SANS DFIR SQLite Reference Guide

Lee Crognale, Sarah Edwards and Heather Mahalik — sans.org/for518 | sans.org/for585

Purpose

This guide is a supplement to the SANS **FOR518: Mac and iOS Forensic Analysis and Incident Response** and **FOR585: Smartphone Forensic Analysis In-Depth** courses, and enhances concepts covered in other courses. It covers some of the core methods to extracting data from SQLite databases. Definitions, sample queries, and SQLite terminology will help you conduct manual extractions from databases of interest found on Macs, smartphones, and PCs. *Commercial tools will only get you so far. Manual recovery and extracting contents **is** what really matters!*

How To Use This Document

SQLite databases are used to store a vast amount of data recovered from digital media. This handout cannot begin to scratch the surface of the great and powerful things you can do with SQL queries. Use this document as a "memory jog" for some of the queries, tools available and examples for normalizing data formats recovered from databases.

We think you will find this reference as a place to start or simply help remember which operators and queries will assist you in your investigation. The tools listed provide support for CLI and GUI.

SQLite References & Tutorials

- **Official SQLite Documentation:** https://sqlite.org
- **Great Tutorials:**
  - https://www.tutorialspoint.com/sqlite/
  - http://www.sqlitetutorial.net/
  - http://zetcode.com/db/sqlite/
  - http://sandersonforensics.com/forum/content.php?2 75-How-NOT-to-examine-SQLite-WAL-files
- **FOR518: Mac and iOS Forensic Analysis and Incident Response** – FOR518.com
- **FOR585: Smartphone Forensic Analysis In-Depth** – FOR585.com

Table Joins

Taking data from two (or more!) tables that have a column in common and joining them into one table. Identify tables of interest that contain unique values.

LEFT JOIN – Resulting rows are returned from the **LEFT** table even if there are no matches in the right. Using the **LEFT JOIN** produced all the text messages including those with and without attachments.

```
SELECT
ZVIBERMESSAGE.ZTEXT AS "Message Text",
ZATTACHMENT.ZNAME AS "Attachment Filename",
datetime(ZVIBERMESSAGE.ZDATE+978307200,'unixepoch',
'localtime') AS "Message Date",
ZVIBERMESSAGE.ZSTATE AS "Message Direction/State"
FROM
ZVIBERMESSAGE
LEFT JOIN ZATTACHMENT on
ZATTACHMENT.Z_PK=ZVIBERMESSAGE.ZATTACHMENT
```

INNER JOIN – Resulting rows are returned when both items are a match. Using the **INNER JOIN** (also achieved by typing "**JOIN**" in the query) returned just the messages that included attachments.

Useful Stuff

Column Renaming:

```
A_TABLE.ZAWKWARDCOLUMNNAME AS "Chat Messages"
```

Counting:

```
SELECT COUNT(*) FROM A_TABLE;
```

Aggregating with GROUP BY and COUNT (Count chat messages per contact):

```
SELECT MESSAGES,COUNT(*) FROM CHAT GROUP BY CONTACT;
```

Sorting with ORDER BY:

```
SELECT * FROM CHAT ORDER BY A_TIMESTAMP ASC
```

ASC = Ascending
DESC = Descending

Searching with WHERE and LIKE:

```
SELECT CONTACT, MESSAGE FROM CHAT WHERE CONTACT LIKE
'%Hank%'
```

Timestamp Conversion

Timestamps are stored in the databases as one of several numerical representations. *(Timestamps are assumed to be stored in UTC, you may need to verify this.)*

UNIX Epoch (10-digit number - number of seconds since 01/01/1970 00:00:00):

- `SELECT datetime(TS_COLUMN,'unixepoch')`
  Or in **local time** as suggested by the device settings (this can be done for all the following timestamps):
- `SELECT datetime(TS_COLUMN,'unixepoch', 'localtime')`

UNIX Epoch MILLISECONDS (13-digit number - number of milliseconds since 01/01/1970 00:00:00):

- SELECT datetime(TS_COLUMN/1000,'unixepoch');

Mac Absolute time, number of seconds since 01/01/2001 00:00:00. To correctly convert this timestamp, first, add the number of seconds since UNIXEPOCH time to Mac Absolute Time (978307200), then convert.

- `SELECT datetime(TS_COLUMN + 978307200, 'unixepoch');`

Chrome time accounts for time accurate to the MICROSECOND, which requires dividing the number by 1,000,000:

- `SELECT datetime(TS_COLUMN/1000000 + (strftime('%s','1601-01- 01')),'UNIXEPOCH');`

`sqlite3` CLI Options

Use the command line version of the `sqlite3` program (sqlite.org/cli.html) either in a SQLite shell, or just query via CLI:

```
$ sqlite3 <db_file>
$ sqlite3 <db_file> 'select * from a_table'
```

- `.help` – Provides a list of these 'dot-commands'
- `.tables` – Show the table names in the database
- `.headers on` – Show the column names in the output
- `.mode column` – Show left-aligned columns
- `.mode tabs` – Show tab separated columns
- `.output <filename>` – Send output to file
- `.dump` – Dump database contents (use with .output)
- `.quit` – Quit sqlite3 shell

Is the Database Using WAL or Journaling

The SQLite header for every database will contain offsets enabling you to differentiate if a journal or WAL is being used to support the database.

- File Offset 18 (1 byte) = x01 = Journaling
- File Offset 19 (1 byte) = x01 = Journaling

OR

- File Offset 18 (1 byte) = x02 = WAL
- File Offset 19 (1 byte) = x02 = WAL

SQLite Deletion

Examine the table to determine if data is moved to the free pages or a Boolean value is entered to mark the data deleted.

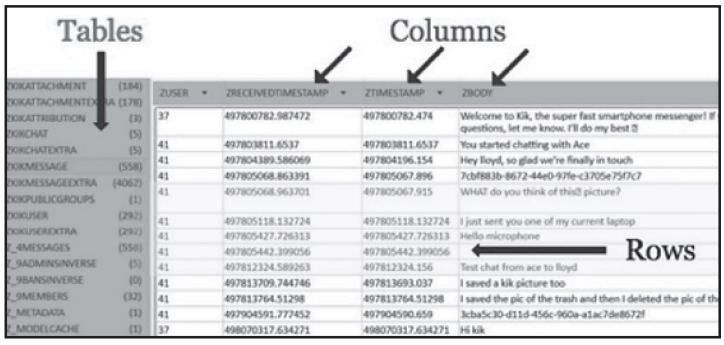
Use a SQLite Editor to examine the free pages in a Hex view to carve for deleted artifacts.

Use scripts and tools available to conduct a cursory scan of the free pages for deleted SQLite entries.

SQLite Deleted Records Parser – Python script and GUI to parse deleted data from SQLite databases – https://github.com/mdegrazia/SQLite-Deleted-Records-Parser/tree/master/Old_Versions/sqlparse_v1_1

SQLite Database Basics

SQLite databases are a self-contained database stored as a file system file (but may have a few supporting files that will also be needed for analysis!) Files have the magic number "`SQLite format 3`". SQLite files correspond to a database that contains tables. **Tables** contain **rows** of data with corresponding columns that describe the data in the row.



Some temporary files may also be created, including **Journal files** and **Write Ahead Logs**. **Journal files** store original data before a transaction change so the database can be restored to a known state if an error occurs. They are created by default.

Write Ahead Logs (WAL) contain new data changes, leaving original database untouched. After a set number of page changes, the WAL is used to update the actual database. Write ahead logs are optional. Journal files – stores original data before a transaction change so the database can be restored to a known state if an error occurs. (created by default)

SQLite Analysis Tools

- **DB Browser for SQLite** – Free GUI utility available for Mac/Windows/*nix – sqlitebrowser.org
- **sqlite3** – Free CLI utility available for (or native to) Mac/iOS/Android/*nix/Windows - sqlite.org/cli.html
- **SQLite Spy** – Free GUI utility for Windows – https://www.yunqa.de/delphi/products/sqlitespy/index
- **Sanderson Forensic Toolkit for SQLite** – Commercial GUI utility for Windows – sandersonforensics.com
- **SQLite Miner** – Free script to extract blobs from databases – https://github.com/threeplanetssoftware/sqlite_miner
- **Commercial tools and Browser Plug-ins** – A SQLite editor is available in most forensic tool suites

Basic Analysis Query Structure

Get everything from a single table:

```
SELECT * FROM A_TABLE;
```

Get two columns from a single table:

```
SELECT COLUMN_A, COLUMN_B FROM A_TABLE;
```

Data Types

- **NULL** – NULL Value
- **INTEGER** – Signed Integer
- **REAL** – Floating Point Number
- **TEXT** – Text String (UTF-8, UTF-16BE or UTF-16LE)
- **BLOB** – (**B**inary **L**arge **OB**jects) to store large chunks of data. This data may be a picture, video, audio, or archive (Gzip) files. This data is not defined in the database, it may contain anything an app developer desires. This data is often overlooked but may contain forensic nuggets of gold!

Operators

Arithmetic:

- Addition [+]
- Subtraction [-]
- Multiplication [*]
- Division [/]
- Modulus [%]

Comparison:

- Equal [==] or [=], Not Equal [!=] or [<>]
- Greater Than [>] or Greater Than or Equal [>=]
- Less Than [<] or Less Than or Equal [<=]

Logic:

- **IS/IS NOT** – Equal/Not Equal
- **IN/NOT IN** – Is value in (or not) a list of values?
- **LIKE (Case Insensitive)/GLOB (Case Sensitive)** – Is value like this other value? Uses wildcards.
- **AND/OR** – Use with WHERE clause to create complex logic.
- **BETWEEN** – Look for values between two values.