...making Linux just a little more fun!

<-- prev | next -->

SQLite Tutorial: Common Commands and Triggers

By Mike Chirico

This article explores common methods in SQLite such as running commands from the shell prompt and creating triggers for populating time stamped fields with either UTC or local time values. In addition, delete, update, and insert trigger examples will be explored in detail.

All examples can be found in sqlite_examples.tar.gz (local copy), and I would encourage you to download and run these examples as you read through this document.

The home page for sqlite3 is www.sqlite.org and the source for sqlite3 can be downloaded at www.sqlite.org/download.htm. This tutorial was done with the source version 3.0.8

Getting Started - Common Commands

To create a database file, run the command "sqlite3" followed by the database name. For example, to create the database "test.db" run the sqlite3 command as follows:

```
$ sqlite3 test.db
SQLite version 3.0.8
Enter ".help" for instructions
sqlite> .quit
$
```

The database file test.db will be created, if it does not already exist. Running this command will leave you in the sqlite3 environment. There are 3 ways to safely exit this environment (.q, .quit, or .exit).

You do not have to enter the sqlite3 interactive environment. Instead, you could perform all commands at the shell prompt, which is ideal when running bash scripts and commands in an ssh string. Below is an example of how you would create a simple table from the command prompt.

```
$ sqlite3 test.db  "create table t1 (t1key INTEGER
             PRIMARY KEY,data TEXT,num double,timeEnter DATE);"
```

After table t1 has been created, data can be inserted as follows:

```
$ sqlite3 test.db  "insert into t1 (data,num) values ('This is sample data',3);"
$ sqlite3 test.db  "insert into t1 (data,num) values ('More sample data',6);"
$ sqlite3 test.db  "insert into t1 (data,num) values ('And a little more',9);"
```

As expected, doing a select returns the data in the table. Note, the primary key "t1key" auto increments; however, there are no default values for timeEnter. To populate the timeEnter field with the time, an update trigger is needed. An important note on the PRIMARY KEY: do not use the abbreviated "INT" when working with the PRIMARY KEY. You must use "INTEGER", for the primary key to update.

```
$ sqlite3 test.db  "select * from t1 limit 2";
1|This is sample data|3|
2|More sample data|6|
```

In the statement above, the limit clause is used and only 2 rows are displayed. For a quick reference of SQL syntax statements available with SQLite, see the link syntax. There is an offset option to the limit clause. For instance, the third row is equal to the following: "limit 1 offset 2".

```
$ sqlite3 test.db "select * from t1 order by t1key limit 1 offset 2";
3|And a little more|9|
```

The ".table" command shows the table names. For a more comprehensive list of tables, triggers, and indexes created in the database, query the master table "sqlite_master" as shown below.

```
$ sqlite3 test.db ".table"
t1

$ sqlite3 test.db "select * from sqlite_master"
table|t1|t1|2|CREATE TABLE t1 (t1key INTEGER
```

```
                PRIMARY KEY,data TEXT,num double,timeEnter DATE)
```

All SQL information and data inserted into a database can be extracted with the ".dump" command.

```
$ sqlite3 test.db ".dump"
BEGIN TRANSACTION;
CREATE TABLE t1 (t1key INTEGER
             PRIMARY KEY,data TEXT,num double,timeEnter DATE);
INSERT INTO "t1" VALUES(1, 'This is sample data', 3, NULL);
INSERT INTO "t1" VALUES(2, 'More sample data', 6, NULL);
INSERT INTO "t1" VALUES(3, 'And a little more', 9, NULL);
COMMIT;
```

The contents of the ".dump" can be filtered and piped to another database. Below table t1 is changed to t2 with the sed command, and it is piped into the test2.db database.

```
 $ sqlite3 test.db ".dump"|sed -e s/t1/t2/|sqlite3 test2.db
```

Triggers

An insert trigger is created below in the file "trigger1". The Coordinated Universal Time (UTC) will be entered into the field "timeEnter", and this trigger will fire after a row has been inserted into the table t1. Again, this trigger will fire after the row has been inserted.

```
-- ******************************************************************
--   Creating a trigger for timeEnter
--     Run as follows:
--            $ sqlite3 test.db < trigger1
-- ******************************************************************
CREATE TRIGGER insert_t1_timeEnter AFTER  INSERT ON t1
BEGIN
 UPDATE t1 SET timeEnter = DATETIME('NOW')  WHERE rowid = new.rowid;
END;
-- ******************************************************************
```

The AFTER specification in ..."insert_t1_timeEnter AFTER..." is necessary. Without the AFTER keyword, the rowid would not have been generated. This is a common source of errors with triggers, since AFTER is NOT the default, so it must be specified. In summary, if your trigger depends on newly created data, in any of the fields from the created row, which was the case for us in this example since we need the rowid, then, the AFTER specification is needed. Otherwise, the trigger is a BEFORE trigger, and will fire before rowid or other pertinent data is entered into the field.

Comments are preceded by "--". If this script was created in the file "trigger1", you could easily execute this script as follows.

```
$ sqlite3 test.db < trigger1
```

Now try entering a new record as before, and you should see the time in the field timeEnter.

```
sqlite3 test.db  "insert into t1 (data,num) values ('First entry with timeEnter',19);"
```

Doing a select reveals the following data:

```
$ sqlite3 test.db "select * from t1";
1|This is sample data|3|
2|More sample data|6|
3|And a little more|9|
4|First entry with timeEnter|19|2004-10-02 15:12:19
```

If you look at the statement above, the last value has the timeEnter filled in automatically with Coordinated Universal Time - or (UTC). If you want local time, then, use select datetime('now','localtime'). See the note at the end of this section regarding UTC and local time.

For examples that follow the table "exam" and the database "examScript" will be used. The table and trigger are defined below. Just like the trigger above, UTC time will be used.

```
-- ******************************************************************
--  examScript: Script for creating exam table
--   Usage:
--       $ sqlite3 examdatabase < examScript
--
--   Note: The trigger insert_exam_timeEnter
--         updates timeEnter in exam
-- ******************************************************************
-- ******************************************************************
CREATE TABLE exam (ekey      INTEGER PRIMARY KEY,
                   fn        VARCHAR(15),
                   ln        VARCHAR(30),
                   exam      INTEGER,
                   score     DOUBLE,
                   timeEnter DATE);

CREATE TRIGGER insert_exam_timeEnter AFTER  INSERT ON exam
BEGIN
```

```
UPDATE exam SET timeEnter = DATETIME('NOW')
        WHERE rowid = new.rowid;
END;
-- ******************************************************************
-- ******************************************************************
```

After the script file, it can be executed, by redirecting the contents of the script file into the sqlite3 command, followed by the database name. See the example below:

```
$ sqlite3 examdatabase < examScript
$ sqlite3 examdatabase "insert into exam (ln,fn,exam,score)
       values ('Anderson','Bob',1,75)"

$ sqlite3 examdatabase "select * from exam"

1|Bob|Anderson|1|75|2004-10-02 15:25:00
```

And, as a check, the PRIMARY KEY and current UTC time have been updated correctly, as seen from the above example.

Logging All Inserts, Updates, and Deletes

The script below creates the table examlog and three triggers update_examlog, insert_examlog, and delete_examlog to record update, inserts, and deletes made to the exam table. In other words, anytime a change is made to the exam table, the changes will be recorded in the examlog table, including the old value and the new value. By the way if you are familiar with MySQL, the functionality of this log table is similar to MySQL's binlog. See (TIP 2, TIP 24 and TIP 25) if you would like more information on MySQL's log file.

```
-- ******************************************************************
--  examLog: Script for creating log table and related triggers
--   Usage:
--       $ sqlite3 examdatabase < examLOG
--
--
-- ******************************************************************
-- ******************************************************************
CREATE TABLE examlog (lkey INTEGER PRIMARY KEY,
                  ekey INTEGER,
                  ekeyOLD INTEGER,
                  fnNEW   VARCHAR(15),
                  fnOLD   VARCHAR(15),
                  lnNEW   VARCHAR(30),
                  lnOLD   VARCHAR(30),
                  examNEW INTEGER,
                  examOLD INTEGER,
                  scoreNEW DOUBLE,
                  scoreOLD DOUBLE,
                  sqlAction VARCHAR(15),
                  examtimeEnter    DATE,
                  examtimeUpdate   DATE,
                  timeEnter        DATE);

--  Create an update trigger
CREATE TRIGGER update_examlog AFTER UPDATE  ON exam
BEGIN

  INSERT INTO examlog  (ekey,ekeyOLD,fnOLD,fnNEW,lnOLD,
                        lnNEW,examOLD,examNEW,scoreOLD,
                        scoreNEW,sqlAction,examtimeEnter,
                        examtimeUpdate,timeEnter)

          values (new.ekey,old.ekey,old.fn,new.fn,old.ln,
                  new.ln,old.exam, new.exam,old.score,
                  new.score, 'UPDATE',old.timeEnter,
                  DATETIME('NOW'),DATETIME('NOW') );

END;
--
--  Also create an insert trigger
--    NOTE  AFTER keyword ------v
CREATE TRIGGER insert_examlog AFTER INSERT ON exam
BEGIN
INSERT INTO examlog  (ekey,fnNEW,lnNEW,examNEW,scoreNEW,
                      sqlAction,examtimeEnter,timeEnter)

          values (new.ekey,new.fn,new.ln,new.exam,new.score,
                  'INSERT',new.timeEnter,DATETIME('NOW') );

END;

--  Also create a DELETE trigger
CREATE TRIGGER delete_examlog DELETE ON exam
BEGIN

INSERT INTO examlog  (ekey,fnOLD,lnNEW,examOLD,scoreOLD,
                      sqlAction,timeEnter)

          values (old.ekey,old.fn,old.ln,old.exam,old.score,
```

```
                    'DELETE',DATETIME('NOW') );

END;
-- ************************************************************
-- ************************************************************
```

Since the script above has been created in the file examLOG, you can execute the commands in sqlite3 as shown below. Also shown below is a record insert, and an update to test these newly created triggers.

```
$ sqlite3 examdatabase < examLOG

$ sqlite3 examdatabase "insert into exam
                          (ln,fn,exam,score)
                        values
                          ('Anderson','Bob',2,80)"

$ sqlite3 examdatabase "update exam set score=82
                           where
                         ln='Anderson' and fn='Bob' and exam=2"
```

Now, by doing the select statement below, you will see that examlog contains an entry for the insert statement, plus two updates. Although we only did one update on the command line, the trigger "insert_exam_timeEnter" performed an update for the field timeEnter -- this was the trigger defined in "examScript". On the second update we can see that the score has been changed. The trigger is working. Any change made to the table, whether by user interaction or another trigger is recorded in the examlog.

```
$ sqlite3 examdatabase "select * from examlog"

1|2||Bob||Anderson||2||80||INSERT|||2004-10-02 15:33:16
2|2|2|Bob|Bob|Anderson|Anderson|2|2|80|80|UPDATE||2004-10-02 15:33:16|2004-10-02 15:33:16
3|2|2|Bob|Bob|Anderson|Anderson|2|2|82|80|UPDATE|2004-10-02 15:33:16|2004-10-02 15:33:26|2004-10-02 15:33:26
```

Again, pay particular attention to the AFTER keyword. Remember by default triggers are BEFORE, so you must specify AFTER to insure that all new values will be available, if your trigger needs to work with any new values.

UTC and Local time

Note, select DATETIME('NOW') returns UTC or Coordinated Universal Time. But select datetime('now','localtime') returns the current time.

```
sqlite> select datetime('now');
2004-10-18 23:32:34

sqlite> select datetime('now','localtime');
2004-10-18 19:32:46
```

There is an advantage to inserting UTC time like we did with the triggers above, since UTC can easily be converted to local time after UTC has been entered in the table. See the command below. By inserting UTC, you avoid problems when working with multiple databases that may not share the same time zone and or daylight savings time settings. By starting with UTC, you can always obtain the local time.
(Reference: Working with Time)

```
CONVERTING TO LOCAL TIME:

sqlite> select datetime(timeEnter,'localtime') from exam;
```

Other Date and Time Commands

If you look in the sqlite3 source file "./src/date.c", you will see that datetime takes other options. For example, to get the local time, plus 3.5 seconds, plus 10 minutes, you would execute the following command:

```
sqlite> select datetime('now','localtime','+3.5 seconds','+10 minutes');
2004-11-07 15:42:26
```

It is also possible to get the weekday where 0 = Sunday, 1 = Monday, 2 = Tuesday ... 6 = Saturday.

```
  sqlite> select datetime('now','localtime','+3.5 seconds','weekday 2');
  2004-11-09 15:36:51
```

The complete list of options, or modifiers as they are called in this file, are as follows:

```
  NNN days
  NNN hours
  NNN minutes
  NNN.NNNN seconds
  NNN months
  NNN years
  start of month
  start of year
  start of week
```

```
start of day
weekday N
unixepoch
localtime
utc
```

In addition, there is the "strftime" function, which will take a time string, and convert it to the specified format, with the modifications. Here is the format for this function:

```
**    strftime( FORMAT, TIMESTRING, MOD, MOD, ...)
**
** Return a string described by FORMAT.  Conversions as follows:
**
**   %d  day of month
**   %f  ** fractional seconds  SS.SSS
**   %H  hour 00-24
**   %j  day of year 000-366
**   %J  ** Julian day number
**   %m  month 01-12
**   %M  minute 00-59
**   %s  seconds since 1970-01-01
**   %S  seconds 00-59
**   %w  day of week 0-6  sunday==0
**   %W  week of year 00-53
**   %Y  year 0000-9999
```

Below is an example.

```
sqlite> select strftime("%m-%d-%Y %H:%M:%S %s %w %W",'now','localtime');
11-07-2004 16:23:15 1099844595 0 44
```

Simple Everyday Application

Keeping Notes in a Database

This simple bash script (part of the sqlite_examples tarball) allows you to take notes. The notes consist of a line of text followed by an optional category without the additional typing.

```
 "sqlite3 <database> <sql statement>",
```

Instead, it is a simple one letter command.

```
$ n 'Take a look at sqlite3 transactions -
      http://www.sqlite.org/lang.html#transaction' 'sqlite3'
```

The above statement enters the text into a notes table under the category 'sqlite3'. Anytime a second field appears, it is considered the category. To extract records for the day, I enter "n -l", which is similar to "ls -l", to "note list".

With just "n" help is listed on all the commands.

```
$ n
This command is used to list notes in
a database.

n <option>
 -l list all notes
 -t list notes for today
 -c list categories
 -f <search string> search for text
 -e <cmd> execute command and add to notes
 -d delete last entry
```

REFERENCES:

Source Code: Source code for this article.

SQLite Tutorial: If you are hungry for more information, this tutorial covers the ATTACH command, the power of the SIGN function, how to modify the SQLite source code, and how to create C and C++ APIs using SQLite. There is even a Perl example. There are many, many examples and this document is updated weekly.

Over 100 Linux Tips: See TIP 50 on working with the libraries in C and C++. This tip details how to create dynamic and static libraries, as well make use of the -Wl,-R switch in gcc. If you create a C or C++ applications that uses SQLite, consider using dynamic libraries.

Solving Complex SQL Problems: Growing list of examples using the sign function.

www.sqlite.org: Home page for the SQLite project.

Working with Time: This article defines what is meant by UTC, shows you how to use the date command to calculate the date for any time zone including with or without daylight savings time. Plus you will see how to setup and confirm that NTP is working. There is also

a program to calculate sunrise and sunset given longitude and latitude in degrees.

Lemon Parser Generator Tutorial: Tutorial on the parser used with sqlite.

Mike Chirico, a father of triplets (all girls) lives outside of Philadelphia, PA, USA. He has worked with Linux since 1996, has a Masters in Computer Science and Mathematics from Villanova University, and has worked in computer-related jobs from Wall Street to the University of Pennsylvania. His hero is Paul Erdos, a brilliant number theorist who was known for his open collaboration with others. Mike's notes page is souptonuts.

Published in Issue 109 of Linux Gazette, December 2004