**3. ЭТАП 3 НИР**

**3.1. Цель этапа**

Цель этой части научно-исследовательской работы – получение первичного опыта создания программы для работы с базой данных с использованием среды Python. В результате этого этапа должна получиться база данных, обладающая следующей структурой:

|  |  |
| --- | --- |
| Содержание поля | Тип информации в поле |
| Код дисциплины по учебному плану | Текст |
| Название дисциплины | Текст |
| Номер семестра с аттестацией по дисциплине | Целое число |
| Тип аттестации (экзамен/зачет) | Текст |
| Дата аттестации | Дата |
| ФИО преподавателя, проводившего аттестацию | Текст |
| Должность преподавателя | Текст |
| Полученная оценка | Целое число |
| Дата занесения/обновления записи | Дата |

**3.2. Содержание этапа**

Эта часть содержит выполнение следующих операций:

1. Создать БД, содержащую одну таблицу с заданной структурой. Имена БД, таблицы и полей произвольные. Операция выполняется в консоли и в файл со скриптом не заносится.
2. Разработать программу, обеспечивающую открытие этой БД, отображение списка имён полей таблицы и выполнение следующих функций по выбору пользователя:

* Отображение текущего содержимого БД в виде таблицы
* Сохранение текущего содержимого таблицы БД в текстовый файл с задаваемым именем
* Выбор пользователем имени поля и задание условия по значениям этого поля (логическое выражение). Отображение в виде таблицы подмножества строк, удовлетворяющих этому условию
* Выбор операции с подмножеством строк: удаление из БД, замена значений во всех строках в указанном поле на заданное значение
* Добавление новой строки с заданными значениями полей в таблицу БД
* Завершение работы с программой.

**3.2.1. Создание таблицы в базе данных**

Эти действия должны быть выполнены однократно, поэтому они выполнены через терминал IDLE Shell. Первым делом, перейдем в целевую директорию:

>>> import os

>>> os.chdir("d:/STUDY/NIR")

>>> import sqlite3

Так как файла с таким именем сейчас нет, то при выполнении этой команды он будет создан:

>>> con = sqlite3.connect("vuz3.sqlite")

>>> cur = con.cursor()

Далее необходимо инициализировать создание таблицы, поля полностью соответствуют таблице в пункте 3.1.

>>> cur.execute("""

CREATE TABLE attestation (

sub\_code TEXT,

sub\_name TEXT,

semester INTEGER,

att\_type TEXT,

att\_date DATE,

prof\_name TEXT,

prof\_status TEXT,

mark INTEGER,

edit\_date DATE

)

""")

<sqlite3.Cursor object at 0x000002415F76EB90>

*Примечание: используемые сокращения – sub (subject), att (attestation), prof (professor).*

Изменения обязательно нужно закоммитить, после чего закрыть курсор и подключение к БД.

>>> con.commit()

>>> cur.close()

>>> con.close()

**3.2.2. Описание структуры проекта**

NIR/

├── main.py

├── query/

│ ├── db\_show.py

│ ├── db\_edit.py

│ └── db\_save.py

└── input/

└── data\_input.py

Коренная папка называется /NIR. В ней есть две папки-модуля и main.py – точка входа в проект.

Папка query («запрос») содержит три исполнительных файла:

* db\_show.py содержит функции, которые отображают данные из таблицы, но не изменяют её.
* db\_edit.py содержит функции, которые изменяют, удаляют и добавляют данные в таблицу.
* db\_save.py содержит функцию, которая сохраняет таблицу в файл.

Папка input («ввод») содержит файл data\_input.py, где есть две вспомогательные функции для работы db\_add(). Эти функции осуществляют ввод с клавиатуры и из файла.

main.py содержит функцию menu(), выводящую опции, доступные для выбора. Далее описываются действия и вызываются функции для каждого выбора.

**3.2.3. Описание функциональных участков программы**

1. Файл db\_show.py

import sqlite3  
  
def show\_names():  
 *"""Функция возвращает имена полей таблицы"""* with sqlite3.connect("D:/STUDY/NIR/vuz3.sqlite") as con:

*"""Здесь и далее используется with, т.к. он закрывает подключение и освобождает ресурсы автоматически"""*  
 cur = con.cursor()  
 cur.execute("PRAGMA table\_info(attestation)")

*#команда PRAGMA используется для демонстрации различных параметров и настроек и управления ими*  
 output = cur.fetchall()  
 cur.execute("SELECT COUNT(\*) FROM attestation")

*#подсчёт количества строк в таблице*  
 count = cur.fetchone()[0] # Используем fetchone и извлекаем только число  
  
 headers = [f"{n[1]} ({n[2]})" for n in output]

*#n[1] – имя, n[2] – тип. List comprehension для вывода*  
 format\_headers = " | ".join(headers)  
*#Разделитель для столбцов*  
 print("Имена и типы полей таблицы attestation: \n", format\_headers)  
 print("-" \* len(format\_headers))  
  
 print(f"Количество записей в таблице: {count}")

def table\_show():  
 *"""Функция возвращает всё содержимое таблицы, форматировав внешний вид с помощью символов ASCII, сверху подписаны поля"""* with sqlite3.connect("D:/STUDY/NIR/vuz3.sqlite") as con:   
 cur = con.cursor()  
 cur.execute("PRAGMA table\_info(attestation)")  
 fields = cur.fetchall()  
  
 if not fields:  
 print("Таблица 'attestation' не существует.")  
 return  
  
 names = [now[1] for now in fields] *#Выделение имён полей*  
 cur.execute("SELECT \* FROM attestation")  
 rows = cur.fetchall()  
 if not rows: *#Пустой список имеет логическое значение False*  
 print("Таблица пуста!")  
 return  
 widths = [max(len((name)), max(len(str(row[now])) for row in rows)) for now, name in enumerate(names)]

*“““для каждого заголовка колонки (name), этот генератор вычисляет максимальную длину между длиной самого заголовка колонки len(name)и максимальной длиной значений в текущей колонке для всех строк max(len(str(row[now])) for row in rows).“““*

header = " | ".join([f"{name:<{widths[now]}}" for now, name in enumerate(names)])  
 seps = "-+-".join(['-' \* now for now in widths])

*“““Строка создаёт заголовок таблицы, в котором имена колонок (names) выравниваются по ширине, определённой ранее в списке widths. Цель — отформатировать каждое имя колонки так, чтобы оно занимало определённую ширину, и затем объединить их в строку с разделителем   
" | " (вертикальная черта с пробелами). “““*  
  
 print(header)  
 print(seps)  
  
 for now in rows:  
 print(" | ".join([f"{str(value):<{widths[i]}}" for i, value in enumerate(now)]))

*# Форматированный вывод каждой строки таблицы*

def db\_filter(field:str, cond:str):  
 *"""Функция возвращает поля, соответствующие условию.  
 Если условие введено некорректно и не выполнилось, возвращается ошибка"""* with sqlite3.connect("D:/STUDY/NIR/vuz3.sqlite") as con:  
 cur = con.cursor()  
 try:  
 cur.execute(f"SELECT \* FROM attestation WHERE {field} {cond}")  
 rows = cur.fetchall()  
 if rows:  
 print("Строки, удовлетворяющие условию:")  
 [print(now) for now in rows]  
 else:  
 print("Строки не найдены")  
 except Exception as err: *# Обрабатывает все ошибки, в том числе несуществующие поля или некорректные условия*  
 print(f"Произошла ошибка: {err}")

2. Файл db\_save.py

import sqlite3  
def db\_saving(file):

*"""Функция сохраняет все строки в файл без форматирования"""*  
 with sqlite3.connect("D:/STUDY/NIR/vuz3.sqlite") as con:  
 try:  
 cur = con.cursor()  
 cur.execute("SELECT \* FROM attestation")  
 with open(file, 'w') as f:  
 [f.write(str(now) + '\n') for now in cur.fetchall()] *#Построчный ввод содержимого курсора после SELECT*  
 print(f"Таблица сохранена в файле {file}")  
 except Exception: *#Если произошла ошибка, она бросается и обрабатывается в main.py*  
 print(f"Произошла ошибка: {err}")

3. Файл db\_edit.py

import sqlite3  
def db\_delete(field:str, cond:str): *#Риск ошибки уменьшается за счёт статической типизации параметров*

*"""Функция удаляет из БД записи, где одно из заданных полей  
соответствует заданному условию"""*

with sqlite3.connect("D:/STUDY/NIR/vuz3.sqlite") as con:  
 try:  
 cur = con.cursor()  
 cur.execute(f"DELETE FROM attestation WHERE {field} {cond}")  
 if cur.rowcount == 0: *# Атрибут cur.rowcount показывает, сколько строк сейчас в курсоре*  
 print("Строки не найдены")  
 else:  
 confirm = input("Подтвердить изменения? [y/n]: ")  
 if confirm == "y":  
 con.commit()  
 print(f"Удалено строк: {cur.rowcount}")  
 elif confirm == "n":  
 con.rollback()  
 return  
 else:  
 print("Не понимаю. Повторите. [y/n]: ")  
*# Здесь и далее все изменения дополнительно подтверждаются и могут быть отменены.*  
 except Exception as err:  
 print(f"Произошла ошибка: {err}")

def db\_update\_field(field:str, cond:str, new):

*"""Функция изменяет в таблице ячейку, в определенном поле, которое соответствует  
заданному условию"""*

with sqlite3.connect("D:/STUDY/NIR/vuz3.sqlite") as con:  
 try:  
 cur = con.cursor()  
 cur.execute((f"UPDATE attestation SET {field} = ? WHERE {cond}"), (new,)) *#В метод execute() передается сразу и запрос, и новое значение. Оно не строго типизировано.*  
 if cur.rowcount == 0:  
 print("Строки не найдены")  
 else:  
 confirm = input("Подтвердить изменения? [y/n]: ")  
 if confirm == "y":  
 con.commit()

print(f"Значения поля {field}, соответствующие условию {cond}, изменены на {new}")

*#Выводится строка, подытоживающая коммит*  
 elif confirm == "n":  
 con.rollback()  
 return  
 else:  
 print("Не понимаю. Повторите. [y/n]: ")  
 except Exception as err:  
 print(f"Произошла ошибка: {err}")

def db\_add(values):

*"""Функция добавляет запись в таблицу"""*  
 with sqlite3.connect("D:/STUDY/NIR/vuz3.sqlite") as con:  
 try:  
 cur = con.cursor()  
 query = '''INSERT INTO attestation   
 (sub\_code, sub\_name, semester, att\_type, att\_date, prof\_name, prof\_status, mark, edit\_date)VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)'''  
 cur.execute(query, values)  
*"""Для удобства запрос записан в отдельную переменную. Ввод осуществляется в main.py, причем первые 8 с клавиатуры, а последний - автоматически"""*  
 confirm = input("Подтвердить изменения? [y/n]: ")  
 if confirm == "y":  
 con.commit()  
 print("В таблицу добавлена строка")  
 return  
 elif confirm == "n":  
 con.rollback()  
 return  
 else:  
 print("Не понимаю. Повторите. [y/n]: ")  
 except Exception as err:  
 print(f"Произошла ошибка: {err}")

4. Файл main.py

import query.db\_edit as e *#импорт всех кастомных модулей*  
import query.db\_show as sh  
import query.db\_save as s  
from input import data\_input as i  
  
def menu():  
 print("""--- ВЫБЕРИТЕ ОПЦИЮ ---  
[f] Показать список полей таблицы *#f - field*[a] Показать всю таблицу *#a - all*  
[s] Сохранить содержимое таблицы в файл *#s - save*  
[w] Фильтр по полю *#w - where*[d] Удалить данные по условию *#d - delete*  
[u] Изменить содержимое по условию *#u - update*[i] Добавить новую строкy *#i - insert*  
[q] Выйти""") *#q - quit*  
 return input("Выберите действие: ")  
  
while True:  
 option = menu() *# Меню показывается всегда после выполнения опции, и останавливает ее только [q]*  
  
 if option.lower() == "f":  
 sh.show\_names()  
  
 elif option.lower() == "a":  
 sh.table\_show()  
  
 elif option.lower() == "s":  
 try:  
 file = input("Введите имя файла для сохранения: ")  
 s.db\_saving(file)  
 except Exception as e:  
 print (f"Ошибка: {e}")  
  
 elif option.lower() == "w":  
 while True:  
 try:  
 field = input("Введите имя поля, к которому применяется условие: ")  
 cond = input("Введите условие фильтрации: ")  
 sh.db\_filter(field, cond)  
 break  
 except Exception as err:  
 print (f"Возникла ошибка: {err}. Повторите ввод:")  
  
 elif option.lower() == "d":  
 while True:  
 try:  
 field = input("Введите имя поля, к которому применяется условие: ")  
 cond = input("Введите условие фильтрации: ")  
 e.db\_delete(field, cond)  
 break  
 except Exception as err:  
 print(f"Возникла ошибка: {err}. Повторите ввод:")  
  
 elif option.lower() == "u":  
 while True:  
 try:  
 field = input("Введите имя поля, к которому применяется условие: ")  
 cond = input("Введите условие фильтрации: ")  
 new = input("Введите новое значение: ")  
 break  
 except Exception as err:  
 print(f"Возникла ошибка: {err}. Повторите ввод:")  
  
  
 elif option.lower() == "i":  
 while True:  
 try:  
*“““Есть два варианта ввода: с клавиатуры по одной строке и из файла сразу много строк, но есть нюанс: в файле слова, относящиеся к одному полю, записываются через нижнее подчеркивание. Например, «sub\_name: «Методы\_оптимизации» вместо «Методы оптимизации»». “““*  
 print("[h] - Построчный ввод с подсказками (подписан тип и имя поля)") *#h - hint* print("[f] - Ввод из файла") *#f - file*  
 type = input("Выберите тип ввода: ")  
  
 if type.lower() == "h":  
 vals = i.hints\_input()  
 e.db\_add(vals)  
  
 elif type.lower() == "f":  
 try:  
 file = input("Введите имя файла: ")  
 custom\_lines = input("Хотите указать количество строк? \nПри отказе считаются все строки [y/n]: ")  
 if custom\_lines.lower() == "y":  
 lines\_to\_read = input("Введите количество строк для считывания: ")  
 else:  
 count = sum(1 for \_ in open(file, "r", encoding="utf-8")) *#Подсчет количества строк в файле с помощью list comprehension*  
 lines\_to\_read = count  
 i.file\_input(file, lines\_to\_read)  
 break  
  
 except Exception as err:  
 print(f"Ошибка: {err}.")  
  
 except Exception as err:  
 print(f"Возникла ошибка: {err}. Повторите ввод:")  
  
 elif option.lower() == "q":  
 print("Сеанс работы завершен")  
 break

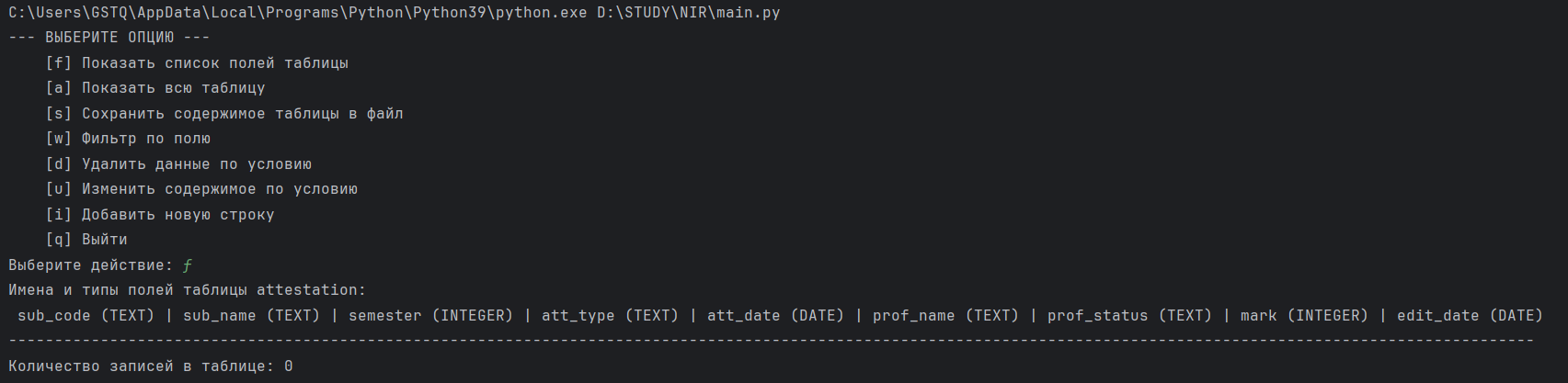
else:  
 print ("Команда не распознана. Повторите ввод.")

5. Файл data\_input.py

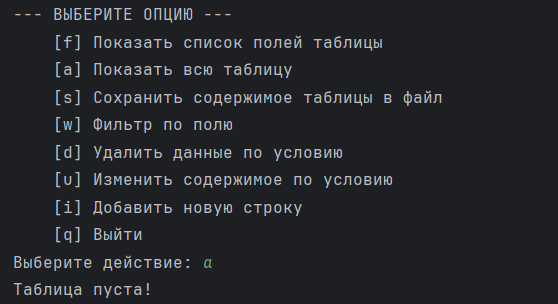
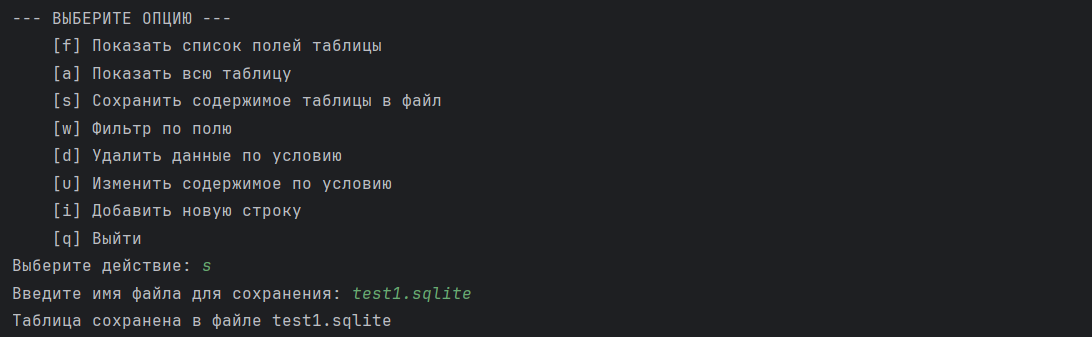
from datetime import datetime  
import query.db\_edit as e

def hints\_input():  
 vals = [0] \* 9  
 vals[0] = input("Код дисциплины по учебному плану [текст]: ")  
 vals[1] = input("Название дисциплины [текст]: ")  
 vals[2] = int(input("Номер семестра с аттестацией по дисциплине [целое число]: "))  
 vals[3] = input("Тип аттестации (экзамен/зачет) [текст]: ")  
 vals[4] = input("Дата аттестации в формате DD.MM.YYYY [текст]: ")  
 vals[5] = input("ФИО преподавателя, проводившего аттестацию [текст]: ")  
 select\_status = ["ассистент", "преподаватель", "старший преподаватель", "доцент", "профессор",  
 "заведующий кафедрой"]  
 status\_str = "\n".join([f"\* {i}" for i in select\_status]) *#вывод подсказки с должностями в столбик, каждая строка начинается с звёздочки*  
 vals[6] = input(f"Выберите должность: \n{status\_str}\n Должность преподавателя [строка]: ")  
 vals[7] = int(input("Полученная оценка [целое число]: "))  
 vals[8] = datetime.today().strftime('%d.%m.%Y') *#сегодняшняя дата*  
 return vals  
  
def file\_input(file, lines\_to\_read):  
 with open(file, 'r', encoding='utf-8') as f:  
 try:  
 if lines\_to\_read == sum(1 for \_ in f):  
 f.seek(0)  
 lines = f.readlines()  
 else:  
 *# Преобразуем количество строк в число и читаем соответствующее количество строк* lines = [next(f) for \_ in range(int(lines\_to\_read))]  
  
 for line in lines:  
 *# Разбиваем строку на элементы по пробелам и очищаем от лишних символов (например, \n)* data = line.strip().split()  
 for i in range(len(data)):  
 if "\_" in data[i]:  
 data[i] = data[i].replace("\_", " ")  
  
 if len(data) == 8: *# Убедимся, что в строке ровно 8 элементов* data.append(datetime.today().strftime('%d.%m.%Y'))  
 e.db\_add(data)  
 else:  
 print(f"Ошибка: строка не содержит 8 элементов. Пропускаем строку: {line.strip()}")  
 except Exception: print(f"Произошла ошибка: {err}")  
  
def confirm\_changes(con):  
 confirm = input("Подтвердить изменения? [y/n]: ")  
 if confirm == "y":  
 con.commit()  
 print("В таблицу добавлена строка")  
 return  
 elif confirm == "n":  
 con.rollback()  
 return  
 else:  
 print("Не понимаю. Повторите. [y/n]: ")

**3.2.4. Тестирование.**

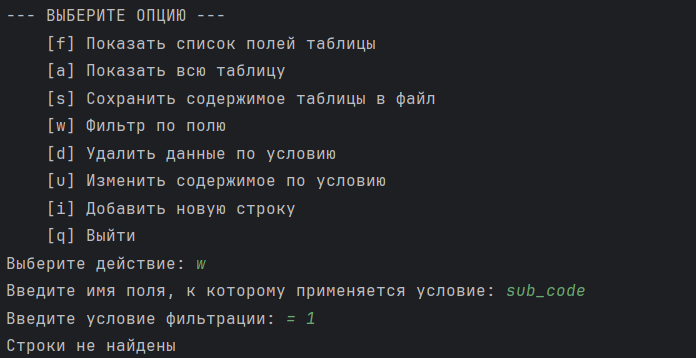
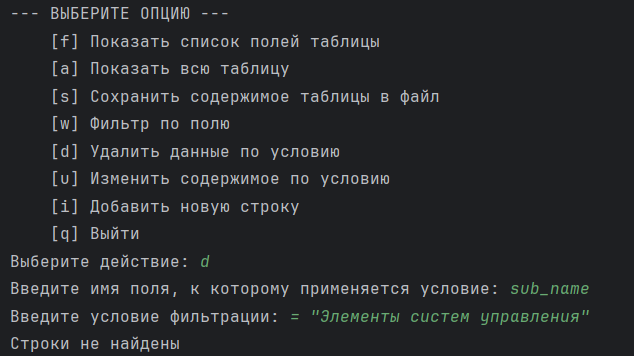
1. Тестирование на пустой базе данных.

*Рис.1. Тестирование функции show\_names()*



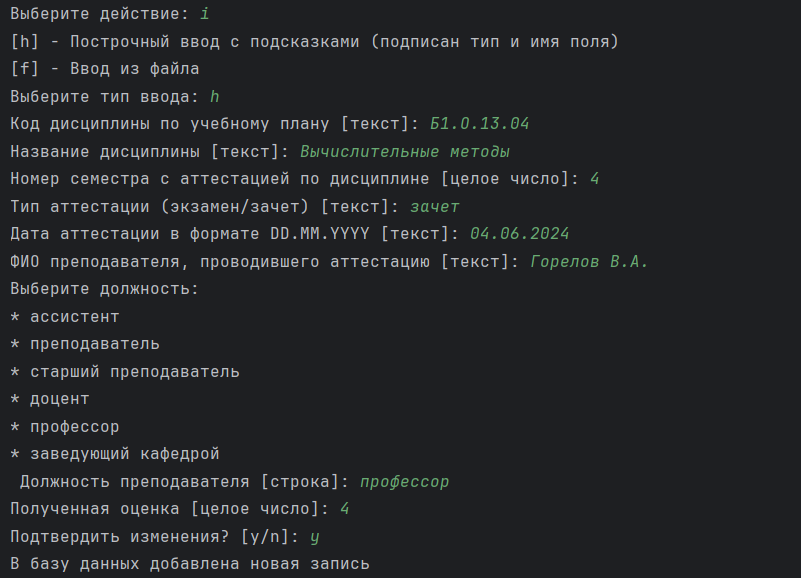
*Рис.3. Тестирование функции db\_saving()*

*Рис.2. Тестирование функции table\_show()*

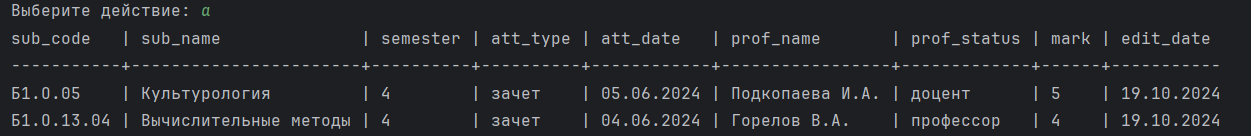


*Рис.5. Тестирование функции db\_delete()*

*Рис.4. Тестирование функции db\_filter()*

****

*Рис.6. Тестирование функции db\_add()*

****

*Рис.7. Проверка результата функции db\_add()*