

SQLite

1. [WHAT IS SQLite?](#)

SQLite is a software library that provides a relational database management system. The lite in SQLite means lightweight in terms of setup, database administration, and required resource.

SQLite features: self-contained, serverless, zero-configuration, transactional.

A. Serverless

An RDBMS such as MySQL, PostgreSQL, etc., requires a separate server process to operate. The applications that want to access the database server use the TCP/IP protocol to send and receive requests. This is called a client/server architecture.

SQLite does NOT require a server to run. SQLite database is integrated with the application that accesses the database. The applications that interact with the SQLite database read and write directly from the database files stored on disk.

B. Self-Contained

SQLite requires minimal support from the operating system or external library. This makes SQLite usable in any environments especially in embedded devices like iPhones, Android phones, game consoles, handheld media players, etc.

SQLite is developed using ANSI-C. The source code is available as a big `sqlite3.c` and its header file `sqlite3.h`. If you want to develop an application that uses SQLite, you just need to drop these files into your project and compile it with your code.

C. Zero-configuration

Because it is serverless, you don't need to "install" SQLite before using it. There is no server process that needs to be configured, started, and stopped.

In addition, SQLite does not use any configuration files.

D. Transactional

All transactions in SQLite are fully ACID-compliant. It means all queries and changes are Atomic, Consistent, Isolated, and Durable.

In other words, all changes within a transaction take place completely or not at all even when an unexpected situation occurs, like application crash, power failure, or operating system crash.

E. Non SQL-standard

It is important to note that SQLite does not implement the [SQL standard](#). As such, there are differences between the SQL-standard and the SQL dialect in SQLite.

There are also a few unsupported features of SQL92 in SQLite.

2. HOW TO DOWNLOAD & INSTALL SQLite

A. Download SQLite

To download SQLite, you open the [download page](#) of the SQLite website.

SQLite provides various versions for various platforms e.g., Windows, Linux, Mac, etc. You should choose an appropriate version to download.

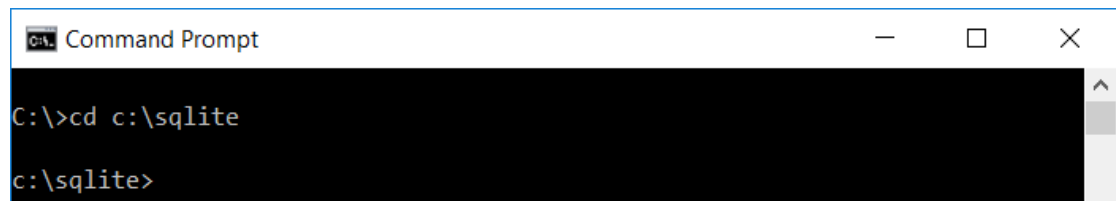
For example, to work with SQLite on Windows, you download the command-line shell program, included in the bundle “command-line tools for managing SQLite database files”: [sqlite-tools-win32-x86-3170000.zip](#)

B. Install SQLite

Installing SQLite is simple and straightforward.

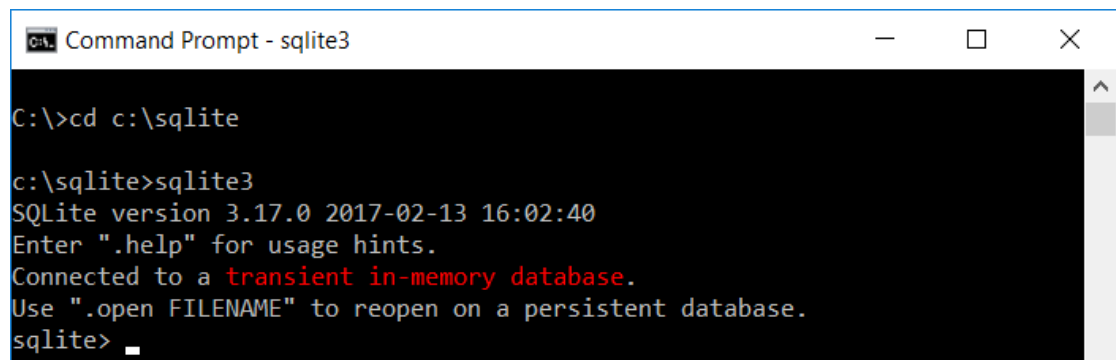
- i. First, create a new folder e.g., C:\sqlite.
- ii. Second, extract the content of the file that you downloaded in the previous section to the C:\sqlite folder. You should see the sqlite3.exe is in the C:\sqlite folder.

To verify the installation, open the command line window and navigate to the C:\sqlite folder.



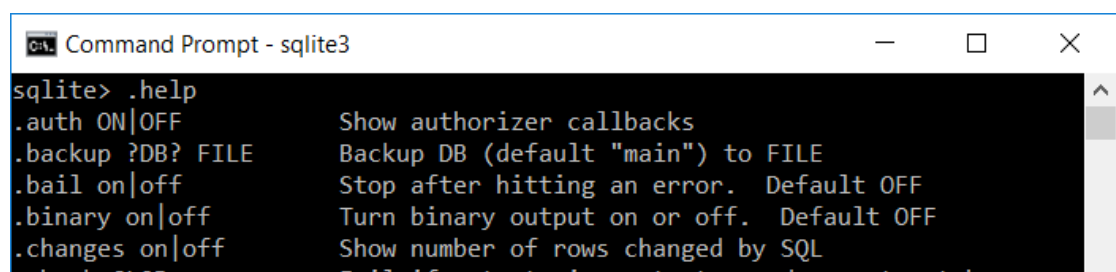
```
C:\>cd c:\sqlite
c:\sqlite>
```

Enter sqlite3, you should see the following window:



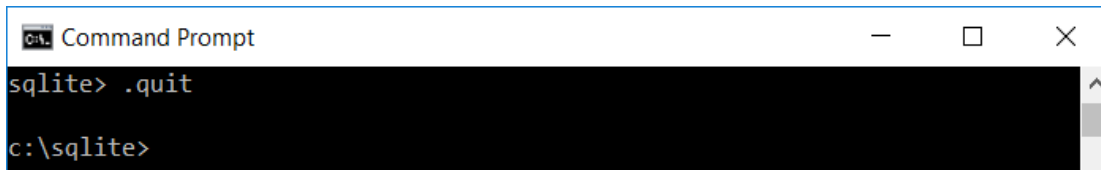
```
C:\>cd c:\sqlite
c:\sqlite>sqlite3
SQLite version 3.17.0 2017-02-13 16:02:40
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> _
```

You can type the .help command from the sqlite> prompt to see all available commands in sqlite3.



```
sqlite> .help
.auth ON|OFF          Show authorizer callbacks
.backup ?DB? FILE      Backup DB (default "main") to FILE
.bail on|off           Stop after hitting an error. Default OFF
.binary on|off         Turn binary output on or off. Default OFF
.changes on|off        Show number of rows changed by SQL
.check CLIP            Fail if output since testcase does not match
```

To quit the sqlite>, you use .quit command as follows:



```

C:\> Command Prompt
sqlite> .quit
c:\sqlite>
  
```

C. Install SQLite GUI tool

The sqlite3 command is excellent... However, sometimes, you want to work with the SQLite databases using an intuitive GUI tool.

There are many GUI tools for managing SQLite databases available ranging from a freeware to commercial licenses.

You can use [SQLite Online](#), an online user-friendly interface for managing databases.

[SQLite Studio](#) is another free GUI tool for managing SQLite databases. You can [download](#) the SQLite studio at the homepage of SQLite, extract the file into a folder (e.g., C:\sqlite\gui\) and run it by executing the SQLiteStudio application file.

3. SQLite COMMANDS

SQLite commands – also called dot commands because they start with a dot (.) – are useful for programmers. Contrary to most SQL commands, they should not be terminated by a semi-colon (;).

For a listing of the [available dot commands](#), you can enter ".help" at any time.

Some relevant commands:

Command	Description
.databases	List names and files of attached databases
.dump	Dump the database or a specific table in an SQL text format
.exit / .quit	Exit the SQLite shell
.headers	Turn display of headers on or off, when displaying output of SQL statements
.help	Show available commands
.import	Import data from a file into a table
.mode	Set output mode
.open	Open a database from a file
.output	Redirect the output to a file
.read	Execute SQL statements from a file
.save	Write in-memory database into a file
.schema	Show the CREATE statements used for the whole database or for a specific table
.show	Show current values for various settings
.tables	List names of tables

4. USING THE SQLite SHELL

A. Executing SQL statements

When interacting with the SQLite shell, there are essentially two ways to run SQL statements. The simplest way is simply to write the statements directly in the prompt. For instance, consider writing the following sequence of commands and statements in the prompt (you can also copy/paste these lines directly into SQLite):

```
.mode columns
.headers on
drop table if exists T;
create table T (A text, B text);
insert into T values ('Hello,', 'world!');
select * from T;
```

If all goes well, you will see the following output:

A	B
-----	-----
Hello,	world!

B. Reading from a file

Another, more powerful way is to save the sequence of commands and statements in a text file (say, “helloworld.sql”), thus creating a script that you can run via .read:

```
Sqlite> .read helloworld.sql
A          B
-----
Hello,     world!
```

C. Saving to a file

In any case, you can later save the SQL statements that you’ve used both to create the database (CREATE TABLE statements) and to populate its tables (INSERT INTO statements). For that, you can use the .schema (for the schema creation statements only) or the .dump (for both creation and insertion statements) commands:

```
sqlite> .schema
CREATE TABLE T (A text, B text);

sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE T (A text, B text);
INSERT INTO "T" VALUES('Hello,', 'world!');
COMMIT;
sqlite>
```

If you want to save these statements into a file you can later read from (using the .read command as shown above), all you need to do is to use the .output command to redirect any output to a file:

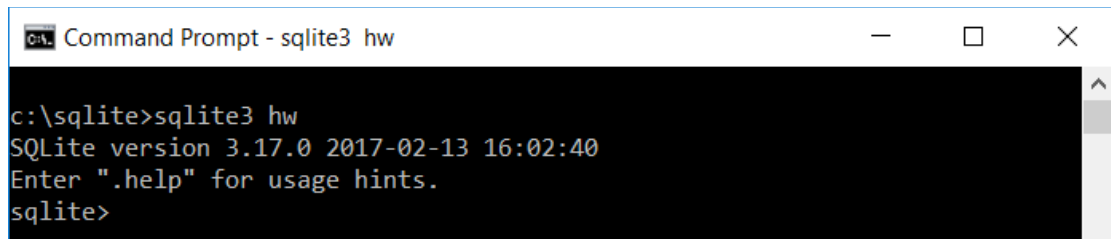
```
sqlite> .output helloworld
sqlite> .dump
```

The contents of the “helloworld” file should be the statements listed above.

D. Keeping the Database state in a file

The state of the database can be maintained persistently in a file, avoiding the need to store SQL statements that can be later read (using `.read` as shown above).

In order to do that, you simply need to add an argument to the `sqlite3` application: the name of the database to use.

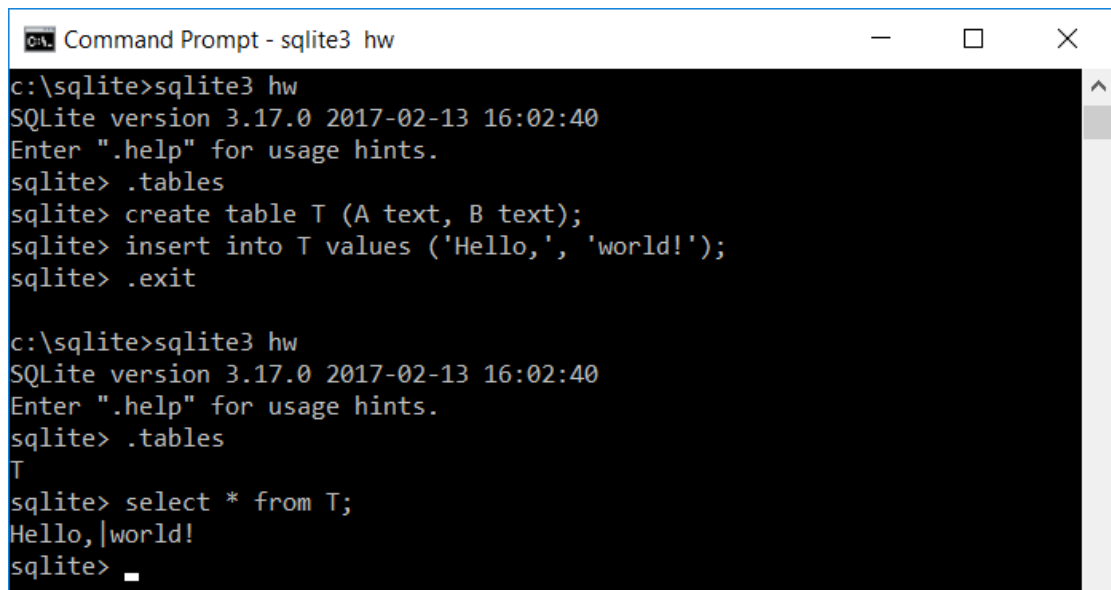


```
C:\> Command Prompt - sqlite3 hw

c:\sqlite>sqlite3 hw
SQLite version 3.17.0 2017-02-13 16:02:40
Enter ".help" for usage hints.
sqlite>
```

If the file provided as an argument to `sqlite3` does not exist, SQLite will create it automatically. Any changes made in the database are persistently stored in that file, without the need for specific save commands (such as `.dump`).

The following example illustrates the creation and use of a persistent database:



```
C:\> Command Prompt - sqlite3 hw

c:\sqlite>sqlite3 hw
SQLite version 3.17.0 2017-02-13 16:02:40
Enter ".help" for usage hints.
sqlite> .tables
sqlite> create table T (A text, B text);
sqlite> insert into T values ('Hello,', 'world!');
sqlite> .exit

c:\sqlite>sqlite3 hw
SQLite version 3.17.0 2017-02-13 16:02:40
Enter ".help" for usage hints.
sqlite> .tables
T
sqlite> select * from T;
Hello,|world!
sqlite>
```

An alternative to using the database file name as an argument to `sqlite3` is to use the `.open` command within SQLite. This still ensures that any changes to the database are automatically stored in the persistent file.

Similarly, if no file was provided as an argument to `sqlite3` you can use the `.save` command to save the database to a file. Notice however that this does not trigger automatic saves of any additional changes that are made in the database.

5. PRAGMA STATEMENTS

The PRAGMA statement is an SQL extension specific to SQLite and used to modify the operation of the SQLite library or to query the SQLite library for internal (non-table) data.

A. Syntax

A PRAGMA statement is formed by the keyword PRAGMA followed by the actual “pragma” that is being defined. A pragma has a name and can take one argument. The argument may be in parentheses or separated from the pragma name by an equal sign. The two syntaxes yield identical results. In many pragmas, the argument is a Boolean and can be one of: 1/yes/true/on; 0/no/false/off.

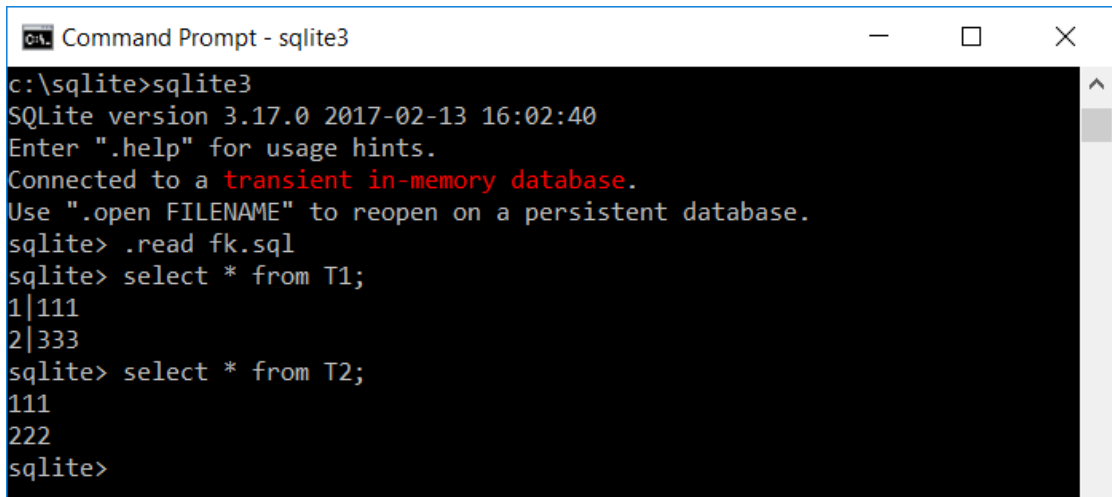
B. PRAGMA foreign_keys

This is a very important pragma, as it may be used to set the enforcement of [foreign key constraints](#).

Consider the following SQL statements, which violate a foreign key constraint:

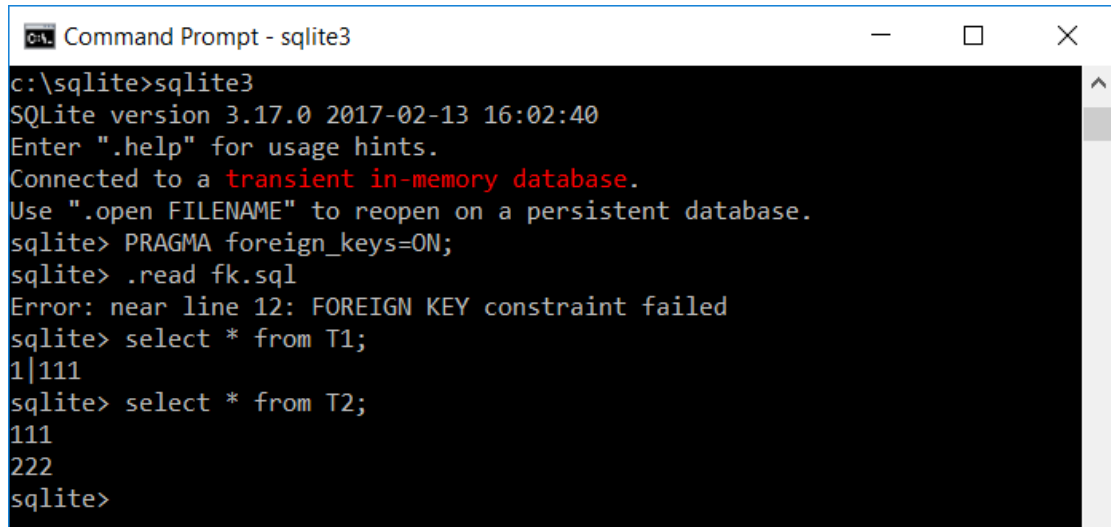
```
create table T1 (  
    Id NUMBER PRIMARY KEY,  
    Id2 NUMBER REFERENCES T2 (Id) );  
  
create table T2 (  
    Id NUMBER PRIMARY KEY);  
  
insert into T2 values (111);  
insert into T2 values (222);  
  
insert into T1 values (1,111);  
insert into T1 values (2,333);
```

Since in the current version of SQLite foreign key constraint enforcement is disabled by default, loading these statements from a file, say “fk.sql”, does not raise any error:



```
C:\> Command Prompt - sqlite3  
c:\sqlite>sqlite3  
SQLite version 3.17.0 2017-02-13 16:02:40  
Enter ".help" for usage hints.  
Connected to a transient in-memory database.  
Use ".open FILENAME" to reopen on a persistent database.  
sqlite> .read fk.sql  
sqlite> select * from T1;  
1|111  
2|333  
sqlite> select * from T2;  
111  
222  
sqlite>
```

If we turn on foreign key constraint enforcement, we get an error detecting the constraint violation:



```
Command Prompt - sqlite3
c:\sqlite>sqlite3
SQLite version 3.17.0 2017-02-13 16:02:40
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> PRAGMA foreign_keys=ON;
sqlite> .read fk.sql
Error: near line 12: FOREIGN KEY constraint failed
sqlite> select * from T1;
1|111
sqlite> select * from T2;
111
222
sqlite>
```

6. DOCUMENTATION AND TUTORIALS

Official SQLite documentation can be found at <https://www.sqlite.org/docs.html>

Relevant tutorials include:

- <http://www.sqlitetutorial.net/>
- <https://www.tutorialspoint.com/sqlite/>