МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет информационных технологий и компьютерной безопасности

(факультет)

Кафедра компьютерных интеллектуальных технологий проектирования

КУРСОВОЙ ПРОЕКТ

по дисциплине «Основы программирования на Python»

Тема «Разработка интерфейса для работы с БД (sqlite)»

Расчетно-пояснительная записка

Разработал студент М. Е. Федоров

гр. бПО-221 Подпись, дата Инициалы, фамилия

Руководитель А. А. Филимонова Подпись, дата Инициалы, фамилия

Нормоконтролер

А. А. Филимонова Подпись, дата Инициалы, фамилия

Защищена Оценка

дата

Воронеж 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ФГБОУ ВО «ВГТУ», ВГТУ)

Кафедра компьютерных интеллектуальных технологий проектирования

Задание на курсовой проект

по дисциплине «Основы программирования на Python»

Тема работы «Разработка интерфейса для работы с БД (sqlite)»

Вариант работы 14

Студент группы бПО-221, Федоров Михаил Евгеньевич

Фамилия, имя, отчество

Технические условия: PyCharm 2024, Qt Designer, Microsoft Word 2021

Содержание и объем работы (графические работы, расчеты и прочее): 27 страниц, 26 рисунков, 0 таблиц, 0 приложений

Сроки выполнения этапов Рассмотрение теоретических сведений (сентябрь-декабрь 2024); оформление пояснительной записки (декабрь 2024);

Срок защиты курсового проекта сентябрь-декабрь 2024

Руководитель А. А. Филимонова

Подпись, дата Инициалы, фамилия

Задание принял студент М. Е. Федоров

Подпись, дата Инициалы, фамилия

# Замечания руководителя

Оглавление

[Задание на курсовой проект 2](#_Toc184810165)

[Замечания руководителя 3](#_Toc184810166)

[Введение 5](#_Toc184810167)

[1 Постановка задачи 6](#_Toc184810168)

[1.1 Анализ 6](#_Toc184810169)

[1.2 Требования к системе 6](#_Toc184810170)

[2 Проектирование интерфейса для работы с sqlite 8](#_Toc184810171)

[2.1 Разработка интерфейса 8](#_Toc184810172)

[2.2 Основной функционал 10](#_Toc184810173)

[3 Разработка 11](#_Toc184810174)

[3.1 Архитектура решения 11](#_Toc184810175)

[3.2 Диаграмма вариантов использования 12](#_Toc184810176)

[3.3 Структура проекта 14](#_Toc184810177)

[3.4 Особенности программного кода 15](#_Toc184810178)

[4 Тестирование 18](#_Toc184810179)

[4.1 Тестирование программного кода Python. 18](#_Toc184810180)

[4.2 Руководство пользователя 23](#_Toc184810181)

[Список литературы 27](#_Toc184810182)

# Введение

Python является одним из самых популярных языков программирования благодаря простоте синтаксиса, широкому выбору библиотек, и универсальности. К его преимуществам относят:

* кроссплатформенность. Программы на Python работают на Windows, Linux, и macOS без необходимости значительных изменений кода;
* обширную стандартную библиотеку. Python предоставляет готовые инструменты для работы с файлами, сетями, базами данных, и многими другими задачами;
* поддержку сообщества. Огромное количество готовых библиотек, таких как sqlite3 для работы с базами данных и pyside6 для создания пользовательских интерфейсов, сокращают время разработки.

Для реализации проекта выбраны следующие модули:

* sqlite3 – для управления базой данных, модуль поставляется совместно с Python;
* pyside6 – для создания GUI. Этот модуль не предусмотрен стандартной библиотекой Python и его необходимо устанавливать отдельно;
* os – встроенный модуль для работы с файловой системой.

# Постановка задачи

## Анализ

В ходе курсовой работы разрабатывается настольное приложение для управления базой данных sqlite.

Аналоги, такие как Microsoft Access или LibreOffice Base, предоставляют мощные средства управления базами данных и для работы с базой данных sqlite требуют установки дополнительных модулей. Данные среды разработки позволяют создавать и редактировать таблицы, а также визуализируют данные при помощи создания диаграмм, графиков и отчетов на основе данных.

## Требования к системе

Ввиду небольшого срока выполнения курсовой работы были выдвинуты следующие ограничения к системе:

* удобный и минималистичный пользовательский интерфейс;
* функциональность: управление базами данных и таблицами, а также данными, хранящимися в них;
* простота установки и использования без необходимости использования дополнительного ПО;
* работа с SQLite в качестве базы данных.

Глобальные требования к архитектуре:

* модульность приложения (выделение определенного функционала приложения в отдельные классы).
* минимальное использование внешних зависимостей.

Курсовой проект выполняется в процессе изучения дисциплины «Основы программирования Python» и выполняется поэтапно.

Этапы курсового проекта:

* получение задания – 1-2 недели;
* постановка задачи, анализ предметной области, поиск аналогов – 4 неделя;
* концептуальное и логическое моделирование структуры приложения – 7 неделя;
* выбор подключаемых модулей для расчетов, представления данных,
* отображения графиков, построения графического интерфейса– 9 неделя;
* алгоритмическое решение базового функционала – 11 неделя;
* проектирование приложения – 13 неделя;
* выбор среды разработки, реализация приложения – 15 неделя;
* верификация и отладка приложения – 16 неделя;
* оформление РПЗ и предоставление на проверку, защита курсового проекта – 17, 18 неделя;

# Проектирование интерфейса для работы с sqlite

## Разработка интерфейса

Разработка интерфейса основана на требованиях к функционалу desktop приложения. Необходимо разработать графические окна, предоставляющие пользователю управление с базой данных: от её создания и удаления, до внесения записей в таблицу.

Первым этапом проектирования интерфейса является создание главного окна, в котором должны вместиться рычаги управления следующего характера:

* выбор рабочей директории (каталог с базами данных пользователя);
* визуализация баз данных и таблиц;
* поле для представления данных, содержащихся в отдельной таблице;
* поле с возможностью создания SQL-запросов;
* поле с результатами выполнения SQL-запросов.

Решение первого этапа проектирования можно привести в виде рисунков, на которых изображены начальное представление (Рисунок 1) и реализация в qt designer (Рисунок 2).

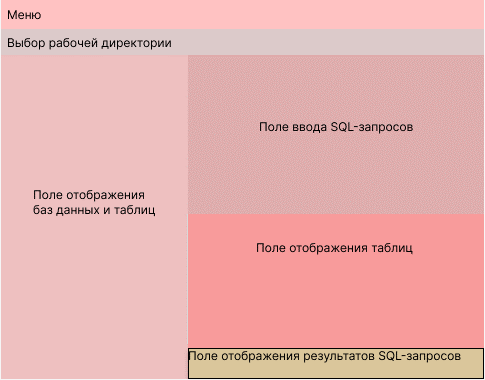


Рисунок 1 – Макет главного окна приложения.

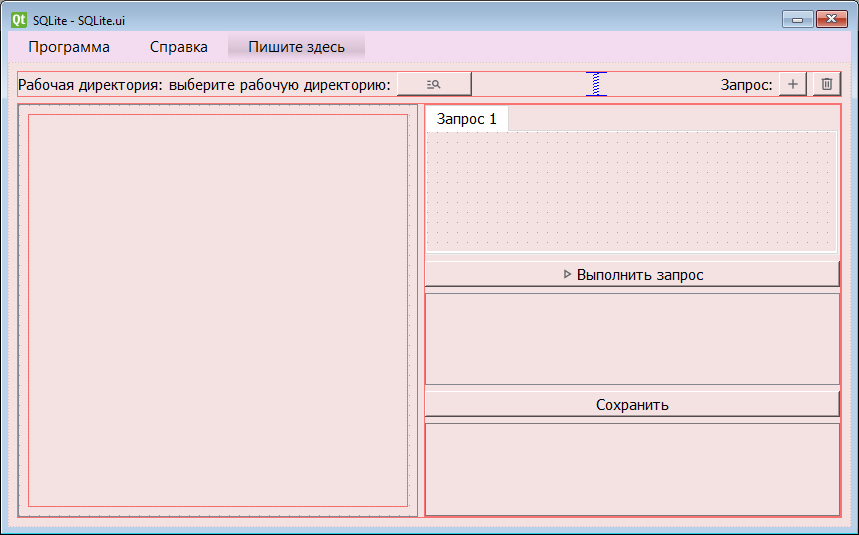


Рисунок 2 – Реализация главного окна приложения.

Следующим этапом необходимо разработать меню, позволяющее создавать базы данных. Требования к меню минимальны, необходимо информировать пользователя о том, где создается база данных и указать поле ввода базы данных макет – Рисунок 3, реализация – Рисунок 4.

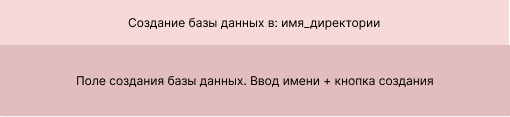


Рисунок 3 – Макет полей создания базы данных.

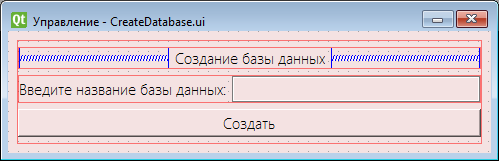


Рисунок 4 – Реализация макета создания БД.

В приложении также необходимы окна, позволяющие пользователю создавать и редактировать таблицу базы данных. Реализация меню одинакова, поскольку значения некоторых ограничений аттрибутов нужно лишь поменять ­– Рисунок 5.

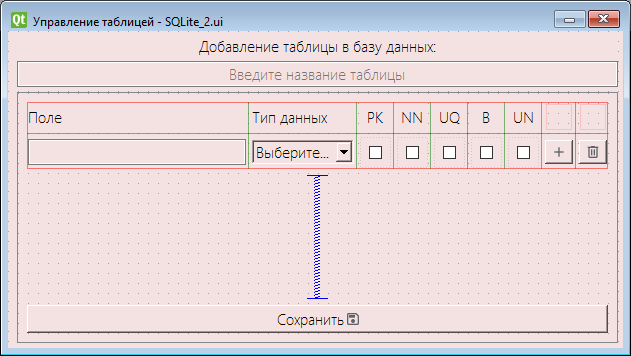


Рисунок 5 – Окно редактирования таблицы БД.

## Основной функционал

К функционалу разрабатываемого приложения можно отнести следующие пункты:

* создание базы данных;
* удаление базы данных;
* создание таблицы базы данных;
* изменение таблицы базы данных;
* удаление таблицы базы данных;
* отображение данных таблицы базы данных;
* изменение значений в таблице базы данных;
* выполнение рукописных SQL-запросов;
* информирование о результатах SQL-запросов;
* визуальное представление данных SQL-запросов;
* интерактивный помощник написания SQL-запросов.

# Разработка

## Архитектура решения

Для реализации программы, предоставляющей пользователю удобный интерфейс работы с базой данных sqlite, необходимо определить её архитектуру. Из постановления задач следует, что приложению необходимо вести работу с директориями операционной системы пользователя, обращаться к файлу базы данных sqlite и, самое главное, всё это должно происходить в пользовательском интерфейсе.

Для работы с каталогами операционной системы пользователя был выбран встроенный модуль языка программирования Python – «os», который предоставляет весь необходимый функционал. Данный модуль прост в понимании и при его помощи можно легко сделать такие операции, как проверка существования пути; является данный путь файлом, директорией; получение абсолютного пути; приведение строки к нормальному виду пути в каталоге; объединение строк и приведение их к нормальному виду пути и многое другое.

Для работы с базой данных sqlite был выбран базовый модуль «sqlite3», который предоставляет все базовые возможности для работы с базой данных: установление подключения к базе данных, выполнение запросов, проведение транзакций.

Разработку программного интерфейса было решено проводить при помощи модуля «pyside6», который предоставляет гибкую работу с элементами. Было проведено сравнение с такими GUI как «tkinter» и «kivy», к минусам первого можно отнести неподходящую систему размещения элементов на рабочем поле (нельзя указать точное местоположение, а «kivy», в свою очередь, имеет недостаточно документации для таких гибких задач.

Выбор модуля «pyside6» основан на возможности быстрой стилизации элементов, создания гибкого интерфейса. Кроме того, из программы легко обращаться ко множеству свойств определенного элемента: от указания иконки элементу до обращения к сигналам, отправляемых элементом в процессе выполнения программного кода. Кроме того, следует отметить возможность создания графического интерфейса прямиком из пользовательского интерфейса «Qt Designer», в котором при помощи всего лишь мышки возможно составить стилизованный согласно вашим условиям проект.

Для сборки всего приложения в исполняемый файл используется модуль «pyinstaller», позволяющий преобразовать весь программный код в исполняемый файл всего лишь при помощи одной команды: «pyinstaller –onefile –noconsole main.py». Данная команда автоматически собирает все зависимости и генерирует итоговый файл.

## Диаграмма вариантов использования

Диаграмма вариантов использования представляет собой визуальное представление взаимодействия между акторами и системой, а также описывает функциональные возможности, которые система предоставляет пользователям. В данном случае, акторы включают обычных пользователей, студентов и преподавателей, которые взаимодействуют с системой для выполнения различных задач.

К вариантам использования следует отнести:

* просмотр каталогов;
* выполнение операций с файлами;
* работа с базой данных.

К прецедентам относится пользователь, система базы данных, система отчетности, окно приложения.

К действиям пользователя можно отнести его работу с окном приложения, которое предоставляет пользовательский интерфейс и всю отчетность работы с базой данных sqlite, к этим действиям можно отнести: выбор рабочей директории, управление базами данных, управление таблицами, управление данными в таблице, создание запросов.

На действия пользователя реагирует окно приложения, которое совместно с системой базы данных проводит все значимые операции: подключение к базе данных, обработка запросов и получение отчетности о выполнении операций. Система отчетности, в свою очередь, обрабатывает ошибки, возникающие в ходе работы с базой данных sqlite и передает итоговый результат операции в окно приложения через систему базы данных. Диаграмму вариантов использования можно представить в виде рисунка – Рисунок 6.

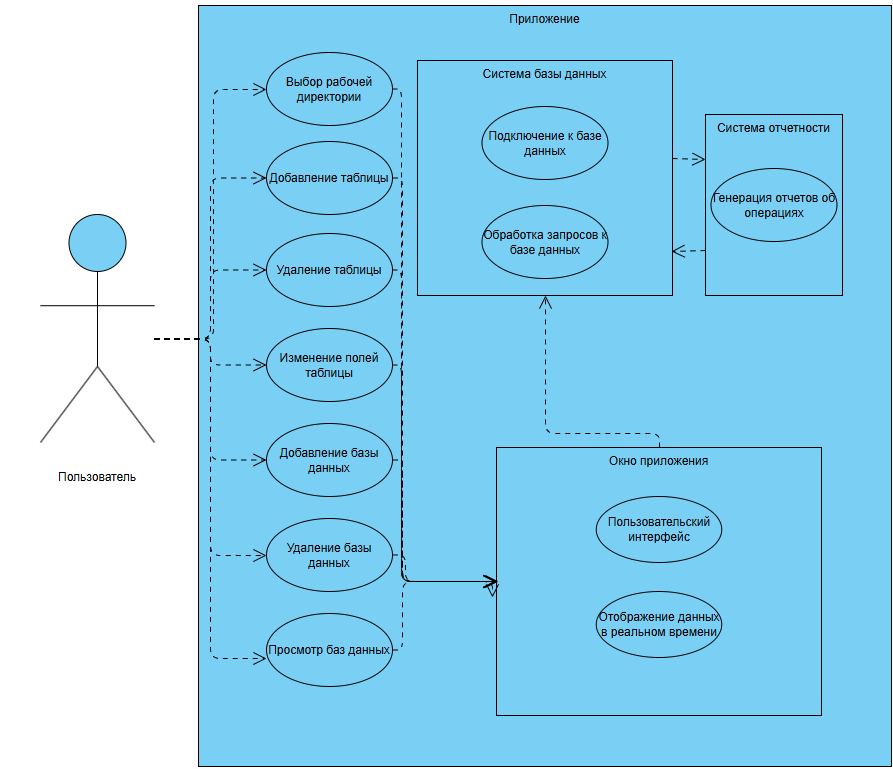


Рисунок 6 – Диаграмма вариантов использования.

## Структура проекта

Структура проекта представлена на рисунке 7.

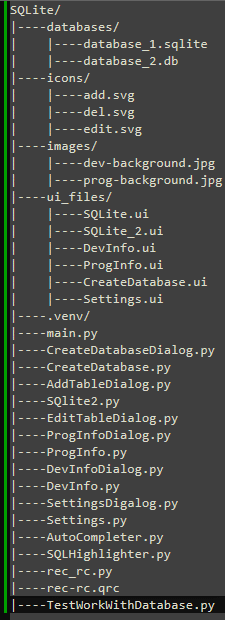


Рисунок 7 – Структура проекта.

Каталог databases предназначен для хранения файлов баз данных sqlite, icons- каталог иконок, требуемый для pyside6, images хранит задний фон для экранов отображения информации о программе и разработчике, ui\_files каталог предназначен для хранение файлов с расширением «ui» – созданных в среде «qt widgets». Каталог «.venv» предназначен для изоляции проектов от внешней среды для избежания конфликтов библиотек. Файлы, относящиеся к корню директории SQLite являются классами, сгенерированными при помощи конструктора pyside6 для отображения диалоговых окон, а также там хранятся классы для работы с этими диалоговыми окнами. Например, main.py подключает главное диалоговое окно SQLite.py и отвечает за работу со всеми элементами, хранящимся в этом классе. Аналогично устроена работа классов для управления базами данными, таблицами.

## Особенности программного кода

В процессе написания программного кода было обнаружено множество важных деталей, связанных с работой элементов в «qt\_designer». Одна из них очень важная – отображение элементов в определенном блоке. Если мы хотим отобразить список баз данных в определенном месте, то необходимо позаботиться об очистке устаревших данных. Если мы начнем отображать данные в необходимом для нас месте, то произойдет наложение элементов друг на друга, что ведет к путанице. Был разработан метод очистки полотна, который принимает на вход элемент QLayout и находит в нем все вложенные элементы. Если в QLayout вложен еще один слой, то метод становится рекурсивным – Листинг 1.

Листинг 1 – Очистка QLayout.

def clear\_layout(self, layout):

"""Рекурсивно очищает указанный QLayout от всех элементов."""

while layout.count():

item = layout.takeAt(0)

if widget := item.widget():

widget.deleteLater() # Удаляем виджет

elif sub\_layout := item.layout():

self.clear\_layout(sub\_layout) # Рекурсивно очищаем вложенные макеты

del item # Удаляем сам объект item из памяти

Благодаря данному методу стало возможным корректное отображения списка баз данных. При начале отображения новых элементов вызывается метод очистки, а затем отрисовка виджетов.

Следующая деталь связана с отображением определенного набора элементов в необходимом для нас месте. Для реализации функционала динамической вставки виджетов Qt\_Designer предлагает нам вставку определенного виджета по индексу. Нам лишь необходимо обратиться к объекту по его имени и понять, какой он по счету в определенном слое. Затем, прибавив к индексу единицу можно вставить элемент вслед за ним, предварительно очистив предыдущий вставляемый слой QLabel. Пример динамического добавления виджета – Листинг 2.

Листинг 2 – Динамическая вставка виджета.

def display\_tables(self, db\_name, tables, button):

# Ищем уже существующий блок для базы данных

index = -1

for i in range(self.ui.DatabasesVBoxLayoutForWidget.count()):

item = self.ui.DatabasesVBoxLayoutForWidget.itemAt(i)

if isinstance(item, QLayout) and item.objectName() == f"DB\_HS\_{db\_name}":

index = i

if index != -1: # Если блок найден

# Очищаем старые элементы с таблицами

self.clear\_finded\_table\_layout(db\_name)

# Если таблиц нет, добавляем метку "Нет таблиц"

if not tables:

table\_hs\_all = QHBoxLayout()

table\_hs\_all.setObjectName(f"Table\_{db\_name}\_HS\_ALL")

table\_hs\_all.setContentsMargins(0, 0, 0, 5)

spacer\_item = QSpacerItem(10, 20, QSizePolicy.Policy.Minimum, QSizePolicy.Policy.Minimum)

table\_hs\_all.addItem(spacer\_item)

no\_tables\_label = QLabel("Нет таблиц")

no\_tables\_label.setObjectName(f"NoTablesLabel\_{db\_name}")

no\_tables\_label.setStyleSheet("color: gray; font-style: italic;")

table\_hs\_all.addWidget(no\_tables\_label)

self.ui.DatabasesVBoxLayoutForWidget.insertLayout(index + 1, table\_hs\_all)

Работа с таблицей SQLite, а в частности изменение её данных – трудоемкая задача. Необходимо понимать, что при изменении данных в строке необходимо каким-то образом отразить проделанные изменения в базу данных. Но как это сделать, если все данные строки изменены? Нет критерия, по которому можно обратиться к необходимой строке в базе данных. На помощь приходит очевидное решение – ввод автоинкрементного поля «id», которое обязательно ввиду создания, изменения таблицы базы данных при помощи разрабатываемого графического интерфейса работы с базой данных sqlite. При создании таблицы мы будем автогенерировать поле id с ограничениями primary key и autoincrement соответственно, при чем вся работа, связанная с изменениями таблицами будет завязана на id. Минус подхода заключается в том, что нет возможности создания поля с своим аттрибутом, имеющим ограничение primary key. Для сложной организации БД такое решение является неприемлемым и неработоспособным, поскольку дальнейшая работа полностью зависит от ручного управления программиста: отображение таблицы, изменение и добавление данных производится путем написания sql запросов, никак не при помощи нажатия на кнопку.

К особенностям программного кода можно отнести работу с классами QCompleter, библиотеки PySide6.QtWidgets. Расширен базовый класс QTextEdit в AutoCompleteTextEdit. При создании экземпляра класса автоматически создается объект класса QCompleter, который отслеживает изменения в текстовом поле. При совпадении нескольких букв (от двух) текстовый редактор предлагает заменить данные буквы на подходящее служебное слово, определенное в некотором массиве класса. Кроме того, служебные слова имеют определенную подсветку: ключевые слова SQL помечаются синим цветом, функции багровым, тип данных зеленым.

# Тестирование

## Тестирование программного кода Python.

Тестирование приложений — это важный этап разработки программного обеспечения, который обеспечивает высокое качество и надежность конечного продукта. В контексте нашего приложения тестирование проводилось над такими операциями, как чтение, добавление и удаление базы данных, а также работа с данными в таблице. Опустим инициализацию класса TestWorkWithDatabase, в которой создается временный каталог с базами данных, и покажем тест на корректное обновление списка баз данных – Листинг 3.

Листинг 3 – Обновление списка баз данных.

def test\_refresh\_databases(self):

"""

Тестирует метод refresh\_databases на корректное обновление списка баз данных.

"""

# Вызываем метод

self.main\_window.refresh\_databases()

# Ожидаемый список баз данных (файлы с расширениями .db и .sqlite)

expected\_databases = [file\_name for file\_name in self.db\_files if file\_name.lower().endswith(('.db', '.sqlite'))]

expected\_count = len(expected\_databases)

# Получаем фактический список баз данных из интерфейса

layout = self.main\_window.ui.DatabasesVBoxLayoutForWidget

actual\_databases = []

# Проходим по всем элементам в layout, начиная с индекса 1, так как первый элемент — кнопка создания БД

for i in range(1, layout.count()):

item = layout.itemAt(i)

if isinstance(item, QHBoxLayout):

label = item.itemAt(0).widget()

if isinstance(label, QLabel):

text = label.text()

actual\_databases.append(text)

# Сравниваем ожидаемые и фактические базы данных

self.assertEqual(sorted(actual\_databases), sorted(expected\_databases), "Списки баз данных не совпадают.")

# Проверяем, что количество отображаемых баз данных соответствует ожидаемому

actual\_count = len(actual\_databases)

self.assertEqual(actual\_count, expected\_count, "Количество отображаемых баз данных не соответствует ожидаемому.")

Во время первого тестирования в программе была обнаружена уязвимость: если расширения базы данных будет написано прописными буквами, то чтение не произойдет – Рисунок 8. Решением послужило приведение имени файла с его расширением к строчным буквам – Рисунок 9, 10.

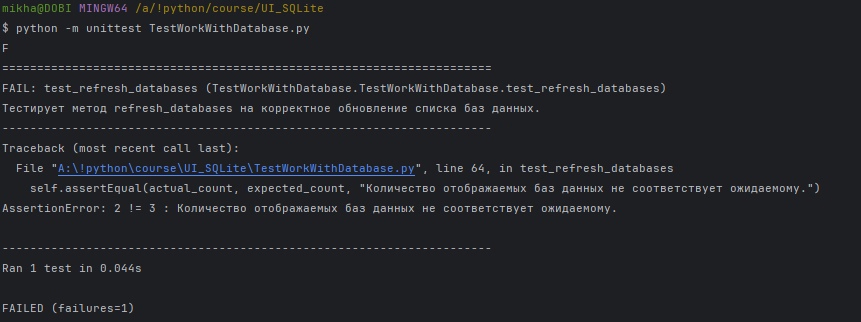


Рисунок 8 – Тестирование чтения списка базы данных

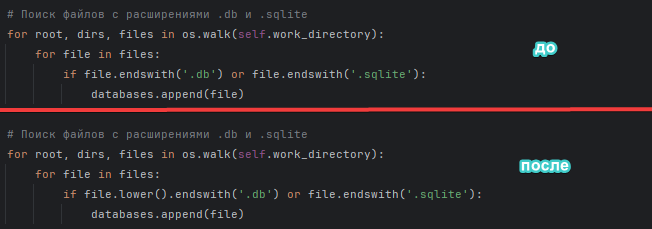


Рисунок 9 – Исправление программы.

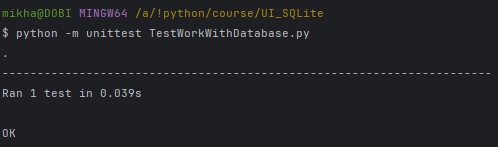


Рисунок 10 – Повторное тестирование.

Следующим этапом тестирования является создание класса TestCreateDbDialog, направленный на проверку создания базы данных. Данный класс включает в себя 4 теста: создание базы данных без расширения; создание существующей базы данных; создание базы данных без имени; создание базы данных в несуществующей директории. Результат выполнения тестирования – Рисунок 11.

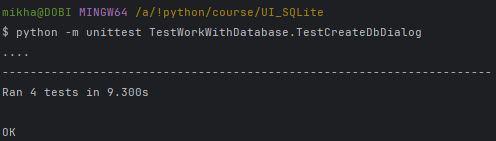


Рисунок 11 – Тестирование создания базы данных.

Был разработан тест удаления базы данных, который включает в себя проверку удаления базы данных при подтверждении удаления, при отказе удаления, попытку удаления несуществующей БД и проверку вызова метода обновления базы данных после удаления одной из них. Результат тестирования созданного класса – Рисунок 12.

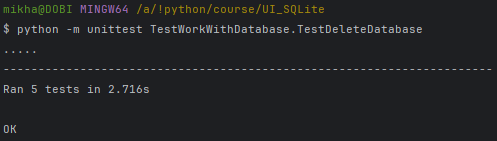


Рисунок 12 – Тестирование удаления базы данных.

Следующее тестирование направлено на выявления ошибок в программе при работе с изменениями в таблице QTableWidget. Создан класс TestSaveChanges, который исследует работу сохранения данных в таблицу базы данных из QTableWidget. Первый тест создает новую строку в QTableWidget и вносит в неё значения, после чего идет сохранение данных в таблицу базы данных. Следующим этапом выполняется sql-запрос для обнаружения необходимой нам строки. Если её не существует, то выводится ошибка. Если она существует, но не все данные занесены корректно – ошибка. Результат выполнения – Рисунок 13.

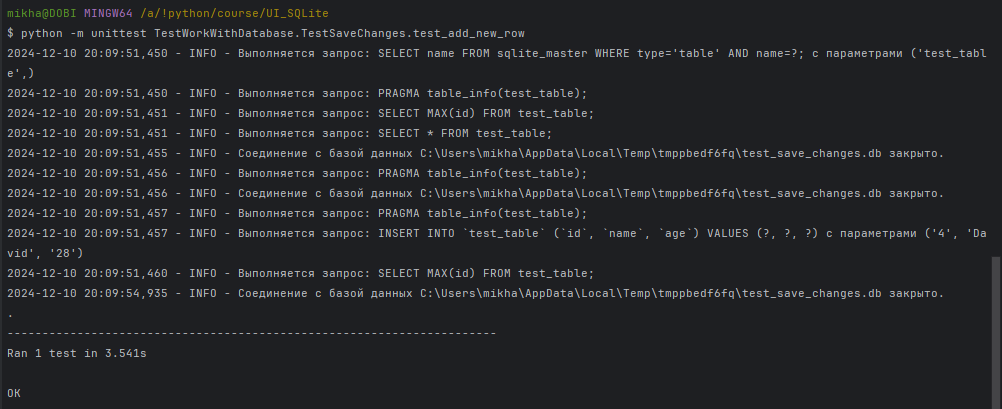


Рисунок 13 – Сохранение новой строки в таблицу БД.

Следующий тест направлен на выявление ошибок при обновлении существующей строки. Для этого изменим в существующей таблице QTableWidget возраст пользователя «Alice», сохраним изменения и проверим значения в таблице базы данных. Результат тестирования – Рисунок 14.

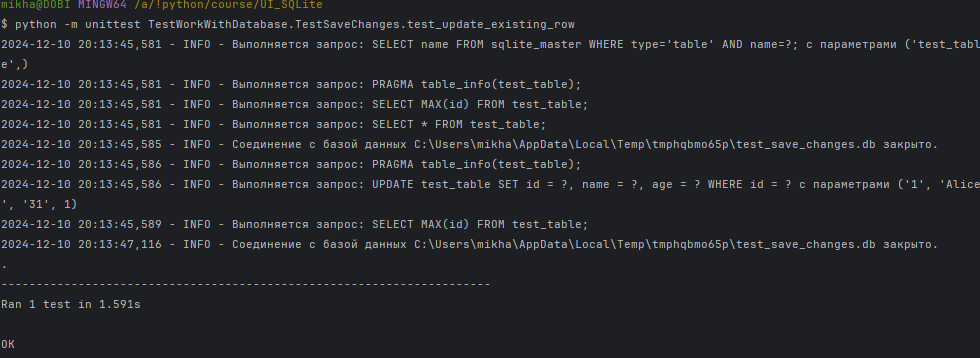


Рисунок 14 – Сохранение измененной строки.

Тестирование, направленное на ожидаемое поведение элементов, выдало ошибку. При попытке сохранения значений без каких-либо изменений, поля в таблице QTableWidget неожиданно помечались флагом «isChanged», что связано с первоначальным отображением таблицы – Рисунок 15.

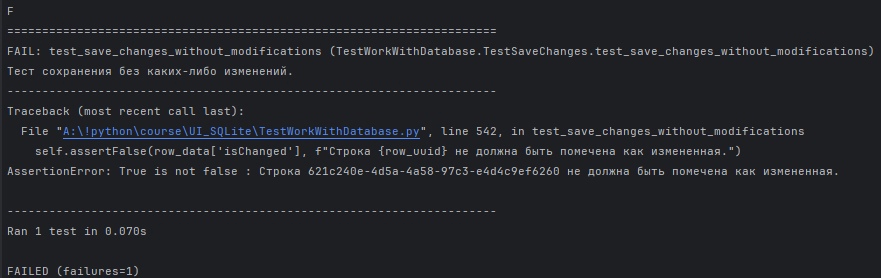


Рисунок 15 – Ошибка тестирования.

Решением послужила заморозка служебных сигналов таблицы в момент её внесения в QTableWidget – «self.ui.tableWidget.blockSignals(True)».

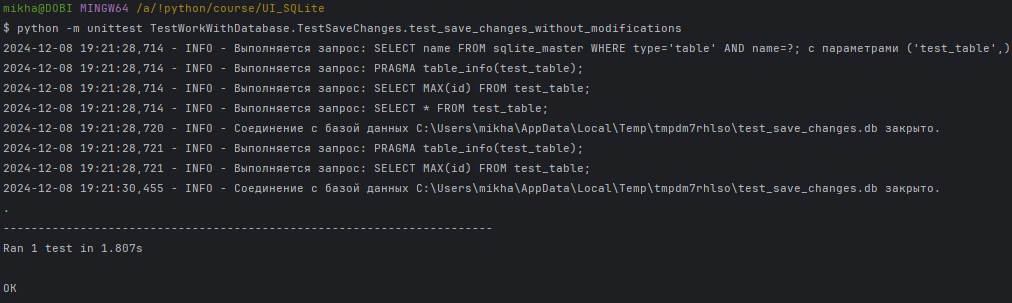


Рисунок 16 – Повторное тестирование.

В результате проведения комплексного тестирования пользователь защищен от неожиданного поведения программы. Итоговое тестирование – Рисунок 17.

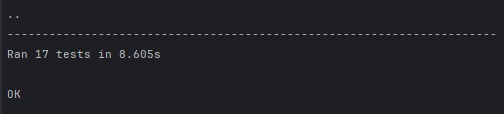


Рисунок 17 – Комплексное тестирование программы.

## Руководство пользователя

Для запуска программы необходимо запустить исполняемый файл, после чего открывается главное окно, в котором необходимо выбрать рабочую директорию – Рисунок 18.

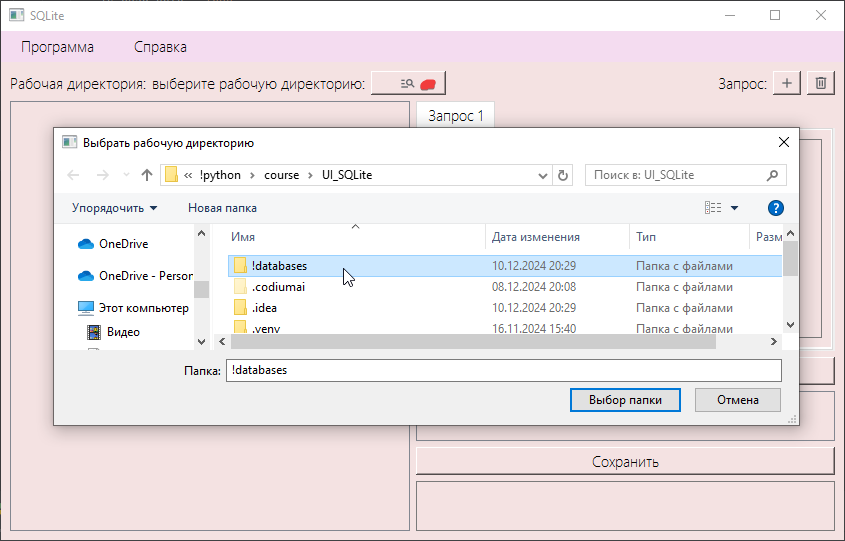


Рисунок 18 – Выбор рабочей директории.

После выбора рабочей директории необходимо создать базу данных. Для этого необходимо нажать кнопку возле надписи создания базы данных, после чего открывается окно, в котором необходимо указать название базы данных – Рисунок 19.

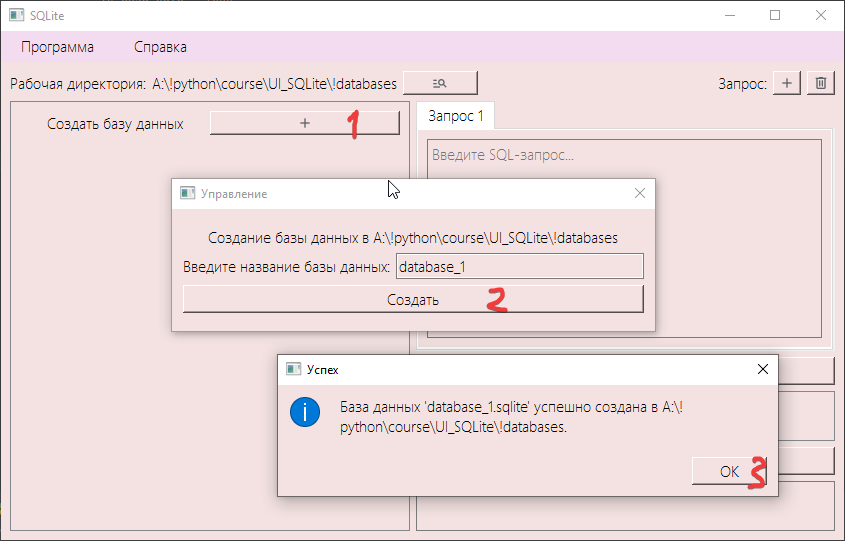


Рисунок 19 – Создание базы данных.

После создания БД необходимо создать таблицу – Рисунок 20.

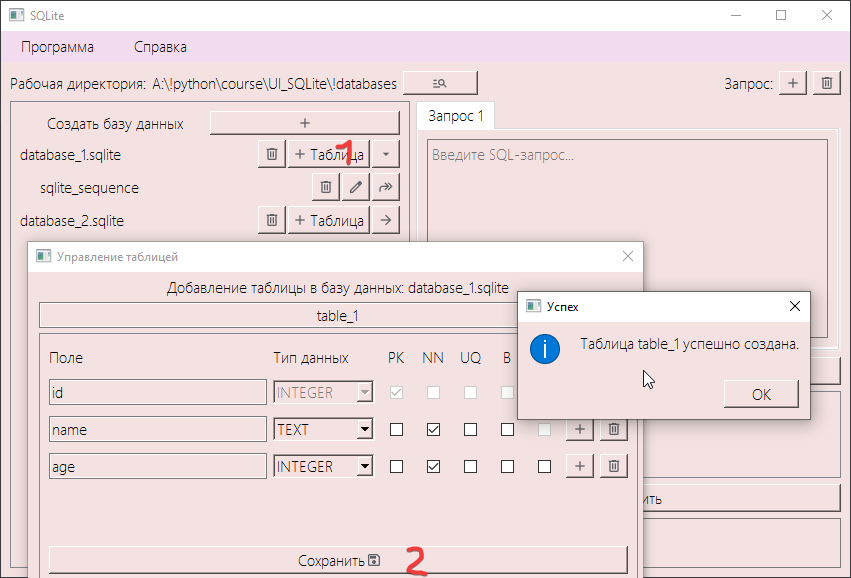


Рисунок 20 – Создание таблицы базы данных.

Отображение таблицы находится рядом с кнопкой добавления. Таблица появится в правом нижнем поле. Добавим значения в таблицу и попробуем их сохранить – Рисунок 21.

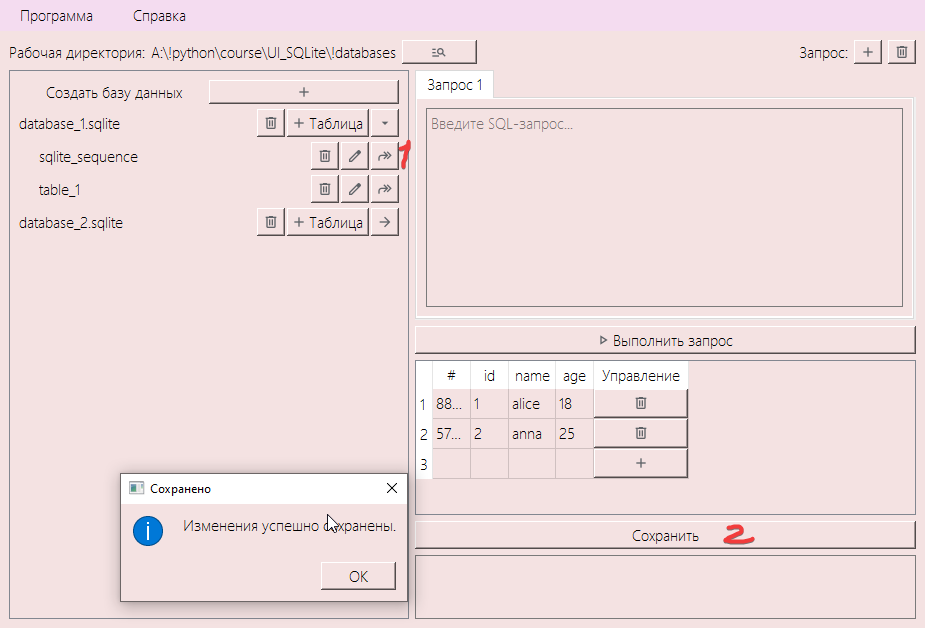


Рисунок 21 – Добавление значений в таблицу.

Для изменения таблицы можно воспользоваться базовым инструментом приложения, который также постарается перенести данные в измененную таблицу, если соответствующие типы данных и ограничения не конфликтуют – Рисунок 22. Структура новой таблицы – Рисунок 23.

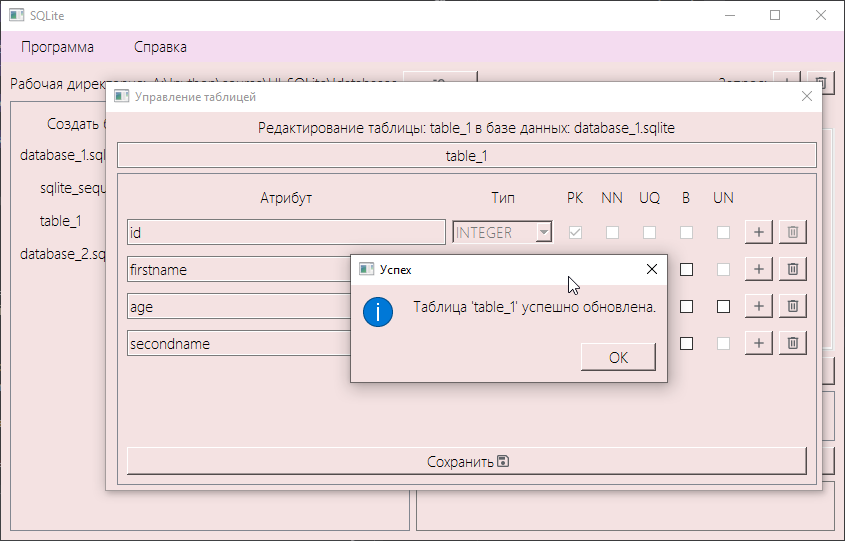


Рисунок 22 – Изменение структуры таблицы.

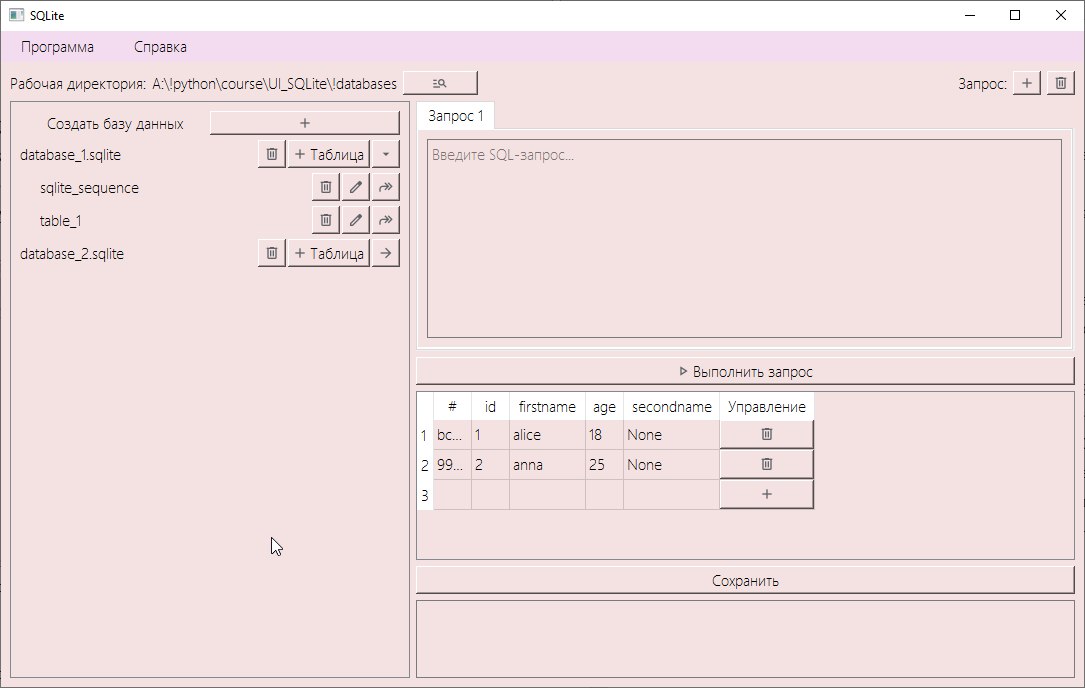


Рисунок 23 – Новая структура таблицы.

Приложение также поддерживает интерактивный текстовый редактор. Покажем содержимое таблицы при помощи SQL-запроса, рисунок 24-26.

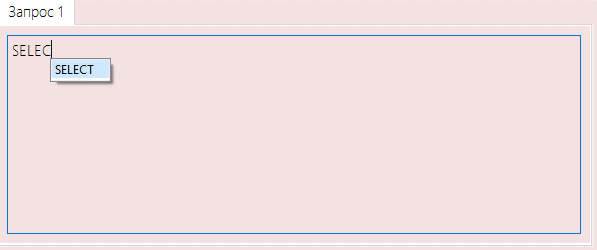


Рисунок 24 – Написание запроса.

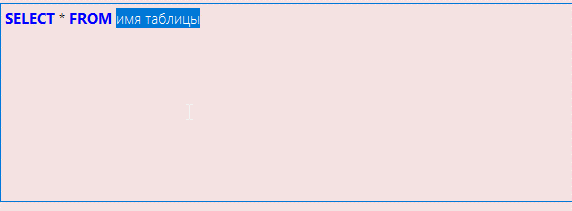


Рисунок 25 – Написание запроса.

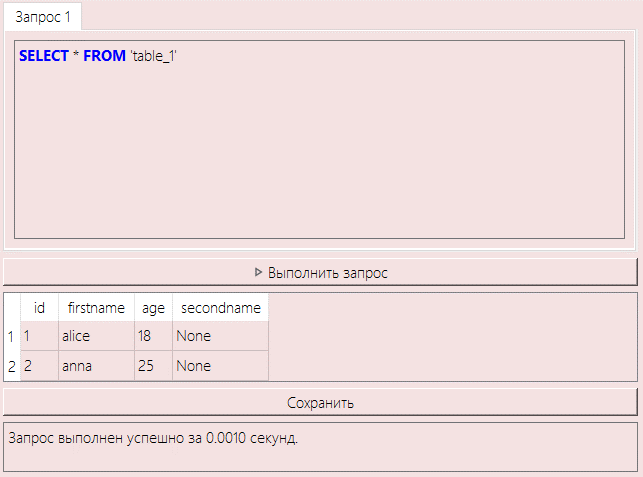


Рисунок 26 – Результат выполнения запроса.

# Список литературы

1. Гаффарова Н.А. Основы программирования на Python: Учебное пособие / Н.А. Гаффарова, И.В. Воронин. — М.: Изд-во МГТУ им. Н.Э. Баумана, 2018. — 432 с.
2. Седунова И.С. Python для начинающих. Пошаговое руководство / И.С. Седунова. — СПб.: Питер, 2020. — 352 с.
3. Кнут Д. Искусство программирования. Том 1: Основы / Д. Кнут; Перевод с англ. И.В. Гомзякова. — М.: Вильямс, 2019. — 672 с.
4. Павлов С. В. Программирование на Python. Простое введение в Python для начинающих / С.В. Павлов. — М.: ДМК Пресс, 2017. — 256 с.
5. Стивенс Р. Введение в Python. Теория и практика / Р. Стивенс; Перевод с англ. — М.: Альпина Паблишер, 2018. — 504 с.
6. Питон В. Разработка приложений на Python: Введение в программирование / В. Питон. — М.: БХВ-Петербург, 2021. — 312 с.