**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра АПУ**

отчет

**по учебной практике**

**Тема: Разработка пользовательского приложения   
для работы с базой данных**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1371 |  | Ларионов Н.М. |
| Руководитель |  | Михайлов Ф. |

Санкт-Петербург

2023

**ЗАДАНИЕ**

**на УЧЕБНУЮ практику**

|  |  |  |
| --- | --- | --- |
| Студент Ларионов Н.М. | | |
| Группа 1371 | | |
| Тема практики: разработка базы данных | | |
| Задание на практику:  Разработать базу данных  База развертывается на персональном компьютере на языке SQL при помощи СУБД. Для базы данных разрабатывается приложение с графическим интерфейсом на языке Python для удобной работы с запросами.  Тема учебной практики - менеджмент успеваемости и распределение по группам студентов. Приложение должно поддерживать следующие функции:   * Создание, редактирование, удаление предметов * Создание, редактирование, удаление групп * Создание, изменение, перераспределение, удаление студентов, исходя из выбранной группы * Добавление и удаление оценок, исходя из выбранных группы, студента и предмета | | |
| Сроки прохождения практики: 06.02.2023 – 01.06.2023 | | |
| Дата сдачи отчета: 07.06.2023 | | |
| Дата защиты отчета: : 07.06.2023 | | |
|  | | |
| Студент |  | Ларионов Н.М. |
| Руководитель |  | Михайлов Ф. |

**Аннотация**

Цель работы: получить практические навыки в разработке баз данных, работе с объектно-ориентированном программированием и в разработке пользовательского интерфейса. Также, изучение приложений и библиотек, позволяющих реализовать взаимосвязь программы с базой данных.

**ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ и СОКРАЩЕНИЯ**

Реляционная база данных – набор данных в формате таблиц с предопределенными связями между ними. В таких базах данных активно используются первичные и внешние ключи.

Первичный ключ - набор полей со значениями, которые являются уникальными для всей таблицы. Значения ключа могут использоваться для обозначения всех записей, при этом каждая запись имеет отдельное значение ключа.

Внешний ключ - набор полей, ссылающихся на первичный ключ другой таблицы. Таблица с внешним ключом называется дочерней, а с первичным — родительской.

SQL - декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных. Является, прежде всего, информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных.

SQL-запрос – запрос к базе данных на выполнение конкретного действия.

СУБД – аббревиатура от Системы Управления Базами Данных

Кортеж - последовательность элементов, имеет фиксированную длину. В Python представляет собой список с неизменяемой длиной.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 6 |
| 1. | Среда разработки | 7 |
| 2. | Системы управления базами данных | 7 |
| 3. | Приложение | 8 |
| 3.1. | Описание приложения | 8 |
| 3.2. | Главное меню | 9 |
| 3.3. | Меню учителя | 9 |
| 3.4. | Меню администратора | 11 |
| 4. | База данных | 16 |
| 5. | Функции SQL-запросов | 19 |
| 6. | Пользовательский интерфейс | 22 |
| 6.1 | Создание окна | 22 |
| 6.2. | Диалоговое окно | 25 |
| 6.3. | Переключатели | 30 |
|  | Заключение | 32 |
|  | Список использованных источников | 33 |

**введение**

В современной сфере информационных технологий важным направлением является разработка баз данных – структур, позволяющих быстро и удобно работать с большими объемами данных. Среди баз данных довольно сильно популярны так называемые реляционные базы данных, основанные на языке SQL (Structured Query Language). Такие базы данных предоставляют необходимую возможность быстрой и точной работы с большими данными, и к тому же крайне удобный в написании. Современный специалист обязан разбираться в данном направлении.

Прохождение практики нацелено на получении практических навыков в проектировании приложения, позволяющего работать с базой данных через пользовательский интерфейс. Реализация данного проекта тесно связана с изучением объектно-ориентированного программирования.

1. **Среда разработки**

**Pycharm** – кроссплатформенная интегрированная среда разработки для языка программирования Python. Предоставляет пользователю комплекс средств для написания кода и визуальный отладчик. Позволяет пользователю комфортно работать с разными библиотеками.

1. **системы управления базами данных**

**PgAdmin** - популярная многофункциональная платформа администрирования и разработки с открытым исходным кодом для PostgreSQL. Использовалась для создания и настройки базы данных, так как удобна для администрирования.

**DBeaver** - клиентское программное приложение SQL и инструмент администрирования базы данных. Позволяет комфортно выполнять SQL-запросы.

1. **Приложение**
2. **Описание приложения**

Конечный учебный проект представляет собой приложение, написанное на языке Python, которое позволяет удобно выполнять требуемые пользователю SQL-запросы в развернутой на персональном компьютере базе данных.

Приложение поддерживает следующий функционал:

* Создание, редактирование, удаление предметов
* Создание, редактирование, удаление групп
* Создание, изменение, перераспределение, удаление студентов, исходя из выбранной группы
* Добавление и удаление оценок, исходя из выбранных группы, студента и предмета

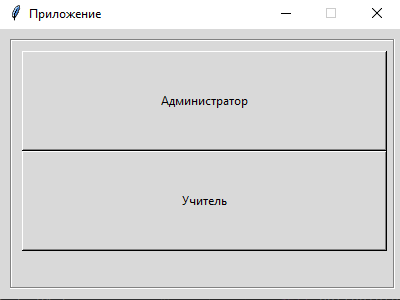
Реализация учебного проекта состоит из трех частей:

* 1. Создании и настройке базы данных
  2. Написании на языке Python набора функций, позволяющей осуществлять SQL-запросы непосредственно через программу минуя предназначенный для этого инструментарий СУБД
  3. Написании пользовательского интерфейса приложения, позволяющего комфортно выполнять заготовленные запросы.

Приложение является проектом Python и состоит из нескольких файлов, которые связаны друг с другом.

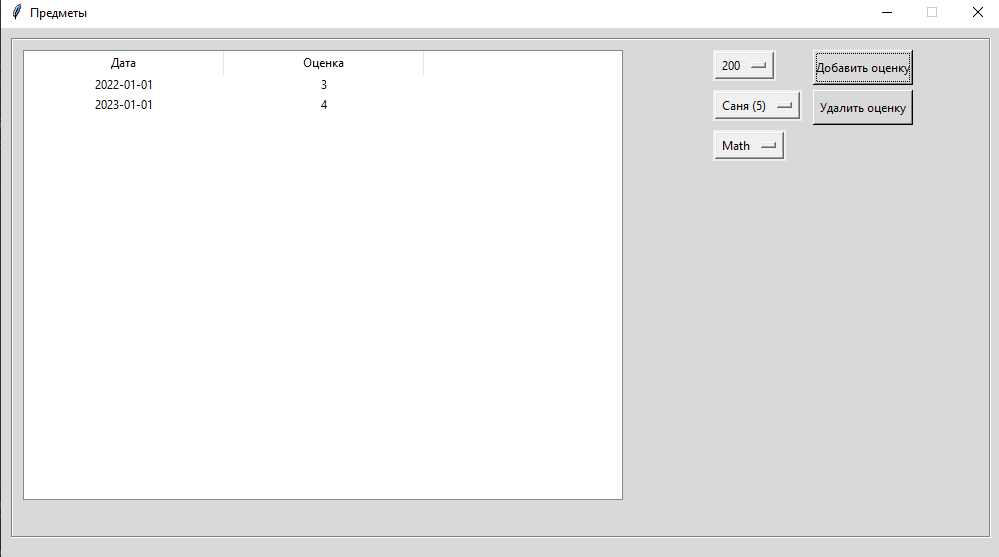
1. **Главное меню**

Приложение открывается при запуске файла, отвечающего за главное меню, и выглядит следующим образом:

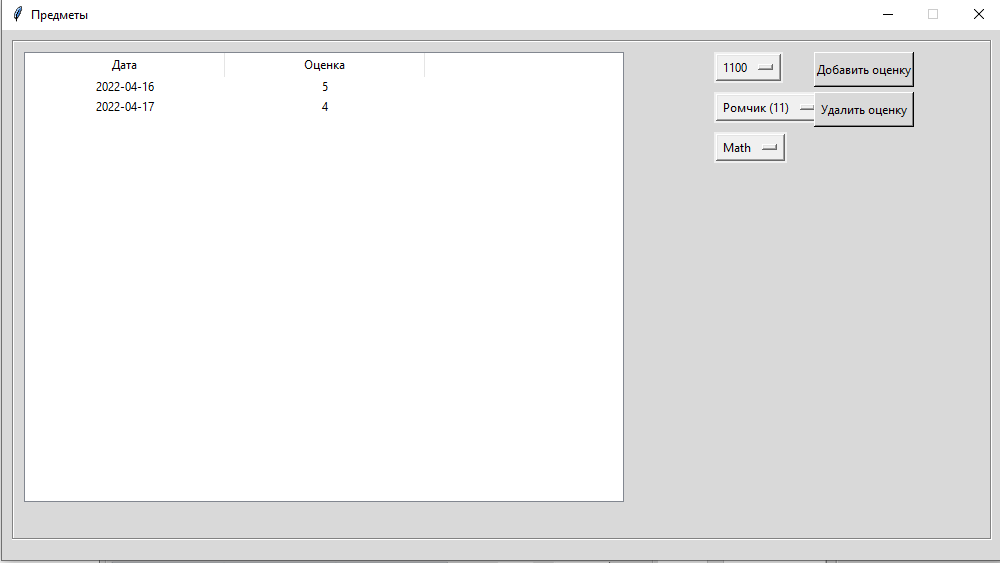


1. **Меню учителя**

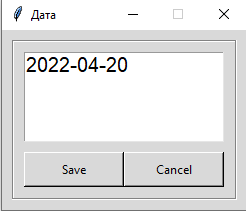
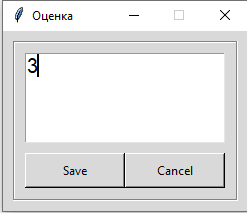
Меню учителя позволяет пользователю добавлять и удалять оценку у заданного ученика по заданному предмету.

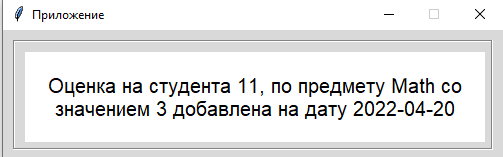


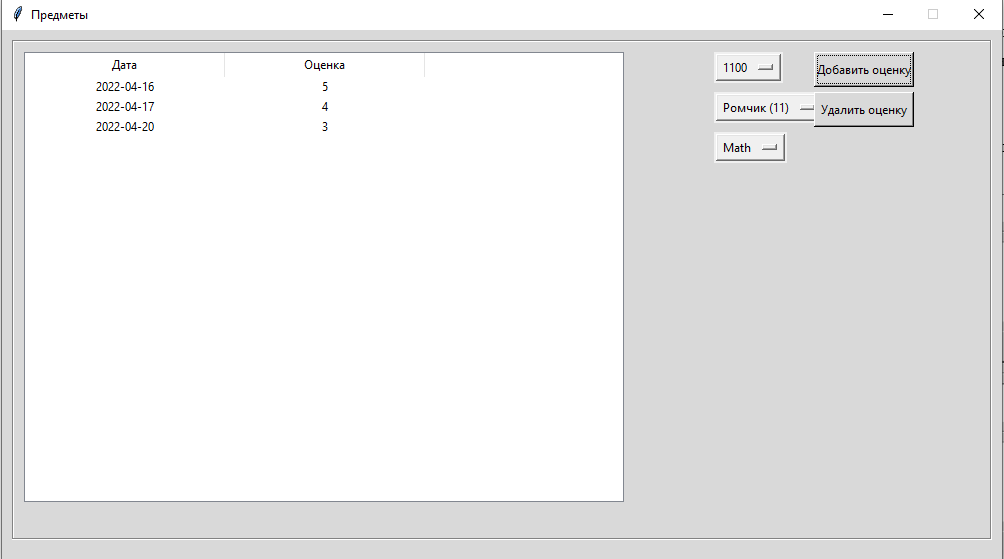
Список оценок требуемого ученика по заданному предмету загружается после изменения состояния одного из трех переключателей – группы, студента, предмета. При этом, при первом открытии меню предусмотрены стартовые состояния.



Добавление новой оценки происходит при нажатии соответствующей кнопки:



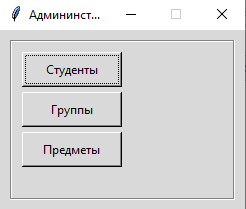




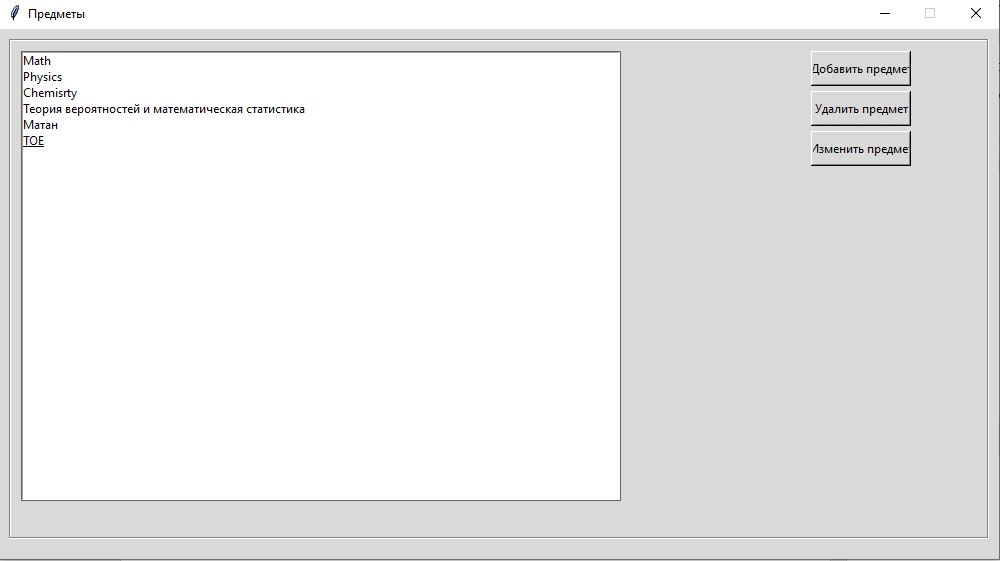
1. **Меню администратора**

Меню администратора позволяет добавлять, изменять и удалять значения из трех таблиц – предметов, групп и студентов.

После нажатия кнопки администратора, появляется следующее окно, позволяющее пройти в дальнейшее меню.



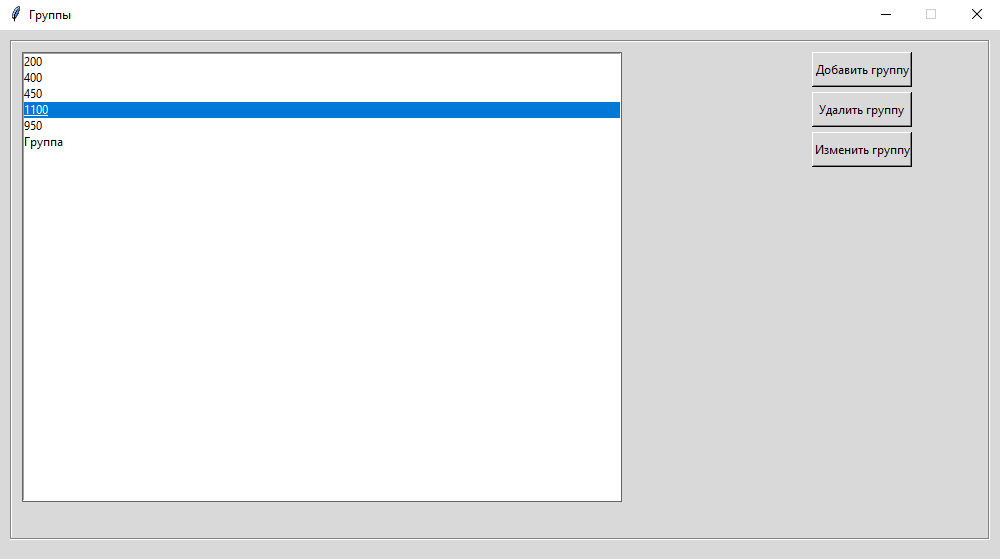
Меню групп и меню предметов довольно похожи и выглядят следующим образом:

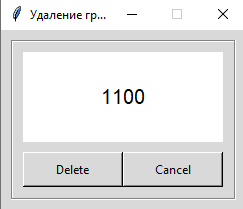




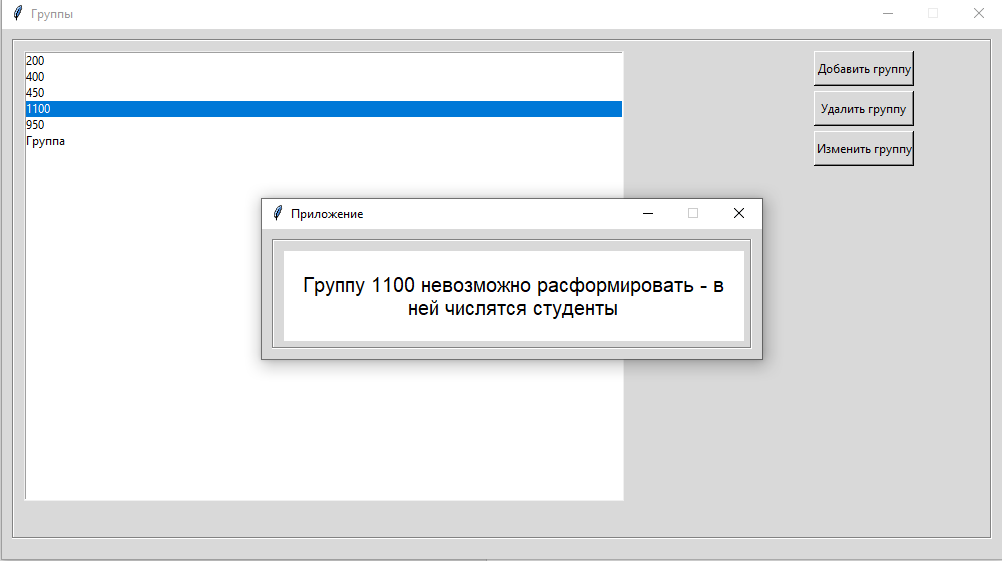
Взаимодействие с заданной группой происходит через данные, которые передает курсор.

Например, так будет выглядеть удаление группы:

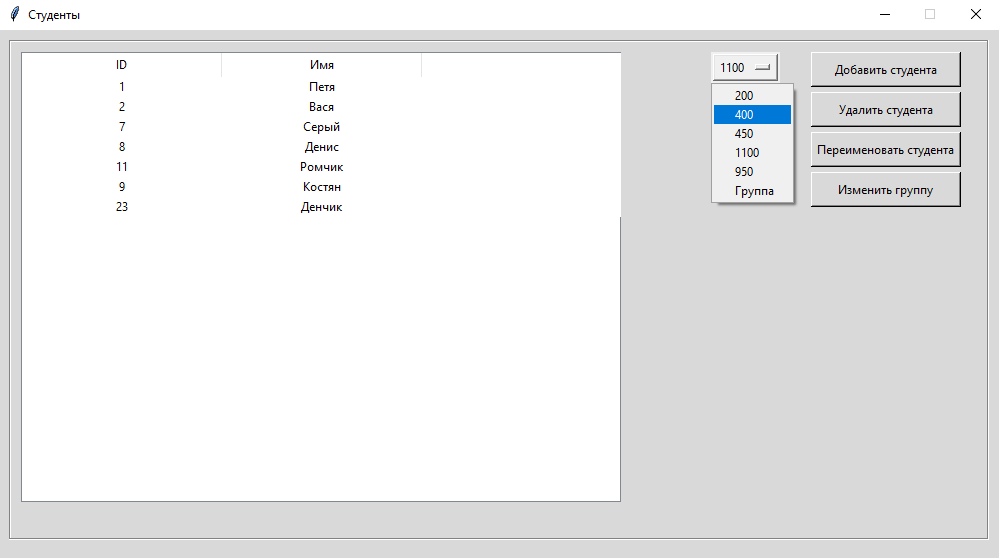


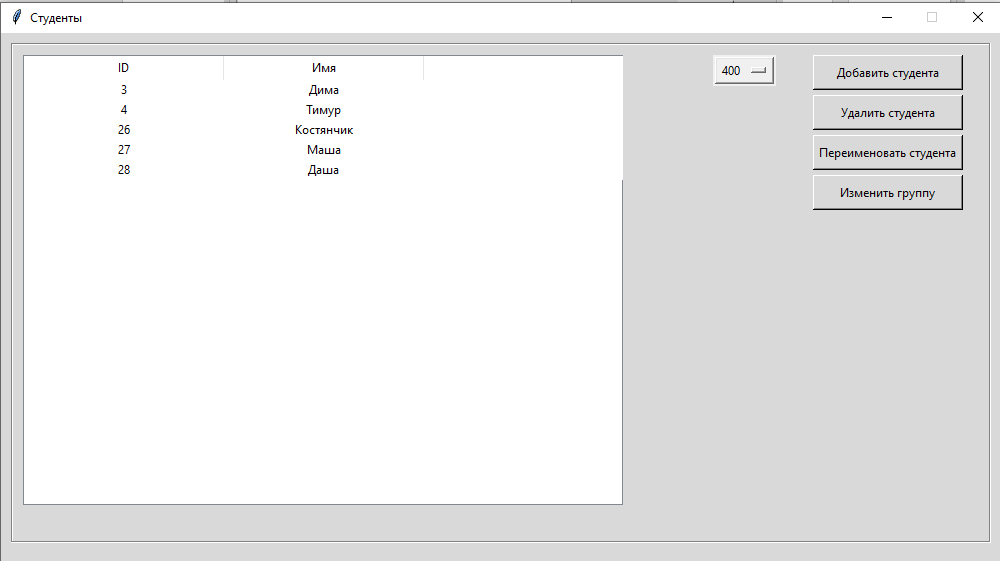


В программе предусмотрена защита от удаления строк таблицы, присутствующих в других таблицах в виде внешних ключей. Так, например, нельзя удалить группу, если в ней числятся студенты.



Немного отличается меню студентов – так как каждый студент привязан к конкретной группе, был добавлен соответствующий переключатель состояния:

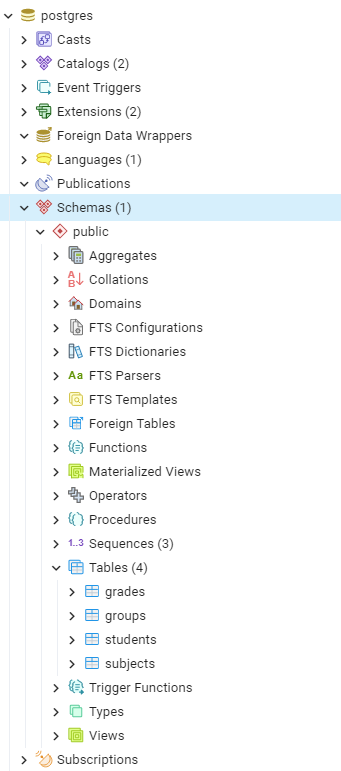




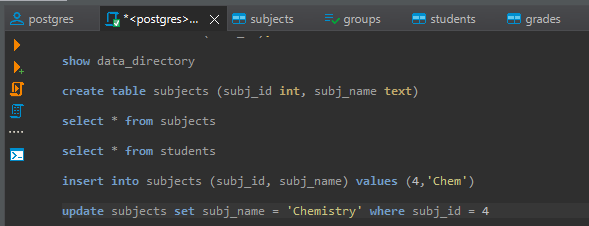
Получение данных для действий с конкретным студентом происходит также через курсор и переменную состояния группы.

1. **база данных**

База данных создана и развернута на персональном компьютере при помощи СУБД PgAdmin, где так же были настроены соединения.

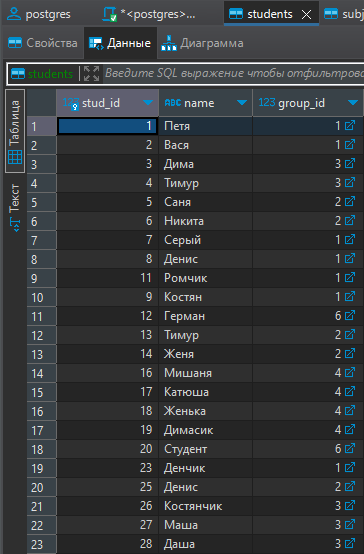


Все SQL-запросы были произведены в СУБД DBeaver.

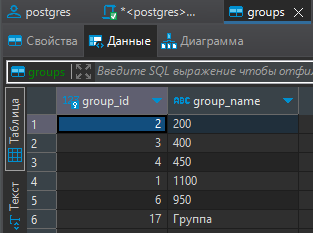


Также, через СУБД DBeaver можно просматривать заполненные таблицы

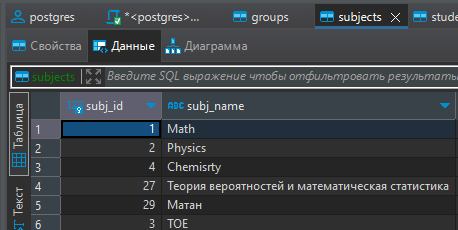
Студенты:



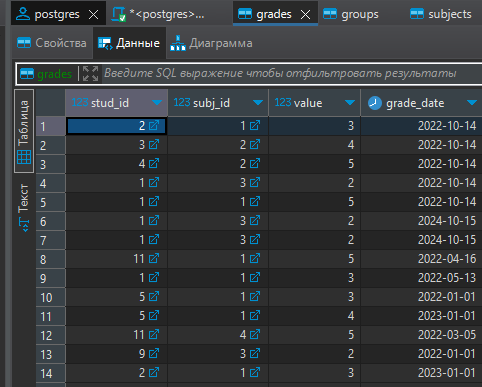
Группы:



Предметы:



Оценки:



1. **Функции sql запросов**

Весь функционал функций, выполняющих SQL-запросы прописаны в отдельном файле с использованием библиотеки Psycopg.

В первую очередь описывается класс соединения, за подключение и разрыв соединения с развернутой базой данных.

class Connection:  
 *# Статическая переменная* connection = None  
  
 @staticmethod  
 def get\_connection():  
 if Connection.connection == None:  
 Connection.connection = psycopg2.connect( *# Открытие соединения к базе данных* database="postgres",  
 user="postgres",  
 password="1111",  
 host="127.0.0.1",  
 port="5432"  
 )  
 return Connection.connection  
  
 @staticmethod  
 def close\_connection():  
 if Connection.connection != None:  
 Connection.connection.close()

Далее описываются функции, выполняющие SQL-запросы.

В составе каждой функции есть:

* 1. Установка соединения
  2. Открытие курсора
  3. Выполнение курсора – выполнение через курсор SQL-запроса
  4. Заполнение курсора данными, полученными при помощи данного запроса
  5. Закрытие транзакции в случае, если запрос изменял содержание таблицы
  6. Print – выводит в консоль отладочные сообщения
  7. Return – возвращает текстовую строку, которая используются как текст всплывающего информационного окна.

Вот пример функций, которые описывают SQL-запросы для таблицы предметов:

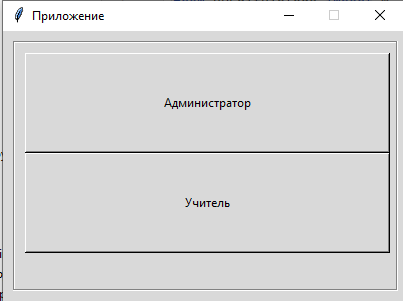
def subject\_exists(subject\_name):  
 *# Открытие курсора* connection = Connection.get\_connection()  
 cur = connection.cursor() *# Открытие курсора* cur.execute(  
 f"select count(\*) from subjects s where subj\_name = '{subject\_name}'"  
 )  
  
 if cur.fetchone()[0] > 0:  
 cur.close() *# Закрытие курсора* return True  
 else:  
 cur.close() *# Закрытие курсора* return False  
  
def subject\_get\_id(subject\_name):  
 connection = Connection.get\_connection()  
 cur = connection.cursor() *# Открытие курсора* cur.execute(  
 f"select subj\_id from subjects s where subj\_name = '{subject\_name}'"  
 )  
  
 subject\_id = cur.fetchone()  
  
 cur.close() *# Закрытие курсора* print(subject\_id)  
 if subject\_id:  
 return subject\_id[0]  
 else:  
 return 0  
  
def subject\_create(subject\_name):  
 if not subject\_exists(subject\_name):  
 *# Открытие курсора* connection = Connection.get\_connection()  
 cur = connection.cursor() *# Открытие курсора* cur.execute(  
 f"insert into subjects (subj\_id, subj\_name) values (nextval('subj\_id\_seq'), '{subject\_name}')"  
 )  
  
 cur.close() *# Закрытие курсора* connection.commit() *# Закрытие транкзации* print(f"Предмет {subject\_name} успешно собран")  
 else:  
 print(f"Предмет {subject\_name} уже существует")  
  
def subject\_delete(subject\_name):  
 if subject\_exists(subject\_name):  
 if not grade\_subject\_exists(subject\_name):  
 *# Открытие курсора* connection = Connection.get\_connection()  
 cur = connection.cursor() *# Открытие курсора* cur.execute(  
 f"delete from subjects where subj\_name = '{subject\_name}'"  
 )  
  
 cur.close() *# Закрытие курсора* connection.commit() *# Закрытие транкзации* print(f"Предмет {subject\_name} успешно удален")  
 return f"Предмет {subject\_name} успешно удален"  
 else:  
 print(f"По предмету {subject\_name} проставлены оценки")  
 return "Необходимо удалить оценки по предмету"  
 else:  
 print(f"Предмета {subject\_name} не существует")  
 return False  
  
def subject\_rename(subject\_name, new\_subject\_name):  
 if subject\_exists(subject\_name):  
 if not subject\_exists(new\_subject\_name):  
 *# Открытие курсора* connection = Connection.get\_connection()  
 cur = connection.cursor() *# Открытие курсора* cur.execute(  
 f"update subjects set subj\_name = '{new\_subject\_name}' where subj\_name = '{subject\_name}'"  
 )  
  
 cur.close() *# Закрытие курсора* connection.commit() *# Закрытие транкзации* print(f"Название предмета {subject\_name} изменено на {new\_subject\_name}")  
 return f"Название предмета {subject\_name} изменено на {new\_subject\_name}"  
 else:  
 print(f"Предмет с названием {new\_subject\_name} уже существует")  
 return f"Предмет с названием {new\_subject\_name} уже существует"  
 else:  
 print(f"Предмета с названием {subject\_name} не существует")  
 return f"Предмета с названием {subject\_name} не существует"  
  
def subjects\_get\_list():  
 subjects = []  
  
 *# Открытие курсора* connection = Connection.get\_connection()  
 cur = connection.cursor() *# Открытие курсора* cur.execute(  
 f"select subj\_name from subjects"  
 )  
  
 for line in cur.fetchall():  
 subjects.append(line[0])  
  
 cur.close() *# Закрытие курсора* return subjects

1. **Пользовательский интерфейс**

Пользовательский интерфейс выполнен с использованием библиотеки Tkinter. Для удобства разработки и чтения, каждое окно описано в отдельном файле.

1. **Создание окна**
   1. Объявляется родительское окно в случае, если новое окно было вызвано. От родительского окна зависит местоположение дочернего окна на экране. Кроме этого, при работе с дочерним окном, все родительские окна блокируются и при стандартных условиях с ними нельзя взаимодействовать, пока не будет закрыто дочернее окно.
   2. Объявляется класс нового окна, задаются базовые параметры – его размеры, возможность масштабирования, цвет фона и текста.
   3. Задаются параметры рамки, ее размер, цвет, рельеф и т.д. Весь дальнейший интерфейс
   4. Создаются остальные объекты – кнопки, переключатели и т.д.  
      Прописывается функция, которая вызывается командой при взаимодействии с объектом.
   5. Прописываются функции, которые вызываются командами объектов. В данном приложении функции кнопок и переключателей либо вызывают дочерние окна, либо вызывают функции, выполняющие SQL-запросы, либо меняют состояние.

В качестве примера рассмотрим окно, которое пользователь видит при открытии главного меню.



class TopLevelWindow:  
 def \_\_init\_\_(self, root):  
 *# Создаем основное окно* root.geometry("400x270") *# Размеры окна* root.resizable(False, False) *# Может ли изменять размер* root.title("Приложение") *# Заголовок окна* root.configure(background="#d9d9d9") *# Фоновый цвет окна* root.configure(highlightbackground="#d9d9d9") *# Фоновый цвет текста* root.configure(highlightcolor="black") *# Цвет текста* self.root = root  
  
 *# Создаем рамку* self.Frame1 = tk.Frame(self.root) *# Создаем окно, наследник приложения* self.Frame1.place(x=10, y=10, height=250, width=385) *# Размеры окна* self.Frame1.configure(relief='groove') *# Рельеф границ окна* self.Frame1.configure(borderwidth="2") *# Ширина границ окна* self.Frame1.configure(background="#d9d9d9") *# Фоновый цвет окна* self.Frame1.configure(highlightbackground="#d9d9d9") *# Фоновый цвет текста* self.Frame1.configure(highlightcolor="black") *# Цвет текста  
  
 # Создаем кнопку (Администратора)* self.ButtonAdmin = tk.Button(self.Frame1) *# Привязывается к рамке* self.ButtonAdmin.place(x=10, y=10, height=100, width=365) *# Размеры* self.ButtonAdmin.configure(activebackground="#d9d9d9") *# Фон активной кнопки* self.ButtonAdmin.configure(activeforeground="black") *# Цвет фона активной кнопки* self.ButtonAdmin.configure(background="#d9d9d9") *# Цвет фона пассивной кнопки* self.ButtonAdmin.configure(compound='left')  
 self.ButtonAdmin.configure(disabledforeground="#a3a3a3") *# Цвет нерабочей кнопки* self.ButtonAdmin.configure(foreground="#000000") *# Цвет текста* self.ButtonAdmin.configure(highlightbackground="#d9d9d9") *# Прицелился на кнопку* self.ButtonAdmin.configure(highlightcolor="black")  
 self.ButtonAdmin.configure(pady="0") *# Отступ по y* self.ButtonAdmin.configure(text='''Администратор''')  
 self.ButtonAdmin.configure(command=self.CreateAdminWindow) *# Сюда закидываем еще одно окно  
  
 # Создаем кнопку (Учителя)* self.ButtonTutor = tk.Button(self.Frame1) *# Привязывается к рамке* self.ButtonTutor.place(x=10, y=110, height=100, width=365) *# Размеры* self.ButtonTutor.configure(activebackground="#d9d9d9") *# Фон активной кнопки* self.ButtonTutor.configure(activeforeground="black") *# Цвет фона активной кнопки* self.ButtonTutor.configure(background="#d9d9d9") *# Цвет фона пассивной кнопки* self.ButtonTutor.configure(compound='left')  
 self.ButtonTutor.configure(disabledforeground="#a3a3a3") *# Цвет нерабочей кнопки* self.ButtonTutor.configure(foreground="#000000") *# Цвет текста* self.ButtonTutor.configure(highlightbackground="#d9d9d9") *# Прицелился на кнопку* self.ButtonTutor.configure(highlightcolor="black")  
 self.ButtonTutor.configure(pady="0") *# Отступ по y* self.ButtonTutor.configure(text='''Учитель''')  
 self.ButtonTutor.configure(command=self.CreateTeacherWindow)  
 *# self.ButtonAdmin.configure(command=fw\_support.next\_word)* def CreateAdminWindow(self, \*args):  
 dialog = AdminWindow(self)  
  
 dialog.show()  
  
 def CreateTeacherWindow(self, \*args):  
 dialog = GradesStudentWindow(self)  
 dialog.show()

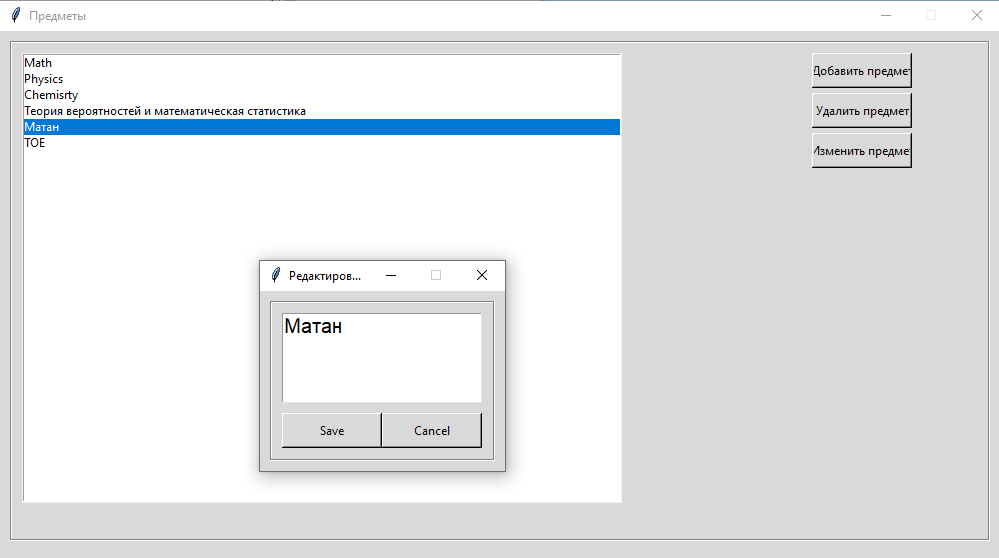
1. **Диалоговое окно**

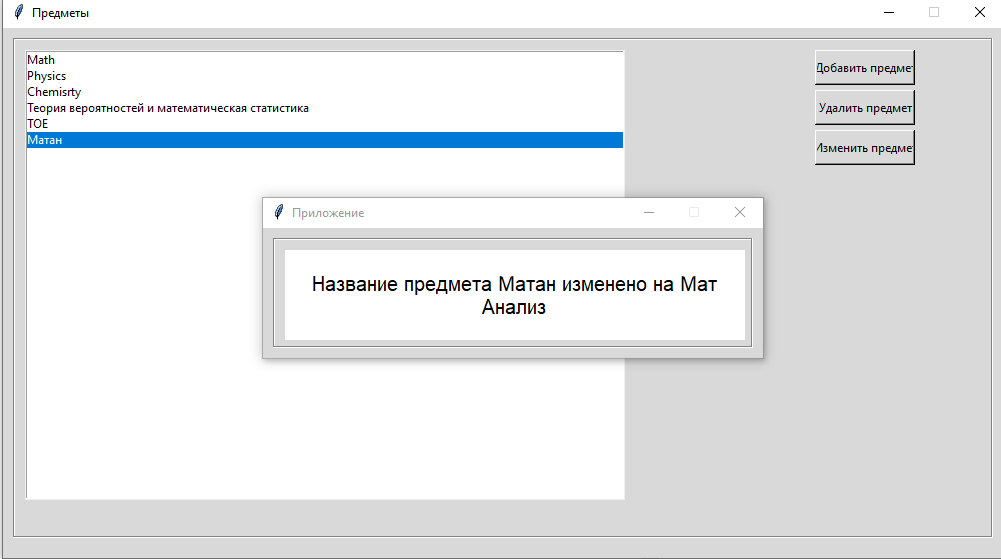
Диалоговые окна – вспомогательные окна, которые вызываются для работы с текстовыми данными.

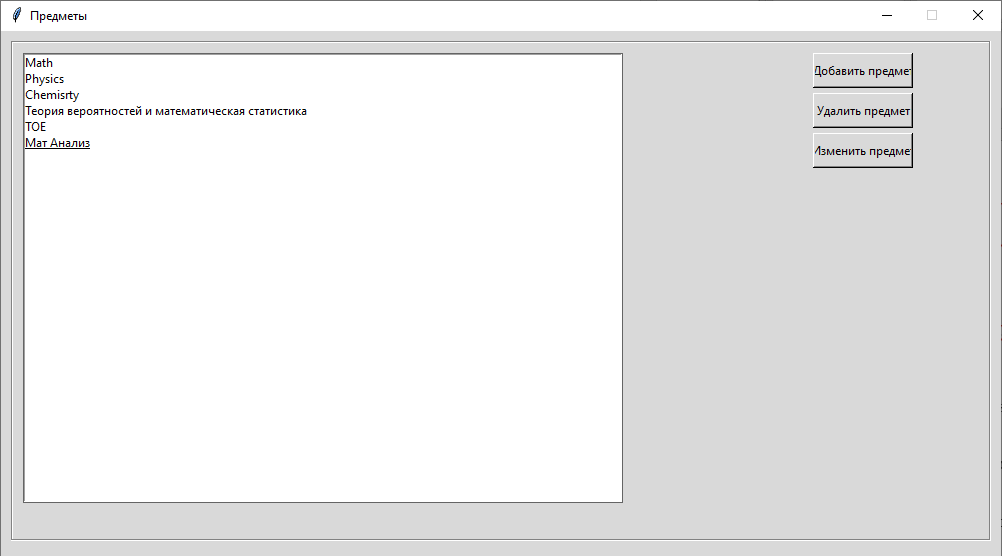
Окна редактирования позволяют вводить данные, которые после нажатия кнопки подтверждения используются в вызове функции SQL-запроса.

Окна удаления выводят информацию удаляемых данных и имеют дополнительную кнопку подтверждения, что бы данные нельзя было удалить случайно одним нажатием.

Ниже предоставлен пример функции, отвечающей за редактирование названия предмета







Код функции:

def SubjectEdit(self):  
 if len(self.SubjectsList.curselection()) == 0:  
 return  
  
 subject\_name = self.SubjectsList.get(self.SubjectsList.curselection()[0])  
  
 edit\_dialog = EditDialog(self, "Редактирование предмета", subject\_name)  
 edit\_dialog.show()  
  
 if edit\_dialog.processed:  
 text = subject\_rename(subject\_name=subject\_name, new\_subject\_name=edit\_dialog.new\_parameter)  
 print("Вывод: ", text)  
 if text:  
 WarningWindow(self, text)  
 self.LoadList()

* 1. Нажатие кнопки редактирования предмета вызывает данную функцию
  2. Функция получает информацию о редактируемых данных из курсора
  3. Кнопка вызывает окно редактирование, в которое пользователь вводит новые данные.
  4. Вызывается функция, выполняющая SQL-запрос. Функция на вход получает данные, которые были получены из курсора и диалогового окна.
  5. В случае, если данная функция SQL-запроса возвращает текст, вызывается предупреждающее окно, которое демонстрирует данный текст. Данное окно имеет только одну кнопку, через которое может быть закрыто
  6. Список, отображаемый внутри окна обновляется, используя в том числе новые данные.

Следующим образом выглядит класс диалогового окна редактирования

class EditDialog():  
 def \_\_init\_\_(self, parentWin, parameter\_name, parameter=''):  
 self.parent = parentWin  
 self.parameter\_name = parameter\_name  
 self.parameter = parameter  
 self.processed = False  
  
 def show(self, settings=False):  
 self.root = Toplevel(self.parent.root)  
 x = self.parent.root.winfo\_x() + 260  
 y = self.parent.root.winfo\_y() + 260  
 self.root.geometry(f"245x180+{x}+{y}")  
 self.root.configure(background="#d9d9d9")  
 self.root.configure(highlightbackground="#d9d9d9")  
 self.root.configure(highlightcolor="black")  
 self.root.title(self.parameter\_name)  
 self.root.resizable(False, False)  
  
 self.Frame = tk.Frame(self.root)  
 self.Frame.place(x=10, y=10, height=160, width=225)  
 self.Frame.configure(relief='groove')  
 self.Frame.configure(borderwidth="2")  
 self.Frame.configure(relief="groove")  
 self.Frame.configure(background="#d9d9d9")  
 self.Frame.configure(highlightbackground="#d9d9d9")  
 self.Frame.configure(highlightcolor="black")  
  
 self.ButtonOk = tk.Button(self.Frame)  
 self.ButtonOk.place(x=10, y=110, height=35, width=100)  
 self.ButtonOk.configure(activebackground="#d9d9d9")  
 self.ButtonOk.configure(activeforeground="black")  
 self.ButtonOk.configure(background="#d9d9d9")  
 self.ButtonOk.configure(compound='left')  
 self.ButtonOk.configure(disabledforeground="#a3a3a3")  
 self.ButtonOk.configure(foreground="#000000")  
 self.ButtonOk.configure(highlightbackground="#d9d9d9")  
 self.ButtonOk.configure(highlightcolor="black")  
 self.ButtonOk.configure(pady="0")  
 self.ButtonOk.configure(text='''Save''')  
 self.ButtonOk.bind("<Button-1>", self.save\_parameter)  
  
 self.ButtonCancel = tk.Button(self.Frame)  
 self.ButtonCancel.place(x=110, y=110, height=35, width=100)  
 self.ButtonCancel.configure(activebackground="#d9d9d9")  
 self.ButtonCancel.configure(activeforeground="black")  
 self.ButtonCancel.configure(background="#d9d9d9")  
 self.ButtonCancel.configure(compound='left')  
 self.ButtonCancel.configure(disabledforeground="#a3a3a3")  
 self.ButtonCancel.configure(foreground="#000000")  
 self.ButtonCancel.configure(highlightbackground="#d9d9d9")  
 self.ButtonCancel.configure(highlightcolor="black")  
 self.ButtonCancel.configure(pady="0")  
 self.ButtonCancel.configure(text='''Cancel''')  
 self.ButtonCancel.configure(command=self.root.destroy)  
  
 self.Text = tk.Text(self.Frame)  
 self.Text.place(x=10, y=10, height=90, width=200)  
 self.Text.configure(font=("Arial", 15))  
 self.Text.configure(background="#FFFFFF")  
 self.Text.configure(foreground="#000000")  
 self.Text.configure(highlightbackground="#d9d9d9")  
 self.Text.configure(highlightcolor="black")  
 self.Text.configure(pady="0")  
 self.Text.delete("1.0", END)  
 self.Text.insert("1.0", self.parameter)  
  
 self.root.grab\_set()  
 self.Text.focus\_set()  
 self.root.wait\_window()  
  
 def save\_parameter(self, \*args):  
 self.processed = True  
 self.new\_parameter = self.Text.get(1.0, END).replace("\n", "")  
 self.root.grab\_release()  
 self.root.destroy()

Следующим образом выглядит класс всплывающего предупреждающего окна

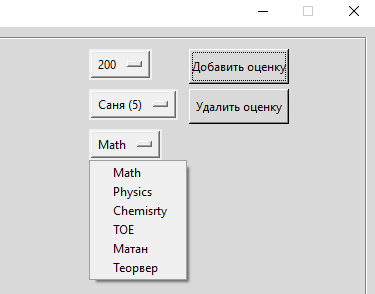
class WarningWindow():  
  
 def \_\_init\_\_(self, parentWin, text):  
 self.parent = parentWin  
  
 self.root = Toplevel(self.parent.root)  
 x = self.parent.root.winfo\_x() + 260  
 y = self.parent.root.winfo\_y() + 200  
 self.root.geometry(f"500x130+{x}+{y}")  
 self.root.configure(background="#d9d9d9")  
 self.root.configure(highlightbackground="#d9d9d9")  
 self.root.configure(highlightcolor="black")  
 self.root.resizable(False, False)  
  
 self.Frame = tk.Frame(self.root)  
 self.Frame.place(x=10, y=10, height=110, width=480)  
 self.Frame.configure(relief='groove')  
 self.Frame.configure(borderwidth="2")  
 self.Frame.configure(relief="groove")  
 self.Frame.configure(background="#d9d9d9")  
 self.Frame.configure(highlightbackground="#d9d9d9")  
 self.Frame.configure(highlightcolor="black")  
  
 self.Text = tk.Label(self.Frame)  
 self.Text.place(x=10, y=10, height=90, width=460)  
 self.Text.configure(font=("Arial", 15))  
 self.Text.configure(background="#FFFFFF")  
 self.Text.configure(foreground="#000000")  
 self.Text.configure(highlightbackground="#d9d9d9")  
 self.Text.configure(highlightcolor="black")  
 self.Text.configure(text=text)  
 self.Text.configure(wraplength=450)  
  
 self.root.grab\_set()  
 self.Text.focus\_set()  
 self.root.wait\_window()

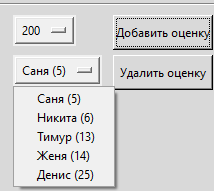
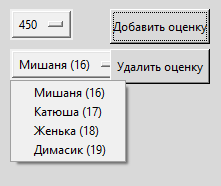
1. **Переключатели**

Курсора окна не всегда достаточно, чтобы однозначно передавать данные, необходимые для функций, выполняющих SQL-запросы. По этой причине в некоторых окнах реализованы состояния, сохраняющие данные в переменной состояния. Функции SQL-запросов берут необходимые им данные из курсора и переменных состояния. Состояния могут быть изменены через объекты переключателей.

Пример переключателей из учительского меню

Переключатели каскадно меняют состояния выбранной группы, студента и предмета. Например, переключатель состояний студентов зависит от состояния группы.



Код переключателей, который содержится в классе учительского окна

OptionMenu – вызывает переключатель, который может сам подстраиваться под размеры и длину текстовых строк

Value\_inside – переменная состояния, которая используется при вызове функций SQL-запросов.

group\_list = groups\_get\_list()  
self.value\_inside\_group = tk.StringVar(self.root)  
self.value\_inside\_group.set(group\_list[0])  
self.GroupList = tk.OptionMenu(self.Frame, self.value\_inside\_group, \*group\_list, command=self.GroupChanged)  
self.GroupList.place(x=700, y=10)  
  
student\_list = group\_special\_string\_get(self.value\_inside\_group.get())  
self.value\_inside\_student = tk.StringVar(self.root)  
self.value\_inside\_student.set(student\_list[0])  
self.StudentList = tk.OptionMenu(self.Frame, self.value\_inside\_student, \*student\_list,  
 command=self.StudentChanged)  
self.StudentList.place(x=700, y=50)  
  
subject\_list = subjects\_get\_list()  
print(subject\_list)  
self.value\_inside\_subject = tk.StringVar(self.root)  
if subject\_list:  
 self.value\_inside\_subject.set(subject\_list[0])  
else:  
 self.value\_inside\_subject.set("")

Пример функции, которая принимает на вход переменную состояния

def GradesCreate(self):  
 edit\_dialog1 = EditDialog(self, "Оценка", "")  
 edit\_dialog1.show()  
  
 edit\_dialog2 = EditDialog(self, "Дата", "")  
 edit\_dialog2.show()  
  
 if edit\_dialog1.processed:  
 if edit\_dialog2.processed:  
 text = grade\_add(self.scissors(self.value\_inside\_student.get()), self.value\_inside\_subject.get(),  
 edit\_dialog1.new\_parameter, edit\_dialog2.new\_parameter)  
 print("Вывод: ", text)  
 if text:  
 WarningWindow(self, text)  
 self.LoadList("")

**заключение**

В ходе работы была разработана и развернута база данных и приложение на Python с пользовательским интерфейсом. Были использованы библиотеки, позволяющие программам на Python осуществлять SQL-запросы на развернутой базе данных.

Использование приложения позволяет пользователю комфортно взаимодействовать с ведомостью студентов и их распределением по группам. Помимо функций создания, редактирования и удаления, пользователь может с удобством просматривать имеющиеся данные.

При написании данной работы были получен опыт в разработке реляционных баз данных на языке запросов SQL, были получены навыки объектно-ориентированного программирования и навыки создания и работы с пользовательским интерфейсом. Для работы использовалось несколько разных приложений – среда для программирования и системы управления базами данных. Было использованы две невстроенных библиотеки python, информация о работе с которыми была получена из интернет-документации.

**список использованных источников**

1. Техническая документация библиотеки Tkinter.  
   URL: https://docs.python.org/3/library/tkinter.html (дата обращения: 21.04.2023).
2. Техническая документация библиотеки Psycopg.  
   URL: https://www.psycopg.org/docs/ (дата обращения: 21.04.2023)
3. Техническая документация по использованию СУБД DBeaver  
   URL: https://dbeaver.com/docs/wiki/ (дата обращения: 21.04.2023)
4. Техническая документация по использованию СУБД PgAdmin  
   URL: https://www.pgadmin.org/docs/