, height=30)  
 Appellation\_label = Label(s, text='Название').place(x=20, y=60, width=100, height=30)  
 Number\_label = Label(s, text='Номер').place(x=20, y=100, width=100, height=30)  
 View\_code\_label = Label(s, text='Код\_вида').place(x=20, y=140, width=100, height=30)  
 Documentcodes\_label = Label(s, text='Код\_отдела-отправителя').place(x=20, y=180, width=135, height=30)  
 Documentcoder\_label = Label(s, text='Код\_отдела-получателя').place(x=20, y=220, width=135, height=30)  
 Reg\_date\_label = Label(s, text='Дата регистрации').place(x=20, y=260, width=120, height=30)  
   
   
   
   
 name\_1 = StringVar()  
 name\_2 = StringVar()  
 name\_3 = StringVar()  
 name\_4 = StringVar()  
 name\_5 = StringVar()  
 name\_6 = StringVar()  
 name\_7 = StringVar()  
  
  
 Document\_code\_entry = Entry(s, width=50, textvariable=name\_1).place(x=155, y=10, width=120, height=30)  
 Appellation\_entry = Entry(s, width=50, textvariable=name\_2).place(x=155, y=60, width=120, height=30)  
 Number\_entry = Entry(s, width=50, textvariable=name\_3).place(x=155, y=100, width=120, height=30)  
 View\_code\_entry = Entry(s, width=50, textvariable=name\_4).place(x=155, y=140, width=120, height=30)  
 Documentcodes\_entry = Entry(s, width=50, textvariable=name\_5).place(x=155, y=180, width=120, height=30)  
 Documentcoder\_entry = Entry(s, width=50, textvariable=name\_6).place(x=155, y=220, width=120, height=30)  
 Reg\_date\_entry = Entry(s, width=50, textvariable=name\_7).place(x=155, y=260, width=120, height=30)  
  
   
   
 add\_button = Button(s, text='Добавить', command=lambda:  
 (self.db.records3(name\_1.get(), name\_2.get(), name\_3.get(), name\_4.get(), name\_5.get(), name\_6.get(), name\_7.get()), self.orde.viewRecords(), self.orde.tableDocuments()))  
 add\_button.place(x=10, y=300, width=100, height=30)  
  
 close\_button = Button(s, text='Закрыть', command=closeFunc)  
 close\_button.place(x=115, y=300, width=100, height=30)  
  
 self.add\_image = tk.PhotoImage(file='add.png')  
 Document\_types\_add\_button = Button(root, text='Добавить', image=self.add\_image, compound='top', command=addWindow)  
 Document\_types\_add\_button.place(x=10, y=50, width=100, height=80)  
  
 self.edit\_image = tk.PhotoImage(file='edit.png')  
 Document\_types\_edit\_button = Button(root, text='Редактировать', image=self.edit\_image, compound='top', command=lambda:  
 self.orde.editDocuments())  
 Document\_types\_edit\_button.place(x=120, y=50, width=100, height=80)  
   
 self.delete\_image = tk.PhotoImage(file='delete.png')   
 Document\_types\_edit\_button = Button(root, text='Удалить', image=self.delete\_image, compound='top', command=lambda:  
 (self.orde.idDelete(), self.orde.lineDelete()))  
 Document\_types\_edit\_button.place(x=230, y=50, width=100, height=80)  
  
 def ChoosingTheRightOption():  
 if column\_menu.get() == 'Код документа':  
 s = 'Код документа'  
 return s  
 if column\_menu.get() == 'Название':  
 s = 'Название'  
 return s  
 if column\_menu.get() == 'Номер':  
 s = 'Номер'  
 return s  
 if column\_menu.get() == 'Код\_вида':  
 s = 'Код\_вида'  
 return s  
 if column\_menu.get() == 'Код\_отдела-отправителя':  
 s = 'Код\_отдела-отправителя'  
 return s  
 if column\_menu.get() == 'Код\_отдела-получателя':  
 s = 'Код\_отдела-получателя'  
 return s  
 if column\_menu.get() == 'Дата регистрации':  
 s = 'Дата регистрации'  
 return s  
  
  
   
 column\_menu = ttk.Combobox(root, values=['Код документа','Название', 'Номер', 'Код\_вида', 'Код\_отдела-отправителя', 'Код\_отдела-получателя', 'Дата регистрации'])  
 column\_menu.place(x=395, y=75, width=110, height=30)  
 column\_menu.current(0)  
   
 mb = Menubutton (root, text='Сортировка', relief=RAISED)  
 mb.place(x=537, y=75, width=100, height=30)  
   
 mb.menu = Menu( mb,tearoff =0 )  
 mb['menu'] = mb.menu  
   
 mb.menu.add\_command(label='Сортировать по возрастанию', command=lambda:  
 (self.orde.generalIncreaseSort(ChoosingTheRightOption()), self.orde.tableDocuments()))  
   
 mb.menu.add\_command(label='Сортировать по убыванию', command=lambda:  
 (self.orde.generalDescendingSort(ChoosingTheRightOption()), self.orde.tableDocuments()))  
  
   
   
 Documents\_button = Button(root, text='Documents', command=lambda:  
 (unionFunc(), self.orde.viewRecords(), self.orde.tableDocuments()))  
 Documents\_button.place(x=230, y=10, width=100, height=30)

# **Описание спецификаций к модулям**

В данном разделе описаны публичные члены модулей курсового проекта.

В главном модуле «main.py» содержатся публичные методы классов «Document\_types», «Departments», «Documents», «tkDocument\_types», «tkDepartments», «tkDocument\_types», которые взаимодействуют с методами класса «DB1» из модуля «db.py».

В данном модуле также есть пара объектов:

1. root - объект, который создает системное окно и объект самого приложения.
2. db – объект, которые принимает экземпляр класса «DB1» для вызова функций.
3. sale– объект, которые принимает экземпляр класса «Documents\_type» для вызова функций.
4. cust– объект, которые принимает экземпляр класса «Departmentst» для вызова функций.
5. orde– объект, которые принимает экземпляр класса «Documents» для вызова функций

**Листинг 4. Методы класса «DB1»**

**#Методы DB (Виды документов)**  
def records1(self, view\_code, name):  
 ''' Ввод новых данных '''

def updateRecord1(self, view\_code, name, ID\_1):  
 ''' Редактирование данных '''

def deleteRecords1(self, ID\_1):  
 ''' Удаление результата '''

**#Методы DB (Отделы)**  
def records2(self, department\_code, title):  
 ''' Ввод новых данных '''

def updateRecord2(self, department\_code, title, ID\_2):  
 ''' Редактирование данных '''

def deleteRecords2(self, ID\_2):  
 ''' Удаление результата '''

**#Методы DB (Документы)**  
def records3(self, Document\_code, Appellation, Number, View\_code, Documentcodes, Documentcoder, Reg\_date):  
 ''' Ввод новых данных '''

def updateRecord3(self, Document\_code, Appellation, Number, View\_code, Documentcodes, Documentcoder, Reg\_date, ID\_3):  
 ''' Редактирование данных '''

def deleteRecords3(self, ID\_3):  
 ''' Удаление результата '''

# **Описание модулей**

Кроме главного модуля программа содержит еще один модуль который реализует работу с базой данных «db.py».

Полный код модуля приведен в «Приложение 2». В этом разделе приведены все методы, которые содержаться в классe «DB1» из модуля «db.py».

**Листинг 4. Методы класса «DB1»**

**#Методы DB (Виды документов)**  
def records1(self, view\_code, name):  
 ''' Ввод новых данных '''

def updateRecord1(self, view\_code, name, ID\_1):  
 ''' Редактирование данных '''

def deleteRecords1(self, ID\_1):  
 ''' Удаление результата '''

**#Методы DB (Отделы)**  
def records2(self, department\_code, title):  
 ''' Ввод новых данных '''

def updateRecord2(self, department\_code, title, ID\_2):  
 ''' Редактирование данных '''

def deleteRecords2(self, ID\_2):  
 ''' Удаление результата '''

**#Методы DB (Документы)**  
def records3(self, Document\_code, Appellation, Number, View\_code, Documentcodes, Documentcoder, Reg\_date):  
 ''' Ввод новых данных '''

def updateRecord3(self, Document\_code, Appellation, Number, View\_code, Documentcodes, Documentcoder, Reg\_date, ID\_3):  
 ''' Редактирование данных '''

defdeleteRecords3(self, ID\_3):  
 ''' Удаление результата '''

# **Описание тестовых наборов модулей**

В этом разделе будут продемонстрированы результаты тестирования программы.

**Тест 1. Переключение между таблицами**

Действия: Нажать на кнопку с названием таблицы(Document\_types).

Ожидаемый результат: Смена таблицы «Document\_types» на таблицу «Departments».

Результат теста:

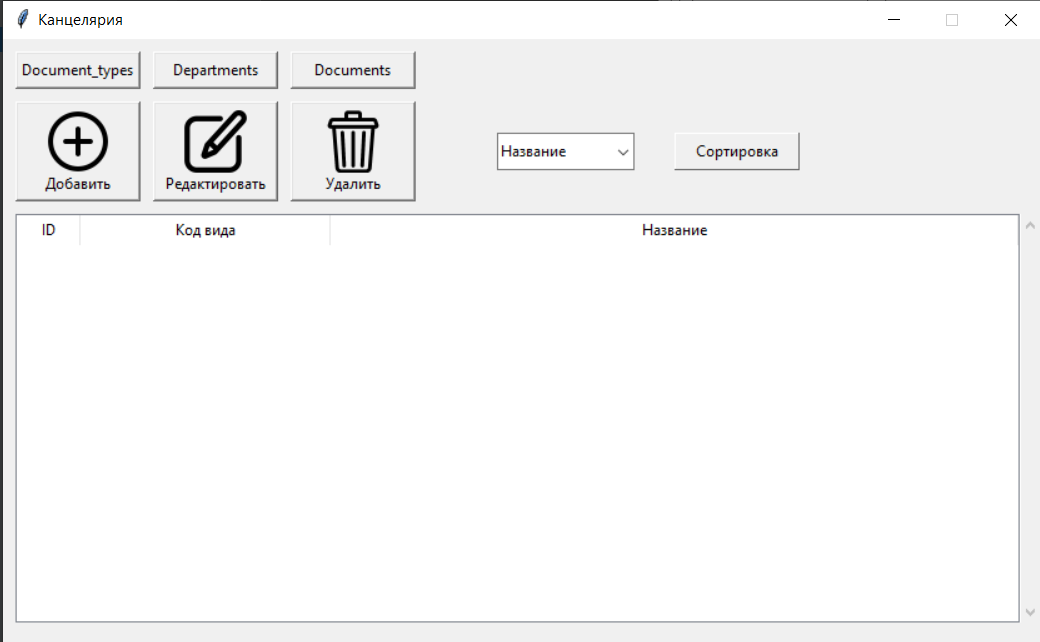


Рисунок 6. Таблица «Document\_types»

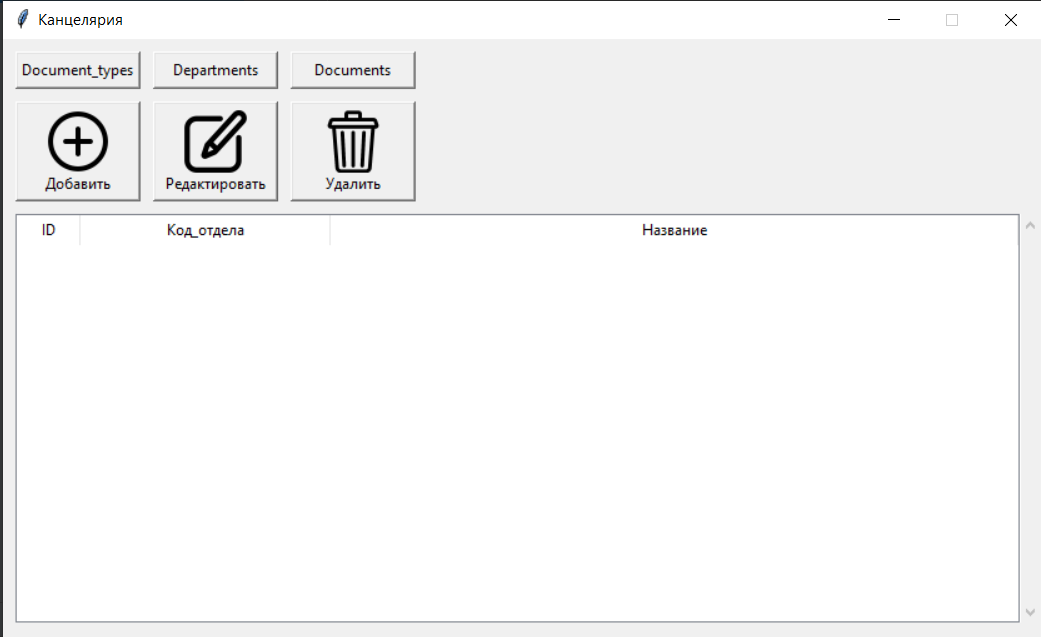


Рисунок 7. Таблица «Departments»

**Тест 2. Добавление значений**

Действия: Нажать на кнопку «Добавить», в появившейся новой строке таблицы ввести необходимую информацию и нажать на кнопку «Добавить».

Ожидаемый результат: Добавление строки в таблицу, добавление в нее информации и сохранение редактированной таблицы.

Результат теста:

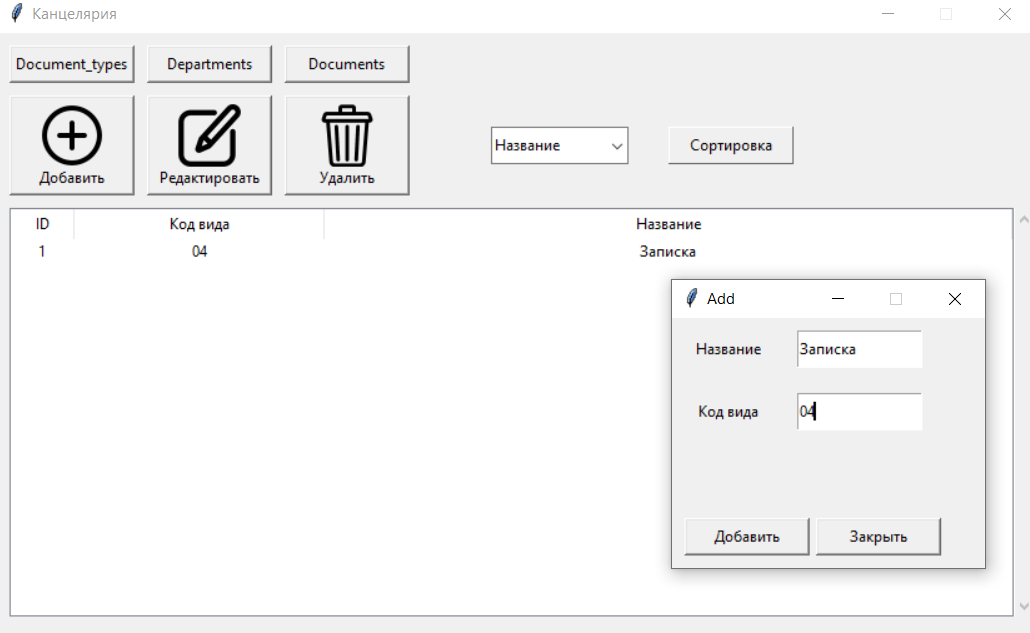


Рисунок 8. Добавление значений

**Тест 3. Редактирование значений**

Действия: Выбрать строку и нажать на кнопку «Редактировать», в появившейся строке таблицы отредактировать необходимую информацию и нажать на кнопку «Редактировать».

Ожидаемый результат: Редактирование строки в таблицу, добавление в нее информации и сохранение редактированной таблицы.

Результат теста:

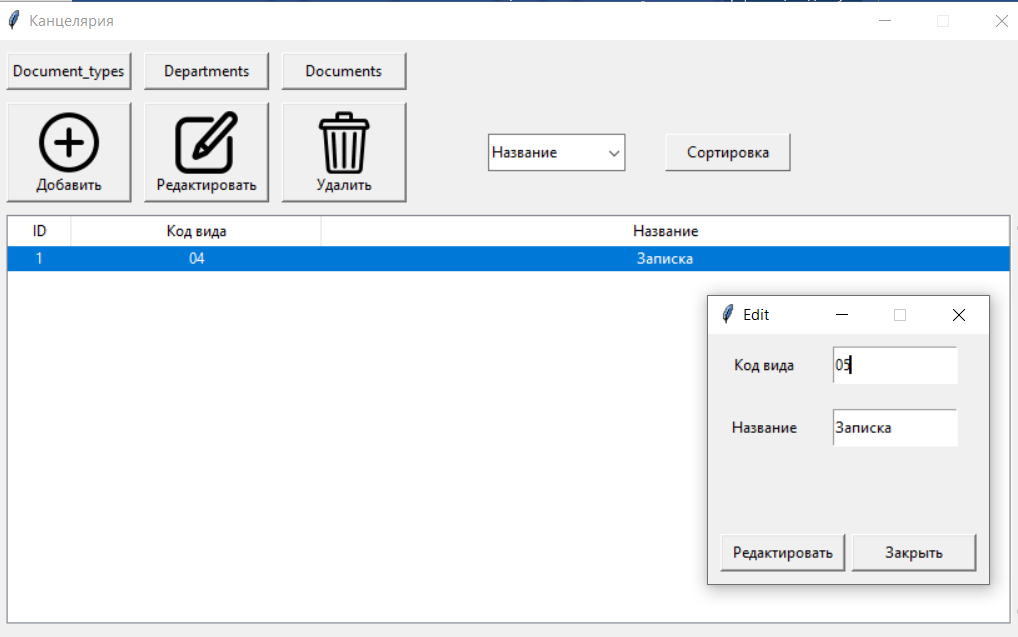


Рисунок 9. Редактирование значения

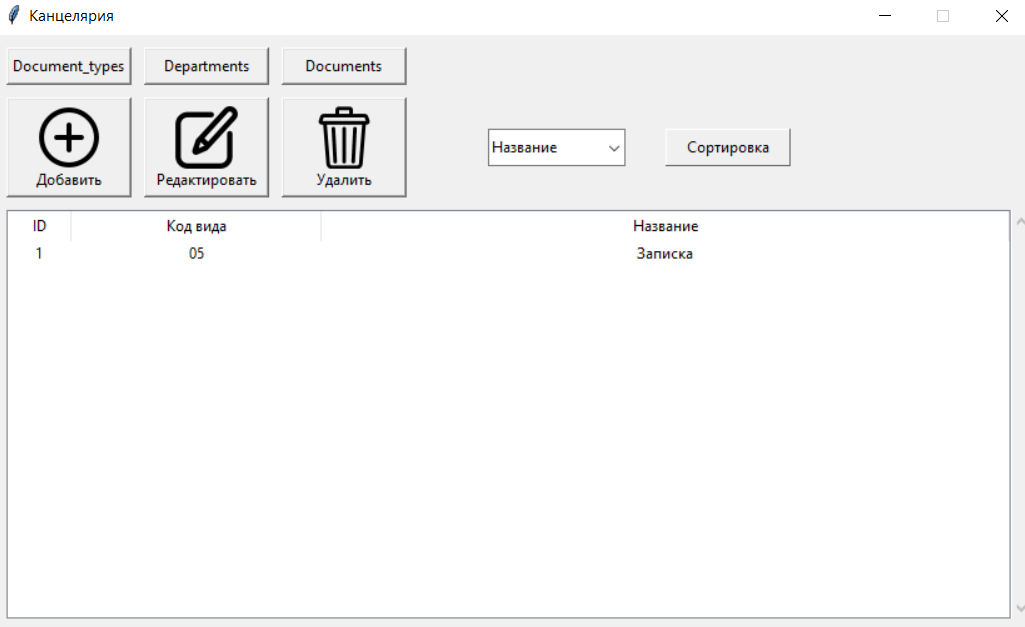


Рисунок 10. Результат редактирование значения

**Тест 4. Удаление значения**

Действия: Выбрать строку и нажать на кнопку «Удалить»

Ожидаемый результат: Удаление строки из таблицы

Результат теста:

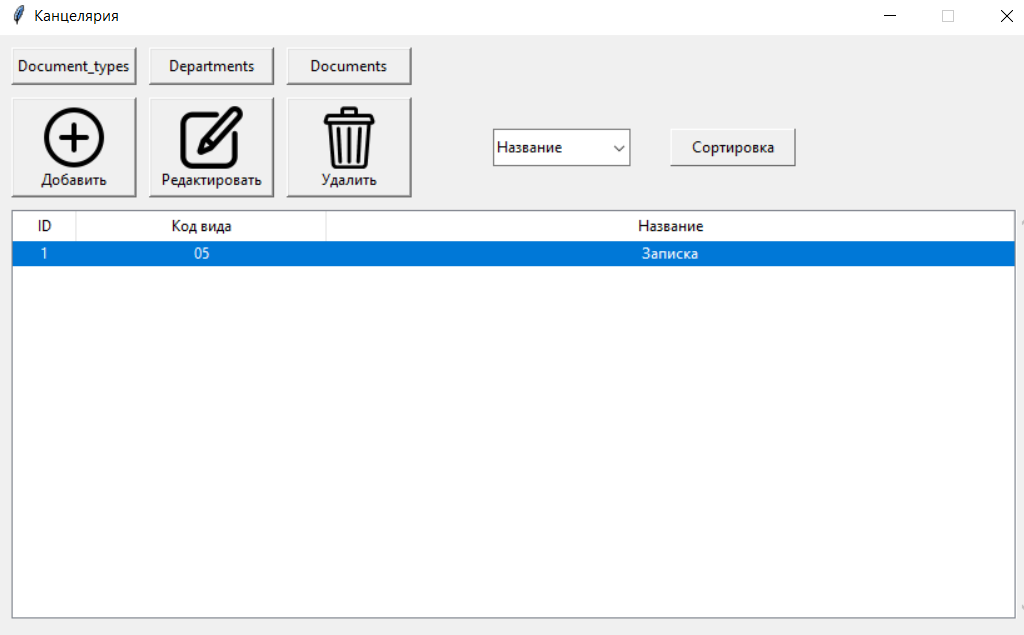


Рисунок 11. Удаление значения

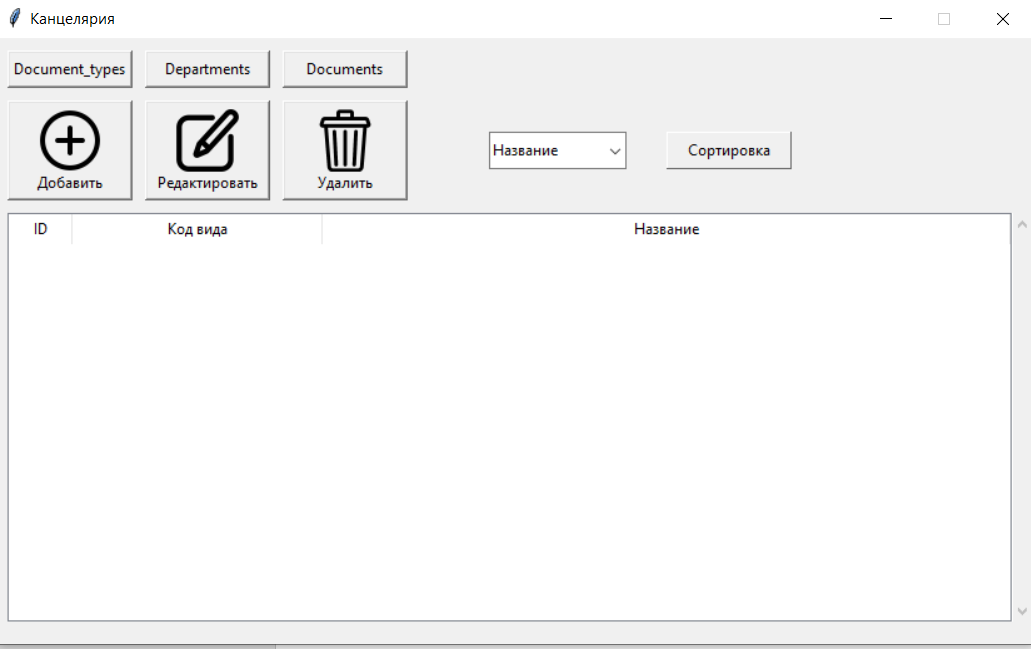


Рисунок 12. Результат Удаление значения

# **Описание применения средств отладки**

В этом разделе показано умение применять средства отладки. Так как я работаю в PyCharm, отладчик уже встроен в программу.

Для проверки была искусственно создана ошибка в коде. Вот что показал отладчик:

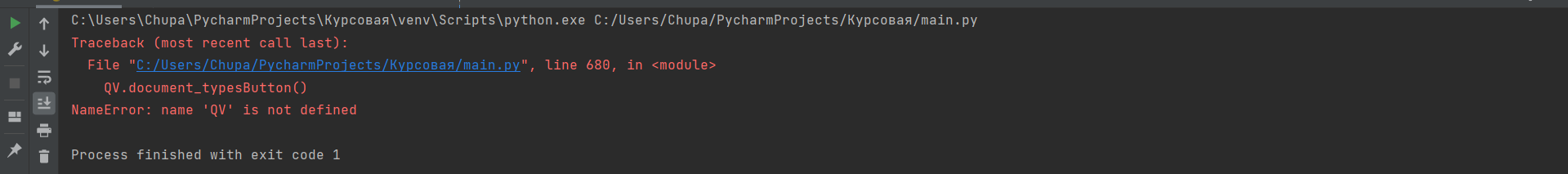


Рисунок 13. Ошибка в коде

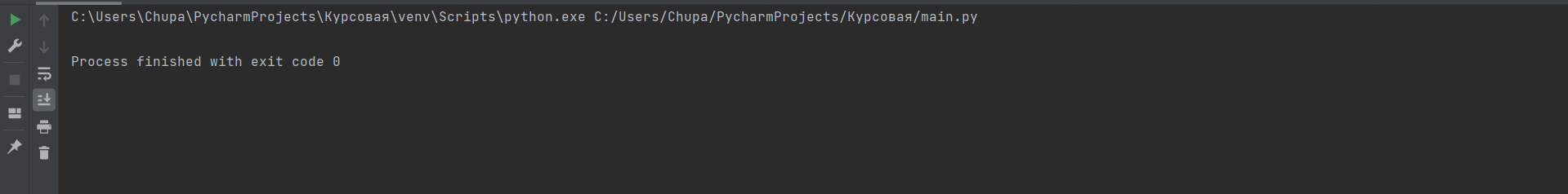


Рисунок 14. Исправление ошибки

# **Эксплуатационная часть**

# **Руководство оператора**

В данном программном документе приведено руководство оператора по применению и эксплуатации программы «Канцелярия».

В данном программном документе, в разделе «Назначение программы» указаны сведения о назначении программы и информация, достаточная для понимания функций программы и ее эксплуатации.

В разделе «Условия выполнения программы» указаны условия, необходимые для выполнения программы (минимальный состав аппаратных и программных средств и т.п.).

В данном программном документе, в разделе «Выполнение программы» указана последовательность действий оператора, обеспечивающих загрузку, запуск, выполнение и завершение программы, приведено описание функций, формата и возможных вариантов команд, с помощью которых оператор осуществляет загрузку и управляет выполнением программы, а также ответы программы на эти команды.

Оформление программного документа «Руководство оператора» произведено по требованиям ЕСПД (ГОСТ 19.101-77 [[1]](#footnote-1)1), ГОСТ 19.103-77 [[2]](#footnote-2)2), ГОСТ 19.104-78\* [[3]](#footnote-3)3), ГОСТ 19.105-78\* [[4]](#footnote-4)4), ГОСТ 19.106-78\* [[5]](#footnote-5)5), ГОСТ 19.505-79\* [[6]](#footnote-6)6), ГОСТ 19.604-78\* [[7]](#footnote-7)7)).

1. **Назначение программы**
   1. **Функциональное назначение программы**

Специальное программное обеспечение «Канцелярия» используется для добавления товаров в ассортимент приложения, добавления товаров в корзину покупок и хранения информации о персонале.

* 1. **Эксплуатационное назначение программы**

Специальное программное обеспечение «Канцелярия» может эксплуатироваться на объектах любого масштаба в сфере магазинов для облегчения работы персонала.

* 1. **Состав функций**
     1. **Функция смены таблицы**

Эта функция позволяет менять таблицу в зависимости от необходимости.

* + 1. **Функция добавления элемента в таблицу.**

Эта функция позволяет добавлять нужную информацию в таблицу.

* + 1. **Функция удаления элемента из таблицы.**

Эта функция позволяет удалить ненужные элементы в таблице.

* + 1. **Функция редактирования элемента.**

Эта функция позволяет редактировать выбранный элемент в таблице.

1. **Условия выполнения программы**
   1. **Минимальный состав аппаратных средств**

ОС: Windows 10

Процессор: Как минимум 1 ГГц или SoC.

ОЗУ: 1 ГБ (для 32-разрядных систем) или 2 ГБ (для 64-разрядных систем).

Место на жестком диске: 16 ГБ (для 32-разрядных систем) или 20 ГБ (для 64-разрядных систем).

Видеоадаптер: DirectX версии не ниже 9 с драйвером WDDM 1.0.

Дисплей: 800 x 600.

* 1. **Минимальный состав программных средств**

Дополнительные программные средства не требуются.

* 1. **Требование к пользователю**

Конечный пользователь программы должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы.

1. **Выполнение программы**
   1. **Загрузка и запуск программы**

Запустите программу «Канцелярия», после чего откроется окно:

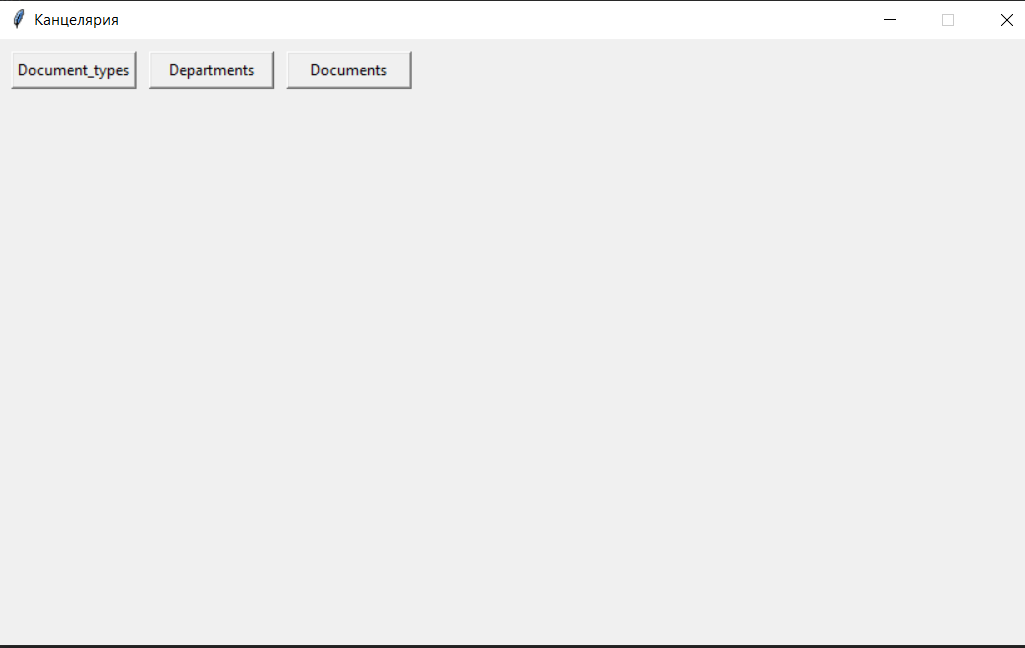


Рисунок 15. Окно приложения при запуске

Кнопки меню имеют следующие функции:

* Document\_types – Открытие таблицы Document\_types
* Departments– Открытие таблицы Departments
* Documents – Открытие таблицы Documents
  1. **Выполнение программы**
     1. **Выполнение функции открытия таблицы.**

Выберите кнопку «Document\_types», после этого откроется окно с таблицей.

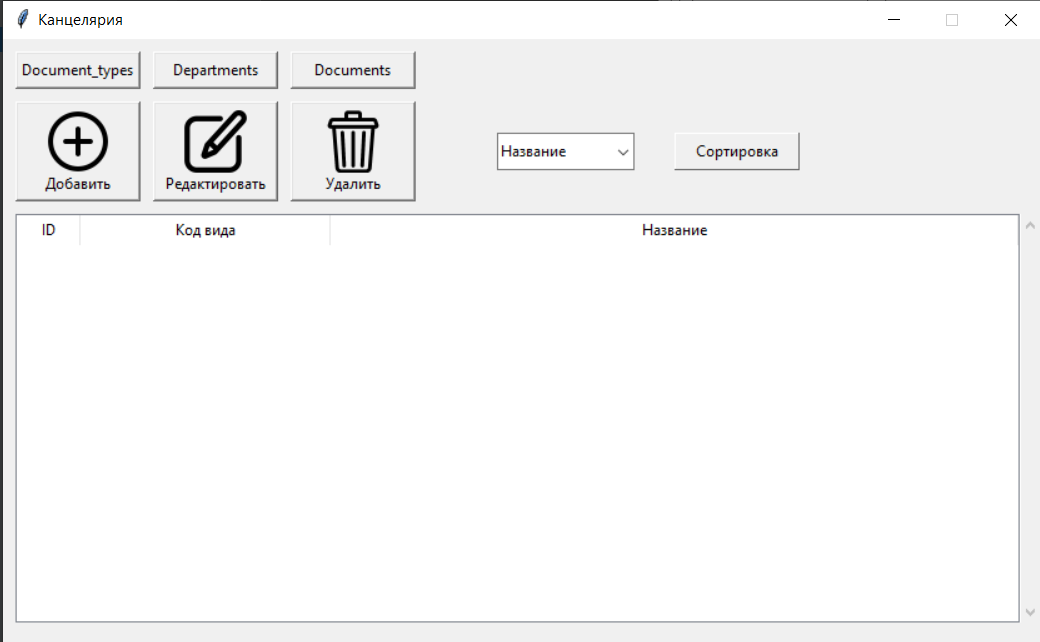


Рисунок 16. Окно Таблицы «Document\_types»

* + 1. **Выполнение функции добавления элемента в таблицу.**

Нажмите на кнопку «Добавить», появится новое поле в выбранной таблице, введите всю необходимую информацию в таблицу и нажмите на кнопку «Добавить», если этого не сделать, то все что вы ввели не будет сохранено.

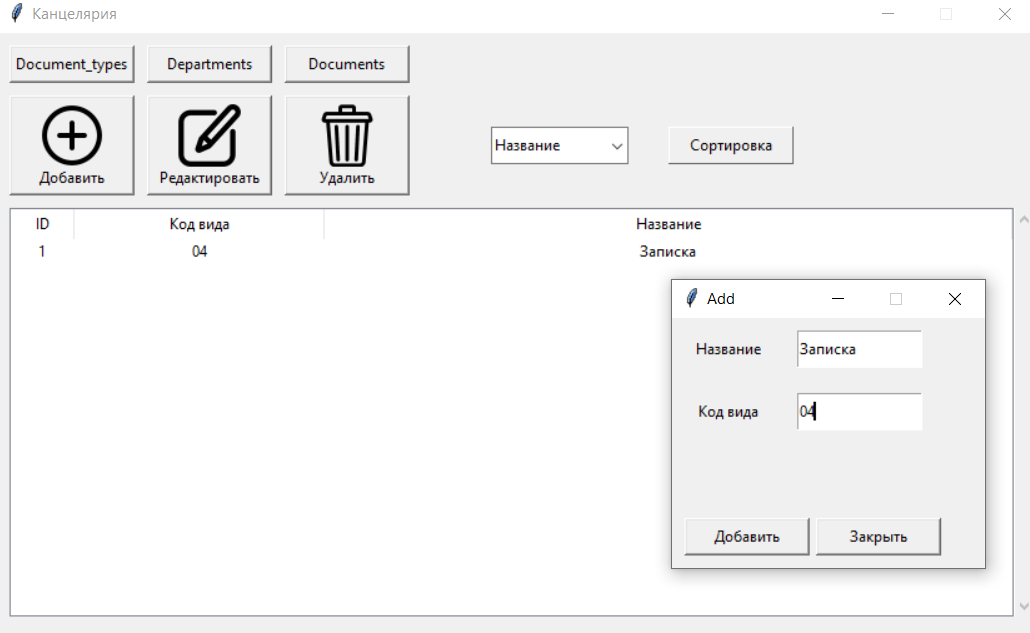


Рисунок 17. Окно Добавление значения в таблицу «Document\_types»

* + 1. **Выполнение функции удаления элемента из таблицы.**

Нажмите на строку, которую вы хотите удалить и нажмите на кнопку «Удалить». Будьте внимательны, удаленные данные нельзя восстановить!

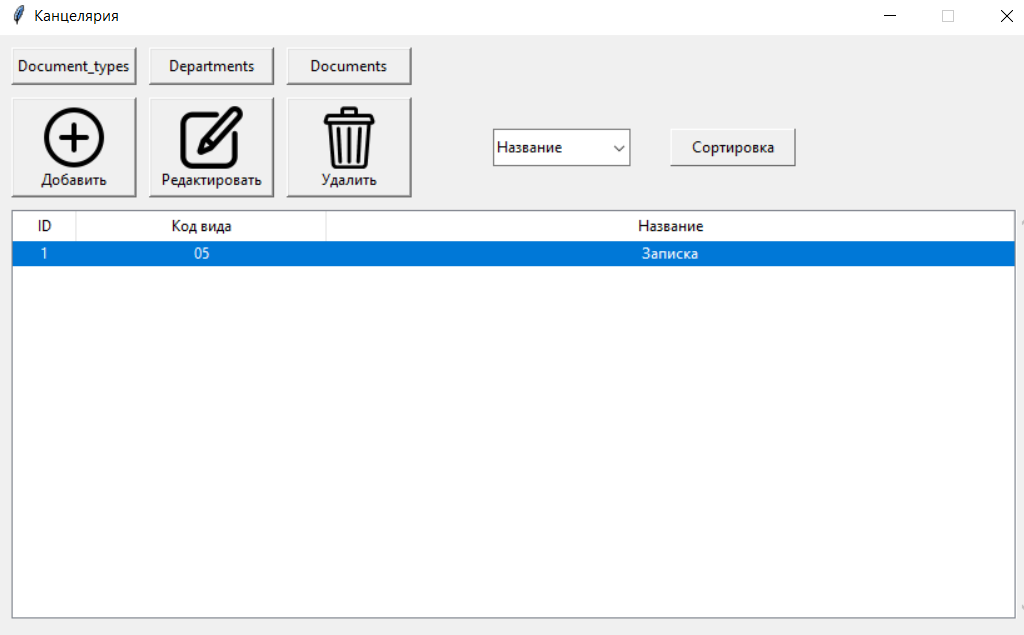
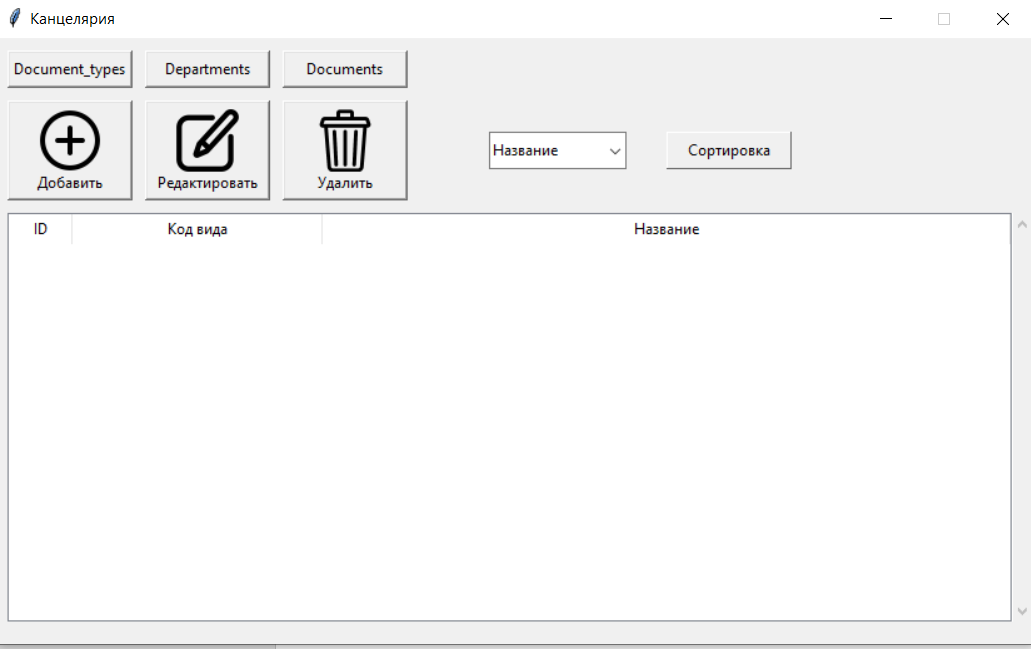


Рисунок 18. Таблица до удаления значения из таблицы «Document\_types»

Рисунок 19. Таблица после удаления значения из таблицы «Document\_types»

* + 1. **Выполнение функции редактирования элемента из таблицы.**

Нажмите на строку, которую вы хотите редактировать и нажмите на кнопку «Редактировать».

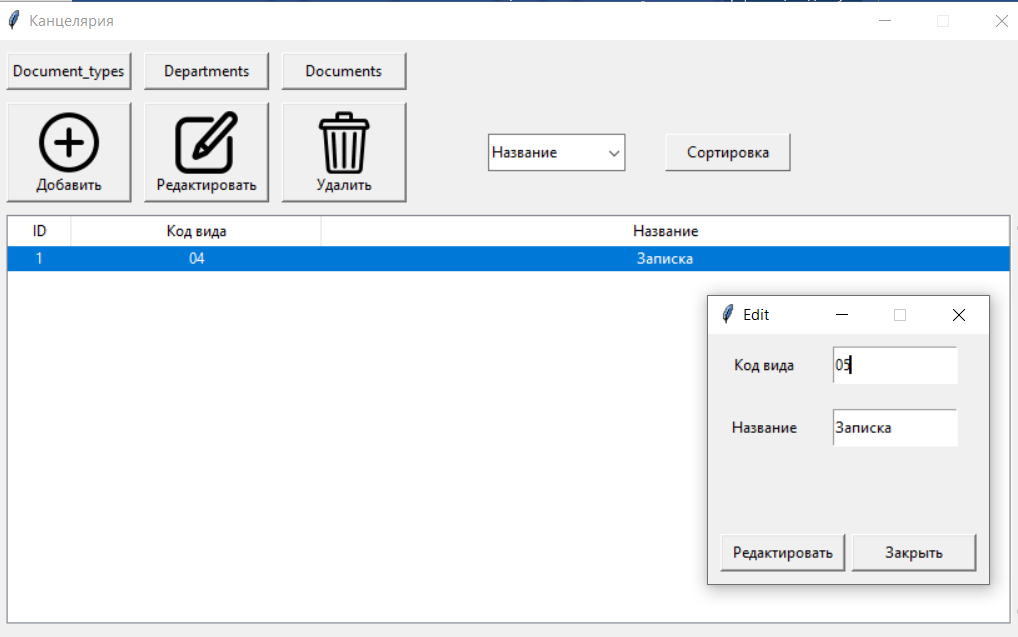


Рисунок 20. Редактирование значения

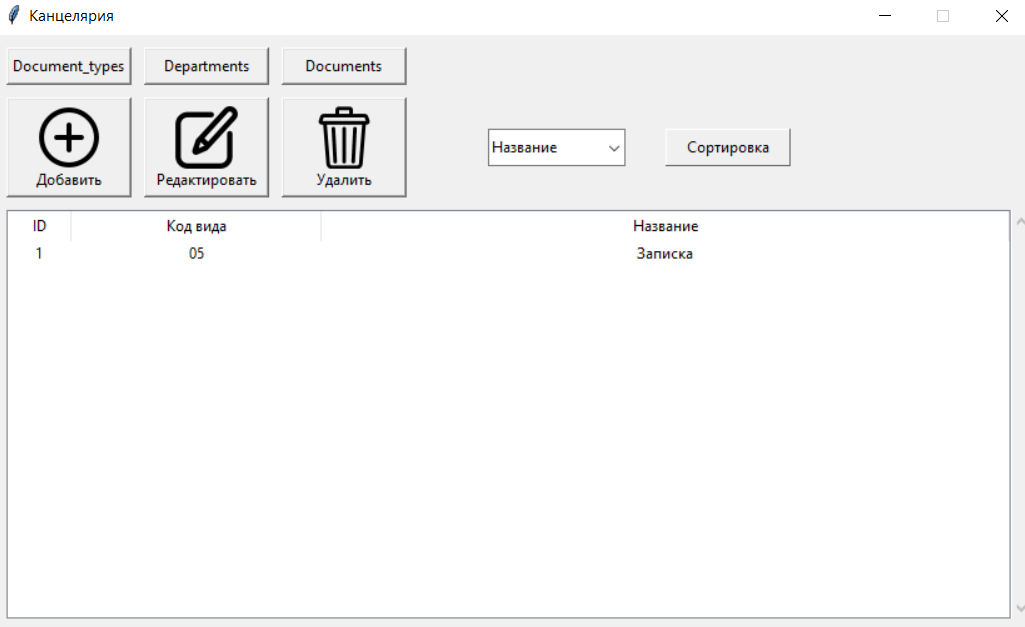


Рисунок 21. Результат редактирование значения

* 1. **Завершение работы программы**

Чтобы завершить работу программы нажмите на кнопку «Закрыть» в верхнем правом углу окна.

# **To-do лист**

1. Переписать интерфейс на PyQt.
2. Сделать меню с возможностью создания нескольких баз данных, по нажатию на которые откроется новое окно с вносимыми изменениями.
3. Импорт/экспорт баз данных.
4. Исправить редактирование в таблице «Documents»

# **Заключение**

В результате выполнения курсового проекта была написана программа «Канцелярия» для упрощения и удобства ведения документооборота.

В ходе работы были проанализированы: предметная область, существующие разработки, получены навыки по созданию интерфейса с помощью библиотеки tkinter и базами данных с помощью библиотеки sqlite.

Также планируется продолжение работы над данным проектом с целью расширения возможностей и удобства приложения для пользователей. Планы по доработкам представлены в «3.2. To-Do лист».

# **Список литературы и интернет источников**

1. Tutorialspoint – библиотека онлайн учебников.

[Электронный ресурс]. (дата) –

URL: https://www.tutorialspoint.com/sqlite/index.htm

(Дата обращения: 20.06.2022)

1. PyCharm - IDE для профессиональной разработки на Python

[Электронный ресурс]. (дата) –

URL: https://www.jetbrains.com/ru-ru/pycharm/

(Дата обращения: 20.06.2022)

1. Alphapedia -Свободная Российская Энциклопедия

[Электронный ресурс]. (дата) –

URL: https://alphapedia.ru/w/SQL-92

(Дата обращения: 20.06.2022)

1. Мой проект на GitHub:

https://github.com/Chupakabrik/Course-Work-2022

# **Приложения**

**Приложение 1. «main.py»**

# ------------------------------------ << БИБЛИОТЕКИ >> ------------------------------------- #  
import tkinter as tk  
from tkinter import \*  
from tkinter import ttk  
import sqlite3  
from db import \*  
  
  
# ------------------------------------------------------------------------------------------- #  
  
  
# ------------------------------------ << РАБОТА С БД >> ------------------------------------ #  
  
# ------------------------------------------------------------------------- #  
  
class Document\_types():  
  
 def \_\_init\_\_(self):  
 self.db = db # экземпляр класса DB  
 self.variable = []  
 self.selected\_item = 0  
 self.viewRecords()  
 #self.funcs()  
  
 # ------------------------ ФУНКЦИИ БД (Виды документов) ----------------------- #  
 def viewRecords(self):  
 ''' Вывод данных '''  
 self.db.cur.execute(  
 '''SELECT \* FROM Document\_types''')  
 self.variable.clear() # очищаем прошлые данные (чтобы не дублировались)  
 [self.variable.append(row) for row in self.db.cur.fetchall()] # записываем новый результат  
  
 # ------------------------- TKINTER (Виды документов) ------------------------- #  
  
 def tableDocument\_types(self):  
 ''' Создание таблицу с помощью ttk.Treeview() '''  
 # Задаем расположение таблицы  
 frame = tk.Frame(root, width=100, height=100)  
 frame.place(x=10, y=140)  
  
 # Создаем заголовоки для таблицы  
 headers = ['ID', 'Код вида', 'Нaзвание']  
 self.table = ttk.Treeview(frame, columns=headers, height=15, show='headings')  
  
 self.table.column('ID', width=50, anchor='center')   
 self.table.column('Код вида', width=200, anchor='center')   
 self.table.column('Нaзвание', width=550, anchor='center')  
   
 for header in headers: # заполняем заголовоки  
 self.table.heading(header, text=header)  
   
 for i in self.variable: # заполняем значения  
 self.table.insert('', tk.END, values=i)  
  
 # Создаем скролл для таблицы  
 scroll = ttk.Scrollbar(frame, command=self.table.yview)  
 self.table.configure(yscrollcommand=scroll.set)  
 scroll.pack(side=tk.RIGHT, fill=tk.Y)  
 self.table.pack(expand=tk.YES, fill=tk.BOTH)  
   
 def editDocument\_types(self):  
 ''' Редактирование таблицы '''  
 def closeFunc():  
 z.destroy() # закрываем окно  
  
 # Создаем новое окно  
 z = Toplevel()  
 z.geometry('225x200')  
 z.title('Edit')  
 z.resizable(False, False)  
  
 # Получаем данные строки по клику  
 selected\_item = self.table.selection()[0]  
 values = self.table.item(selected\_item, option='values')  
  
 # Заносим данные строки в переменные  
 id = values[0]  
 view\_code = values[1]  
 name = values[2]  
  
 # Создаем метки и их расположение  
 view\_code\_label = Label(z, text='Код вида').place(x=10, y=10, width=70, height=30)  
 name\_label = Label(z, text='Нaзвание').place(x=10, y=60, width=70, height=30)  
  
 # Переменные для ввода значений  
 view\_code\_1 = StringVar()  
 name\_2 = StringVar()  
  
 # Строки для ввода значений  
 view\_code\_entry = Entry(z, width=50, textvariable=view\_code\_1)  
 name\_entry = Entry(z, width=50, textvariable=name\_2)  
  
 # Записываем значения из строк ввода  
 view\_code\_entry.insert(0, str(view\_code))  
 name\_entry.insert(0, str(name))  
  
 # Задаем расположение строк ввода  
 view\_code\_entry.place(x=100, y=10, width=100, height=30)  
 name\_entry.place(x=100, y=60, width=100, height=30)  
  
 # Создаем кнопку "Редактировать"  
 edit\_button = Button(z, text='Редактировать', command=lambda:  
 (self.db.updateRecord1(view\_code\_1.get(), name\_2.get(), id), self.viewRecords(), self.tableDocument\_types()))  
 edit\_button.place(x=10, y=160, width=100, height=30)  
  
 # Создаем кнопку "Закрыть"  
 close\_button = Button(z, text='Закрыть', command=closeFunc)  
 close\_button.place(x=115, y=160, width=100, height=30)  
   
 def idDelete(self):  
 ''' Удаление строки по id из БД '''  
 self.selected\_item = self.table.selection()[0] # получаем строку  
 values = self.table.item(self.selected\_item, option='values') # получаем значения строки  
 delete\_id = values[0] # получаем id строки  
 self.db.deleteRecords1(delete\_id) # удаляем строку по полученному id  
   
 def lineDelete(self):  
 ''' Удаление выбранной строки из таблицы tkinter '''  
 self.table.delete(self.selected\_item) # удаляем полученную строку  
  
  
# ------------------------------ Отделы ------------------------------ #  
# ----------------------------------------------------------------------- #  
   
class Departments():  
  
 def \_\_init\_\_(self):  
 self.db = db # экземпляр класса DB  
 self.variable = []  
 self.selected\_item = 0  
 self.viewRecords()  
  
 # ------------------------- ФУНКЦИИ БД (Отделы) ------------------------ #  
  
 def viewRecords(self):  
 ''' Вывод данных '''  
 self.db.cur.execute(  
 '''SELECT \* FROM Departments''')  
 self.variable.clear() # очищаем прошлые данные (чтобы не дублировались)  
 [self.variable.append(row) for row in self.db.cur.fetchall()] # записываем новый результат  
  
 # -------------------------- TKINTER (Отделы) -------------------------- #  
  
 def tableDepartments(self):  
 ''' Создание таблицу с помощью ttk.Treeview() '''  
 # Задаем расположение таблицы  
 frame = tk.Frame(root, width=100, height=100)  
 frame.place(x=10, y=140)  
  
 # Создаем заголовоки для таблицы  
 headers = ['ID', 'Код\_отдела', 'Название']  
 self.table = ttk.Treeview(frame, columns=headers, height=15, show='headings')  
   
 self.table.column('ID', width=50, anchor='center')  
 self.table.column('Код\_отдела', width=200, anchor='center')   
 self.table.column('Название', width=550, anchor='center')  
   
   
 for header in headers: # заполняем заголовоки  
 self.table.heading(header, text=header)  
   
 for i in self.variable: # заполняем значения  
 self.table.insert('', tk.END, values=i)  
  
 # Создаем скролл для таблицы  
 scroll = ttk.Scrollbar(frame, command=self.table.yview)  
 self.table.configure(yscrollcommand=scroll.set)  
 scroll.pack(side=tk.RIGHT, fill=tk.Y)  
 self.table.pack(expand=tk.YES, fill=tk.BOTH)  
   
 def editDepartments(self):  
 ''' Редактирование таблицы '''  
 def closeFunc():  
 z.destroy() # закрываем окно  
  
 # Создаем новое окно  
 z = Toplevel()  
 z.geometry('270x230')  
 z.title('Edit')  
 z.resizable(False, False)  
  
 # Получаем данные строки по клику  
 selected\_item = self.table.selection()[0]  
 values = self.table.item(selected\_item, option='values')  
  
 # Заносим данные строки в переменные  
 id = values[0]  
 department\_code = values[1]  
 title = values[2]  
  
  
 # Создаем метки и их расположение  
 department\_code\_label = Label(z, text='Код\_отдела').place(x=10, y=10, width=80, height=30)  
 title\_label = Label(z, text='Название').place(x=10, y=60, width=80, height=30)  
   
  
 # Переменные для ввода значений  
 department\_code\_1 = StringVar()  
 title\_3 = StringVar()  
  
  
 # Строки для ввода значений  
 department\_code\_entry = Entry(z, width=50, textvariable=department\_code\_1)  
 title\_entry = Entry(z, width=50, textvariable=title\_3)  
  
  
 # Записываем значения из строк ввода  
 department\_code\_entry.insert(0, str(department\_code))  
 title\_entry.insert(0, str(title))  
  
  
 # Задаем расположение строк ввода  
 department\_code\_entry.place(x=130, y=10, width=130, height=30)  
 title\_entry.place(x=130, y=60, width=130, height=30)  
  
 # Создаем кнопку "Редактировать"  
 edit\_button = Button(z, text='Редактировать', command=lambda:  
 (self.db.updateRecord2(department\_code\_1.get(), title\_3.get(), id), self.viewRecords(), self.tableDepartments()))  
 edit\_button.place(x=10, y=190, width=100, height=30)  
  
 # Создаем кнопку "Закрыть"  
 close\_button = Button(z, text='Закрыть', command=closeFunc)  
 close\_button.place(x=115, y=190, width=100, height=30)  
   
 def idDelete(self):  
 ''' Удаление строки по id из БД '''  
 self.selected\_item = self.table.selection()[0] # получаем строку  
 values = self.table.item(self.selected\_item, option='values') # получаем значения строки  
 delete\_id = values[0] # получаем id строки  
 self.db.deleteRecords2(delete\_id) # удаляем строку по полученному id  
   
 def lineDelete(self):  
 ''' Удаление выбранной строки из таблицы tkinter '''  
 self.table.delete(self.selected\_item) # удаляем полученную строку  
  
  
# ------------------------------ Документы ------------------------------ #  
# -------------------------------------------------------------------- #  
  
class Documents():  
  
 def \_\_init\_\_(self):  
 self.db = db # экземпляр класса DB  
 self.variable = []  
 self.selected\_item = 0  
 self.viewRecords()  
  
 # -------------------------- ФУНКЦИИ БД (Документы) -------------------------- #  
 def viewRecords(self):  
 ''' Вывод данных '''  
 self.db.cur.execute(  
 '''SELECT \* FROM Documents''')  
 self.variable.clear() # очищаем прошлые данные (чтобы не дублировались)  
 [self.variable.append(row) for row in self.db.cur.fetchall()] # записываем новый результат  
  
 # ---------------------------- TKINTER (Документы) --------------------------- #  
  
 def tableDocuments(self):  
 ''' Создание таблицу с помощью ttk.Treeview() '''  
 # Задаем расположение таблицы  
 frame = tk.Frame(root, width=100, height=100)  
 frame.place(x=10, y=140)  
  
 # Создаем заголовоки для таблицы  
 headers = ['ID', 'Код документа', 'Название', 'Номер', 'Код\_вида', 'Код\_отдела-отправителя', 'Код\_отдела-получателя', 'Дата регистрации']  
 self.table = ttk.Treeview(frame, columns=headers, height=15, show='headings')  
   
 self.table.column('ID', width=50, anchor='center')  
 self.table.column('Код документа', width=100, anchor='center')   
 self.table.column('Название', width=100, anchor='center')  
 self.table.column('Номер', width=50, anchor='center')  
 self.table.column('Код\_вида', width=100, anchor='center')  
 self.table.column('Код\_отдела-отправителя', width=135, anchor='center')  
 self.table.column('Код\_отдела-получателя', width=135, anchor='center')  
 self.table.column('Дата регистрации', width=130, anchor='center')  
  
   
 for header in headers: # заполняем заголовоки  
 self.table.heading(header, text=header)  
   
 for i in self.variable: # заполняем значения  
 self.table.insert('', tk.END, values=i)  
  
 # Создаем скролл для таблицы  
 scroll = ttk.Scrollbar(frame, command=self.table.yview)  
 self.table.configure(yscrollcommand=scroll.set)  
 scroll.pack(side=tk.RIGHT, fill=tk.Y)  
 self.table.pack(expand=tk.YES, fill=tk.BOTH)  
   
 def editDocuments(self):  
 ''' Редактирование таблицы '''  
 def closeFunc():  
 z.destroy() # закрываем окно  
   
  
 # Создаем новое окно  
 z = Toplevel()  
 z.geometry('260x350')  
 z.title('Edit')  
 z.resizable(False, False)  
  
 # Получаем данные строки по клику  
 selected\_item = self.table.selection()[0]  
 values = self.table.item(selected\_item, option='values')  
  
 # Заносим данные строки в переменные  
 id = values[0]  
 Document\_code = values[1]  
 Appellation = values[2]  
 Number = values[3]  
 View\_code = values[4]  
 Documentcodes = values[5]  
 Documentcoder = values[6]  
 Reg\_date = values[7]  
  
 # Создаем метки и их расположение  
 Document\_code\_label = Label(z, text='Код документа').place(x=20, y=10, width=100, height=30)  
 Appellation\_label = Label(z, text='Название').place(x=20, y=60, width=100, height=30)  
 Number\_label = Label(z, text='Номер').place(x=20, y=100, width=100, height=30)  
 View\_code\_label = Label(z, text='Код\_вида').place(x=20, y=140, width=100, height=30)  
 Documentcodes\_label = Label(z, text='Код\_отдела-отправителя').place(x=20, y=180, width=135, height=30)  
 Documentcoder\_label = Label(z, text='Код\_отдела-получателя').place(x=20, y=220, width=135, height=30)  
 Reg\_date\_label = Label(z, text='Дата регистрации').place(x=20, y=260, width=120, height=30)  
  
 # Переменные для ввода значений  
 Document\_code\_1 = StringVar()  
 Appellation\_2 = StringVar()  
 Number\_3 = StringVar()  
 View\_code\_4 = StringVar()  
 Documentcodes\_5 = StringVar()  
 Documentcoder\_6 = StringVar()  
 Reg\_date\_7 = StringVar()  
  
 # Строки для ввода значений  
 Document\_code\_entry = Entry(z, width=50, textvariable=Document\_code\_1)  
 Appellation\_entry = Entry(z, width=50, textvariable=Appellation\_2)  
 Number\_entry = Entry(z, width=50, textvariable=Number\_3)  
 View\_code\_entry = Entry(z, width=50, textvariable=View\_code\_4)  
 Documentcodes = Entry(z, width=50, textvariable=Documentcodes\_5)  
 Documentcoder = Entry(z, width=50, textvariable=Documentcoder\_6)  
 Reg\_date = Entry(z, width=50, textvariable=Reg\_date\_7)  
  
 # Записываем значения из строк ввода  
 Document\_code\_entry.insert(0, str(Document\_code))  
 Appellation\_entry.insert(0, str(Appellation))  
 Number\_entry.insert(0, str(Number))  
 View\_code\_entry.insert(0, str(View\_code))  
 #Documentcodes\_entry.insert(0, str(Documentcodes))  
 #Documentcoder\_entry.insert(0, str(Documentcoder))  
 #Reg\_date\_entry.insert(0, str(Reg\_date))  
  
 # Задаем расположение строк ввода  
 Document\_code\_entry.place(x=110, y=10, width=120, height=30)  
 Appellation\_entry.place(x=110, y=60, width=120, height=30)  
 Number\_entry.place(x=110, y=100, width=120, height=30)  
 View\_code\_entry.place(x=110, y=140, width=120, height=30)  
 #Documentcodes\_entry.place(x=100, y=180, width=120, height=30)  
 #Documentcoder\_entry.place(x=100, y=220, width=120, height=30)  
 #Reg\_date\_entry.place(x=100, y=260, width=120, height=30)  
  
  
 # Создаем кнопку "Редактировать"  
 edit\_button = Button(z, text='Редактировать', command=lambda:  
 (self.db.updateRecord3(Document\_code\_1.get(), Appellation\_2.get(), Number\_3.get(), View\_code\_4.get(), Documentcodes\_5.get(), Documentcoder\_6.get(), Reg\_date\_7.get(), id), self.viewRecords(), self.tableDocuments()))  
 edit\_button.place(x=10, y=300, width=100, height=30)  
  
 # Создаем кнопку "Закрыть"  
 close\_button = Button(z, text='Закрыть', command=closeFunc)  
 close\_button.place(x=115, y=300, width=100, height=30)  
   
 def idDelete(self):  
 ''' Удаление строки по id из БД '''  
 self.selected\_item = self.table.selection()[0] # получаем строку  
 values = self.table.item(self.selected\_item, option='values') # получаем значения строки  
 delete\_id = values[0] # получаем id строки  
 self.db.deleteRecords3(delete\_id) # удаляем строку по полученному id  
   
 def lineDelete(self):  
 ''' Удаление выбранной строки из таблицы tkinter '''  
 self.table.delete(self.selected\_item) # удаляем полученную строку  
  
  
  
  
# ------------------------------------- << TKINTER >> --------------------------------------- #  
  
  
  
# ----------------------------- Document\_types.tk ---------------------------- #  
# ------------------------------------------------------------------------- #  
  
class tkDocument\_types():  
  
 def \_\_init\_\_(self):  
 self.sale = sale # экземпляр класса Document\_types()  
 self.db = db  
 def Document\_typesButton(self):  
 ''' Кнопка для открытия таблицы и её функций '''  
 def unionFunc():  
 ''' Компонует все кнопки функции выше '''  
 def addWindow():  
 ''' Создание нового окна для кнопки <Добавить> '''  
 def closeFunc():  
 s.destroy() # закрываем окно  
  
 # Создаем новое окно  
 s = Toplevel()  
 s.geometry('250x200')  
 s.title('Add')  
 s.resizable(False, False)  
  
 view\_code\_label = Label(s, text='Код вида').place(x=10, y=60, width=70, height=30)  
 name\_label = Label(s, text='Нaзвание').place(x=10, y=10, width=70, height=30)  
   
 name\_1 = StringVar()  
 name\_2 = StringVar()  
  
 view\_code\_entry = Entry(s, width=50, textvariable=name\_1).place(x=100, y=60, width=100, height=30)  
 name\_entry = Entry(s, width=50, textvariable=name\_2).place(x=100, y=10, width=100, height=30)  
  
   
 add\_button = Button(s, text='Добавить', command=lambda:  
 (self.db.records1(name\_1.get(), name\_2.get()), self.sale.viewRecords(), self.sale.tableDocument\_types()))  
 add\_button.place(x=10, y=160, width=100, height=30)  
  
 close\_button = Button(s, text='Закрыть', command=closeFunc)  
 close\_button.place(x=115, y=160, width=100, height=30)  
  
 self.add\_image = tk.PhotoImage(file='add.png')  
 Document\_types\_add\_button = Button(root, text='Добавить', image=self.add\_image, compound='top', command=addWindow)  
 Document\_types\_add\_button.place(x=10, y=50, width=100, height=80)  
  
 self.edit\_image = tk.PhotoImage(file='edit.png')  
 Document\_types\_edit\_button = Button(root, text='Редактировать', image=self.edit\_image, compound='top', command=lambda:  
 self.sale.editDocument\_types())  
 Document\_types\_edit\_button.place(x=120, y=50, width=100, height=80)  
   
 self.delete\_image = tk.PhotoImage(file='delete.png')   
 Document\_types\_edit\_button = Button(root, text='Удалить', image=self.delete\_image, compound='top', command=lambda:  
 (self.sale.idDelete(), self.sale.lineDelete()))  
 Document\_types\_edit\_button.place(x=230, y=50, width=100, height=80)  
  
 def ChoosingTheRightOption():  
 if column\_menu.get() == 'Нaзвание':  
 s = 'Нaзвание'  
 return s  
   
 column\_menu = ttk.Combobox(root,values=['Нaзвание'])  
 column\_menu.place(x=395, y=75, width=110, height=30)  
 column\_menu.current(0)  
  
 mb = Menubutton(root, text='Сортировка', relief=RAISED)  
 mb.place(x=537, y=75, width=100, height=30)  
   
 mb.menu = Menu(mb, tearoff=0)  
 mb['menu'] = mb.menu  
   
 mb.menu.add\_command(label='ВЫВОД В ЭКСЕЛЬ СЛУЖЕБОЙ ЗАПИСКИ ПЛАНОВОГО ОТДЕЛА', command=lambda:  
 (self.sale.generalIncreaseSort(ChoosingTheRightOption()), self.sale.tableDocument\_types()))  
 mb.menu.add\_command(label='ВЫВОД ТАБЛИЦЫ', command=lambda:  
 (self.sale.generalIncrease(ChoosingTheRightOption()), self.sale.tableDocument\_types()))  
   
   
 Document\_types\_button = Button(root, text='Document\_types', command=lambda:  
 (unionFunc(), self.sale.viewRecords(), self.sale.tableDocument\_types()))  
 Document\_types\_button.place(x=10, y=10, width=100, height=30)  
  
# ------------------------------ Departments.tk ----------------------------- #  
# ------------------------------------------------------------------------- #  
  
class tkDepartments():  
  
 def \_\_init\_\_(self):  
 self.cust = cust # экземпляр класса Departments()  
 self.db = db  
 def DepartmentsButton(self):  
 ''' Кнопка для открытия таблицы и её функций '''  
 def unionFunc():  
 ''' Компонует все кнопки функции выше '''  
 def addWindow():  
 ''' Создание нового окна для кнопки <Добавить> '''  
 def closeFunc():  
 s.destroy() # закрываем окно  
  
 # Создаем новое окно  
 s = Toplevel()  
 s.geometry('250x200')  
 s.title('Add')  
 s.resizable(False, False)  
  
 department\_code\_label = Label(s, text='Код отдела').place(x=10, y=10, width=80, height=30)  
 title\_label = Label(s, text='Название').place(x=10, y=60, width=80, height=30)  
   
 name\_1 = StringVar()  
 name\_3 = StringVar()  
  
  
 department\_code\_entry = Entry(s, width=50, textvariable=name\_1).place(x=130, y=10, width=100, height=30)  
 title\_entry = Entry(s, width=50, textvariable=name\_3).place(x=130, y=60, width=100, height=30)  
  
  
 add\_button = Button(s, text='Добавить', command=lambda:  
 (self.db.records2(name\_1.get(), name\_3.get()), self.cust.viewRecords(), self.cust.tableDepartments()))  
 add\_button.place(x=10, y=150, width=100, height=30)  
  
 close\_button = Button(s, text='Закрыть', command=closeFunc)  
 close\_button.place(x=115, y=150, width=100, height=30)  
  
 self.add\_image = tk.PhotoImage(file='add.png')  
 Document\_types\_add\_button = Button(root, text='Добавить', image=self.add\_image, compound='top', command=addWindow)  
 Document\_types\_add\_button.place(x=10, y=50, width=100, height=80)  
  
 self.edit\_image = tk.PhotoImage(file='edit.png')  
 Document\_types\_edit\_button = Button(root, text='Редактировать', image=self.edit\_image, compound='top', command=lambda:  
 self.cust.editDepartments())  
 Document\_types\_edit\_button.place(x=120, y=50, width=100, height=80)  
   
 self.delete\_image = tk.PhotoImage(file='delete.png')   
 Document\_types\_edit\_button = Button(root, text='Удалить', image=self.delete\_image, compound='top', command=lambda:  
 (self.cust.idDelete(), self.cust.lineDelete()))  
 Document\_types\_edit\_button.place(x=230, y=50, width=100, height=80)  
   
  
 Departments\_button = Button(root, text='Departments', command=lambda:  
 (unionFunc(), self.cust.viewRecords(), self.cust.tableDepartments()))  
 Departments\_button.place(x=120, y=10, width=100, height=30)  
  
  
  
# ------------------------------- Documents.tk ------------------------------- #  
# ------------------------------------------------------------------------- #  
  
class tkDocuments():  
  
 def \_\_init\_\_(self):  
 self.orde = orde # экземпляр класса Documents()  
 self.db = db  
 def DocumentsButton(self):  
 ''' Кнопка для открытия таблицы и её функций '''  
 def unionFunc():  
 ''' Компонует все кнопки функции выше '''  
 def addWindow():  
 ''' Создание нового окна для кнопки <Добавить> '''  
 def closeFunc():  
 s.destroy() # закрываем окно  
  
  
  
 # Создаем новое окно  
 s = Toplevel()  
 s.geometry('280x350')  
 s.title('Add')  
 s.resizable(False, False)  
  
  
 Document\_code\_label = Label(s, text='Код документа').place(x=20, y=10, width=100, height=30)  
 Appellation\_label = Label(s, text='Название').place(x=20, y=60, width=100, height=30)  
 Number\_label = Label(s, text='Номер').place(x=20, y=100, width=100, height=30)  
 View\_code\_label = Label(s, text='Код\_вида').place(x=20, y=140, width=100, height=30)  
 Documentcodes\_label = Label(s, text='Код\_отдела-отправителя').place(x=20, y=180, width=135, height=30)  
 Documentcoder\_label = Label(s, text='Код\_отдела-получателя').place(x=20, y=220, width=135, height=30)  
 Reg\_date\_label = Label(s, text='Дата регистрации').place(x=20, y=260, width=120, height=30)  
   
   
   
   
 name\_1 = StringVar()  
 name\_2 = StringVar()  
 name\_3 = StringVar()  
 name\_4 = StringVar()  
 name\_5 = StringVar()  
 name\_6 = StringVar()  
 name\_7 = StringVar()  
  
  
 Document\_code\_entry = Entry(s, width=50, textvariable=name\_1).place(x=155, y=10, width=120, height=30)  
 Appellation\_entry = Entry(s, width=50, textvariable=name\_2).place(x=155, y=60, width=120, height=30)  
 Number\_entry = Entry(s, width=50, textvariable=name\_3).place(x=155, y=100, width=120, height=30)  
 View\_code\_entry = Entry(s, width=50, textvariable=name\_4).place(x=155, y=140, width=120, height=30)  
 Documentcodes\_entry = Entry(s, width=50, textvariable=name\_5).place(x=155, y=180, width=120, height=30)  
 Documentcoder\_entry = Entry(s, width=50, textvariable=name\_6).place(x=155, y=220, width=120, height=30)  
 Reg\_date\_entry = Entry(s, width=50, textvariable=name\_7).place(x=155, y=260, width=120, height=30)  
  
   
   
 add\_button = Button(s, text='Добавить', command=lambda:  
 (self.db.records3(name\_1.get(), name\_2.get(), name\_3.get(), name\_4.get(), name\_5.get(), name\_6.get(), name\_7.get()), self.orde.viewRecords(), self.orde.tableDocuments()))  
 add\_button.place(x=10, y=300, width=100, height=30)  
  
 close\_button = Button(s, text='Закрыть', command=closeFunc)  
 close\_button.place(x=115, y=300, width=100, height=30)  
  
 self.add\_image = tk.PhotoImage(file='add.png')  
 Document\_types\_add\_button = Button(root, text='Добавить', image=self.add\_image, compound='top', command=addWindow)  
 Document\_types\_add\_button.place(x=10, y=50, width=100, height=80)  
  
 self.edit\_image = tk.PhotoImage(file='edit.png')  
 Document\_types\_edit\_button = Button(root, text='Редактировать', image=self.edit\_image, compound='top', command=lambda:  
 self.orde.editDocuments())  
 Document\_types\_edit\_button.place(x=120, y=50, width=100, height=80)  
   
 self.delete\_image = tk.PhotoImage(file='delete.png')   
 Document\_types\_edit\_button = Button(root, text='Удалить', image=self.delete\_image, compound='top', command=lambda:  
 (self.orde.idDelete(), self.orde.lineDelete()))  
 Document\_types\_edit\_button.place(x=230, y=50, width=100, height=80)  
  
 def ChoosingTheRightOption():  
 if column\_menu.get() == 'Код документа':  
 s = 'Код документа'  
 return s  
 if column\_menu.get() == 'Название':  
 s = 'Название'  
 return s  
 if column\_menu.get() == 'Номер':  
 s = 'Номер'  
 return s  
 if column\_menu.get() == 'Код\_вида':  
 s = 'Код\_вида'  
 return s  
 if column\_menu.get() == 'Код\_отдела-отправителя':  
 s = 'Код\_отдела-отправителя'  
 return s  
 if column\_menu.get() == 'Код\_отдела-получателя':  
 s = 'Код\_отдела-получателя'  
 return s  
 if column\_menu.get() == 'Дата регистрации':  
 s = 'Дата регистрации'  
 return s  
  
  
   
 column\_menu = ttk.Combobox(root, values=['Код документа','Название', 'Номер', 'Код\_вида', 'Код\_отдела-отправителя', 'Код\_отдела-получателя', 'Дата регистрации'])  
 column\_menu.place(x=395, y=75, width=110, height=30)  
 column\_menu.current(0)  
   
 mb = Menubutton (root, text='Сортировка', relief=RAISED)  
 mb.place(x=537, y=75, width=100, height=30)  
   
 mb.menu = Menu( mb,tearoff =0 )  
 mb['menu'] = mb.menu  
   
 mb.menu.add\_command(label='Сортировать по возрастанию', command=lambda:  
 (self.orde.generalIncreaseSort(ChoosingTheRightOption()), self.orde.tableDocuments()))  
   
 mb.menu.add\_command(label='Сортировать по убыванию', command=lambda:  
 (self.orde.generalDescendingSort(ChoosingTheRightOption()), self.orde.tableDocuments()))  
  
   
   
 Documents\_button = Button(root, text='Documents', command=lambda:  
 (unionFunc(), self.orde.viewRecords(), self.orde.tableDocuments()))  
 Documents\_button.place(x=230, y=10, width=100, height=30)  
  
   
# --------------------------------------- << БД >> ------------------------------------------ #  
#ОТДЕЛЬНЫЙ ФАЙЛ "db"  
# ------------------------------------------------------------------------------------------- #  
  
if \_\_name\_\_ == "\_\_main\_\_":  
   
 db = DB1()  
  
 root = tk.Tk()  
 root.geometry('830x485')  
 root.title('Канцелярия')  
 root.resizable(False, False)  
  
 sale = Document\_types()  
 cust = Departments()  
 orde = Documents()  
  
 s = tkDocument\_types()  
 c = tkDepartments()  
 o = tkDocuments()  
  
 s.Document\_typesButton()  
 c.DepartmentsButton()  
 o.DocumentsButton()  
  
  
 root.mainloop()  
  
# ------------------------------------------------------------------------------------------- #

**Приложение 2. «db.py»**

import sqlite3  
  
  
class DB1:  
   
 def \_\_init\_\_(self):  
 # Создаем подключение к БД  
 self.conn = sqlite3.connect('Chancellery.db')  
 self.cur = self.conn.cursor()  
 self.selected\_item = 0  
  
 # Создаем таблицу Document\_types(Виды документов)  
 self.cur.execute(  
 '''CREATE TABLE IF NOT EXISTS Document\_types (ID\_1 integer primary key, 'Код\_вида' text, 'Нaзвание' text)''')  
   
 # Создаем таблицу Departments(Отделы)  
 self.cur.execute(  
 '''CREATE TABLE IF NOT EXISTS Departments (ID\_2 integer primary key, 'Код\_отдела' text, 'Нaзвание' text)''')  
  
 # Создаем таблицу Documents(Документы)  
 self.cur.execute(  
 '''CREATE TABLE IF NOT EXISTS Documents (ID\_3 integer primary key, 'Код документа' text, 'Название' text, 'Номер' text, 'Код\_вида' text, 'Код\_отдела-отправителя' text, 'Код\_отдела-получателя' text, 'Дата регистрации' text)''')  
 self.conn.commit()  
  
 def insertDataDocument\_types(self, view\_code, name):  
 ''' Добавление данных для Document\_types '''  
 self.cur.execute(  
 '''INSERT INTO Document\_types('Код\_вида', 'Нaзвание') VALUES (?, ?)''', (view\_code, name))  
 self.conn.commit()  
  
 def insertDataDepartments(self, department\_code, title):  
 ''' Добавление данных для Departments '''  
 self.cur.execute(  
 '''INSERT INTO Departments('Код\_отдела', 'Нaзвание') VALUES (?, ?)''', (department\_code, title))  
 self.conn.commit()  
  
 def insertDataDocuments(self, Document\_code, Appellation, Number, View\_code, Documentcodes, Documentcoder, Reg\_date):  
 ''' Добавление данных для Documents '''  
 self.cur.execute(  
 '''INSERT INTO Documents('Код документа', 'Название', 'Номер', 'Код\_вида', 'Код\_отдела-отправителя', 'Код\_отдела-получателя', 'Дата регистрации') VALUES (?, ?, ?, ?, ?, ?, ?)''', (Document\_code, Appellation, Number, View\_code, Documentcodes, Documentcoder, Reg\_date))  
 self.conn.commit()  
  
 #Методы DB (Виды документов)  
 def records1(self, view\_code, name):  
 ''' Ввод новых данных '''  
 self.insertDataDocument\_types(view\_code, name)  
  
 def updateRecord1(self, view\_code, name, ID\_1):  
 ''' Редактирование данных '''  
 self.cur.execute(  
 '''UPDATE Document\_types SET 'Код\_вида'=?, 'Нaзвание'=? WHERE ID\_1=?''', (view\_code, name, ID\_1))  
 self.conn.commit()  
  
 def deleteRecords1(self, ID\_1):  
 ''' Удаление результата '''  
 self.cur.execute(  
 '''DELETE FROM Document\_types WHERE ID\_1=?''', (ID\_1,))  
 self.conn.commit()  
   
  
 #Методы DB (Отделы)  
 def records2(self, department\_code, title):  
 ''' Ввод новых данных '''  
 self.insertDataDepartments(department\_code, title)  
  
 def updateRecord2(self, department\_code, title, ID\_2):  
 ''' Редактирование данных '''  
 self.cur.execute(  
 '''UPDATE Departments SET 'Код\_отдела'=?, 'Нaзвание'=? WHERE ID\_2=?''', (department\_code, title, ID\_2))  
 self.conn.commit()  
  
 def deleteRecords2(self, ID\_2):  
 ''' Удаление результата '''  
 self.cur.execute(  
 '''DELETE FROM Departments WHERE ID\_2=?''', (ID\_2,))  
 self.conn.commit()  
   
 #Методы DB (Документы)  
 def records3(self, Document\_code, Appellation, Number, View\_code, Documentcodes, Documentcoder, Reg\_date):  
 ''' Ввод новых данных '''  
 self.insertDataDocuments(Document\_code, Appellation, Number, View\_code, Documentcodes, Documentcoder, Reg\_date)  
  
 def updateRecord3(self, Document\_code, Appellation, Number, View\_code, Documentcodes, Documentcoder, Reg\_date, ID\_3):  
 ''' Редактирование данных '''  
 self.cur.execute(  
 '''UPDATE Documents SET 'Код документа'=?, 'Название'=?, 'Номер'=?, 'Код\_вида'=?, 'Код\_отдела-отправителя'=?, 'Код\_отдела-получателя'=?, 'Дата регистрации'=? WHERE ID\_3=?''', (Document\_code, Appellation, Number, View\_code, Documentcodes, Documentcoder, Reg\_date, ID\_3))  
 self.conn.commit()  
  
 def deleteRecords3(self, ID\_3):  
 ''' Удаление результата '''  
 self.cur.execute(  
 '''DELETE FROM Documents WHERE ID\_3=?''', (ID\_3,))  
 self.conn.commit()

1. 1) ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов [↑](#footnote-ref-1)
2. 2) ГОСТ 19.103-77 ЕСПД. Обозначение программ и программных документов [↑](#footnote-ref-2)
3. 3) ГОСТ 19.104-78\* ЕСПД. Основные надписи [↑](#footnote-ref-3)
4. 4) ГОСТ 19.105-78\* ЕСПД. Общие требования к программным документам [↑](#footnote-ref-4)
5. 5) ГОСТ 19.106-78\* ЕСПД. Общие требования к программным документам, выполненным печатным способом [↑](#footnote-ref-5)
6. 6) ГОСТ 19.505-79\* ЕСПД. Руководство оператора. Требования к содержанию и оформлению [↑](#footnote-ref-6)
7. 7) ГОСТ 19.604-78\* ЕСПД. Правила внесения изменений в программные документы, выполненные печатным способом [↑](#footnote-ref-7)