МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

ХАБАРОВСКИЙ ИНСТИТУТ ИНФОКОММУНИКАЦИЙ (ФИЛИАЛ)

СРЕДНЕЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

КУРСОВОЙ ПРОЕКТ

по МДК 01.02 Прикладное программирование

на тему

Программа формирования списков «Библиотека»

Выполнил: А.В. Медведев

Проверил: А.Н. Поляков

Группа: ПКС-320

2022

СОДЕРЖАНИЕ

|  |  |
| --- | --- |
| Введение | 4 |
| 1 Постановка задачи | 6 |
| 1.1 Описание задачи | 6 |
| 1.2 Описание состава входной и выходной информации | 6 |
| 1.3 Задачи курсового проекта | 7 |
| 1.4 Ограничения на входные данные и методы их контроля | 7 |
| 2 Функциональная схема программы | 9 |
| 3 Блок-схема программы | 10 |
| 4 Описание принципа работы программы | 11 |
| 5 Описание применяемых компонентов, их свойств, событий и методов | 18 |
| 6 Тестовый пример | 19 |
| 7 Листинг программы | 22 |
| Заключение | 33 |
| Список использованных источников | 34 |

ВВЕДЕНИЕ

Современные крупные предприятия ежедневно работают с данными. Данные — поддающееся многократной интерпретации представление информации в формализованном виде, пригодном для передачи, связи или обработки. Информация о сотрудниках корпорации, продуктах гипермаркета, медицинском центре может быть организована в базы данных. Обычно это делается для обеспечения одновременного доступа к данным множества пользователей, нередко расположенных достаточно далеко друг от друга. В них тем или иным путем решаются специфические проблемы параллельных процессов, целостности (правильности) и безопасности данных. Целью данной курсовой работы является разработка программы формирования списков библиотеки, которая должна решать задачу регистрации, обработки и хранения информации о читателях. База данных для читателей должна содержать всю необходимую личную информацию, а также адрес проживания.

Курсовой проект будет выполняться в высокоуровневом языке программирования Python при помощи интегрированной среды разработки PyCharm.

Чтобы перейти к постановке задачи проекта следует дать определение использованной парадигмы программирования и основных её концепций.

Объектно-ориентированное программирование (ООП) — это подход, при котором программа рассматривается как набор объектов, взаимодействующих друг с другом. Объектно-ориентированное программирование используется, с целью:

– структурировать информацию и не допускать путаницы;

– точно определять взаимодействие одних элементов с другими;

– повышать управляемость программы;

– быстрее масштабировать код под различные задачи;

– лучше понимать написанное;

– эффективнее поддерживать готовые программы;

– внедрять изменения без необходимости переписывать весь код.

Объектно-ориентированное программирование основано на принципах:

– абстракции – способ выделить набор наиболее важных атрибутов и методов и исключить незначимые;

– инкапсуляции – объединение функций и данных в рамках одной структуры, внутреннее состояние которой (данные) скрыто от внешнего мира;

– наследование – свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью.

– полиморфизм – свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

1 ПОСТАНОВКА ЗАДАЧИ

1.1 Описание задачи

Программа формирования списков “Библиотека” обрабатывает исходные данные и выводит требуемые списки по запросу пользователя. Программа исполняется в консоли операционной системы. Данные о читателях библиотеки хранятся в базе данных. Работа с программой осуществляется посредством взаимодействия пользователя с базой данных при помощи языка структурированных запросов. На выбор пользователю даётся набор функций, для взаимодействия с программой:

– добавить запись;

– удалить запись;

– обновить запись;

– вывести списки;

– помощь;

– экспорт базы данных в формат JSON.

Выходная информация о читателях сортируется по номеру читательского билета.

1.2 Описание состава входной и выходной информации

Входной информацией являются сведения о читателях: номер читательского билета, фамилия и инициалы читателя, год рождения читателя, возраст, образование (высшее, среднее, школьное), пол, адрес.

Выходной информацией является список, сформированный по требованию пользователя:

— по заданному пользователем диапазону возраста и образованию;

— по полу;

— по образованию, отсортированном по году рождения читателя;

— по заданному пользователем диапазону номера читательского билета

— по заданной пользователем букве фамилии, отсортированной по возрасту.

1.3 Задачи курсового проекта

Задачей курсового проекта является формирование программы для работы с данными о читателях библиотеки. Необходимо:

– разработать черновой прототип программы;

– составить базу данных читателей и определить необходимые поля для описания их;

– разработать программу для взаимодействия библиотекаря с базой данных читателей;

– составить и устранить список возможных ошибок, связанных с работой в программе;

1.4 Ограничения на входные данные и методы их контроля

Пользователь вводит данные в соответствии с их типом данных. К целочисленному типу данных относится номер читательского билета, год рождения и возраст читателя. К строковому типу данных относится фамилия и инициалы, адрес проживания и уровень образования читателя.

В программе предусмотрены проверки на соответствие вводимых данных их типу, а также на область допустимых значений для данного поля.

def surname\_np(self, value):

while True:

try:

k = input("Введите фамилию, имя и отчество\t")

except (TypeError, ValueError) as err:

print("Введите допустимую запись ФИО")

else:

try:

surname, name, patronymic = k.split()

if surname.isdigit() or name.isdigit() or patronymic.isdigit():

print("Введите правильное ФИО\n\tНапример: Бузова Ольга Петровна")

else:

lower = surname.lower()

self.\_\_surname\_np = f"{lower.replace(lower[0], lower[0].upper(), 1)} {name[0].upper()}.{patronymic[0].upper()}.".format( \*\*vars())

break

except (TypeError, ValueError) as err:

print("Введите правильное ФИО\n\tНапример: Бузова Ольга Петровна")

2 ФУНКЦИОНАЛЬНАЯ СХЕМА ПРОГРАММЫ

На рисунке 2.1 представлена функциональная схема программы.

 Рисунок 2.1 - Функциональная схема программы

3 БЛОК-СХЕМА ПРОГРАММЫ

На рисунке 3.1 представлена блок-схема программы.



Рисунок 3.1 – Блок-схема программы

4 ОПИСАНИЕ ПРИНЦИПА РАБОТЫ ПРОГРАММЫ

Запуская программу нас встречает приветственное окно с готовым списком читателей приведённом на рисунке 4.1.

На выбор пользователю даётся 7 функций для работы с программой приведённых на рисунке 4.2.

При выборе функции добавления записи, пользователь может заполнить запись полностью для одного читателя, или заполнить данные для одной из таблиц. Реализация данной функции приведена на рисунках 4.3–4.5.

При выборе функции удаления записи, пользователь может удалить запись из всех таблиц в зависимости от введённого номера читательского билета. Реализация данной функции приведена на рисунке 4.6.

При выборе функции обновления записи, пользователь может обновить данные одной из таблиц. Реализация данной функции приведена на рисунке 4.7.

При выборе функции вывода списков, пользователь может вывести данные главной таблицы, которые будут отсортированы в зависимости от выборки пользователя. Реализация данной функции приведена на рисунках 4.8, 4.9.

Пользователь может экспортировать базу данных в формат JSON, для этого ему нужно выбрать функцию, приведённую на рисунке 4.10.

Если пользователь нуждается в помощи для работы с программой, то он может выбрать функцию, приведённую на рисунке 4.11.

Для выхода из программы пользователь может выбрать функцию, приведённую на рисунке 4.12.

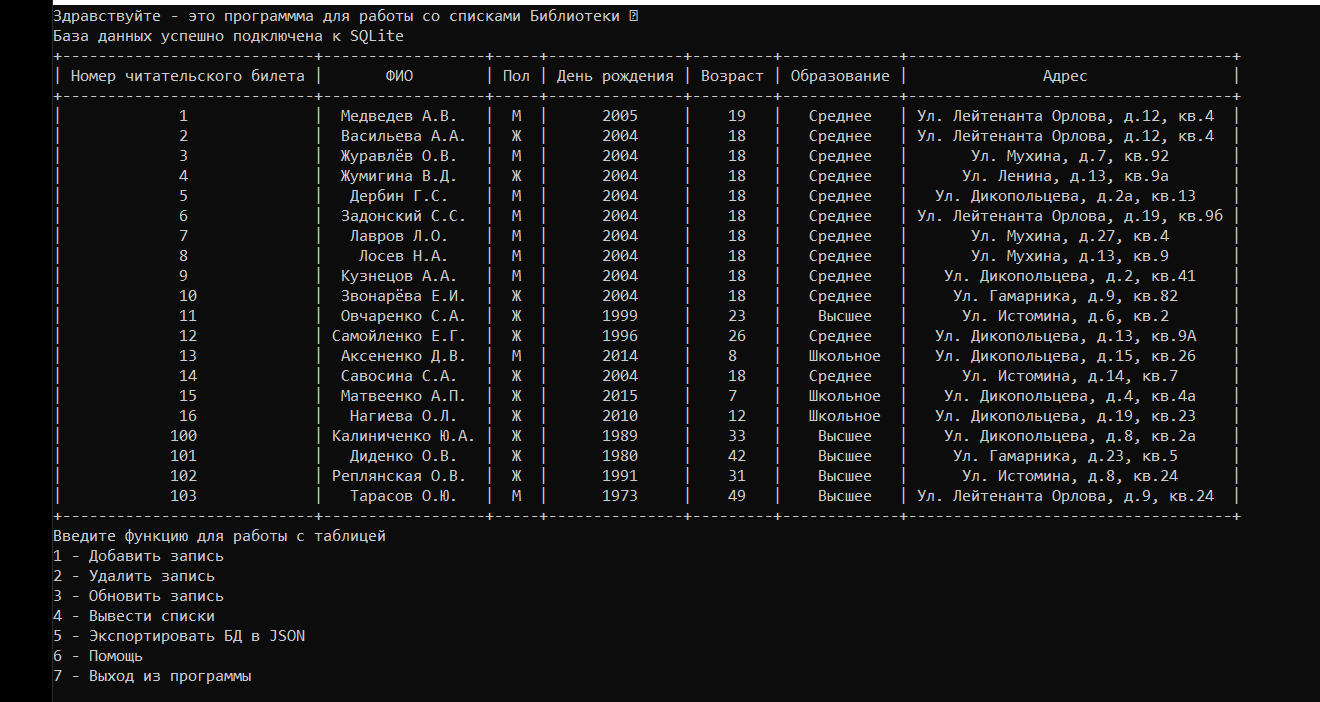


Рисунок 4.1 – Приветственное окно программы

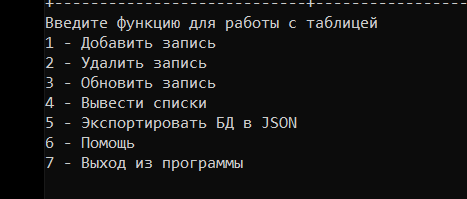


Рисунок 4.2 – Выбор функции

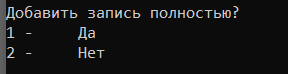


Рисунок 4.3 – Выбор функции добавления записи во все таблицы

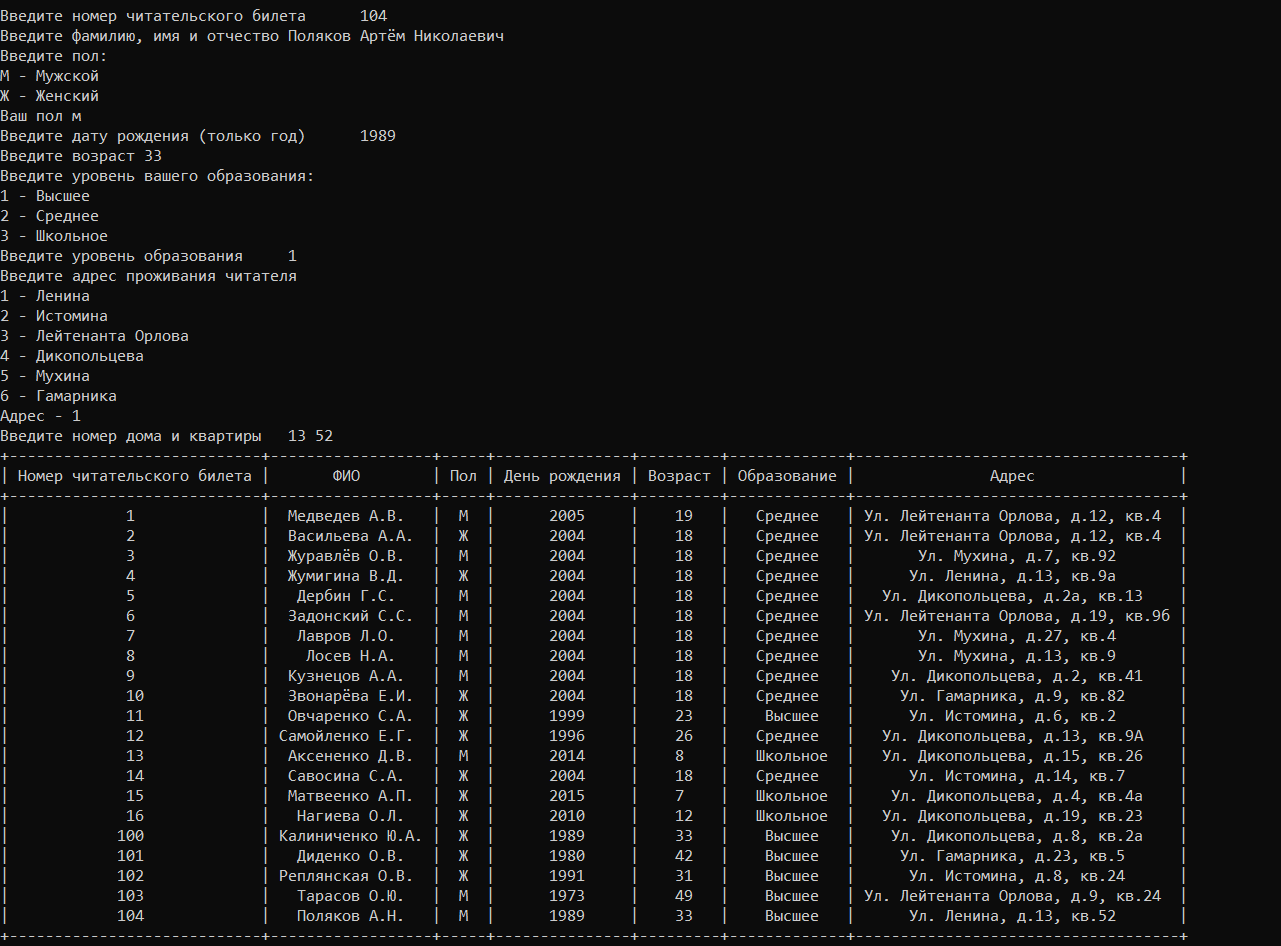


Рисунок 4.4 – Вывод главной таблицы с новой записью

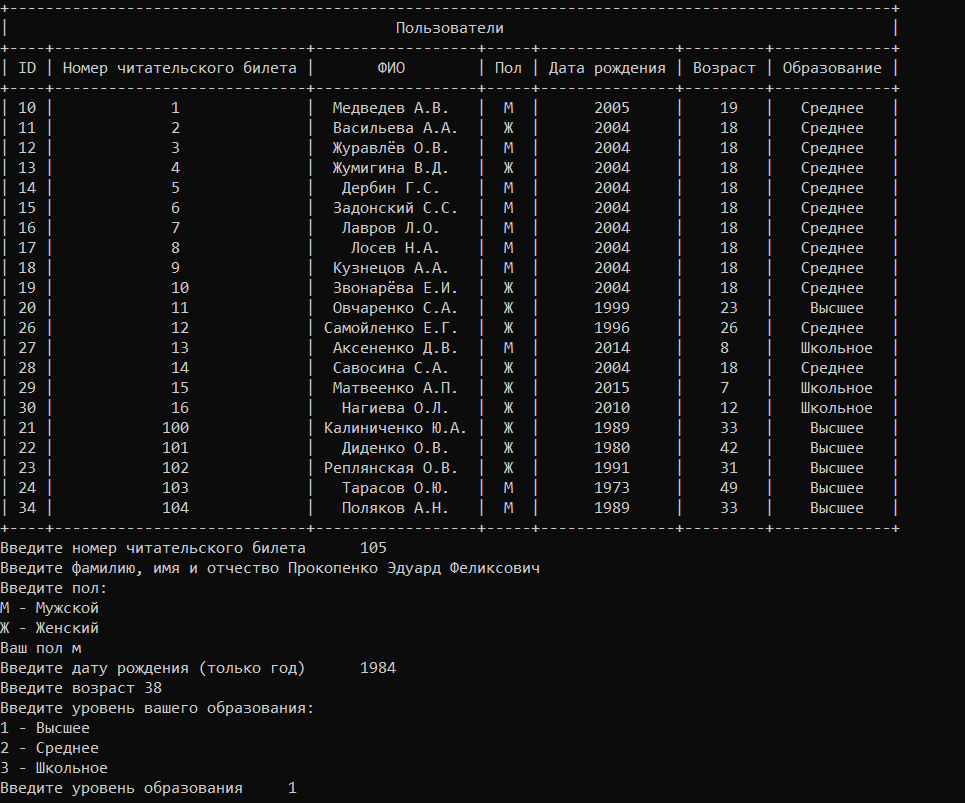


Рисунок 4.5 – Пример заполнения для неполной записи

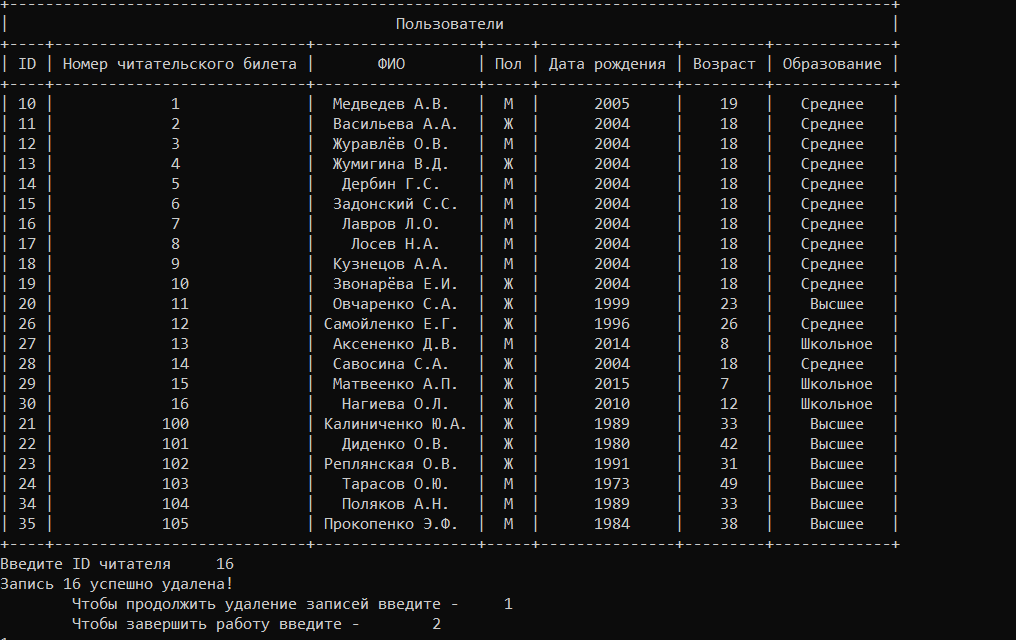


Рисунок 4.6 – Удаление читателя из базы данных

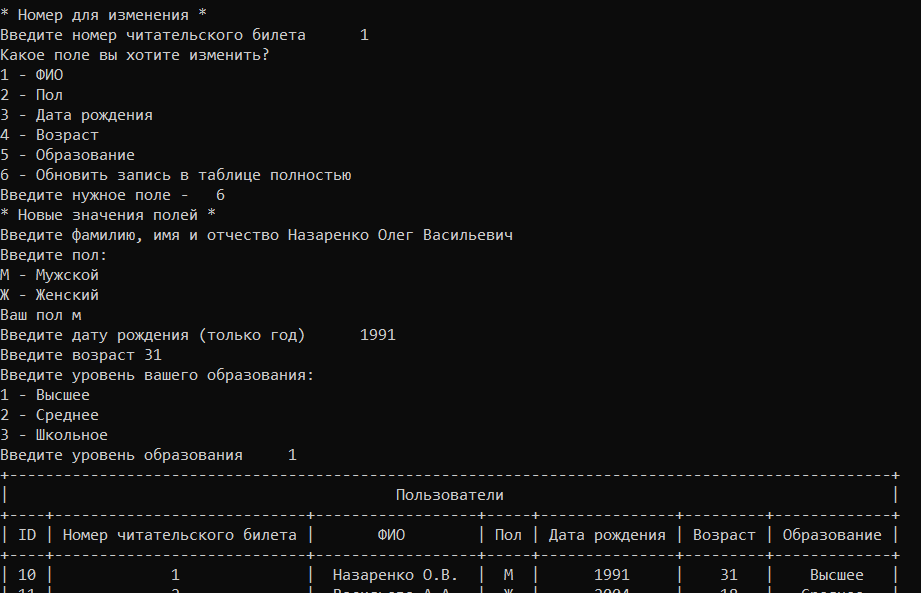


Рисунок 4.7 – Обновление данных о читателе

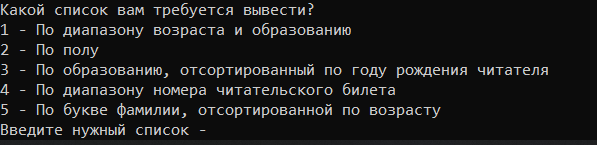


Рисунок 4.8 – Выбор сортировки списков

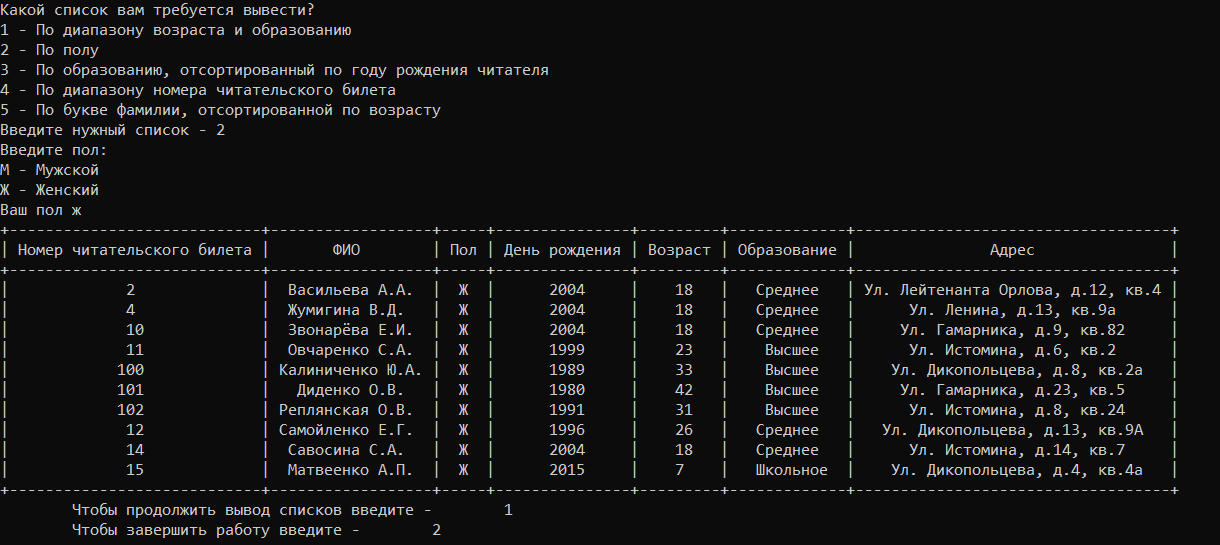


Рисунок 4.9 – Пример вывода списка отсортированного по полу

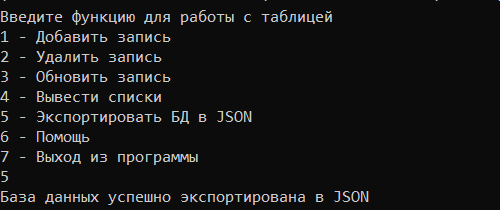


Рисунок 4.10 – Экспорт базы данных в формат JSON

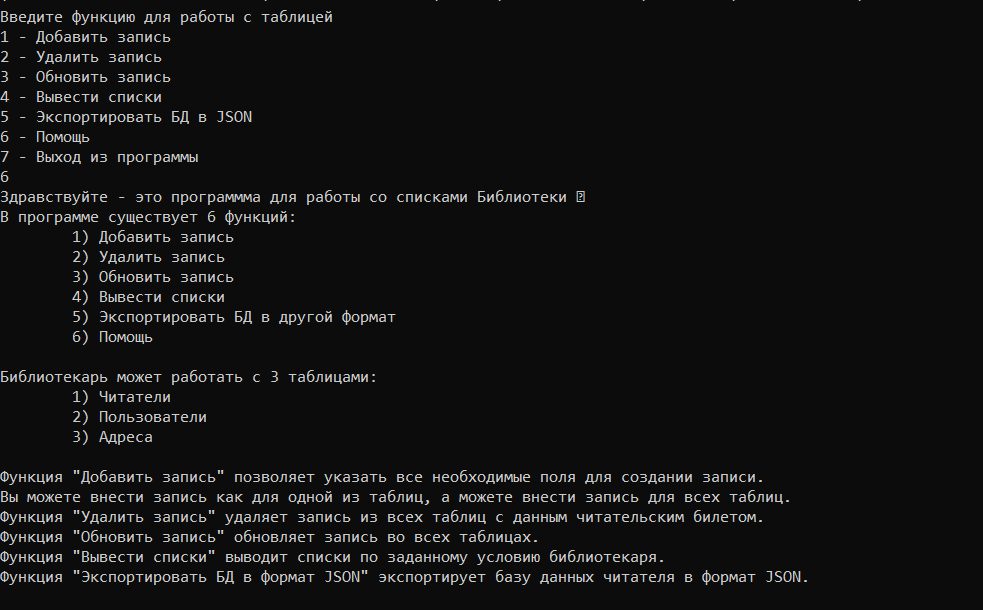


Рисунок 4.11 – Окно помощи для работы с программой

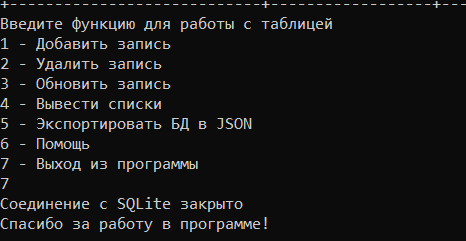


Рисунок 4.12 – Выход из программы

5 ОПИСАНИЕ ПРИМЕНЯЕМЫХ КОМПОНЕНТОВ, ИХ СВОЙСТВ, СОБЫТИЙ И МЕТОДОВ

В курсовом проекте использовались библиотеки:

– time – модуль для работы со временем;

– sqlite3 – файловая база данных SQL, которая поставляется в комплекте с Python и может использоваться в приложениях Python;

– json – модуль для работы с форматом обмена данными;

– prettytable – библиотека с классом PrettyTable, позволяющим выводить данные в виде таблицы.

В проекте был создан класс читателей, определяющий поля и методы для взаимодействия с базой данных.

class LibraryReaders:

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

# Номер читательского билета

self.\_\_lib\_card\_num = None

# Фамилия и инициалы читателя

self.\_\_surname\_np = None

# Дата рождения читателя

self.\_\_birthday\_date = None

# Возраст читателя

self.\_\_age = None

# Пол читателя

self.\_\_gender = None

# Адрес читателя

self.\_\_address = None

# Уровень образования

self.\_\_education = None

6 ТЕСТОВЫЙ ПРИМЕР

Пользователь открывает программу для формирования списков, ему предлагается выбрать одну из семи функций, представленных на рисунке 6.1.

Выбирая функции для работы с записями, пользователь может вносить, изменять или удалять записи, процесс выполнения представлен на рисунках 6.2-6.4.

Если пользователю требуется экспортировать базу данных в формат JSON, он может выбрать соответствующую функцию, представленную на рисунке 6.5.

Для вывода небольшой справки о работе программы, пользователь может выбрать соответствующую функцию, представленную на рисунке 6.6.

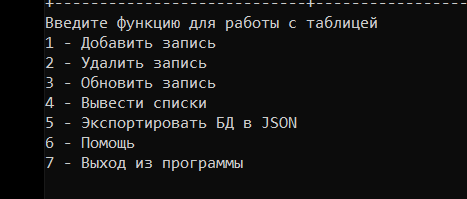


Рисунок 6.1 – Выбор функции

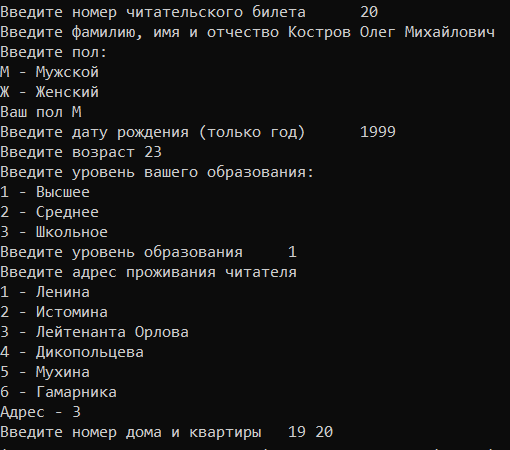


Рисунок 6.2 – Добавление читателя

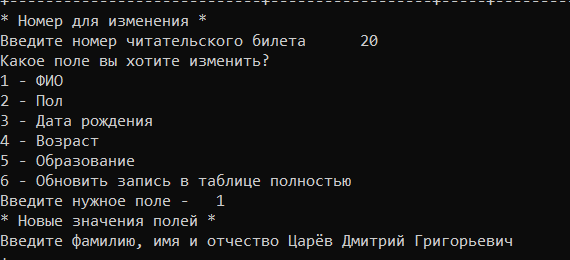


Рисунок 6.3 – Изменение ФИО читателя

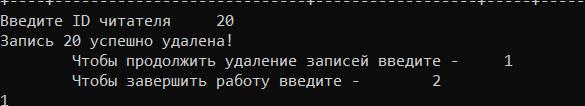


Рисунок 6.4 – Удаление читателя из базы

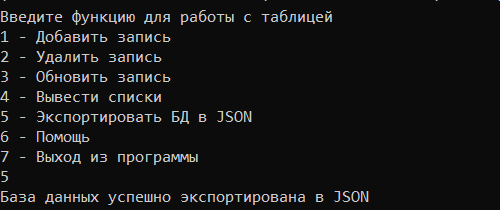


Рисунок 6.5 – Экспорт базы данных в JSON

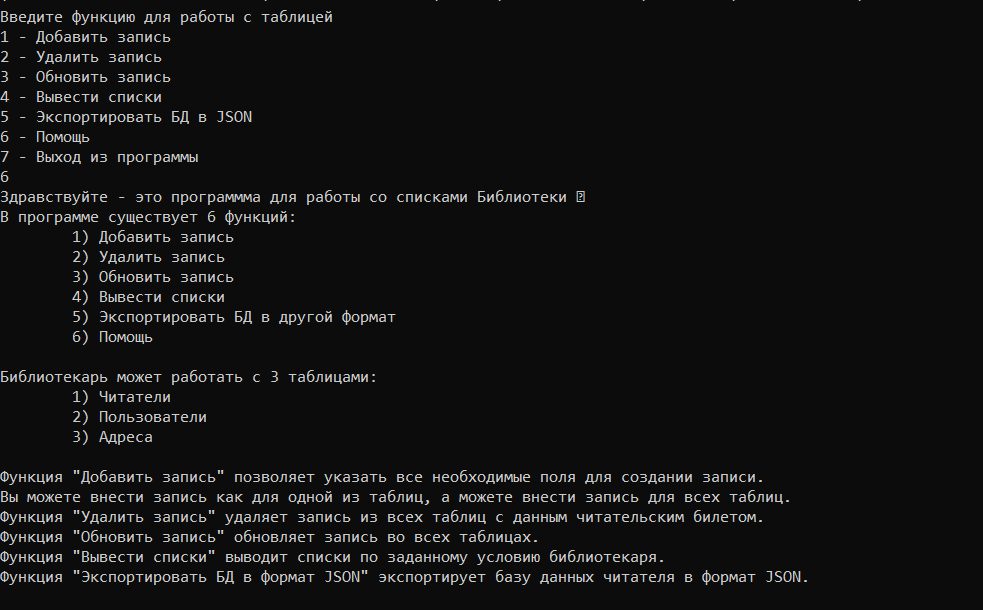


Рисунок 6.6 – Вызов помощи

7 ЛИСТИНГ ПРОГРАММЫ

import sqlite3  
import json  
from prettytable import PrettyTable  
def sqlite\_connect():  
try:  
sql\_con = sqlite3.connect(r"D:\Курсовая работа\Программа курсовой\SQLite DataBase\Library Readers.db")  
print("База данных успешно подключена к SQLite")  
return sql\_con  
except sqlite3.Error as err:  
print("Ошибка подключения базы данных", err)  
def sqlite\_disconnect(sql\_con):  
if sql\_con:  
sql\_con.close()  
print("Соединение с SQLite закрыто")  
def sqlite\_table(sql\_con):  
cursor = sql\_con.cursor()  
sql = ("""  
PRAGMA foreign\_keys = ON;  
CREATE TABLE IF NOT EXISTS Readers (  
id\_reader INTEGER PRIMARY KEY AUTOINCREMENT  
UNIQUE);  
CREATE TABLE IF NOT EXISTS Persons (  
id\_person INTEGER PRIMARY KEY AUTOINCREMENT,  
id\_reader\_person INTEGER REFERENCES Readers (id\_reader) ON DELETE CASCADE  
ON UPDATE CASCADE,  
fio VARCHAR (70) NOT NULL ON CONFLICT ROLLBACK,  
gender VARCHAR (15) NOT NULL ON CONFLICT ROLLBACK,  
birthday INTEGER NOT NULL ON CONFLICT ROLLBACK,  
age INTEGER NOT NULL ON CONFLICT ROLLBACK,  
education VARCHAR (90) NOT NULL ON CONFLICT ROLLBACK);  
CREATE TABLE IF NOT EXISTS Addresses (  
id\_reader\_address INTEGER REFERENCES Readers (id\_reader) ON DELETE CASCADE  
ON UPDATE CASCADE,  
address VARCHAR (350) NOT NULL ON CONFLICT ROLLBACK);""")  
cursor.executescript(sql)  
sql\_con.commit()  
cursor.close()  
def sqlite\_select(sql\_con, table\_name, result=None):  
output\_table = PrettyTable()  
cursor = sql\_con.cursor()  
if result is not None:  
sql\_ex = """  
SELECT id\_reader,  
fio, gender, birthday, age, education,  
address  
FROM Readers   
LEFT JOIN   
Persons on Readers.id\_reader = Persons.id\_reader\_person   
LEFT JOIN  
Addresses on Addresses.id\_reader\_address = Readers.id\_reader;"""  
cursor.execute(sql\_ex)  
rows = cursor.fetchall()  
output\_table.field\_names = ["Номер читательского билета", "ФИО",  
"Пол", "День рождения", "Возраст", "Образование",  
"Адрес"]  
for row in rows:  
output\_table.add\_row(row)  
print(output\_table)  
elif result is None:  
if table\_name == 1:  
cursor.execute("SELECT \* FROM Readers;")  
rows = cursor.fetchall()  
output\_table.field\_names = ["Номер читательского билета"]  
output\_table.title = "Читатели"  
for row in rows:  
output\_table.add\_row(row)  
print(output\_table)  
if table\_name == 2:  
cursor.execute("SELECT \* FROM Persons ORDER BY

id\_reader\_person ASC;")  
rows = cursor.fetchall()  
output\_table.field\_names = ["ID", "Номер читательского билета", "ФИО", "Пол", "Дата рождения", "Возраст",  
"Образование"]  
output\_table.title = "Пользователи"  
for row in rows:  
output\_table.add\_row(row)  
print(output\_table)  
if table\_name == 3:  
cursor.execute("SELECT \* FROM Addresses ORDER BY

id\_reader\_address ASC;")  
rows = cursor.fetchall()  
output\_table.field\_names = ["Номер читательского билета", "Адрес"]  
output\_table.title = "Адреса"  
for row in rows:  
output\_table.add\_row(row)  
print(output\_table)  
def sqlite\_is\_exists(sql\_con, table\_name, lib\_num):  
cursor = sql\_con.cursor()  
if table\_name == 1:  
try:  
statement = cursor.execute(f"""  
SELECT EXISTS (SELECT \* FROM Readers where id\_reader = {lib\_num});""")  
if statement.fetchone()[0]:  
return True  
except sqlite3.DatabaseError as err:  
print("Ошибка базы данных")  
def sqlite\_insert(sql\_con, table\_name, znash):  
cursor = sql\_con.cursor()  
try:  
if table\_name == 1:  
cursor.execute("""  
INSERT INTO Readers VALUES(?)  
""", znash)  
elif table\_name == 2:  
cursor.execute("""  
INSERT INTO Persons VALUES(NULL,?,?,?,?,?,?)  
""", znash)  
elif table\_name == 3:  
cursor.execute("""  
INSERT INTO Addresses VALUES(?,?)  
""", znash)  
elif table\_name == "all":  
readers = [znash[0]]  
persons = [znash[0], znash[1], znash[2],  
znash[3], znash[4], znash[5]]  
addresses = [znash[0], znash[6]]  
cursor.execute("INSERT INTO Readers VALUES(?)", readers)  
cursor.execute("INSERT INTO Persons VALUES(NULL,?,?,?,?,?,?)", persons)  
cursor.execute("INSERT INTO Addresses VALUES(?,?)", addresses)  
else:  
print("Не найдено такой таблицы")  
except (sqlite3.IntegrityError, sqlite3.DatabaseError) as err:  
print("\tВы ввели неправильный номер читательского билета")  
print("Выберите номер билета из данного списка:")  
sqlite\_select(sql\_con, 1)  
sql\_con.commit()  
def sqlite\_delete(sql\_con, table\_name, id):  
cursor = sql\_con.cursor()  
if table\_name == 1:  
cursor.execute(f"""  
DELETE FROM Readers WHERE id\_reader = {id}""")  
elif table\_name == 2:  
cursor.execute(f"""  
DELETE FROM Persons WHERE id\_person = {id""")  
elif table\_name == 3:  
cursor.execute(f"""  
DELETE FROM Addresses WHERE id\_reader\_address = {id}""")  
else:  
print("Не найдено такой таблицы")  
sql\_con.commit()  
cursor.close()  
def sqlite\_update(sql\_con, table\_name, znash, statement=None):  
cursor = sql\_con.cursor()  
try:  
if table\_name == 1:  
cursor.execute(f"""  
UPDATE Readers SET id\_reader = ? WHERE id\_reader = ?  
""", znash)  
elif table\_name == 2:  
if statement == 1:  
cursor.execute("""  
UPDATE Persons SET fio = ? WHERE id\_reader\_person = ?  
""", znash)  
elif statement == 2:  
cursor.execute("""  
UPDATE Persons SET gender = ? WHERE id\_reader\_person = ?  
""", znash)  
elif statement == 3:  
cursor.execute("""  
UPDATE Persons SET birthday = ? WHERE id\_reader\_person = ?  
""", znash)  
elif statement == 4:  
cursor.execute("""  
UPDATE Persons SET age = ? WHERE id\_reader\_person = ?  
""", znash)  
elif statement == 5:  
cursor.execute("""  
UPDATE Persons SET education = ? WHERE id\_reader\_person = ?  
""", znash)  
elif statement == 6:  
cursor.execute("""  
UPDATE Persons SET   
fio = ?,  
gender = ?,  
birthday = ?,  
age = ?,  
education = ?  
WHERE id\_reader\_person = ?  
""", znash)  
elif table\_name == 3:  
cursor.execute("""  
UPDATE Addresses SET address = ? WHERE id\_reader\_address = ?  
""", znash)  
else:  
print("Не найдено такой таблицы")  
except (sqlite3.IntegrityError, sqlite3.DatabaseError) as err:  
print(err)  
print("\tВы ввели неправильный номер читательского билета")  
print("Выберите номер билета из данного списка:")  
sqlite\_select(sql\_con, 1)  
sql\_con.commit()  
def sqlite\_output\_group(sql\_con, znash, statement=None):  
output\_table = PrettyTable()  
cursor = sql\_con.cursor()  
if statement == 1:  
sql\_ex = """  
SELECT id\_reader,  
fio, gender, birthday, age, education,  
address  
FROM Readers  
LEFT JOIN   
Persons on Readers.id\_reader = Persons.id\_reader\_person  
LEFT JOIN  
Addresses on Addresses.id\_reader\_address = Readers.id\_reader  
WHERE age BETWEEN ? AND ? AND education = ?;"""  
elif statement == 2:  
sql\_ex = """  
SELECT id\_reader,  
fio, gender, birthday, age, education,  
address  
FROM Readers  
LEFT JOIN   
Persons on Readers.id\_reader = Persons.id\_reader\_person  
LEFT JOIN  
Addresses on Addresses.id\_reader\_address = Readers.id\_reader  
WHERE gender = ?"""  
elif statement == 3:  
sql\_ex = """  
SELECT id\_reader,  
fio, gender, birthday, age, education,  
address  
FROM Readers  
LEFT JOIN   
Persons on Readers.id\_reader = Persons.id\_reader\_person  
LEFT JOIN  
Addresses on Addresses.id\_reader\_address = Readers.id\_reader  
WHERE education = ? ORDER BY birthday;"""  
elif statement == 4:  
sql\_ex = """  
SELECT id\_reader,  
fio, gender, birthday, age, education,  
address  
FROM Readers  
LEFT JOIN   
Persons on Readers.id\_reader = Persons.id\_reader\_person  
LEFT JOIN  
Addresses on Addresses.id\_reader\_address = Readers.id\_reader  
WHERE id\_reader BETWEEN? AND?"""  
elif statement == 5:  
sql\_ex = """  
SELECT id\_reader,  
fio, gender, birthday, age, education,  
address  
FROM Readers  
LEFT JOIN   
Persons on Readers.id\_reader = Persons.id\_reader\_person  
LEFT JOIN  
Addresses on Addresses.id\_reader\_address = Readers.id\_reader  
WHERE fio LIKE ? ORDER BY age;"""  
try:  
cursor.execute(sql\_ex, znash)  
except sqlite3.ProgrammingError as err:  
print("Пожалуйста введите значения правильно!")  
else:  
rows = cursor.fetchall()  
output\_table.field\_names = ["Номер читательского билета", "ФИО",  
"Пол", "День рождения", "Возраст", "Образование","Адрес"]  
for row in rows:  
output\_table.add\_row(row)  
print(output\_table)  
def sqlite\_export\_json(sql\_con):  
def dict\_factory(cursor, row):  
d = {}  
for j, i in enumerate(cursor.description):  
d[i[0]] = row[j]  
return d  
tables = ['Readers', 'Persons', 'Addresses']  
data = {}  
sql\_con.row\_factory = dict\_factory  
cursor = sql\_con.cursor()  
for table in tables:  
cursor.execute(f"SELECT \* FROM {table}")  
rows = cursor.fetchall()  
data[table] = rows  
with open("SQLite DataBase/LibraryReaders.json", 'w') as fp:  
json.dump(data, fp, indent=4)

def insert\_data(iterat, table\_name, \_readers):

if table\_name == 1:

return [\_readers[iterat].lib\_card\_num]

if table\_name == 2:

return [\_readers[iterat].lib\_card\_num, \_readers[iterat].surname\_np, \_readers[iterat].gender,

\_readers[iterat].birthday\_date, \_readers[iterat].age, \_readers[iterat].education]

if table\_name == 3:

return [\_readers[iterat].lib\_card\_num, \_readers[iterat].address]

if table\_name == "all":

return [\_readers[iterat].lib\_card\_num, \_readers[iterat].surname\_np,

\_readers[iterat].gender, \_readers[iterat].birthday\_date,

\_readers[iterat].age, \_readers[iterat].education,

\_readers[iterat].address]

i = 0

if insert\_all == 1:

DefSQL.sqlite\_select(sql\_con, 1, True)

\_readers[i].lib\_card\_num = ()

is\_exists = DefSQL.sqlite\_is\_exists(sql\_con, 1, \_readers[i].lib\_card\_num)

if is\_exists:

print("Выберите другой читательский билет")

elif not is\_exists:

\_readers[i].surname\_np = ()

\_readers[i].gender = ()

\_readers[i].birthday\_date = ()

\_readers[i].age = ()

\_readers[i].education = ()

\_readers[i].address = ()

DefSQL.sqlite\_insert(sql\_con, "all", insert\_data(i, "all", \_readers))

DefSQL.sqlite\_select(sql\_con, table\_name, True)

elif insert\_all == 2:

DefSQL.sqlite\_select(sql\_con, table\_name)

if table\_name == 1:

\_readers[i].lib\_card\_num = ()

elif table\_name == 2:

\_readers[i].lib\_card\_num = ()

is\_exists = DefSQL.sqlite\_is\_exists(sql\_con, 1, \_readers[i].lib\_card\_num)

if is\_exists:

\_readers[i].surname\_np = ()

\_readers[i].gender = ()

\_readers[i].birthday\_date = ()

\_readers[i].age = ()

\_readers[i].education = ()

elif not is\_exists:

print("Выберите другой читательский билет")

else:

print("Не существует такого читательского билета")

elif table\_name == 3:

\_readers[i].lib\_card\_num = ()

is\_exists = DefSQL.sqlite\_is\_exists(sql\_con, 1, \_readers[i].lib\_card\_num)

if is\_exists:

\_readers[i].address = ()

if not is\_exists:

print("Выберите другой читательский билет")

else:

print("Не существует такого читательского билета")

DefSQL.sqlite\_insert(sql\_con, table\_name, insert\_data(i, table\_name, \_readers))

ЗАКЛЮЧЕНИЕ

Результатом курсовой работы является разработка приложения для работы с базой данных «Библиотека», предназначенного для повышения эффективности работы сотрудников библиотеки.

Разработанная программа автоматизации работы формирования списков читателей достаточно проста в эксплуатации, легка в использовании, не требует дополнительного специального обучения персонала и реализует необходимые функции для учёта читателей библиотеки, для возможной правки данных записей и экспорта записей в другой формат.

Программа может быть использована как начальный этап разработки полноценной автоматизированной информационной системы для библиотеки. Программу можно значительно расширить, введя новые базы данных и функционал программы, это может быть, к примеру введение базы данных о книгах библиотеки или расширение программы на веб-сайт.

Программа оформлена по требованиям, она имеет официальный интерфейс, поддерживает целостность и непротиворечивость данных и обеспечивает вывод на экран всех необходимых списков читателей.

В ходе выполнения курсовой работы решены все поставленные задачи.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Баженова, И.Ю. Языки программирования: Учебник для студентов учреждений высш. проф. образования / И.Ю. Баженова; Под ред. В.А. Сухомлин. — М.: ИЦ Академия, 2018. — 368 c.

2 Гавриков, М.М. Теоретические основы разработки и реализации языков программирования: Учебное пособие / М.М. Гавриков, А.Н. Иванченко, Д.В. Гринченков. — М.: КноРус, 2018. — 184 c.

3 Гергель, В.П. Современные языки и технологии параллельного программирования: Учебник/ предисл.: В.А. Садовничий. / В.П. Гергель. — М.: Изд. МГУ, 2016. — 408 c.

4 Гордеев, С. И. Организация баз данных в 2 ч. Часть 2 : учебник для вузов / С. И. Гордеев, В. Н. Волошина. — 2-е изд., испр. и доп. — М. : Издательство Юрайт, 2019. — 501 с.

5 Гуриков, С.Р. Основы алгоритмизации и программирования на Python / С.Р. Гуриков. - М.: Форум, 2018. - 991 c.

6 Дейт, К. Дж. SQL и реляционная теория. Как грамотно писать код на SQL / К.Дж. Дейт. - М.: Символ-плюс, 2017. - 480 c.

7 Златопольский, Д. М. Основы программирования на языке Python / Д.М. Златопольский. - М.: ДМК Пресс, 2017. - 277 c.

8 Лутц, М. Программирование на Python. Т. 2 / М. Лутц. - М.: Символ, 2019. - 992 c.

9 Маркин, А. В. Программирование на sql в 2 ч. Часть 2 : учебник и практикум для бакалавриата и магистратуры / А. В. Маркин. — М. : Издательство Юрайт, 2019. — 292 с.

10 Стасышин, В. М. Базы данных: технологии доступа : учеб. пособие для СПО / В. М. Стасышин, Т. Л. Стасышина. — 2-е изд., испр. и доп. — М. : Издательство Юрайт, 2018. — 164 с.

11 Федоров, Д. Ю. Программирование на языке высокого уровня python : учеб. пособие для СПО / Д. Ю. Федоров. — М. : Издательство Юрайт, 2019. — 126 с.