**Ссылка на GitHub:** <https://github.com/Vanovango/SQL_bd_Profi>

**Кейс 1.**

1. Создать структуру базы данных, состоящую из таблиц:

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица |  |  | Атрибут |
| Пациент |  |  |  |
|  |  |  | Фамилия |
|  |  |  | Имя |
|  |  |  | Отчество |
|  |  |  | Пол |
|  |  |  | Дата рождения |
|  |  |  | Номер полиса ОМС |
| Врач |  |  |  |
|  |  |  | Фамилия |
|  |  |  | Имя |
|  |  |  | Отчество |
|  |  |  | СНИЛС |
|  |  |  | Специальность (использовать  справочник специальностей) |
| Случаи помощи | оказания | медицинской |  |
|  |  |  | Дата начала |
|  |  |  | Дата окончания |
|  |  |  | Ответственный врач |
|  | | | Результат (справочник: выздоровление, улучшение, динамическое наблюдение, направление в стационар,  ухудшение, летальный исход) |
| Диагноз, установленный в рамках случая оказания медицинской помощи | | |  |
|  | | | Дата установления |
|  | | | Код диагноза(по справочнику МКБ 10) |
|  | | | Тип диагноза (основной,  сопутствующий) |
| Медицинские услуги, оказанные в рамках случая | | |  |
|  | | | Дата оказания услуги |
|  | | | Код услуги (справочник  номенклатуры) |
|  | | | Врач |
| Справочник диагнозов | | |  |
|  | | | Код МКБ |
|  | | | Наименование |
| Справочник специальностей | | |  |
|  | | | Код специальности |
|  | | | Наименование |
| Справочник номенклатуры услуг | | |  |
|  | | | Код услуги |
|  | | | Наименование |

Наименование таблиц и полей, типы данных установить в соответствии с принятыми правилами формирования структуры БД. Установить в каждой таблице первичный ключ.

# Таблица пациентов

cursor.execute("""  
CREATE TABLE IF NOT EXISTS Patients (  
 patient\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 last\_name TEXT NOT NULL,  
 first\_name TEXT NOT NULL,  
 middle\_name TEXT,  
 gender TEXT CHECK (gender IN ('М', 'Ж')) NOT NULL,  
 birth\_date DATE NOT NULL,  
 oms\_policy\_number TEXT NOT NULL  
);  
""")

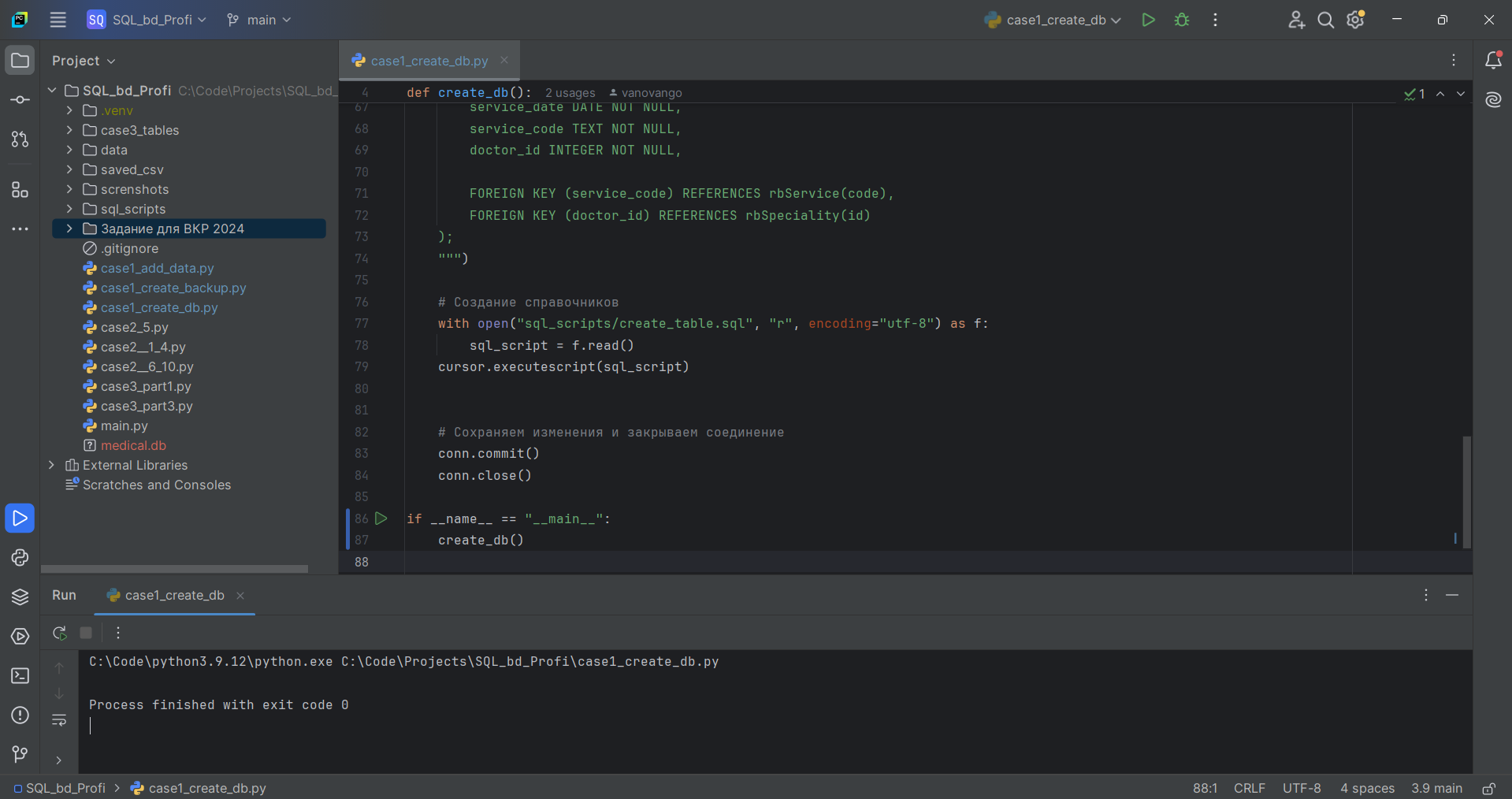
# Таблица врачей  
cursor.execute("""  
CREATE TABLE IF NOT EXISTS Doctors (  
 doctor\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 last\_name TEXT NOT NULL,  
 first\_name TEXT NOT NULL,  
 middle\_name TEXT,  
 snils TEXT UNIQUE NOT NULL,  
 specialty\_code TEXT NOT NULL,  
   
 FOREIGN KEY (specialty\_code) REFERENCES rbSpeciality(id)  
);  
""")

# Таблица случаев оказания медицинской помощи  
cursor.execute("""  
CREATE TABLE IF NOT EXISTS MedicalCases (  
 case\_id INTEGER PRIMARY KEY AUTOINCREMENT,   
 start\_date DATE NOT NULL,  
 end\_date DATE,  
 doctor\_id INTEGER NOT NULL,  
 result TEXT NOT NULL,  
   
 FOREIGN KEY (doctor\_id) REFERENCES rbSpeciality(id)  
);  
""")

# Диагнозы, установленные в рамках случая  
cursor.execute("""  
CREATE TABLE IF NOT EXISTS CaseDiagnoses (  
 diagnosis\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 diagnosis\_date DATE NOT NULL,  
 diagnosis\_code TEXT NOT NULL,  
 diagnosis\_type TEXT CHECK (diagnosis\_type IN ('основной', 'сопутствующий')),  
   
 FOREIGN KEY (diagnosis\_code) REFERENCES mkb(code)  
);  
""")

# Медицинские услуги, оказанные в рамках случая  
cursor.execute("""  
CREATE TABLE IF NOT EXISTS CaseServices (  
 service\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 service\_date DATE NOT NULL,  
 service\_code TEXT NOT NULL,  
 doctor\_id INTEGER NOT NULL,  
  
 FOREIGN KEY (service\_code) REFERENCES rbService(code),  
 FOREIGN KEY (doctor\_id) REFERENCES rbSpeciality(id)  
);  
""")

# Создание справочников  
with open("sql\_scripts/create\_table.sql", "r", encoding="utf-8") as f:  
 sql\_script = f.read()  
cursor.executescript(sql\_script)



1. Для таблиц-справочников загрузить содержимое из файлов средствами DBeaver.

В начале выполните скрипт create\_table.sql, который создаст таблицы:

Справочник диагнозов с названием таблицы MKB.

Справочник специальностей с названием таблицы rbSpeciality. Справочник номенклатурных услуг с названием таблицы rbService.

Далее выполните следующие sql скрипты для заполнения справочников: - Spec.sql

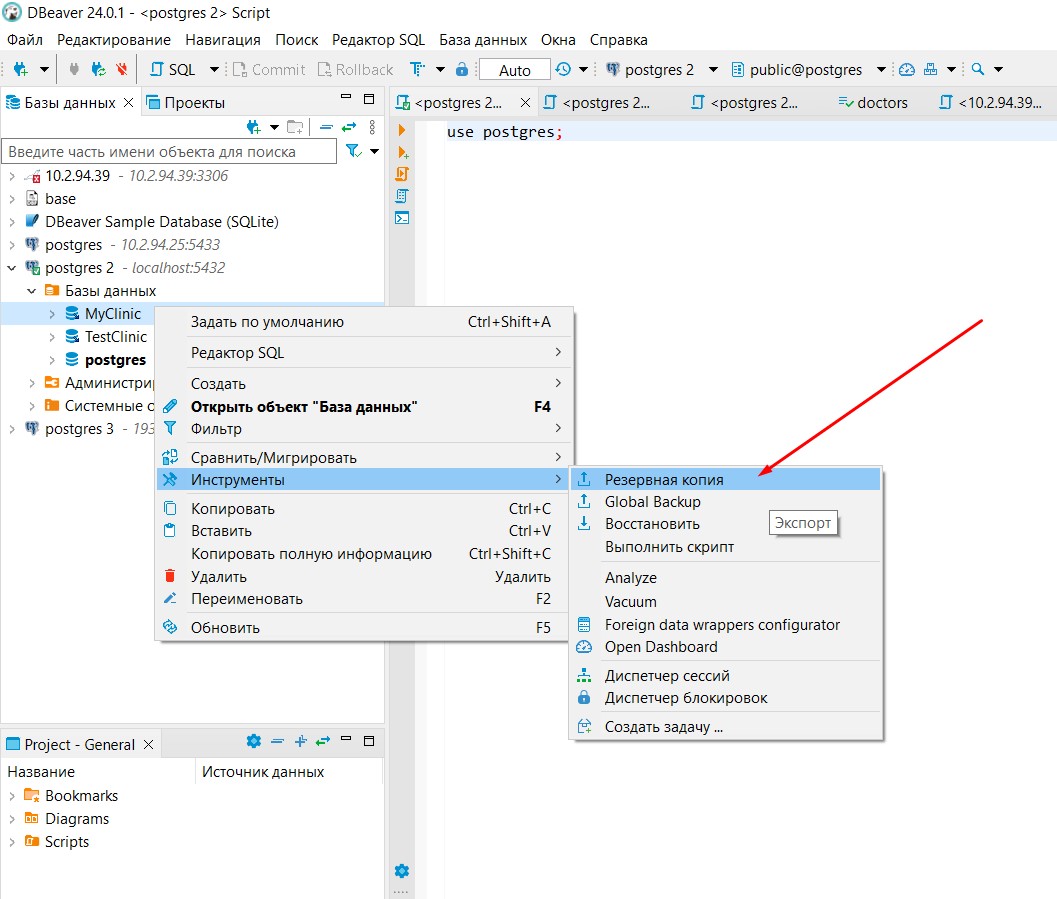
* MKB.sql
* Usl.sql

|  |  |
| --- | --- |
| Справочник | Файл |
| Справочник специальностей | Spec.sql |
| Справочник МКБ | MKB.sql |
| Справочник номенклатуры услуг | Usl.sql |

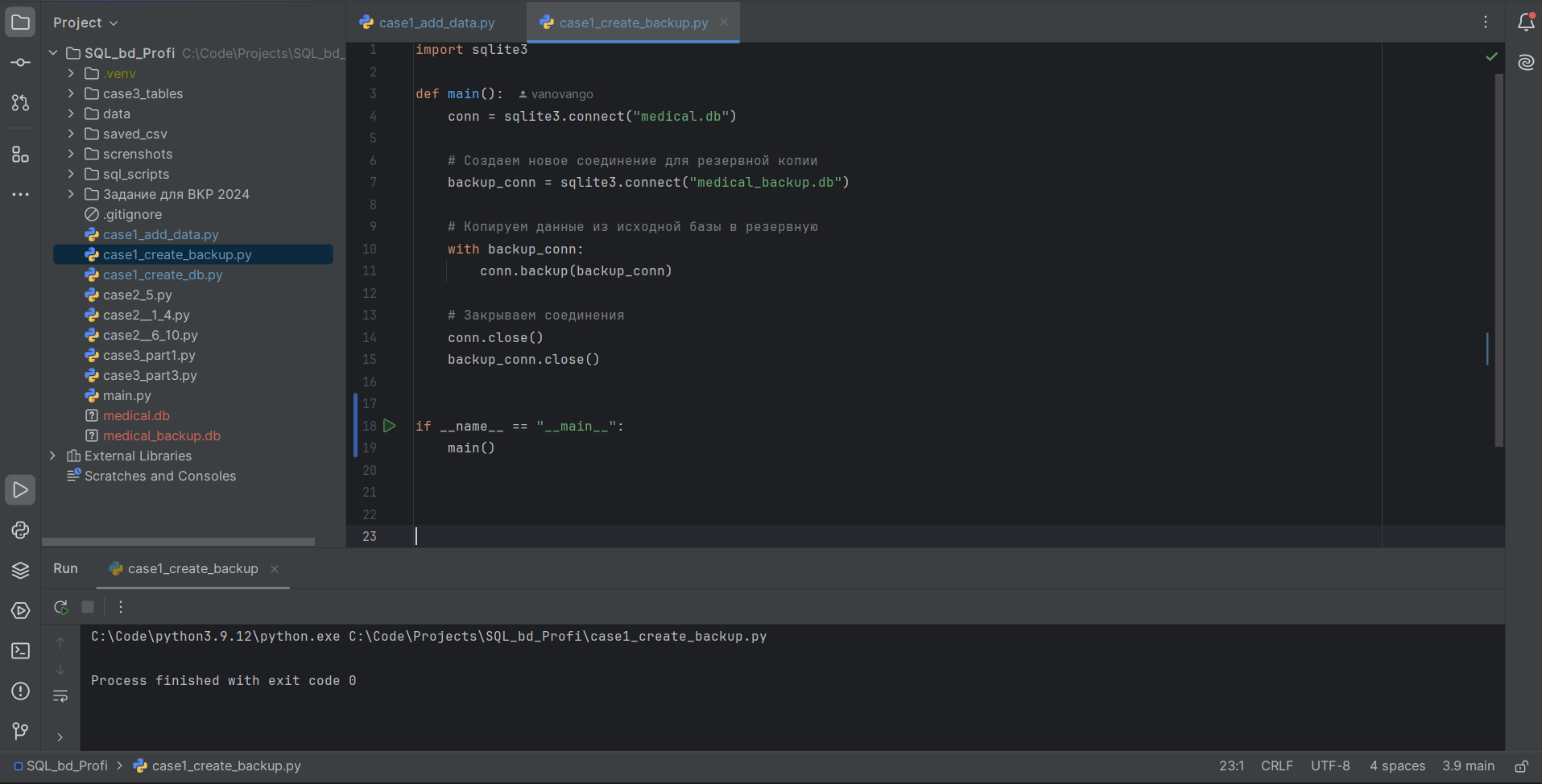
import sqlite3  
  
  
def main():  
 # Подключаемся к БД  
 conn = sqlite3.connect("medical.db")  
 cursor = conn.cursor()  
  
 # Загружаем и выполняем SQL-скрипты из файлов  
 with open("sql\_scripts/mkb.sql", "r", encoding="utf-8") as f:  
 data\_mkb = f.read()  
 cursor.executescript(data\_mkb)  
  
 with open("sql\_scripts/Spec.sql", "r", encoding="utf-8") as f:  
 data\_spec = f.read()  
 cursor.executescript(data\_spec)  
  
 with open("sql\_scripts/Usl.sql", "r", encoding="utf-8") as f:  
 data\_usl = f.read()  
 cursor.executescript(data\_usl)  
  
 # Сохраняем изменения  
 conn.commit()  
 conn.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()



3. Результат разработки и дальнейшей обработки в Модуле 5 выгрузить в резервную копию:



import sqlite3  
  
def main():  
 conn = sqlite3.connect("medical.db")  
  
 # Создаем новое соединение для резервной копии  
 backup\_conn = sqlite3.connect("medical\_backup.db")  
  
 # Копируем данные из исходной базы в резервную  
 with backup\_conn:  
 conn.backup(backup\_conn)  
  
 # Закрываем соединения  
 conn.close()  
 backup\_conn.close()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

****

**Кейс 2.**

Наполнение базы данных из Кейса 1 посредством файлов с данными.

1. Необходимо написать модуль, реализующий функционал по открытию файла Персона.csv и его и загрузке в соответствующую таблицу Пациенты базы данных, созданной в Кейсе 1.

def load\_into\_patients():  
 conn = sqlite3.connect("medical.db")  
 cursor = conn.cursor()  
  
 data = load\_data\_person()  
  
 for row in data:  
 cursor.execute("""  
 INSERT INTO Patients (patient\_id, last\_name, first\_name, middle\_name, gender, birth\_date, oms\_policy\_number)  
 VALUES (?, ?, ?, ?, ?, ?, ?)  
 """, (  
 row["ID"],  
 row["фамилия"],  
 row["имя"],  
 row["отчество"],  
 row["пол"],  
 row["дата рождения"],  
 row["номер полиса ОМС"]  
 ))  
  
 conn.commit()  
 conn.close()

1. Необходимо написать модуль, реализующий функционал по открытию файла Услуги.csv и его и загрузке в соответствующую таблицу «Случаи оказания медицинской помощи» базы данных, созданной в Кейсе 1.

def load\_into\_medical\_cases():  
 conn = sqlite3.connect("medical.db")  
 cursor = conn.cursor()  
  
 data = load\_data\_service()  
  
 for row in data:  
 cursor.execute("""  
 INSERT INTO MedicalCases (start\_date, end\_date, doctor\_id, result)  
 VALUES (?, ?, ?, ?)  
 """, (  
 row["дата начала"],  
 row["дата окончания"],  
 row["Специальность врача"],  
 row["Исход обращения"]  
 ))  
  
 conn.commit()  
 conn.close()

1. Необходимо написать модуль, реализующий функционал по открытию файла Услуги.csv и его и загрузке в соответствующую таблицу «Диагнозы, установленный в рамках случая» базы данных, созданной в Кейсе 1.

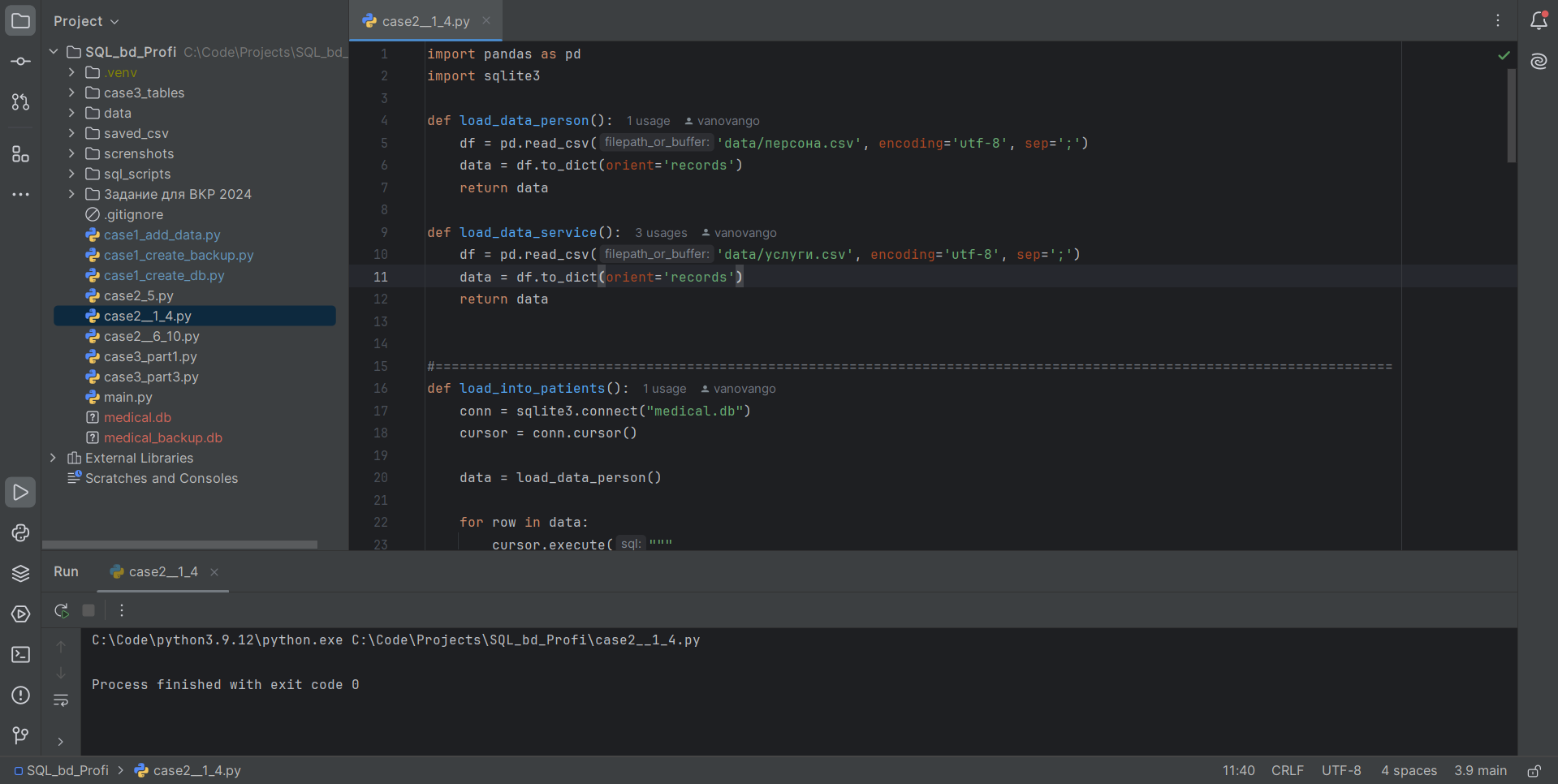
def load\_into\_case\_diagnoses():  
 conn = sqlite3.connect("medical.db")  
 cursor = conn.cursor()  
  
 data = load\_data\_service()  
  
 for row in data:  
 cursor.execute("""  
 INSERT INTO CaseDiagnoses (diagnosis\_date, diagnosis\_code)  
 VALUES (?, ?)  
 """, (  
 row["дата окончания"],  
 row["Диагноз"]  
 ))  
  
 conn.commit()  
 conn.close()

1. Необходимо написать модуль, реализующий функционал по открытию файла Услуги.csv и его и загрузке в соответствующую таблицу «Медицинские услуги» базы данных, созданной в Кейсе 1.

def load\_into\_case\_services():  
 conn = sqlite3.connect("medical.db")  
 cursor = conn.cursor()  
  
 data = load\_data\_service()  
  
 for row in data:  
 cursor.execute("""  
 INSERT INTO CaseServices (service\_date, service\_code, doctor\_id)  
 VALUES (?, ?, ?)  
 """, (  
 row["дата начала"],  
 row["Услуга"],  
 row["Специальность врача"]  
 ))  
  
 conn.commit()  
 conn.close()

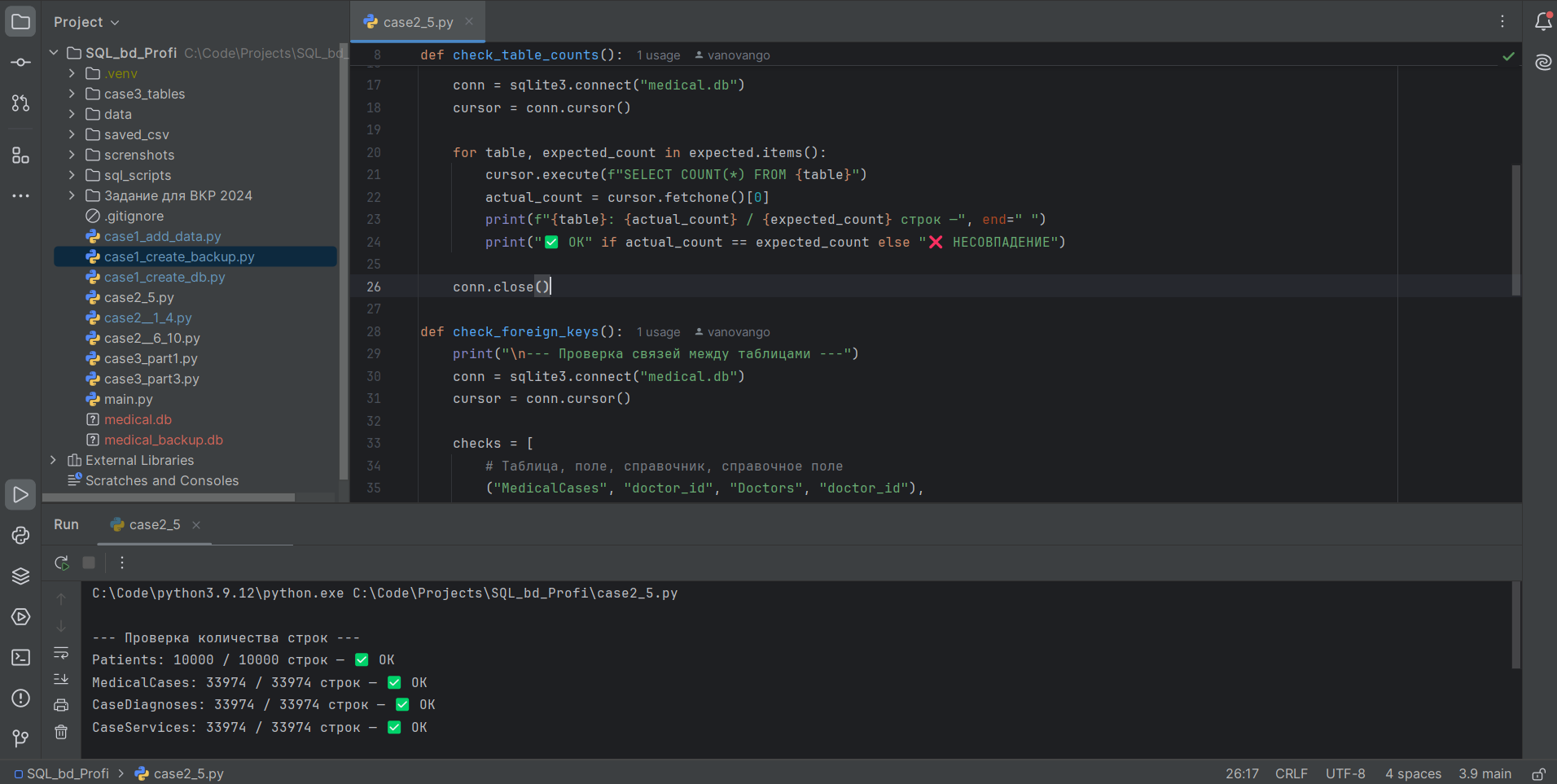
Функции для загрузки таблиц:

def load\_data\_person():  
 df = pd.read\_csv('data/персона.csv', encoding='utf-8', sep=';')  
 data = df.to\_dict(orient='records')  
 return data  
  
def load\_data\_service():  
 df = pd.read\_csv('data/услуги.csv', encoding='utf-8', sep=';')  
 data = df.to\_dict(orient='records')  
 return data



1. Необходимо написать модуль для осуществления контроля загруженных из файлов данных. Данный модуль должен обеспечивать функционал для контроля количества загруженных строк в соответствии с количеством строк в файлах. Также модуль должен обеспечивать функционал по контролю корректности построенных связей между таблицами по ключевым полям.

import sqlite3  
  
def count\_lines\_in\_csv(filepath, skip\_header=True):  
 with open(filepath, "r", encoding="utf-8") as f:  
 lines = f.readlines()  
 return len(lines) - 1 if skip\_header else len(lines)  
  
def check\_table\_counts():  
 print("\n--- Проверка количества строк ---")  
 expected = {  
 "Patients": count\_lines\_in\_csv("./data/персона.csv"),  
 "MedicalCases": count\_lines\_in\_csv("./data/услуги.csv"),  
 "CaseDiagnoses": count\_lines\_in\_csv("./data/услуги.csv"),  
 "CaseServices": count\_lines\_in\_csv("./data/услуги.csv"),  
 }  
  
 conn = sqlite3.connect("medical.db")  
 cursor = conn.cursor()  
  
 for table, expected\_count in expected.items():  
 cursor.execute(f"SELECT COUNT(\*) FROM {table}")  
 actual\_count = cursor.fetchone()[0]  
 print(f"{table}: {actual\_count} / {expected\_count} строк —", end=" ")  
 print("✅ OK" if actual\_count == expected\_count else "❌ НЕСОВПАДЕНИЕ")  
  
 conn.close()  
  
def check\_foreign\_keys():  
 print("\n--- Проверка связей между таблицами ---")  
 conn = sqlite3.connect("medical.db")  
 cursor = conn.cursor()  
  
 checks = [  
 # Таблица, поле, справочник, справочное поле  
 ("MedicalCases", "doctor\_id", "Doctors", "doctor\_id"),  
 ("CaseDiagnoses", "diagnosis\_code", "mkb", "code"),  
 ("CaseServices", "service\_code", "rbService", "code"),  
 ("CaseServices", "doctor\_id", "Doctors", "doctor\_id")  
 ]  
  
 for table, fk\_field, ref\_table, ref\_field in checks:  
 query = f"""  
 SELECT COUNT(\*)   
 FROM {table} t  
 LEFT JOIN {ref\_table} r ON t.{fk\_field} = r.{ref\_field}  
 WHERE r.{ref\_field} IS NULL  
 """  
 cursor.execute(query)  
 missing = cursor.fetchone()[0]  
 print(f"{table}.{fk\_field} → {ref\_table}.{ref\_field}: ", end="")  
 print("✅ OK" if missing == 0 else f"❌ {missing} не найдены в {ref\_table}")  
  
 conn.close()  
  
def main():  
 check\_table\_counts()  
 check\_foreign\_keys()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()



1. Необходимо написать модуль, обеспечивающий посредством SQL запросов вывод данных в терминал из таблицы Пациенты, в том числе с использованием предикатов по возрасту и полу. Также в модуле должна быть обеспечена функциональность по группировке пациентов (исключая вывод персональных данных) по возрасту и полу с выводом результата в терминал. Группировка по возрасту должна быть обеспечена с шагом 10 лет.

def export\_patients\_grouped\_by\_age\_gender():  
 conn = sqlite3.connect("medical.db")  
 cursor = conn.cursor()  
  
 cursor.execute("SELECT gender, birth\_date FROM Patients")  
 rows = cursor.fetchall()  
  
 from collections import defaultdict  
  
 def parse\_birth(date\_str):  
 for fmt in ("%d.%m.%Y", "%Y-%m-%d"):  
 try:  
 return datetime.strptime(date\_str.strip(), fmt)  
 except:  
 continue  
 return None  
  
 result = defaultdict(int)  
 for gender, birth\_str in rows:  
 date = parse\_birth(birth\_str)  
 if not date: continue  
 age = int((datetime.now() - date).days / 365.25)  
 age\_group = (age // 10) \* 10  
 result[(gender, age\_group)] += 1  
  
 data = [(g, f"{a}-{a+9}", count) for (g, a), count in sorted(result.items())]  
 save\_to\_csv("patients\_by\_age\_gender.csv", ["Пол", "Возрастная группа", "Количество"], data)  
 conn.close()

1. Необходимо написать модуль, обеспечивающий посредством SQL запросов вывод данных в терминал из таблицы «Случаи оказания медицинской помощи», в том числе с использованием предикатов по периоду лечения и исходу обращения. Также в модуле должна быть обеспечена функциональность по группировке случаев по месяцу и году или исходу обращения с выводом результата в терминал.

def export\_case\_diagnoses\_grouped():  
 conn = sqlite3.connect("medical.db")  
 cursor = conn.cursor()  
  
 cursor.execute("SELECT diagnosis\_code FROM CaseDiagnoses")  
 rows = cursor.fetchall()  
  
 from collections import defaultdict  
 counts = defaultdict(int)  
 for (code,) in rows:  
 if code:  
 counts[code.strip().upper()] += 1  
  
 data = [(code, count) for code, count in sorted(counts.items(), key=lambda x: -x[1])]  
 save\_to\_csv("diagnoses\_grouped.csv", ["Код МКБ", "Количество"], data)  
 conn.close()

1. Необходимо написать модуль, обеспечивающий посредством SQL запросов вывод данных в терминал из таблицы «Диагнозы, установленный в рамках случая», в том числе с использованием предикатов по коду МКБ. Также в модуле должна быть обеспечена функциональность по группировке случаев по коду МКБ с выводом результата в терминал.

def export\_case\_services\_grouped():  
 conn = sqlite3.connect("medical.db")  
 cursor = conn.cursor()  
  
 cursor.execute("SELECT service\_code FROM CaseServices")  
 rows = cursor.fetchall()  
  
 from collections import defaultdict  
 counts = defaultdict(int)  
 for (code,) in rows:  
 if code:  
 counts[code.strip().upper()] += 1  
  
 data = [(code, count) for code, count in sorted(counts.items(), key=lambda x: -x[1])]  
 save\_to\_csv("services\_grouped.csv", ["Код услуги", "Количество"], data)  
 conn.close()

1. Необходимо написать модуль, обеспечивающий посредством SQL запросов вывод данных в терминал из таблицы «Медицинские услуги», в том числе с использованием предикатов по коду услуги. Также в модуле должна быть обеспечена функциональность по группировке случаев по коду услуги с выводом результата в терминал.

def export\_medical\_cases\_by\_month():  
 conn = sqlite3.connect("medical.db")  
 cursor = conn.cursor()  
  
 cursor.execute("SELECT start\_date FROM MedicalCases")  
 rows = cursor.fetchall()  
  
 from collections import defaultdict  
 counts = defaultdict(int)  
  
 def parse\_date(date\_str):  
 for fmt in ("%d.%m.%Y", "%Y-%m-%d"):  
 try:  
 return datetime.strptime(date\_str.strip(), fmt)  
 except:  
 continue  
 return None  
  
 for (d,) in rows:  
 date = parse\_date(d)  
 if date:  
 key = date.strftime("%Y-%m")  
 counts[key] += 1  
  
 data = [(k, v) for k, v in sorted(counts.items())]  
 save\_to\_csv("medical\_cases\_by\_month.csv", ["Месяц", "Количество случаев"], data)  
 conn.close()

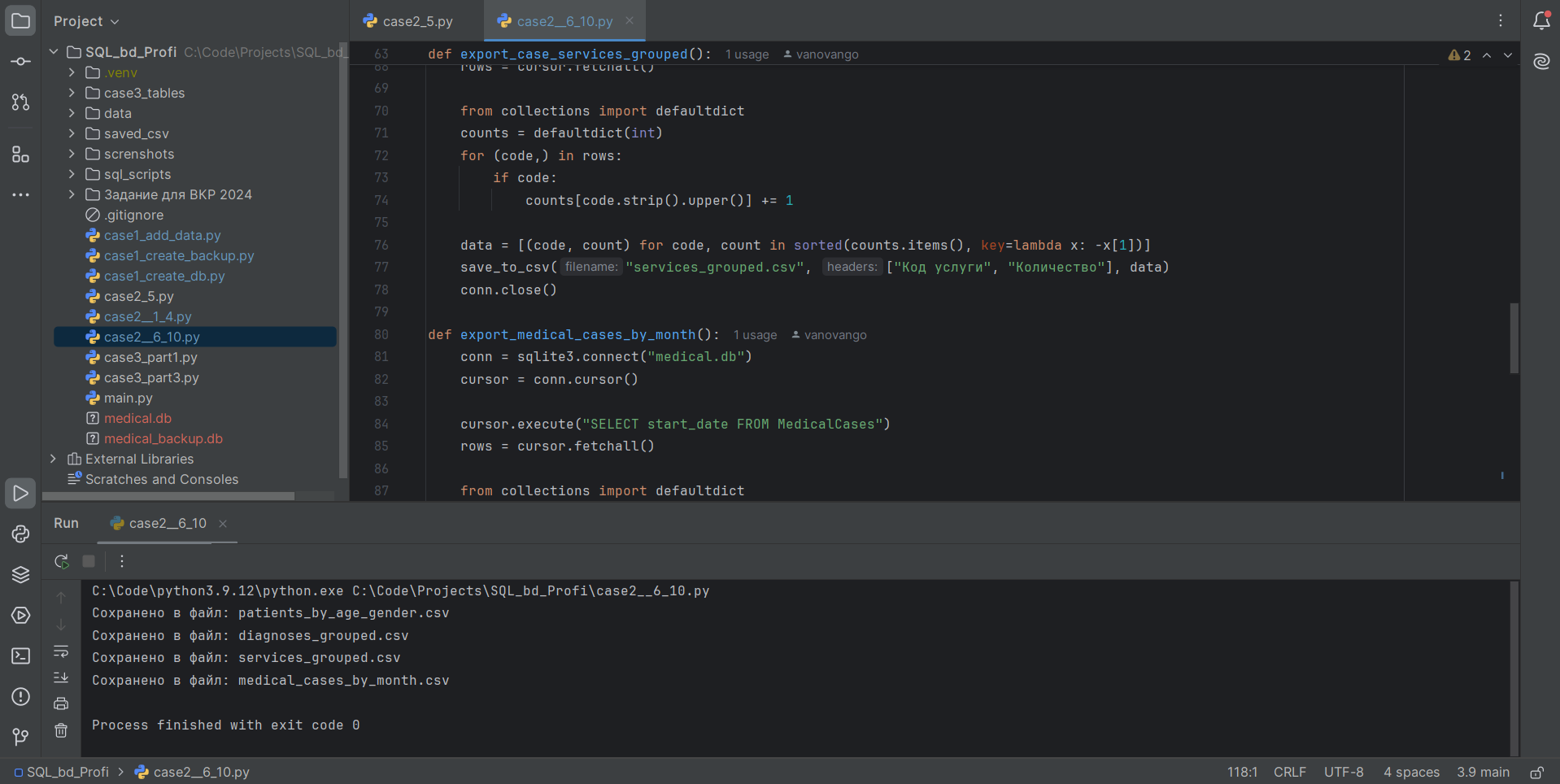
1. Необходимо обеспечить возможность сохранения результатов выборок в формат CSV с заголовками.

Все результаты выборок сохраняются в CSV формате после выполнения функции, по средствам выполнения следующих строк кода:

save\_to\_csv("medical\_cases\_by\_month.csv", ["Месяц", "Количество случаев"], data)

Дополнительные функции для реализации:

def save\_to\_csv(filename, headers, data):  
 export\_dir = "./saved\_csv"  
 if not os.path.exists(export\_dir):  
 os.makedirs(export\_dir)  
  
 with open(f'{export\_dir}/{filename}', "w", encoding="utf-8", newline="") as f:  
 writer = csv.writer(f)  
 writer.writerow(headers)  
 writer.writerows(data)  
 print(f"Сохранено в файл: {filename}")



**Кейс 3.**

Блок 1. Определение строк для выборки.

* 1. Для дальнейшей работы вам необходимо использоваться два файла «услуги.csv», и «персона.csv», данные из этих файлов необходимо загрузить в табличный редактор.
  2. Вам необходимо определить какие строки из файла «услуги.csv» должны использоваться для выполнения вашего задания по модулю 3, далее эти строки мы будем называть выборкой.
  3. Для определения выборки вам необходимо взять первую букву вашей фамилии и далее оставить только те строки, которые указаны ниже в таблице. Данные строки мы будем называть **выборкой**. Правило формирование выборки в таблице указано для первых 6-и строк, для последующих строк необходимо использовать сдвиг на 6, т.е. если в вашу выборке указано что необходимо использовать строку 1, то вам необходимо использовать строки 1,7,13,19,25,31,37 и т.д.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Строки** | **А** | **Б** | **В** | **Г** | **Д** | **Е** | **Ж** | **З** | **И** | **К** | **Л** | **М** | **Н** | **О** | **П** | **Р** | **С** | **Т** | **У** | **Ф** | **Х** | **Ц** | **Ч** | **Ш**  **/**  **Щ** | **Э** | **Ю** | **Я** |
| **1** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| **3** | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| **4** | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| **5** | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| **6** | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |

**Пример**: Допустим ваша фамилия начинается на букву «Ш», в этом случае вам необходимо использовать строки 2,4,5 и далее 8=2+6, 10=4+6, 11=5+6. Суммарно у вас должно остаться 16987 строк.

**(!) Важное**: Выборка распространяется только на Кейс 3, для остальных кейсов необходимо использовать все данные из файлов.

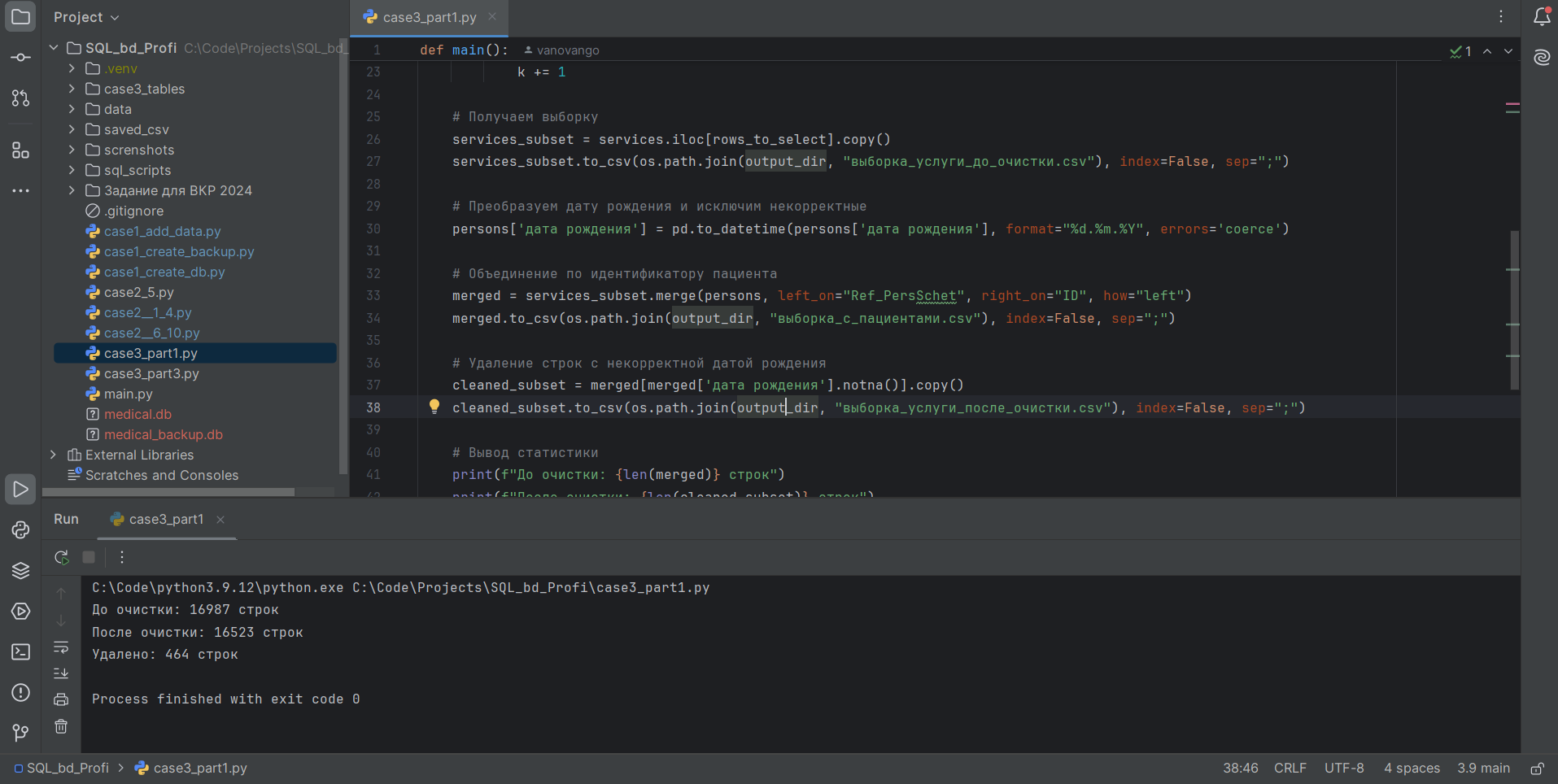
Блок 2. Проверка данных

Вам необходимо найти и исключить из вашей Выборки строки услуг с ошибками. Под ошибкой подразумевается – некорректно указана дата рождения. Все дальнейшие расчеты в модуле 3 необходимо проводить только с оставшимися строками.

**Пример:** Для фамилии, которая начинается на букву «Ш» количество строк с ошибками равно 453. Значит в нашей выборке должно остаться 16534 строки=16987-453. В

Код для реализации блока 1 и 2

import pandas as pd  
import os  
  
# Создание папки для результатов  
output\_dir = "case3\_tables"  
os.makedirs(output\_dir, exist\_ok=True)  
  
# Загрузка исходных данных  
persons = pd.read\_csv("data/персона.csv", delimiter=";")  
services = pd.read\_csv("data/услуги.csv", delimiter=";")  
  
# Определение базовых строк (строки 2, 3, 4 → индексы 1, 2, 3)  
base\_indices = [1, 2, 3]  
max\_index = len(services)  
rows\_to\_select = []  
  
# Формируем индексы: каждая базовая строка + 6 \* k  
for base in base\_indices:  
 k = 0  
 while (index := base + 6 \* k) < max\_index:  
 rows\_to\_select.append(index)  
 k += 1  
  
# Получаем выборку  
services\_subset = services.iloc[rows\_to\_select].copy()  
services\_subset.to\_csv(os.path.join(output\_dir, "выборка\_услуги\_до\_очистки.csv"), index=False, sep=";")  
  
# Преобразуем дату рождения и исключим некорректные  
persons['дата рождения'] = pd.to\_datetime(persons['дата рождения'], format="%d.%m.%Y", errors='coerce')  
  
# Объединение по идентификатору пациента  
merged = services\_subset.merge(persons, left\_on="Ref\_PersSchet", right\_on="ID", how="left")  
merged.to\_csv(os.path.join(output\_dir, "выборка\_с\_пациентами.csv"), index=False, sep=";")  
  
# Удаление строк с некорректной датой рождения  
cleaned\_subset = merged[merged['дата рождения'].notna()].copy()  
cleaned\_subset.to\_csv(os.path.join(output\_dir, "выборка\_услуги\_после\_очистки.csv"), index=False, sep=";")  
  
# Вывод статистики  
print(f"До очистки: {len(merged)} строк")  
print(f"После очистки: {len(cleaned\_subset)} строк")  
print(f"Удалено: {len(merged) - len(cleaned\_subset)} строк")



Блок 3. Пять вопросов

1. Попали ли в вашу выборку услуги с диагнозами
   1. A09.0 = ДА
   2. B35.4 = ДА
   3. C20 = ДА

Ответ: 3 значения ДА/НЕТ

1. Какое количество услуг попало в вашу выборку по следующим разрезам:
   1. Мужчины, до 30-и лет = 926
   2. Мужчина, от 30-и до 50-и лет = 753
   3. Мужчина от 50-и лет. = 1270
   4. Женщины, до 30-и лет = 894
   5. Женщины, от 30-и до 50-и лет = 919
   6. Женщины от 50-и лет = 1818

Ответ: 6 чисел

1. Какое количество персон (уникальных пациентов), попали в вашу выборку?

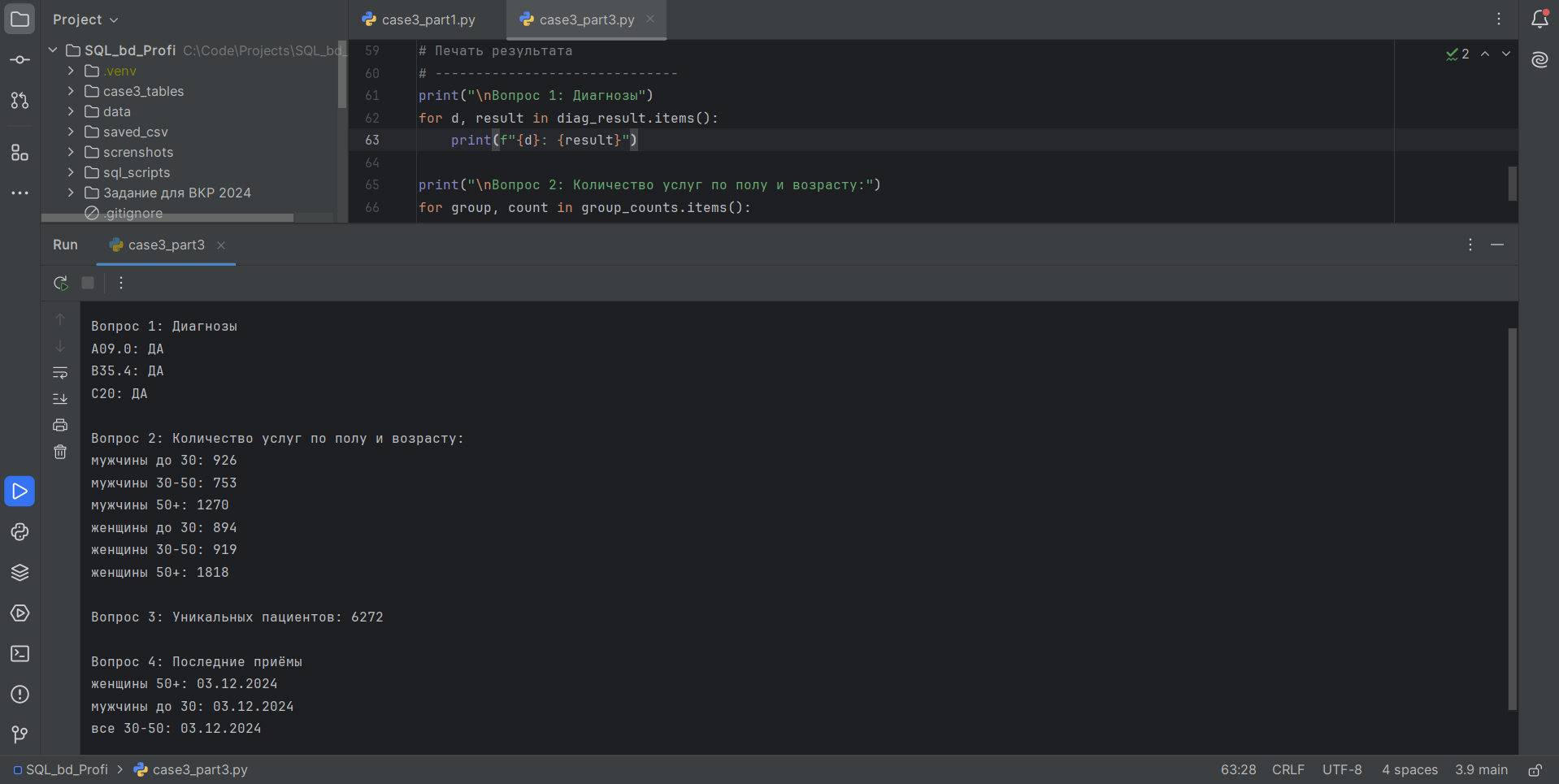
Ответ: Число = 6272

1. Дата последнего приема любого пациента, соответствующему критериями:
   1. Женщина старше 50-ти лет. = 03.12.2024
   2. Мужчина до 30-и лет = 03.12.2024
   3. Пациент (пол не важен) от 30 до 50-и лет. = 03.12.2024

Ответ: 3 числа

Код для ответа на заданные вопросы:

import pandas as pd  
from datetime import datetime  
  
# Загрузка очищенной выборки  
df = pd.read\_csv("case3\_tables/выборка\_услуги\_после\_очистки.csv", delimiter=";")  
  
# Преобразуем даты  
df["дата рождения"] = pd.to\_datetime(df["дата рождения"], errors="coerce")  
df["дата начала"] = pd.to\_datetime(df["дата начала"], errors="coerce")  
  
# ------------------------------  
# Вопрос 1: Диагнозы в выборке  
# ------------------------------  
diag\_check = ["A09.0", "B35.4", "C20"]  
diag\_result = {code: "ДА" if code in df["Диагноз"].values else "НЕТ" for code in diag\_check}  
  
# ------------------------------  
# Вопрос 2: Группировка по полу и возрасту  
# ------------------------------  
# Текущий год для расчёта возраста  
current\_year = datetime.now().year  
  
# Возраст на момент обращения  
df["возраст"] = df["дата начала"].dt.year - df["дата рождения"].dt.year  
  
# Условия по группам  
conditions = {  
 "мужчины до 30": ((df["пол"] == "М") & (df["возраст"] < 30)),  
 "мужчины 30-50": ((df["пол"] == "М") & (df["возраст"] >= 30) & (df["возраст"] < 50)),  
 "мужчины 50+": ((df["пол"] == "М") & (df["возраст"] >= 50)),  
 "женщины до 30": ((df["пол"] == "Ж") & (df["возраст"] < 30)),  
 "женщины 30-50": ((df["пол"] == "Ж") & (df["возраст"] >= 30) & (df["возраст"] < 50)),  
 "женщины 50+": ((df["пол"] == "Ж") & (df["возраст"] >= 50)),  
}  
  
group\_counts = {group: df[cond].shape[0] for group, cond in conditions.items()}  
  
# ------------------------------  
# Вопрос 3: Уникальные пациенты  
# ------------------------------  
unique\_patients = df["ID"].nunique()  
  
# ------------------------------  
# Вопрос 4: Последние визиты по группам  
# ------------------------------  
def last\_visit\_date(condition):  
 subset = df[condition]  
 return subset["дата начала"].max() if not subset.empty else None  
  
last\_visits = {  
 "женщины 50+": last\_visit\_date(conditions["женщины 50+"]),  
 "мужчины до 30": last\_visit\_date(conditions["мужчины до 30"]),  
 "все 30-50": last\_visit\_date(  
 ((df["возраст"] >= 30) & (df["возраст"] < 50))  
 ),  
}  
  
# ------------------------------  
# Печать результата  
# ------------------------------  
print("\nВопрос 1: Диагнозы")  
for d, result in diag\_result.items():  
 print(f"{d}: {result}")  
  
print("\nВопрос 2: Количество услуг по полу и возрасту:")  
for group, count in group\_counts.items():  
 print(f"{group}: {count}")  
  
print(f"\nВопрос 3: Уникальных пациентов: {unique\_patients}")  
  
print("\nВопрос 4: Последние приёмы")  
for group, date in last\_visits.items():  
 print(f"{group}: {date.strftime('%d.%m.%Y') if pd.notna(date) else 'нет данных'}")



**Кейс 4.**

1. Определить категории ПДн, содержащиеся в выборке

Ответ:

В выборке содержатся **персональные данные, включая специальную категорию**, а именно:

* идентификационные (ФИО, пол, дата рождения, СНИЛС),
* данные о здоровье (диагнозы, обращения, исход лечения)

1. Считать, что количество субъектов ПДн в общей БД больше, чем в выборке в x1 000, x10 000, x50 000 (три варианта)
2. Определить уровень защищённости ПДн общей БД

Ответ:

* При масштабе **x1 000**: 2 уровень защищенности.
* При **x10 000** и **x50 000**: 1 уровень защищенности.

1. В соответствии с Приказом ФСТЭК России от 18 февраля 2013 г. N 21 определить базовый набор мер для соответствующего уровня защищенности ПДн.

Ответ:

Вот примерный **базовый набор мер защиты ПДн**, которые требуются для разных уровней:

**2 уровень (x1 000):**

1. Разграничение доступа (по ролям, паролям)
2. Защита каналов передачи (VPN, шифрование)
3. Защита от НСД (несанкционированного доступа)
4. Контроль целостности данных
5. Учет действий пользователей
6. Антивирусная защита
7. Аудит и журналы событий

**1 уровень (x10 000 и x50 000):**

Все из вышеуказанного плюс:

1. Применение сертифицированных СЗИ (средств защиты информации)
2. Аппаратные средства доверенной загрузки и мониторинга
3. Межсетевые экраны и DLP-системы
4. Регулярный аудит безопасности и тесты на проникновение
5. Физическая охрана серверных помещений

| **Масштаб БД** | **Категория ПДн** | **Уровень защищённости** | **Базовые меры** |
| --- | --- | --- | --- |
| x1 000 | Специальная | 2 | Меры 1–7 |
| x10 000 | Специальная | 1 | Меры 1–12 |
| x50 000 | Специальная | 1 | Меры 1–12 |