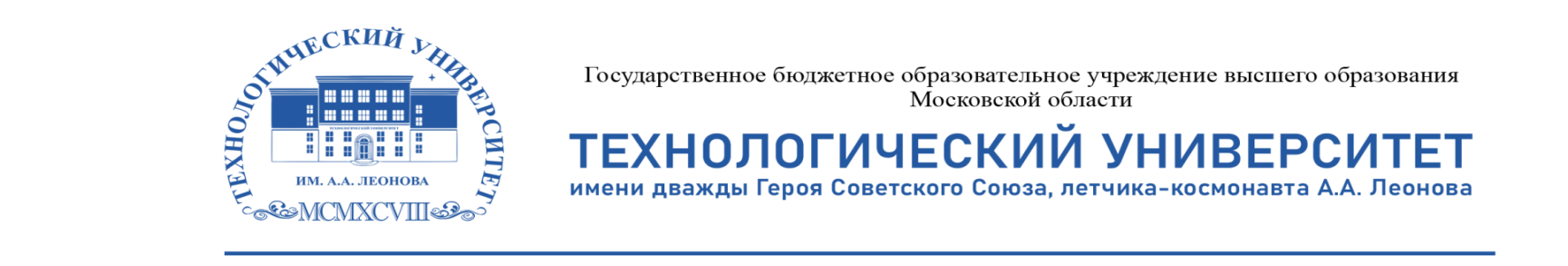
****

Колледж космического машиностроения и технологии

**Курсовой проект**

**Разработка приложения «менеджер баз данных sqlite»**

Пояснительная записка

КП.09.02.03.22.08ПЗ

Обучающийся группы П2-19 Ковалев А. Г.

Руководитель курсового проекта Гусятинер Л. Б.

Результат защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Королев, 2022 г.

Оглавление

[Введение 3](#_Toc106825302)

[1. Теоретическая часть 4](#_Toc106825303)

[1.1. Описание предметной области 4](#_Toc106825304)

[1.2. Описание существующих разработок 8](#_Toc106825305)

[1.2.1. DB Browser for SQLite [2] 9](#_Toc106825306)

[1.2.2. SQLiteStudio [3] 10](#_Toc106825307)

[2. Проектная часть 11](#_Toc106825308)

[2.1. Построение диаграммы прецедентов 11](#_Toc106825309)

[2.2. Выбор инструментов 12](#_Toc106825310)

[2.3. Проектирование сценария 13](#_Toc106825311)

[2.4. Построение диаграммы классов 14](#_Toc106825312)

[2.5. Описание главного модуля 15](#_Toc106825313)

[2.6. Описание спецификаций к модулям 21](#_Toc106825314)

[2.7. Описание модулей 22](#_Toc106825315)

[2.7. Описание тестовых наборов модулей 23](#_Toc106825316)

[2.8. Описание применения средств отладки 26](#_Toc106825317)

[2.9. Анализ оптимальности использования памяти и быстродействия 27](#_Toc106825318)

[3. Эксплуатационная часть 28](#_Toc106825319)

[3.1. Руководство оператора 28](#_Toc106825320)

[3.2. To-Do лист 35](#_Toc106825321)

[Заключение 36](#_Toc106825322)

[Список использованной литературы и интернет-ресурсов 37](#_Toc106825323)

[Приложения 38](#_Toc106825324)

# Введение

Целью данного курсового проекта является написание менеджера баз данных для создания баз данных с тонкой настройкой параметров таблиц, столбцов и строк, а также вводом данных и их сортировкой по вводимому запросу. Данная тема является актуальной и даже необходимой. Курсовой проект поможет облегчить настройку таблицы в базе данных, а именно: столбцов, их модификаций – по типу: integer, primary key, text, autoincrement, bool; ввод данных с возможностью их редактирования; удаление таблицы или ещё очистка; сортировка по введённым запросам пользователя.

В первой части будет рассмотрена предметная область данной темы, а также несколько актуальных продуктов по этой области.

Во второй части будут рассмотрены инструменты и модули, которые были разработаны, структура программной части и листинги ключевых частей программных модулей.

В третьей части будет рассмотрено руководство для пользователей по данному проекту.

В заключительной части будет приведен To-do лист с планами по доработки программы, а также сделаны общие выводы о получившемся проекте.

# Теоретическая часть

* 1. Описание предметной области

Менеджер баз данных – это компьютерная программа, которая обеспечивает базовые функции управления базами данных, включая создание, редактирование, удаление и прочие обслуживания баз данных.

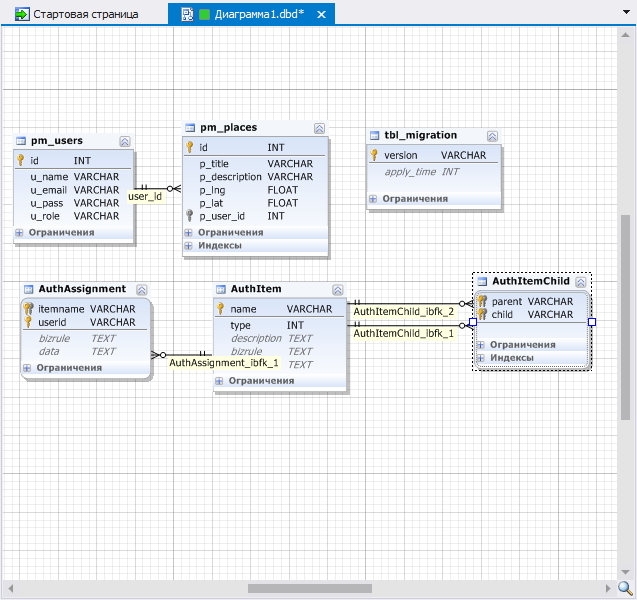
Для управления базами данных зачастую нанимают специалиста, чтобы он обслуживал её, производил запросы и прочее. Она не сказать, что подходит обычным пользователям, не знакомым с предметной областью. Для грамотного использования полного функционала менеджера баз данных, нужно знать язык SQL и желательно языки программирования, для более тонкого взаимодействия с данными.

Используется же менеджер баз данных практически повсюду: корпорации, больницы, школы, государственные органы. Все стремятся автоматизировать поступающие данные и именно менеджер баз данных помогает в осуществлении данной задачи.

Нельзя разделить менеджеры баз данных на какие-то определенные виды, но можно рассмотреть по наличию определённого функционала:

* Возможность представить БД в графическом виде и создание диаграмм;

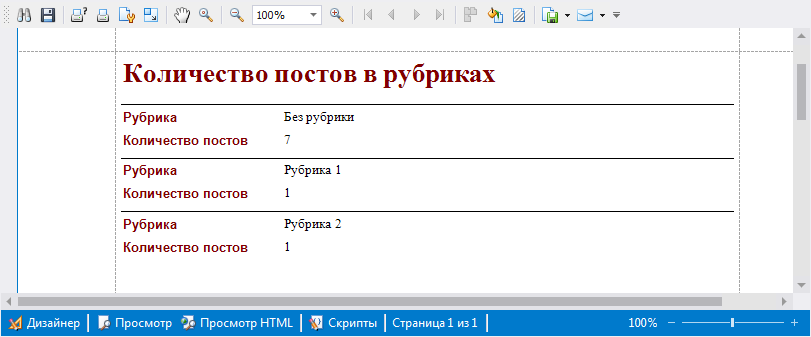
Данный инструмент очень полезен при разработке структуры баз данных. Можно добавлять новые таблицы на диаграмму, устанавливать связи, создавать поля, либо сформатировать диаграмму из существующих таблиц.



*Рисунок 1. Создание диаграмм.* [1]

* Возможность создания отчётов по БД;

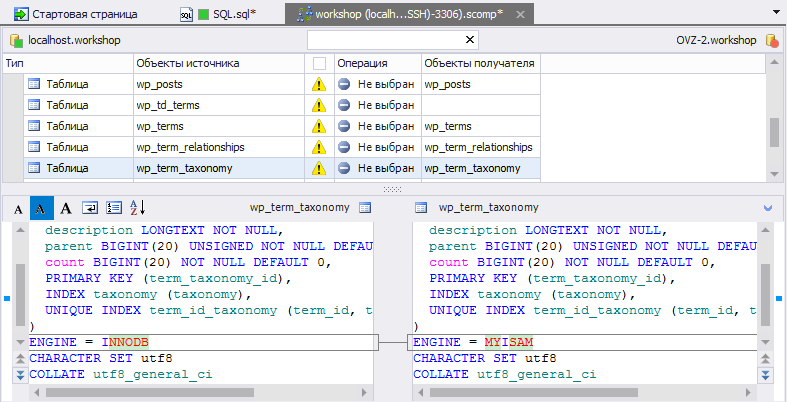
Данная возможность очень похожа на создание отчёта в Access. Отчёты создаются при помощи специального инструмента, в котором можно выбрать существующие таблицы или создать отчёт по собственному запросу. Также существует возможность выбора оформления, сортировки и прочего. Имеется возможность печати результата или его экспорта.



*Рисунок 2. Отчёт.* [1]

* Возможность сравнения БД;

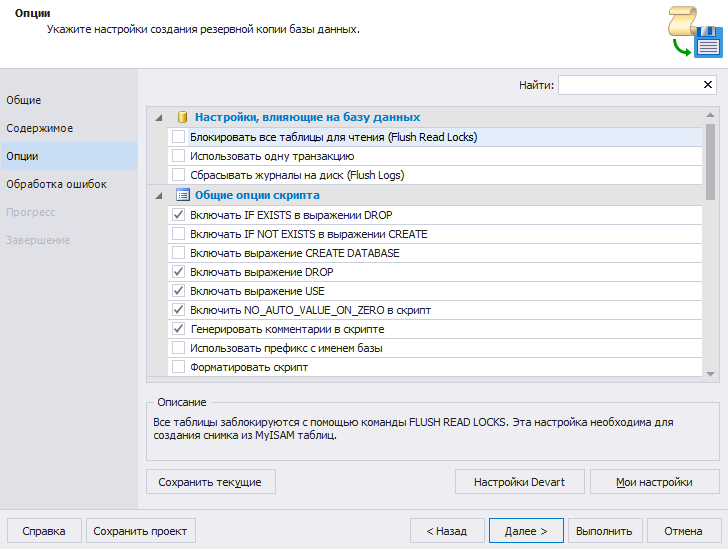
Вообще эта функция не особо и нужна, из-за того, что структуры баз данных у разработчиков должны совпадать, но нельзя всё предсказать, а потому возможность быстрого сравнения баз данных не помешает для экономии времени при разработке.



*Рисунок 3. Сравнение.* [1]

* Возможность резервного копирования и восстановления данных БД;

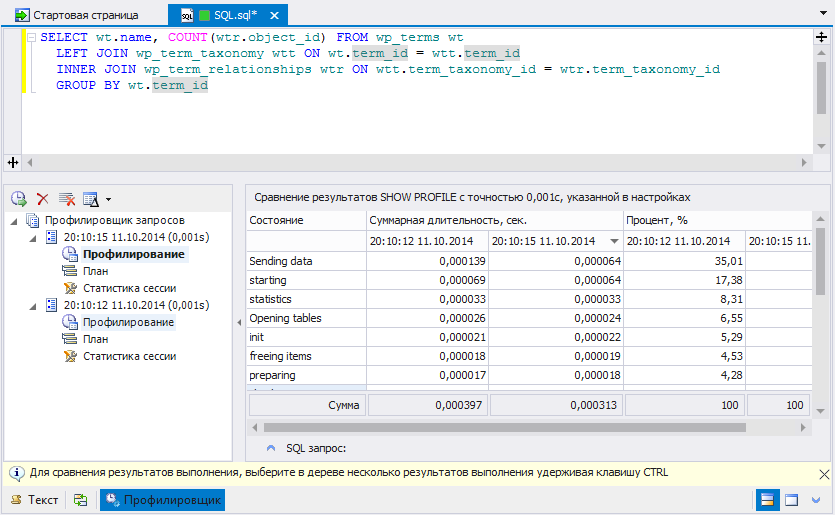
Функционал говорит сам за себя. Единственное что можно добавить, так это то, что существует более тонкая настройка создания копии. При создании копии можно заблокировать таблицы и прочее.



*Рисунок 4. Копирование и восстановление.* [1]

* Возможность профилирования запросов.

Информацию можно получить при помощи SQL запросов, но некоторые программы позволяют выбрать несколько результатов и сравнить их. В отдельное окно будут выводиться данные с возможностью взаимодействия с ними.



*Рисунок 5. Профилирование.* [1]

Существует ещё множество разных возможностей менеджера баз данных, которые переходят от одной программы к другой. Я показал лишь наиболее интересные. В каких-то менеджерах баз данных есть все эти функции, а в каких-то может не быть.

Всё это сделано для более удобного использования и автоматизации процесса. Какие-то менеджеры баз данных создают для администрирования сетей, другие для предприятий, третьи для компаний, иными словами, к чему-то приспособлены больше.

* 1. Описание существующих разработок

В этом разделе рассмотрены некоторые уже имеющиеся программы по менеджеру баз данных. Интерфейс подобных программ чаще всего сложный и непонятный для обычных пользователей. Для специалистов же, он будет вполне понятен, а потому будут рассмотрены удобные, по моему мнению, программы.

Цели использования менеджера баз данных:

* управления локальными и удаленными;
* управление таблицами;
* резервное копирование;
* выполнение запросов;
* определение новых исправлений и прочее…

Основные требования:

* знание языка SQL;
* понимание для чего используется программа;
* знание прочих языков программирования.

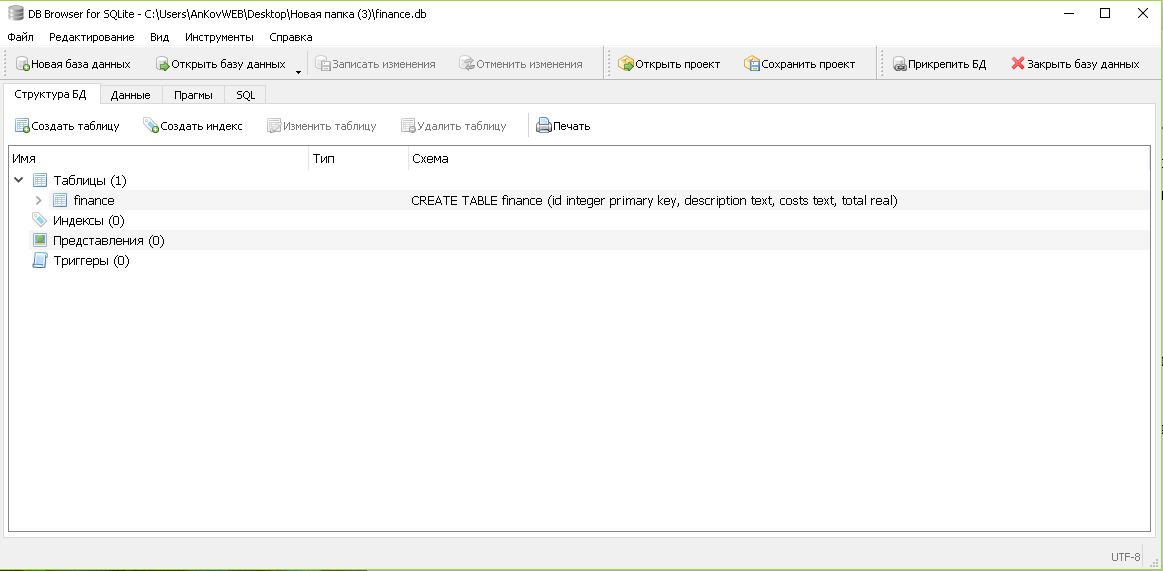
### DB Browser for SQLite [2]

DB Browser for SQLite – это менеджер баз данных SQLite. Представляет собой визуальный инструмент для создания, редактирования и разработки файлов БД.

Данный инструмент для тех, кому нужно создавать БД, редактировать и искать данные в них. Он достаточно прост, а потому знание SQL-команд не столь обязательны.

Программа позволяет:

* создавать файлы;
* создавать/изменять/определять/удалять таблицы;
* создавать/определять/удалять индексы;
* просматривать/редактировать/добавлять/удалять/искать записи;
* импортировать/экспортировать записи, таблицы, БД;
* выполнять SQL-запросы и видеть их результат.



*Рисунок 6. Интерфейс DB Browser for SQLite.*

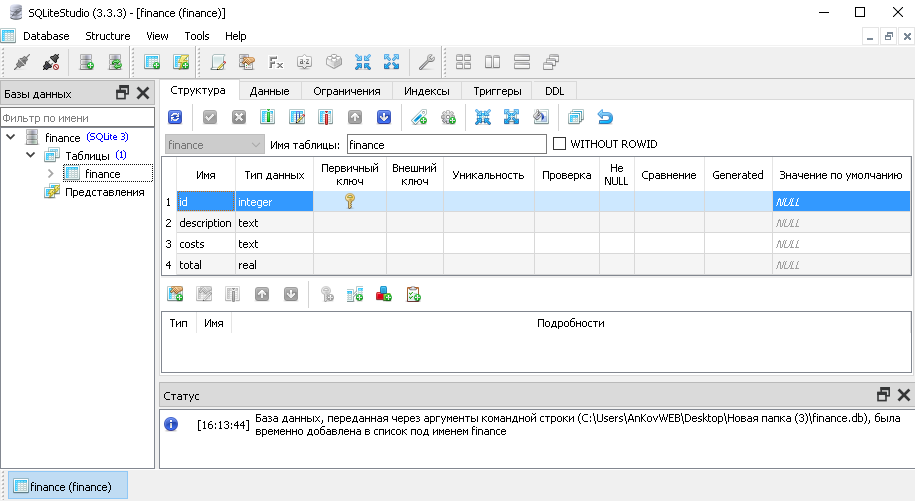
Данная программа не требует знакомства с командами SQL или каких-то дополнительных требований. Это инструмент, который может использоваться как пользователем, так и разработчиком. Он остается максимально простым в использовании и интуитивно понятным.

### SQLiteStudio [3]

SQLiteStudio – это менеджер баз данных SQLite. Проект стартовал в начале 2007 года. Разрабатывался как SQLite-менеджер, которым легко пользоваться. Для быстрого создания таблицы, в которую нужно поместить данные.

Достоинства:

* интуитивно понятный интерфейс;
* кроссплатформенность;
* импорт/экспорт из/в различные форматы;
* быстрая обработка таблиц с большим количеством данных;
* портативный: достаточно скачать, распаковать и запустить.



*Рисунок 7. Интерфейс SQLiteStudio.*

На панели слева отображаются все доступные базы данных вместе с их данными. Отображение имеет иерархическое древо с возможностью выбрать любую таблицу, процедуру или оператор и осмотреть данные в главном окне.

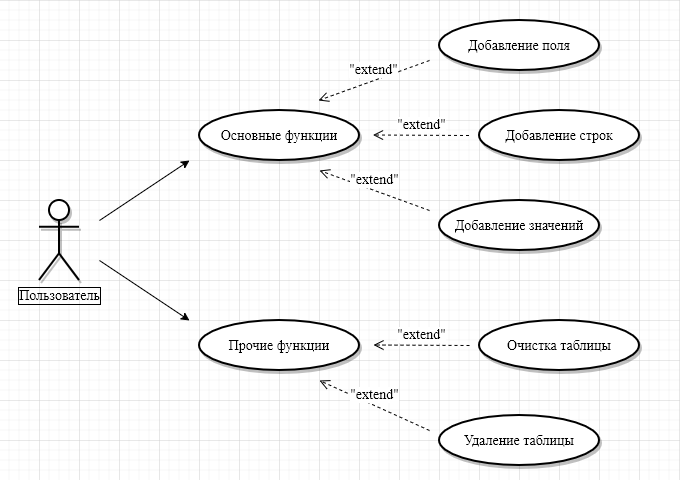
Через меню можно открыть файл БД разных форматов и подключиться к нему. Кроме того, можно импортировать схему из других баз данных или из файлов различных форматов, таких как CSV. После этого можно просматривать все существующие данные и манипулировать ими.

# Проектная часть

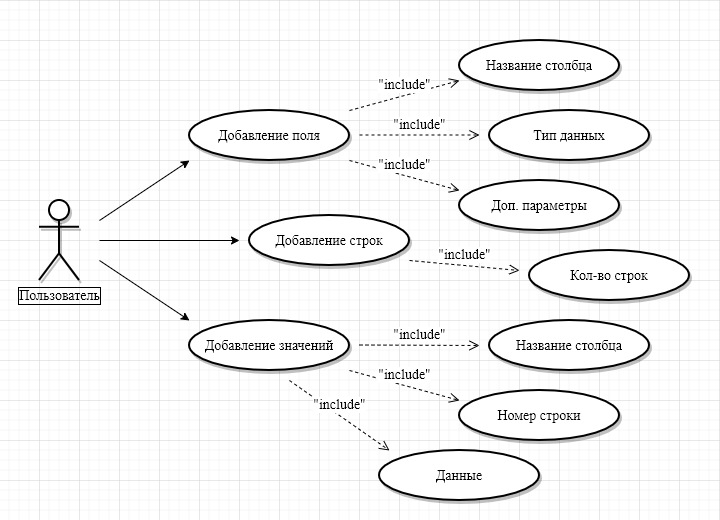
* 1. Построение диаграммы прецедентов

В этом разделе представлена диаграмма прецедентов. На диаграмме показаны все имеющиеся функции менеджера баз данных.

Функционал представлен в «3. Эксплуатационная часть» раздела «1. Назначение программы».



*Рисунок 8. Диаграмма прецедентов – функции.*



*Рисунок 9. Диаграмма прецедентов – работа функций.*

* 1. Выбор инструментов

При выборе инструментов было проведено сравнение по критериям, представленных в «Таблица 1».

Степень важности критерия выбиралась из: низкая, ниже средней, средняя, ниже высокой, высокая.

Таблица 1.

Критерии выбора инструмента.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Критерий** | Участие в корпоративном проекте | Простота сопровождения | Наличие библиотек | Наличие документации на русском языке | Скорость разработки |
| **Важность критерия** | Высокая | Средняя | Высокая | Низкая | Высокая |

Исходя из этих критериев, я сравнил 3 языка программирования от 0 до 10 баллов за критерий.

Таблица 2.

Оценка языков программирования.

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | **C++** | **Python** | **Kotlin** |
| Участие в корпоративном проекте | 10 | 8 | 7 |
| Простота сопровождения | 7 | 10 | 5 |
| Наличие библиотек | 6 | 10 | 8 |
| Наличие документации на русском языке | 8 | 6 | 5 |
| Скорость разработки | 6 | 10 | 5 |
| Итого баллов | 37 | 44 | 30 |

По результатам сравнения в «Таблица 2», был выбран язык программирования Python.

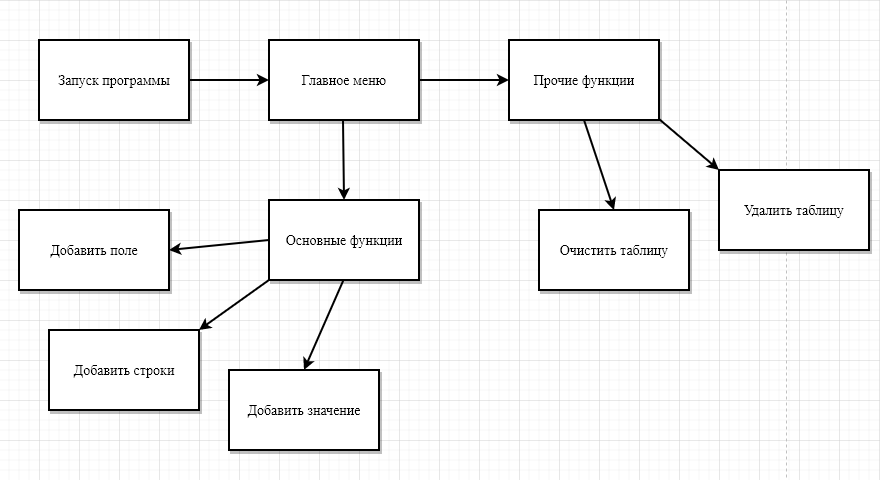
* 1. Проектирование сценария

В данном разделе приведен сценарий использования программы.

Пользователь после запуска может выполнить 2 действия: использовать основные функции и прочие.

При выборе основных функций, дается 3 варианта на выбор: добавить поле, добавить строки, добавить значение. Каждый вариант открывает новое окно, в котором заполняются данные.

При выборе прочих функций, дает 2 варианта на выбор: очистить таблицу и удалить таблицу. Соответственно таблицу или очищается, или удаляется.



*Рисунок 10. Сценарий использования.*

* 1. Построение диаграммы классов

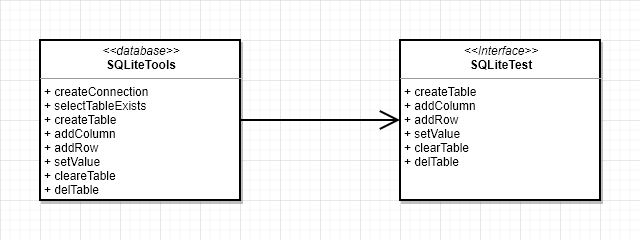
В данном разделе представлены все классы, использующиеся в проекте, а также их отношения между собой.

Класс «SQLiteTools» содержит следующие методы:

* createTable – создает соединение с базой данных
* selectTableExists – определяет, существует ли указанная таблица
* createTable – создает общую таблицу
* addColumn – добавляет столбцы указанной таблице
* addRow – добавляет строки указанной таблице
* setValue – обновляет значение указанной позиции в таблице
* clearTable – очищает указанную таблицу
* delTable – удаляет указанную таблицу

Класс «SQLiteTest» содержит следующие методы:

* createTable – создание таблицы
* addColumn – создание поля
* addRow – создание строк (их количество)
* setValue – заполнение записи
* clearTable – очищает таблицу
* delTable – удаляет таблицу



*Рисунок 11. Диаграмма классов проекта.*

* 1. Описание главного модуля

В главный модуль входит класс «SQLiteTest», который отвечает за функционал программы.

Также в главный модуль входит код, который запускает саму программу при ее запуске. Полный код главного модуля находится в «Приложение 1».

В классе «SQLiteTest» реализованы следующие методы:

**Листинг 1. Обработка всех событий в программе**

def \_\_init\_\_(self):

self.database\_file = "database.db"

self.table\_name = "Trade"

self.number\_of\_columns = 0

self.column\_list = []

if os.path.exists(self.database\_file):

os.remove(self.database\_file)

self.sqlite = SQLiteTools()

self.sqlite.createConnection(self.database\_file)

Данный метод создает имя базы данных и название таблицы, а также проверяет существующий файл, и если находит, то удаляет.

**Листинг 2. Создание таблицы**

def createTable(self):

""" Создание таблицы """

if not self.sqlite.selectTableExists(self.table\_name):

self.sqlite.createTable(self.table\_name)

Данный метод создает таблицу в том случае, если имя не совпадает с существующим, что проверяется методом «selectTableExists» и создается методом «createTable».

**Листинг 3. Создание поля**

def addColumn(self):

""" Создание поля """

def addColumnInner():

def closeButton():

a.destroy()

def clearFunc():

self.get\_column = name\_entry\_1.get()

self.column\_list.append(self.get\_column)

name\_entry\_1.delete(0, END)

name\_entry\_3.delete(0, END)

a = Toplevel()

a.geometry('250x200')

a.title("Создание поля")

a.resizable(False, False)

name\_1 = StringVar()

name\_3 = StringVar()

name\_label\_1 = Label(a, text="Столбец:")

name\_label\_1.place(x=10, y=10, width=110, height=30)

name\_label\_2 = Label(a, text="Тип данных:")

name\_label\_2.place(x=10, y=60, width=110, height=30)

name\_label\_3 = Label(a, text="Доп. параметры:")

name\_label\_3.place(x=10, y=110, width=110, height=30)

name\_entry\_1 = Entry(a, textvariable=name\_1)

name\_entry\_1.place(x=130, y=10, width=110, height=30)

combobox = ttk.Combobox(a, values=[u'integer',

u'real',

u'text',

u'blob'])

combobox.current(2)

combobox.place(x=130, y=60, width=110, height=30)

name\_entry\_3 = Entry(a, textvariable=name\_3)

name\_entry\_3.place(x=130, y=110, width=110, height=30)

message\_button = Button(a, text="Добавить", command=lambda:

(self.sqlite.addColumn(self.table\_name,

name\_1.get(),

combobox.get(),

name\_3.get()),

clearFunc()))

message\_button.place(x=10, y=160, width=100, height=30)

message\_button = Button(a, text="Закрыть окно",

command=closeButton)

message\_button.place(x=140, y=160, width=100, height=30)

name\_label = Label(root, text="Основные функции")

name\_label.place(x=50, y=10, width=100, height=30)

name\_label\_header\_button = Button(text="Добавить поле",

command=addColumnInner)

name\_label\_header\_button.place(x=10, y=40, width=110, height=30)

Данный метод создает кнопку «Добавить поле» при нажатии на которую открывается новое окно с выбором данных для добавления поля по кнопке «Добавить». Также есть кнопка закрытия окна «Закрыть окно».

**Листинг 4. Создание строк**

def addRow(self):

""" Создание строк(количества) """

def blockFunc():

name\_label\_header['state'] = 'disabled'

def addRowInner():

def blockFuncInner():

message\_button['state'] = 'disabled'

def closeFunc():

b.destroy()

def columnID():

self.number\_of\_columns = name\_entry.get()

b = Toplevel()

b.geometry('250x200')

b.title("Создание строк")

b.resizable(False, False)

name = StringVar()

name\_label = Label(b, text="Кол-во строк:")

name\_label.place(x=10, y=10, width=110, height=30)

name\_entry = Entry(b, textvariable=name)

name\_entry.place(x=130, y=10, width=110, height=30)

message\_button = Button(b, text="Добавить", command=lambda:

(columnID(), blockFuncInner(),

self.sqlite.addRow(self.table\_name,

int(name.get()))))

message\_button.place(x=10, y=160, width=100, height=30)

message\_button\_closure = Button(b, text="Закрыть окно",

command=closeFunc)

message\_button\_closure.place(x=140, y=160, width=100, height=30)

name\_label\_header = Button(text="Добавить строки", command=lambda:

(addRowInner(), blockFunc()))

name\_label\_header.place(x=130, y=40, width=110, height=30)

Данный метод создает кнопку «Добавить строки» при нажатии на которую открывается новое окно с выбором количества строк для добавления по кнопке «Добавить». Также есть кнопка закрытия окна «Закрыть окно».

**Листинг 5. Заполнение записи**

def setValue(self):

""" Заполнение записи """

def setValueInner():

def closeFunc():

c.destroy()

def cleareFunc():

name\_entry\_3.delete(0, END)

c = Toplevel()

c.geometry('250x200')

c.title("Ввод данных")

c.resizable(False, False)

name\_3 = StringVar()

name\_label\_1 = Label(c, text="Столбец:")

name\_label\_1.place(x=10, y=10, width=110, height=30)

name\_label\_2 = Label(c, text="ID:")

name\_label\_2.place(x=10, y=60, width=110, height=30)

name\_label\_3 = Label(c, text="Текст:")

name\_label\_3.place(x=10, y=120, width=110, height=30)

row\_list = [str(i) for i in range(1, int(self.number\_of\_columns)+1)]

combobox\_1 = ttk.Combobox(c, values=self.column\_list)

combobox\_1.current(0)

combobox\_1.place(x=130, y=10, width=110, height=30)

combobox\_2 = ttk.Combobox(c, values=row\_list)

combobox\_2.current(0)

combobox\_2.place(x=130, y=60, width=110, height=30)

name\_entry\_3 = Entry(c, textvariable=name\_3)

name\_entry\_3.place(x=130, y=110, width=110, height=30)

message\_button = Button(c, text="Добавить", command=lambda:

(self.sqlite.setValue(

self.table\_name,

combobox\_1.get(),

combobox\_2.get(),

name\_3.get()),

cleareFunc()))

message\_button.place(x=10, y=160, width=100, height=30)

message\_button = Button(c, text="Закрыть окно",

command=closeFunc)

message\_button.place(x=140, y=160, width=100, height=30)

name\_label\_header\_button = Button(text="Добавить значение",

command=setValueInner)

name\_label\_header\_button.place(x=250, y=40, width=110, height=30)

Данный метод создает кнопку «Добавить значение» при нажатии на которую открывается новое окно с выбором параметров для заполнения данных по кнопке «Добавить». Также есть кнопка закрытия окна «Закрыть окно». Становится доступной только после взаимодействия с прошлыми методами.

**Листинг 6. Очистка таблицы**

def clearTable(self):

""" Очистка таблицы """

name\_label = Label(root, text="Прочие функции")

name\_label.place(x=50, y=100, width=100, height=30)

message\_button = Button(text="Очистить таблицу", command=lambda:

self.sqlite.clearTable(self.table\_name))

message\_button.place(x=10, y=130, width=110, height=30)

Данный метод очищает данные из таблицы по кнопке «Очистить таблицу».

**Листинг 7. Удаление таблицы**

def delTable(self):

""" Удаление таблицы """

message\_button = Button(text="Удалить таблицу", command=lambda:

self.sqlite.delTable(self.table\_name))

message\_button.place(x=130, y=130, width=110, height=30)

Данный метод удаляет таблицу по кнопке «Удалить таблицу».

* 1. Описание спецификаций к модулям

В данном разделе описаны публичные члены модулей курсового проекта.

В главном модуле «main.py» содержатся публичные методы класса «SQLiteTest» которые взаимодействуют с методами класса «SQLiteTools» из модуля «database.py». Так как все эти методы были описаны в разделе 2.5, в данном разделе описание данных методов опущено.

В данном модуле также есть пара объектов:

1. root - объект, который создает системное окно и объект самого приложения.
2. win – объект, которые принимает экземпляр класса «SQLiteTest» для вызова функций.

В модуле, который реализует работу с базами данных «database.py» так же содержатся публичные методы класса «SQLiteTools», которые будут описаны ниже в разделе 2.7, в данном разделе они описаны не будут.

* 1. Описание модулей

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

Полный код модуля приведен в «Приложение 2». В этом разделе приведены все методы, которые содержаться в класс «SQLiteTools» из модуля «database.py».

**Листинг 8. Методы класса «SQLiteTools»**

def createConnection(self, database\_name):

""" Создать соединение с базой данных """

def selectTableExists(self, table\_name):

""" Определить, существует ли указанная таблица """

def createTable(self, table\_name):

""" Создать общую таблицу """

def addColumn(self, table\_name, column\_name, data\_type, other):

""" Добавить столбцы указанной таблице """

def addRow(self, table\_name, row\_num):

""" Добавить строки указанной таблице """

def setValue(self, table\_name, column, row, value):

""" Обновить значение указанной позиции в таблице """

def clearTable(self, table\_name):

""" Очистить указанную таблицу """

def delTable(self, table\_name):

""" Удалить указанную таблицу """

* 1. Описание тестовых наборов модулей

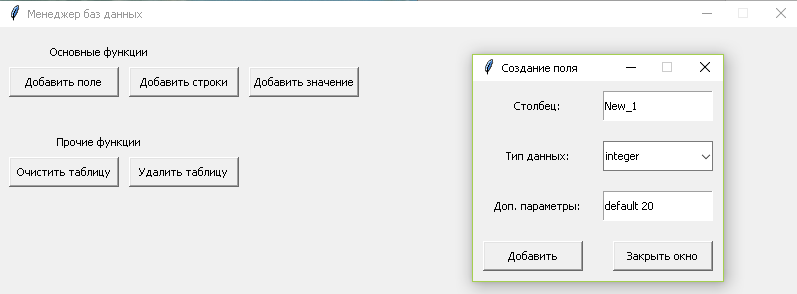
В этом разделе будут продемонстрированы результаты тестирования программы.

**Тест 1. Добавление поля**

Действия: Нажать на кнопку «Добавить поле» и ввести данные нового столбца, после чего нажать «Добавить», а после «Закрыть» или продолжить.

Ожидаемый результат: Создание нескольких новых полей в БД.

Результат теста:



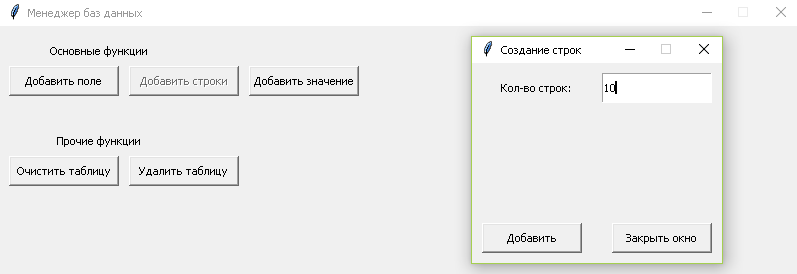
*Рисунок 12. Добавление поля*

**Тест 2. Добавление строк**

Действия: Нажать на кнопку «Добавить строки» и ввести их количество, после чего нажать «Добавить», а после «Закрыть». Повторно добавить строки не получится, т.к. кнопка «Добавить» блокируется после первого ввода, а после выхода из окна, блокирует и сама кнопка добавления строк.

Ожидаемый результат: Создание нескольких новых строк.

Результат теста:



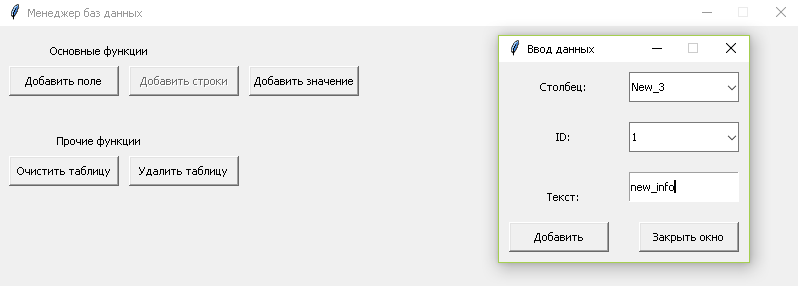
*Рисунок 13. Добавление строк.*

**Тест 3. Добавление значений**

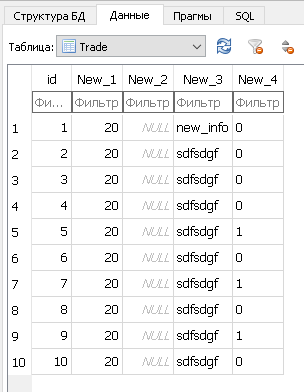
Действия: Нажать на кнопку «Добавить значение» и выбрать параметры ввода (в какой столбец и какую строку), после чего ввести данные и нажать «Добавить», а после «Закрыть» или продолжить.

Ожидаемый результат: Добавление новых значений в строки поля таблицы.

Результат теста:



*Рисунок 14. Добавление значений.*



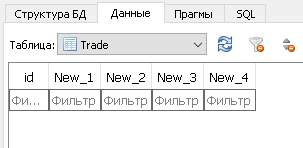
*Рисунок 15. Полученная таблица.*

**Тест 4. Очистка данных**

Действия: Нажать на кнопку «Очистить таблицу», после чего все строки с данными будут очищены из таблицы.

Ожидаемый результат: У таблицы пропадут все данные из строк.

Результат теста:



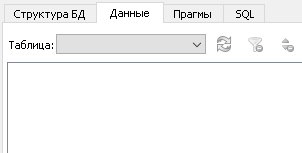
*Рисунок 16. Очистка данных.*

**Тест 5. Удаление данных**

Действия: Нажать на кнопку «Удалить таблицу», после чего таблица будет удалена.

Ожидаемый результат: Таблица будет удалена из базы данных.

Результат теста:

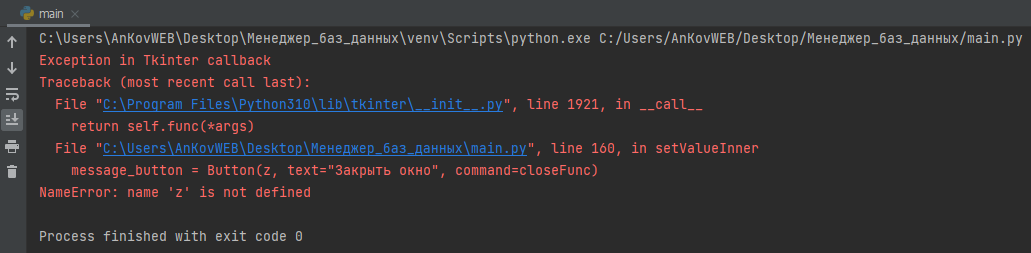


*Рисунок 17. Удаление данных.*

* 1. Описание применения средств отладки

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

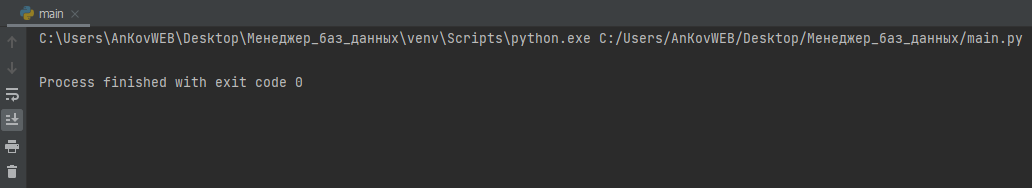
Для проверки была искусственно создана ошибка в коде. Вот что показал отладчик:



*Рисунок 18. Ошибка.* [4]

После получения данных из отладчика, была просмотрена 90 строка модуля «main.py» и выявлена ошибка в работе класса «SQLiteTest» метода «setValue».

Ошибка заключалась в том, что была введена неправильная переменная, что отвечает за создание кнопки в новом окне. После исправления ошибки было получено следующее:



*Рисунок 19. Исправлено.* [4]

* 1. [Анализ оптимальности использования памяти и быстродействия](#_Toc43724851)

В данном разделе будет проведен анализ оптимальности использования памяти и быстродействия программы.

Решение:

* Использование генератора списков.

Использование генераторов списков не только упрощает чтение кода, но и позволяет ускорить данную операцию почти в 2 раза. Например, в методе «setValue» класса «SQLiteTest», был использован генератор списков:

**Листинг 8. Генератор списка**

row\_list = [str(i) for i in range(1, int(self.number\_of\_columns)+1)]

Можно написать и так:

**Листинг 9. Цикл**

row\_list = []

for i in range(1, int(self.number\_of\_columns) + 1):

row\_list.append(str(i))

# Эксплуатационная часть

* 1. Руководство оператора

**АННОТАЦИЯ**

В данном программном документе приведено руководство оператора по применению и эксплуатации программы «Менеджер баз данных».

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 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**

Программа «Менеджер баз данных» используется для управления базой данных: добавление, удаление, редактирование и тд.

* 1. **Эксплуатационное назначение программы**

Программа «Менеджер баз данных» может эксплуатироваться с почти любым объемом данных.

* 1. **Состав функций**
     1. **Функция добавления поля.**

Данная функция позволяет добавлять новые поля в таблицу и настроить их параметры.

* + 1. **Функция добавления строк.**

Данная функция позволяет единожды добавить указанное количество строк в таблицу.

* + 1. **Функция добавления значения.**

Данная функция позволяет добавить данные в указанные столбец и строку в таблице.

* + 1. **Функция очистки таблицы.**

Данная функция очищает все строки таблицы.

* + 1. **Функция удаления таблицы.**

Данная функция полностью удаляет таблицу.

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

ОС: Windows 7/8/8.1/10/11

Процессор: от 1 ГГц.

ОЗУ: от 1 ГБ .

Место на жестком диске: от 1 мб.

Видеоадаптер: DirectX версии не ниже 9.

Дисплей: от 800 x 600.

* 1. **Минимальный состав программных средств**

Дополнительные программные средства не требуются.

* 1. **Требование к пользователю**

Пользователь программы должен обладать практическими навыками работы с графическим интерфейсом операционной системы.

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

Запустите программу «Менеджер баз данных», после чего откроется окно главного меню:

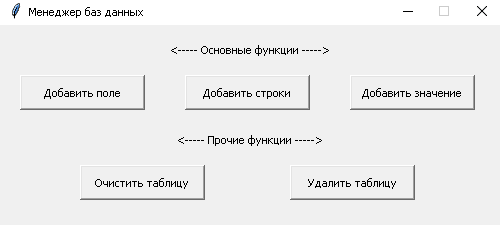
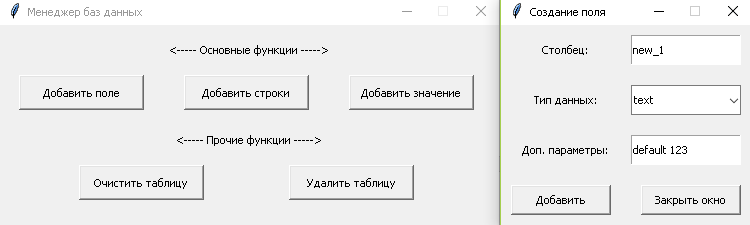


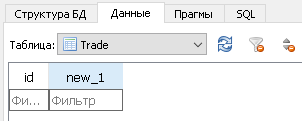
Рисунок 20. Главное меню.

Кнопки меню имеют следующие функции (описаны в 1.3 Состав функций):

* Добавить поле
* Добавить строки
* Добавить значение
* Очистить таблицу
* Удалить таблицу
  1. **Выполнение программы**
     1. **Выполнение функции добавления поля.**

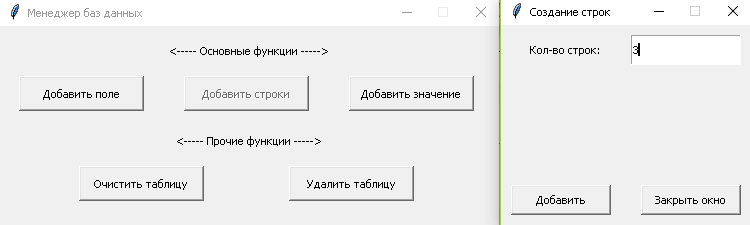


*Рисунок 21. Окно для работы с добавлением поля.*

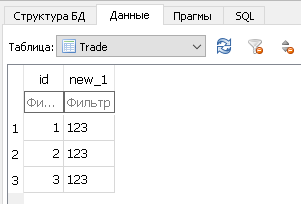


*Рисунок 22. Просмотр поля в программе.*

* + 1. **Выполнение функции добавления строк.**

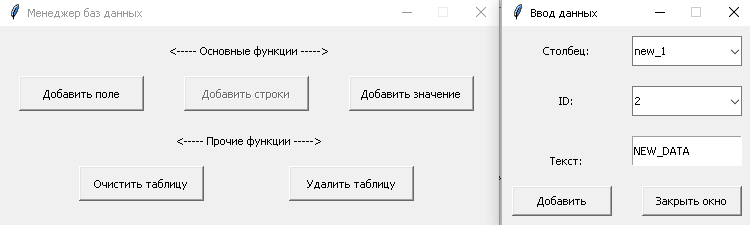


*Рисунок 23. Окно для работы с добавлением строк.*

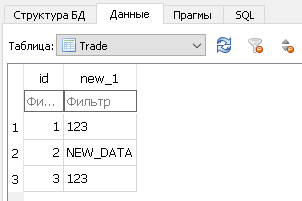


*Рисунок 24. Просмотр строк в программе.*

* + 1. **Выполнение функции добавления значений.**

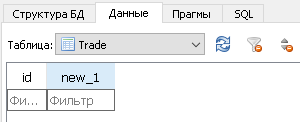


*Рисунок 25. Окно для работы с добавлением значений.*



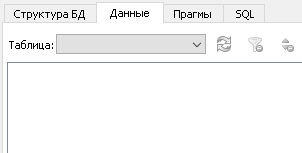
*Рисунок 26. Просмотр добавленных значений.*

* + 1. **Выполнение функции очистки таблицы.**



*Рисунок 27. Просмотр использования очистки таблицы.*

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



*Рисунок 28. Просмотр использования удаления таблицы.*

* 1. **Завершение работы программы**

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

* 1. 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. Отображение таблицы в интерфейсе.
4. Импорт/экспорт баз данных.
5. Добавление возможности ввода запросов для сортировки таблицы.

# Заключение

В результате выполнения курсового проекта была написана программа «Менеджер баз данных» для создания баз данных с тонкой настройкой параметров таблиц, столбцов и строк.

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.

На данном этапе данная программа хорошо подходит для создания шаблона таблицы баз данных.

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 использованной литературы и интернет-ресурсов

1. Пример функционала менеджера баз данных.

<https://shternt.wordpress.com/2017/06/07/220/amp/>

1. Менеджер баз данных – DB Browser for SQLite

<https://sqlitebrowser.org/>

1. Менеджер баз данных – SQLiteStudio

<https://sqlitestudio.pl/>

1. Среда разработки – PyCharm

<https://www.jetbrains.com/ru-ru/pycharm/>

1. Мой проект на GitHub.

<https://github.com/AnKovWEB/SQLiteProject.git>

# Приложения

**Приложение 1. «main.py»**

import tkinter as tk

from database.tools import SQLiteTools

import tkinter.ttk as ttk

from tkinter import \*

import os

class SQLiteTest:

def \_\_init\_\_(self):

self.database\_file = "database.db"

self.table\_name = "Trade"

self.number\_of\_columns = 0

self.column\_list = []

if os.path.exists(self.database\_file):

os.remove(self.database\_file)

self.sqlite = SQLiteTools()

self.sqlite.createConnection(self.database\_file)

def createTable(self):

""" Создание таблицы """

if not self.sqlite.selectTableExists(self.table\_name):

self.sqlite.createTable(self.table\_name)

def addColumn(self):

""" Создание поля """

def addColumnInner():

def closeButton():

a.destroy()

def clearFunc():

self.get\_column = name\_entry\_1.get()

self.column\_list.append(self.get\_column)

name\_entry\_1.delete(0, END)

name\_entry\_3.delete(0, END)

a = Toplevel()

a.geometry('250x200')

a.title("Создание поля")

a.resizable(False, False)

name\_1 = StringVar()

name\_3 = StringVar()

name\_label\_1 = Label(a, text="Столбец:")

name\_label\_1.place(x=10, y=10, width=110, height=30)

name\_label\_2 = Label(a, text="Тип данных:")

name\_label\_2.place(x=10, y=60, width=110, height=30)

name\_label\_3 = Label(a, text="Доп. параметры:")

name\_label\_3.place(x=10, y=110, width=110, height=30)

name\_entry\_1 = Entry(a, textvariable=name\_1)

name\_entry\_1.place(x=130, y=10, width=110, height=30)

combobox = ttk.Combobox(a, values=[u'integer',

u'real',

u'text',

u'blob'])

combobox.current(2)

combobox.place(x=130, y=60, width=110, height=30)

name\_entry\_3 = Entry(a, textvariable=name\_3)

name\_entry\_3.place(x=130, y=110, width=110, height=30)

message\_button = Button(a, text="Добавить", command=lambda:

(self.sqlite.addColumn(self.table\_name,

name\_1.get(),

combobox.get(),

name\_3.get()),

clearFunc()))

message\_button.place(x=10, y=160, width=100, height=30)

message\_button = Button(a, text="Закрыть окно",

command=closeButton)

message\_button.place(x=140, y=160, width=100, height=30)

name\_label = Label(root, text="Основные функции")

name\_label.place(x=50, y=10, width=100, height=30)

name\_label\_header\_button = Button(text="Добавить поле",

command=addColumnInner)

name\_label\_header\_button.place(x=10, y=40, width=110, height=30)

def addRow(self):

""" Создание строк(количества) """

def blockFunc():

name\_label\_header['state'] = 'disabled'

def addRowInner():

def blockFuncInner():

message\_button['state'] = 'disabled'

def closeFunc():

b.destroy()

def columnID():

self.number\_of\_columns = name\_entry.get()

b = Toplevel()

b.geometry('250x200')

b.title("Создание строк")

b.resizable(False, False)

name = StringVar()

name\_label = Label(b, text="Кол-во строк:")

name\_label.place(x=10, y=10, width=110, height=30)

name\_entry = Entry(b, textvariable=name)

name\_entry.place(x=130, y=10, width=110, height=30)

message\_button = Button(b, text="Добавить", command=lambda:

(columnID(), blockFuncInner(),

self.sqlite.addRow(self.table\_name,

int(name.get()))))

message\_button.place(x=10, y=160, width=100, height=30)

message\_button\_closure = Button(b, text="Закрыть окно",

command=closeFunc)

message\_button\_closure.place(x=140, y=160, width=100, height=30)

name\_label\_header = Button(text="Добавить строки", command=lambda:

(addRowInner(), blockFunc()))

name\_label\_header.place(x=130, y=40, width=110, height=30)

def setValue(self):

""" Заполнение записи """

def setValueInner():

def closeFunc():

c.destroy()

def cleareFunc():

name\_entry\_3.delete(0, END)

c = Toplevel()

c.geometry('250x200')

c.title("Ввод данных")

c.resizable(False, False)

name\_3 = StringVar()

name\_label\_1 = Label(c, text="Столбец:")

name\_label\_1.place(x=10, y=10, width=110, height=30)

name\_label\_2 = Label(c, text="ID:")

name\_label\_2.place(x=10, y=60, width=110, height=30)

name\_label\_3 = Label(c, text="Текст:")

name\_label\_3.place(x=10, y=120, width=110, height=30)

row\_list = [str(i) for i in range(1, int(self.number\_of\_columns)+1)]

combobox\_1 = ttk.Combobox(c, values=self.column\_list)

combobox\_1.current(0)

combobox\_1.place(x=130, y=10, width=110, height=30)

combobox\_2 = ttk.Combobox(c, values=row\_list)

combobox\_2.current(0)

combobox\_2.place(x=130, y=60, width=110, height=30)

name\_entry\_3 = Entry(c, textvariable=name\_3)

name\_entry\_3.place(x=130, y=110, width=110, height=30)

message\_button = Button(c, text="Добавить", command=lambda:

(self.sqlite.setValue(

self.table\_name,

combobox\_1.get(),

combobox\_2.get(),

name\_3.get()),

cleareFunc()))

message\_button.place(x=10, y=160, width=100, height=30)

message\_button = Button(c, text="Закрыть окно",

command=closeFunc)

message\_button.place(x=140, y=160, width=100, height=30)

name\_label\_header\_button = Button(text="Добавить значение",

command=setValueInner)

name\_label\_header\_button.place(x=250, y=40, width=110, height=30)

def clearTable(self):

""" Очистка таблицы """

name\_label = Label(root, text="Прочие функции")

name\_label.place(x=50, y=100, width=100, height=30)

message\_button = Button(text="Очистить таблицу", command=lambda:

self.sqlite.clearTable(self.table\_name))

message\_button.place(x=10, y=130, width=110, height=30)

def delTable(self):

""" Удаление таблицы """

message\_button = Button(text="Удалить таблицу", command=lambda:

self.sqlite.delTable(self.table\_name))

message\_button.place(x=130, y=130, width=110, height=30)

if \_\_name\_\_ == '\_\_main\_\_':

root = tk.Tk()

root.geometry('800x500')

root.title("Менеджер баз данных")

root.resizable(False, False)

win = SQLiteTest() # Соединение

win.createTable() # Создать таблицу

win.addColumn() # Создание поля

win.addRow() # Создание строк

win.setValue() # Заполнение данными

win.clearTable() # Очистка таблицы

win.delTable() # Удаление таблицы

root.mainloop()

**Приложение 2. «database.py»**

import sqlite3

class SQLiteTools:

def \_\_init\_\_(self):

pass

def createConnection(self, database\_name):

""" Создать соединение с базой данных """

self.db = sqlite3.connect(database\_name)

self.cursor = self.db.cursor()

def selectTableExists(self, table\_name):

""" Определить, существует ли указанная таблица """

sql = "SELECT \* FROM sqlite\_master WHERE type = 'table' AND name = '" + table\_name + "';"

self.cursor.execute(sql)

cur = self.cursor.fetchall()

return True if cur else False

def createTable(self, table\_name):

""" Создать общую таблицу """

sql = "CREATE TABLE IF NOT EXISTS " + table\_name + " (id INTEGER PRIMARY KEY AUTOINCREMENT);"

self.cursor.execute(sql)

self.db.commit()

def addColumn(self, table\_name, column\_name, data\_type, other):

""" Добавить столбцы указанной таблице """

sql = "ALTER TABLE " + table\_name + " ADD " + column\_name + " " + data\_type + " " + other + ";"

self.cursor.execute(sql)

self.db.commit()

def addRow(self, table\_name, row\_num):

""" Добавить строки указанной таблице """

number = [(i + 1,) for i in range(row\_num)]

sql = "INSERT INTO " + table\_name + " (id) VALUES (?);"

self.cursor.executemany(sql, number)

self.db.commit()

def setValue(self, table\_name, column, row, value):

""" Обновить значение указанной позиции в таблице """

sql = "UPDATE " + table\_name + " SET " + column + " = '" + value + "' WHERE id = " + str(row) + ";"

self.cursor.execute(sql)

self.db.commit()

def clearTable(self, table\_name):

""" Очистить указанную таблицу """

sql = "DELETE FROM " + table\_name + ";"

self.cursor.execute(sql)

self.db.commit()

def delTable(self, table\_name):

""" Удалить указанную таблицу """

sql = "DROP TABLE " + table\_name + ";"

self.cursor.execute(sql)

self.db.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)