4. РАБОТА С БАЗАМИ ДАННЫХ В PYTHON.

Python позволяет существенно расширить возможности баз данных в области визуализации найденных зависимостей и закономерностей.

1. Начнем с создания и заполнения базы данных.

Примечание. Если БД уже создана, ее можно открыть в Google Colab или Anaconda, но для изучения особенностей работы в Python будем создавать ее заново.
Повторим уже спроектированную БД HOSPITAL, но используя библиотеки Python, позволяющие упростить написание кода почти до чистого SQL.
Для этого открываем Google Colab или Anaconda и создаем новый блокнот.

2. Подключаем библиотеки pandas и sqlite3:

```
import pandas as pd
import sqlite3
```

Примечание 1. Здесь для pandas мы создаем алиас (псевдоним) pd, по которому и будем далее обращаться к этой библиотеке.
Примечание 2. Библиотека pandas включает разнообразные инструменты для работы с датафреймами - аналогами таблиц в SQL.
Примечание 3. Удобно писать отдельные компоненты кода в отдельных ячейках блокнота. Это важно, когда нужно выводить на экран результаты различных запросов.

4. WORKING WITH DATABASES IN PYTHON.

Watch the video (Just the beginning. Mute)

Python allows you to significantly expand the capabilities of databases in the field of visualization of found dependencies and patterns.

1. Let's start by creating and filling in the database.

Note. If the database has already been created, it can be opened in Google Collab or Anaconda, but to study the specifics of working in Python, we will create it again.
Let's repeat the HOSPITAL database that has already been designed, but using Python libraries that make it easier to write code almost to pure SQL.
To do this, open Google Colab or Anaconda and create a new notepad.

2. Connect the pandas and sqlite3 libraries:

```
import pandas as pd
import sqlite3
```

Note 1. Here, for pandas, we create an alias (pseudonym) pd, by which we will continue to access this library.

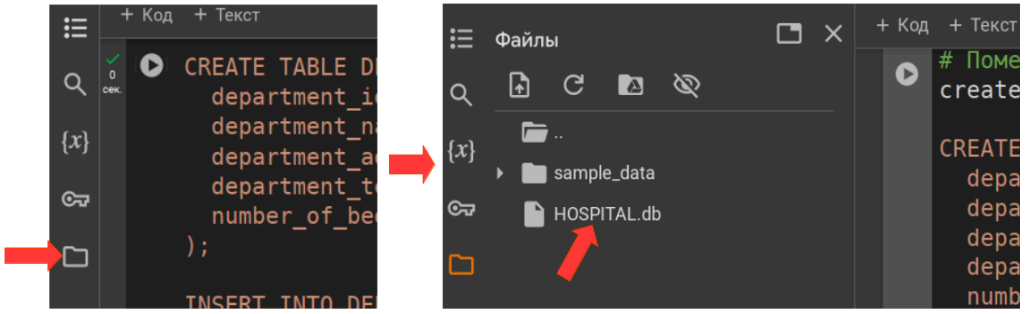
Note 2. The pandas library includes a variety of tools for working with data frames - analogs of tables in SQL.

Note 3. It is convenient to write individual code components in separate cells of a notebook. This is important when you need to display the results of various queries.

3. Подключаем БД HOSPITAL (в нашем случае - создаем):

```
conn = sqlite3.connect('HOSPITAL.db')
```

Чтобы посмотреть, что БД создана в Google Colab щелкните по значку папки на панели слева:



В открывшейся структуре должен появиться файл HOSPITAL.db

В Anaconda файл базы данных создается в той же папке, в которой был создан блокнот Jupyter Notebook.

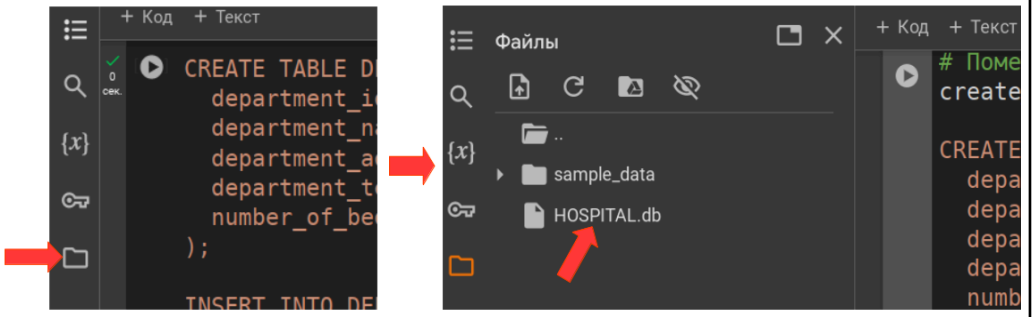
Примечание. Для дальнейшего использования после завершения работы файл можно скачать.

Для открытия ранее созданной БД нужно:
-в Google Colab:

3. Connect the HOSPITAL database (in our case, create it):

```
conn = sqlite3.connect('HOSPITAL.db')
```

To see that the database has been created in Google Colab, click on the folder icon in the left panel:

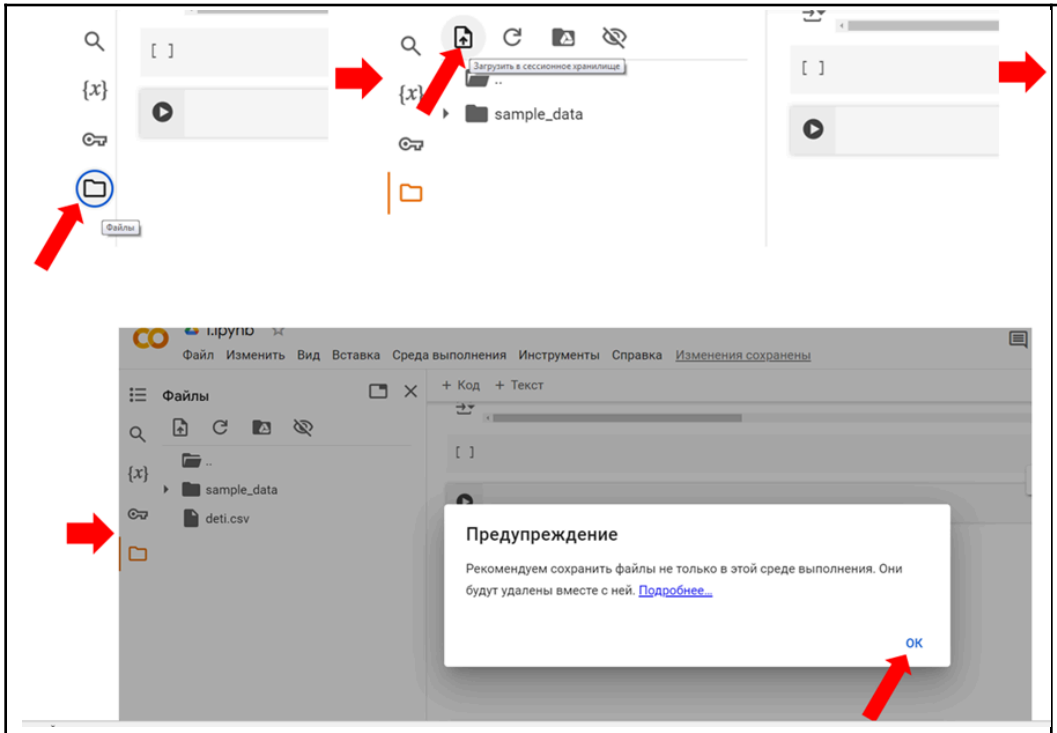


The HOSPITAL.db file should appear in the structure that opens

In Anaconda, the database file is created in the same folder where the Jupiter Notebook was created.

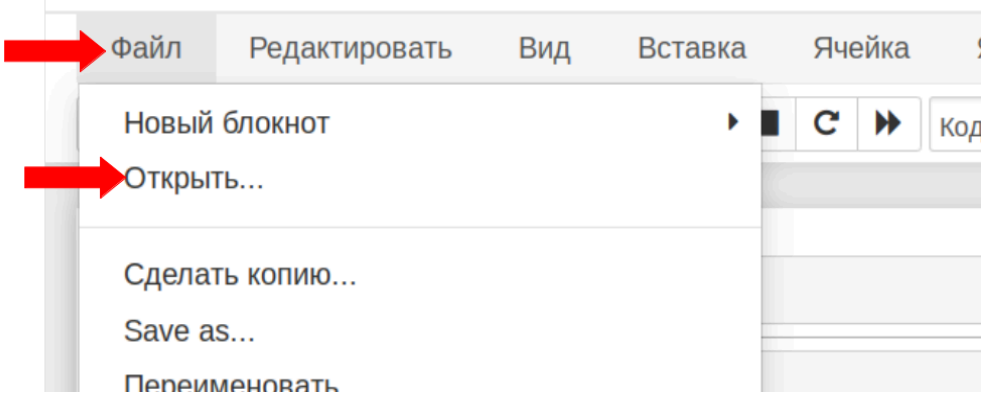
Note. You can download the file for further use after completion of the work.

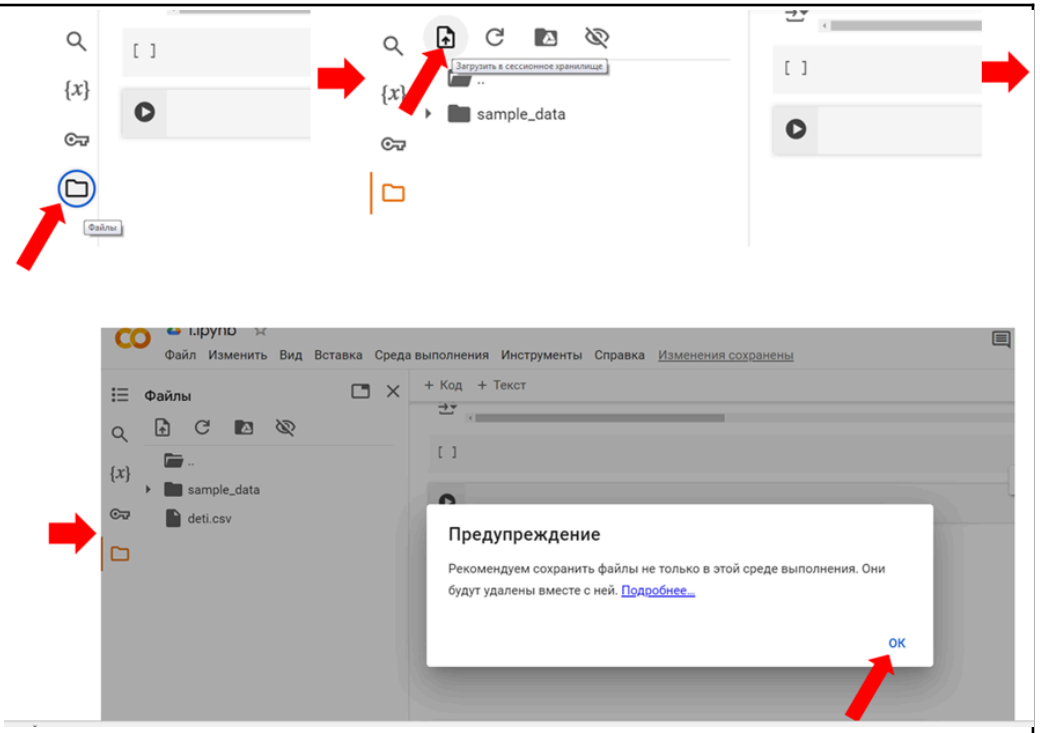
To open a previously created database, you need to:
- in Google Colab:



-в Anaconda:

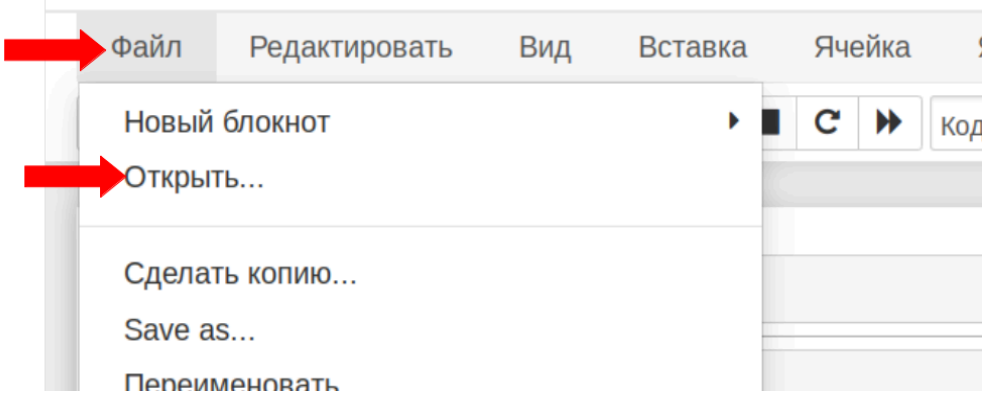
Если не указывать путь до нужной папки, то предварительно необходимо загрузить файл следующим образом:





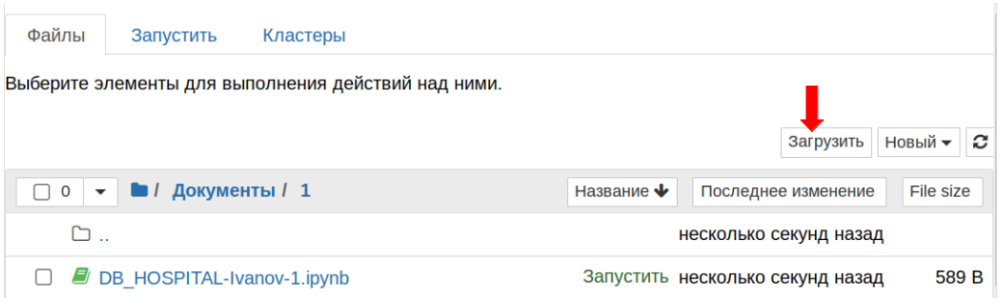
- in Anaconda:

If you do not specify the path to the desired folder, then you must first download the file as follows:

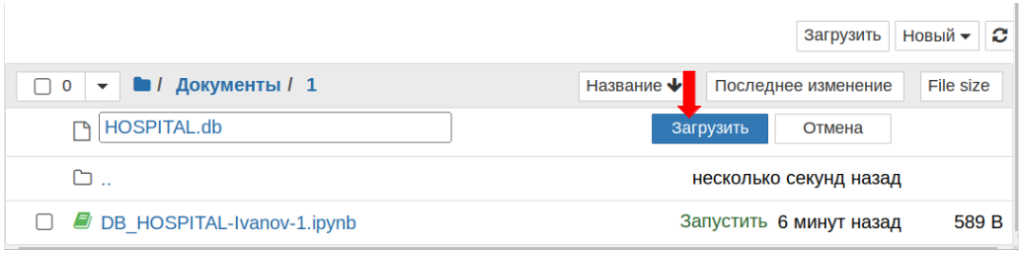


Click the “ download” button:

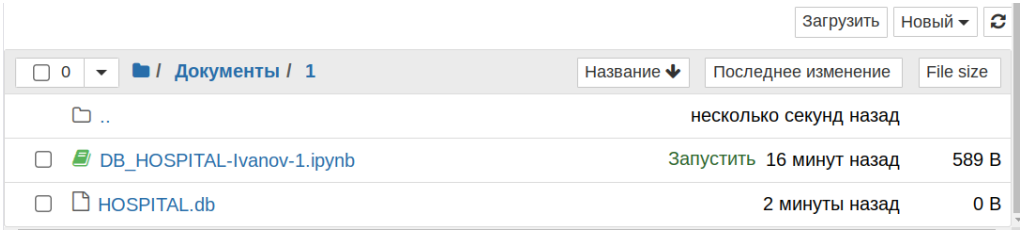
Щелкаем кнопку “Загрузить”:



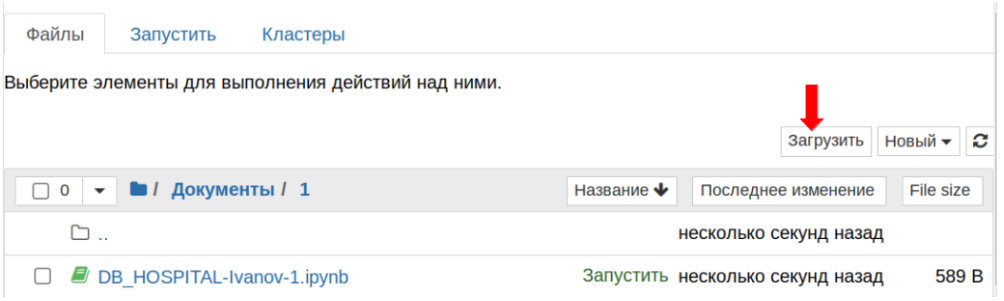
Выбираем нужный файл на компьютере и загружаем в систему:



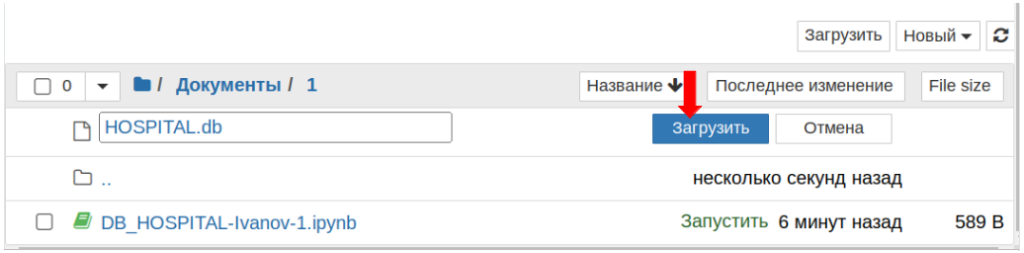
Получаем:



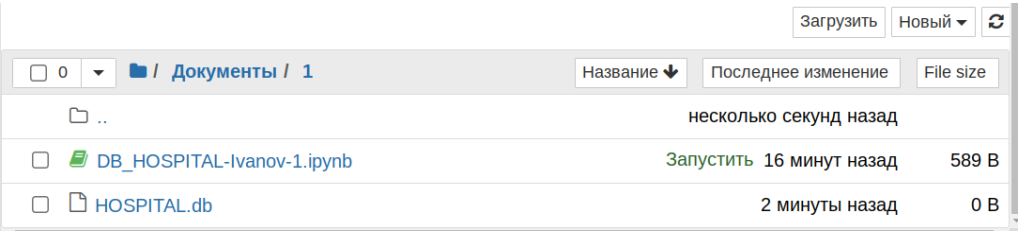
После размещения файла в среде, подключаемся к БД, как указано в начале пункта.



Select the necessary file on the computer and download it to the system:



We get:



After placing the file in the environment, we connect to the database, as indicated at the beginning of the paragraph.

4. Create database tables and fill them with data. To do this, use the following syntax:

4. Создаем таблицы БД и заполняем их данными. Для этого используем следующий синтаксис:

<Имя_переменной> = '''
Сюда копируем все запросы на создание и заполнение 4-х таблиц БД (без запроса на создание таблицы RATING). Поля этой таблицы добавляем сразу в запрос на создание таблицы DOCTORS. А в SERVICES не забываем поле patient_fk.
'''
conn.executescript(<Имя_переменной_созданной_выше>)

Примечание 1. Здесь тоже при создании таблиц БД не учитывается ссылочная целостность данных, как, например, в MS Access, но правильнее будет в начале написать все запросы на создание таблиц, учитывая очередность - какая из таблиц на какую ссылается. А затем заполнять их данными в том же порядке. Например, в такой последовательности: DEPARTMENTS -> DOCTORS -> PATIENTS -> SERVICES. Все эти запросы пишем в одной переменной.
Примечание 2. Метод .executescript() позволяет выполнять несколько SQL-запросов за один раз. Если нужно выполнить один запрос, используют метод .execute().

В качестве примера приведем запрос на создание и заполнение нескольких строк таблицы DEPARTMENTS (!!!Вам нужно написать запросы для создания и заполнения всей БД!!!).

-Помещаем в переменную (в примере – это create_database) запросы на создание и заполнение таблиц:

<Variable name> = '''

Here we copy all requests to create and fill in 4 database tables (without a request to create a RATING table). We add the fields of this table immediately to the request to create the DOCTORS table. And in SERVICES, do not forget the patient_fk field.
'''
conn.executescript(<Name of the variable Created above>)

Note 1. Here, too, when creating database tables, the referential integrity of data is not taken into account, as, for example, in MS Access, but it would be more correct to write all queries for creating tables at the beginning, taking into account the order - which of the tables refers to which. And then fill them with data in the same order. For example, in this sequence: DEPARTMENTS -> DOCTORS -> PATIENTS -> SERVICES. We write all these queries in one variable.
Note 2. Method.execute script() allows you to execute multiple SQL queries at a time. If you need to run a single query, use the method.execute().

As an example, here is a request to create and fill in several rows of the DEPARTMENTS table (!!!You need to write queries to create and populate the entire database!!!).

-We put requests for creating and filling tables in a variable (in the example, this is create_database):

```
create_database = '''

CREATE TABLE DEPARTMENTS (
  department_id INTEGER PRIMARY KEY AUTOINCREMENT,
  department_name TEXT NOT NULL,
  department_address TEXT NOT NULL,
  department_tel_no TEXT,
  number_of_beds INTEGER DEFAULT 0
);

INSERT INTO DEPARTMENTS (department_name, department_address,
                         department_tel_no, number_of_beds)
VALUES
('Administration', 'Medicsburg, Medical Street, 15', '8(999)888-77-66', 0),
('Office for the issuance, registration and storage of certificates of incapacity for work',
 'Medicsburg, Medical Street, 15 ',' 8(999)888-77-55 ', 0),
('Clinical and Expert Department', 'Medicsburg, Medical Street, 15', '8(999)888-77-11', 0),
('Admission Department', 'Medicsburg, Surgical Street, 1', '8(999)777-77-77', 0),
('Cardiology Department No. 1 with Intensive Care Unit',
 'Medicsburg, Surgical Street, 1 ',' 8(999)777-77-55 ', 30);
'''
```

-Выполняем запросы:

```
conn.executescript(create_database)
```

5. Для запросов на выборку и сортировку можно использовать следующую запись:

<Имя_переменной1> = '''
Пишем здесь запрос SQL
'''
<Имя_переменной2> =
pd.read_sql_query(<Имя_переменной1>, conn)
display(<Имя_переменной2>)
Примечание. Команду display() можно опустить, оставив только имя переменной. Также можно использовать команду print(), но тогда результат выводится как текст.

```
create_database = '''

CREATE TABLE DEPARTMENTS (
  department_id INTEGER PRIMARY KEY AUTOINCREMENT,
  department_name TEXT NOT NULL,
  department_address TEXT NOT NULL,
  department_tel_no TEXT,
  number_of_beds INTEGER DEFAULT 0
);

INSERT INTO DEPARTMENTS (department_name, department_address,
                         department_tel_no, number_of_beds)
VALUES
('Administration', 'Medicsburg, Medical Street, 15', '8(999)888-77-66', 0),
('Office for the issuance, registration and storage of certificates of incapacity for work',
 'Medicsburg, Medical Street, 15 ',' 8(999)888-77-55 ', 0),
('Clinical and Expert Department', 'Medicsburg, Medical Street, 15', '8(999)888-77-11', 0),
('Admission Department', 'Medicsburg, Surgical Street, 1', '8(999)777-77-77', 0),
('Cardiology Department No. 1 with Intensive Care Unit',
 'Medicsburg, Surgical Street, 1 ',' 8(999)777-77-55 ', 30);
'''
```

You can download this file with the code: [example]

-We are fulfilling requests:

```
conn.executescript(create_database)
```

5. For selection and sorting requests, you can use the following entry:

< Variable name1> = '''
We are writing an SQL query here
'''
< Variable name2> =
pd.read_sql_query(<Variable_name1>, conn)
display(<Variable_name2>)
Note. The display() command can be omitted, leaving only the variable name. You can also use the print() command, but then the result is output as text.

Для просмотра созданной выше таблицы DEPARTMENTS, напишем запрос на выборку данных и вывод результатов на экран в виде таблицы. Он будет иметь вид:

```
select_DEPARTMENTS = '''
SELECT * FROM DEPARTMENTS
'''

df_departments = pd.read_sql_query(select_DEPARTMENTS, conn)
display(df_departments)
```

где
select_DEPARTMENTS - переменная, содержащая SQL-запрос
df_departments - переменная, где будем хранить итоговый датафрейм (таблицу).
Получим:

	department_id	department_name	department_address	department_tel_no	number_of_beds
0	1	Administration	Medicsburg, Medical Street, 15	8(999)888-77-66	0
1	2	Office for the issuance, registration and stor...	Medicsburg, Medical Street, 15	8(999)888-77-55	0
2	3	Clinical and Expert Department	Medicsburg, Medical Street, 15	8(999)888-77-11	0
3	4	Admission Department	Medicsburg, Surgical Street, 1	8(999)777-77-77	0
4	5	Cardiology Department No. 1 with Intensive Car...	Medicsburg, Surgical Street, 1	8(999)777-77-55	30

…
Примечание. Напомним, что все приведенные запросы нужно повторить

Задание. Загрузите в переменные df_doctors, df_patients, df_services все данные по остальным таблицам.

Задание. Напишите код для выборки данных:
1) Из таблицы DOCTORS выберите всех гастроэнтерологов с рейтингом, не бо'льшим номера Вашего варианта. Результат запроса на выборку столбцов full_name_of_doctor и rating поместите в

To view the DEPARTMENTS table created above, we will write a query to select data and display the results on the screen in the form of a table. It will look like:

```
select_DEPARTMENTS = '''
SELECT * FROM DEPARTMENTS
'''

df_departments = pd.read_sql_query(select_DEPARTMENTS, conn)
display(df_departments)
```

where
select_DEPARTMENTS is a variable containing the SQL query
df_departments - a variable where we will store the final dataframe (table).
We will get:

	department_id	department_name	department_address	department_tel_no	number_of_beds
0	1	Administration	Medicsburg, Medical Street, 15	8(999)888-77-66	0
1	2	Office for the issuance, registration and stor...	Medicsburg, Medical Street, 15	8(999)888-77-55	0
2	3	Clinical and Expert Department	Medicsburg, Medical Street, 15	8(999)888-77-11	0
3	4	Admission Department	Medicsburg, Surgical Street, 1	8(999)777-77-77	0
4	5	Cardiology Department No. 1 with Intensive Car...	Medicsburg, Surgical Street, 1	8(999)777-77-55	30

…
Note. We remind you that all the above queries need to be repeated

Task. Load all data on the remaining tables into the variables df_doctors, df_patients, and df_services.

Task. Write the code for data sampling:
1) From the DOCTORS table, select all gastroenterologists with a rating that does not exceed the number of your option (variant). Put the result of the query to select the

переменную df_gastro. Отобразите на экране в виде таблицы.

В итоге может быть выведено (в зависимости от номера варианта):

	full_name_of_doctor	rating
0	Volkov Ivan Sergeevich	9
1	Isanov Petr Sergeevich	6
2	Kosina Polina Semenovna	1

2) Выведите в алфавитном порядке все отделения, расположенные на улице Микробиологическая, число койко-мест в которых, больше номера Вашего варианта. Результат запроса на выборку столбцов department_name и number_of_beds поместите в переменную df_Microbe. Отобразите на экране в виде таблицы.

В результате выполнения запроса может быть выведено (в зависимости от варианта):

	department_name	number_of_beds
0	Endocrinology Department	40
1	Gastroenterology Department	35
2	Hematology Department	25
3	Infection Department	20

3) Выведите всех пациенток и типы бонусных карт, им принадлежащих, в порядке убывания даты

full_name_of_doctor and rating columns in the df_gastro variable. Display it on the screen as a table.

As a result, it can be output (depending on the option (variant) number):

	full_name_of_doctor	rating
0	Volkov Ivan Sergeevich	9
1	Isanov Petr Sergeevich	6
2	Kosina Polina Semenovna	1

2) Print in alphabetical order all the departments located on Microbiologicheskaya Street, the number of beds in which is greater than the number of your option. Put the result of the query to select the department_name and number_of_beds columns in the df_Microbe variable. Display it on the screen as a table.

As a result of executing the query, it can be output (depending on the option):

	department_name	number_of_beds
0	Endocrinology Department	40
1	Gastroenterology Department	35
2	Hematology Department	25
3	Infection Department	20

3) Display all the patients and the types of bonus cards they own, in descending order of date of birth. The result of the

рождения. Результат запроса на выборку столбцов full_name_of_patient и bonus_card_type поместите в переменную df_female. Отобразите на экране в виде таблицы.

Результат:

	full_name_of_patient	bonus_card_type
0	Fedorova Elena Dmitrievna	gold
1	Vasilievich Darya Aleksandrovna	NULL
2	Gorinovich Elena Petrovna	classic
3	Zaytseva Natalya Mikhailovna	NULL
4	Petrova Anna Sergeevna	silver
5	Smirnovskaya Olga Petrovna	classic
6	Morozovets Natalya Andreevna	silver
7	Kuznetsova Maria Ivanovna	platinum
8	Smirnova Olga Viktorovna	classic

6. После окончания работы с SQLite обязательно закрываем соединение с БД:

```
conn.close()
```

Примечание. Теперь можно работать с полученными данными, как с датафреймами (для более полного ознакомления с библиотекой pandas см. лабораторную работу “Библиотека pandas и анализ данных”).

7. Визуализируем найденную информацию при помощи библиотек matplotlib и seaborn. Вначале импортируем их.

query to select the columns full_name_of_patient and bonus_card_type should be placed in the variable df_female. Display it on the screen as a table.

Result:

	full_name_of_patient	bonus_card_type
0	Fedorova Elena Dmitrievna	gold
1	Vasilievich Darya Aleksandrovna	NULL
2	Gorinovich Elena Petrovna	classic
3	Zaytseva Natalya Mikhailovna	NULL
4	Petrova Anna Sergeevna	silver
5	Smirnovskaya Olga Petrovna	classic
6	Morozovets Natalya Andreevna	silver
7	Kuznetsova Maria Ivanovna	platinum
8	Smirnova Olga Viktorovna	classic

8. After finishing working with SQLite, be sure to close the database connection:

```
conn.close()
```

Note. Now you can work with the received data as with dataframes (for a more complete introduction to the pandas library, see the laboratory work “Pandas Library and data Analysis”).

8. Visualize the information found using the matplotlib and seaborn libraries. First, we import them.

Библиотеки в Python можно подключать в любом месте программы, но принято делать это в самом начале, поэтому вернемся к первой ячейке блокнота и после нее добавим и запустим еще одну ячейку:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Примечание. Как и pd для pandas, здесь plt и sns - алиасы (псевдонимы) для соответствующих библиотек, которые будем использовать для обращения к этим библиотекам.

Теперь вернемся к последней ячейке и продолжим работу.

➢ Но вначале проанализируем статистику по таблице DEPARTMENTS:

```
df_departments.describe()
```

Получим:

	department_id	number_of_beds
count	22.000000	22.000000
mean	11.500000	23.863636
std	6.493587	15.881392
min	1.000000	0.000000
25%	6.250000	5.000000
50%	11.500000	30.000000
75%	16.750000	35.000000
max	22.000000	45.000000

Libraries in Python can be connected anywhere in the program, but it is customary to do this at the very beginning, so let's go back to the first cell of the notebook and after it add and run another cell:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Note. Like pd for pandas, here plt and sns are aliases for the corresponding libraries that will be used to access these libraries.

Now let's go back to the last cell and continue working.

➢ But first , let's analyze the statistics on the DEPARTMENTS table:

```
df_departments.describe()
```

We will get:

	department_id	number_of_beds
count	22.000000	22.000000
mean	11.500000	23.863636
std	6.493587	15.881392
min	1.000000	0.000000
25%	6.250000	5.000000
50%	11.500000	30.000000
75%	16.750000	35.000000
max	22.000000	45.000000

Команда выдает отдельные статистические сведения по столбцам с числовыми данными. Подобная информация по department_id, очевидно, ни о чем нам не скажет, а вот по number_of_beds можно понять, что: ⩒ количество непустых значений count = 22; ⩒ среднее число койко-мест по отделениям mean = 23,863636 (вряд ли имеет смысл в данном контексте); ⩒ среднее квадратическое отклонение (разброс, отклонение от среднего), str = 15.881392; ⩒ min = 0, max = 45 (а вот это уже важная информация) ⩒ процентили 25%, 50%, 75% подскажут, соответствующие границы значений, т.е. 25% отделений имеют не более 5 койко-мест (собственно, сюда попали все отделения не предполагающие госпитализацию), 50% - не более 30 и 75% - не более 35 койко-мест.	The command outputs separate statistical information for columns with numeric data. Detailed information on department_id, obviously, will not tell us anything, but from number_of_beds we can understand that: v the number of non-empty values count = 22; v average number of beds by department mean = 23,863636 (hardly makes sense in this context); v standard deviation (spread, deviation from the mean), str = 15.881392; v min = 0, max = 45 (and this is important information) The v percentiles of 25%, 50%, 75% will tell you the appropriate limits of values, i.e. 25% of departments have no more than 5 beds (actually, all departments that do not involve hospitalization got here), 50% - no more than 30 and 75% - no more than 35 beds.
Задание. Проанализируйте статистику по остальным датафреймам, полученным по таблицам БД HOSPITAL.	***Task.*** Analyze the statistics for the rest of the dataframes obtained from the DB_HOSPITAL tables
Примечание. В Google Colab, щелкнув по значку с графиком справа от выведенной таблицы, можно получить некоторую визуализацию полученных данных. Вообще Colab предлагает собственные варианты визуализации для таблиц, которые можно посмотреть и выбрать подходящий график или видоизменить его, при необходимости, исправив предложенный код.	*Note.* In Google Colab, by clicking on the graph icon to the right of the resulting table, you can get some visualization of the data obtained. In general, Ecolab offers its own visualization options for tables, which you can view and select a suitable graph or modify it, if necessary, by correcting the proposed code.
➢ Построим линейчатую диаграмму по количеству отделений, находящихся по каждому из адресов (по датафрейму df_departments):	➢ Let's build a bar chart based on the number of departments located at each of the addresses (according to the df_departments dataframe):

```
df_departments.groupby('department_address').size().plot(kind='barh',
                                                         color=sns.cubehelix_palette(10))

plt.xlabel ('Number of departments', fontsize = 14)
plt.ylabel ('Department address', fontsize = 14)
plt.title ('Number of departments by address', fontsize = 16)
plt.gca().spines[['top', 'right']].set_visible(False)
plt.show()
```

Здесь

`.groupby().size()` - сочетание методов группировки и определения количества элементов позволяет сгруппировать отделения по их количеству, расположенному по каждому из адресов.

`.plot()` - построение графика:

- `kind` - вид графика,
  - `kind='barh'` - гистограмма с горизонтальным расположением
- `color` - цвет
  - `color=sns.cuberhelix_palette(10)` - выбираем палитру из библиотеки seaborn, но можно указать один цвет в виде английского эквивалента (например, `color='blue'`) или значения в формате RGB (например, `color='#ff00ff'`).

`plt.xlabel(), plt.ylabel(), plt.title()` - подписи осей x, y и заголовок гистограммы, соответственно. Первый параметр - текст подписи, второй - размер шрифта.

`plt.gca().spines[['top','right']].set_visible(False)` - убираем (делаем невидимыми) верхнюю и левую границы области построения диаграммы.

`plt.show()` - отображение графика. В Jupyter Notebook необязателен, но желателен.

```
df_departments.groupby('department_address').size().plot(kind='barh',
                                                         color=sns.cubehelix_palette(10))

plt.xlabel ('Number of departments', fontsize = 14)
plt.ylabel ('Department address', fontsize = 14)
plt.title ('Number of departments by address', fontsize = 16)
plt.gca().spines[['top', 'right']].set_visible(False)
plt.show()
```

Here

`.groupby().size()` - the combination of methods of grouping and determining the number of elements that allows you to group departments by their number, located at each of the addresses.

`.plot()` – building a graph:

- `kind` - type of graph,
  - `kind='barh'` - a histogram with a horizontal arrangement
- `color` - цвет
  - `color=sns.cuberhelix_palette(10)` – we select a palette from the seaborn library, but you can specify one color as the English equivalent (for example, `color='blue'`) or values in RGB format (for example, `color='#ff00ff'`).

`plt.xlabel(), plt.ylabel(), plt.title()` – the labels of the x, y axes and the title of the histogram, respectively. The first parameter is the signature text, the second is the font size.

`plt.gca().spines[['top','right']].set_visible(False)` - we remove (make invisible) the upper and left borders of the chart plotting area.

`plt.show()` - displaying a graph. In Jupiter Notebook it is optional, but desirable.

➢ Построим по датафрейму df_gastro столбчатую диаграмму по рейтингу гастроэнтерологов:

```
plt.figure(figsize=(4, 6))
plt.bar(df_gastro['full_name_of_doctor'], df_gastro['rating'], color='#555fff')

plt.xticks(rotation=90)
plt.xlabel('Full name of doctor', fontsize=14)
plt.ylabel('Rating', fontsize=14)
plt.title('Gastroenterology rating', fontsize=16)
plt.gca().spines[['top', 'right',]].set_visible(False)
plt.show()
```

Здесь
`plt.figure()` задает контейнер для диаграммы с размерами сетки figuresize, где первый параметр - размер по оси x, второй - по y.
`plt.bar()` строит столбчатую гистограмму, где первый параметр - значения по оси x (), второй - по оси y
`plt.xticks()` задает выравнивание текста подписей по оси x (по умолчанию - выравнивание по горизонтали; при rotation=90 текст подписи располагается вертикально)

➢ По датафрейму df_Microbe постройте следующую линейчатую диаграмму, отображающую число койко-мест по отделениям, расположенным на Микробиологической улице:

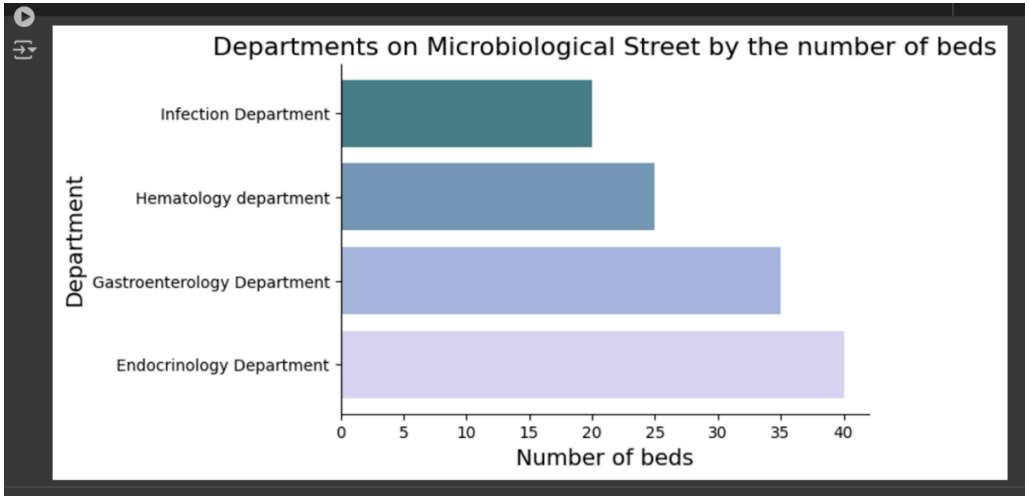
➢ Let 's build a bar chart based on the df_gastro dataframe according to the rating of gastroenterologists:

```
plt.figure(figsize=(4, 6))
plt.bar(df_gastro['full_name_of_doctor'], df_gastro['rating'], color='#555fff')

plt.xticks(rotation=90)
plt.xlabel('Full name of doctor', fontsize=14)
plt.ylabel('Rating', fontsize=14)
plt.title('Gastroenterology rating', fontsize=16)
plt.gca().spines[['top', 'right',]].set_visible(False)
plt.show()
```

Here
`plt.figure()` sets a container for a diagram with the dimensions of the figuresize grid, where the first parameter is the size on the x axis, the second one is on the y axis.
`plt.bar()` builds a columnar histogram, where the first parameter is the values on the x axis (), the second is on the y axis
`plt.xticks()` sets the alignment of the caption text along the x axis (by default, horizontal alignment; with rotation=90, the caption text is positioned vertically)

➢ Using the df_Microbe dataframe, build the following bar chart showing the number of beds distributed in the departments which are located in Microbiological Street:



Цветовая палитра:

```
color=sns.cubehelix_palette(start=2)
```

➢ Построим круговую диаграмму по датафрейму df_female - по типам бонусных карт у пациенток клиники:

```
plt.figure(figsize=(8, 8))
df_female.groupby('bonus_card_type').size().plot(kind='pie',
                                                 autopct='%1.1f%%',
                                                 startangle=90,
                                                 colors=sns.color_palette('pastel')[0:10],
                                                 wedgeprops=dict(linewidth=2, edgecolor='black'))
plt.axis('equal')
plt.title('Share of bonus cards by type in clinic female patients')
plt.ylabel('')
plt.show()
```

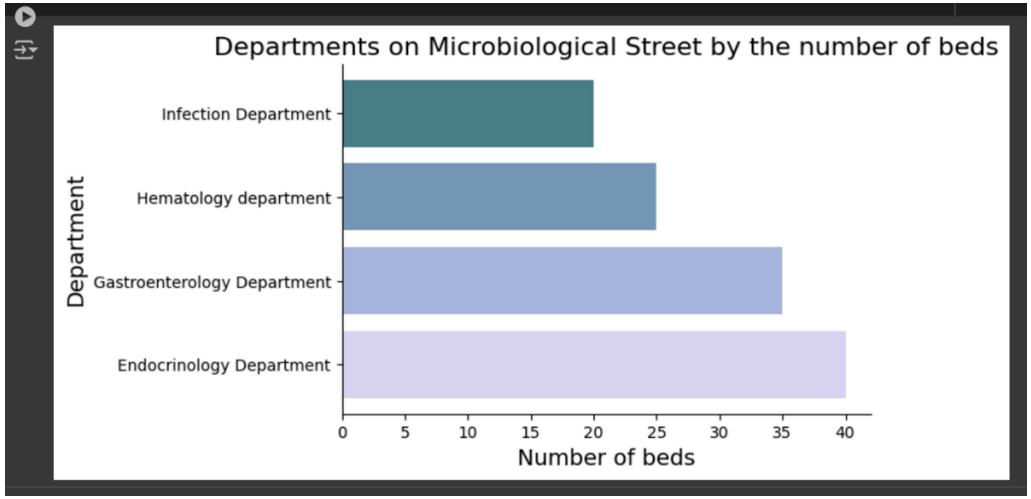
Из неизвестных параметров здесь:

`kind='pie'` - тип диаграммы - круговая (пирог);

`autopct` - задает процент для каждого сектора;

`startangle = 90` - устанавливает начальный угол, чтобы улучшить внешний вид диаграммы;

`colors` - задает цвета секторов;



The color palette:

```
color=sns.cubehelix_palette(start=2)
```

➢ Let 's build a pie chart based on the df_female dataframe - by types of bonus cards for clinic female patients:

```
plt.figure(figsize=(8, 8))
df_female.groupby('bonus_card_type').size().plot(kind='pie',
                                                 autopct='%1.1f%%',
                                                 startangle=90,
                                                 colors=sns.color_palette('pastel')[0:10],
                                                 wedgeprops=dict(linewidth=2, edgecolor='black'))
plt.axis('equal')
plt.title('Share of bonus cards by type in clinic female patients')
plt.ylabel('')
plt.show()
```

The unknown parameters here are:

`kind='pie'` - the chart type is circular (pie);

`autopct` - sets the percentage for each sector;

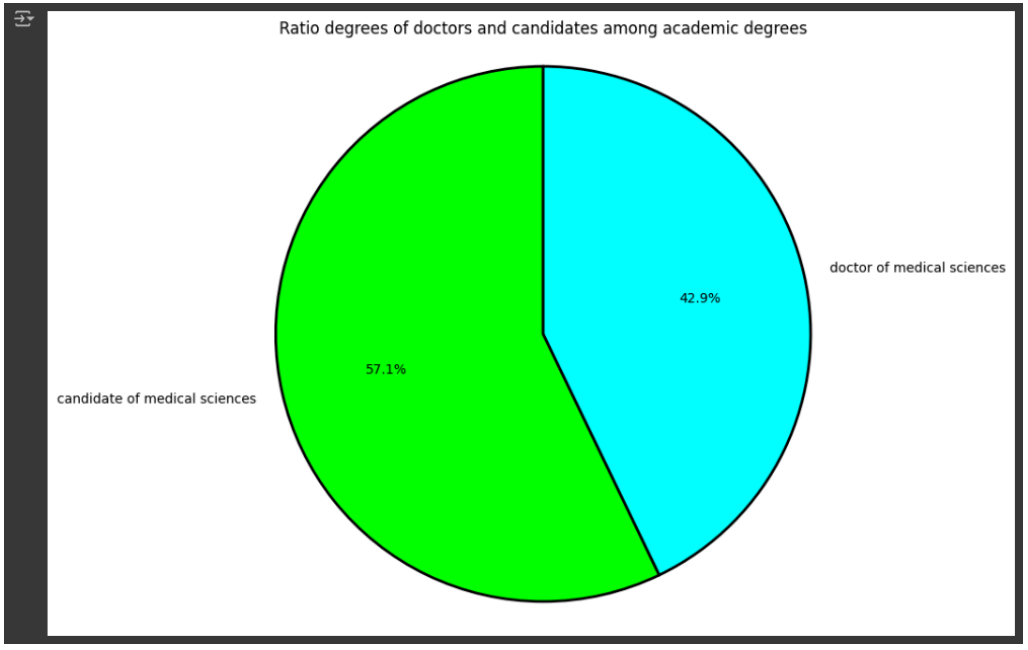
`startangle = 90` - sets the initial angle to improve the appearance of the chart;

`colors` - sets the colors of the sectors;

`wedgeprops` - sets the border parameters for the chart. The parameters are listed in the dictionary: dict:

`wedgeprops` - задает параметры границ для диаграммы. Параметры перечисляются в словаре: dict:
`linewidth` - толщина линии,
`edgecolor` - цвет линии;
`plt.axis('equal')` - равные размеры по осям, позволяет изобразить круг, а не овал;
`plt.ylabel('')` - убирает подпись оси y.

➢ По датафрейму df_doctors постройте круговую диаграмму отражающую соотношение ученых степеней у врачей клиники. Оформите как на рисунке ниже.

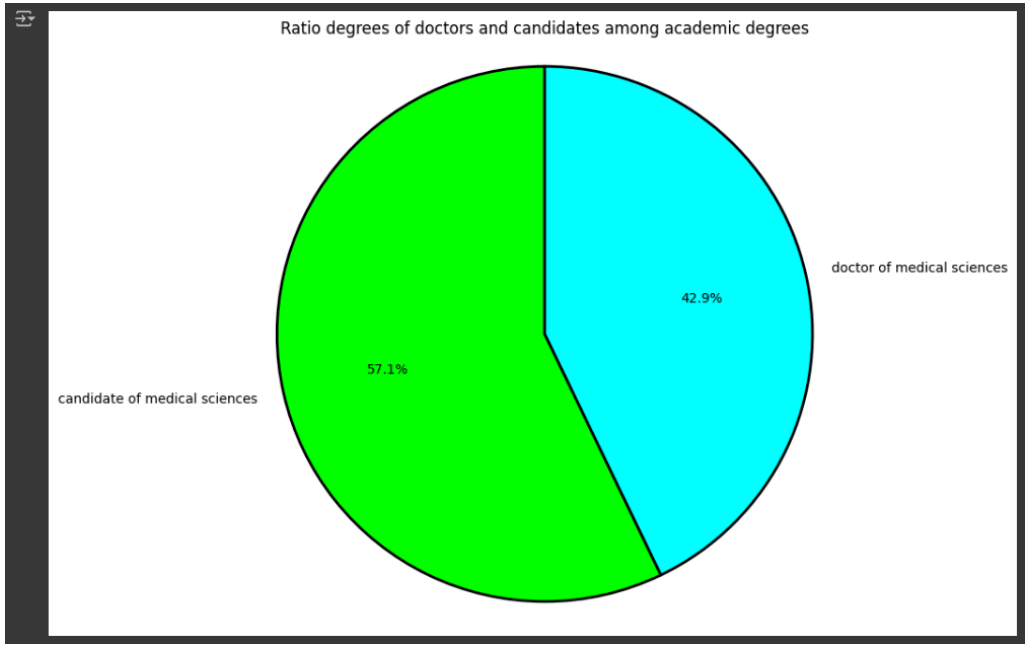


Для задания этих или своих цветов до кода для построения диаграммы нужно создать переменную, например,

```
colors = ['lime', 'aqua']
```

`linewidth` - line thickness,
`edgecolor` - line color;
`plt.axis('equal')` - equal dimensions along the axes, allows you to draw a circle, not an oval;
`plt.ylabel('')` - removes the y-axis signature.

➢ Using the df_doctors dataframe, build a pie chart reflecting the ratio of academic degrees among the clinic's doctors. Arrange it as shown in the picture below.
The ratio of academic degrees of the doctors of the clinic



To set these or your own colors before the code for building the diagram, you need to create a variable, for example,

```
colors = ['lime', 'aqua']
```

and then assign it to the corresponding chart parameter:

```
colors=colors,
```

а затем присвоить ее соответствующему параметру диаграммы:

```
colors=colors,
```

Использованная литература.

1. Что такое база данных? // https://www.oracle.com/cis/database/what-is-database/
2. Интерактивный тренажер по SQL // https://stepik.org/course/63054/syllabus
3. Руководство по SQLite // https://metanit.com/sql/sqlite/
4. Руководство по проектированию реляционных баз данных // https://metanit.com/sql/tutorial/
5. Схемы баз данных // https://translated.turbopages.org/proxy_u/en-ru.ru.5509b082-6720c80c-65148dfb-74722d776562/https/www.geeksforgeeks.org/database-schemas/
6. Основы SQL // https://sberuniversity.online/
7. Сборник индивидуальных заданий по технологиям баз данных: Учеб.-практ. пособие / В.С. Оскерко, О.А. Сосновский, О.А. Лаврова и др. – Мн.: БГЭУ, 2005. – 65 с.
8. Chat GPT
9. Tproger // https://tproger.ru/
10. Груздев А.В., Хейдт М. Изучаем pandas. - М. : ДМК Пресс, 2019. - 700с.
11. Пасхавер Б. Pandas в действии. - СПб. : Питер, 2023. - 512с.
12. Болье А. Изучаем SQL. Генерация, выборка и обработка данных - СПб. : Питер, 2020. - 400с.
13. Учебник по SQLite3 в Python // https://digitology.tech/posts/uchebnik-po-sqlite3-v-python/

List of literature.

1. Что такое база данных? // https://www.oracle.com/cis/database/what-is-database/
2. Интерактивный тренажер по SQL // https://stepik.org/course/63054/syllabus
3. Руководство по SQLite // https://metanit.com/sql/sqlite/
4. Руководство по проектированию реляционных баз данных // https://metanit.com/sql/tutorial/
5. Схемы баз данных // https://translated.turbopages.org/proxy_u/en-ru.ru.5509b082-6720c80c-65148dfb-74722d776562/https/www.geeksforgeeks.org/database-schemas/
6. Основы SQL // https://sberuniversity.online/
7. Сборник индивидуальных заданий по технологиям баз данных: Учеб.-практ. пособие / В.С. Оскерко, О.А. Сосновский, О.А. Лаврова и др. – Мн.: БГЭУ, 2005. – 65 с.
8. Chat GPT
9. Tproger // https://tproger.ru/
10. Груздев А.В., Хейдт М. Изучаем pandas. - М. : ДМК Пресс, 2019. - 700с.
11. Пасхавер Б. Pandas в действии. - СПб. : Питер, 2023. - 512с.
12. Болье А. Изучаем SQL. Генерация, выборка и обработка данных - СПб. : Питер, 2020. - 400с.
13. Учебник по SQLite3 в Python // https://digitology.tech/posts/uchebnik-po-sqlite3-v-python/
