DocsLite for SQLite

Peter Lavin

2016-10-22

Table of Contents

You have so many passwords that you need a database to keep track of them. But what you don't need is the overhead usually associated with databases.

You need SQLite—small, efficient and available for any operating system.

This article documents creating, populating, updating and altering a table using SQLite. Some knowledge of SQL is assumed.

Using SQLite

You can install SQLite using your package manager or download it from the SQLite website (https://www.sqlite.org/).

You open a SQLite database by opening a command window and passing the database name as a parameter, for example, **`sqlite3 mydatabase.sqlite`**. If the database does not exist, a database named `mydatabase.sqlite` is created in the current directory. Running this command opens the SQLite shell and shows the prompt, `sqlite>`.

The SQLite shell is used for executing SQL statements but there are also "dot commands" that simplify a number of operations. A list of commands used in this article follows:

- **.exit** – exit the SQLite shell and return to the command prompt
- **.import** *`filename table`* – import the specified file into the named table. If the table doesn't exist, create it using the first row for the column names.
- **.mode** csv | tabs | line | html | ... – change the way that output is formatted. The `tabs` and `csv` parameters also affect files imported using **.import**.

- **.dump** [*table*] – dump the contents of the current database or of the specified table.
- **.output** [*filename*] – send output to the specified file. When no parameter is specified, output is sent to the screen. This command is often used in conjunction with the **.dump** command to direct output to a file.
- **.read** *filename* – execute the SQL commands in the specified file. This command is often used with the output of the **.dump** command.
- **.headers** on | off – show column names as the first record
- **.help** – view all available dot commands

If you find that you constantly use specific commands you can specify them in a `.sqliterc` configuration file stored in your home directory. For example, if you regularly import and export comma-separated values (CSV) files, create a configuration file with the contents:

```
.mode csv
```

This mode uses a comma separator when you issue `SELECT` statements and expects CSV files with the **.import** command.

> ***Note***
>
> *Only commands used in this article are mentioned here. For complete documentation see https://www.sqlite.org/docs.html.*

Creating a table

SQLite makes it very easy to create a table and populate it with data. Suppose you have the following CSV file where the first row contains the field names:

```
type,url,username,password,notes
online,http://www.example.com,myusername,mypassword,comments about this record
local,,username,password,login for laptop
...
```

If this file is named `file.csv`, you can create a table *and* import your data by opening SQLite and entering the following dot commands:

```
sqlite> .mode csv
sqlite> .import file.csv tblpasswords
```

> ***Note***
>
> *To import data from a tab-delimited file change the mode to `tabs`.*

Importing this file loads your data and creates a bare-bones table defined in the following way:

```
CREATE TABLE tblpasswords(
  "type" TEXT,
  "url" TEXT,
  "username" TEXT,
  "password" TEXT,
  "notes" TEXT
);
```

Such a table may be perfectly suitable for your purposes but if you need something a bit more sophisticated you can explicitly create a table by using a SQL statement such as the following:

```
CREATE TABLE tblpasswords(
  type TEXT DEFAULT "online",
  url TEXT NOT NULL,
  username TEXT DEFAULT "username@email.com",
  password TEXT,
  notes TEXT,
  PRIMARY KEY (url),
  CHECK(type = "local" OR type = "online" OR type = "other")
);
```

This creates a table with default values for the `type` and `username` fields, a primary key and constraints on the values that can be assigned to the `type` field. You can import data into this table using the exact same dot commands as shown above. Just make sure that the first row doesn't contain the field names.

If you know exactly what you want then create a table with constraints but otherwise create a bare-bones table. It's easy enough to alter your table after the fact; doing this is described in the section called “Restructuring your table” [6].

Adding records

If you have comma-separated or tab-separated data as shown in the section called “Creating a table” [2] you can also insert additional data using the dot command, **.import**. Be sure to set the appropriate mode first and make sure that your data file doesn't include headers. This can be especially useful if you are importing data from a spreadsheet.

To insert a record using SQL, execute the following statement:

```
INSERT INTO
  tblpasswords
  VALUES(
    'online',
    'http://example.com',
    'myusername',
    'password',
    'some notes'
  );
```

If you want to take advantage of default values for fields you must use the following syntax:

```
INSERT INTO
  tblpasswords (
    url,
    password
  )
  VALUES(
    'http://example.com',
    'password'
  );
```

If you created a table with the defaults shown in the section called “Creating a table” [2], the `type` field will default to `online` and the `username` field will be `username@email.com`. The `notes` field is not specified and has no default value so will be NULL. Note that when you load data using the **.import** command you cannot take advantage of default values.

Viewing data

The simplest way to view your data is to use a `SELECT` statement. You can do this from the SQLite prompt or in batch mode but, without too much effort, you can view your data as a web page by using the `-html` option. The following script (for Unix-like operating systems) creates an HTML page of all your records:

```
#!/bin/bash
echo "<html><body><h1>Passwords</h1><table>" > /path/to/passwords.html
sqlite3 -html /path/to/passwords.sqlite \
  "SELECT * FROM tblpasswords ORDER BY url;" >> /path/to/passwords.html
echo "</table></body></html>" >> /path/to/passwords.html
```

> ***Note***
>
> *The SQLite options are useful when running in batch mode. They are similar to the dot commands; to view them issue the command* ***`sqlite3 --help`****.*

To create a script for Windows save the following text file:

```
echo ^<html^>^<body^>^<h1^>Passwords^</h1^>^<table^> > C:\path\passwords.html
sqlite3 -html C:\path\passwords.sqlite ^
  "SELECT * FROM tblpasswords ORDER BY url;" >> C:\path\passwords.html
echo ^</table^>^</body^>^</html^> >> C:\path\passwords.html
```

Select specific fields or add to the `ORDER BY` clause to change how your data is displayed. You can also add a command to open the output file in your favorite browser.

> ***Warning***
>
> *Be sure you have rights to the directory where you write the file and be sure to remove the HTML file after use.*

Getting specific records

To look up a specific record use a `LIKE` clause with the wild card character '`%`':

```
sqlite3 passwords.sqlite "SELECT * FROM tblpasswords WHERE url LIKE '%amazon%';"
```

Since this is a command that you will often repeat, you can create a script for Linux or macOS by saving the following:

```
#!/bin/bash
sqlite3 -line passwords.sqlite \
"SELECT * FROM tblpasswords WHERE url LIKE '%${1}%';"
```

This script file uses the `-line` option to output each field on a separate line, improving readability. The use of the SQLite wild card character, '`%`', means that you can look for a partial rather than a complete word. You can also search additional fields by adding an `OR` clause such as `OR notes LIKE '%${1}%';`. The parameter `${1}` is passed from the command line by invoking this script as follows: `scriptname.sh amazon`, where 'amazon' is the value you are looking for in the `url` field.

The equivalent script for Windows is:

```
sqlite3 -line passwords.sqlite ^
"SELECT * FROM tblpasswords WHERE url LIKE '%%%1%%';"
```

Creating a Windows batch file is a bit messier because the '`%`' character identifies a parameter passed to a batch file and must be escaped (by doubling it) when used as a wild card.

Backing up your database

You should always have a backup of your database and with SQLite you can simply use the operating system to make a copy.

If you'd rather have the SQL needed to reconstruct your tables and data you can go to the command line and issue the command:

```
sqlite3 password.sqlite .dump > outfile.sql
```

This command dumps the contents of your database to a file named `outfile.sql`. This file includes `INSERT` statements needed to recreate all your data and the `CREATE` statements for all database objects.

Use the output file, `outfile.sql`, to create a new database as follows:

```
sqlite3 newpassword.sqlite ".read outfile.sql"
```

If `newpassword.sqlite` doesn't already exist, this command will create it.

Updating records

There are no dot commands for updating records; you must use a SQL statement. For example, you can update a password in the following way:

```
UPDATE
  tblpasswords
  SET
    password = 'newpassword'
  WHERE
    url = 'http://www.amazon.com';
```

> ***Warning***
>
> *Back up your database prior to updating it. If your table does not have a primary key, be sure to use a unique value or a unique combination of values when you update records. Otherwise you may change more than one record.*

Restructuring your table

You don't always know the exact structure of a table right from the start so you may have to make alterations. SQLite's `ALTER TABLE` syntax is limited; you can rename tables and you can only add columns following the last column.

Nevertheless, the dot commands make an iterative approach to table design relatively simple. Follow these steps to change your table design:

1. Backup your existing database and rename your existing table. The syntax for renaming is: `ALTER TABLE tblpasswords RENAME TO old_tblpasswords;`

2. If convenient you can use a `.sqliterc` file to change default settings. Settings that you may wish to use are `.headers on` and `.mode csv`. By turning the headers on and setting the mode to `csv` you can create an output file capable of creating a table and populating it when it is reimported. If you prefer, you can change these settings from the SQLite prompt.

3. If you are not going to create a table as you import data, create a new table as described in the section called “Creating a table” [2].

4. Create a `SELECT` statement suitable for the new table structure. If need be, change the order of the fields. You can create new fields by selecting a literal string (empty or otherwise), `'8080' AS port`, for example.

5. Test that your `SELECT` statement produces the desired result.

6. Redirect output to file by entering **`.output outfile.csv`**.

7. Create a data file by executing your `SELECT` statement.

8. Import your data file using the dot command: **`.import outfile.csv tblpasswords`**.

9. Redirect output to the screen by using the command **.output** with no parameter and then verify that the data has been imported.

10. Drop the old table if required.

About the Author

Peter Lavin is a technical writer who has been published in a number of print and online magazines. He is the author of "Object Oriented PHP", published by No Starch Press, "DocBook for Writers", published by softcoded.com and a contributor to "PHP Hacks" by O'Reilly Media. His latest book, "HTML to MadCap Flare", is available from XML Press.