Содержание

Введение

SQLite — это быстрая и легкая встраиваемая однофайловая СУБД на языке C, которая не имеет сервера и позволяет хранить всю базу локально на одном устройстве. Для работы SQLite не нужны сторонние библиотеки или службы. Понятие «встраиваемый» означает, что СУБД не использует парадигму клиент-сервер. Движок SQLite — не отдельно работающий процесс, с которым взаимодействует программа, а библиотека. Программа компонуется с ней, и движок служит составной частью программы. В качестве протокола обмена применяются вызовы функций (API) библиотеки SQLite.

Дистрибутив и установка

Дистрибутив загрузки SQLite можно скачать по данной ссылке - https://www.sqlite.org/download.html

Установка SQLite в Windows

- **Шаг 1** - Переходим на страницу загрузки SQLite и загружаем предварительно скомпилированные двоичные файлы из раздела Windows.
- **Шаг 2.** Загружаем файлы zlip-sqlite-win32 - *. Zip и sqlite-dll-win32 - *.zip.
- **Шаг 3** - Создаем папку C:\> sqlite и разархивируем над двумя zip-файлами в этой папке, которые предоставят sqlite3.def, sqlite3.dll и sqlite3.exe файлы.
- **Шаг 4** – Добавляем C: \> sqlite в переменную среды PATH и переходим в командную строку, выполняем команду sqlite3, которая должна отобразить следующий результат, как показано в *Примере 1*.

```
C:\>sqlite3
SQLite version 3.25.3 2018-11-29 17:11:07
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

Пример: 1.

Установка SQLite в Linux

- **Шаг 1** - Переходим на страницу загрузки SQLite и загружаем sqlite-autoconf - *. Tar.gz из раздела исходного кода.
- **Шаг 2** - Запускаем следующую команду: *(Пример: 2)*

```
$tar xvfz sqlite-autoconf-3071502.tar.gz
$cd sqlite-autoconf-3071502
$./configure --prefix = /usr/local
$make
$make install
```

Пример: 2

Применение

База данных — это набор структурированной информации. Для ее изменения требуются системы управления — СУБД. Как и любая СУБД, SQLite позволяет записывать новую и запрашивать существующую информацию, изменять ее, настраивать доступ.

Благодаря свойствам SQLite применяется:

- на сайтах с низким и средним трафиком;
- в локальных однопользовательских, мобильных приложениях или играх, не предназначенных для масштабирования;
- в программах, которые часто выполняют прямые операции чтения/записи на диск;
- в приложениях для тестирования бизнес-логики.

SQLite не требует администрирования и работает на мобильных устройствах, игровых приставках, телевизорах, беспилотных летательных аппаратах, камерах, автомобильных мультимедийных системах и т.д. СУБД использует множество программ: Firefox, Chrome, Safari, Skype, XnView, AIMP, Dropbox, Viber и другие

Как работает SQLite

Большинство СУБД используют клиент-серверную архитектуру: данные хранятся и обрабатываются на сервере, а запросы к нему посылает клиент. «Клиент» — это часть программы, с которой взаимодействует пользователь. «Сервером» может быть и отдельный процесс на том же компьютере (так называемый демон), и стороннее устройство, как в случае с сайтами.

SQLite устроена иначе и не имеет сервера. Это значит, что все данные программное обеспечение хранит на одном устройстве. СУБД встраивается в приложение и работает как его составная часть. Если установить на компьютер программу, использующую SQLite, то база данных тоже будет храниться на нем же. Формат базы — один текстовый файл, который можно прочитать на любой платформе. Такой подход повышает производительность и скорость работы.

Работать с SQLite можно как с библиотекой или через SQLite3.

Преимущества SQLite

Высокая скорость. Благодаря особенностям архитектуры SQLite работает быстро, особенно на чтение. Компоненты СУБД встроены в приложение и вызываются в том же процессе. Поэтому доступ к ним быстрее, чем при взаимодействии между разными процессами.

Хранение данных в одном файле. База данных состоит из табличных записей, связей между ними, индексов и других компонентов. В SQLite они хранятся в едином файле (database file), который находится на том же устройстве, что и программа. Чтобы при работе не возникало ошибок, файл блокируется для сторонних процессов перед записью. Раньше это приводило к тому, что записывать данные в базу мог только один процесс единовременно. Но в новых версиях это решается перенастройкой режима работы СУБД.

Минимализм. Создатели SQLite пользуются принципом «минимального полного набора». Из всех возможностей SQL в ней есть наиболее нужные. Поэтому SQLite отличают малый размер, простота решений и легкость администрирования. Для повышения базовой функциональности можно использовать стороннее программное обеспечение и расширения.

Надежность. Код на 100% покрыт тестами. Это означает, что протестирован каждый компонент ПО. Поэтому SQLite считается надежной СУБД с минимальным риском непредсказуемого поведения.

Нулевая конфигурация. Перед использованием СУБД не нужна сложная настройка или длительная установка. Для решения большинства задач ей можно пользоваться «из коробки», без установки дополнительных компонентов.

Малый размер. Полностью сконфигурированный SQLite со всеми настройками занимает меньше 400 Кб. Если использовать СУБД без дополнительных компонентов, размер можно уменьшить до 250 Кб. Он зависит только от количества загруженной информации. Несмотря на малый размер, SQLite поддерживает большинство функций стандарта SQL2 и имеет ряд собственных.

Доступность. SQLite находится в публичном доступе. На ее использование нет правовых ограничений, а владельцем считается общество. Можно открывать, просматривать и изменять исходный код установленного ПО.

Кроссплатформенность. СУБД подходит для UNIX-подобных систем, MacOS и Windows.

Автономность. Система независима от стороннего ПО, библиотек или фреймворков. Чтобы приложение с базой на SQLite работало, дополнительные компоненты не требуются. Также не обязателен доступ в интернет: вся база хранится на устройстве, получить данные можно локально.

Недостатки SQLite

Ограниченная поддержка типов данных. SQLite поддерживает только четыре типа данных, которые реализованы в SQL:

- INTEGER — целое число;
- REAL — дробное число;
- TEXT — текст;
- BLOB — двоичные данные.

Также существует особое значение NULL — отсутствие данных.

Отсутствие хранимых процедур. Так называются блоки кода на SQL, которые сохраняются в базу данных. Хранимые процедуры можно вызывать как отдельные функции, и это удобно, если нужно последовательно выполнить несколько однотипных действий. Но SQLite их не поддерживает из-за особенностей архитектуры.

Ограничения в применении. Отсутствие сервера — преимущество и недостаток одновременно. Без сервера возможности СУБД меньше. Например, к одной базе не смогут обращаться несколько разных устройств.

В SQLite ограничена многопоточность — единовременное выполнение нескольких процессов. Одновременно читать из базы могут несколько процессов, а писать в нее по умолчанию — только один. В версии 3.7.0 в SQLite внедрили возможность записи разными приложениями, но даже так

она уступает клиент-серверным СУБД по возможностям работы с потоками. Поэтому SQLite не подойдет для многопользовательских приложений или программ, записывающих большой объем данных.

Отсутствие бесплатной техподдержки. Стоимость профессиональной технической поддержки от разработчиков — от $1500 в год. Чтобы получить информацию бесплатно, потребуется пользоваться форумами и руководствами от пользователей, а также официальной документацией.

Отсутствие встроенной поддержки Unicode. Unicode — это популярный стандарт кодирования символов. Он включает практически все существующие знаки и буквы, поэтому считается самым распространенным в мире. Без его поддержки приложение не сможет корректно работать с кириллицей, иероглифами и многими другими символами. SQLite «из коробки» не поддерживает Unicode, его нужно настраивать отдельно. Это может вызвать сложности с локализацией.

Конфигурационные файлы

SQLite не нуждается в инсталляции перед использованием. Процедура "setup" для него отсутствует. Отсутствует серверный процесс, который нуждался бы в запуске, остановке или конфигурировании. SQLite не нуждается в администраторе для создания нового экземпляра базы данных или для присвоения пользователям прав доступа. SQLite не использует конфигурационные файлы. Ничто не обязано сообщать системе о выполнении SQLite. Не требуется никаких действий для восстановления после краха системы или некорректного завершения. Здесь ничего не конфликтует. SQLite работает просто и правильно.

Другие, более известные движки баз данных превосходно работают, когда вы запускаете их. Но начальная инсталляция и конфигурация могут быть пугающе сложными

Взаимодействие с веб-серверами и PHP

Хостинг с поддержкой mysql дороже чем хостинг только с php. Тем более количество баз данных ограниченно тарифным планом. Конечно без бд то же никак, но отказаться от использования mysql можно. Ему на замену приходит sqlite.

В отличии от mysql, sqlite является библиотекой, а не серверной программой. Эта библиотека уже встроена в интерпритатор php и не требует каких то ухищрений с администрированием. Для работы достаточно наличия на сервере только интерпритатора php. Это значительно снижает стоимость у хостера тарифного плана. Sqlite пишет все свои данные в файл, который создаётся на сервере. Для каждой бд создаётся свой файл. Ограничением по количеству бд может являться только объём жёсткого диска под сайт. Так как все бд хранятся в файлах то для резервирования базы или переезда к другому хостеру файлы бд всего лишь нужно скопировать.

Разберём подключение sqlite и sqlite3 под windows.

Подключение sqlite

Будем считать что веб сервер apache и php у нас уже установлен. В зависимости от версии интерпритатора в папке ext корневой директории PHP должны быть файлы php_pdo.dll, php_pdo_sqlite.dll, php_sqlite.dll. Для того что бы подключить sqlite, открываем файл php.ini в любом текстовом редакторе и раскомментируем директивы подключения нужных нам

модулей. Модули, закомментированные при помощи точки с запятой ;, как правило, требуют дополнительных внешних библиотек. Директивы должны выглядеть так:

- **extension = имя_модуля**

Для подключения php_sqlite.dll в php.ini, нужно сначала подключить расширения php_pdo.dll и php_pdo_sqlite.dll. Только в такой последовательности:

- **extension=php_pdo.dll**
- **extension=php_pdo_sqlite.dll**
- **extension=php_sqlite.dll**

Строки extension=php_pdo.dll может в ini и не быть, раскомментируем тогда только extension=php_pdo_sqlite.dll и extension=php_sqlite.dll После всего этого перезапустим апачь. Для проверки правильности установки запустим строку кода:

```
1 echo sqlite_libversion();
```

Должно показать версию sqlite.

Php sqlite примеры:

Создание файла БД

```
1     $db = sqlite_open("mytest.db");
2 //ООП интерфейс
3 //создаём объект от спец. класса SQLiteDatabase
4     $db1 = new SQLiteDatabase("mytest1.db");
5     sqlite_close($db);
6 //ООП интерфейс
7     unset($db1);//убиваем объект
```

Пример 3.

Создание таблицы в базе данных

```php
//проверим существует ли БД
//если БД нет то создадим её и таблицу в ней msages
    if(!file_exists("mytest.db")){
        $db=new SQLiteDatabase("mytest.db");
        $sql="CREATE TABLE msages(
            id INTEGER PRIMARY KEY,
            fname TEXT,
            email TEXT,
            msage TEXT,
            datetime INTEGER,

        )";
        $db->query($sql);
    }else{
//если бд есть то просто подключ. к ней
        $db=new SQLiteDatabase("mytest.db");
    }
```

Пример 4.

Выполнение запроса

```php
    $sql = "SELECT * FROM table";
    $db = sqlite_open("test.db");
    $res = sqlite_query($db, $sql);
    sqlite_close($db);
//ООП интерфейс
    $db = new SQLiteDatabase("test.db");
    $res = $db->query($sql);
    unset($db);
```

Пример 5.

Получение результата

```
    $result = sqlite_query($db, $sql);
    $row = sqlite_fetch_array($result, [TYPE]);
// ООП-интерфейс
    $result = $db->query($sql);
    $row = $result->fetch([TYPE]);
Константы типов:
SQLITE_BOTH // По умолчанию
SQLITE_NUM
SQLITE_ASSOC
```

Пример 6.

Получение результата в виде массива

```
    $result = sqlite_array_query($db,
    $sql);
// ООП-интерфейс
    $result = $db->arrayQuery($sql);
    foreach($result as $row){
        echo $row[0];
    }
```

Пример 7.

Создание и удаление таблицы. Прикрепление базы данных

Для создания таблиц используется команда CREATE TABLE. Общий формальный синтаксис команды CREATE TABLE:

```
CREATE TABLE название_таблицы
(название_столбца1 тип_данных атрибуты_столбца1,
 название_столбца2 тип_данных атрибуты_столбца2,
 .....................................................
 название_столбцаN тип_данных атрибуты_столбцаN,
 атрибуты_уровня_таблицы
)
```

После команды CREATE TABLE указывается название таблицы. Имя таблицы выполняет роль ее идентификатора в базе данных, поэтому оно должно быть уникальным. Кроме того, оно не должно начинаться на "sqlite_", поскольку названия таблиц, которые начинаются на "sqlite_", зарезервированы для внутреннего пользования.

Затем после названия таблицы в скобках перечисляются названия столбцов, их типы данных и атрибуты. В самом конце можно определить атрибуты для всей таблицы. Атрибуты столбцов, а также атрибуты таблицы указывать необязательно.

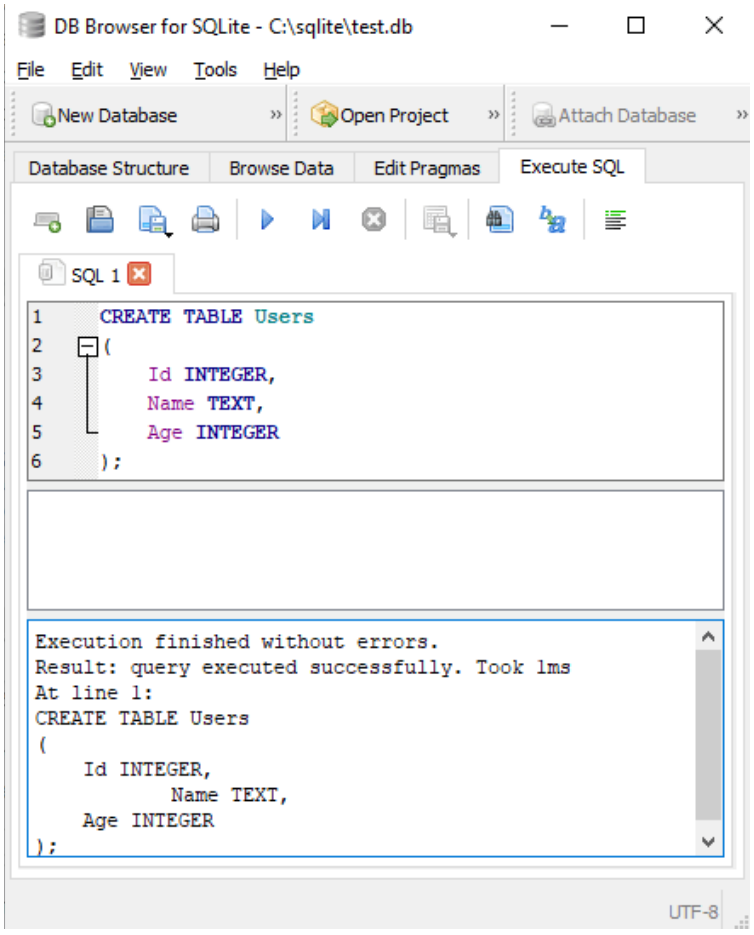
Создадим простейшую таблицу. Перед выполнением команды CREATE TABLE вне зависимости, что мы используем - консольный клиент sqlite3, графический клиент DB Browser for SQLite или какой-то другой клиент, вначале откроем базу данных, где мы хотим создать таблицу.

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.



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
);
```

Если таблицы Users нет, она будет создана. Если она есть, то никаких действий не будет производиться, и ошибки не возникнет.

Прикрепление базы данных

Также мы можем прикрепить базу данных и затем в ней уже создать базу данных.

Для прикрепления базы данных применяется команда ATTACH DATABASE:

```
ATTACH DATABASE 'C:\sqlite\test.db' AS test;
```

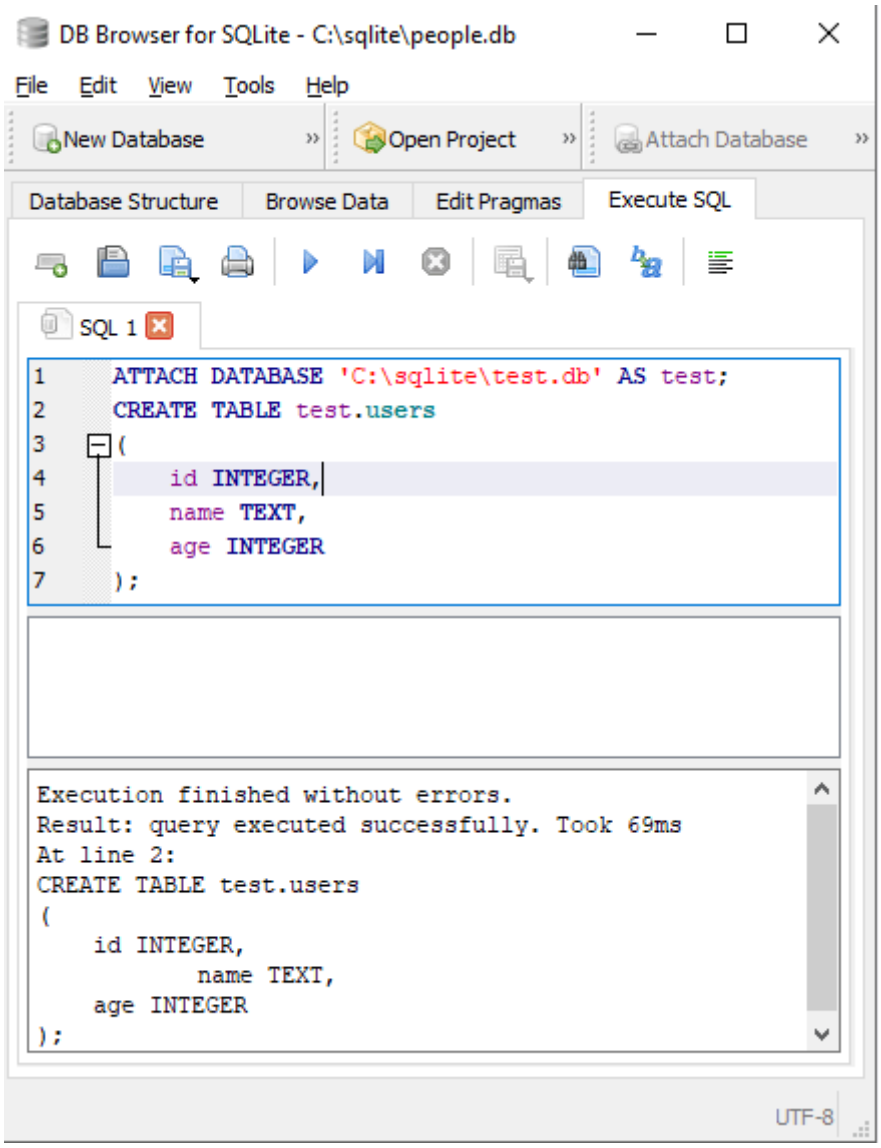
После команды ATTACH DATABASE указывается путь к файлу базы данных (в данном случае это путь "C:\sqlite\test.db"). Затем после оператора AS идет псевдоним, на который будет проецироваться база данных. То есть в коде для обращения к базе данных "C:\sqlite\test.db" будет применяться имя "test". При обращении к таблице из этой базы данных, сначала указывается псевдоним базы данных и через точку название таблицы:

```
псевдоним_бд.таблица
```

Например, создадим таблицу в прикрепленной базе данных:

```
ATTACH DATABASE 'C:\sqlite\test.db' AS test;
CREATE TABLE test.users
(
    id INTEGER,
    name TEXT,
    age INTEGER
);
```

Для создания таблицы users в бд test.db название таблицы предваряется псевдонимом: test.users.



И после открытия базы данных test.db в ней можно будеть увидеть таблицу users.

Удаление таблиц

Для удаления таблицы применяется команда DROP TABLE, после которой указывается название удаляемой таблицы. Например, удалим таблицу users:

```
DROP TABLE users;
```

По аналогии с созданием таблицы, если мы попытаемся удалить таблицу, которая не существует, то мы столкнемся с ошибкой. В этом случае опять же с помощью операторов IF EXISTS проверять наличие таблицы перед удалением:

```
DROP TABLE IF EXISTS users;
```

Основные операции с данными

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:

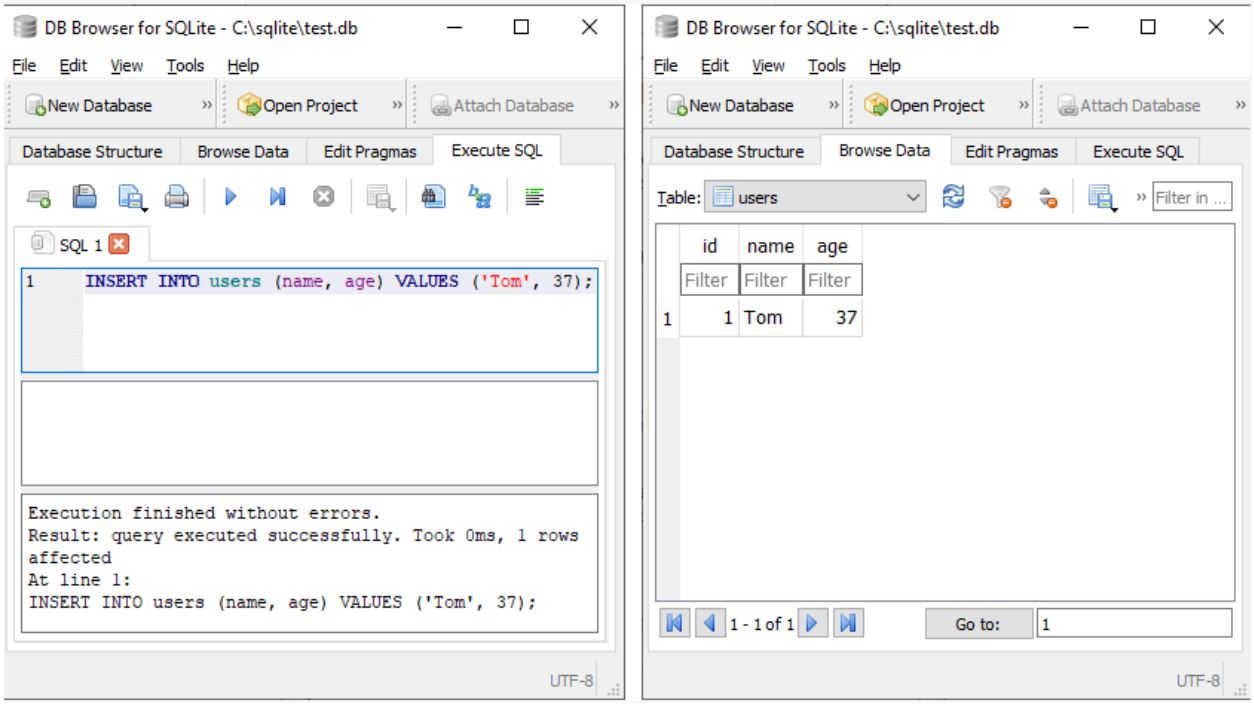
```
CREATE TABLE users
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    age INTEGER
);
```

Добавим в эту таблицу одну строку с помощью следующего кода:

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

После названия таблицы указываны два стобца, в которые мы хотим

выполнить добавление данные - (name, age). После оператора VALUES указаны значения для этих столбцов. Значения будут передаваться столбцам по позиции. То есть стобцу name передается строка "Tom', столбцу age - число 37. И после успешного выполнения данной команды в таблице появится новая строка:



Стоит отметить, что при добавлении данных 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 можно было бы не указывать названия столбцов:

```
1 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 NULL

Также мы можем опускать при добавлении такие столбцы, которые поддерживают значение NULL (которые не имеют ограничения NOT NULL):

```
INSERT INTO users (name) VALUES ('Sam');
```

В данном случае для столбца age не указано значение, и поскольку данный столбец поддерживает значение NULL, то для него будет установлено значение NULL.

Также подобным столбцам, которые поддерживают NULL, можно явным образом передать NULL:

```
INSERT INTO users (name, age) VALUES (NULL, NULL);
```

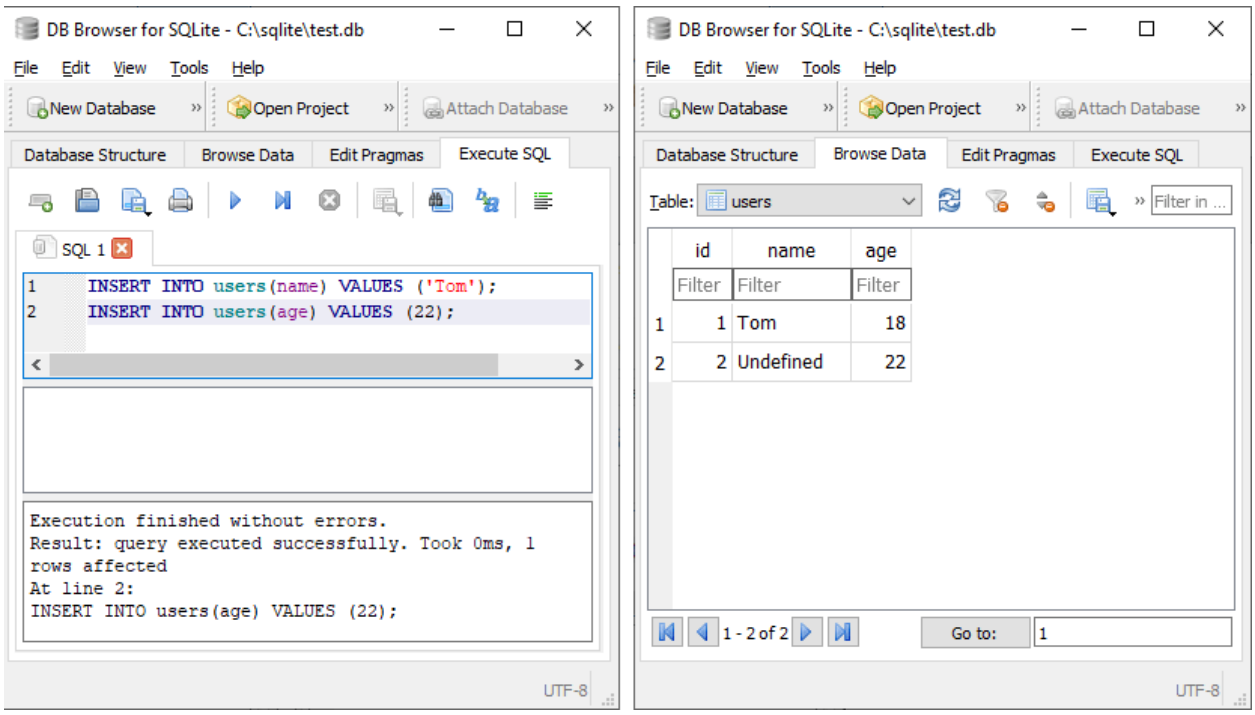
Значения по умолчанию

Если для столбца задано ограничение DEFAULT, то есть значение по умолчанию, то для него тоже можно не передавать значение. Например, возьмем следующую таблицу:

```
CREATE TABLE users
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT DEFAULT 'Undefined',
    age INTEGER DEFAULT 18
);
```

Теперь столбцы name и age имеют значения по умолчанию. При добавлении данных из можно опустить:

```
INSERT INTO users(name) VALUES ('Tom');
INSERT INTO users(age) VALUES (22);
```



Если все столбцы поддерживают значения по умолчанию или автогенерацию или значение NULL, то с помощью ключевого слова DEFAULT можно явно указать, что в качестве значения будут использоваться значения по умолчанию:

```
INSERT INTO users DEFAULT VALUES;
```

В этом случае столбцы, для которых определено значение по умолчанию, получат это значение. Остальные столбцы получать значение NULL.