**Лабораторная работа 6. Система управления данными SQLite**

**Часть 1. Создание, наполнение баз данных в SQLite**

Цель: освоить азы инженерии данных, т.е. научиться создавать и наполнять базы данных с помощью SQL/DDL для СУБД SQLite.

Задачи:

* познакомиться с СУБД SQLite;
* освоить инструменты для работы с БД SQLite: консольная программа sqlite3 и графические оболочки DB Browser for SQLite или SQLiteStudio;
* изучить типы данных SQLite, научиться описывать и создавать таблицы на SQL;
* уметь определять ограничения целостности данных и ссылок на SQL;
* освоить инструкции SQL для добавления записей в таблицу;
* научиться формировать сценарий SQL для создания и наполнения БД.

Задание для выполнения части 1 лабораторной работы:

1. С помощью инструкций SQL **CREATE TABLE** в выбранном инструменте создайте новую БД, опишите 3 связанные таблицы, созданные в рамках выполнения ЛР1-5, используя подходящие типы SQLite. Установите нужные ограничения ссылочной целостности и некоторые ограничения целостности данных, определите, какие поля допускают **NULL**.
2. Наполните созданные таблицы тестовыми данными с помощью инструкции **INSERT**.
3. Создайте файл сценария с расширением .sql, который должен содержать все необходимые инструкции **CREATE** и **INSERT** для создания и наполнения тестовой БД. Это можно сделать средствами экспорта из Browser for SQLite.
4. Создайте новую БД, используя файл сценария в режиме импорта в Browser for SQLite.
5. Оформите ход выполнения и результаты работы в виде отчёта.

***Общая характеристика и назначение СУБД SQLite***

SQLite представляет библиотеку, написанную на языке C (ANSI-C) и реализующую движок реляционных баз данных. Сегодня SQLite – широко распространённая СУБД, которую можно найти в каждом устройстве на Android, iOS, Mac, Windows 10/11, ее используются большинство распространенных браузеров - Firefox, Chrome, Safari и т.д.

SQLite в отличие от других СУБД (MS SQL Server, MySQL, Postgres) представляет встраиваемый движок БД, для которого не требуется сервер базы данных, он обращается напрямую к файлу БДх на диске. Поэтому для работы с БД нам не надо явно устанавливать или как-то настраивать SQLite.

SQLite имеет полноценную поддержку большинства возможностей, которыми обладают другие реляционные СУБД.

SQLite для работы с БД применяет язык SQL (), реализация которого в целом похожа на диалекты в других реляционных СУБД.

Формат файла базы данных является кроссплатформенным – переносимым между устройствами в разных операционных системах.

Для разработки приложений большинство языков программирования (Python, C#, Java, и т.д.) имеют поддержку SQLite, что позволяет использовать эту СУБД в различных сценариях и самых различных типах приложений.

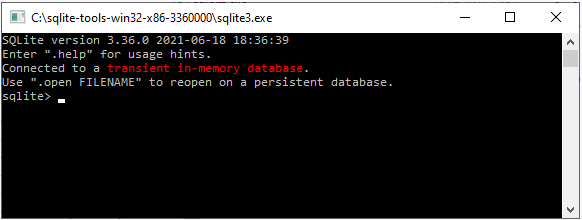
SQLite развивается как проект с открытым исходным кодом.

***Консольный клиент sqlite3***

Для работы с SQLite разработчики данной СУБД предоставляют консольный клиент sqlite3.

Прежде всего нам надо загрузить sqlite3 со страницы <https://www.sqlite.org/download.html>. Название необходимого нам пакета начинается с sqlite-tools. И на странице загрузки мы можем найти версии для Windows, Linux, MacOS.

В загруженной и распакованной папке мы сможем найти файл с названием sqlite3**,** который и представляет собственно консольную оболочку для работы с БД SQLite. Запустим ее:

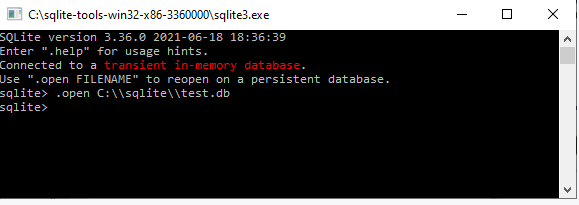


Открытие базы данных

Для открытия базы данных необходимо ввести команду **.open**, после которой указывается путь к базе данных. Например,

|  |
| --- |
| sqlite>.open test.db |

В данном случае будет открыта база данных под названием "test.db', которая находится в той же папке, что и консольная утилита. Если базы данных не существует, то она будет создаться.



Также можно передать абсолютный путь::

|  |
| --- |
| sqlite>.open C:\\sqlite\\test.db |

После открытия мы сможем работать с этой БД.

Создание таблицы

Для создания таблицы после открытия базы данных необходимо ввести команды **CREATE TABLE**, после которой указываются название таблицы и спецификация ее столбцов:

|  |
| --- |
| sqlite>create table users(name text, age integer); |

В данном случае создается таблица users, в которой два столбца: столбец name, который имеет тип text, и столбец age, который имеет тип integer. Обратите внимание, что команда завершается точкой с запятой, благодаря чему SQLite может идентифицировать, что выполняется SQL-команда.

Операции с данными

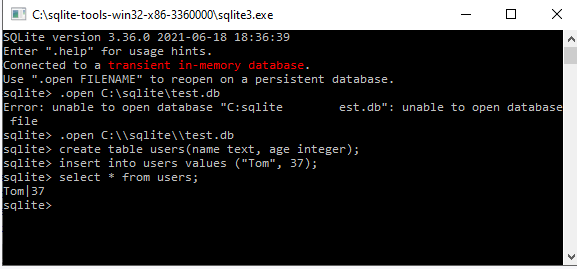
Для добавления данных применяется команда **INSERT INTO**. Например, добавим в таблицу users одну строку:

|  |
| --- |
| sqlite>insert into users values ('Tom', 37); |

Теперь получим ранее добавленные данные. Для этого используем команду **SELECT**:

|  |
| --- |
| sqlite>select \* from users |

И SQLite выведет нам все данные из таблицы users:



Для получения справки о командах введите:

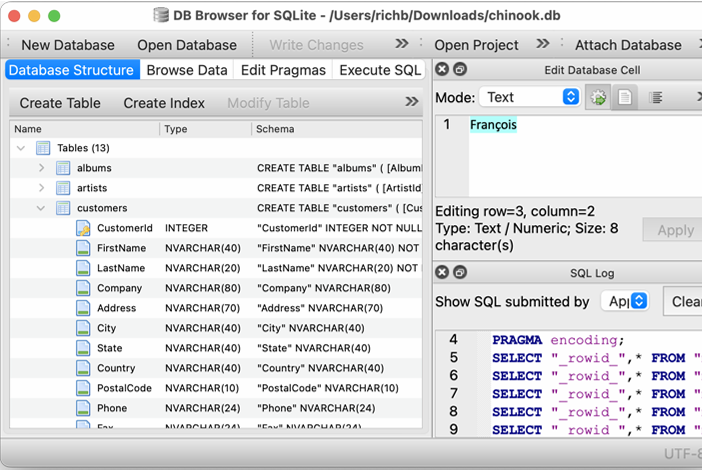
|  |
| --- |
| sqlite>.help |

Для завершения работы с утилитой введите следующую команду:

|  |
| --- |
| sqlite>.quit |

**Браузер БД для SQLite**

DB Browser for SQLite (DB4S) – это высококачественный, визуальный инструмент с открытым исходным кодом предназначен для тех, кто хочет создавать, искать и редактировать файлы баз данных SQLite. DB4S предоставляет знакомый интерфейс, похожий на электронные таблицы, для работы с базой данных в дополнение к предоставлению полного функционала SQL-запросов. Он работает с и Windows, macOS, и большинство версий Linux и Unix. Документация по программе находится на wiki.



DB4S предоставляет доступ ко всем возможностям базовой базы данных SQLite.

Элементы управления и мастера доступны для:

* Создание и сжатие файлов базы данных
* Создание, определение, изменение и удаление таблиц
* Создание, определение и удаление индексов
* Просмотр, редактирование, добавление и удаление записей
* Сортировка и поиск записей Импорт и экспорт записей в виде текста
* Импорт и экспорт таблиц из/в файлы CSV
* Импорт и экспорт баз данных из/в файлы дампа SQL
* Выполнение SQL-запросов и проверка результатов
* Просмотр журнала всех SQL-команд, выдаваемых приложением
* Построение простых графиков на основе данных таблиц или запросов.

Текущей официальной, выпущенной версией DB4S является 3.13.0. Скачать с: <https://sqlitebrowser.org/dl>

**Определение структуры данных в SQLite**

***Создание таблицы***

Для создания таблиц используется инструкция **CREATE TABLE** со следующим формальный синтаксисом:

|  |
| --- |
| **CREATE TABLE** *название\_таблицы*  (*название\_столбца1 тип\_данных свойства\_столбца1*,  *название\_столбца2 тип\_данных свойства\_столбца2*,   ................................................  *название\_столбцаN тип\_данных свойства\_столбцаN*,  *свойства\_уровня\_таблицы*  ) |

После инструкции CREATE TABLE указывается название таблицы. Имя таблицы выполняет роль её идентификатора в БД, поэтому оно должно быть уникальным. Кроме того, оно не должно начинаться на "sqlite\_", поскольку названия таких таблиц зарезервированы для внутреннего пользования.

После названия таблицы в скобках перечисляются названия столбцов, их типы данных и другие свойства. В самом конце можно определить свойства для всей таблицы. Свойства столбцов и таблицы указывать необязательно.

Прежде выполнения инструкции CREATE TABLE вне зависимости, что мы используем - консольный клиент sqlite3, графический клиент DB Browser for SQLite иликакой-то другой клиент, *вначале создадим новую БД или откроем имеющуюся*, где мы хотим создать таблицу.

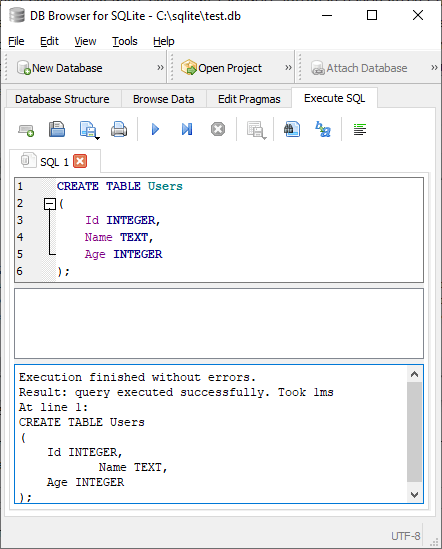


Рис. 1. Создание таблицы

Для создания таблицы выполним следующий скрипт:

|  |
| --- |
| **CREATE TABLE** Users  (Id INTEGER, Name TEXT, Age INTEGER); |

В данном случае таблица называется "Users". В ней определено три столбца: Id, Age, Name. Первые два столбца представляют идентификатор пользователя и его возраст и имеют тип INTEGER, то есть будут хранить числовые значения. Столбец "Name" представляет имя пользователя и имеет тип TEXT, то есть представляет строку. В данном случае для каждого столбца определены имя и тип данных, при этом атрибуты столбцов и таблицы в целом отсутствуют.

И в результате выполнения этой команды будет создана таблица Users с тремя столбцами (рис.1).

***Создание таблицы при ее отсутствии***

Если мы повторно выполним выше определенную sql-команду для создания таблицы Users, то мы столкнемся с ошибкой - ведь мы уже создали таблицу с таким названием. Но могут быть ситуации, когда мы можем точно не знать или быть не уверены, есть ли в базе данных такая таблица (например, когда мы пишем приложение на каком-нибудь языке программирования и используем базу данных, которая не нами создана). И чтобы избежать ошибки, с помощью выражения **IF NOT EXISTS** мы можем задать создание таблицы, если она не существует:

**CREATE TABLE IF NOT EXISTS** Users

(Id INTEGER, Name TEXT, Age INTEGER);

При определении столбцов таблицы для них необходимо указать тип данных. Каждый столбец имеет определенный тип данных. Для хранения данных в в SQLite применяются следующие типы:

* **INTEGER**: представляет целое число, которое может быть положительным и отрицательным и в зависимости от своего значения может занимать 1, 2, 3, 4, 6 или 8 байт
* **REAL**: представляет число с плавающей точкой, занимает 8 байт в памяти
* **TEXT**: строка текста в одинарных кавычках, которая сохраняется в кодировке базы данных (UTF-8, UTF-16BE или UTF-16LE)
* **BLOB**: бинарные данные

Стоит отметить, что SQLite оперирует концепцией **классов хранения** или **storage class**. Концепция классов хранения несколько шире, чем тип данных. Например, класс **INTEGER** объединяет 6 различных целочисленных типов данных разной длины. Однако это больше относится к внутренней работе SQLite.

***Ограничения столбцов и таблиц***

При определении столбцов и таблиц для них можно задать ограничения. Ограничения позволяют   
настроить поведение столбцов и таблиц. Ограничения столбцов указываются после типа столбца:

*имя\_столбца* *тип\_столбца* ***ограничения\_столбца***

Ограничения таблицы указываются после определения всех столбцов.

Рассмотрим, какие ограничения столбцов мы можем использовать.

Свойство **PRIMARY KEY** задает первичный ключ таблицы, который уникально идентифицирует строку в таблице. Например:

**CREATE TABLE** Users

(   id **INTEGER PRIMARY KEY**,

    name **TEXT**,

    age **INTEGER**

);

Здесь столбец id выступает в качестве первичного ключа, он будет уникально идентифицировать строку и его значение должно быть уникальным. То есть у нас не может быть таблице users более одной строки, где в столбце id было бы одно и то же значение.

Установка первичного ключа на уровне таблицы:

**CREATE TABL**E Users

(   id INTEGER,

    name TEXT,

    age INTEGER,

**PRIMARY KEY**(id)

);

Первичный ключ может быть составным. Такой ключ использовать сразу несколько столбцов, чтобы уникально идентифицировать строку в таблице. Например:

**CREATE TABLE** Users

(  id INTEGER,

    name TEXT,

    age INTEGER,

**PRIMARY KEY**(id, name)

);

В данном случае в качестве первичного ключа выступает связка столбцов id и name. То есть в таблице Users не может быть двух строк, где для обоих из этих полей одновременно были бы одни и те же значения.

Ограничение **AUTOINCREMENT** позволяет указать, что значение столбца будет автоматически увеличиваться при добавлении новой строки. Данное ограничение работает для столбцов, которые представляют тип **INTEGER** с ограничением **PRIMARY KEY**:

**CREATE TABLE** users

(   id INTEGER **PRIMARY KEY AUTOINCREMENT**,

    name TEXT,

    age INTEGER

);

В данном случае значение столбца id каждой новой добавленной строки будет увеличиваться на единицу.

Ограничение **UNIQUE** указывает, что столбец может хранить только уникальные значения.

**CREATE TABLE** Users

(   id INTEGER PRIMARY KEY AUTOINCREMENT,

    name TEXT,

    age INTEGER,

    email TEXT **UNIQUE**

);

В данном случае столбец email, который представляет телефон пользователя, может хранить только уникальные значения. И мы не сможем добавить в таблицу две строки, у которых значения для этого столбца будет совпадать.

Также мы можем определить это ограничение на уровне таблицы:

**CREATE TABLE** Users

(    id INTEGER PRIMARY KEY AUTOINCREMENT,

    name TEXT,

    age INTEGER,

    email TEXT,

**UNIQUE** (name, email)

);

В данном случае уникальность значений установлена сразу для двух столбцов - name и email.

По умолчанию любой столбец, если он не представляет первичный ключ, может принимать значение **NULL**, то есть фактически отсутствие формального значения. Но если мы хотим запретить подобное поведение и установить, что столбец обязательно должен иметь какое-либо значение, то для него следует установить ограничение **NOT NULL**:

**CREATE TABLE** Users

(   id INTEGER PRIMARY KEY,

    name TEXT **NOT NULL**,

    age INTEGER

);

В данном случае столбец name не допускает значение NULL.

Ограничение **DEFAULT** определяет значение по умолчанию для столбца. Если при добавлении данных для столбца не будет предусмотрено значение, то для него будет использоваться значение по умолчанию.

**CREATE TABLE** Users

(  id INTEGER PRIMARY KEY,

    name TEXT,

    age INTEGER **DEFAULT** 18

);

Здесь столбец age в качестве значения по умолчанию имеет число 18.

Ограничение **CHECK** задает ограничение для диапазона значений, которые могут храниться в столбце. Для этого после CHECK указывается в скобках условие, которому должен соответствовать столбец или несколько столбцов. Например, возраст пользователей не может быть меньше 0 или больше 100:

**CREATE TABLE** Users

(   id INTEGER PRIMARY KEY,

    name TEXT NOT NULL **CHECK**(name !=''),

    age INTEGER NOT NULL **CHECK**(age >0 AND age < 100)

);

Кроме проверки возраста здесь также проверяется, что столбец name не может иметь пустую строку в качестве значения (пустая строка не эквивалентна значению **NULL**).

Для соединения условий используется ключевое слово **AND**. Условия можно задать в виде операций сравнения больше (>), меньше (<), не равно (!=).

Также **CHECK** можно использовать на уровне таблицы:

**CREATE TABLE** Users

( id INTEGER PRIMARY KEY,

    name TEXT NOT NULL,

    age INTEGER NOT NU

**CHECK** ((age >0 AND age < 100) **AND** (name !=''))

);

С помощью оператора **CONSTRAINT** можно задать имена ограничений. Они указываются после ключевого слова **CONSTRAINT** перед ограничениями уровня таблицы:

**CREATE TABLE** Users

(  id INTEGER,

    name TEXT NOT NULL,

    email TEXT NOT NULL,

    age INTEGER NOT NULL,

**CONSTRAINT** users\_pk PRIMARY KEY(id),

**CONSTRAINT** user\_email\_uq UNIQUE(email),

**CONSTRAINT** user\_age\_chk CHECK(age >0 AND age < 100)

);

В данном случае ограничение для PRIMARY KEY называется users\_pk, для UNIQUE - user\_phone\_uq, а для CHECK - user\_age\_chk. Смысл установки имен ограничений заключается в том, что впоследствии через эти имена мы сможем управлять ограничениями - удалять или изменять их.

**Основные операции с данными**

***Добавление данных с помощью инструкции INSERT***

Для добавления данных в SQLite применяется команда **INSERT**, которая имеет следующее формальное определение:

**INSERT INTO** имя\_таблицы [(столбец1, стобец2, ... стобецN)]

**VALUES** (значение1, значение2, ... значениеN)

После выражения **INSERT INTO** в скобках можно указать список столбцов через запятую, в которые надо добавлять данные, и в конце после слова **VALUES** скобках перечисляют добавляемые для столбцов значения.

Например, пусть в базе данных SQLite есть следующая таблица users:

**CREATE TABLE** users

(   id INTEGER PRIMARY KEY AUTOINCREMENT,

    name TEXT,

    age INTEGER

);

Добавим в эту таблицу одну строку с помощью следующего кода:

INSERT INTO users (name, age) VALUES ('Tom', 37);

После названия таблицы указаны два столбца, в которые мы хотим выполнить добавить данные (name, age). После оператора **VALUES** указаны значения для этих столбцов. Значения будут передаваться столбцам по позиции. То есть столбцу name передается строка "Tom', столбцу age - число 37. И после успешного выполнения команды в таблице появится новая строка (рис. 2):

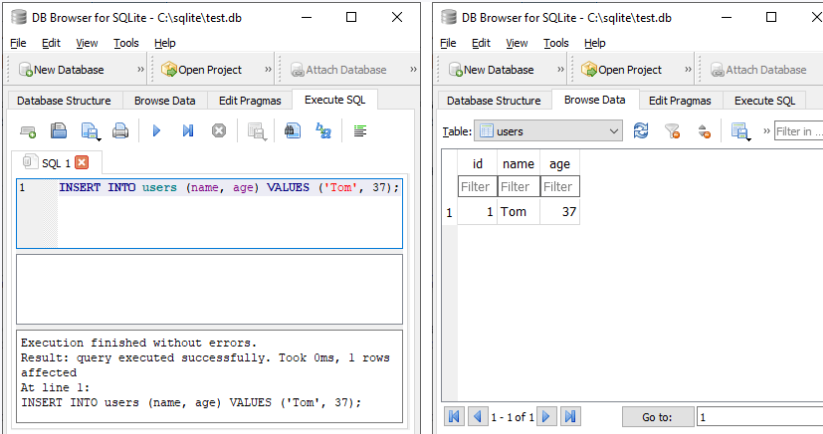


Рис. 2. Добавление строки данных

Стоит отметить, что при добавлении данных необязательно указывать значения абсолютно для всех столбцов таблицы. Например, в примере выше не указано значение для стобца id, поскольку для данного столбца значение будет автоматически генерироваться.

Также можно было бы не указывать названия столбцов:

**INSERT INTO** users **VALUES** (2, 'Bob', 41);

Однако в этом случае потребовалось бы указать значения для всех его столбцов, в том числе для столбца id. Причем значения передавались столбцам в том порядке, в котором они идут в таблице.

***Добавление нескольких строк***

Также мы можем добавить сразу несколько строк:

**INSERT INTO** users(name, age) VALUES

('Tom', 37),

('Bob', 41),

('Sam', 28)

;

В данном случае в таблицу будут добавлены три строки.

***Динамическая типизация***

Стоит отметить, что в SQLite (в отличие от многих других популярных систем баз данных) действует динамическая типизиация, т.е. несмотря на объявленный тип данных, можно в поле записать значение другого типа. Иногда это является плюсом, но чаще вызывает проблемы, потому что СУБД никак не сообщает о таких нарушениях.