

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

С. В. Рындина

**ЦИФРОВЫЕ ТЕХНОЛОГИИ УПРАВЛЕНИЯ
ПОЛУЧЕНИЕМ, ХРАНЕНИЕМ, ПЕРЕДАЧЕЙ
И ОБРАБОТКОЙ БОЛЬШИХ ДАННЫХ:
SQLite**

Учебно-методическое пособие

ПЕНЗА 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Пензенский государственный университет» (ПГУ)

С. В. Рындина

Цифровые технологии управления получением, хранением, передачей и обработкой больших данных: SQLite

Учебно-методическое пособие

Пенза
Издательство ПГУ
2023

УДК 004.6 (075)

P95

Р е ц е н з е н т

кандидат технических наук, доцент

А. А. Масленников

Рындина, Светлана Валентиновна.

P95

Цифровые технологии управления получением, хранением, передачей и обработкой больших данных: SQLite : учеб.-метод. пособие / С. В. Рындина. – Пенза : Изд-во ПГУ, 2023. – 82 с.

Рассмотрены возможности языка составления запросов к реляционной базе данных SQL на основе системы управления базами данных SQLite. Приведены задания для лабораторных работ по созданию запросов к базе данных для решения бизнес-задач.

Издание подготовлено на кафедре «Цифровая экономика» и предназначено для обучающихся по направлению подготовки 38.03.05 «Бизнес-информатика», соответствует программе дисциплины «Цифровые технологии управления получением, хранением, передачей и обработкой больших данных», может быть использовано также при изучении дисциплины «Бизнес-аналитика на основе больших данных» и написании выпускной работы бакалавра.

УДК 004.6 (075)

© Пензенский государственный
университет, 2023

СОДЕРЖАНИЕ

Введение	4
1. Настройка программного окружения	5
1.1. Реляционная база данных SQLite	5
1.2. Работа в терминале SQLite (sqlite3). Создание базы данных и загрузка данных из файла .csv	8
1.3. Менеджер баз данных DBeaver	17
1.4. Работа с SQLite в Python	29
2. Работа с данными	43
2.1. Операции DDL	44
2.2. Операции DML: INSERT, UPDATE, DELETE	52
2.3. Операции DML: SELECT	55
2.3.1. Условие WHERE, операторы сравнения LIKE и IN	57
2.3.2. Исключение дублей, сортировка, группировка и агрегации	61
2.3.3. Соединения	70
3. Лабораторные работы	78
3.1. Лабораторная работа «Настройка программного обеспечения»	78
3.2. Лабораторная работа «Операции DDL»	78
3.3. Лабораторная работа «Операции DML»	79
Список литературы	81

Введение

В соответствии с учебным планом студенты четвертого курса направления подготовки 38.03.05 «Бизнес-информатика» изучают дисциплину «Цифровые технологии управления получением, хранением, передачей и обработкой больших данных».

Важным элементом аналитики является получение данных из различных источников для последующего анализа и интерпретации.

В учебно-методическом пособии рассматривается, как выполнять запросы к базе данных SQLite, как настраивать программное окружение, как решать бизнес-проблемы с помощью запросов и переходить от бизнес-задач к формулированию последовательности запросов, а при получении результата интерпретировать его с точки зрения полученного решения бизнес-проблемы.

В пособии используются материалы ресурсов [1, 2]. Есть великолепный форум по SQLite [3]. Пособие автора по SQL на примере Postgres [4].

1. Настройка программного окружения

Для того чтобы работать с данными необходимо их откуда-то получать, где-то хранить.

1.1. Реляционная база данных SQLite

Система управления реляционными базами данных (СУБД или СУБД) SQLite относится к open source и является встраиваемой, т.е. не использующей парадигму клиент-сервер.

В большинстве современных СУБД с клиент-серверной архитектурой имеется тяжеловесный серверный пакет (ядро базы данных) и библиотеки клиентского программного обеспечения, которые должны быть интегрированы в клиентское приложение для получения доступа к серверу базы данных (рис. 1.1).

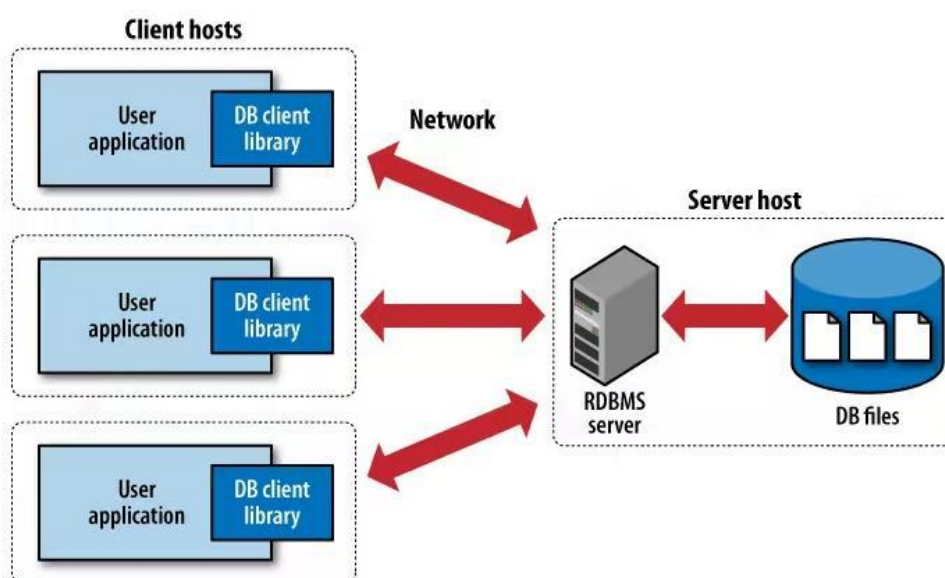


Рис. 1.1. Клиент-серверная архитектура СУБД [1]

Экземпляр базы данных обычно состоит из большого количества файлов, организованных в одно или несколько деревьев каталогов в файловой системе сервера.

Отсутствие архитектуры клиент-сервер у СУБД SQLite позволяет хранить всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором выполняется программа. Библиотека SQLite напрямую обращается к файлам своего хранилища, которые являются кросс-

платформенными. Единая библиотека содержит всю систему базы данных, которая интегрируется непосредственно в хост-приложение.

То есть движок базы данных не организует отдельный процесс, с которым взаимодействуют приложения, а становится составной частью программы (рис. 1.2).

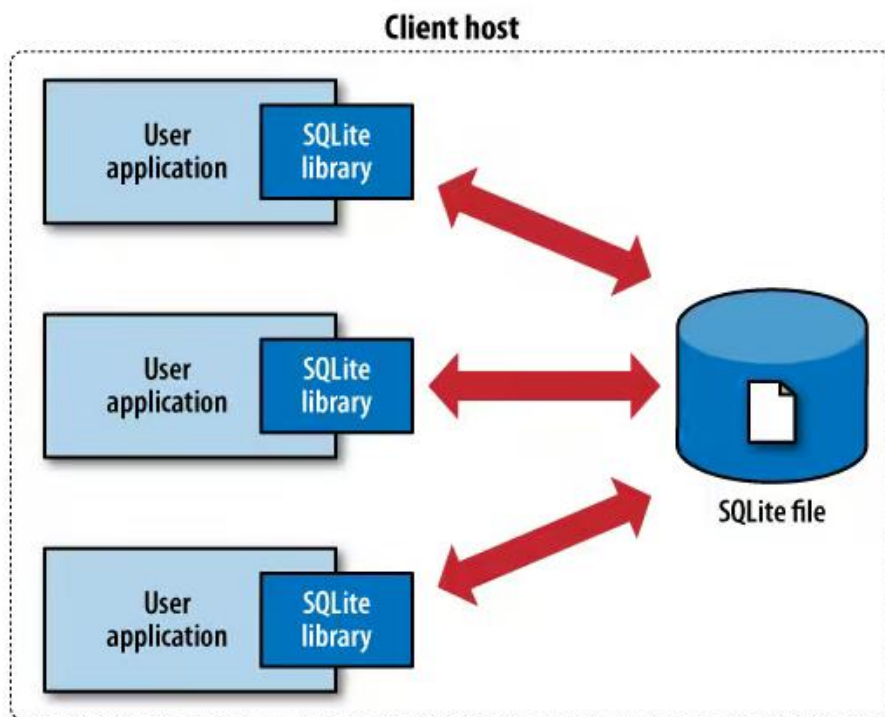


Рис. 1.2. Бессерверная архитектура SQLite [1]

В качестве протокола обмена используются вызовы функций (API, Application Programming Interface, программный интерфейс приложения) библиотеки SQLite.

Отсутствие клиент-серверной архитектуры накладывает ряд ограничений на СУБД SQLite. Скорость обработки транзакций, которую обеспечивает SQLite – средняя, так как обрабатывать несколько транзакций параллельно без риска потери данных возможно только при наличии сервера.

Так как SQLite помещает все в один файл, то для больших объемов данных размер файла может составлять гигабайт и более, а его обработка может напрямую зависеть от возможностей операционной системы оперировать такими тяжеловесными файлами.

Данные аутентификации или авторизации в СУБД SQLite отсутствуют. Вместо этого можно настроить разрешения файловой системы для управления доступом к необработанному файлу базы данных. Настройки доступа ограничены тремя вариантами: полный доступ для

чтения/записи, доступ только для чтения или отсутствие доступа вообще. Доступ для записи является абсолютным и позволяет изменять как данные, так и структуру самой базы данных.

Базовый механизм авторизации на уровне приложения реализуется через API SQLite, но его легко обойти, если у пользователя есть прямой доступ к файлу базы данных. В связи с этим SQLite нельзя использовать для хранилищ конфиденциальных данных, требующих индивидуального контроля доступа.

Также SQLite не имеет внутренней поддержки репликации, но копируя файл базы данных, можно создать реплику самостоятельно.

Перейдем к установке SQLite (рис. 1.3).

SQLite Small. Fast. Reliable. Choose any three.

Home About Documentation Download License Support Purchase Search

SQLite Download Page

Source Code

[sqlite-amalgamation-3410000.zip](#) (2.50 MiB) C source code as an [amalgamation](#), version 3.41.0. (SHA3-256: 052c721508b208403ab5ba939ee3cead56a8bd03225ffc778948e3c59c83b7ee)

[sqlite-autoconf-3410000.tar.gz](#) (2.98 MiB) C source code as an [amalgamation](#). Also includes a "configure" script and [TEA](#) makefiles for the [TCL](#) Interface. (SHA3-256: d783ab44a2b44394331d392b8b8d4d2ea4964cbb2bfc7c6c649bcbfdb3c9ffe)

Documentation

[sqlite-doc-3410000.zip](#) (10.47 MiB) Documentation as a bundle of static HTML files. (SHA3-256: 61be0faa6599bc8ac2596b3be3db3edc0d6b5fc5d0a8f0b6b2f897ea22dedae4)

Precompiled Binaries for Android

[sqlite-android-3410000.aar](#) (3.26 MiB) A precompiled Android library containing the core SQLite together with appropriate Java bindings, ready to drop into any Android Studio project. (SHA3-256: 20a7da773fdb2560f90cfe4f8c088b930df905039b73c2db3a893cb972700884)

Precompiled Binaries for Linux

[sqlite-tools-linux-x86-3410000.zip](#) (2.16 MiB) A bundle of command-line tools for managing SQLite database files, including the [command-line shell](#) program, the [sqldiff](#) program, and the [sqlite3_analyzer](#) program. (SHA3-256: 7f6601bfcd9ae63ff12d0136d71f41468618342e944c6777f0b3c5ec35646e37)

Precompiled Binaries for Mac OS X (x86)

[sqlite-tools-osx-x86-3410000.zip](#) (1.54 MiB) A bundle of command-line tools for managing SQLite database files, including the [command-line shell](#) program, the [sqldiff](#) program, and the [sqlite3_analyzer](#) program. (SHA3-256: 11a765584ee7819b5b734f95c89f788bcacf68912c7ca13082758181034f3ce5)

Precompiled Binaries for Windows

[sqlite-dll-win32-x86-3410000.zip](#) (562.90 KiB) 32-bit DLL (x86) for SQLite version 3.41.0. (SHA3-256: 4494f81f53e713fe17823088ee3574db21a7fde6d477916b9c8bb7d2e293c93e)

Рис. 1.3. Страница загрузки установочных файлов для SQLite

Дистрибутивы доступны по ссылке <https://www.sqlite.org/download.html/>

По первой ссылке скачивается архив из четырех файлов объединенной конфигурации для Windows (рис. 1.3):

- `sqlite3.c` – это исходный файл объединения, который включает все ядро SQLite, а также общие расширения;
- `sqlite3.h` – заголовочный файл объединения, который предоставляет основной API;
- `sqlite3ext.h` – файл заголовка расширения, который используется для создания расширений SQLite;
- `shell.c` – источник приложения `sqlite3`, который предоставляет интерактивную оболочку командной строки.

Для интеграции SQLite в большинство приложений достаточно `sqlite3.c` и `sqlite3.h`, которые нужно скопировать в исходный каталог приложения.

Для работы с SQLite в интерфейсе командной строки скачаем архив `sqlite-tools-win32-x86-3410000.zip` (рис. 1.4) и распакуем его в созданную папку на диске C (`C:\sqlite`).

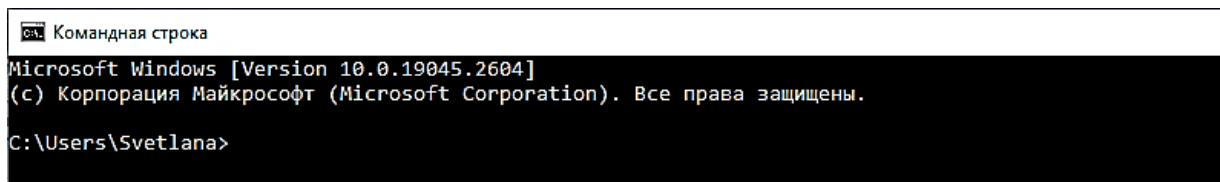
Precompiled Binaries for Windows

<u>sqlite-dll-win32-x86-3410000.zip</u> (562.90 KiB)	32-bit DLL (x86) for SQLite version 3.41.0. (SHA3-256: 4494f81f53e713fe17823088ee3574db21a7fde6d477916b9c8bb7d2e293c93e)
<u>sqlite-dll-win64-x64-3410000.zip</u> (903.85 KiB)	64-bit DLL (x64) for SQLite version 3.41.0. (SHA3-256: e58b92f93d54348cea1351479719f9685fc86a043841b569bc3528b8b2ba1536)
<u>sqlite-tools-win32-x86-3410000.zip</u> (1.91 MiB)	A bundle of command-line tools for managing SQLite database files, including the <u>command-line shell</u> program, the <u>sqldiff.exe</u> program, and the <u>sqlite3_analyzer.exe</u> program. (SHA3-256: 94c8e42e1cc9cb92a3781dbbcd36d3a3227e94ebee6e0ff7aa12fce78a745210)

Рис. 1.4. Исходные установочные файлы для работы в операционной системе Windows

1.2. Работа в терминале SQLite (sqlite3). Создание базы данных и загрузка данных из файла .csv

Для запуска терминала SQLite (`sqlite3`) необходимо вызвать командную строку ОС Windows. Для вызова командной строки ОС Windows можно использовать разные способы. Сделаем это через диалоговое окно поиска. В зависимости от версии ОС Windows можно увидеть опцию Поиск в меню Пуск, при его отсутствии, правой кнопкой мыши кликнув по Панели задач, вызываем контекстное меню и выбираем опцию Показать поле поиска, в котором набираем искомое приложение `cmd`, а затем запускаем его (рис. 1.5).



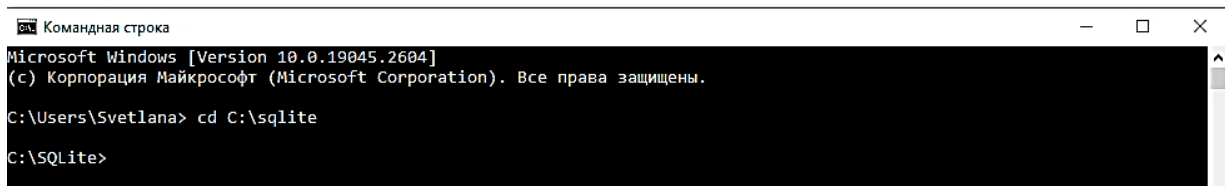
```
Командная строка
Microsoft Windows [Version 10.0.19045.2604]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Svetlana>
```

Рис. 1.5. Запуск Командной строки

Изменяем рабочую директорию (рис. 1.6):

```
cd C:\sqlite
```



```
Командная строка
Microsoft Windows [Version 10.0.19045.2604]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

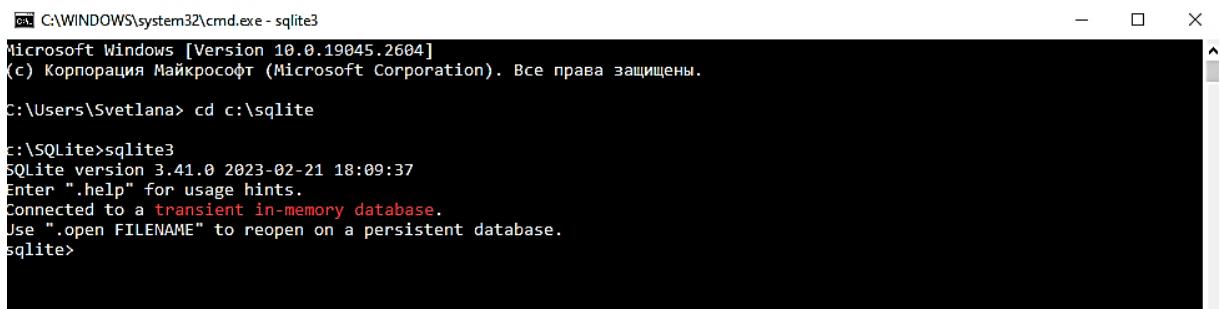
C:\Users\Svetlana> cd C:\sqlite
C:\SQLite>
```

Рис. 1.6. Изменение рабочей директории

Обратите внимание, что регистр имен неважен (SQLite, sqlite и т.д. будет интерпретироваться одинаково).

Запускаем терминал СУБД SQLite (рис. 1.7):

```
sqlite3
```

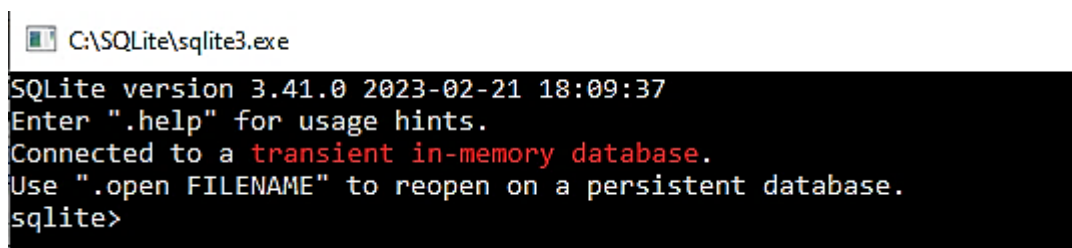


```
C:\WINDOWS\system32\cmd.exe - sqlite3
Microsoft Windows [Version 10.0.19045.2604]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Svetlana> cd c:\sqlite
c:\SQLite>sqlite3
SQLite version 3.41.0 2023-02-21 18:09:37
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

Рис. 1.7. Запуск терминала SQLite

Терминал SQLite можно запускать и из папки на диске, отправляя на выполнение файл sqlite3.exe (рис. 1.8).



```
C:\SQLite\sqlite3.exe
SQLite version 3.41.0 2023-02-21 18:09:37
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

Рис. 1.8. Запуск исполнимого файла sqlite3.exe

Можно (но совсем необязательно) внести в число переменных окружения ОС Windows переменную C:\sqlite и тогда смена рабочей директории будет не нужна, а терминал будет запускаться сразу при обращении к sqlite3 (рис. 1.9).

```
C:\WINDOWS\system32\cmd.exe - sqlite3
Microsoft Windows [Version 10.0.19045.2604]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Svetlana>sqlite3
SQLite version 3.41.0 2023-02-21 18:09:37
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

Рис. 1.9. Запуск терминала для SQLite при внесении изменений в переменные окружения

Для доступа к переменным окружения воспользуемся, как и раньше, поиском в меню Пуск или возможностью поиска на Панели задач и наберем в поле поиска sysdm.cpl. Появится диалоговое окно Свойства системы. На вкладке Дополнительно кликнем кнопку Переменные среды (рис. 1.10).

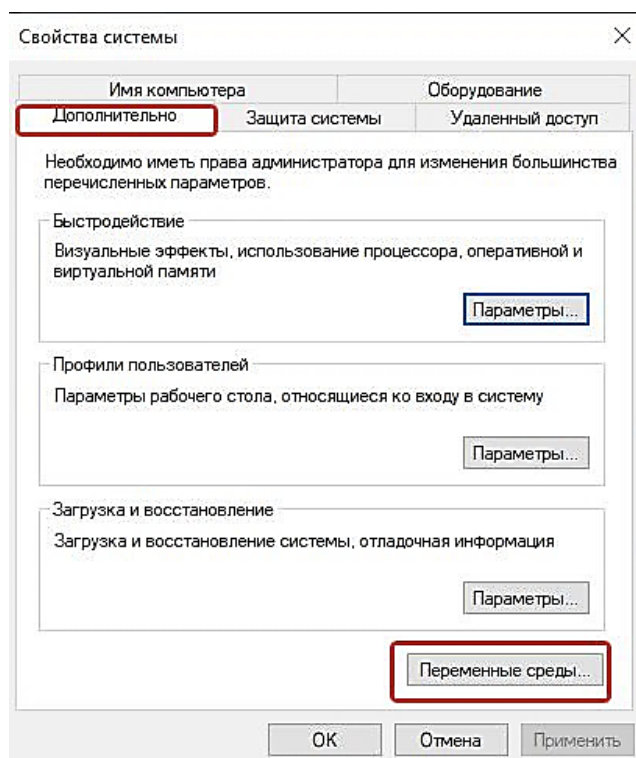


Рис. 1.10. Свойства системы

Перейдем к настройке переменных среды, выбрав системные переменные Path (рис. 1.11).

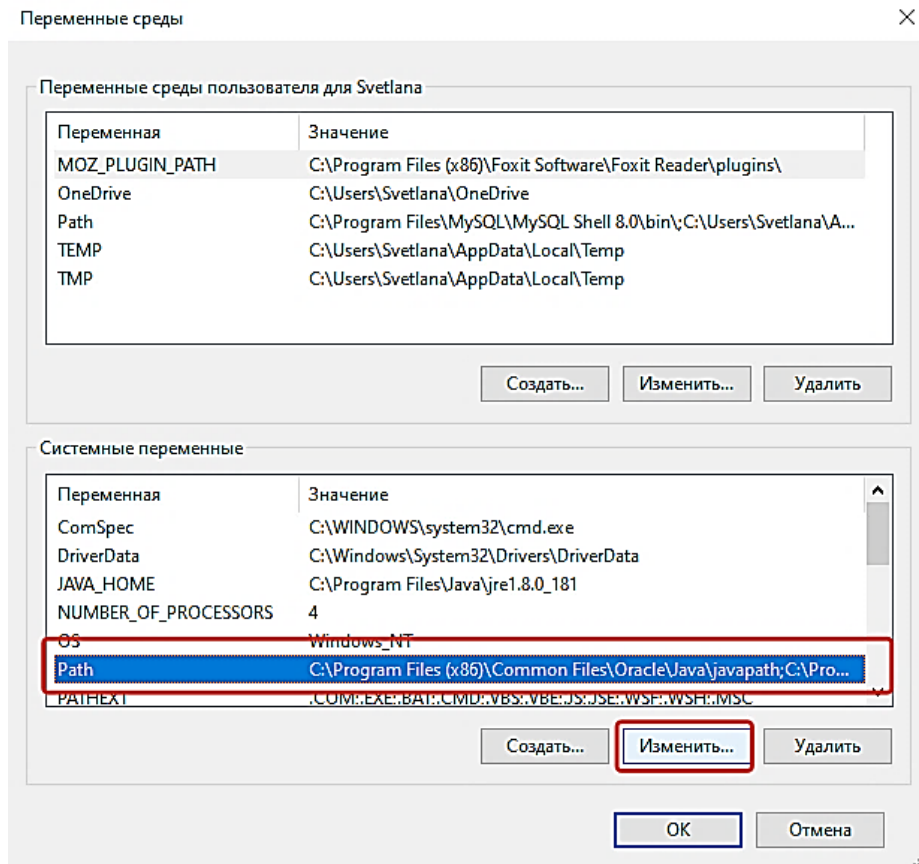


Рис. 1.11. Изменение переменных среды

Создадим переменную C:\sqlite (рис. 1.12) и последовательно закроем все диалоговые окна, кликнув по кнопке ОК.

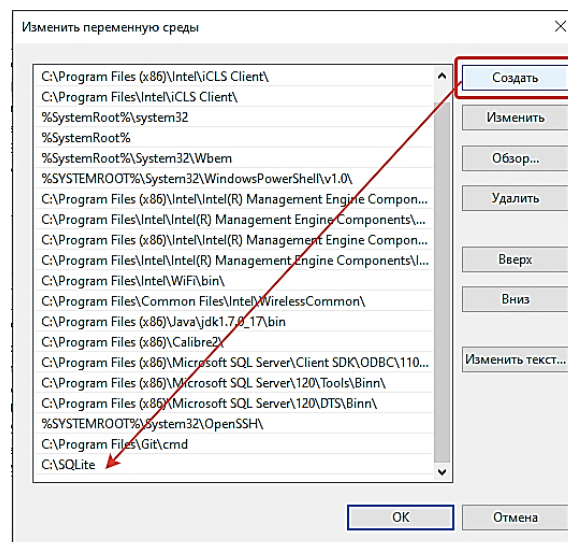
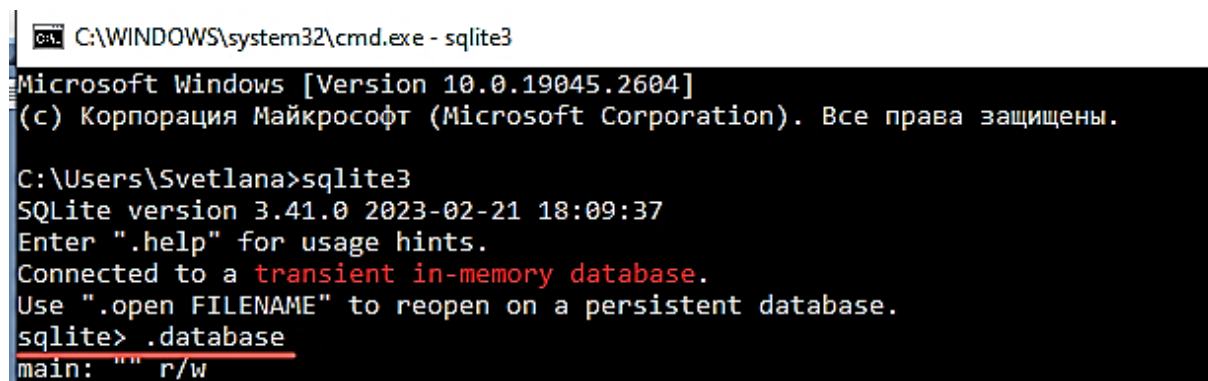


Рис. 1.12. Создание новой переменной среды

Важно: у SQLite при открытии терминала можно создавать таблицы во временной базе данных, которая будет уничтожена после закрытия интерфейса командной строки. Для того чтобы сохранить результаты работы, нужно создать/подключиться к базе данных.

Откроем интерфейс командной строки (если не было внесено изменений в переменные окружения, то см. рис. 1.7, иначе см. рис. 1.9) и проверим подключение к базе данных:

```
.database
```



```
C:\WINDOWS\system32\cmd.exe - sqlite3
Microsoft Windows [Version 10.0.19045.2604]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Svetlana>sqlite3
SQLite version 3.41.0 2023-02-21 18:09:37
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .database
main: "" r/w
```

Рис. 1.13. Проверка подключенной базы данных

Как видим, ожидаемо подключенной базы данных нет. Необходимо либо открыть ранее созданную базу данных, либо создать новую.

Команды в консоли вводятся после приглашения `sqlite>` (где `sqlite` – это СУБД SQLite). Команды, которые обращаются к консоли, начинаются с символа `«.»`. Приведем некоторые команды «с точкой»:

- `.help` – список команд консоли;
- `.open` – подключение к базе данных;
- `.mode column` – выравнивание выводимых таблиц по столбцам;
- `.headers on` – вывод названий столбцов;
- `.quit` – завершение работы терминала `sqlite3`.

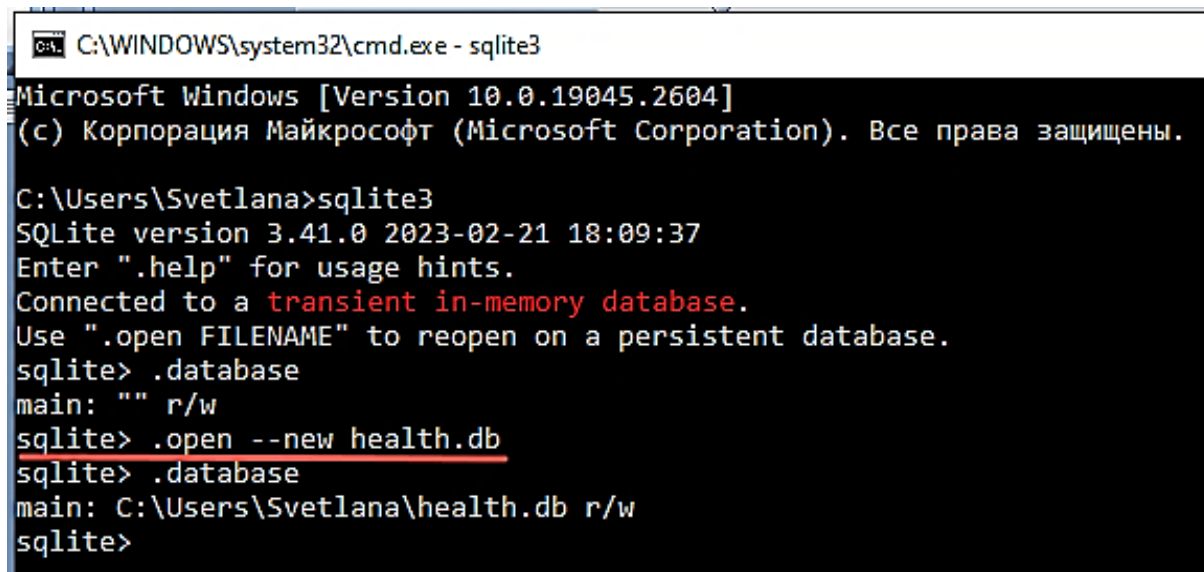
Подключимся к первой базе данных `health.db` (рис. 1.14), которая одновременно и создается, так как такой базы в рабочей директории еще не существует:

```
.open --new health.db
```

Наличие в коде указателя `--new` было излишним. Подключение к существующей и пока отсутствующей базе выполняется с помощью одной и той же команды `.open`, так как если базы данных нет в рабочей директории, то она будет создана.

Импортируем в новую таблицу (которой присвоим имя) подключенной базы данных `health.db` данные из файла с расширением `.csv`.

Имена таблиц в пределах базы данных, к которой выполнено подключение, должны быть уникальными.



```
C:\WINDOWS\system32\cmd.exe - sqlite3
Microsoft Windows [Version 10.0.19045.2604]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Svetlana>sqlite3
SQLite version 3.41.0 2023-02-21 18:09:37
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .database
main: "" r/w
sqlite> .open --new health.db
sqlite> .database
main: C:\Users\Svetlana\health.db r/w
sqlite>
```

Рис. 1.14. Создание базы данных health.db

Данные, которые мы используем для импорта – расходы на медицинское обслуживание тех, кто имеет медицинскую страховку:

<https://www.kaggle.com/mirichoi0218/insurance>.

Описание переменных набора:

- age: возраст основного бенефициара;
- sex: пол застрахованного;
- bmi: индекс массы тела;
- children: число детей, охваченных медицинским страхованием / число иждивенцев;
- smoker: курит ли застрахованный;
- region: жилой район получателя в США: Северо-Восток, Юго-Восток, Юго-Запад, Северо-Запад.
- charges: индивидуальные медицинские расходы, оплачиваемые страховкой.

Разместить файл с данными необходимо в рабочей директории. Открываем терминал SQLite и, подключившись к ранее созданной базе health.db, создадим в ней таблицу insurance, импортировав в нее данные файла insurance.csv (рис. 1.15):

```
.import --csv insurance.csv insurance
```

Для обращения к записям в созданной базе данных используем язык структурированных запросов (structured query language, SQL).

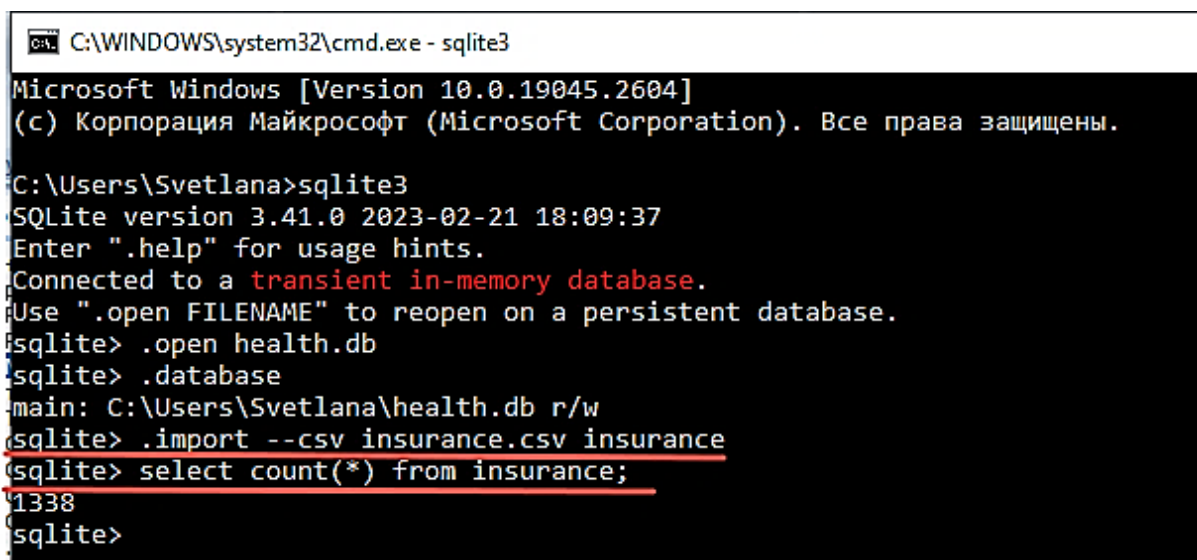
Все выражения SQL начинаются с ключевого слова и заканчиваются «;». SQL регистронезависимый язык. Есть негласное соглашение, по которому для ключевых слов и функций принято использовать прописные буквы, а для названий таблиц и атрибутов – строчные, однако, если используются только строчные/прописные, то это влияет исключительно на читабельность, а не на исполнимость кода.

В скриптах SQL для повышения читабельности кода принято добавлять комментарии: для оформления однострочного комментария используется удвоение тире --, многострочный заключается в комбинацию двух символов /* в начале и */ в конце:

```
-- однострочный комментарий
/*
  многострочный
  комментарий
*/
```

Запросим количество записей в таблице insurance, в которую были импортированы данные из файла (рис. 1.15):

```
select count(*) from insurance;
```



```
C:\WINDOWS\system32\cmd.exe - sqlite3
Microsoft Windows [Version 10.0.19045.2604]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Svetlana>sqlite3
SQLite version 3.41.0 2023-02-21 18:09:37
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open health.db
sqlite> .database
main: C:\Users\Svetlana\health.db r/w
sqlite> .import --csv insurance.csv insurance
sqlite> select count(*) from insurance;
1338
sqlite>
```

Рис. 1.15. Импорт данных из файла с расширением .csv

Обратите внимание, что в случае с СУБД SQLite при импорте данных из файла .csv в таблицу предварительно не создавали таблицу, не определяли имена полей таблицы. Они были заимствованы из строки заголовков файла .csv. Тип данных для импортируемых полей задан максимально общим, т.е. TEXT. Запросим схему таблицы, созданную при импорте файла .csv (рис. 1.16):

```
.schema insurance
```

```
C:\WINDOWS\system32\cmd.exe - sqlite3
C:\Users\Svetlana>sqlite3
SQLite version 3.41.0 2023-02-21 18:09:37
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open health.db
sqlite> .database
main: C:\Users\Svetlana\health.db r/w
sqlite> .import --csv insurance.csv insurance
sqlite> select count(*) from insurance;
1338
sqlite> .schema insurance
CREATE TABLE IF NOT EXISTS "insurance"(
"age" TEXT, "sex" TEXT, "bmi" TEXT, "children" TEXT,
"smoker" TEXT, "region" TEXT, "charges" TEXT);
sqlite>
```

Рис. 1.16. Схема таблицы данных insurance

Так относительно корректно (за исключением типов данных) импортироваться будут данные не из любого файла .csv. Во-первых, в файле insurance.csv в первой строке были заголовки (имена столбцов). Во-вторых, разделитель между данными различных столбцов в строке был «,», т.е. ровно тот, который ожидался. Также кодировка данных в файле была 'utf-8', что также соответствовало настройкам импорта файла .csv по умолчанию. Однако и строка заголовков может отсутствовать, и разделитель может использоваться другой («;», «|», и т.п.), и кодировка может также быть иной.

Поэтому в будущем желательно создавать таблицу, определяя ее структуру, т.е. имена столбцов и тип данных, а также при импорте данных из файла задавать все необходимые параметры (разделитель, наличие заголовков и т.п.).

Выполним запрос на вывод всех записей из таблицы insurance (рис. 1.17):

```
SELECT * FROM insurance LIMIT(5);
```

Такой вывод неудобен для чтения, данные не выровнены по столбцам и отсутствуют названия столбцов. Добавим команды, которые форматируют вывод строк, и повторим запрос (рис. 1.18):

```
.mode column
.headers on
SELECT * FROM insurance LIMIT(5);
```

```
C:\WINDOWS\system32\cmd.exe - sqlite3
Microsoft Windows [Version 10.0.19045.2604]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Svetlana>sqlite3
SQLite version 3.41.0 2023-02-21 18:09:37
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open health.db
sqlite> .database
main: C:\Users\Svetlana\health.db r/w
sqlite> SELECT * FROM insurance;
19|female|27.9|0|yes|southwest|16884.924
18|male|33.77|1|no|southeast|1725.5523
28|male|33|3|no|southeast|4449.462
33|male|22.705|0|no|northwest|21984.47061
32|male|28.88|0|no|northwest|3866.8552
31|female|25.74|0|no|southeast|3756.6216
46|female|33.44|1|no|southeast|8240.5896
37|female|27.74|3|no|northwest|7281.5056
```

Рис. 1.17. Вывод всех записей таблицы insurance

```
C:\WINDOWS\system32\cmd.exe - sqlite3
50|male|30.97|3|no|northwest|10600.5483
18|female|31.92|0|no|northeast|2205.9808
18|female|36.85|0|no|southeast|1629.8335
21|female|25.8|0|no|southwest|2007.945
61|female|29.07|0|yes|northwest|29141.3603
sqlite> .mode column
sqlite> .headers on
sqlite> SELECT * FROM insurance LIMIT(5);
age  sex    bmi    children  smoker  region    charges
---  ---
19   female  27.9    0         yes     southwest 16884.924
18   male    33.77   1         no      southeast 1725.5523
28   male    33      3         no      southeast 4449.462
33   male    22.705  0         no      northwest 21984.47061
32   male    28.88   0         no      northwest 3866.8552
sqlite>
```

Рис. 1.18. Форматирование вывода

Важно также контролировать расположение рабочей директории. Так, в варианте на рис. 1.14 рабочая директория, в которой создан файл базы данных health.db, это C:\Users\Svetlana.

А если запуск терминала произвести из папки C:\sqlite (см. рис. 1.15), то база данных proba.db будет создана уже в папке C:\sqlite, которая в этом случае выступает в роли рабочей директории (рис. 1.19).

```
C:\WINDOWS\system32\cmd.exe - sqlite3
Microsoft Windows [Version 10.0.19045.2604]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Svetlana> cd C:\sqlite

C:\SQLite>sqlite3
SQLite version 3.41.0 2023-02-21 18:09:37
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open proba.db
sqlite> .database
main: C:\SQLite\proba.db r/w
sqlite>
```

Рис. 1.19. Создание базы данных proba.db

1.3. Менеджер баз данных DBeaver

Работа в консоли для взаимодействия с СУБД требует знания специальных команд. Гораздо удобнее воспользоваться менеджером баз данных, в котором взаимодействие с различными СУБД происходит по стандартным сценариям.

Для SQLite есть несколько вариантов графических клиентов, например: DB Browser for SQLite; установочные файлы доступны на сайте <https://sqlitebrowser.org/>.

Однако существует достаточно универсальный менеджер баз данных DBeaver, который умеет работать в том числе, и с SQLite.

Скачаем его инсталлятор с официального сайта (рис. 1.20): <https://dbeaver.io/>

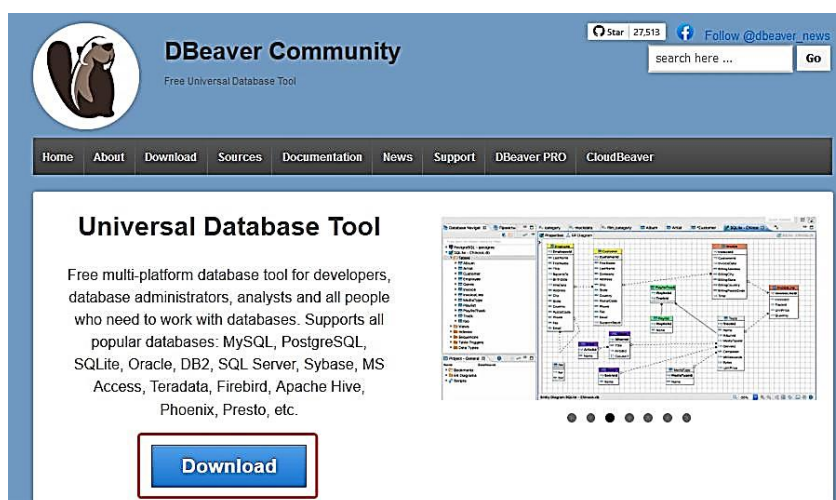


Рис. 1.20. Переход на страницу выбора версии установочных файлов

С учетом операционной системы выберем файл для скачивания (рис. 1.21).

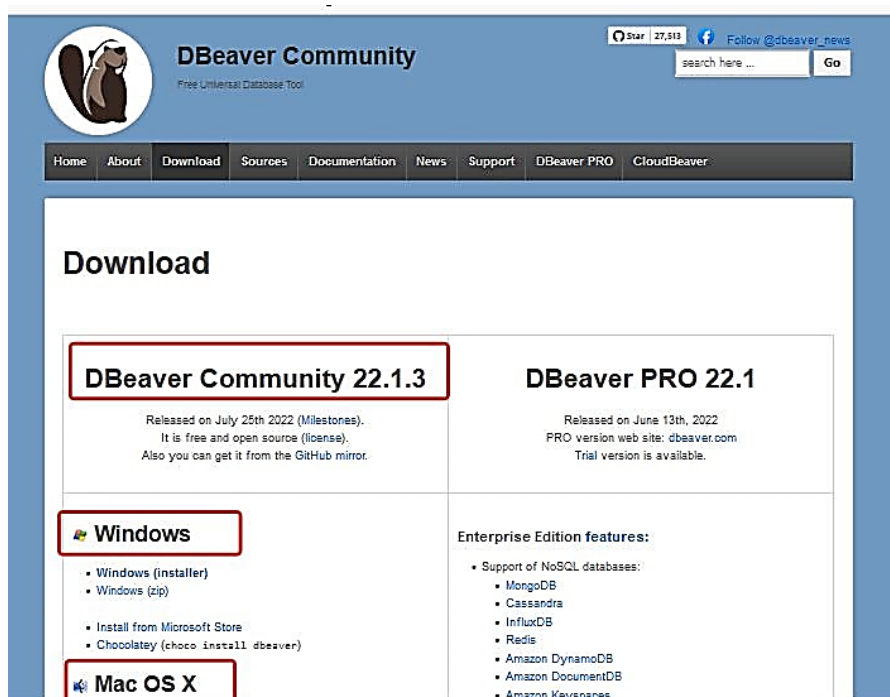


Рис. 1.21. Скачивание установочных файлов

При запуске инсталлятора необходимо выбрать язык для установщика. В мастере установки необходимо принять лицензионное соглашение и выбрать вариант установки (для всех пользователей или для одного, под которым происходит установка программного обеспечения (ПО)). После завершения работы мастера установки запустим менеджер баз данных DBeaver (рис. 1.22).

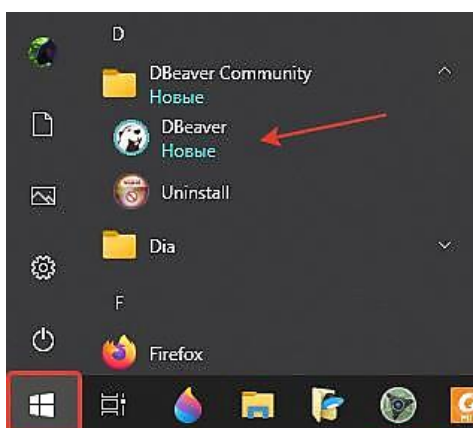


Рис. 1.22. Запуск менеджера DBeaver

В командном меню менеджера создаем новый файл (рис. 1.23).

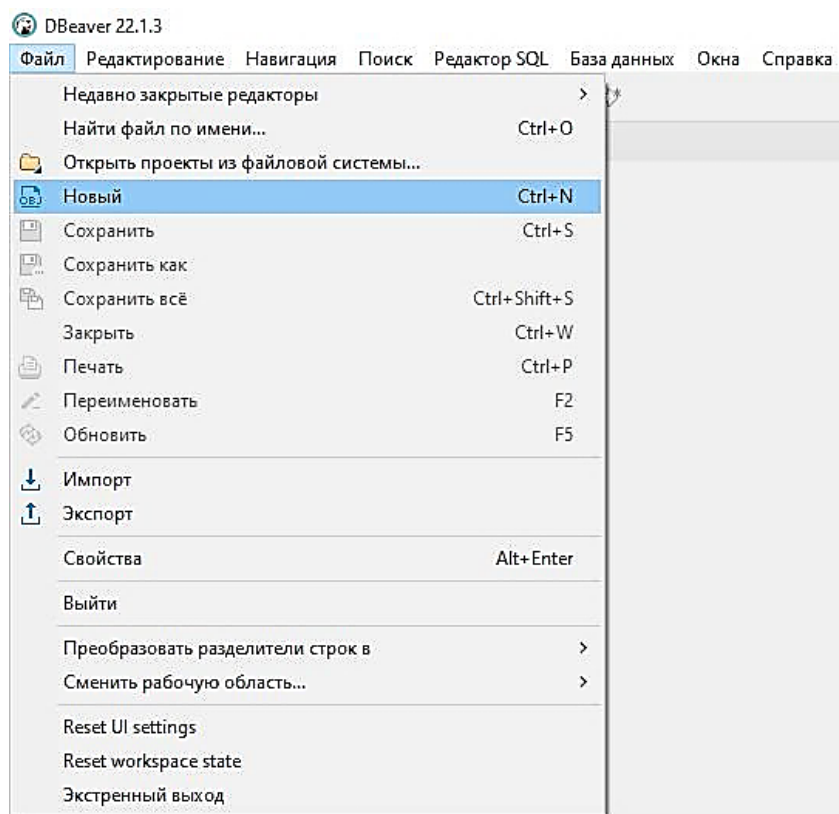


Рис. 1.23. Создание файла

Выбираем тип нового соединения (рис. 1.24).

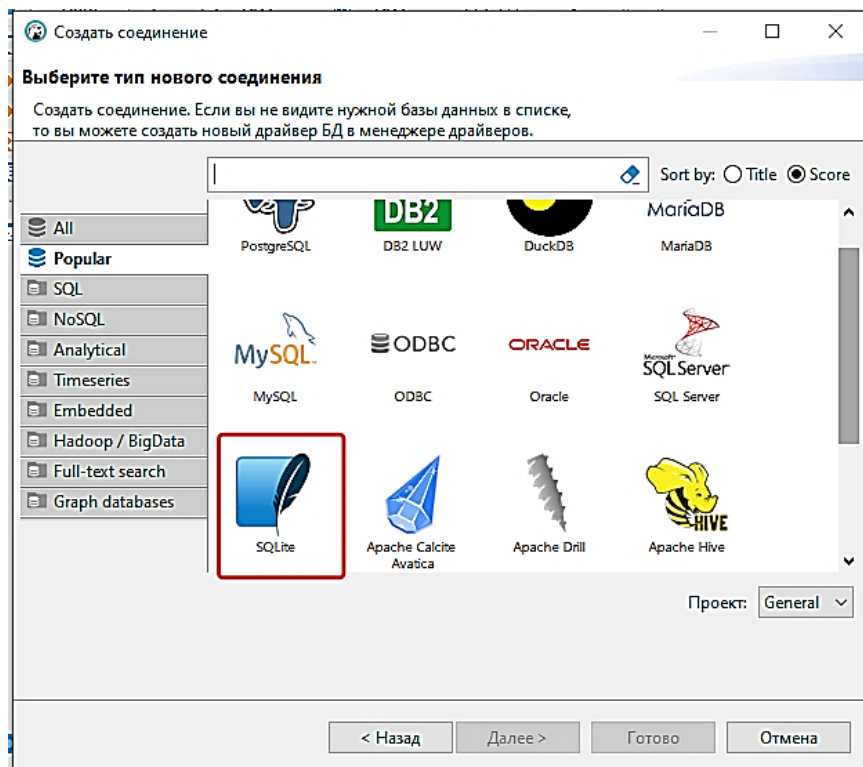


Рис. 1.24. Выбираем базу данных для соединения

Если менеджер DBeaver ранее использовался и есть текущие подключения, то создаем новое соединение нужного типа (рис. 1.25).

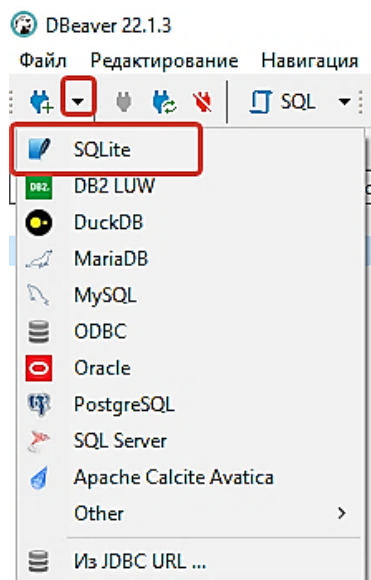


Рис. 1.25. Добавление нового подключения

Если база данных SQLite была создана ранее, то открываем ее (рис. 1.26), если нет, то создаем (рис. 1.27).

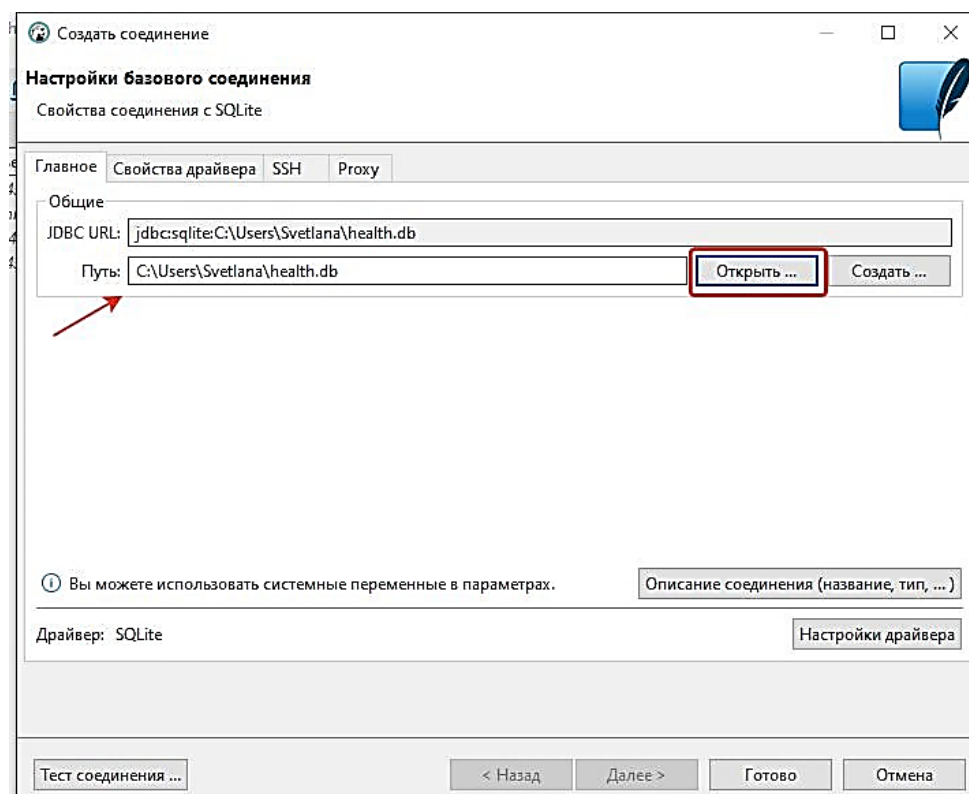


Рис. 1.26. Подключение ранее созданной базы данных

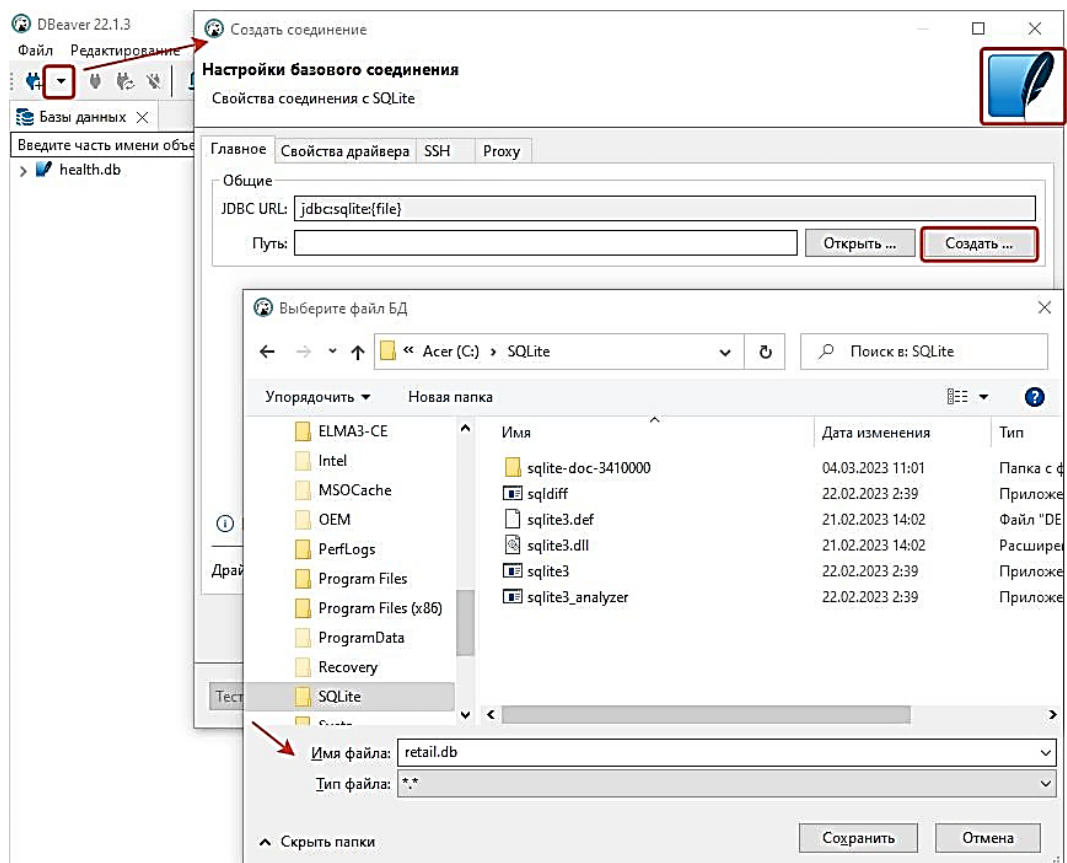


Рис. 1.27. Создание новой базы данных retail.db

Подключенные базы данных будут отображаться слева (рис. 1.28).

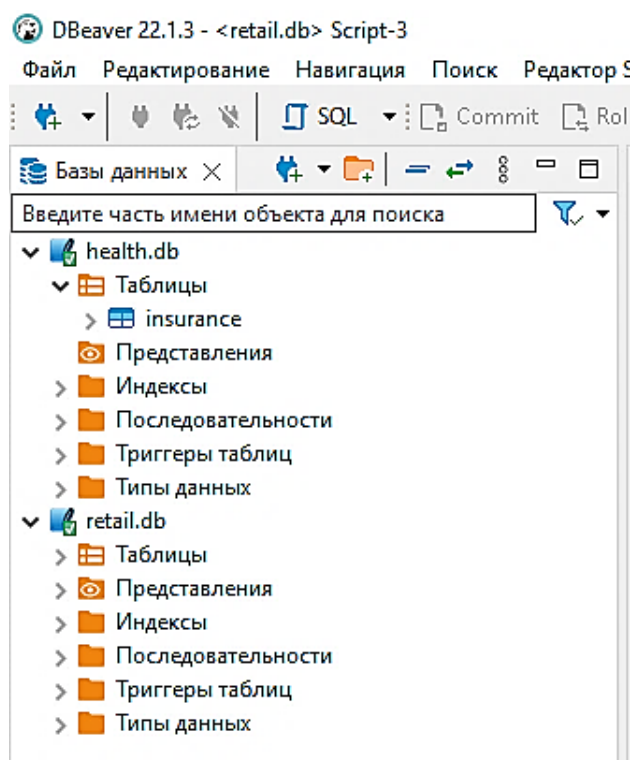


Рис. 1.28. Подключенные базы данных

Если какие-либо изменения не отображаются в подключенной базе данных, то либо в контекстном меню, вызываемом кликом правой клавиши мыши на подключении, выбираем опцию Обновить, либо нажимаем на клавиатуре F5.

Импортируем в базу данных retail.db данные из файла scanner_data.csv (доступен по ссылке:

<https://www.kaggle.com/datasets/marian447/retail-store-sales-transactions>).

Контекст данных: данные сканера о продажах в розничном магазине:

Данные файла scanner_data.csv:

- Date – дата сделки купли-продажи;
- Customer_ID – идентификатор покупателя;
- Transaction_ID – идентификатор транзакции;
- SKU_Category – категория товара;
- SKU – идентификатор товара;
- Quantity – количество проданного товара;
- Sales_Amount – стоимость проданного товара.

Просмотр данных возможен с помощью текстового редактора, например Блокнот (Стандартные программы ОС Windows), или с помощью редактора кода Notepad++ (дистрибутив можно скачать по ссылке: <https://notepad-plus-plus.org/downloads/>).

Как видим, нужные нам имена столбцов можно скопировать из первой строки, также можно определить тип данных в каждом столбце (рис. 1.29).

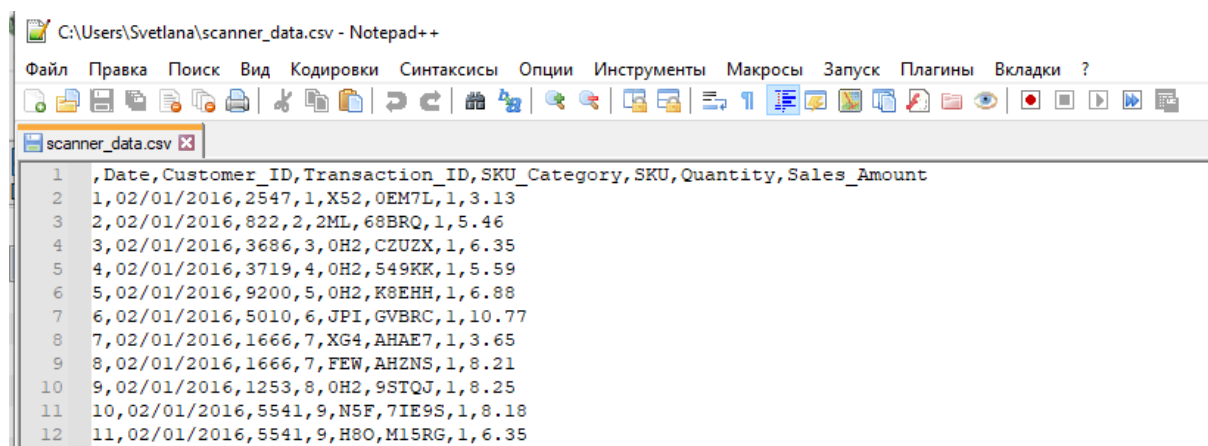


Рис. 1.29. Содержимое файла scanner_data.csv

В SQLite используется более общая система динамического типа для определения типа данных. В SQLite тип данных значения связан с самим значением, а не с его контейнером. Классы хранения SQLite представлены в табл. 1.1.

Типы данных в SQLite

Название	Описание
NULL	Значение – значение NULL
INTEGER	Значение представляет собой целое число со знаком, сохраненное в 1, 2, 3, 4, 6 или 8 байтах в зависимости от величины значения
REAL	Значение представляет собой значение с плавающей запятой, которое хранится как 8-байтовое число с плавающей точкой IEEE
TEXT	Значение представляет собой текстовую строку, хранящуюся с использованием кодировки базы данных (UTF-8, UTF-16BE или UTF-16LE)
BLOB	Значение представляет собой блок данных, который хранится точно так же, как он был введен

SQLite не имеет отдельного класса хранения для хранения дат и/или времени, но SQLite способен хранить даты и время как значения TEXT (дата в формате «YYYY-MM-DD HH:MM:SS.SSS»), REAL (число дней с полудня в Гринвиче 24 ноября 4714 г. до н.э.) или INTEGER (количество секунд с 1970-01-01 00:00:00 UTC).

Пропущенное значение обозначается NULL. NULL – это специальное значение (псевдозначение), которое означает отсутствие значения и не принадлежит ни одному типу данных. Это не то же самое, что например, значение "", которое обозначает пустую строку.

Откроем скрипт SQL для базы данных retail.db (рис. 1.30 или рис. 1.31).

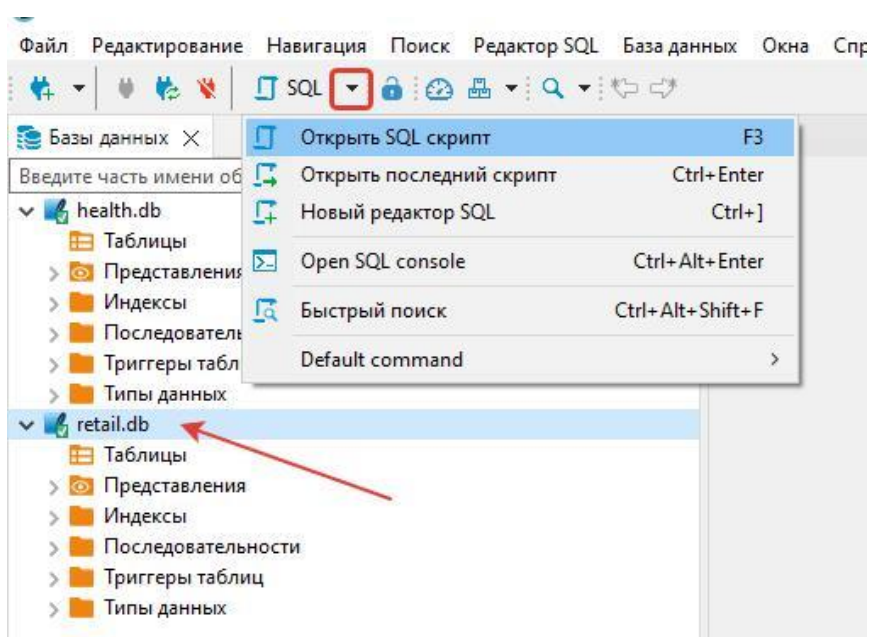


Рис. 1.30. Открытие SQL-скрипта с помощью пиктографического меню

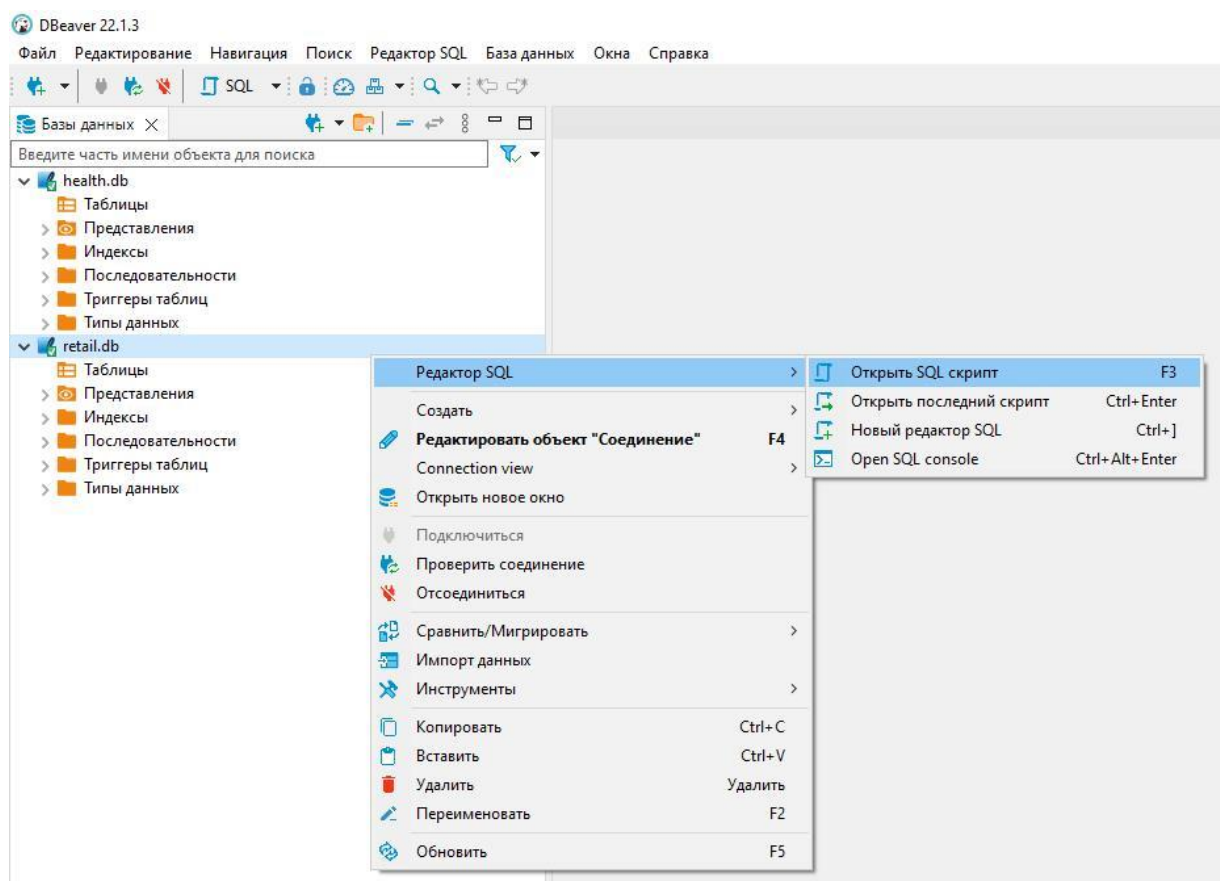


Рис. 1.31. Открытие SQL-скрипта с помощью контекстного меню

Создадим таблицу scanner, в которой имена полей соответствуют строке заголовков, а тип данных – типу значений в соответствующих колонках. Хотя данные в столбцах Customer_ID и Transaction_ID задаются числовыми значениями, операции с числами в этих колонках не имеют смысла, поэтому установим тип данных для соответствующих полей как TEXT (рис. 1.32). Первое поле с пустым именем – это номер строки. Не будем его определять и впоследствии удалим из созданной таблицы:

```
CREATE TABLE scanner(
    Date TEXT,
    Customer_ID TEXT,
    Transaction_ID TEXT,
    SKU_Category TEXT,
    SKU TEXT,
    Quantity INTEGER,
    Sales_Amount REAL
);
```

Теперь используем для загрузки данных в созданную таблицу опцию контекстного меню Импорт данных (рис. 1.33).

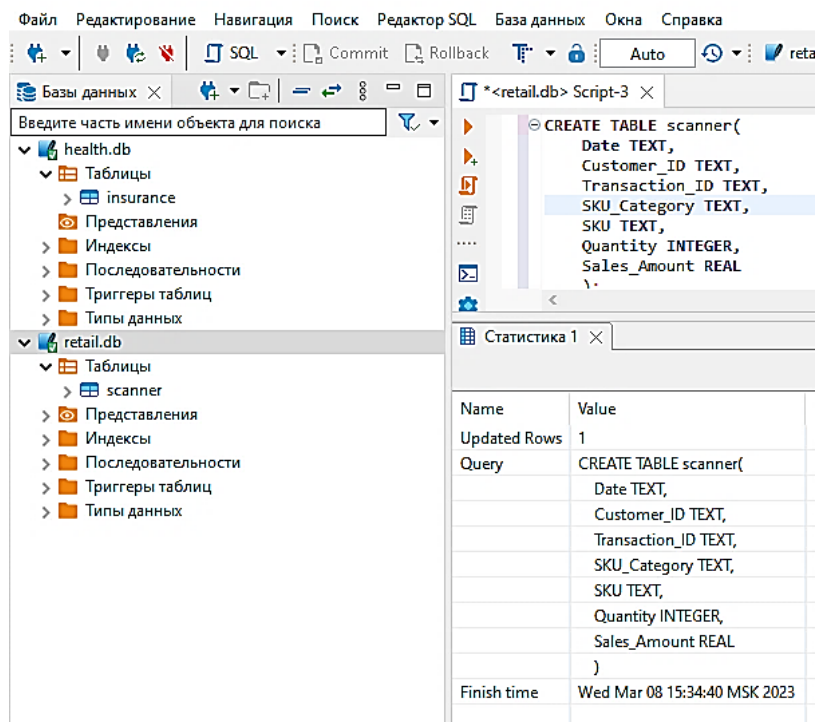


Рис. 1.32. Создание таблицы scanner

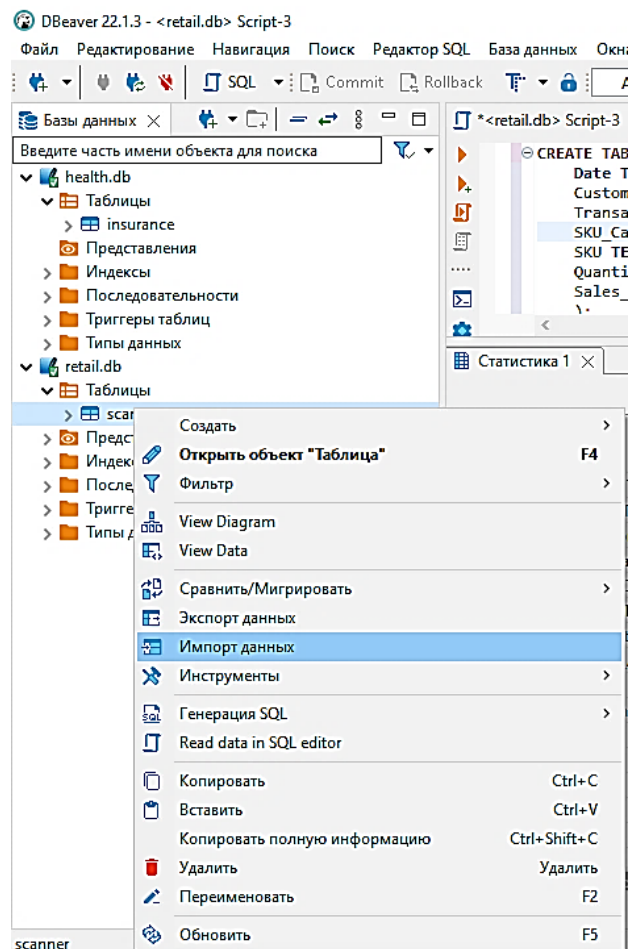


Рис. 1.33. Импорт данных в таблицу scanner

В мастере импорта данных определим необходимые настройки (рис. 1.34–1.37). При определении соответствия столбцов нажмем кнопку Обзор. (рис. 1.36) и, выбрав таблицу scanner, внесем изменения в первый столбец с данными номера строки (рис. 1.37), нажав клавишу <Del>, переведем статус Соответствие в пропущено (рис. 1.38).

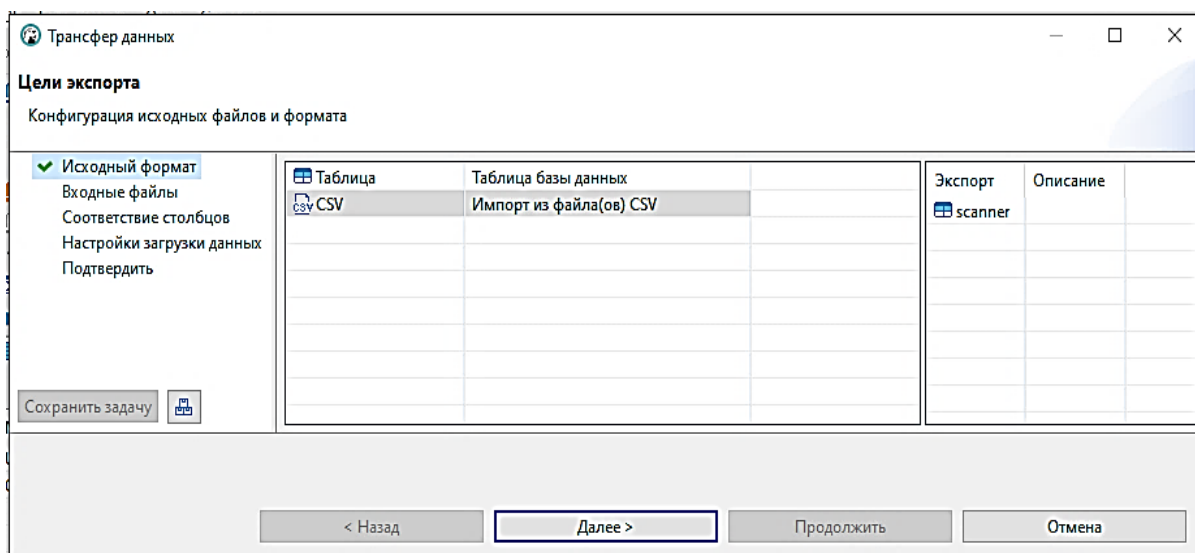


Рис. 1.34. Конфигурация исходных файлов

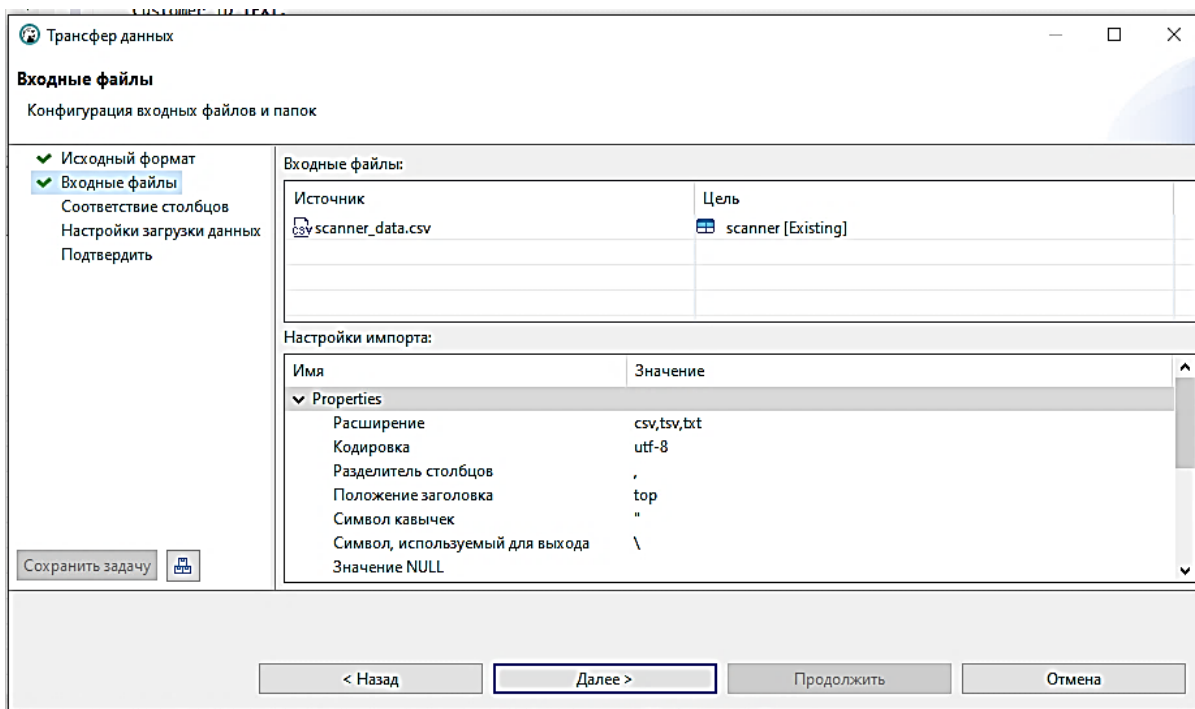


Рис. 1.35. Выбор файла для импорта в браузере

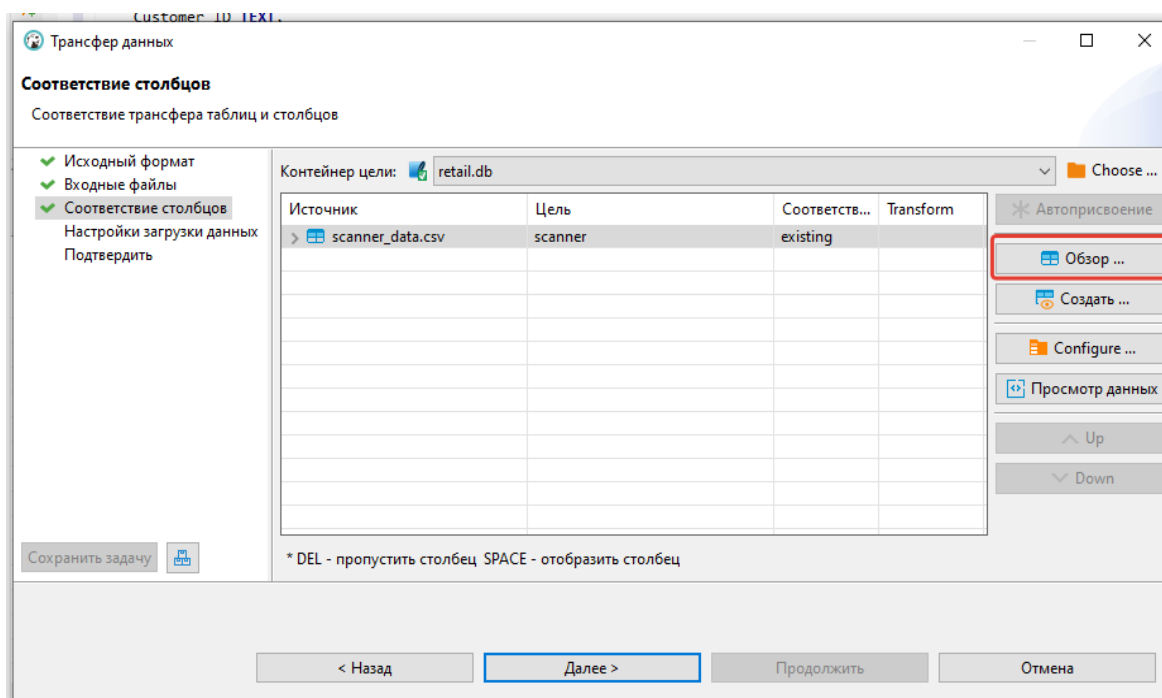


Рис. 1.36. Переход к настройке соответствия столбцов

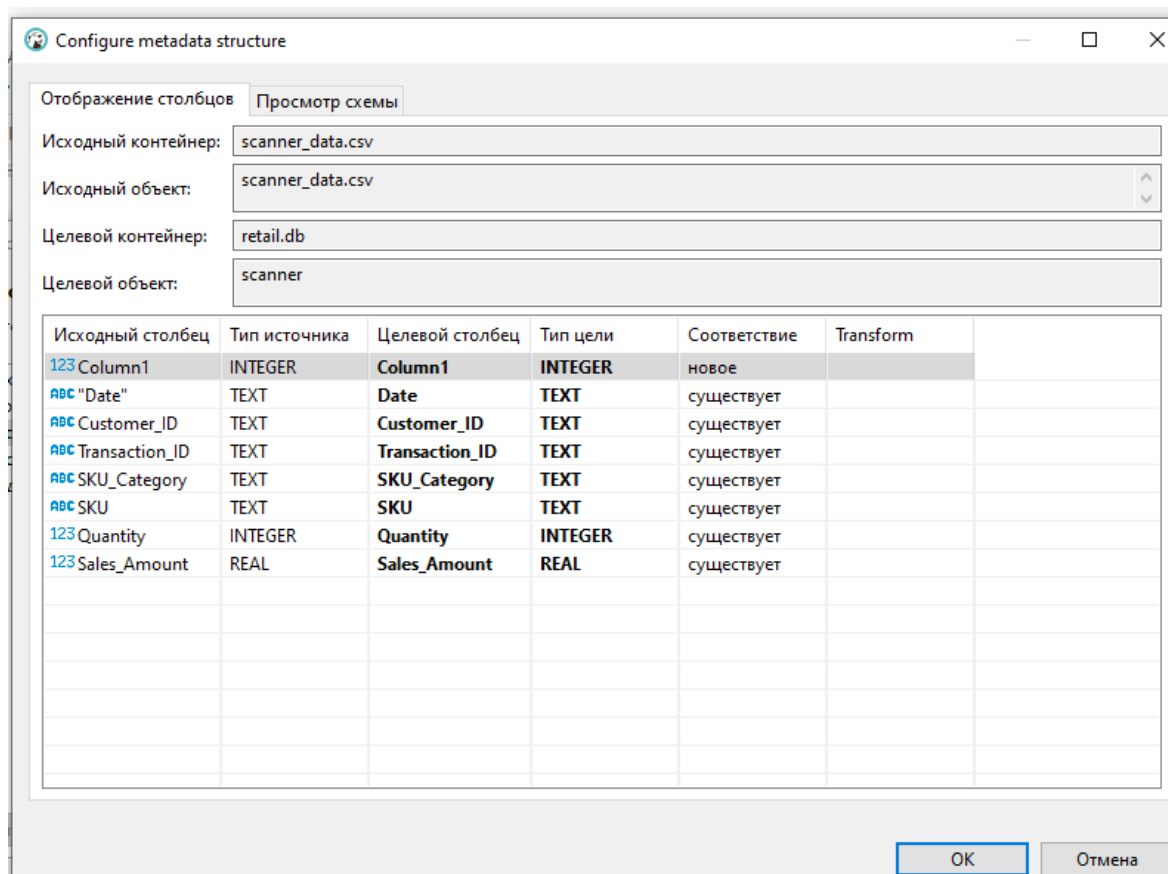


Рис. 1.37. Выбор столбца для редактирования настроек соответствия

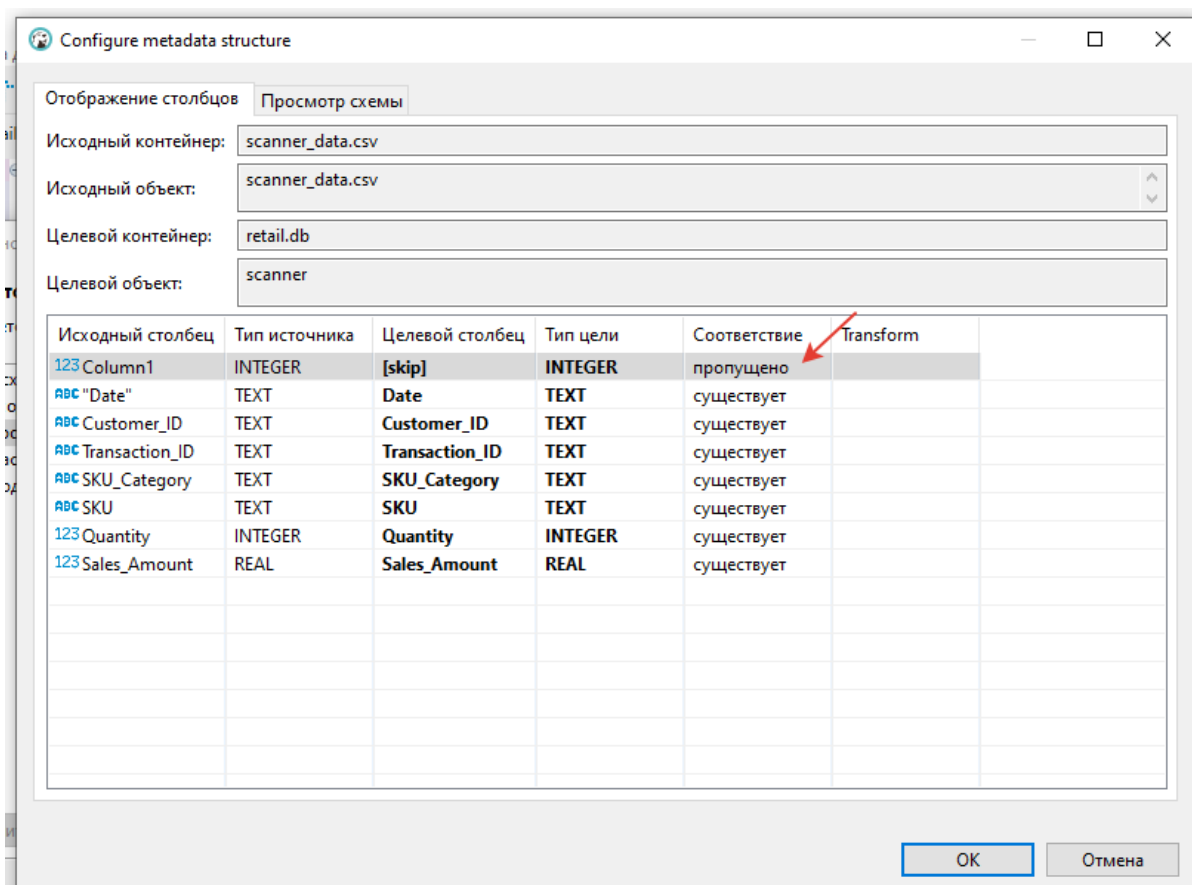


Рис. 1.38. Исключение столбца из числа импортируемых

Завершим импорт данных и откроем таблицу scanner для просмотра (рис. 1.39).

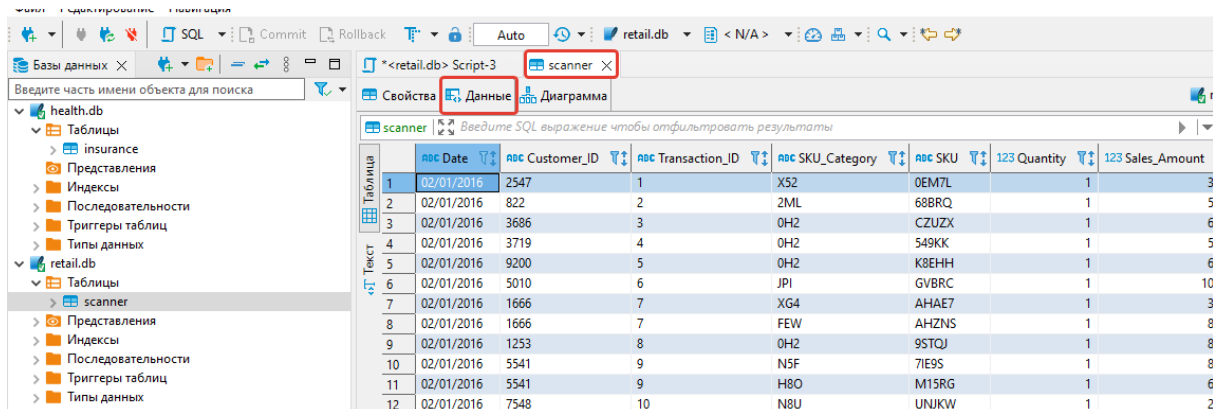


Рис. 1.39. Просмотр результатов импорта

1.4. Работа с SQLite в Python

СУБД SQLite может быть использована в разработке на языке Python. Библиотека SQLite3 является частью стандартного пакета Python 3, поэтому инструкция установки (`pip install sqlite3`) не потребуется. Необходимо только подключить библиотеку `sqlite3`. Однако нам нужна среда для работы с кодом на языке Python.

Для анализа данных на языке Python традиционно используют Notebook либо в среде разработки Anaconda, в которую входит Jupyter Notebook, либо в облачной среде разработки Google Colaboratory (Colab) – Colab Notebook.

Зарегистрируем аккаунт для сервисов Google @gmail.com. И войдем в Google Colaboratory по ссылке: <https://colab.research.google.com/>
Создадим новый блокнот (Notebook) (рис. 1.40).

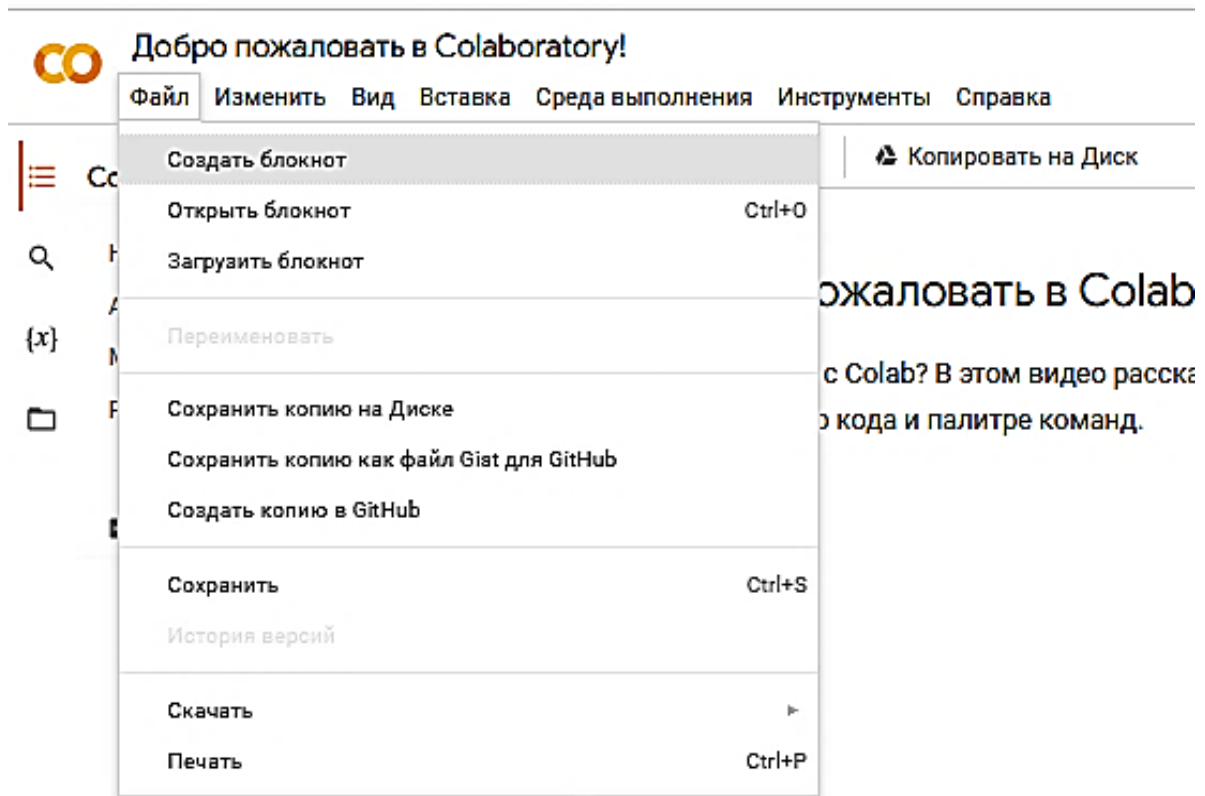


Рис. 1.40. Создание блокнота в Colab

Переименуем открывшийся файл (рис. 1.41) и познакомимся с основными возможностями инструмента (рис. 1.42).

Каждый файл `.ipynb` содержит блоки кода и поясняющего текста. Сохранение происходит налету, как и во всех облачных сервисах Google.

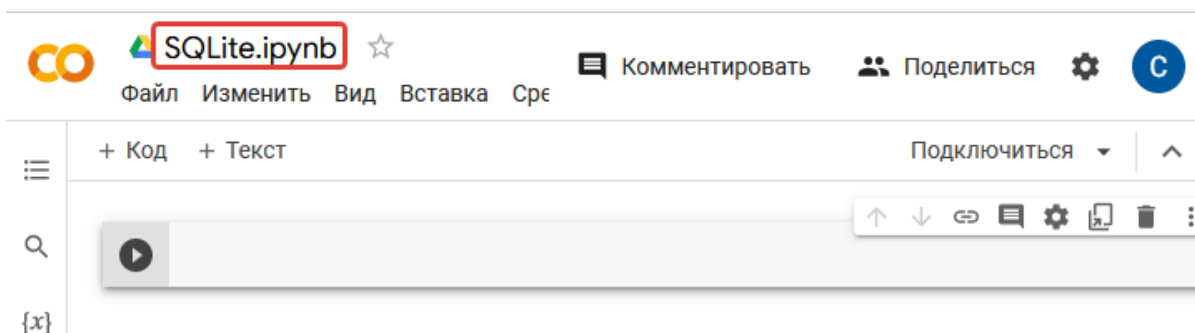


Рис. 1.41. Переименование файла

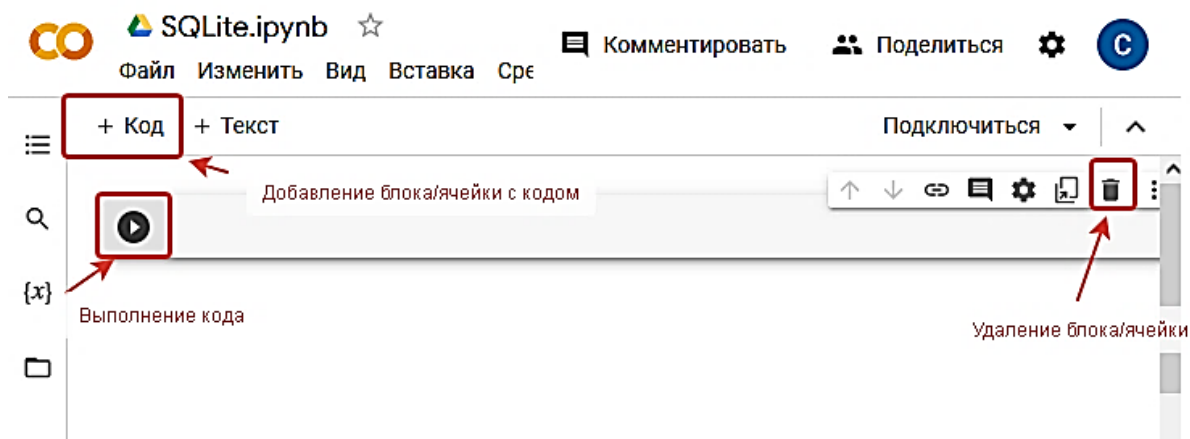


Рис. 1.42. Структура файла .ipynb

Все файлы, созданные в Colab, сохраняются на Диск, связанный с аккаунтом Google (рис. 1.43). На диск можно перейти на странице поиска Google, раскрыв кнопку доступа к сервисам. Вместе с первым сохраненным файлом на диске появляется и папка Colab Notebooks, в которой и будут храниться все блокноты с кодом.

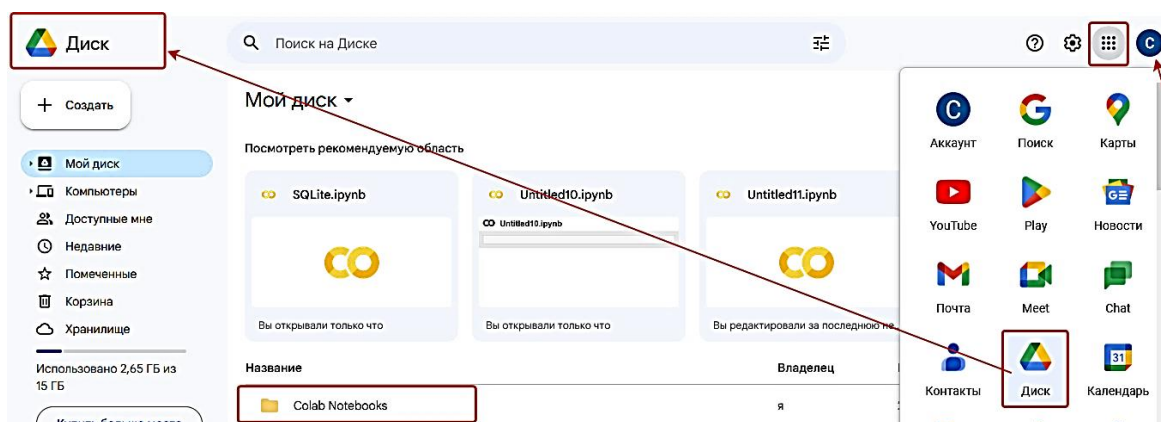


Рис. 1.43. Структура Диска, связанного с аккаунтом @gmail.com

Подключим библиотеку `sqlite3` и назначим ей псевдоним `sl`:

```
import sqlite3 as sl
```

Как видим, код выполнен без ошибок (рис. 1.44).

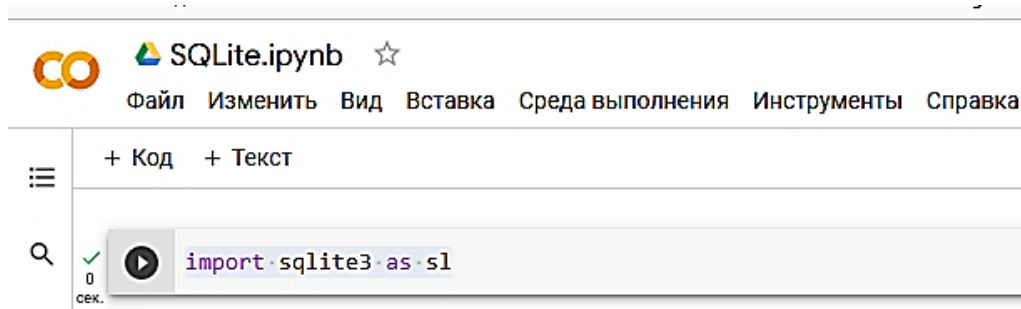


Рис. 1.44. Подключение библиотеки `sqlite3`

Подключимся к базе данных `churn.db`. Одновременно эта база данных будет создана. Одна и та же инструкция (как и раньше в терминале) выполняет подключение к уже имеющейся базе или одновременно создает базу данных и подключается к ней, если в рабочей директории нет базы данных с таким именем:

```
con = sl.connect('churn.db')
```

Раскроем папку рабочей директории на панели слева и увидим созданную базу данных (рис. 1.45).

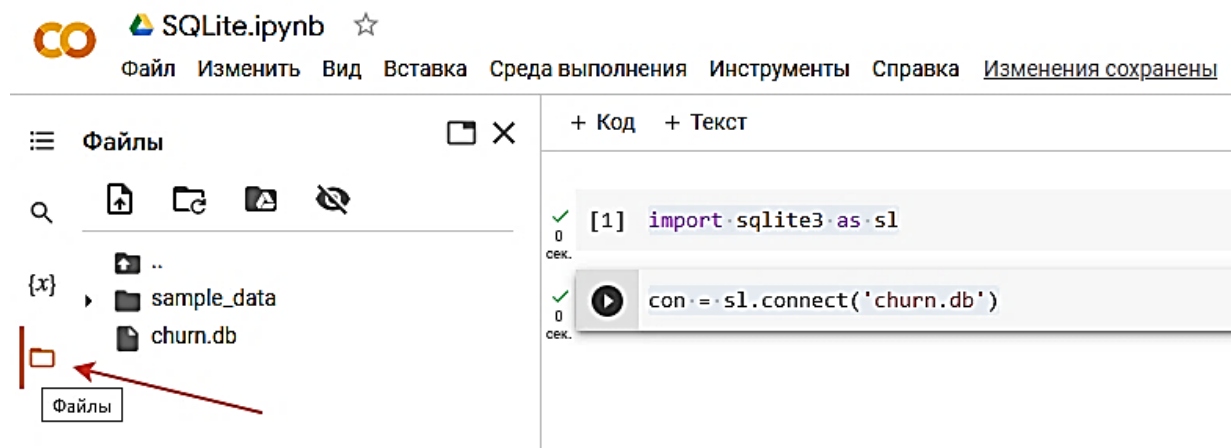


Рис. 1.45. Содержимое рабочей директории в облаке

Переподключимся к ранее созданной базе данных `retail.db`. Для этого вначале загрузим эту базу данных в рабочую директорию облака, щелкнув правой клавишей мыши в свободном поле рабочей директории, открытой слева, и выбрав в контекстном меню опцию Загрузить,

а затем в браузере указав нужный файл. При загрузке возникает предупреждение (рис. 1.46). Если бы работа происходила не в облаке, то все результаты работы сохранялись бы на системном диске компьютера. Но серверное хранилище рабочей директории (в отличие от самих файлов с кодом, сохраняющихся на Диск) периодически очищается, и важные файлы следует загружать из временной рабочей директории в постоянное хранилище (на Диск, на компьютер и т.п.).

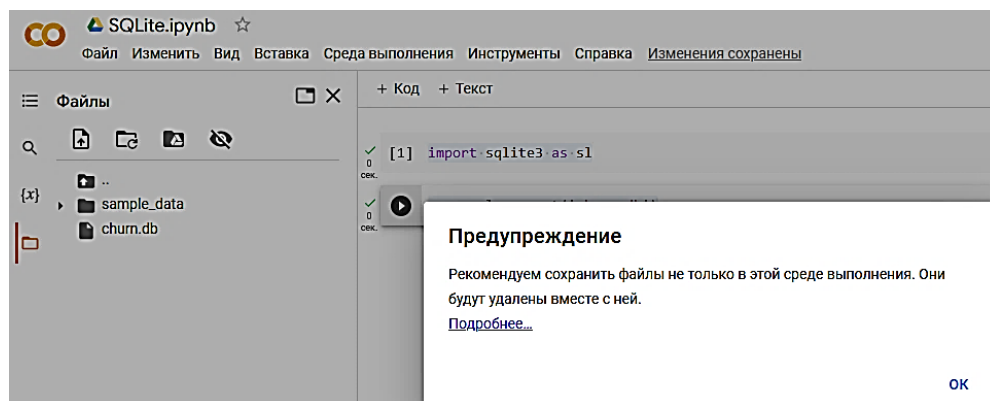


Рис. 1.46. Предупреждение о временном хранении файлов с данными

Рассмотрим также альтернативный способ загрузки с помощью кода в Notebook. Для этого создадим копию файла SQLite.ipynb и откроем его в новом окне. Выполним подключение библиотеки sqlite3 и загрузим файл базы данных retail.db в рабочую директорию Colab с помощью метода files.upload() (рис. 1.47). Кнопкой Обзор в браузере выбирается нужный файл, после загрузки в рабочую директорию он становится доступен для подключения:

```
from google.colab import files
uploaded = files.upload()
```

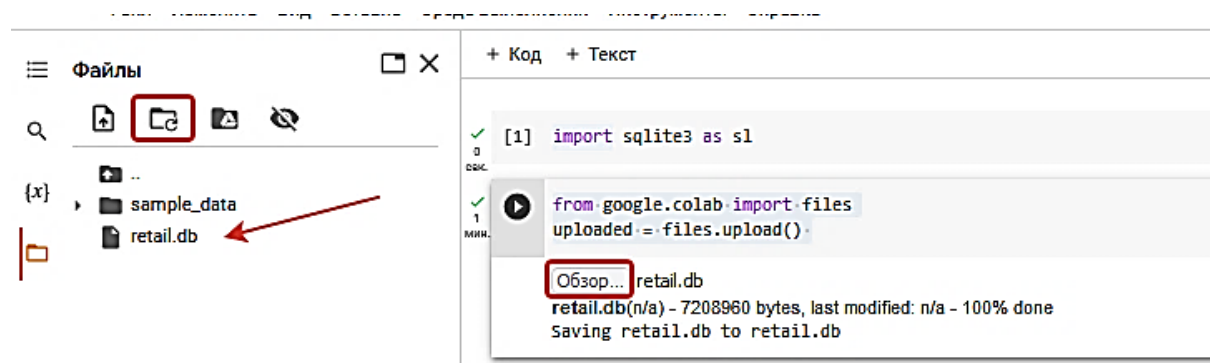


Рис. 1.47. Альтернативный способ загрузки файла базы данных в рабочую среду

Подключимся к базе данных retail.db и сохраним подключение в переменную con:

```
con = sl.connect('retail.db')
```

Преобразование типов данных при передаче из SQLite в Python в табл. 1.2.

Таблица 1.2

Преобразование типов данных

Тип данных	
SQLite	Python
NULL	None
INTEGER	int
REAL	float
TEXT	str
BLOB	bytes

После подключения к базе данных необходимо создать специальный объект cursor, который позволяет выполнять команды SQL:

connection.cursor() – создание объекта cursor;

cursor.execute() – выполнение запроса SQL;

connection.commit() – сохранение изменений;

connection.rollback() – откат изменений;

conn.close() – закрытие соединения.

Для получения результатов запроса SELECT используются команды:

fetchone() – возвращает следующую строку из результата;

fetchmany() – возвращает указанное количество строк;

fetchall() – возвращает все оставшиеся строки.

Для выполнения запросов к базе данных retail.db создадим объект cursor и сохраним его в переменной cur, выполним запрос с помощью метода cur.execute (обратите внимание, что сам запрос помещается в кавычки), а затем выведем полученный результат с помощью метода cur.fetchmany (рис. 1.48):

```
cur = con.cursor()
cur.execute("SELECT * FROM scanner;")
cur.fetchmany(3)
```

Можно вывести все оставшиеся записи таблицы scanner с помощью метода cur.fetchall (рис. 1.49):

```
cur.fetchall()
```

Если запрос состоит из нескольких команд, разделенных точкой с запятой, т.е. является скриптом на SQL, то кавычки в начале и в конце утраиваются.

```

✓ [3] con = sl.connect('retail.db')
0
DBKL

✓ [4] cur = con.cursor()
0
DBKL

✓ [5] cur.execute("SELECT * FROM scanner;")
0
DBKL
cur.fetchmany(3)

[('02/01/2016', '2547', '1', 'X52', '0EM7L', 1, 3.13),
 ('02/01/2016', '822', '2', '2ML', '68BRQ', 1, 5.46),
 ('02/01/2016', '3686', '3', '0H2', 'CZUZX', 1, 6.35)]

```

Рис. 1.48. Выполнение запроса и вывод его результата

+ Код + Текст

```

✓ cur.fetchall()
0
DBKL

('05/01/2016', '3568', '427', 'N8U', 'QM3C8', 1, 5.59),
('05/01/2016', '3100', '428', '7C6', '7YBOX', 1, 13.66),
('05/01/2016', '6283', '429', 'OQA', 'DWJ4C', 1, 12.87),
('05/01/2016', '6283', '429', '01F', 'OJ55N', 2, 51.82),
('05/01/2016', '6896', '430', 'N8U', 'JJLXQ', 1, 4.84),
('05/01/2016', '6896', '430', 'XG4', '8XUSX', 1, 1.27),
('05/01/2016', '2443', '431', 'SJS', '7081I', 2, 67.64),
('05/01/2016', '2443', '431', 'SJS', '2GC0V', 2, 66.57),
('05/01/2016', '2443', '431', 'FUS', 'W53F2', 2, 2.43),
('05/01/2016', '2443', '431', 'A38', '78XS6', 2, 13.17),
('05/01/2016', '2443', '431', 'LPF', '9F801', 1, 6.18),
('05/01/2016', '2443', '431', 'LPF', 'CYAX0', 2, 7.37),
('05/01/2016', '5012', '432', 'J4R', 'T4MAL', 1, 5.05),
('05/01/2016', '5012', '432', 'SFC', 'NHQAO', 1, 3.12),
('05/01/2016', '3704', '433', 'YMJ', '3CUG7', 1, 5.38),
('05/01/2016', '1820', '434', 'FUS', '3ZY0H', 2, 10.0),
('05/01/2016', '1820', '434', '29A', 'LPLTZ', 1, 3.9),
('05/01/2016', '1820', '434', 'N8U', 'TL4WZ', 1, 4.0),
('05/01/2016', '1820', '434', 'LDZ', 'IRQZ4', 1, 36.22),
('05/01/2016', '1820', '434', 'FUS', '29D51', 2, 18.67),
('05/01/2016', '1820', '434', 'USF', 'OGKA9', 2, 7.61),
('05/01/2016', '1820', '434', 'TZB', 'WYFCJ', 1, 11.18),
('05/01/2016', '1820', '434', 'L28', 'U7WMV', 2, 18.71),
('05/01/2016', '7191', '435', 'FEW', 'F79YP', 1, 16.91),
('05/01/2016', '5051', '436', 'N8U', 'H87RV', 1, 6.0),
('05/01/2016', '6986', '437', 'LPF', '92US4', 1, 15.86),
('05/01/2016', '6986', '437', 'N8U', 'TL4WZ', 2, 8.02),
('05/01/2016', '6986', '437', 'FUS', '2IUD5', 2, 4.17),
('05/01/2016', '6986', '437', 'OQA', 'DWJ4C', 1, 12.87),
('05/01/2016', '6986', '437', 'IEV', 'PWG2Z', 1, 5.01),

```

Рис. 1.49. Вывод всех оставшихся записей таблицы scanner

Теперь в новом файле среды Colab создадим таблицу базы данных churn.db на основе файла с расширением .csv.

Импортируем в базу данных churn.db данные из файла Churn_Modelling.csv (доступен по ссылке: https://www.kaggle.com/datasets/shubh0799/churn-modelling?select=Churn_Modelling.csv).

Контекст данных: набор данных – это сведения о клиентах в компании. В колонках указаны предполагаемая зарплата клиента, возраст, пол и т.д.

Данные файла Churn_Modelling.csv:

- RowNumber – номер строки;
- CustomerId – идентификатор заказчика;
- Surname – фамилия заказчика;
- CreditScore – кредитный рейтинг;
- Geography – географическое положение;
- Gender – пол;
- Age – возраст;
- Tenure – собственность;
- Balance – баланс (остаток);
- NumOfProducts – количество продуктов;;
- HasCrCard – имеет кредитную карту;
- IsActiveMember – является активным участником;
- EstimatedSalary – предполагаемая заработная плата;
- Exited – ушел из обслуживания в компании.

Подключим необходимые библиотеки, также подключим базу данных Churn.db (сохраним подключение в переменную con), создадим объект cursor и сохраним его в переменную cur и загрузим ранее скачанный на компьютер файл Churn_Modelling.csv в рабочую директорию среды (рис. 1.50):

```
import sqlite3 as sl
import csv
con = sl.connect('churn.db')
cur = con.cursor()
```

Создадим таблицу churn_mod, вначале в переменную сохраним запрос SQL на создание таблицы с ранее перечисленными названиями столбцов и типом хранимых в них данных (запрос заключается в кавычки). Затем выполним этот запрос в подключенной базе churn.db (рис. 1.51):

```
create_table = """CREATE TABLE churn_mod(
    RowNumber INTEGER,
    CustomerId TEXT,
    Surname TEXT,
    CreditScore INTEGER,
    Geography TEXT,
```

```

Gender TEXT,
Age INTEGER,
Tenure INTEGER,
Balance REAL,
NumOfProducts INTEGER,
HasCrCard INTEGER,
IsActiveMember INTEGER,
EstimatedSalary REAL,
Exited INTEGER
);'''
cur.execute(create_table)

```

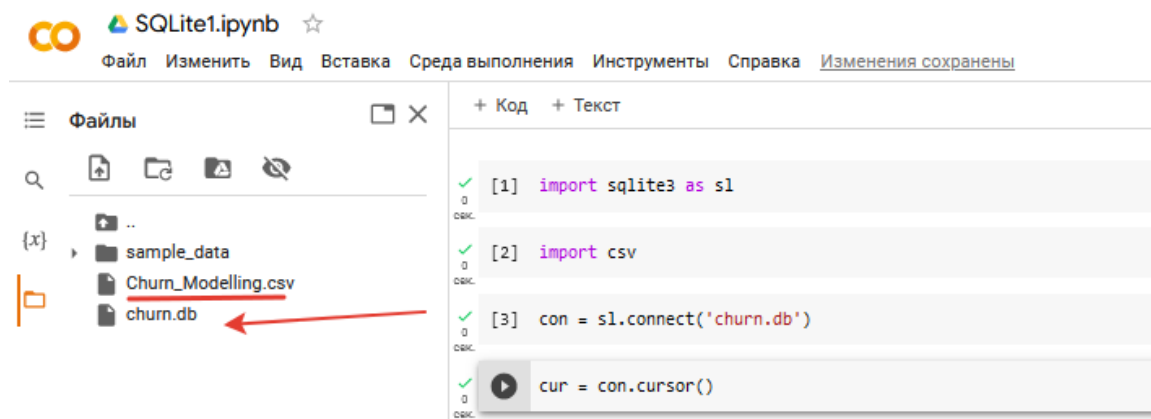


Рис. 1.50. Загрузка файла во временную рабочую среду и выполнение подключения необходимых в работе объектов

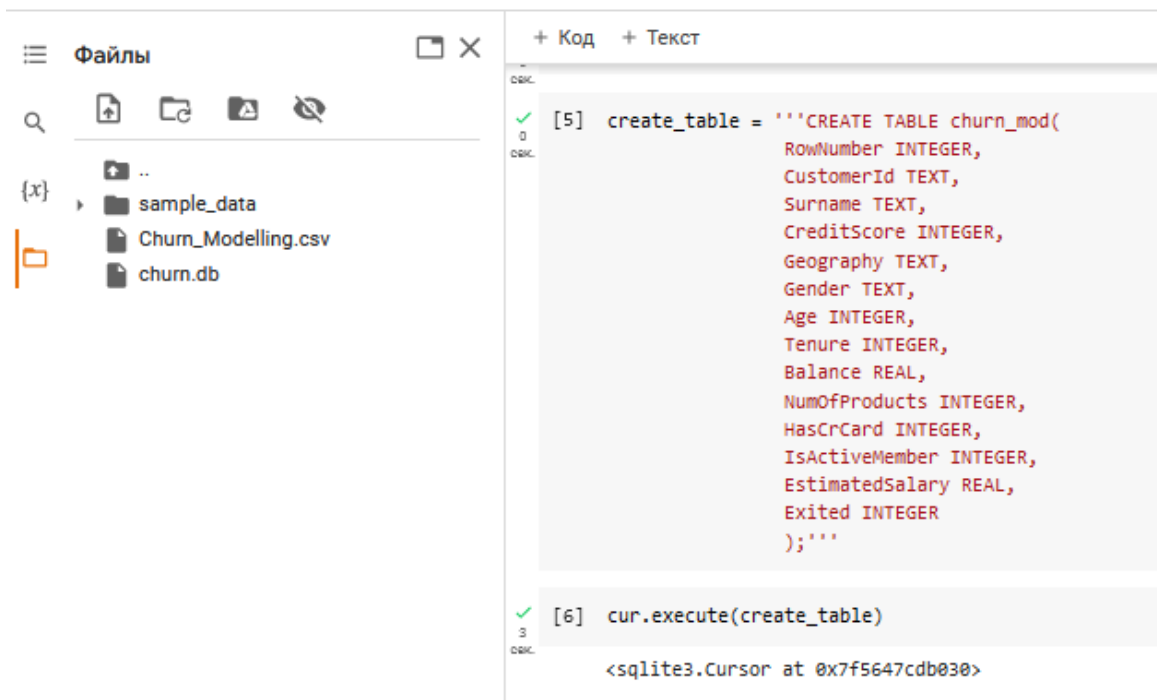


Рис. 1.51. Выполнение запроса на создание таблицы с соответствующей структурой (метаданными)

В переменную `file` импортируем файл `Churn_Modelling.csv`, а затем считаем его строки в переменную `contents`. В переменную `insert_records` сохраним запрос на вставку в столбцы таблицы `churn_mod` значений, разделенных запятой. А затем выполним этот запрос со значениями из строк переменной `contents` (рис. 1.52):

```
file = open("Churn_Modelling.csv")
contents = csv.reader(file, delimiter=",")
insert_records = '''INSERT INTO churn_mod ('RowNumber', 'CustomerId',
'Surname',
'CreditScore', 'Geography', 'Gender', 'Age', 'Tenure',
'Balance', 'NumOfProducts', 'HasCrCard',
'IsActiveMember', 'EstimatedSalary', 'Exited')
VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)'''
cur.executemany(insert_records, contents)
```

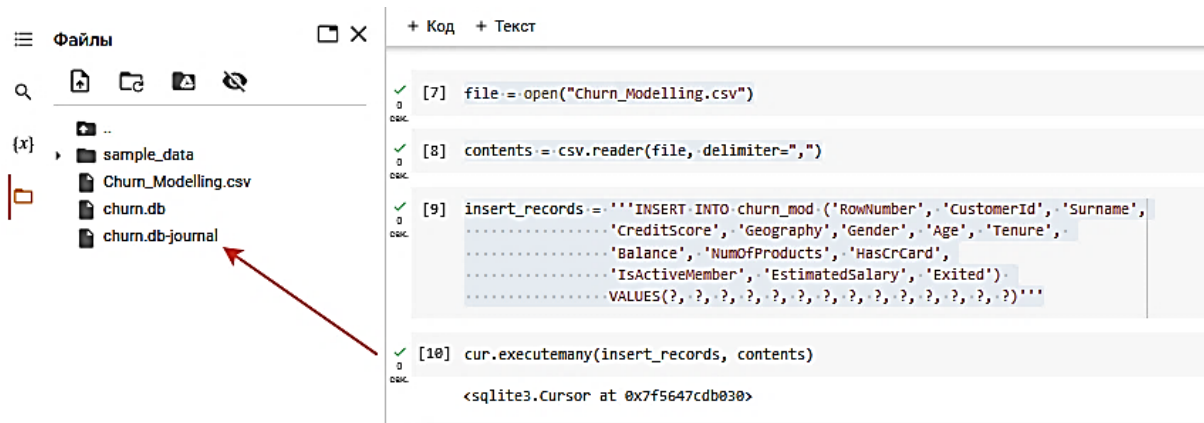


Рис. 1.52. Считывание строк файла .csv
в таблицу базы данных с помощью запроса SQL

Обратите внимание на новый объект в рабочей директории `churn.db-journal`. Все изменения в базе данных `churn.db` временные, и для этих изменений создается журнал. Если не выполнить команду `connection.commit()` и скачать из рабочей среды базу `churn.db`, данных в ней не будет.

Запросим три первые строки созданной таблицы `churn_mod` (рис. 1.53):

```
cur.execute("SELECT * FROM churn_mod;")
cur.fetchmany(3)
```

Как видим, первая строка требует удаления, так как в нее считались названия полей. Условие для удаления составляем с помощью любого столбца для строки, в которой его значение совпадает с именем столбца/поля, а затем еще раз запрашиваем первые три строки таблицы `churn_mod` и видим, что теперь все в порядке (рис. 1.54):

```
cur.execute("DELETE FROM churn_mod WHERE RowNumber='RowNumber';")
cur.execute("SELECT * FROM churn_mod;")
cur.fetchmany(3)
```

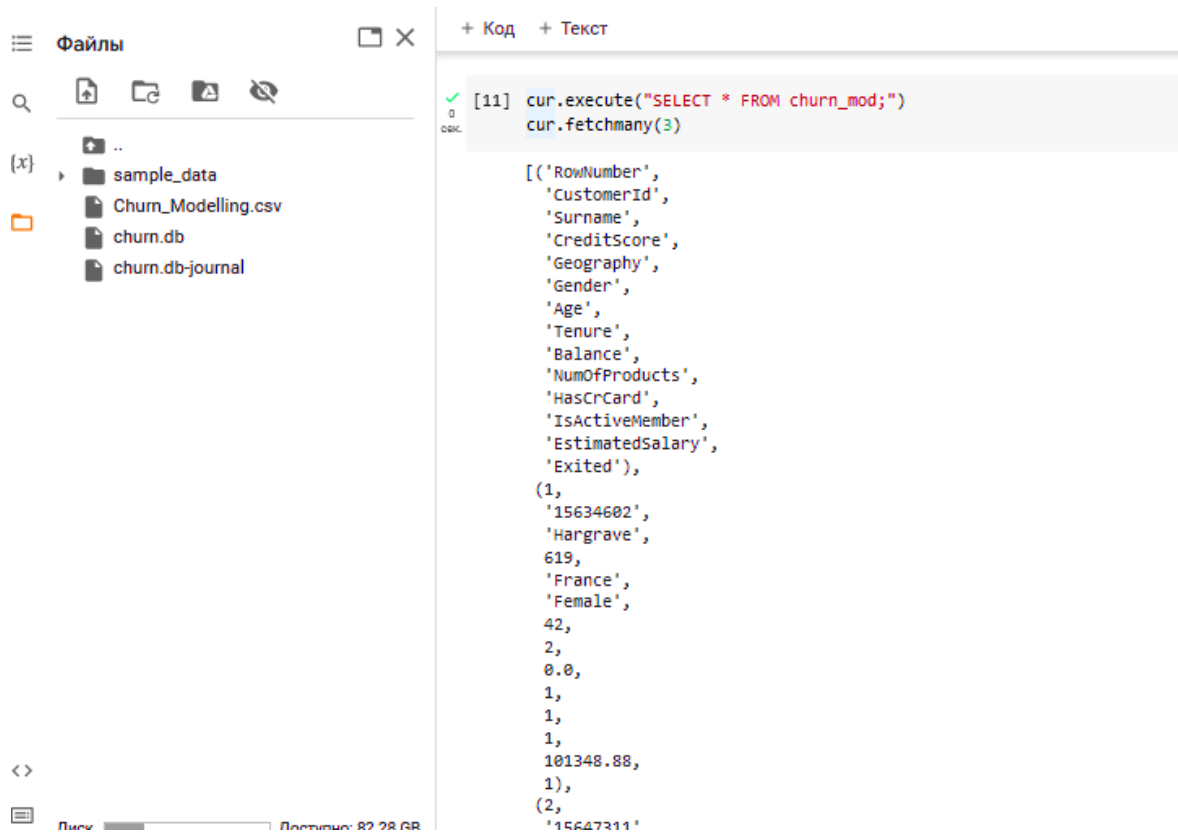


Рис. 1.53. Запрос первых строк таблицы churn_mod

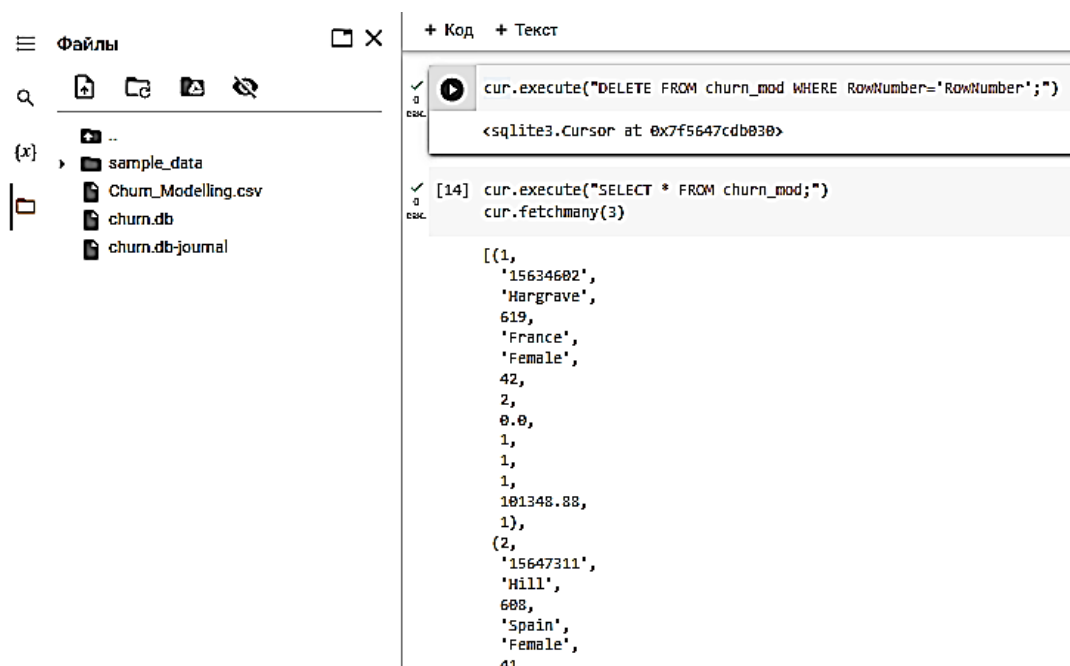


Рис. 1.54. Удаление строки с именами полей

Примем изменения `con.commit()` и журнал `churn.db-journal` автоматически удалится, а у файла `churn.db` после этого будет совершенно иной размер, так как теперь он хранит и структуру таблицы, и ее записи и всю необходимую служебную информацию (рис. 1.55). Также закроем подключение к базе:

```
con.commit()
con.close()
```

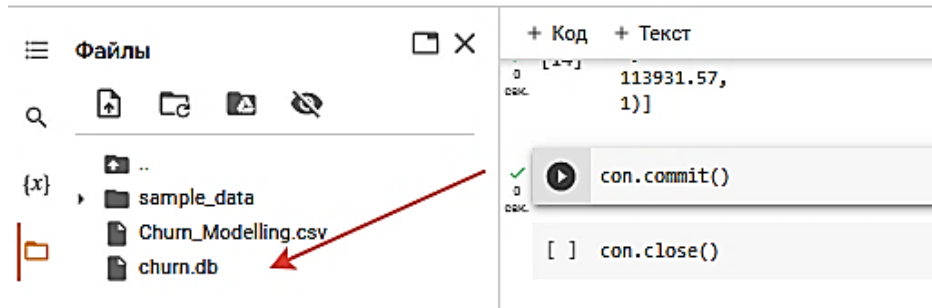


Рис. 1.55. Фиксация изменений в базе данных и закрытие подключения к ней

Скачаем файл `churn.db` из рабочей среды Colab (важно это сделать до закрытия файла `.ipynb`, который с ней работал, так как среда очищается и при повторном открытии этого файла все необходимые элементы должны загружаться или создаваться заново) на компьютер и подключимся к нему через DBeaver (рис. 1.56).

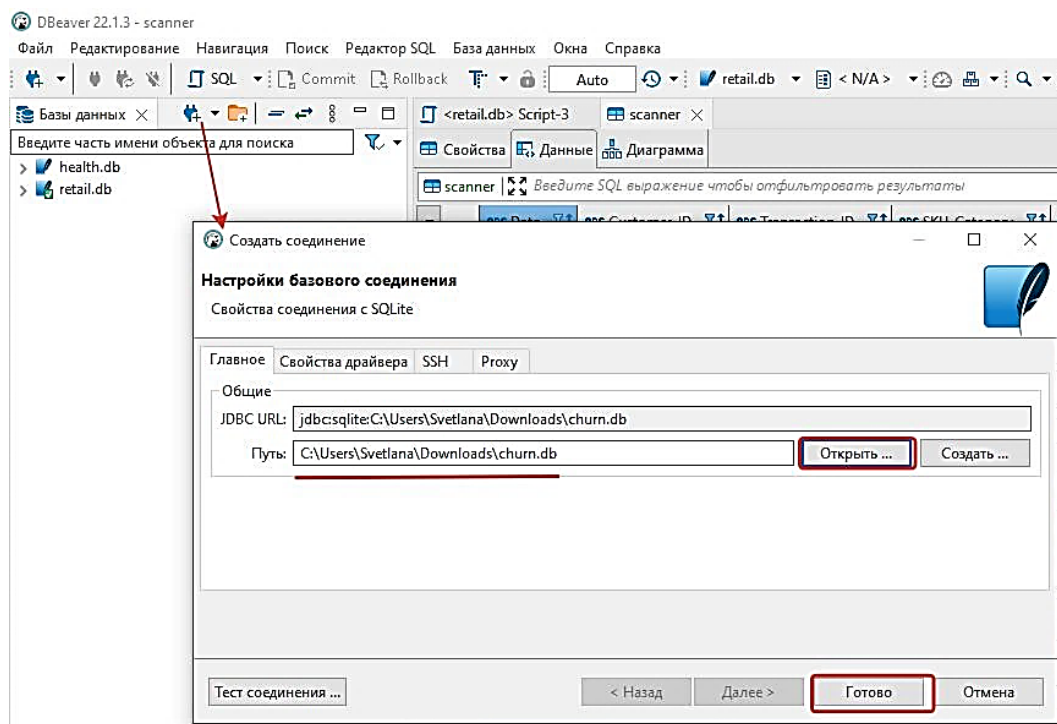
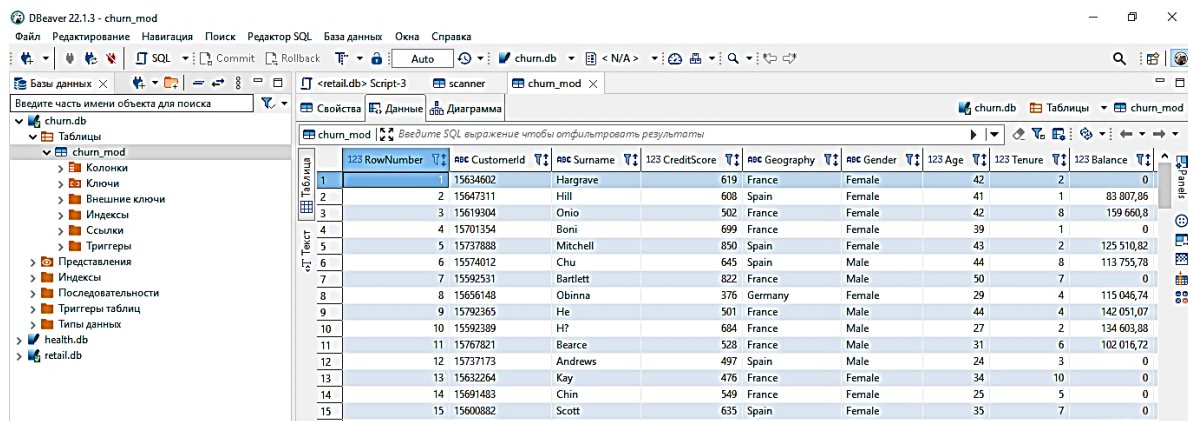


Рис. 1.56. Подключение к файлу `churn.db` на диске компьютера

Откроем churn_mod таблицу с данными для просмотра (рис. 1.57).



The screenshot shows the DBeaver 22.1.3 interface with the 'churn_mod' table selected. The table contains 15 rows of data with columns: RowNumber, CustomerId, Surname, CreditScore, Geography, Gender, Age, Tenure, and Balance. The data is as follows:

RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance
1	15634602	Hargrave	619	France	Female	42	2	0
2	15647311	Hill	608	Spain	Female	41	1	83 807,86
3	15619304	Onio	502	France	Female	42	8	159 660,8
4	15701354	Boni	699	France	Female	39	1	0
5	15737888	Mitchell	850	Spain	Female	43	2	125 510,82
6	15574012	Chu	645	Spain	Male	44	8	113 755,78
7	15592531	Bartlett	822	France	Male	50	7	0
8	15656148	Obinna	376	Germany	Female	29	4	115 046,74
9	15792365	He	501	France	Male	44	4	142 051,07
10	15592389	H?	684	France	Male	27	2	134 603,88
11	15767821	Bearce	528	France	Male	31	6	102 016,72
12	15737173	Andrews	497	Spain	Male	24	3	0
13	15632264	Kay	476	France	Female	34	10	0
14	15691483	Chin	549	France	Female	25	5	0
15	15600882	Scott	635	Spain	Female	35	7	0

Рис. 1.57. Открытие таблицы churn_mod

Данные все загрузились корректно, структура таблицы (метаданные) также соответствует объявленной. Обратите внимание, что считывание первой строки с именами полей не изменило объявленный тип числовых столбцов, хотя данные в первой строке файла – текстовые, тип данных для числовых столбцов не изменился.

Воспользуемся еще одним вариантом создания таблицы базы данных scanner.db на основе файла .csv. В разд 1.3 уже создавалась база данных retail.db на основе файла scanner.csv. В новом Notebook Colab подключим две необходимые библиотеки, одна из которых уже хорошо знакомая sqlite3, а другая очень известная в аналитике табличных данных библиотека pandas (рис. 1.58). Подключимся к базе данных scanner.db (одновременно эта база данных будет создана) и загрузим в рабочую среду файл scanner.csv (ссылку на источник и описание данных файла см. в разд. 1.3):

```
import sqlite3 as sl
import pandas as pd
con = sl.connect('scanner.db')
```

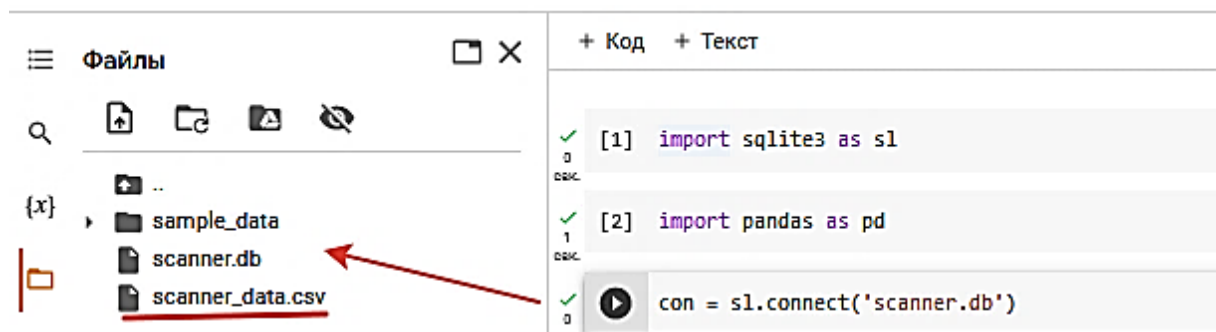


Рис. 1.58. Подключение библиотек и базы данных scanner.db

Считаем файл в переменную `df` типа `dataframe` (это тип данных для табличного представления наборов данных), выведем первые пять строк из набора данных `df` и с помощью метода `.to_sql()` создадим в подключенной базе данных таблицу `scan` на основе данных из таблицы `df` (рис. 1.59). В этой таблице 131706 строк:

```
df=pd.read_csv("scanner_data.csv")
df.head()
df.to_sql('scan', con)
```

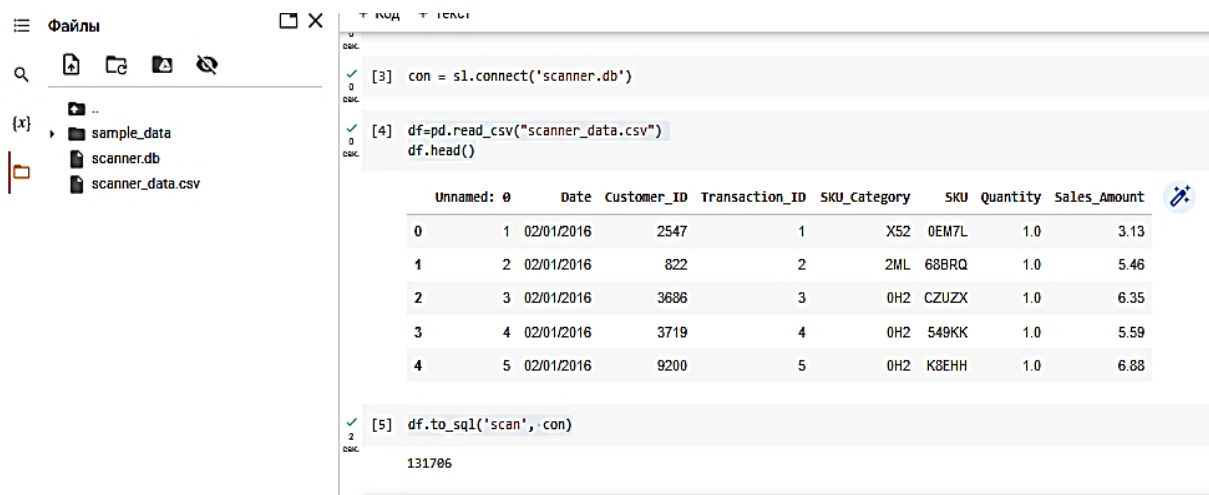


Рис. 1.59. Считывание данных из файла в переменную и создание на ее основе таблицы базы данных SQLite

Данные из `dataframe` сразу передались в созданную таблицу `scan`, без журналирования временных изменений, т.е. изменения в базе уже произошли, и при скачивании файла `scanner.db` из рабочей среды в нем можно обнаружить таблицу с нужными данными. Скачаем файл на компьютер, чтобы впоследствии открыть его в менеджере баз данных DBeaver.

Теперь запросим с помощью созданного объекта `cursor` первые три строки таблицы (рис. 1.60).

```
cur = con.cursor()
cur.execute("SELECT * FROM scan;")
cur.fetchmany(3)
```

Откроем скачанный из рабочей среды Colab файл `scanner.db` в DBeaver (рис. 1.61). Важно, что в этом случае не создавалась структура таблицы – имена полей и их тип были созданы на основе имен столбцов и их типов в `dataframe` (объекте библиотеки `pandas`). И качество этого преобразования вполне удовлетворительное, так как числовые поля получили корректный числовой тип, а не более общий текстовый.

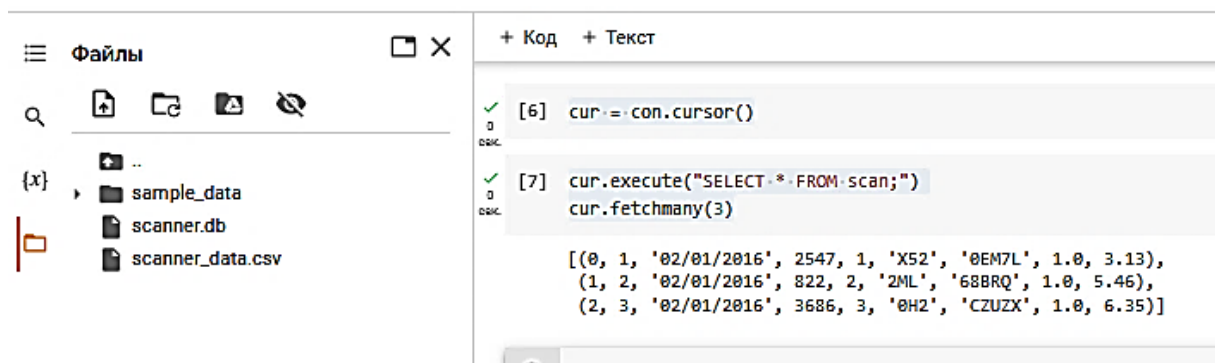


Рис. 1.60. Вывод первых трех строк из таблицы scan

index	Date	Customer_ID	Transaction_ID	SKU_Category	SKU	Quantity	Sales_Amount
0	02/01/2016	2547	1	X52	0EM7L	1	3.13
1	02/01/2016	822	2	2ML	68BRQ	1	5.46
2	02/01/2016	3686	3	0H2	CZUZX	1	6.35
3	02/01/2016	3719	4	0H2	549KK	1	
4	02/01/2016	9200	5	0H2	K8EH4	1	
5	02/01/2016	5010	6	JPI	GVBRG	1	
6	02/01/2016	1666	7	XG4	AHAE7	1	
7	02/01/2016	1666	7	FEW	AHZNS	1	
8	02/01/2016	1253	8	0H2	9STQJ	1	
9	02/01/2016	5541	9	N5F	7IE9S	1	
10	02/01/2016	5541	9	H8O	M15RG	1	
11	02/01/2016	7548	10	N8U	UNJKW	1	
12	02/01/2016	7548	10	JRS	KWDJZ	1	
13	02/01/2016	6865	11	I4V	8ETV5	1	
14	02/01/2016	6044	12	N8U	EMJIS	1	
15	02/01/2016	6044	12	X52	CRK31	1	
16	02/01/2016	592	13	YMU	YJTMP	1	
17	02/01/2016	592	13	P42	B2IW9	1	
18	02/01/2016	6450	14	N8U	4A2M0	1	
19	02/01/2016						

Рис. 1.61. Открытие таблицы scan базы данных scanner.db в Dbeaver

2. Работа с данными

В реляционных системах управления базами данных в качестве высокоуровневого средства выполнения запросов используется язык SQL, который содержит набор операций, необходимых для выполнения действий с базой данных.

Команды SQL делятся на четыре основные категории/языка. Каждый язык определяет подмножество команд для реализации соответствующей цели:

- язык определения/описания данных (Data Definition Language, DDL), команды которого определяют структуру базы данных (в том числе логическую) и позволяют работать с объектами этой базы данных (БД) (например, таблицами или представлениями), среди них:

- CREATE – создание нового объекта БД;
- ALTER – модификация/изменение существующего в БД объекта;
- DROP – удаление существующего в БД объекта;
- TRUNCATE – удаление (усечение) всех записей из таблицы или набора таблиц;

- язык манипулирования данными (Data Manipulation Language, DML), команды которого обеспечивают управление данными внутри схемы, позволяют добавлять, изменять, удалять и выгружать данные из базы, среди них:

- SELECT – выборка данных;
- INSERT – вставка новых данных;
- UPDATE – обновление данных;
- DELETE – удаление данных;
- MERGE – слияние данных;

- язык управления данными (Data Control Language, DCL), команды которого обеспечивают управление доступом к данным, т.е. управления привилегиями, разрешающими или запрещающими выполнение определенных операций над объектами базы данных, среди них:

- GRANT – предоставление пользователям привилегии доступа БД;
- REVOKE – отзыв привилегии, предоставленной командой GRANT;
- DENY – задание запрета, имеющего приоритет над разрешением;

- язык управления транзакциями (Transaction Control Language, TCL). Транзакция – это команда или блок команд DML, которые

рассматриваются как единое целое, т.е. в успешной транзакции фиксируются все внесенные изменения, а при сбое в какой-либо команде транзакции все изменения, внесенные командами транзакции, будут отменены. Команды TCL используются для управления изменениями, сделанными транзакцией, среди них:

- BEGIN – определение начала транзакции;
- COMMIT – сохранение изменений;
- SAVEPOINT – определение точки транзакции, к которой можно будет откатиться;
- ROLLBACK – возвращение БД к состоянию на момент последней команды.

Набор команд, который возвращает информацию из базы данных, называется запросом. Также запрос – это выполняемый фрагмент кода на языке SQL.

Условие – это часть запроса, содержащая по крайней мере одно ключевое слово и соответствующую информацию, которая будет использоваться вместе с этим ключевым словом, например, ссылки на поля и таблицы.

Использование верхнего регистра в названиях команд SQL – это всего лишь соглашение, большинство СУБД нечувствительны к регистру.

В пособии будут рассмотрены только конструкции DDL и DML.

2.1. Операции DDL

Рассмотрим основные операторы языка DDL: CREATE, ALTER, DROP.

С оператором CREATE уже работали при создании таблиц баз данных (в разд. 1.3 в базе данных retail.db была создана таблица scanner, в разд. 1.4 в базе данных churn.db была создана таблица churn_mod). Таблица isurance в базе данных health.db (см. разд. 1.2) и таблица scan в базе данных scanner.db (см. разд. 1.4) были созданы без использования операторов SQL.

Создадим в DBeaver новое подключение – базу данных retail_new.db. Ранее созданные базы данных содержали только одну таблицу. В retail_new.db создадим две связанные таблицы.

Скачаем два связанных файла данных с ресурса kaggle: данные о транзакциях клиентов и данные реакции клиентов на рекламные взаимодействия: https://www.kaggle.com/datasets/regivm/retailtransactiondata?select=Retail_Data_Transactions.csv и https://www.kaggle.com/datasets/regivm/retailtransactiondata?select=Retail_Data_Response.csv.

Файл Retail_Data_Response.csv содержит следующие данные:

- customer_id – идентификатор клиента (текстовая/символьная переменная);
- response – наличие/отсутствие отклика на рекламную компанию (бинарная переменная отклика).

Файл Retail_Data_Transactions.csv содержит следующие данные:

- customer_id – идентификатор клиента (текстовая/символьная переменная);
- trans_date – дата транзакции;
- tran_amount – сумма покупки.

В файле Retail_Data_Response.csv поле customer_id – это первичный ключ (PRIMARY KEY), все значения в этом поле ненулевые и уникальные. В Retail_Data_Transactions.csv первичного ключа нет, так как данные каждого из трех полей могут повторяться, а поле customer_id выступает внешним ключом (FOREIGN KEY), так как идентификаторы клиентов, совершающих транзакции, те же, что и в файле Retail_Data_Response.csv.

При создании таблицы можно задать не только имена и типы столбцов, но и наложить на определенные столбцы дополнительные, необязательные ограничения и модификаторы. Ограничения столбца влияют только на столбец, для которого они определены. Ограничения, которые включают два или более столбца, должны быть определены как ограничения таблицы.

Специальные ключевые слова (CURRENT_TIME, CURRENT_DATE и CURRENT_TIMESTAMP) определяют настройки по умолчанию для столбцов, содержащих дату и время, т.е. будут автоматически добавлять в соответствующее поле для новой строки время, дату или дату и время в формате UTC.

Ограничения для столбца:

NOT NULL – запрещает значения NULL, т.е. пропуски в данных столбца;

UNIQUE – требует уникального значения в соответствующем поле для каждой строки.

Так как NULL не считается значением, то ограничение UNIQUE не гарантирует отсутствие пропусков в данных, т.е. в поле с ограничением UNIQUE могут быть значения NULL и даже не один NULL.

Если в столбце должны быть уникальные значения без пропусков, то у такого столбца должны быть указаны оба ограничения.

Произвольные ограничения задаются пользователем с помощью CHECK (expression).

Также в роли ограничения выступает назначение столбца или набора столбцов на роль первичного ключа. Если набор столбцов обозначен как PRIMARY KEY, то первичный ключ составной. Каждая таблица может иметь только один первичный ключ.

Обозначение PRIMARY KEY подразумевает ограничение UNIQUE, так как значения первичного ключа должны быть уникальными. В случае составного первичного ключа требуется уникальность комбинации значений из набора полей в одной строке.

Первичный ключ также не должен содержать пропусков, однако в SQLite PRIMARY KEY не означает NOT NULL (столбцы PRIMARY KEY, для которых первое условие всегда влечет за собой и второе, это столбцы с типом данных INTEGER). То что PRIMARY KEY не означает NOT NULL, противоречит стандарту SQL и считается ошибкой. Поэтому для столбца с ограничением PRIMARY KEY требуется явное указание ограничения NOT NULL (в случае с составным первичным ключом такое ограничение должно быть хотя бы у одного из столбцов, определяющих первичный ключ).

Если первичный ключ – это столбец с целочисленными значениями, т.е. столбец INTEGER PRIMARY KEY, то он имеет автоматическое ограничение NOT NULL. Такой столбец может быть дополнительно помечен AUTOINCREMENT, т.е. первичный ключ в таком столбце при вставке нового значения будет генерироваться автоматически с соблюдением условия уникальности и отсутствия пропусков.

Помимо PRIMARY KEY, столбцы также могут быть помечены как FOREIGN KEY. Внешний ключ (FOREIGN KEY) – это способ обеспечить ссылочную целостность в базе данных SQLite. Таблица, на поле которой ссылается внешний ключ, называется родительской таблицей, а таблица с внешним ключом называется дочерней таблицей. Внешний ключ в дочерней таблице обычно ссылается на первичный ключ (PRIMARY KEY) в родительской таблице. Внешний ключ может быть определен только в операторе CREATE TABLE.

Поэтому в определении таблиц базы данных retail_new.db зададим не только имена колонок/полей и тип данных в них, но и первичный ключ в первой таблице response, а также связь поля второй таблицы transactions с первичным ключом первой, создав таким образом внешний ключ.

Откроем скрипт для базы данных retail_new.db (рис. 2.1). Для работы со скриптами SQL выполним настройки для DBeaver. Так как есть рекомендация по служебным (ключевым) словам SQL использовать для них верхний регистр, то в командном меню Окна выберем опцию Настройки (рис. 2.2).

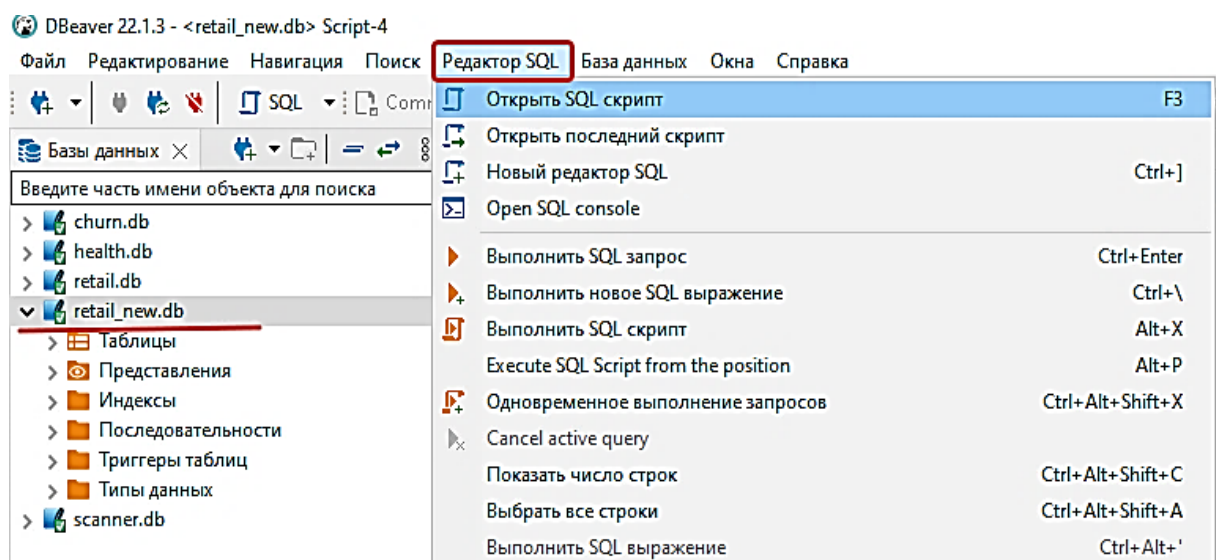


Рис. 2.1. Редактор SQL для базы данных retail_new.db

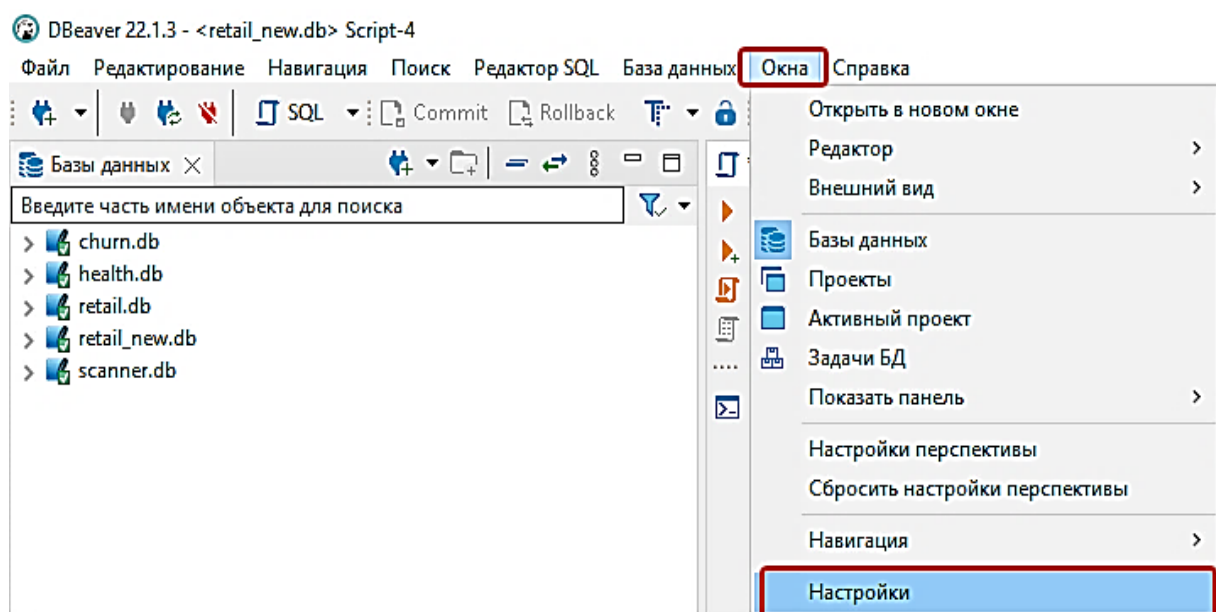


Рис. 2.2. Переход к настройкам DBeaver

Для редактора SQL выберем в навигаторе слева опцию Форматирование и настроим переключатель для Регистра ключевых слов в позицию Upper (рис. 2.3). Теперь автозаполнение по подсказкам команд (клавиша ENTER для автозаполнения) будет происходить в верхнем регистре.

Создадим на основе данных скачанных файлов две связанные таблицы: response и transactions с соответствующими полями и типами данных. Важно объявить не только, какое поле выступает внешним ключом, но и с каким полем родительской таблицы оно связано:

```
CREATE TABLE response (
    customer_id TEXT PRIMARY KEY,
    response INTEGER
);
CREATE TABLE transactions (
    customer_id TEXT,
    trans_date TEXT,
    tran_amount REAL,
    FOREIGN KEY (customer_id) REFERENCES response(customer_id)
);
```

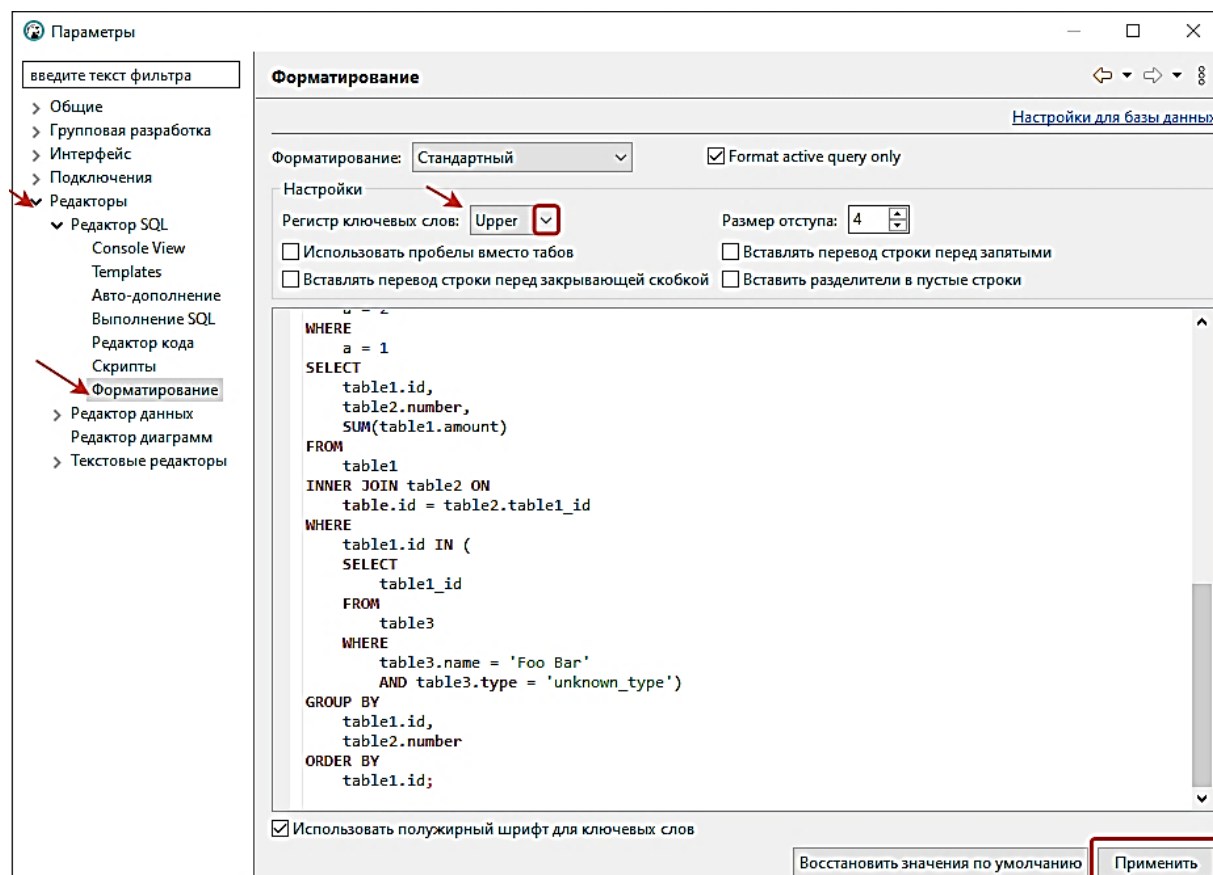


Рис. 2.3. Настройка верхнего регистра для ключевых слов в редакторе SQL

После выполнения кода будут созданы две таблицы, в одной из которых определен первичный ключ, а во второй внешний. Выведем на просмотр схему данных, дважды кликнув на Представление и открыв вкладку Диаграмма (рис. 2.4).

Импортируем данные из файлов (см. разд. 1.3), каждый в соответствующую таблицу. И выведем содержимое этих таблиц на экран (рис. 2.5). Также можно увидеть и схему данных (рис. 2.6).

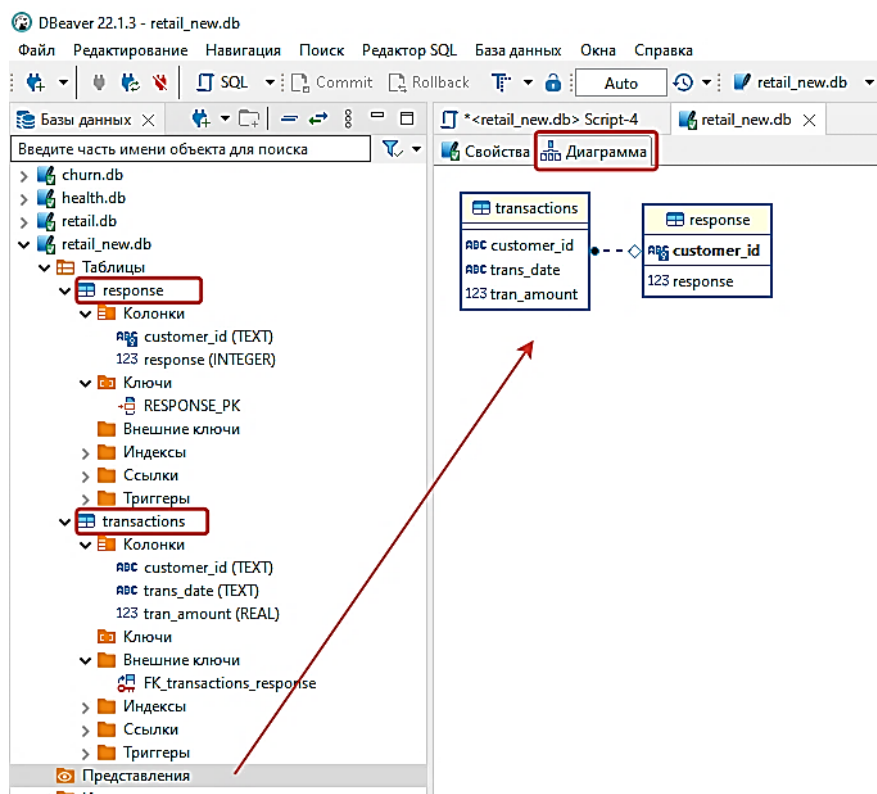


Рис. 2.4. Схема данных для базы retail_new.db

transactions ×

Свойства Данные Диаграмма

retail_new.db Таблицы transactions

transactions

	customer_id	trans_date
448	CS3598	30-Jul-13
449	CS3642	19-May-14
450	CS5948	10-Jul-14
451	CS5582	23-May-14
452	CS6107	23-Jul-11
453	CS5201	02-Jul-11
454	CS1639	05-May-13
455	CS5884	20-Nov-12
456	CS3515	07-Jul-14
457	CS2332	29-Aug-13
458	CS5628	16-Dec-14
459	CS4830	13-Apr-12
460	CS1673	04-Jan-12
461	CS4993	01-Jun-11
462	CS2603	01-Jul-13
463	CS1800	20-Nov-13
464	CS3911	09-Jun-12
465	CS5465	05-Dec-11
466	CS2373	29-Mar-12

Запись

Save Cancel Script

600+ Rows: 1

*<retail_new.db> Script-4 response ×

Свойства Данные Диаграмма

response Введите SQL выражение чтобы от

	customer_id	response
58	CS1169	0
59	CS1170	0
60	CS1171	0
61	CS1172	1
62	CS1173	1
63	CS1174	0
64	CS1175	0
65	CS1176	0
66	CS1177	0
67	CS1178	0
68	CS1179	0
69	CS1180	0
70	CS1181	0
71	CS1182	1
72	CS1183	0
73	CS1184	0
74	CS1185	0
75	CS1186	0
76	CS1187	0
77	CS1188	0
78	CS1189	0
79	CS1190	0
80	CS1191	1
81	CS1192	0

Запись

Save Cancel Script

Рис. 2.5. Параллельный просмотр содержимого двух таблиц базы данных retail_new.db

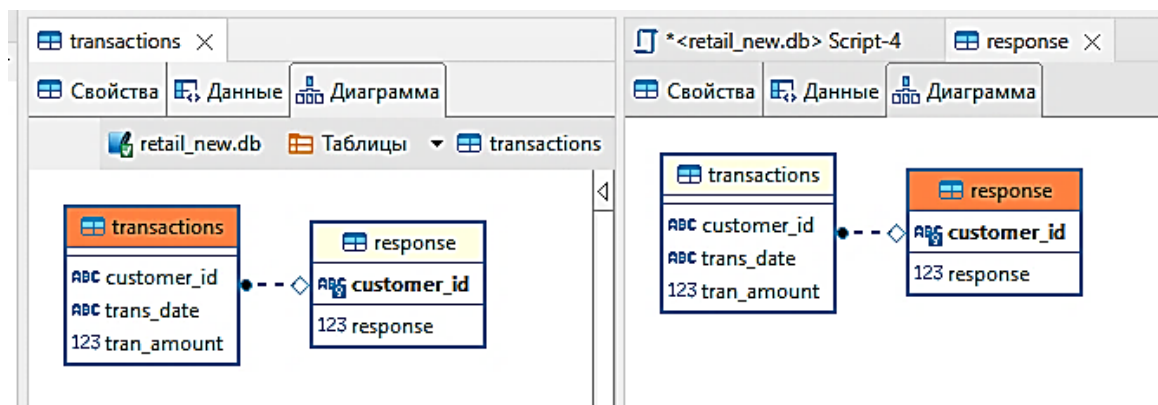


Рис. 2.6. Параллельный просмотр схем двух таблиц базы данных retail_new.db

В SQLite можно использовать оператор ALTER TABLE для добавления столбца.

А вот для переименования столбца или удаления столбца в таблице использовать этот оператор не получится. Для удаления/переименования столбца нужно будет переименовать исходную таблицу, создать новую таблицу и скопировать данные в новую таблицу.

Используем оператор ALTER TABLE для добавления столбца Price (как результата отношения Sales_Amount – стоимости проданного товара к Quantity – количеству проданного товара) в таблицу scan базы данных scanner.db. База данных scanner.db была создана ранее (см. разд. 1.4) и подключена в менеджере баз данных DBeaver.

Откроем для scanner.db новый SQL скрипт и выполним запрос на добавление вычисляемого столбца (рис. 2.7, 2.8):

```
ALTER TABLE scan
ADD Price REAL AS (Sales_Amount / Quantity);
```

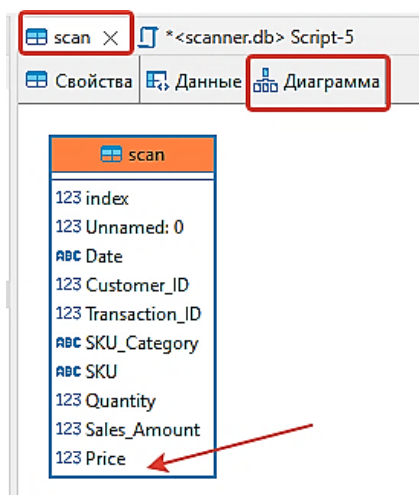


Рис. 2.7. Схема обновленной таблицы scan

	Date	123 Customer_ID	123 Transaction_ID	ABC SKU_Category	ABC SKU	123 Quantity	123 Sales_Amount	123 Price
1	01/2016	2 547	1	X52	0EM7L	1	3,13	3,13
2	01/2016	822	2	2ML	68BRQ	1	5,46	5,46
3	01/2016	3 686	3	0H2	CZUZX	1	6,35	6,35
4	01/2016	3 719	4	0H2	549KK	1	5,59	5,59
5	01/2016	9 200	5	0H2	K8EHH	1	6,88	6,88
6	01/2016	5 010	6	JPI	GVBRC	1	10,77	10,77
7	01/2016	1 666	7	XG4	AHAE7	1	3,65	3,65
8	01/2016	1 666	7	FEW	AHZNS	1	8,21	8,21
9	01/2016	1 253	8	0H2	9STQJ	1	8,25	8,25
10	01/2016	5 541	9	N5F	7IE9S	1	8,18	8,18
11	01/2016	5 541	9	H8O	M15RG	1	6,35	6,35
12	01/2016	7 548	10	N8U	UNJKW	1	2,11	2,11
13	01/2016	7 548	10	JR5	KWDJZ	1	5,38	5,38
14	01/2016	6 865	11	I4Y	8ETY5	1	8,04	8,04
15	01/2016	6 044	12	N8U	EMJ1S	1	3,62	3,62
16	01/2016	6 044	12	X52	CRK31	1	3,96	3,96
17	01/2016	592	13	YMI	YJTMP	1	11,65	11,65
18	01/2016	592	13	P42	B2IW9	1	9	9
19	01/2016	6 450	14	N8U	4A2M0	1	7,02	7,02
20	01/2016	3 874	15	FEW	F79YP	1	16,91	16,91
21	01/2016	3 874	15	SFC	26XOE	2	4,75	2,375
22	01/2016	5 599	16	7C6	OZEQ8	1	7,85	7,85

Рис. 2.8. Записи таблицы scan с новым столбцом Price

А теперь удалим явно неинформативные столбцы Unnamed: 0 и index. Для этого переименуем исходную таблицу scan в scan_old, создадим таблицу scan, определив столбцы, которые должны остаться, и их тип данных, а затем вставим в новую таблицу записи из старой, выбрав только нужные поля (рис. 2.9, 2.10).

Так как таблица scan_old после вставки ее столбцов в новую таблицу больше не нужна, то она удаляется (рис. 2.10):

```
ALTER TABLE scan RENAME TO scan_old;
```

```
CREATE TABLE scan (
    Date TEXT,
    Customer_ID INTEGER,
    Transaction_ID INTEGER,
    SKU_Category TEXT,
    SKU TEXT,
    Quantity REAL,
    Sales_Amount REAL,
    Price REAL
);
```

```
INSERT INTO scan (
    Date, Customer_ID, Transaction_ID, SKU_Category,
    SKU, Quantity, Sales_Amount, Price)
SELECT Date, Customer_ID, Transaction_ID, SKU_Category,
```

```
SKU,Quantity, Sales_Amount, Price
FROM scan_old;
```

```
DROP TABLE scan_old;
```

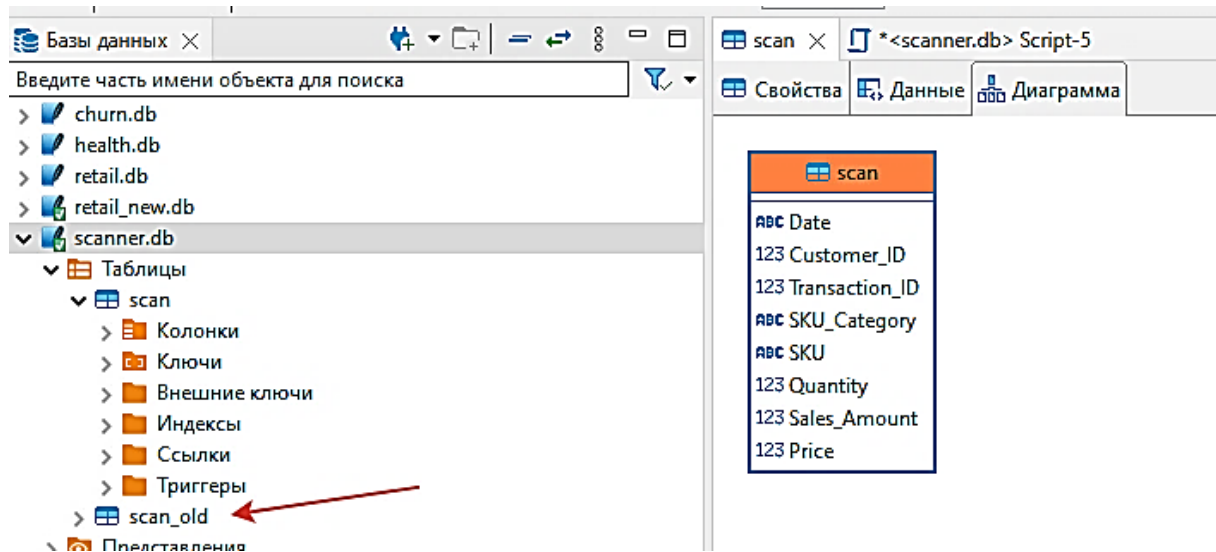


Рис. 2.9. Схема новой таблицы scan без лишних столбцов
(запрос на удаление старой версии еще не выполнен)

	asc Date	123 Customer_ID	123 Transaction_ID	asc SKU_Category	asc SKU	123 Quantity	123 Sales_Amount	123 Price
1	02/01/2016	2 547	1	X52	0EM7L	1	3,13	3,13
2	02/01/2016	822	2	ZML	688RQ	1	5,46	5,46
3	02/01/2016	3 686	3	0H2	CZUZX	1	6,35	6,35
4	02/01/2016	3 719	4	0H2	549KK	1	5,59	5,59
5	02/01/2016	9 200	5	0H2	K8EHH	1	6,88	6,88
6	02/01/2016	5 010	6	JPI	GVBR	1	10,77	10,77
7	02/01/2016	1 666	7	XG4	AHAE7	1	3,65	3,65
8	02/01/2016	1 666	7	FEW	AHZNS	1	8,21	8,21
9	02/01/2016	1 253	8	0H2	9STQJ	1	8,25	8,25
10	02/01/2016	5 541	9	N5F	7IE9S	1	8,18	8,18
11	02/01/2016	5 541	9	H80	M1SRG	1	6,35	6,35
12	02/01/2016	7 548	10	N8U	UNJRW	1	2,11	2,11
13	02/01/2016	7 548	10	JR5	KWDJZ	1	5,38	5,38
14	02/01/2016	6 865	11	I4V	8ETY5	1	8,04	8,04

Рис. 2.10. Записи новой таблицы scan без лишних столбцов

2.2. Операции DML: INSERT, UPDATE, DELETE

SQLite поддерживает основные операторы языка DML: SELECT, INSERT, UPDATE, DELETE.

Начнем знакомство с DML с INSERT, UPDATE, DELETE – это команды изменения содержимого таблиц: обновление, вставка или удаление строк.

Оператор INSERT был использован ранее при вставке данных из файла в таблицу churn_mod базы данных churn.db (см. разд. 1.4 и рис. 1.52) в предыдущем разделе для вставки данных из старой версии таблицы scan_old в новую версию с меньшим количеством столбцов.

Для демонстрации работы операторов INSERT, UPDATE, DELETE используем базу данных scanner.db и таблицу scan, которая была изменена в предыдущем разделе.

Синтаксис для оператора INSERT:

```
INSERT INTO table
(column1, column2, ... )
VALUES
(expression1, expression2, ... ),
(expression1, expression2, ... ),
...;
```

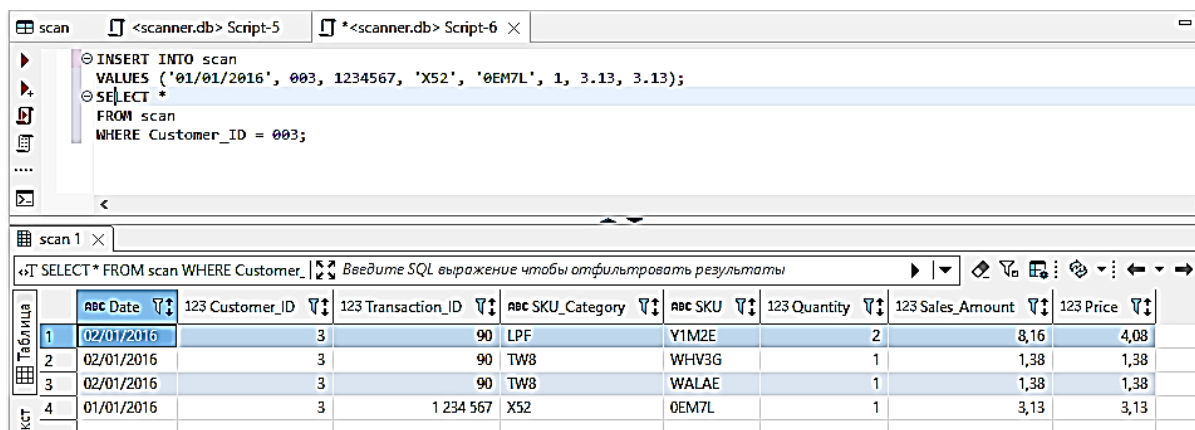
Вставим строку с фиктивными данными в таблицу scan: ('01/01/2016', 003, 1234567, 'X52', '0EM7L', 1, 3.13, 3.13). Если не перечисляются имена столбцов, в которые записываются данные, то вставляемые значения должны соответствовать порядку перечисления имен столбцов при создании таблицы.

Однако, если в любом произвольном порядке, и необязательно в полном составе, указать имена столбцов, в которые планируется добавить данные, то вставляемые значения должны соблюдать тот же порядок следования:

```
INSERT INTO scan
VALUES ('01/01/2016', 003, 1234567, 'X52', '0EM7L', 1, 3.13, 3.13);
```

Запросим все транзакции этого заказчика (рис. 2.11):

```
SELECT *
FROM scan
WHERE Customer_ID = 003;
```



	abc Date	123 Customer_ID	123 Transaction_ID	abc SKU_Category	abc SKU	123 Quantity	123 Sales_Amount	123 Price
1	02/01/2016	3	90	LPF	Y1M2E	2	8,16	4,08
2	02/01/2016	3	90	TW8	WHV3G	1	1,38	1,38
3	02/01/2016	3	90	TW8	WALAE	1	1,38	1,38
4	01/01/2016	3	1 234 567	X52	0EM7L	1	3,13	3,13

Рис. 2.11. Запрос на вывод внесенной записи

Синтаксис для оператора UPDATE:

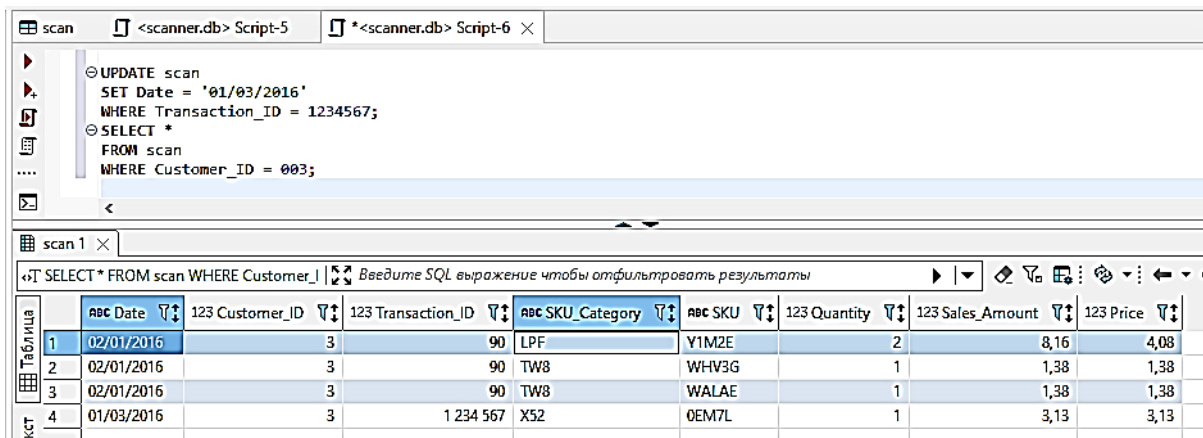
```
UPDATE table
  SET column1 = expression1 | DEFAULT,
    column2 = expression2 | DEFAULT,
    ...
  [WHERE conditions];
```

Обновим только что внесенную запись, изменив дату на '01/03/2016'. Обратите внимание, что при обновлении обязательно указывается условие для отбора записей, в которых произойдет обновление (WHERE <условие>), иначе изменение будет применено для этого столбца/столбцов во всех записях. Условий для определения нужных для обновления записей может быть несколько:

```
UPDATE scan
SET Date = '01/03/2016'
WHERE Transaction_ID = 1234567;
```

Также выполним запрос на вывод транзакций заказчика с этим значением поля Customer_ID (рис. 2.12):

```
SELECT *
FROM scan
WHERE Customer_ID = 003;
```



The screenshot shows a database management interface with two tabs: 'scan' and '<scanner.db> Script-5'. The 'scan' tab is active, displaying the following SQL queries:

```
UPDATE scan
SET Date = '01/03/2016'
WHERE Transaction_ID = 1234567;
SELECT *
FROM scan
WHERE Customer_ID = 003;
```

Below the queries, a table titled 'scan 1' is displayed, showing the results of the SELECT query. The table has 10 columns: Date, Customer_ID, Transaction_ID, SKU_Category, SKU, Quantity, Sales_Amount, and Price. The data is as follows:

ABC Date	123 Customer_ID	123 Transaction_ID	ABC SKU_Category	ABC SKU	123 Quantity	123 Sales_Amount	123 Price
02/01/2016	3	90	LPF	Y1M2E	2	8,16	4,08
02/01/2016	3	90	TW8	WHV3G	1	1,38	1,38
02/01/2016	3	90	TW8	WALAE	1	1,38	1,38
01/03/2016	3	1 234 567	X52	OEM7L	1	3,13	3,13

Рис. 2.12. Результат обновления записи

Синтаксис оператора DELETE:

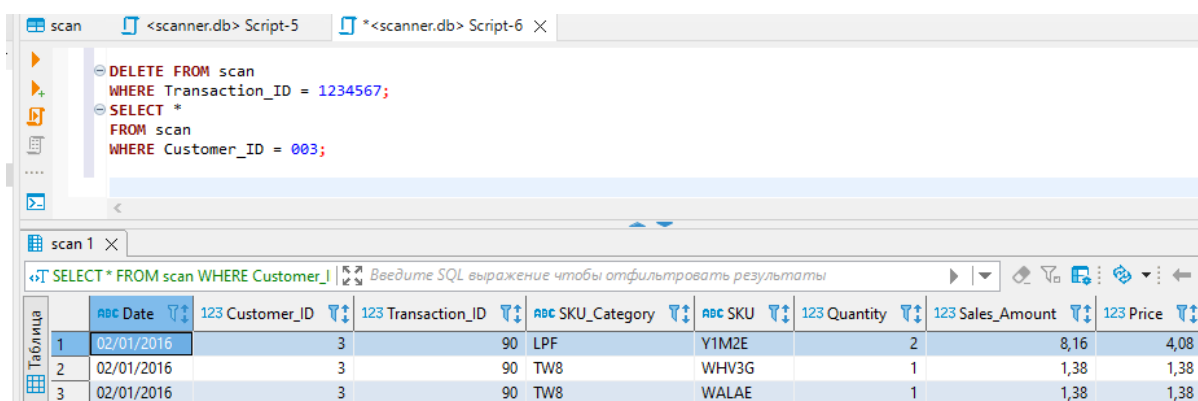
```
DELETE FROM table
  [WHERE conditions];
```

Удалим тренировочную запись из исходного набора данных. Так же как и в случае обновления, чтобы не произошло удаление всех записей таблицы, необходимо указать условие для отбора нужных записей:

```
DELETE FROM scan
WHERE Transaction_ID = 1234567;
```

И снова выполним запрос на вывод записей этого заказчика, чтобы убедиться, что удаление произошло (рис. 2.13):

```
SELECT *
FROM scan
WHERE Customer_ID = 003;
```



The screenshot shows a database management interface with two tabs: '<scanner.db> Script-5' and '<scanner.db> Script-6'. The first tab contains the SQL script: `DELETE FROM scan WHERE Transaction_ID = 1234567;` and `SELECT * FROM scan WHERE Customer_ID = 003;`. The second tab shows the results of the second query in a table with 10 columns: Date, Customer_ID, Transaction_ID, SKU_Category, SKU, Quantity, Sales_Amount, and Price. The results are as follows:

ABC Date	123 Customer_ID	123 Transaction_ID	ABC SKU_Category	ABC SKU	123 Quantity	123 Sales_Amount	123 Price
02/01/2016	3	90	LPF	Y1M2E	2	8,16	4,08
02/01/2016	3	90	TW8	WHV3G	1	1,38	1,38
02/01/2016	3	90	TW8	WALAE	1	1,38	1,38

Рис. 2.13. Результат удаления записи

2.3. Операции DML: SELECT

Команда SELECT используется для извлечения данных из базы данных. Любая операция чтения из базы данных на языке SQL задается оператором SELECT.

В своей простейшей форме оператор SELECT имеет синтаксис:

```
SELECT expressions
FROM tables
[WHERE conditions];
```

Оператор SELECT получает строки из множества таблиц (возможно, пустого). В предложении FROM перечисляются одна или несколько таблиц, служащих источниками данных для SELECT. Если указано несколько источников, результатом будет декартово произведение (перекрестное соединение) всех их строк. В предложении FROM указывается имя таблицы или представления и очень часто алиас (alias, псевдоним), т.е. альтернативное имя для элемента списка FROM, оно используется для сокращения записи и полностью скрывает настоящее имя элемента.

Псевдонимы (алиасы) SQLite можно использовать для создания временного имени для столбцов или таблиц.

Псевдонимы столбцов используются для облегчения чтения заголовков столбцов в результирующем наборе.

Псевдонимы таблиц используются для сокращения SQL-кода, чтобы его было легче читать.

Синтаксис для задания псевдонимов столбцов/таблиц:

```
column_name [ AS ] alias_name  
table_name [ AS ] alias_name
```

Ключевое слово AS не является обязательным в SQLite, его можно опустить.

Откроем новый скрипт для retail_new.db (создана в разд. 1.4) и выполним запрос к таблице transactions, имя которой заменим на псевдоним t, также на псевдонимы заменим и имена столбцов и выставим условие, что сумма должна быть больше 100 (рис. 2.14):

```
SELECT customer_id, trans_date AS date, tran_amount AS amount  
FROM transactions AS t  
WHERE amount>100;
```

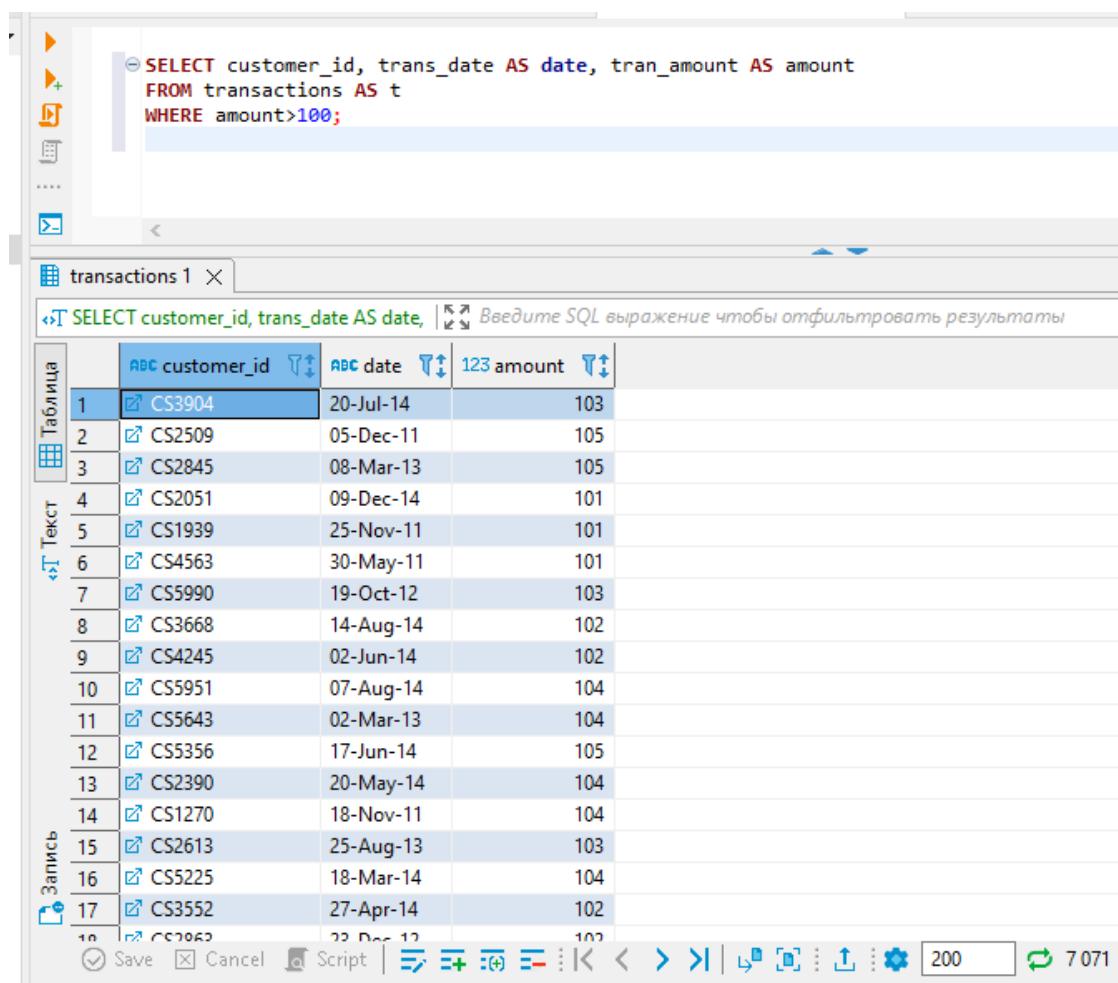


Рис. 2.14. Использование псевдонимов при запросе на записи с суммой больше 100 из таблицы transactions

Просмотреть следующую порцию записей, если количество строк в запросе большое, можно по кнопке  внизу окна с результатом запроса, а кнопка  позволяет подсчитать точное число строк.

2.3.1. Условие WHERE, операторы сравнения LIKE и IN

Необязательное предложение WHERE содержит список условий (выражений), выдающих результат типа boolean (т.е. логический тип данных TRUE/FALSE, но также возможно и значение NULL). Строки, не удовлетворяющие комбинации условий, исключаются из результата.

Рассмотрим множественные условия в запросах. Например, выберем из таблицы transactions тех заказчиков, у которых значение суммы в транзакции находится в интервале от 90 до 100 (с включением границ интервала). Две альтернативные конструкции с использованием ключевого слова BETWEEN и без него дают идентичный результат (рис. 2.15):

```
SELECT *  
FROM transactions  
WHERE tran_amount>90 AND tran_amount<100;  
  
SELECT *  
FROM transactions  
WHERE tran_amount BETWEEN 90 AND 100;
```

Если же необходимо выбрать все записи, в которых поле tran_amount принимает несколько конкретных значений, например 90, 95 и 100, то можно использовать отдельные равенства, соединив их OR (рис. 2.16):

```
SELECT *  
FROM transactions  
WHERE tran_amount=90  
OR tran_amount=95  
OR tran_amount=100;  
  
SELECT *  
FROM transactions  
WHERE tran_amount IN (90, 95, 100);
```

SQL Query:

```
SELECT *
FROM transactions
WHERE tran_amount >= 90 AND tran_amount <= 100;
```

SQL Query:

```
SELECT *
FROM transactions
WHERE tran_amount BETWEEN 90 AND 100;
```

transactions 1

SQL Query:

```
SELECT * FROM transactions WHERE tran
```

Введите SQL выражение чтобы отфильтровать результаты

	customer_id	trans_date	tran_amount
4	CS6043	15-Jan-14	98
5	CS4655	30-Dec-13	93
6	CS4102	09-Jul-11	96
7	CS5910	01-Jan-12	98
8	CS2287	23-Dec-11	94
9	CS3434	22-Feb-13	98
10	CS4905	13-Apr-12	93
11	CS2403	27-Jul-14	94
12	CS5811	23-Aug-13	91
13	CS3158	06-May-12	91
14	CS5653	04-Sep-13	98
15	CS5463	27-Aug-14	91
16	CS3039	01-May-12	92
17	CS1319	26-Sep-11	95
18	CS2437	13-Nov-11	90
19	CS1273	01-Feb-12	91
20	CS4701	15-Jan-15	95

Save Cancel Script 200 15 621

Рис. 2.15. Результат выполнения запроса
(сумма транзакции в интервале от 90 до 100)

SQL Query:

```
SELECT *
FROM transactions
WHERE tran_amount=90
OR tran_amount=95
OR tran_amount=100;
```

SQL Query:

```
SELECT *
FROM transactions
WHERE tran_amount IN (90, 95, 100);
```

transactions 1

SQL Query:

```
SELECT * FROM transactions WHERE tran
```

Введите SQL выражение чтобы отфильтровать результаты

	customer_id	trans_date	tran_amount
4	CS4991	20-Jul-13	90
5	CS2359	04-Jun-12	90
6	CS2302	09-Sep-12	100
7	CS5277	12-Jul-11	90
8	CS1790	04-Oct-12	100
9	CS5155	25-Dec-11	100
10	CS4581	04-Jun-13	95
11	CS2122	20-Feb-12	90
12	CS4094	24-Jul-13	100
13	CS5664	02-Nov-13	95
14	CS4077	22-Jul-12	90
15	CS1710	10-Jun-13	100
16	CS2554	05-Oct-11	100
17	CS2798	30-May-12	100

Save Cancel Script 200 4 253

Рис. 2.16. Результат выполнения запроса
(сумма транзакции принимает одно из значений 90, 95, 100)

Для поиска заданного шаблона в предложении WHERE используется оператор LIKE. В шаблоне, кроме искомых символов, можно использовать подстановочные знаки: знак процента % используется вместо любого числа символов от 0 до фактически максимально возможного в данном поле, а знак нижнего подчеркивания _ заменяет собой один любой символ.

Прочитаем из таблицы transactions те записи, для которых дата транзакции начинается с цифр 11 и буква D – это начало названия месяца. В SQLite нет специального типа дата/время и поле дат имеет в таблице transactions тип TEXT.

Шаблон для поиска '11_D%', альтернативный вариант '11-D%', есть и другие варианты шаблонов, которые приведут к тому же результату. Например '11%D%'. В результате получим 361 запись (рис. 2.17):

```
SELECT *
FROM transactions
WHERE trans_date LIKE '11%D%';
```

	customer_id	trans_date	tran_amount
4	CS4378	11-Dec-11	84
5	CS5975	11-Dec-13	88
6	CS1338	11-Dec-14	92
7	CS2421	11-Dec-12	87
8	CS4367	11-Dec-13	88
9	CS2743	11-Dec-12	59
10	CS4085	11-Dec-11	85
11	CS3077	11-Dec-14	61
12	CS5171	11-Dec-12	54
13	CS3555	11-Dec-11	37
14	CS3975	11-Dec-11	36
15	CS5828	11-Dec-12	81
16	CS4911	11-Dec-12	101
17	CS4629	11-Dec-13	69

Рис. 2.17. Результат выполнения запроса
(дата транзакции соответствует шаблону '11%D%')

Возможные операторы сравнения перечислены в табл. 2.1.

Ответим на вопрос: сколько заказчиков, чей ID заканчивается на цифру 7, в июле 2013 г. совершили транзакции на сумму 90, 95 или 100? Нам нужно составить условия отбора для всех трех полей таблицы и соединить их оператором AND. Обратите внимание, что для данных

декабря в шаблоне достаточно было указать только первую букву месяца, а в случае июля требуется включить в шаблон три (рис. 2.18):

```
SELECT *
FROM transactions
WHERE trans_date LIKE '%Jul%3'
AND customer_id LIKE '%7'
AND tran_amount IN (90, 95, 100);
```

Таблица 2.1

Операторы сравнения

Операторы сравнения	Описание
=	Равно
<>	Не равно
!=	Не равно
>	Больше чем
>=	Больше или равно
<	Меньше чем
<=	Меньше или равно
IN ()	Соответствует значению в списке
NOT	Отрицает условие
BETWEEN	В пределах диапазона (включительно)
IS NULL	Значение NULL
NOT NULL	Не NULL значение
LIKE	Сопоставление с шаблоном % и _

SELECT *

FROM transactions

WHERE trans_date LIKE '%Jul%3'

AND customer_id LIKE '%7'

AND tran_amount IN (90, 95, 100);

transactions 1

SELECT * FROM transactions WHERE tran

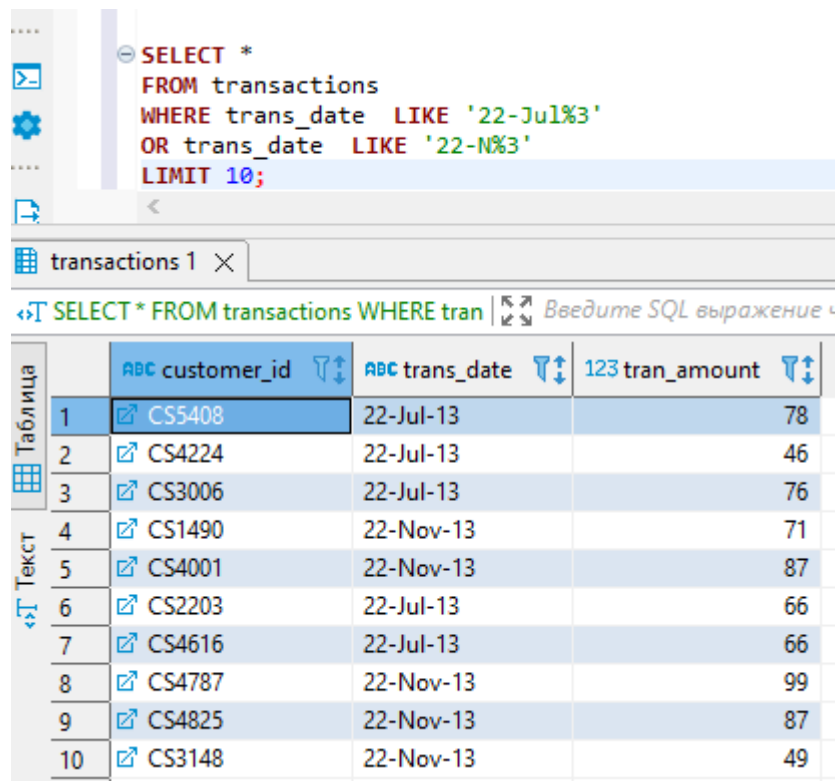
Введите SQL выражение чтобы он

	customer_id	trans_date	tran_amount
1	CS1577	26-Jul-13	95
2	CS3267	22-Jul-13	100
3	CS2637	22-Jul-13	90
4	CS4197	06-Jul-13	100
5	CS3897	20-Jul-13	95
6	CS3897	13-Jul-13	100
7	CS2987	20-Jul-13	95
8	CS2237	21-Jul-13	90
9	CS1367	01-Jul-13	100
10	CS1897	11-Jul-13	100
11	CS2777	17-Jul-13	90

Рис. 2.18. Заказчики, чей ID заканчивается на цифру 7, в июле 2013 г. совершившие транзакции на сумму 90, 95 или 100

В DBeaver на просмотр выводится не более 200 записей запроса (рис. 2.19), последующие порции дозагружаются по кнопке , но можно изначально ограничить количество выводимых записей, добавив в конец запроса оператор LIMIT с указанием количества записей:

```
SELECT *
FROM transactions
WHERE trans_date LIKE '22-Jul%3'
OR trans_date LIKE '22-Nov%3'
LIMIT 10;
```



The screenshot shows the DBeaver interface. At the top, the SQL editor contains the query: `SELECT * FROM transactions WHERE trans_date LIKE '22-Jul%3' OR trans_date LIKE '22-Nov%3' LIMIT 10;`. Below the editor, a tab labeled 'transactions 1' is active. The results are displayed in a table view with columns: 'customer_id', 'trans_date', and 'tran_amount'. The table contains 10 rows of data.

	customer_id	trans_date	tran_amount
1	CS5408	22-Jul-13	78
2	CS4224	22-Jul-13	46
3	CS3006	22-Jul-13	76
4	CS1490	22-Nov-13	71
5	CS4001	22-Nov-13	87
6	CS2203	22-Jul-13	66
7	CS4616	22-Jul-13	66
8	CS4787	22-Nov-13	99
9	CS4825	22-Nov-13	87
10	CS3148	22-Nov-13	49

Рис. 2.19. Ограничение на количество записей в запросе

2.3.2. Исключение дублей, сортировка, группировка и агрегации

Полный синтаксис оператора SELECT:

```
SELECT [DISTINCT]
      expressions
FROM tables
[WHERE conditions]
[GROUP BY expressions]
[HAVING condition]
[ORDER BY expression [ ASC | DESC | USING operator ]]
[LIMIT [ number_rows | ALL]]
```

Если указано SELECT DISTINCT, то все повторяющиеся строки исключаются из результирующего набора, поведение по умолчанию SELECT ALL сохраняет все строки, ALL можно опустить.

Для таблицы scan базы данных scanner.db сделаем выборку по идентификаторам товарных позиций (артикулам) для одной из товарных категорий (2.20):

```
SELECT SKU_Category, SKU
FROM scan
WHERE SKU_Category = 'X52';
```

Таблица	SKU_Category	SKU
1	X52	OEM7L
2	X52	CRK31
3	X52	T5V5W
4	X52	T5V5W
5	X52	MYQAP
6	X52	3N11J
7	X52	KUXIK
8	X52	1W8E0
9	X52	JSNFH
10	X52	JYR24
11	X52	WTLC3
12	X52	T5V5W
13	X52	T5V5W
14	X52	KUXIK
15	X52	QLA1L
16	X52	3N11J
17	X52	QLA1L

Рис. 2.20. Результат выполнения запроса (артикул проданных товаров в категории 'X52')

Как видим, таблица содержит дубли, так как одни и те же товары встречаются в разных транзакциях. Добавим в SELECT предложение DISTINCT (рис. 2.21):

```
SELECT DISTINCT
    SKU_Category, SKU
FROM
    scan
WHERE
    SKU_Category = 'X52';
```

```

SELECT DISTINCT
  SKU_Category, SKU
FROM
  scan
WHERE
  SKU_Category = 'X52';

```

	SKU_Category	SKU
1	X52	OEM7L
2	X52	CRK31
3	X52	T5V5W
4	X52	MYQAP
5	X52	3N11J
6	X52	KUXIK
7	X52	1W8E0
8	X52	JSNFH
9	X52	JYR24
10	X52	WTL3
11	X52	QLA1L
12	X52	GZN4P
13	X52	OA457
14	X52	96A00
15	X52	ELXU9
16	X52	BUR2C
17	X52	IXYNB
18	X52	Q057M4

Рис. 2.21. Результат выполнения запроса
(артикул проданных товаров в категории 'X52' без дубликатов)

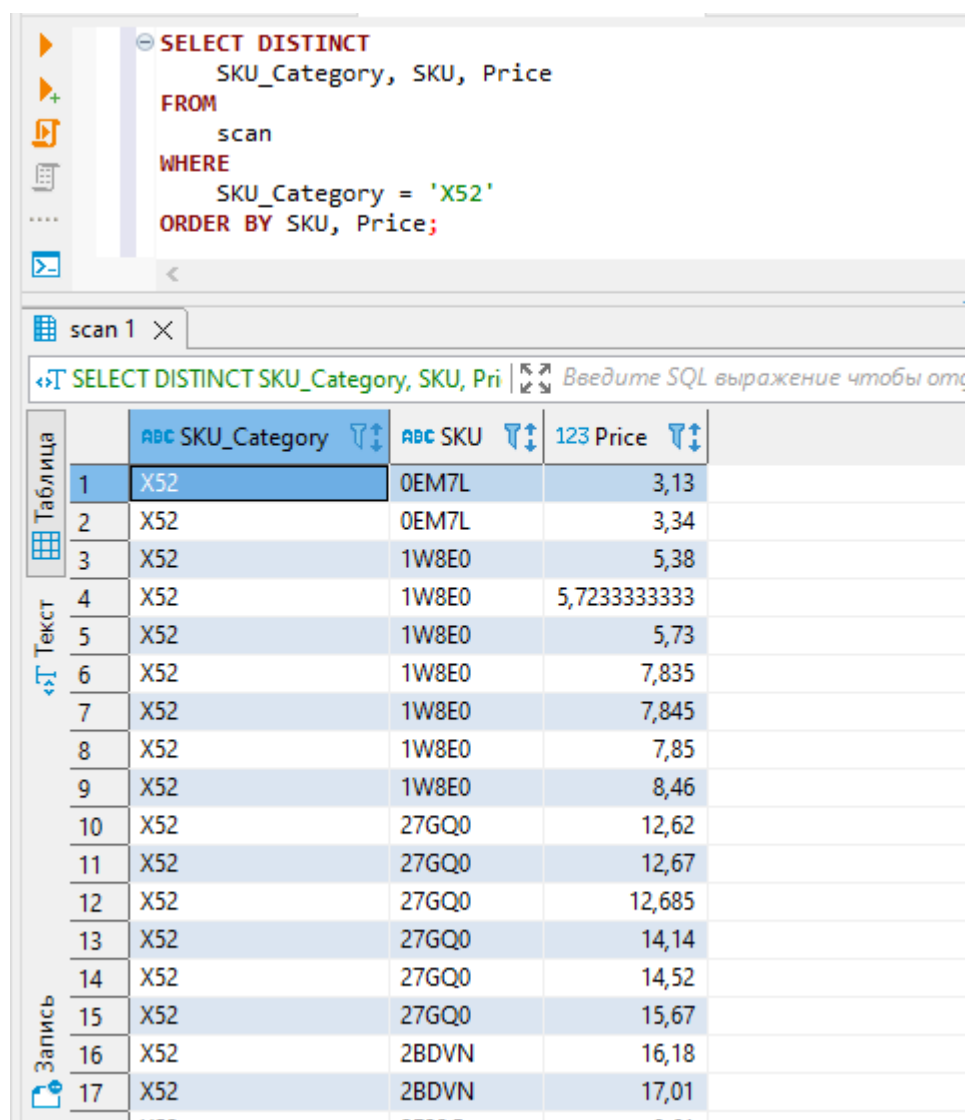
Добавим в вывод информацию о цене товара. Увидим, что количество записей увеличилось. Это означает, что в отдельных транзакциях цена одного и того же товара была различной. Чтобы проанализировать диапазон разброса цен на один и тот же товар, необходимо упорядочить записи по полю артикула товара SKU.

Для сортировки записей в результирующем наборе используется оператор ORDER BY с модификаторами ASC (сортировка в порядке возрастания, вариант по умолчанию – можно не указывать), DESC (сортировка по убыванию). Проведем сортировку результирующего набора вначале по артикулу товара, а потом по цене (рис. 2.22):

```

SELECT DISTINCT
  SKU_Category, SKU, Price
FROM
  scan
WHERE
  SKU_Category = 'X52'
ORDER BY SKU, Price;

```



```

SELECT DISTINCT
  SKU_Category, SKU, Price
FROM
  scan
WHERE
  SKU_Category = 'X52'
ORDER BY SKU, Price;

```

	ABC SKU_Category	ABC SKU	123 Price
1	X52	0EM7L	3,13
2	X52	0EM7L	3,34
3	X52	1W8E0	5,38
4	X52	1W8E0	5,7233333333
5	X52	1W8E0	5,73
6	X52	1W8E0	7,835
7	X52	1W8E0	7,845
8	X52	1W8E0	7,85
9	X52	1W8E0	8,46
10	X52	27GQ0	12,62
11	X52	27GQ0	12,67
12	X52	27GQ0	12,685
13	X52	27GQ0	14,14
14	X52	27GQ0	14,52
15	X52	27GQ0	15,67
16	X52	2BDVN	16,18
17	X52	2BDVN	17,01

Рис. 2.22. Результат выполнения запроса (код самолета, класс обслуживания упорядоченные и без дубликатов)

Значение цены единицы товара в четвертой строке выглядит сомнительно. Однако вспомним, что поле Price – это поле с вычислимыми значениями, которое было добавлено в таблицу как отношение суммы за товарную позицию в транзакции к количеству товара в этой позиции. И в продажах часто встречаются акции, когда четыре товара продаются по цене трех. Возможны и более сложные ценовые правила, например, учитывается персональная скидка конкретного покупателя и т.п.

Определим количество проданных товаров каждого артикула в этой категории, а также их общую стоимость. Оператор GROUP BY используется в операторе SELECT для сбора данных по нескольким записям и группировки результатов по одному или нескольким столбцам. В запрос необходимо включить агрегатную функцию (табл. 2.2), которая

подсчитывает количество проданных единиц товара каждого артикула, и общую сумму, вырученную за этот товар. Для столбцов, выводющих результат применения функций к значениям столбцов, необходимо указать псевдоним, иначе названием столбца станет имя самой функции.

Таблица 2.2

Агрегатные функции

Функция	Описание
avg()	находит среднее значение
sum()	находит сумму значений
min()	находит наименьшее значение
max()	находит наибольшее значение
count()	находит количество строк в запросе

В запросе используется для группировки поле артикула товара SKU и сразу два поля Quantity и Sales_Amount для вычисления значений агрегатных функций sum() (рис. 2.23).

The screenshot displays a database query editor with the following SQL query:

```
SELECT
    SKU_Category,
    SKU,
    sum(Quantity) AS Total_Quantity,
    sum(Sales_Amount) AS Total_Sales
FROM
    scan
GROUP BY
    SKU
ORDER BY
    SKU_Category,
    Total_Sales DESC;
```

Below the query editor, the results are shown in a table with 4 columns: SKU_Category, SKU, Total_Quantity, and Total_Sales. The table contains 12 rows of data, sorted by Total_Sales in descending order.

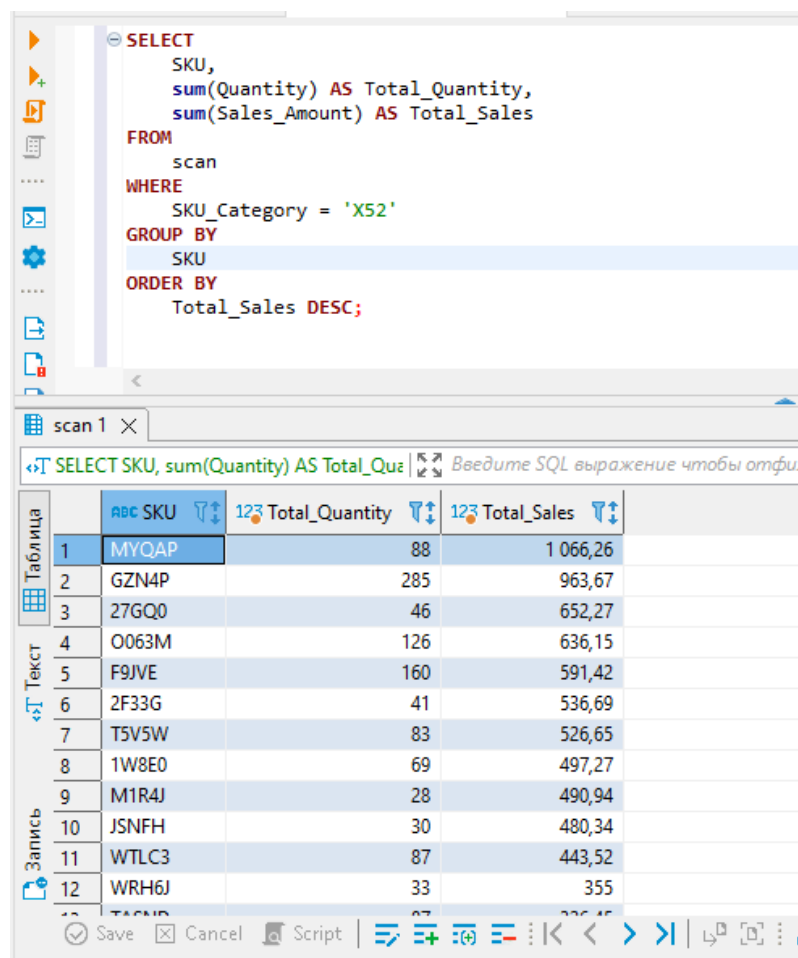
SKU_Category	SKU	Total_Quantity	Total_Sales
01F	OJS5N	194	5 026,52
01F	QMSAV	76	4 735,84
01F	MW1H8	91	2 656,64
01F	TIE11	153	2 527,38
01F	CM8EQ	53	2 502,72
01F	3OB6W	72	1 796,3
01F	9MLCO	29	1 790,86
01F	3GTLH	353	1 424,76
01F	1IX2L	299	1 374,48
01F	8XU9Y	388	1 359,03
01F	GGYTB	147	1 320,6
01F	P6RR0	167	1 316,45

Рис. 2.23. Результат выполнения запроса (общее количество проданных единиц и сумма продаж каждого артикула товара)

Также для упрощения анализа данных проводится сортировка по категории товара (чтобы товары одной категории выводились в одном фрагменте таблицы) и по убыванию общей суммы продаж товара конкретного артикула из категории:

```
SELECT
    SKU_Category,
    SKU,
    sum(Quantity) AS Total_Quantity,
    sum(Sales_Amount) AS Total_Sales
FROM
    scan
GROUP BY
    SKU_Category
ORDER BY
    Total_Sales DESC;
```

Введем дополнительное условие отбора строк: оставим только одну категорию SKU_Category 'X52' (рис. 2.24).



The screenshot shows a SQL query editor with the following query:

```
SELECT
    SKU,
    sum(Quantity) AS Total_Quantity,
    sum(Sales_Amount) AS Total_Sales
FROM
    scan
WHERE
    SKU_Category = 'X52'
GROUP BY
    SKU
ORDER BY
    Total_Sales DESC;
```

Below the query editor, the results are displayed in a table. The table has three columns: SKU, Total_Quantity, and Total_Sales. The results are sorted by Total_Sales in descending order.

SKU	Total_Quantity	Total_Sales
MYQAP	88	1 066,26
GZN4P	285	963,67
27GQ0	46	652,27
O063M	126	636,15
F9JVE	160	591,42
2F33G	41	536,69
T5V5W	83	526,65
1W8E0	69	497,27
M1R4J	28	490,94
JSNFH	30	480,34
WTLC3	87	443,52
WRH6J	33	355

Рис. 2.24. Результат выполнения запроса для категории товаров 'X52'

Так как теперь нет вариации записей по категории товара, то это поле исключим из запроса на вывод и из сортировки, оставив его только в условии отбора записей:

```
SELECT
    SKU,
    sum(Quantity) AS Total_Quantity,
    sum(Sales_Amount) AS Total_Sales
FROM
    scan
WHERE
    SKU_Category = 'X52'
GROUP BY
    SKU
ORDER BY
    Total_Sales DESC;
```

Добавим дополнительное условие по временному интервалу продаж и узнаем количество проданных товаров и суммы продаж различных артикулов товаров в этой категории за январь 2016 г. (рис. 2.25):

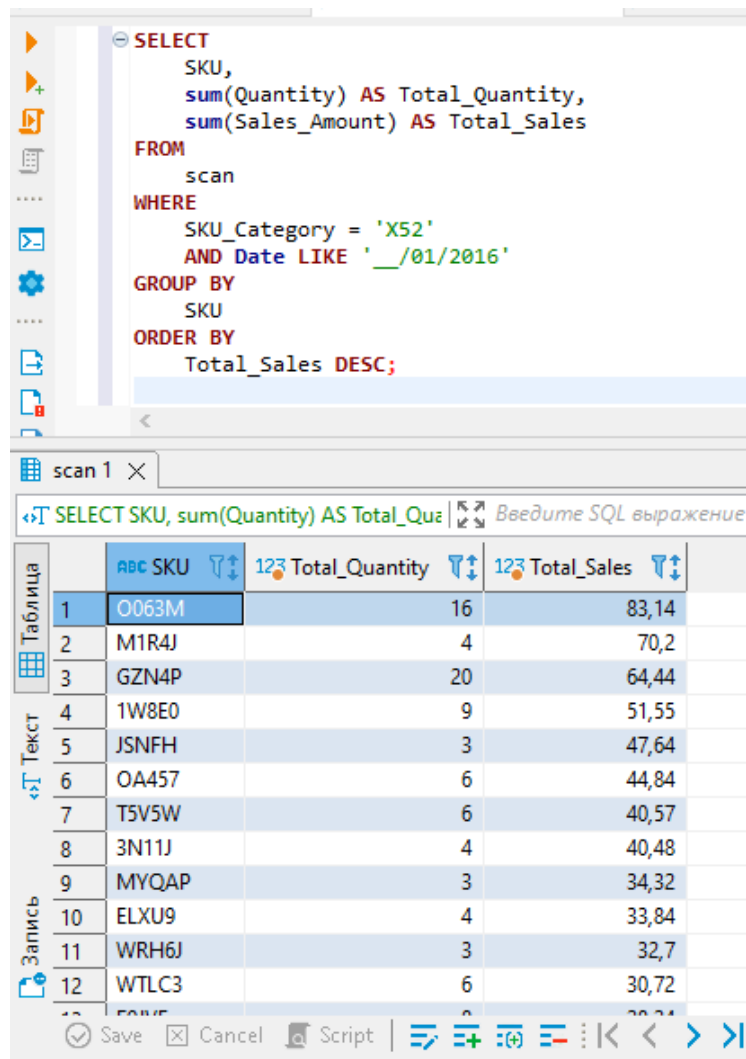
```
SELECT
    SKU,
    sum(Quantity) AS Total_Quantity,
    sum(Sales_Amount) AS Total_Sales
FROM
    scan
WHERE
    SKU_Category = 'X52'
    AND Date LIKE '__/01/2016'
GROUP BY
    SKU
ORDER BY
    Total_Sales DESC;
```

Оставим в исходном запросе (до уточнения временного интервала и категории товаров) в рассмотрении только те артикулы, для которых общая сумма продаж за весь период превосходит 2500. Для этого будем использовать оператор HAVING, который исключает из результата строки групп, не удовлетворяющих условию.

HAVING выполняет ту же роль, что и WHERE, но WHERE фильтрует отдельные строки до применения GROUP BY, а HAVING фильтрует строки групп, созданных предложением GROUP BY.

В запрос нужно добавить условие отбора по значению больше 2500 для столбца с агрегатной функцией sum(Sales_Amount) с псевдонимом Total_Sales. Однако обращаться по псевдониму Total_Sales к результату агрегатной функции нельзя, это связано с порядком выполнения операторов:

1. FROM
2. JOIN
3. WHERE
4. GROUP BY
5. HAVING
6. SELECT
7. DISTINCT
8. ORDER BY



```

SELECT
  SKU,
  sum(Quantity) AS Total_Quantity,
  sum(Sales_Amount) AS Total_Sales
FROM
  scan
WHERE
  SKU_Category = 'X52'
  AND Date LIKE '__/01/2016'
GROUP BY
  SKU
ORDER BY
  Total_Sales DESC;

```

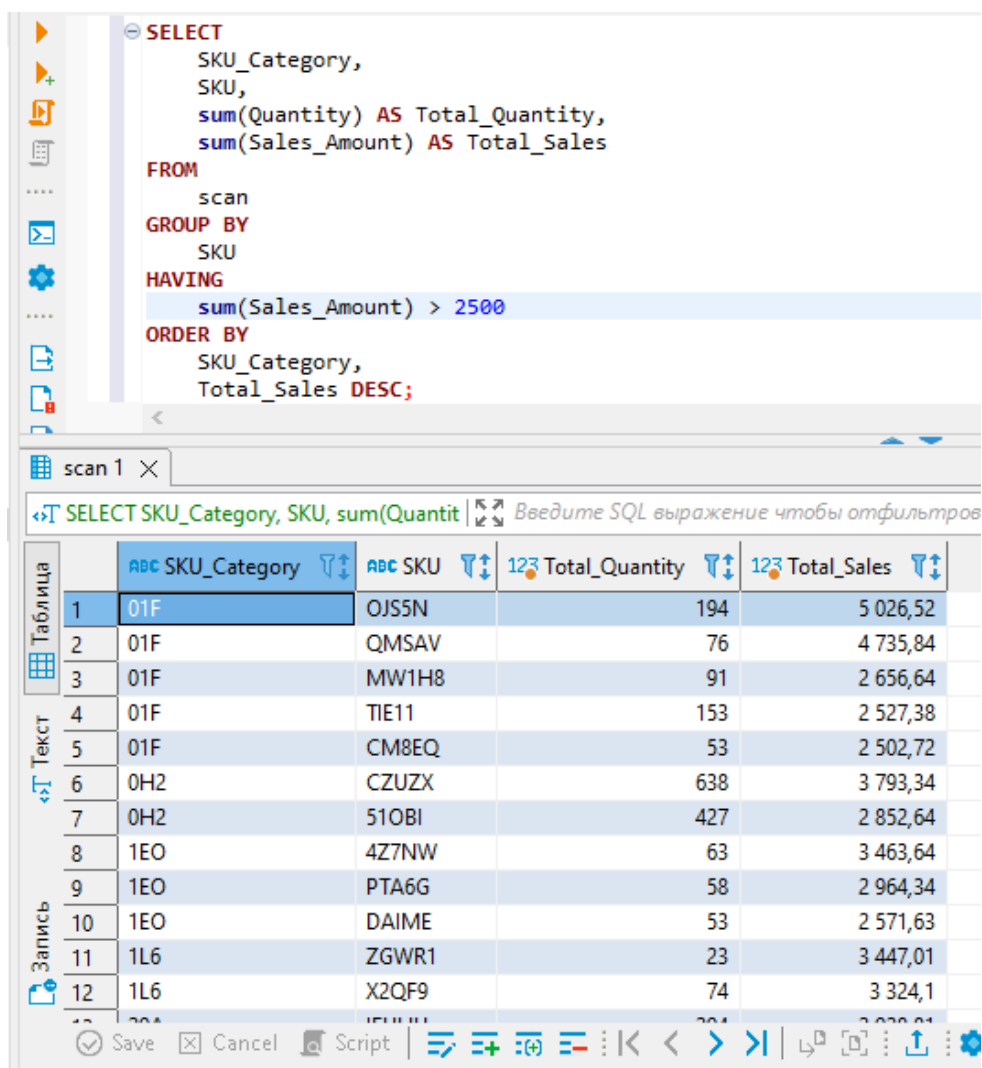
Таблица	SKU	Total_Quantity	Total_Sales
1	O063M	16	83,14
2	M1R4J	4	70,2
3	GZN4P	20	64,44
4	1W8E0	9	51,55
5	JSNFH	3	47,64
6	OA457	6	44,84
7	T5V5W	6	40,57
8	3N11J	4	40,48
9	MYQAP	3	34,32
10	ELXU9	4	33,84
11	WRH6J	3	32,7
12	WTLC3	6	30,72

Рис. 2.25. Результат выполнения запроса для категории товаров 'X52' в январе 2016 г.

К моменту выполнения оператора SELECT уже известны таблицы и представления, из которых требуется забрать столбцы, а также их алиасы (оператор FROM), а вот к моменту выполнения оператора HAVING о псевдониме для столбца с результатом агрегатной функции (оператор SELECT) еще неизвестно. То же и с ORDER BY, к моменту выполнения оператора столбцу с результатом агрегатной функции уже

присвоен псевдоним (оператор SELECT), поэтому обращаться к нему можно только по алиасу (рис. 2.26):

```
SELECT
    SKU_Category,
    SKU,
    sum(Quantity) AS Total_Quantity,
    sum(Sales_Amount) AS Total_Sales
FROM
    scan
GROUP BY
    SKU
HAVING
    sum(Sales_Amount) > 2500
ORDER BY
    SKU_Category,
    Total_Sales DESC;
```



The screenshot shows a SQL query editor with the following query:

```
SELECT
    SKU_Category,
    SKU,
    sum(Quantity) AS Total_Quantity,
    sum(Sales_Amount) AS Total_Sales
FROM
    scan
GROUP BY
    SKU
HAVING
    sum(Sales_Amount) > 2500
ORDER BY
    SKU_Category,
    Total_Sales DESC;
```

The results are displayed in a table with the following columns: SKU_Category, SKU, Total_Quantity, and Total_Sales. The results are sorted by Total_Sales in descending order.

SKU_Category	SKU	Total_Quantity	Total_Sales
01F	OJS5N	194	5 026,52
01F	QMSAV	76	4 735,84
01F	MW1H8	91	2 656,64
01F	TIE11	153	2 527,38
01F	CM8EQ	53	2 502,72
0H2	CZUZX	638	3 793,34
0H2	51OBI	427	2 852,64
1EO	4Z7NW	63	3 463,64
1EO	PTA6G	58	2 964,34
1EO	DAIME	53	2 571,63
1L6	ZGWR1	23	3 447,01
1L6	X2QF9	74	3 324,1

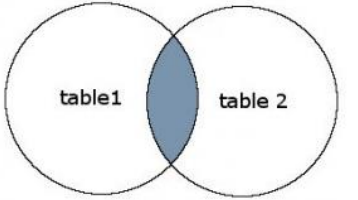
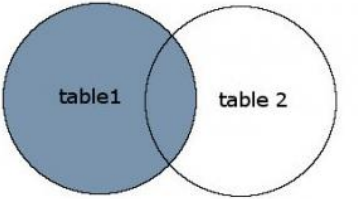
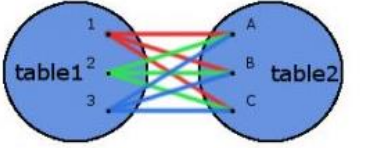
Рис. 2.26. Результат выполнения запроса для отбора артикулов товаров продажи которых были за весь период на сумму свыше 2500

2.3.3. Соединения

Если для получения результата требуется информация из нескольких таблиц, то в операторе FROM используют JOIN (соединение) таблиц (и любых иных объектов базы данных). Типы доступных в SQLite соединений представлены в табл. 2.3. Предложение JOIN дополняется [ON conditions] – условием соединения таблиц.

Таблица 2.3

Типы соединений

Тип соединения	Описание	Визуализация
INNER JOIN JOIN	Простое соединение, которое возвращает все строки декартова произведения, ограниченные условием соединения	 oraclepiql.ru
LEFT OUTER JOIN LEFT JOIN	Левостороннее соединение, которое возвращает все строки ограниченного декартова произведения: все объединенные строки, удовлетворяющие условию соединения и все строки в таблице слева, для которых не находится строк в таблице справа, удовлетворяющих условию (дополняются до полной ширины объединенной таблицы значениями NULL в столбцах таблицы справа)	 oraclepiql.ru
CROSS JOIN	Декартово произведение: каждая строка первой таблицы сопоставляется с каждой строкой второй таблицы	 oraclepiql.ru

Задаваемое условие соединения (ON conditions) представляет собой выражение, выдающее значение типа boolean (как в предложении WHERE), которое определяет, какие строки считаются соответствующими при соединении.

Синтаксис операторов соединения JOIN:

```
SELECT columns  
FROM table1  
[INNER] JOIN table2  
ON table1.column = table2.column;
```

```
SELECT columns  
FROM table1  
LEFT [OUTER] JOIN table2  
ON table1.column = table2.column;
```

```
SELECT columns  
FROM table1  
CROSS JOIN table2;
```

Откроем скрипт для базы данных retail_new.db. Найдем общую сумму покупок клиентов, которые откликнулись на рекламное взаимодействие. Транзакции оплаты покупок клиентов находятся в таблице transactions, а ответ клиентов на рекламное взаимодействие в таблице response, т.е. нужны сведения из двух таблиц. Соединение пройдет по ключевому полю customer_id, которое включено в каждую из двух соединяемых таблиц.

В этом случае становится очевидной польза от псевдонимов для таблиц, так как обе таблицы содержат столбец с одним и тем же именем, и нужна явная отсылка при указании имени столбца, к какой из таблиц он относится. А указывать перед именем столбца одну букву (псевдоним) или полное имя таблицы – это разные затраты времени на кодирование.

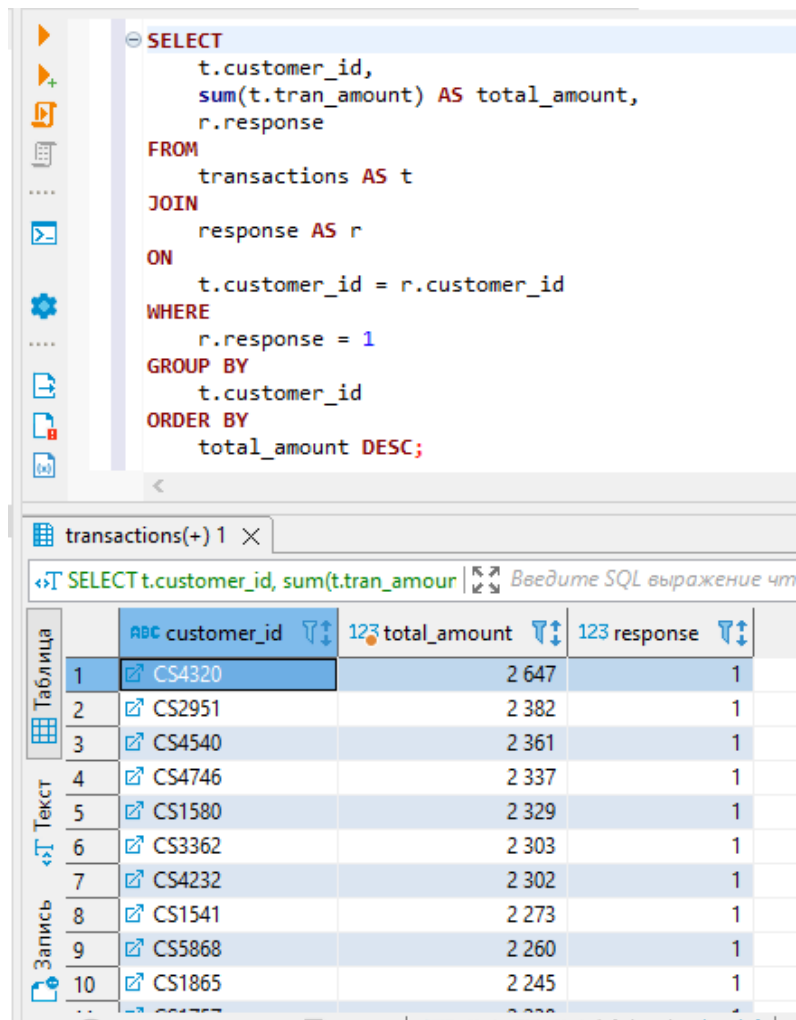
Нам нужно также вычислить агрегирующую функцию sum() для платежей, сгруппированных по клиентам customer_id, и отсортровать по вычисляемому полю по убыванию оплаченной суммы за весь период (рис. 2.27):

```
SELECT  
    t.customer_id,  
    sum(t.tran_amount) AS total_amount,  
    r.response  
FROM  
    transactions AS t  
JOIN  
    response AS r  
ON  
    t.customer_id = r.customer_id  
WHERE
```

```

        r.response = 1
GROUP BY
        t.customer_id
ORDER BY
        total_amount DESC;

```



The screenshot shows a SQL query editor with a query window and a results window. The query window contains the following SQL statement:

```

SELECT
    t.customer_id,
    sum(t.tran_amount) AS total_amount,
    r.response
FROM
    transactions AS t
JOIN
    response AS r
ON
    t.customer_id = r.customer_id
WHERE
    r.response = 1
GROUP BY
    t.customer_id
ORDER BY
    total_amount DESC;

```

The results window shows a table with the following data:

	customer_id	total_amount	response
1	CS4320	2 647	1
2	CS2951	2 382	1
3	CS4540	2 361	1
4	CS4746	2 337	1
5	CS1580	2 329	1
6	CS3362	2 303	1
7	CS4232	2 302	1
8	CS1541	2 273	1
9	CS5868	2 260	1
10	CS1865	2 245	1

Рис. 2.27. Результат выполнения запроса на вывод клиентов, которые откликнулись на рекламное взаимодействие, и потраченных ими сумм

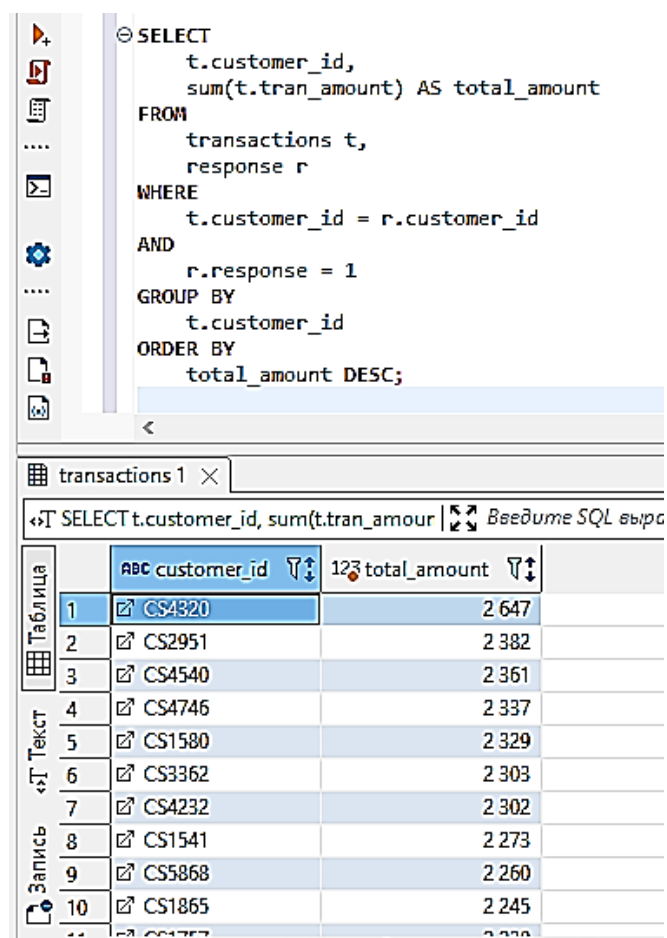
Отметим, что поле `response` лишнее в выдаче, так как для всех записей в запросе оно принимает одно и то же значение, и можно было обойтись без него.

Оператор простого соединения можно не использовать, перечислив обе таблицы в операторе `FROM` и добавив условие соединения таблиц в оператор `WHERE` (рис. 2.28). Как видим, результат аналогичен, также опущено необязательное ключевое слово `AS` при назначении псевдонимов таблицам (для столбцов по негласному соглашению его все же используют) и не запрашивается столбец `response`:

```

SELECT
    t.customer_id,
    sum(t.tran_amount) AS total_amount
FROM
    transactions t,
    response r
WHERE
    t.customer_id = r.customer_id
AND
    r.response = 1
GROUP BY
    t.customer_id
ORDER BY
    total_amount DESC;

```



The screenshot shows a database management tool interface. At the top, the SQL query is displayed in a text editor. Below the editor, a tab labeled 'transactions 1' is active. Underneath the tab, the SQL query is repeated. Below the query, a table titled 'transactions 1' is displayed, showing the results of the query. The table has two columns: 'customer_id' and 'total_amount'. The results are sorted by 'total_amount' in descending order.

	customer_id	total_amount
1	CS4320	2 647
2	CS2951	2 382
3	CS4540	2 361
4	CS4746	2 337
5	CS1580	2 329
6	CS3362	2 303
7	CS4232	2 302
8	CS1541	2 273
9	CS5868	2 260
10	CS1865	2 245

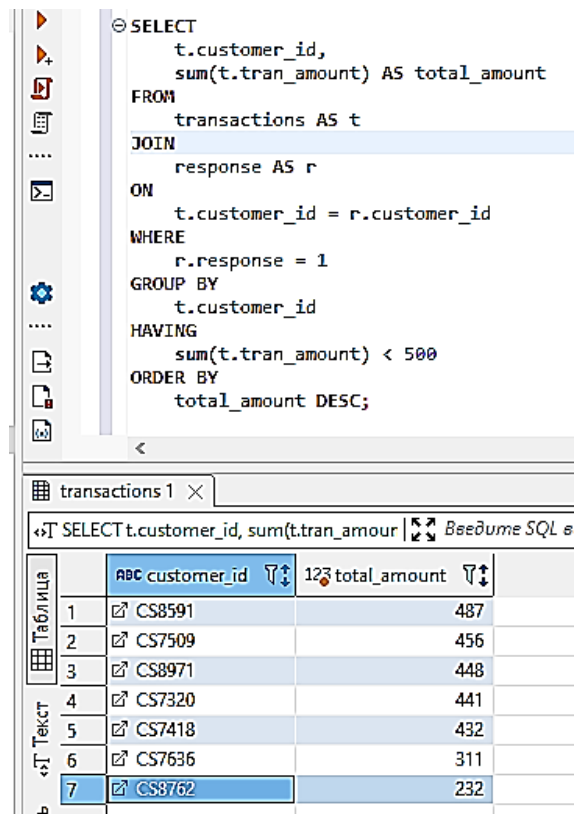
Рис. 2.28. Результат выполнения альтернативной версии запроса (клиенты, которые откликнулись на рекламное взаимодействие, и потраченные ими суммы)

Оставим в предыдущем запросе только те записи, в которых клиенты имеют общую сумму оплаты товаров меньше 500, т.е. используем оператор HAVING для отбора записей по условию (рис. 2.29):

```

SELECT
    t.customer_id,
    sum(t.tran_amount) AS total_amount
FROM
    transactions AS t
JOIN
    response AS r
ON
    t.customer_id = r.customer_id
WHERE
    r.response = 1
GROUP BY
    t.customer_id
HAVING
    sum(t.tran_amount) < 500
ORDER BY
    total_amount DESC;

```



The screenshot shows a SQL IDE interface. The top pane displays the SQL query from the previous block. The bottom pane shows the results of the query in a table format. The table has two columns: 'customer_id' and 'total_amount'. The results are ordered by 'total_amount' in descending order. The last row, with 'customer_id' 'CS8762' and 'total_amount' 232, is highlighted in blue.

	customer_id	total_amount
1	CS8591	487
2	CS7509	456
3	CS8971	448
4	CS7320	441
5	CS7418	432
6	CS7636	311
7	CS8762	232

Рис. 2.29. Результат выполнения запроса на вывод откликнувшихся на рекламное взаимодействие клиентов с потраченными суммами меньше 500

Удалим клиента с идентификатором 'CS8762' из таблицы response:

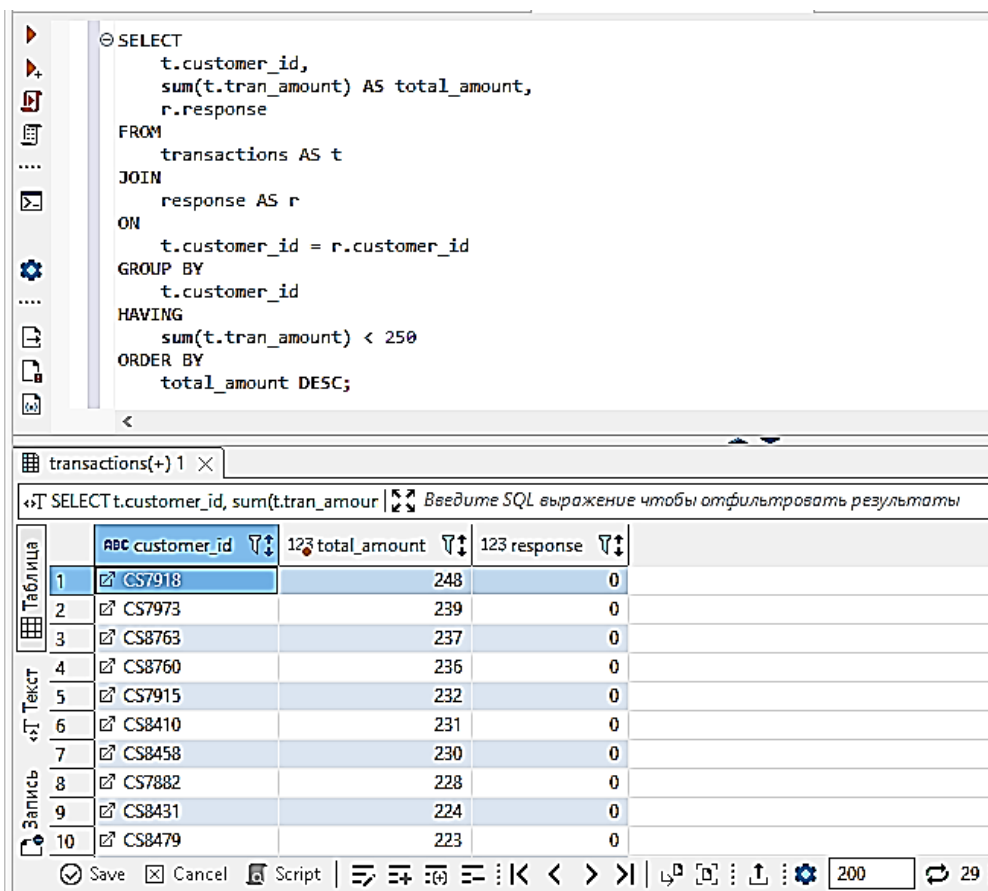
```

DELETE FROM response
WHERE
    customer_id = 'CS8762';

```

Исключим из запроса условие, что клиент откликнулся на рекламное взаимодействие и добавим поле response в вывод, а условие по общей сумме оплаты уменьшим до 250. Использование оператора соединения JOIN даст результат из 29 записей (рис. 2.30):

```
SELECT
    t.customer_id,
    sum(t.tran_amount) AS total_amount,
    r.response
FROM
    transactions AS t
JOIN
    response AS r
ON
    t.customer_id = r.customer_id
GROUP BY
    t.customer_id
HAVING
    sum(t.tran_amount) < 250
ORDER BY
    total_amount DESC;
```



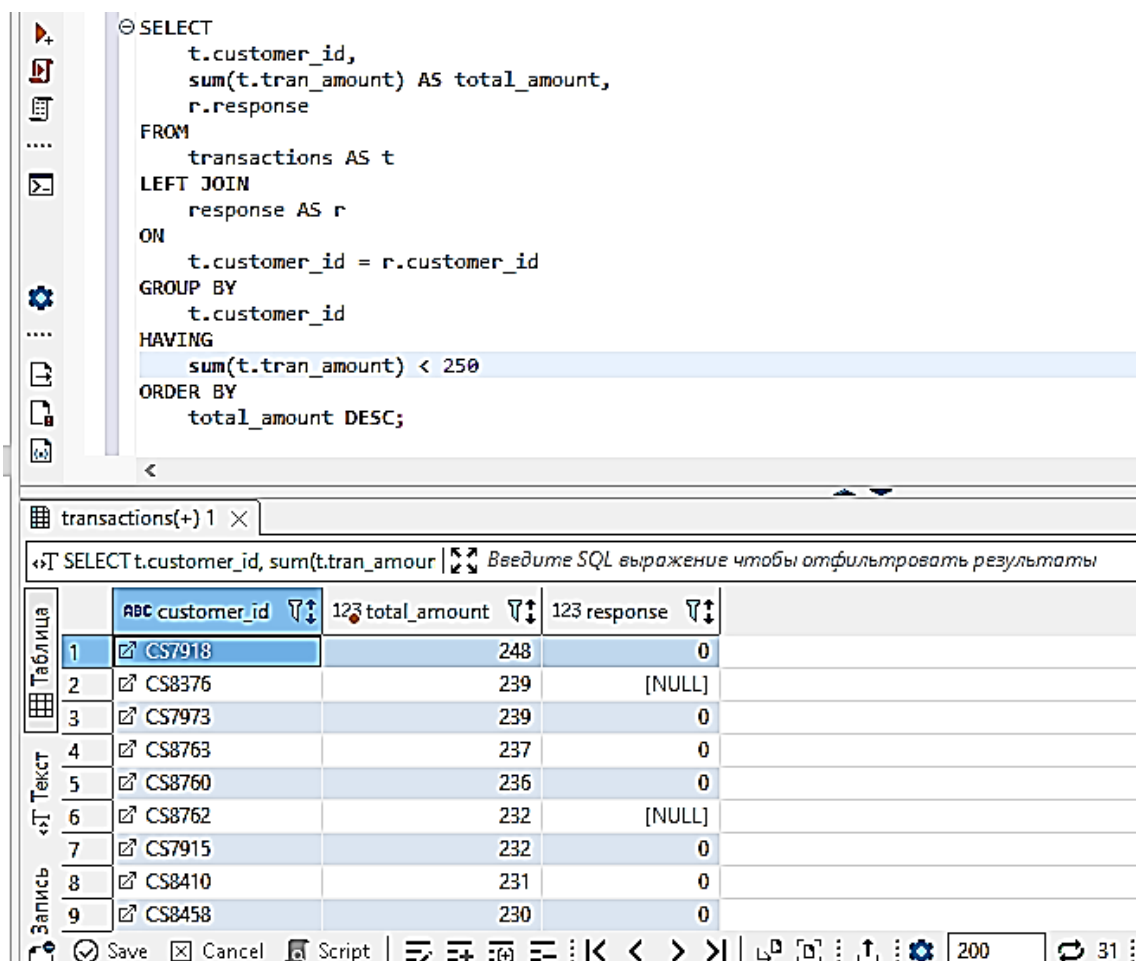
The screenshot shows a database query tool interface. The top pane displays the SQL query: `SELECT t.customer_id, sum(t.tran_amount) AS total_amount, r.response FROM transactions AS t JOIN response AS r ON t.customer_id = r.customer_id GROUP BY t.customer_id HAVING sum(t.tran_amount) < 250 ORDER BY total_amount DESC;`. The bottom pane shows the results of the query in a table view. The table has 10 rows and 3 columns: `customer_id`, `total_amount`, and `response`. The results are ordered by `total_amount` in descending order.

	customer_id	total_amount	response
1	CS7918	248	0
2	CS7973	239	0
3	CS8763	237	0
4	CS8760	236	0
5	CS7915	232	0
6	CS8410	231	0
7	CS8458	230	0
8	CS7882	228	0
9	CS8431	224	0
10	CS8479	223	0

Рис. 2.30. Все клиенты, которые имеют общую сумму меньше 250 и чьи идентификаторы есть в обеих таблицах

Использование соединения LEFT JOIN даст результат из 31 записи (рис. 2.31):

```
SELECT
    t.customer_id,
    sum(t.tran_amount) AS total_amount,
    r.response
FROM
    transactions AS t
LEFT JOIN
    response AS r
ON
    t.customer_id = r.customer_id
GROUP BY
    t.customer_id
HAVING
    sum(t.tran_amount) < 250
ORDER BY
    total_amount DESC;
```



The screenshot shows a SQL IDE interface. The top pane displays the following SQL query:

```
SELECT
    t.customer_id,
    sum(t.tran_amount) AS total_amount,
    r.response
FROM
    transactions AS t
LEFT JOIN
    response AS r
ON
    t.customer_id = r.customer_id
GROUP BY
    t.customer_id
HAVING
    sum(t.tran_amount) < 250
ORDER BY
    total_amount DESC;
```

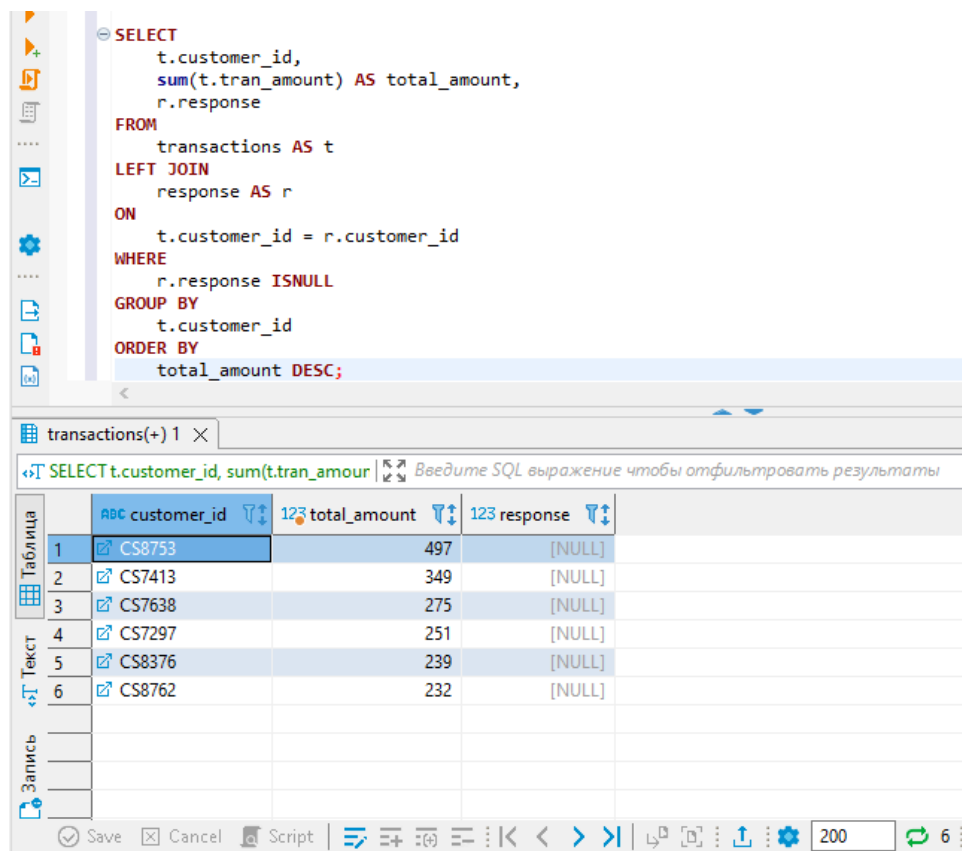
The bottom pane shows the results of the query in a table view. The table has 9 rows and 4 columns: customer_id, total_amount, response, and an empty column. The data is as follows:

	customer_id	total_amount	response	
1	CS7918	248	0	
2	CS8376	239	[NULL]	
3	CS7973	239	0	
4	CS8763	237	0	
5	CS8760	236	0	
6	CS8762	232	[NULL]	
7	CS7915	232	0	
8	CS8410	231	0	
9	CS8458	230	0	

Рис. 2.31. Все клиенты, которые имеют общую сумму меньше 250 и чьи идентификаторы есть в первой таблице, но, возможно, отсутствуют во второй

Один из клиентов с маленькой суммой оплаты был удален из таблицы response ранее. Но оказывается и изначально не о всех клиентах, которые совершали транзакции, известна информация об их отклике на рекламную компанию. Запросим информацию обо всех клиентах, про которых неизвестно об их отклике на рекламную компанию (рис. 2.32):

```
SELECT
    t.customer_id,
    sum(t.tran_amount) AS total_amount,
    r.response
FROM
    transactions AS t
LEFT JOIN
    response AS r
ON
    t.customer_id = r.customer_id
WHERE
    r.response ISNULL
GROUP BY
    t.customer_id
ORDER BY
    total_amount DESC;
```



The screenshot shows a database query editor with the following SQL query:

```
SELECT
    t.customer_id,
    sum(t.tran_amount) AS total_amount,
    r.response
FROM
    transactions AS t
LEFT JOIN
    response AS r
ON
    t.customer_id = r.customer_id
WHERE
    r.response ISNULL
GROUP BY
    t.customer_id
ORDER BY
    total_amount DESC;
```

The results are displayed in a table with the following columns: customer_id, total_amount, and response. The table contains 6 rows of data.

	customer_id	total_amount	response
1	CS8753	497	[NULL]
2	CS7413	349	[NULL]
3	CS7638	275	[NULL]
4	CS7297	251	[NULL]
5	CS8376	239	[NULL]
6	CS8762	232	[NULL]

Рис. 2.32. Все клиенты, чьи идентификаторы есть в первой таблице, но отсутствуют во второй

3. Лабораторные работы

3.1. Лабораторная работа «Настройка программного обеспечения»

Цель работы: научиться устанавливать программное обеспечение и настраивать его работу, научиться импортировать файлы данных с расширением .CSV, научиться работать с SQLite в Python

Формируемые знания, умения и навыки: знать как происходит установка СУБД SQLite, уметь запускать интерфейс командной строки. менеджер баз данных DBeaver и уметь работать с SQLite в Python.

Необходимо:

1. Используя материал настоящего пособия и вспомогательные ресурсы, произвести установку на домашний компьютер необходимого программного обеспечения.
2. Создать несколько баз данных SQLite на основе файлов .csv с помощью интерфейса командной строки, с помощью библиотеки sqlite3 на Python.

Контрольные вопросы и задания

1. Как выполняется подключение к базе данных?
2. Как импортировать файлы .csv в таблицы базы данных?
3. Как импортировать файл с данными с расширением .csv с помощью библиотеки sqlite3 на Python?
4. Как настроить подключения к базам данных SQLite в менеджере баз данных DBeaver?

3.2. Лабораторная работа «Операции DDL»

Цель работы: научиться работать с объектами базы данных, научиться создавать новые объекты, модифицировать и удалять существующие.

Формируемые знания, умения и навыки: знать основные типы данных, синтаксис операций языка DDL, уметь составлять запросы с использованием конструкций языка DDL.

Необходимо:

1. Создать базу данных, создать таблицу в этой базе данных, переименовать таблицу, переименовать столбец.
2. Создать новый столбец и занести в него данные.
3. Удалить таблицу.

Контрольные вопросы и задания

1. Какие типы данных используются в SQLite?
2. Как создать базу данных (в DBeaver, в командной строке, в Notebook Colab на языке Python)?
3. Как создать таблицу (в DBeaver, в командной строке, в Notebook Colab на языке Python)?
4. Как переименовать таблицу?
5. Как переименовать столбец?
6. Как удалить таблицу?

3.3. Лабораторная работа «Операции DML»

Цель работы: научиться переходить от бизнес-задач, т.е. вопросов на естественном языке, включающих запрос необходимых бизнес-данных, к запросам на языке SQL.

Формируемые знания, умения и навыки: знать синтаксис операций языка DML, уметь составлять запросы с использованием конструкций языка DML.

Необходимо:

1. Создать собственную базу данных с таблицей на основе файла .csv (скачать данные с ресурса kaggle). Привести описание данных: название столбцов, контекст хранимой информации, типы данных.
2. Сформулировать несколько бизнес-задач, перевести их на язык запросов и выполнить.
3. Привести примеры запросов с использованием агрегатных функций.
4. Создать новую базу данных, в которой создать две таблицы, добавить в них данные согласно рис. 3.1. Выполнить различные варианты соединений и пояснить полученный результат (особенно важно уделить внимание проверке условий, если в сравниваемых столбцах имеются значения NULL).

Staff table

LastName	DepartmentID
Иванов	31
Сидорова	33
Смирнов	33
Матвеев	34
Максимова	34
Яковлев	NULL

Department table

DepartmentID	DepartmentName
31	Продажи
33	Делопроизводство
34	Закупки
35	Маркетинг
31	Бухгалтерия

Рис. 3.1. Таблицы – образец заполнения данными

Контрольные вопросы и задания

1. Синтаксис операции SELECT.
2. Какие операторы сравнения можно использовать в условиях?
3. Объясните разницу между использованием операторов WHERE и HAVING?
4. Приведите пример использования агрегатной функции в запросе.
5. Как исключить дубли из запроса?
6. Какие типы соединений вы знаете? Чем они отличаются?
7. Какие особенности имеет сравнение столбцов, в которых есть значения NULL?

Список литературы

1. Крейбич Д. А. Использование SQLite – UML. URL: <https://akawah.ru/linux/sqlite.html?ysclid=letpeom7gc284022621> (дата обращения: 20.02.2023).
2. SQLite : учебник. URL: <https://oracleplsql.ru/sqlite-manual.html> (дата обращения: 20.02.2023).
3. SQLite Forum. URL: <https://sqlite.org/forum/forummain> (дата обращения: 20.02.2023).
4. Рындина С. В. Цифровые технологии управления получением, хранением, передачей и обработкой больших данных: SQL (DDL и DML) на примере PostgreSQL : учеб.-метод. пособие. Пенза : Изд-во ПГУ, 2023. 84 с.

Учебное издание

Рындина Светлана Валентиновна

Цифровые технологии управления получением, хранением,
передачей и обработкой больших данных: SQLite

Редактор *А. Г. Темникова*
Технический редактор *М. Б. Жучкова*
Компьютерная верстка *М. Б. Жучковой*

Подписано в печать 13.10.2023.
Формат 60×84¹/₁₆. Усл. печ. л. 4,42.
Тираж 5. Заказ № 512.

Издательство ПГУ.
440026, г. Пенза, ул. Красная, 40.
Тел.: (8412) 66-60-49, 66-67-77; e-mail: iic@pnzgu.ru

