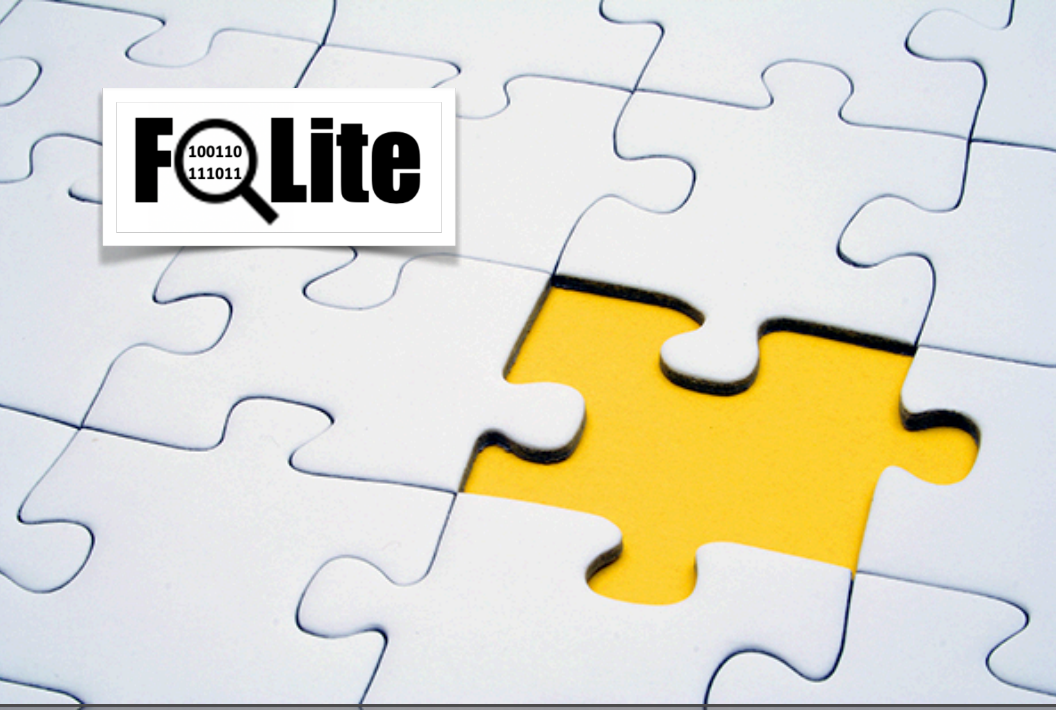


SQLITE FORENSICS WITH

FQLite

The Official User Guide

SQLite Forensics With

FQLite

The Official User Guide

Author: Dirk Pawlaszczyk
Title: *SQLite Forensics with FQLite - The Official User Guide*.
Independently published
First Edition, 2025

Preface

Welcome to the first Release of the FQLite User's Guide. This document provides the necessary information to work with FQLite effectively. It contains detailed information about the following:

- Overview and reference information
- How to navigate through the FQLite Application windows
- How to recover, analyse and export findings.

This preface explains how this user's guide is organised and introduces other sources of information that can help you. This guide is the primary source of information about the FQLite Applications User Interface. It contains overviews as well as task and reference information. This guide includes the following chapters:

- Chapter 1, "Getting Started", introduces you to FQLite and to some basic features. It provides instructions on opening a database, navigating the menu, obtaining help, utilising keyboard shortcuts, and exiting.

- Chapter 2, "FQLite Data Recovery Basics", introduces you to the user interface and provides an overview of the essential concepts and functions required to get started. Specifically, this chapter describes the anatomy of the user interface and guides navigation.

- Chapter 3, “Recovery and Export of Data”, discusses advanced features that help you enter and query data from SQLite databases.
- Chapter 4, „Advanced FQLite features“, gives a short overview regarding the handling of WAL-Archive files as well as Rolback Journals.

Thank you for using FQLite and this user’s guide!

Content

CHAPTER 1
GETTING STARTED

The Forensic SQLite Data Recovery Tool (FQLite) is a tool that helps in recovering data from SQLite database files. FQLite is for users and forensic investigators who want to analyse an SQLite database in a forensically sound manner. Hence, it is a useful tool for digital forensic investigators who need to analyse SQLite files during an investigation. The program also supports the user in finding and restoring deleted records. It therefore examines the database for entries marked as deleted. Those entries can be recovered and displayed.

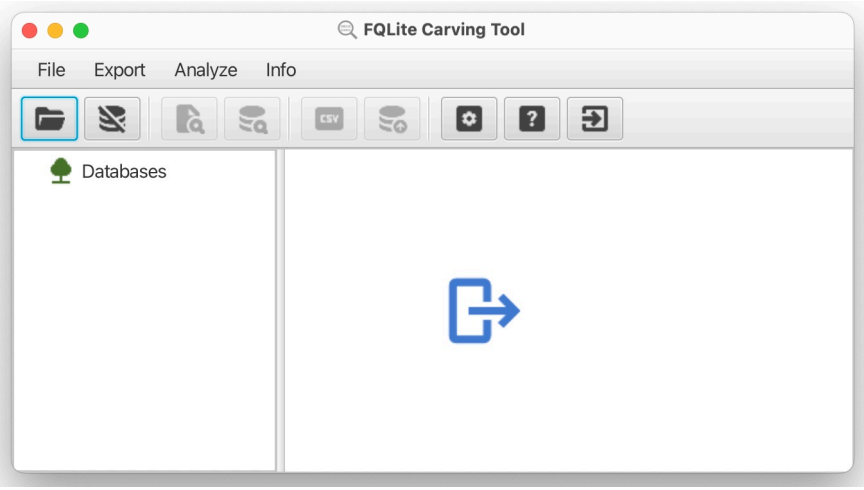


Figure 1: The FQLite Main-Window

In this user guide, we will provide a detailed overview of *FQLite* and its usage. FQLite can be used to recover deleted data, extract data from a specific table, or analyse an entire SQLite database.

Getting started - The Main Window

To start *FQLite*, run the binary file by double-clicking it. After starting FQLite, the user is first presented with the main window (see *Figure 1*). Above the actual view area, a toolbar is displayed by default. The menu or the toolbar can be used to import new databases or to start an export of selected tables or databases. Opening a database with FQLite automatically starts the analysis process. Since FQLite is a forensic tool, the evidence object is never modified. This distinguishes the program from a conventional viewer. Furthermore, when analysing a new database, the program always automatically searches for possibly deleted content.

Import a Database

FQLite was developed for analysing SQLite databases. Besides the actual database file, this also includes the analysis of rollback journals and write-ahead log files. Whenever a database is opened, the system automatically searches for the mentioned companion files with the file extensions *-wal* or *-journal*. If such a file is found and it is not empty, then it will also be loaded automatically. FQLite thus supports the analysis of 3 different data sources at the

moment:

- SQLite database files (normally ending with file extensions like .db or .sqlite)
- Rollback Journal files (extension -journal)
- Write-Ahead Logs (extension -WAL)

To load a database, it can either be selected via the 'Open Database...' dialogue or dragged and dropped from the file explorer.

After opening a database file and possible transaction backup files, you can access the individual files and the data stored in them via the tree view (see *Figure 2*). A database and its tables are represented as nodes of the tree. The database file is defined as an inner node. Tables are leaves.

Tables that are currently storing records are marked with a green icon in the tree view.

Empty tables are marked with a small red dot. The table is part of the database schema, but it is not populated with data rows.

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. This is similar to an index found at the back of a book

An empty index table is marked with a red dot.

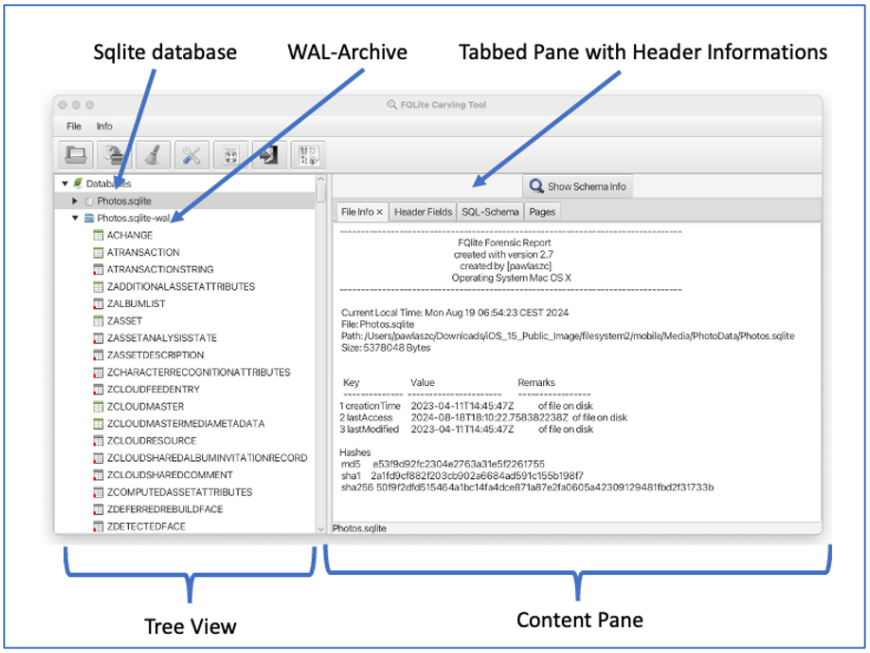


Figure 2: Main Window

As soon as you select a database or a table in the tree view, the view of the Content Pane is automatically updated.

General information about the database, such as storage location and hash sums, can be accessed initially via the File Info tabbed pane.

Various functions can be carried out via the toolbar: (1) opening or closing databases, (2) calling up the settings dialogue box, (3) exporting individual tables or an entire

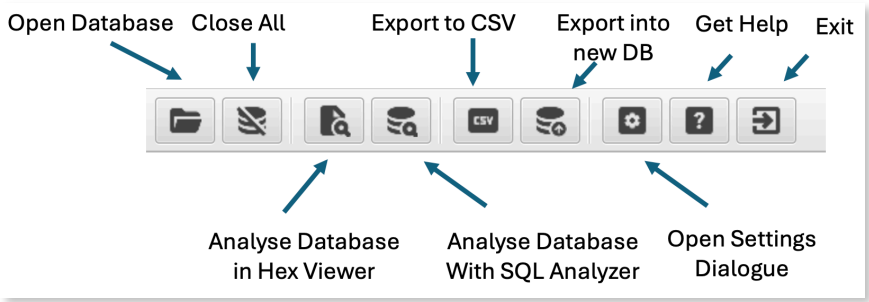


Figure 3: The Application Toolbar

database, (4) calling up the user manual and exiting the application, (5).

The export button can be used to initiate the export of an individual database table or the entire database, depending on the selection in the tree view.

A special feature of FQLite is the ability to read and evaluate several databases in parallel. Once you have completed an investigation, you can close it by clicking on the “Close all...” button. This will close all currently open database data at once.

In particular, the settings button can be used to configure the program's behaviour when exporting restored data. For example, you can specify which separator is to be used when generating the comma-separated value file. You can also specify whether or not the table header should be exported in the export.

With version 3.1 of FQLite, a new component, the *SQL-Analyser*, has been introduced. This small program allows the use of standard SQL commands and thus improves analysis options. The power of SQL is now available to the user.

Clicking this button takes the user directly to this manual.

This button can be used to exit the application. Attention: The application ended directly and without prompting. All open database files are closed immediately.

FQLite allows each hit within the database to be visually confirmed again on a binary level. A built-in hex viewer is supported for this purpose. Simply click on this button to examine the database file and the archive files in hex mode more closely.

Of course, all of the functions described can also be accessed via the menu bar. In addition to the functions already discussed, the Items menu contains other functions that are not listed in the toolbar. This is how log messages are generated during the import process. These can be viewed at any time via the "*Info* | *View Log*..." menu item. The "*Info* | „*Font*..." menu item can be used to adjust the font settings

of the application. This allows the font size and font type to be adapted to the user's needs. Please note that changes to the font are only applied after the application is restarted. Finally, the "*About...*" entry provides background information on the version number and software license. Alternatively, the functions described so far can also be accessed via keyboard shortcuts. The following table describes the most frequently used Shortcuts in FQLite.

Table 1: Frequently used shortcuts

To do this	Press
Open a database.	Ctrl+O
Export database to CSV.	Ctrl+X
Close all.	Ctrl+D
Exit.	Alt+F4

We have now familiarised ourselves with the toolbar and menu functions. In the next chapter, we will take a closer look at the actual analysis functionalities of *FQLite*.

CHAPTER 2

FQLITE DATA RECOVERY BASICS

FQLite offers a range of views for examining the database. This chapter, therefore, focuses on the respective tabs and their function. There are roughly two categories of tabs or views: 1) Global views and lists, 2) Table-specific views. Let's first take a look at the global information and where it can be found within the interface. In Chapter 3, we will then turn our attention to the individual tables and how they can be operated.

Table 2: Core Requirements for Recovery Tools

Required Feature	Description
SFT-CR-01	The tool shall not modify the files being analyzed.
SFT-CR-02.	The tool shall report the database configuration parameters pertinent to data recovery.
SFT-CR-03.	The tool shall report the schema structure of the database tables.

The National Institute of Standards and Technology (NIST) has a series of core requirements that must be fulfilled by every SQLite recovery tool [NIST2021][1]. *FQLite* fulfils all the

[1] SQLite Data Recovery Specification, Test Assertions, and Test Cases (2021)

core functions mentioned. To provide the required information on the structure and configuration settings of the database, *FQLite* offers a series of tabs, which we will examine in more detail below.

The „File Info“ Tab

According to the NIST guidelines, the analyser tool should not alter the database under any circumstances during the analysis. Hash values provide proof of this. FQLite supports three different hash types in the *File Info* tab: MD5, SHA-1, and SHA-256. The Modification, Access, and Change timestamps are also displayed.

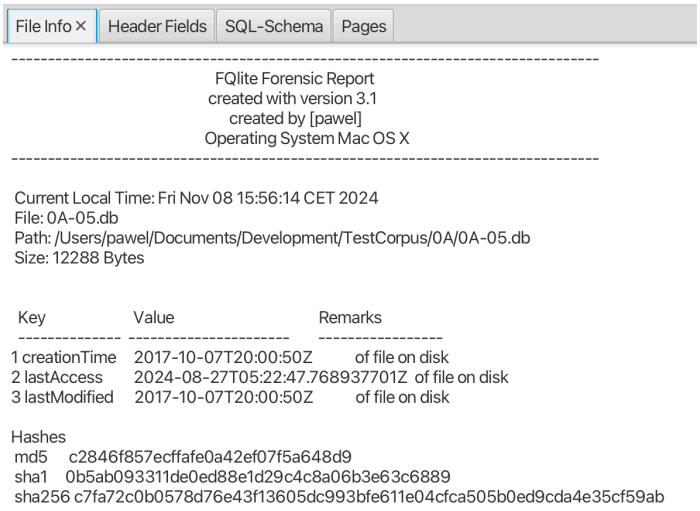
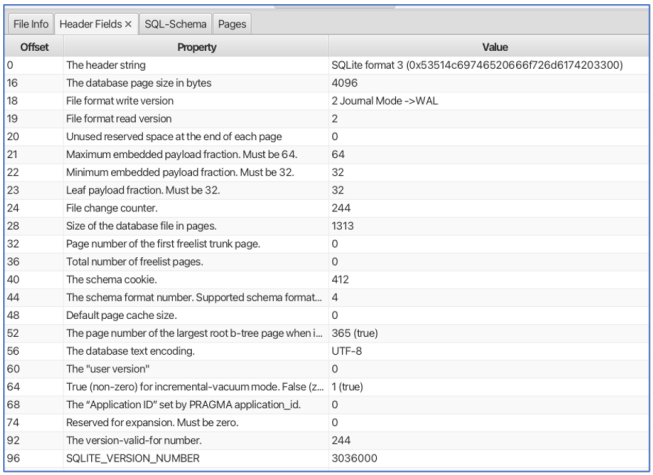


Figure 4: The „File Info“-Tab

In addition to hash values, MAC timestamps (Modification, Access, Change) can be viewed. These are the three

timestamps in the file systems that log file operations. They show when a file was last modified (Modification), accessed (Access) or when its metadata and content were last changed (Change/Creation).



File Info | Header Fields × | SQL-Schema | Pages

Offset	Property	Value
0	The header string	SQLite format 3 (0x53514c69746520666f726d6174203300)
16	The database page size in bytes	4096
18	File format write version	2 Journal Mode ->WAL
19	File format read version	2
20	Unused reserved space at the end of each page	0
21	Maximum embedded payload fraction. Must be 64.	64
22	Minimum embedded payload fraction. Must be 32.	32
23	Leaf payload fraction. Must be 32.	32
24	File change counter.	244
28	Size of the database file in pages.	1313
32	Page number of the first freelist trunk page.	0
36	Total number of freelist pages.	0
40	The schema cookie.	412
44	The schema format number. Supported schema format...	4
48	Default page cache size.	0
52	The page number of the largest root b-tree page when i...	365 (true)
56	The database text encoding.	UTF-8
60	The "user version"	0
64	True (non-zero) for incremental-vacuum mode. False (z...	1 (true)
68	The "Application ID" set by PRAGMA application_id.	0
74	Reserved for expansion. Must be zero.	0
92	The version-valid-for number.	244
96	SQLITE_VERSION_NUMBER	3036000

Figure 5: The Header Fields Information tab

The „Header Fields" Tab

The header of an SQLite database file is a critical component of the database structure, containing essential metadata that defines the database format and guides the SQLite engine's interaction with the database (*Figure 5*). The header is located at the very beginning of the database file and consists of 100 bytes. The SQLite database file header is meticulously organised to store essential metadata, providing information on the file format, page size, encoding, and other parameters *(Figure 6)*. This

structure is crucial for SQLite to efficiently manage and access the database, ensuring compatibility and reliability across different versions and environments. The 23 data fields of the database header are provided via the "*Header Fields*" tab *(Figure 4)*.

PRAGMA settings, such as the *version number or* the *number of free pages* in the database, can be viewed in this view. The header fields and their meanings are shown in *Figure 5*. The tool further reports the SQLite Page Size (in bytes), SQLite Journal Mode (read version), the number of pages in the database, and database text encoding.

The „SQL-Schema" Tab

A database schema defines how data is organised in a relational database. This includes logical constraints such as table names, fields, data types and relationships

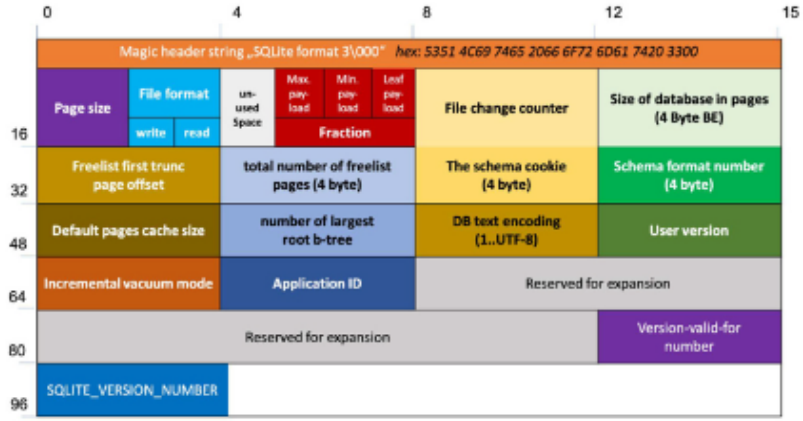
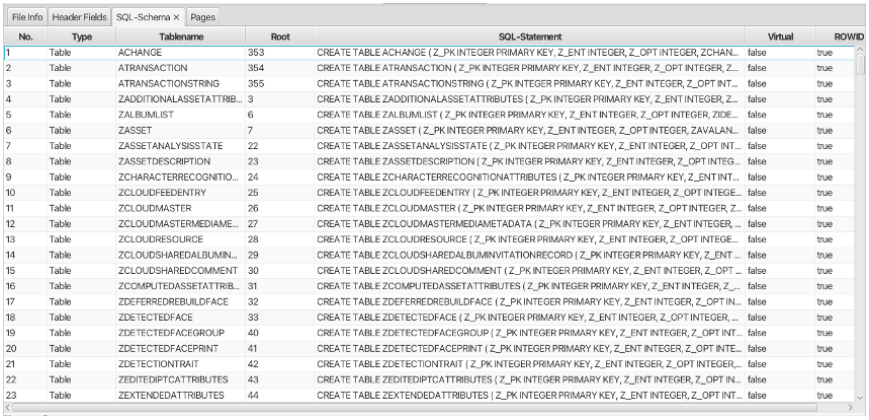


Figure 6: The header fields of a SQLite database file

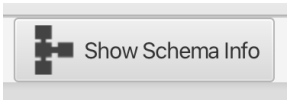


File Info | Header Fields | SQL-Schema × | Pages

No.	Type	Tablename	Root	SQL-Statement	Virtual	ROWID
1	Table	ACHANGE	353	CREATE TABLE ACHANGE (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INTEGER, ZCHAN...	false	true
2	Table	ATRANSACTION	354	CREATE TABLE ATRANSACTION (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INTEGER, Z...	false	true
3	Table	ATRANSACTIONSTRING	355	CREATE TABLE ATRANSACTIONSTRING (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INT...	false	true
4	Table	ZADDITIONALASSETATTRIB...	3	CREATE TABLE ZADDITIONALASSETATTRIBUTES (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z...	false	true
5	Table	ZALBUMLIST	6	CREATE TABLE ZALBUMLIST (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INTEGER, ZIDE...	false	true
6	Table	ZASSET	7	CREATE TABLE ZASSET (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INTEGER, ZAVALAN...	false	true
7	Table	ZASSETANALYSISSTATE	22	CREATE TABLE ZASSETANALYSISSTATE (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INT...	false	true
8	Table	ZASSETDESCRIPTION	23	CREATE TABLE ZASSETDESCRIPTION (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INTEG...	false	true
9	Table	ZCHARACTERRECOGNITIO...	24	CREATE TABLE ZCHARACTERRECOGNITIONATTRIBUTES (Z_PK INTEGER PRIMARY KEY, Z_ENT INT...	false	true
10	Table	ZCLOUDFEEDENTRY	25	CREATE TABLE ZCLOUDFEEDENTRY (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INTEGE...	false	true
11	Table	ZCLOUDMASTER	26	CREATE TABLE ZCLOUDMASTER (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INTEGER, Z...	false	true
12	Table	ZCLOUDMASTERMEDIAME...	27	CREATE TABLE ZCLOUDMASTERMEDIAMETADATA (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, ...	false	true
13	Table	ZCLOUDRESOURCE	28	CREATE TABLE ZCLOUDRESOURCE (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INTEGE...	false	true
14	Table	ZCLOUDSHAREDALBUMIN...	29	CREATE TABLE ZCLOUDSHAREDALBUMINVITATIONRECORD (Z_PK INTEGER PRIMARY KEY, Z_ENT ...	false	true
15	Table	ZCLOUDSHAREDCOMMENT	30	CREATE TABLE ZCLOUDSHAREDCOMMENT (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT ...	false	true
16	Table	ZCOMPUTEDASSETATTRIB...	31	CREATE TABLE ZCOMPUTEDASSETATTRIBUTES (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_...	false	true
17	Table	ZDEFERREDREBUILDFACE	32	CREATE TABLE ZDEFERREDREBUILDFACE (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT IN...	false	true
18	Table	ZDETECTEDFACE	33	CREATE TABLE ZDETECTEDFACE (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INTEGER, ...	false	true
19	Table	ZDETECTEDFACEGROUP	40	CREATE TABLE ZDETECTEDFACEGROUP (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INT...	false	true
20	Table	ZDETECTEDFACEPRINT	41	CREATE TABLE ZDETECTEDFACEPRINT (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INTE...	false	true
21	Table	ZDETECTIONTRAIT	42	CREATE TABLE ZDETECTIONTRAIT (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INTEGER,...	false	true
22	Table	ZEDITEDIPTCATTRIBUTES	43	CREATE TABLE ZEDITEDIPTCATTRIBUTES (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT IN...	false	true
23	Table	ZEXTENDEDATTRIBUTES	44	CREATE TABLE ZEXTENDEDATTRIBUTES (Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INT...	false	true

Figure 7: The SQL-Schema Tab (example)

between these entities. FQLite provides a list view that displays the schema structure of the database tables in the SQL schema.



Between these entities. FQLite currently supports the display of tables and indexes. Triggers are not evaluated. There is an additional button above the tabs. If you select this, an extended view opens automatically in the system's standard browser. This view provides detailed information on each table, including column names, the SQL types of each column, and possible constraints. SQLite employs a dynamic type system, allowing you to store any value in a column. SQLite offers five storage classes: INTEGER, REAL, TEXT, BLOB, and NULL. Therefore, serial types of each column are displayed (see *Figure 8*).

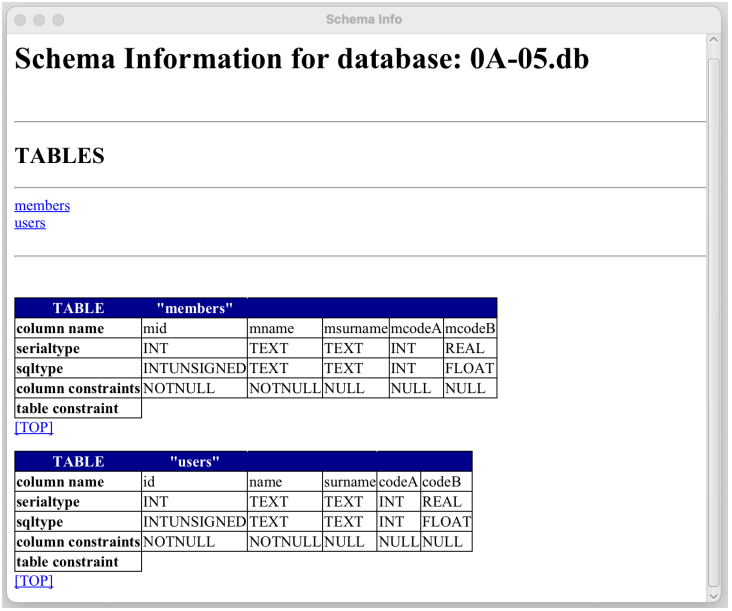


Figure 8: The Schema Info Detail View

The „Pages" Tab

Finally, the program offers a tab that displays all database pages in a list form (Figure 9). The main database file consists of one or more pages. The size of a page is a power of two between 512 and 65536 inclusive.

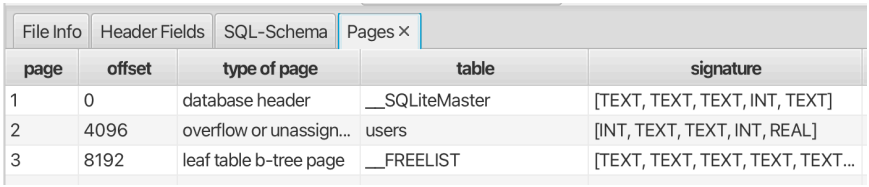


Figure 9: The „Pages“-Tab

All pages within the same database are the same size. At any point in time, every page in the main database has a single use. To be more precise, each database page is linked to exactly one table. SQLite distinguishes between different page types:

- A table B-tree interior page
- A table B-tree leaf page
- An index B-tree interior page
- An index B-tree leaf page
- A freelist page (trunk or leaf)
- A payload overflow page
- A pointer map page
- The lock-byte page

The function of the respective page type will not be discussed here. However, if you are interested, you can read about this on the official SQLite website[2].

The Settings-Dialog

Basic configuration settings can be made via the settings dialogue. This window can be accessed via the File| Settings... menu or via the toolbar of the application window. Basic settings can be configured in this window (Figure 10). For example, you can specify how binary values should be handled when exporting restored data. For CSV-based data export, you can choose between a comma, tab or semicolon as the column separator.

---

[2] https://www.sqlite.org/fileformat.html

Figure 10: The Settings Dialog

For troubleshooting and fixing issues when importing unknown databases, it is sometimes important to collect log messages. However, since logging entries can harm the overall performance of the system and can significantly increase import times, for example, users can determine the accuracy of log messages themselves in FQLite. The Tool offers a set of standard logging levels that can be used

to control logging output. The levels in descending order are:

- SEVERE (highest value)
- WARNING
- INFO
- FINE (lowest value)

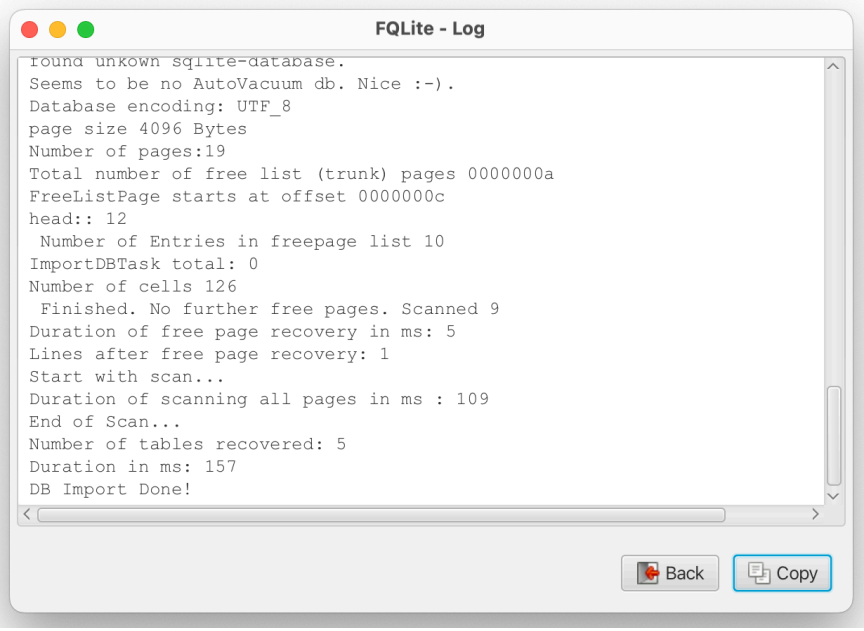


Figure 11: The Log-Window

The Log-View

Log messages are indispensable for troubleshooting. In FQLite, all log messages are written to a rolling log file (*Figure 11*). The log file is stored by default in the hidden .fqlite folder in the user folder (*Figure 12*). The text

Figure 12: the .fqlite folder with the log-files

file can be opened and viewed at any time using a standard text editor.

In addition, the programme offers the option of viewing the most recent log entries directly in the application via a log viewer window. This window can be accessed via the **Info|View Log...** menu item. The log window allows you to select parts of the log file or the entire file and copy it to the clipboard for further processing. So if you encounter a problem when importing a database into FQLite, it is always advisable to first take a look at the log messages in this window.

Selecting a different Display Font

By default, FQLite uses the 'System' font as its default font. The standard font size is 11 points. If you wish to deviate from these settings, for example, because you prefer larger letters, you can do so using a special font dialogue box (*Figure 13*). This window can be accessed via the **Info|Font...** menu item in the application menu. If you have changed the font settings, you must restart FQLite. Only then will the changes usually take effect. The settings for the font used for display are stored in the fqlite.conf file

by default. Like the log file, this file is located in the .fqlite folder in your user directory.

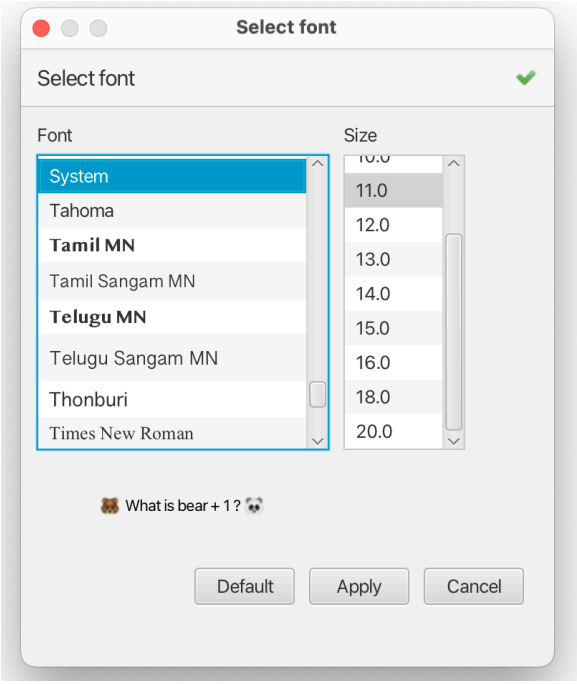


Figure 13: The font selection dialogue

Getting Help

There are several ways to get help with the application. First, you can search for information on the project website[3]. Here you will also find additional resources such as sample databases and short videos. The website also provides links to other documents related to the FQlite

[3] https://www.staff.hs-mittweida.de/~pawlaszc/fqlite/

application. This user guide can be accessed at any time via the menu **Info|Help**. The documentation is displayed in a separate window via FQLite's internal PDF viewer.

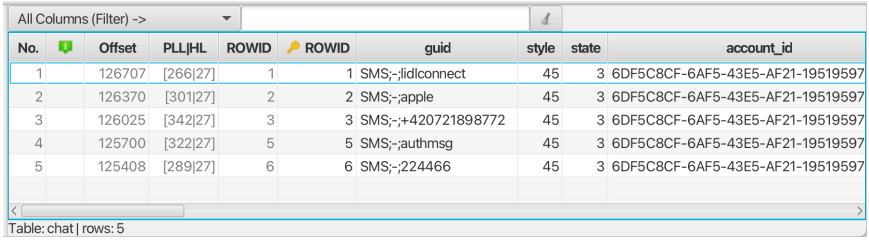
CHAPTER 3

RECOVERY AND EXPORT OF DATA

In this chapter, we will look at data analysis and the recovery of data rows. The central element of the data analysis is the table view. In addition to the regular data records, you will also find deleted data records here if FQLite found them. We also learn how to search the contents of the tables for specific search terms. It also shows how to check the locations of data records using the built-in hex editor. It is shown how to get detailed metadata for all recovered data elements. The handling of timestamps is also discussed.

Table View

The table view represents the central component of FQLite. It displays all data records found in a table. In addition to the actual data, additional columns are shown (*Figure 14*).


All Columns (Filter) ->

No.	Offset	PLL|HL	ROWID	ROWID	guid	style	state	account_id
1	126707	[266|27]	1	1	SMS;-;lidlconnect	45	3	6DF5C8CF-6AF5-43E5-AF21-19519597
2	126370	[301|27]	2	2	SMS;-;apple	45	3	6DF5C8CF-6AF5-43E5-AF21-19519597
3	126025	[342|27]	3	3	SMS;-;+420721898772	45	3	6DF5C8CF-6AF5-43E5-AF21-19519597
4	125700	[322|27]	5	5	SMS;-;authmsg	45	3	6DF5C8CF-6AF5-43E5-AF21-19519597
5	125408	[289|27]	6	6	SMS;-;224466	45	3	6DF5C8CF-6AF5-43E5-AF21-19519597

Table: chat | rows: 5

Figure 14: The Table View

Each table in FQLite starts with the same columns:

- **No.** - The row number (assigned by FQLite).
- **Status** - Status of the data record (regular or deleted).
- **Offset.** Start of the data record. This is a relative address. The first byte of the database file has the offset value 0.
- **Payload Length, Header Length (PLL|HL).** Payload describes a varint, which is the total number of bytes of payload, including any overflow. The second part shows the length of the data record header.
- **ROWID**. Rowid tables are distinguished by the fact that they all have a unique, non-NULL, signed 64-bit integer rowid that is used as the access key for the data in the underlying B-tree storage engine.

These technical columns are immediately followed by the columns with the actual data content. When you move the mouse pointer over a table cell, a tooltip displays the current data type of the table column. The primary key columns of the table are marked with a small key symbol next to the column name. The number of records stored in the table is displayed at the bottom of the screen.

Hex-Viewer

FQLite enables you to inspect every match with a Hex Viewer (*Figure 15*). In addition to the actual data record, the offset – i.e., the position where the data record was found

in the database – is also displayed for each data record. A mouse click on the offset field for a concrete row is sufficient to jump to the hex view. In the hex viewer that then opens, the relevant line of the dataset is highlighted in colour. A record contains a header and a body, in that order. The header begins with a single *variable-length integer (varint),* which determines the total number of bytes in the header. The *varint* value is the size of the header in bytes, including

the size varint itself.

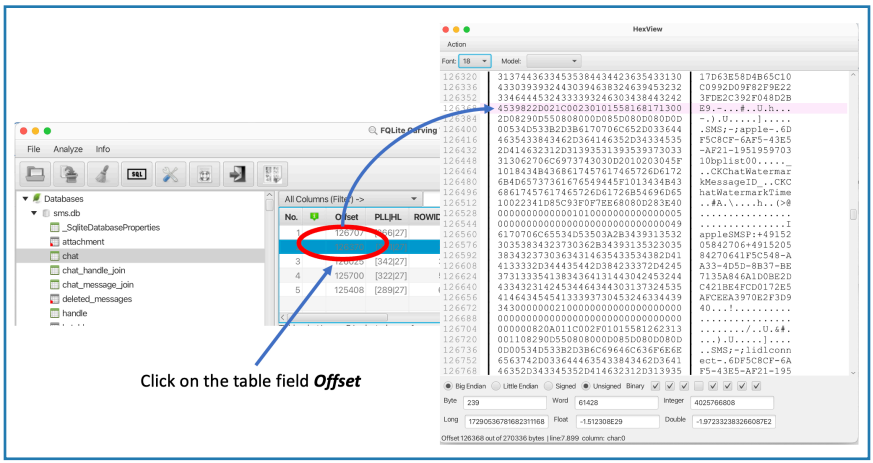


Figure 15: Popup Hex-Viewer for a given Offset

View Deleted Content

As already mentioned, FQLite is capable of restoring deleted records in whole or in part. This is the application's greatest strength. Deleted records that could be found and recovered can be easily identified in the table view by the Status column. For records prefixed with the letter 'D', this

means that they were found within a database page that also contains valid records (*Figure 16*). For a data column prefixed with the letter 'F', it applies that this data column originates from the free list. In this case, the database page in question has already been released for overwriting.
The recovery of deleted data starts automatically in FQLite. No additional steps are required for this. You can sort the column elements according to their natural order by double-clicking on the column name.

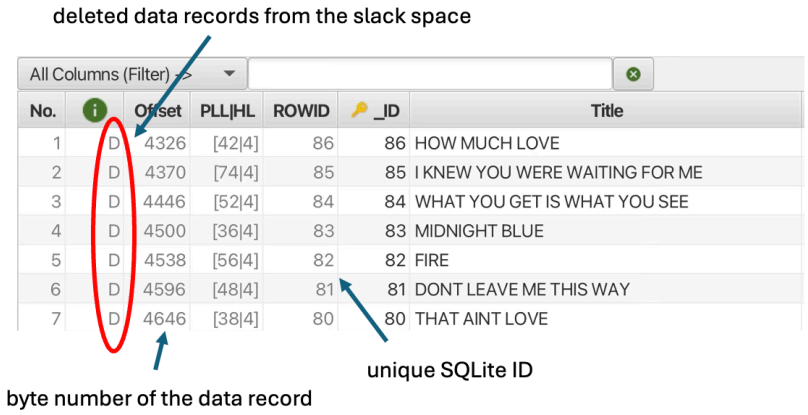


Figure 16: Representation of recovered data records

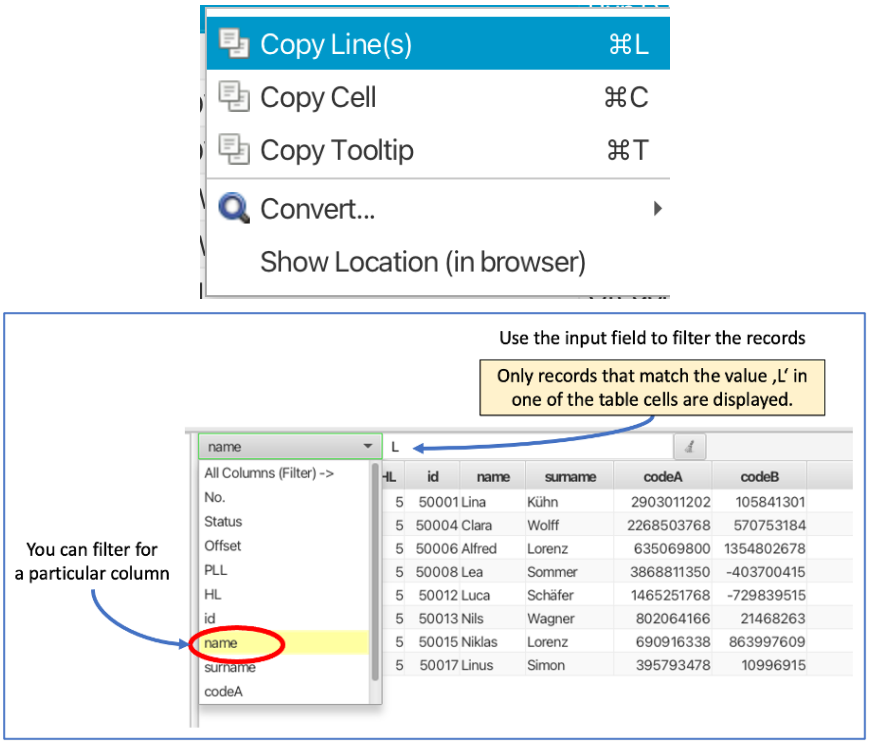
In SQLite, the "payload" of a cell is defined to be the arbitrary-length section of the cell, including the header. The specification includes the actual data length plus some meta information. The latter is now placed in front of the actual data in the form of header bytes. These describe, for example, the specific data type or the length of the

individual table entries. The specific length specifications can be found in the **[PLL|HL]** column.

Filter Content

Records in the table can be easily filtered using the filter input field located above the table view (*Figure 17*). Searching by means of regular expressions is not supported at the moment. However, this feature is planned for one of the next versions. A filter can be applied to the entire table or to an individual column.

Figure 17: The filter function



The filter is not case sensitive. Clicking on the cross symbol to the right of the filter resets the selection criteria. The currently selected cell is copied to the clipboard. Alternatively, you can also move the entire line to the clipboard. Additionally, the tooltip text can be copied.

Inspect the Freelist

A freelist in SQLite is essentially a recycling system for database storage space. Here's how it works:

When data is deleted from an SQLite database, the space it occupied is not immediately released to the operating system. Instead, SQLite tracks these empty database pages in a freelist, which serves as an internal record of available storage for future use. FQLite automatically searches for freelist entries when opening a new database. The entries found can then be viewed in a special table called _FREELIST (*Figure 18*). To be clear: this table is not part of the original database schema. It only serves to make the recovered entries visible.

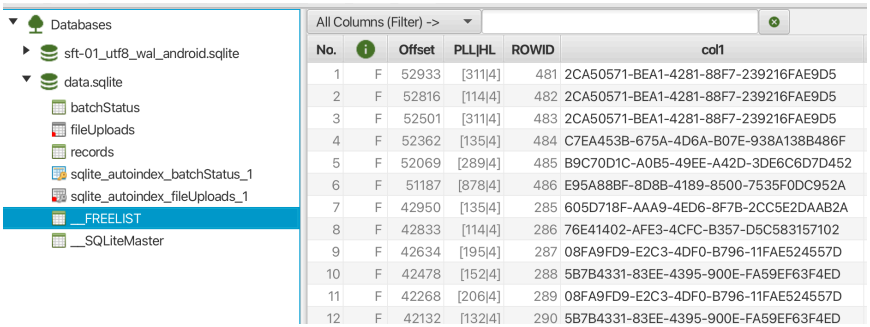


Figure 18: The _FREELIST Table

This means that users do not have to go to the trouble of reconstructing the freelist entries. The table provides them with these entries in one place.

Inspect the Mastertable

Each SQLite database includes a single schema table, which defines all tables, indexes, triggers, and views within the database. The schema table is structured as follows:

```
CREATE TABLE SQLiteMaster(
  type text,
  name text,
  tbl_name text,
  rootpage integer,
  sql text
);
```

FQLite automatically makes the internal table, which would otherwise not be displayed, visible again. The data structure is displayed in the tree view under the name ‚_SQLiteMaster' within the tool (*Figure 19*).

No.	Offset	PLL|HL	ROWID	object	obj name	namespace	root page	
1	3658	[272|6]	1	table	batchStatus	batchStatus	2	CREATE TABLE batchStatus(batchId
2	3933	[53|5]	2	index	sqlite_autoindex_batchStatus_1	batchStatus	3	null
3	3490	[165|6]	3	table	records	records	4	CREATE TABLE records(batchId TEX
4	3145	[287|6]	4	table	fileUploads	fileUploads	5	CREATE TABLE fileUploads(uploadId
5	3435	[53|5]	5	index	sqlite_autoindex_fileUploads_1	fileUploads	6	null

Figure 19: The _SQLiteMaster Table

In addition to the actual data tables, other database objects, such as Indexes or Views, are also stored in this table. The SQL statements are particularly interesting here.

The schema was originally generated using the instructions stored therein. By the way: The SQLite Master table may also contain deleted content or entries. For example, if a table was subsequently removed from the database using the *DROP TABLE* command.

Browse for Location info.

Sometimes, tables contain location information. These are usually represented by *latitude* and *longitude* values that are located in two neighbouring columns. From version 2.7.1, FQLite provides a function that allows you to view the position directly in the browser (*Figure 20*)

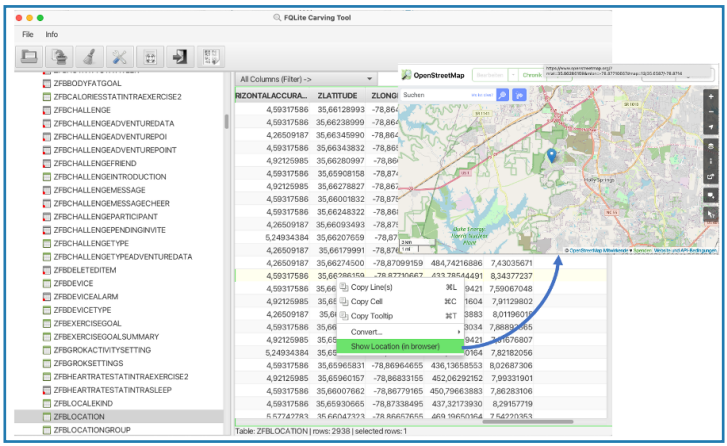


Figure 20: Location functionality in FQLite

To use the function, place the mouse pointer over the relevant cell and then select the menu item "*show location (in browser)*" in the context menu. Note: To use this function, the standard browser is called up, and

OpenStreetMap (OSM) is opened. This requires a working Internet connection.

Finding Timestamps

Timestamp values are critically important in digital forensics as they provide crucial information about the sequence, timing, and context of events within digital systems. These values can serve as key evidence in investigations, helping to establish timelines, verify activities, and link digital artefacts to specific events. A timestamp value can be represented as a string, an integer or even as a float value. It is not always immediately obvious that a cell value is a time specification.

Table 3: common time formats

Time Format	Example
Unix Time	1476199845 would be 11th October 2016 15:30:45
MACTIME	62304000 Would represent 01 January 2010 midnight

FQLite attempts to visualise potential timestamp information on a best-guess basis. Numerical values are converted into the UTC format and displayed if they lie within a corresponding value range (*Figure 21*).

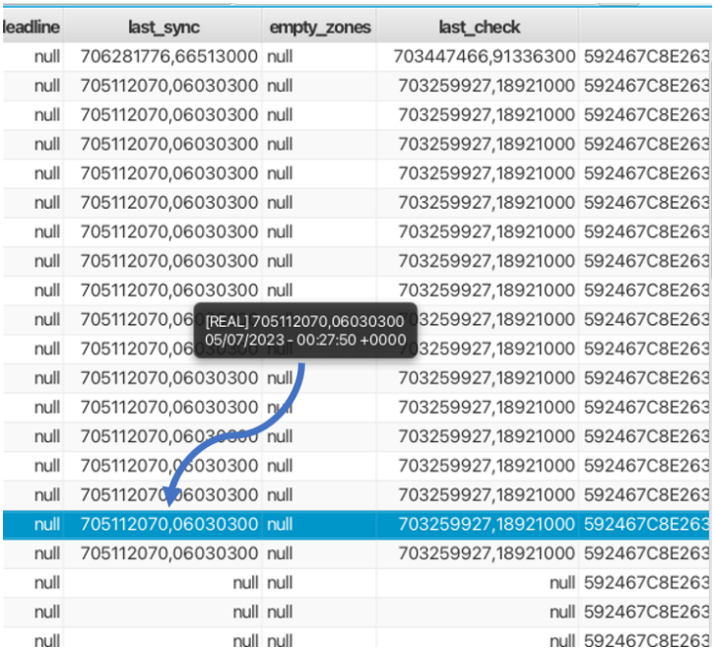


Figure 21: Tooltip with time stamp value

Data Export

For data extraction, a simple copy option to the clipboard is offered in the table overview. This can be accessed via the context menu (activated by double-clicking). Alternatively, if the entire table is to be exported, it must be selected in the tree view. The export dialogue can then be accessed via the context menu or through the toolbar. In this case, it is mandatory to specify a file name for the database object to be exported. In addition to the option to move the entire table row to the clipboard, it is also possible to copy just a single table cell. Since version 2.2, it is also possible to move the tooltip text to the clipboard instead of the cell value.

Table 4: Export - Functionalities of FQLite

Toolbar Icon	Description
	This option can be used to export the content of the complete database file to CSV.
	This option will export all recovered data records to a new database.

When exporting to a new database, all tables except index tables are transferred. The _FREELIST table, containing the freelist records, is also transferred during export.

Overflow Page Recovery

FQLite ensures reliable overflow page recovery for large data entries by automatically detecting overflow pages and linking fragmented data across multiple storage blocks. This process enables the seamless reconstruction of complete records, even when they span multiple pages. Additionally, FQLite is designed to handle multi-byte character encodings, ensuring data integrity and accuracy across diverse datasets.

Working with BLOBs

FQLite incorporates automated BLOB analysis to ensure accurate detection and interpretation of binary data embedded in database fields. The engine can process a wide range of formats, including image files such as PNG, BMP, GIF, JPEG, TIFF, and HEIC, as well as documents like

embedded PDF records. For structured serialization, FQLite supports Protocol Buffers, AVRO, and Thrift, enabling the extraction and reconstruction of complex data models. Additionally, it provides full compatibility with Apple-specific binary plist files and includes decoding mechanisms for BASE64 and other encoding schemes, ensuring reliable recovery and semantic consistency of diverse binary objects (*Figure 22*).

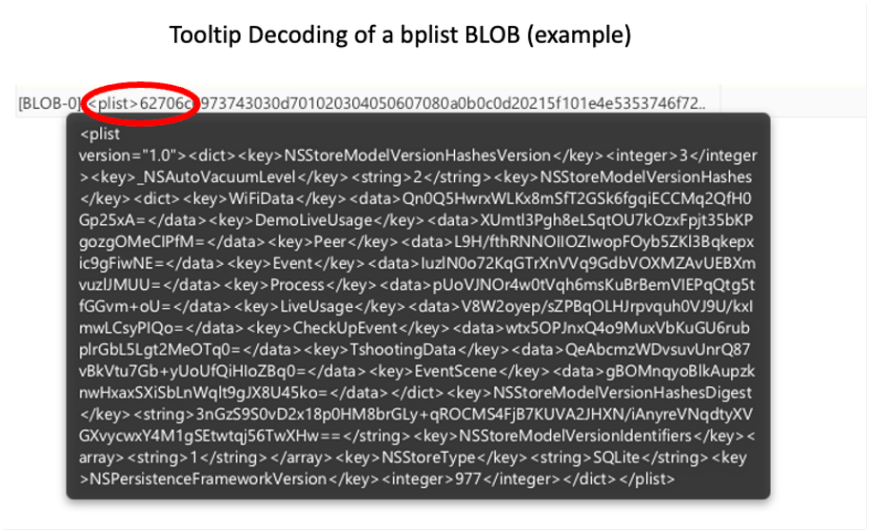


Figure 22: Encoding Example

CHAPTER 4

ADVANCED FQLITE FEATURES

FQLite provides comprehensive support for analysing rollback journals and write-ahead logging (WAL) files, enabling detailed inspection of transactional activity within databases. When rollback journals are present, the engine automatically detects associated files, allowing users to load the main database and examine transaction history, including uncommitted changes, through the Journal Analysis view. For databases employing WAL, FQLite seamlessly processes WAL files located in the same directory as the primary database, reconstructing recent modifications and transaction sequences. This functionality also supports the recovery of data from incomplete or interrupted transactions, ensuring consistency and integrity during forensic or diagnostic analysis. In this chapter, we will provide a first look at advanced analysis techniques.

Write-Ahead Logs

A special feature of the application that distinguishes it from normal SQLite viewers is the display and analysis of write-ahead logs (WAL).
The Write-Ahead Log (WAL) is one of SQLite's journaling modes that provides better concurrency and crash recovery compared to traditional rollback journals. Instead

of modifying the database file directly, changes are first written to a separate WAL file, then later transferred to the main database through a process called **checkpointing**. The WAL files are automatically evaluated by FQLite (*Figure 23*). Only non-empty files are displayed in the file tree. It is important to note at this point that the WAL archive is only read and not transferred to the database, as is usually the case.

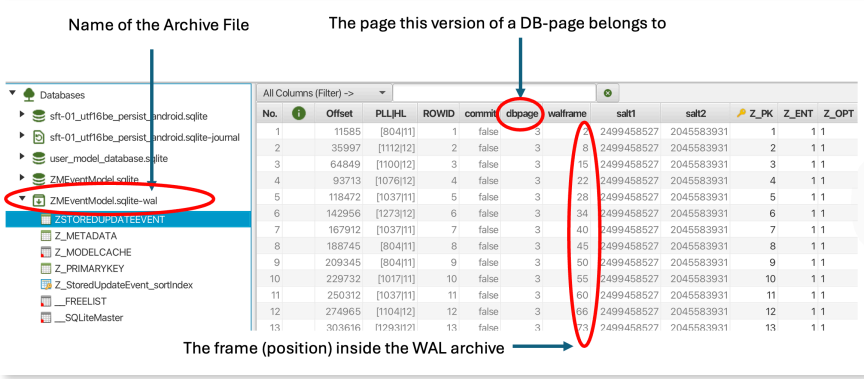


Figure 23: The WAL-Archive Tableview

In a WAL archive, older and newer versions of the same data record can exist side by side. The again belong to a specific checkpoint. A checkpoint is the process of copying data from the WAL file back into the main database file. In addition to the data records stored in the archive, a special tab called „Checkpoints“ is therefore provided at the archive level, which can be used to check how many and which checkpoints are stored in the archive (see *Figure 25*). The WAL archive file also has a header (*Figure 24)*. This can

File Info | Write Ahead Log Header × | Checkpoints

Offset	Property	Value
4	File format version	3007000
8	Database page size	4096
12	Checkpoint sequence number	0
16	Salt-1	2499458527
20	Salt-2	2045583931
24	Checksum1	3030187431
28	Checksum2	2812789445

Figure 24: Write Aheade Log Header

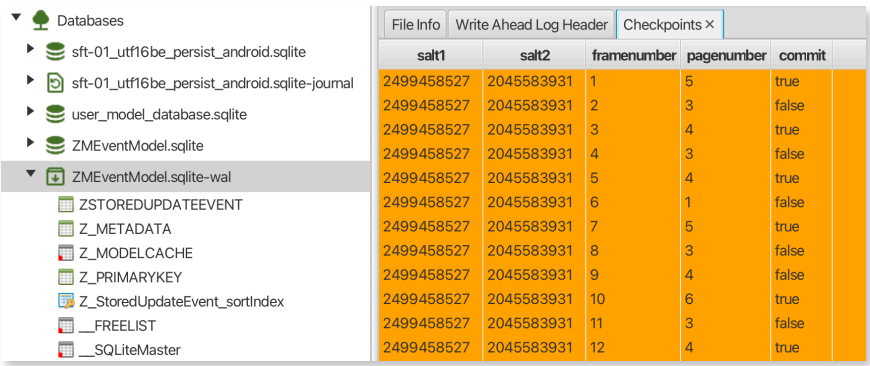


Figure 25: Checkpoint (Overview)

be accessed via a special tab called Write Ahead Log Header (see Figure).

Rollback Journals

A rollback journal in SQLite is a temporary file that enables atomic transactions and crash recovery in the traditional (non-WAL) journaling mode. FQLite also supports the display and evaluation of these special files (*Figure 26*). As with WAL archive files, these are only displayed if they

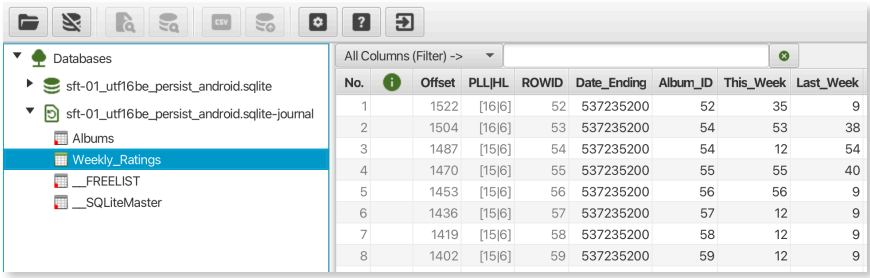


Figure 26: The Rollback Journal Table View

actually contain data. They do not need to be opened separately, but are automatically recognised and displayed.

SQL-Analyzer

The SQL-Analyser in FQLite provides built-in SQL

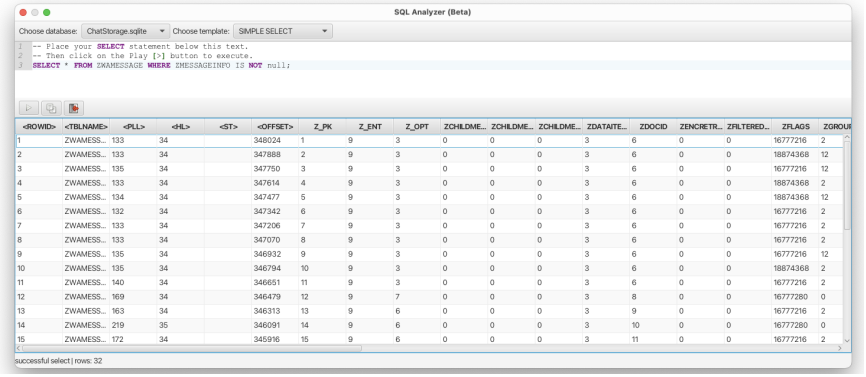


Figure 27: The SQL Analyzer Window

capabilities that extend beyond basic query execution. Users can write and execute custom SQL statements directly against the database, enabling precise data exploration. The analyser also supports schema analysis,

allowing programmatic inspection of database structures such as tables, columns, and relationships.

The SQL-Analyser gives support to simple queries. These queries typically operate on a single table and perform basic data retrieval or manipulation operations without complex logic or relationships.

JOIN operations represent one of the most important concepts in relational databases, allowing queries to combine data from multiple related tables. With the SQL-Analyser component, you can also run these types of analytical queries against a database (see *Figure 28)*:

- INNER JOINs retrieve records that have matching values in both tables, creating result sets that contain only complete relationships.
- LEFT and RIGHT JOINs preserve all records from one table while including matching records from the other, filling in NULL values where relationships don't exist.
- FULL OUTER JOINs combine the behaviour of both LEFT and RIGHT JOINs, ensuring that all records from both tables are included regardless of whether matching relationships exist. These operations enable complex data retrieval scenarios.

All query results are again displayed in tabular form. As with Tableview, you can copy the results to the clipboard at any time so that you can edit them further in another application after pasting.

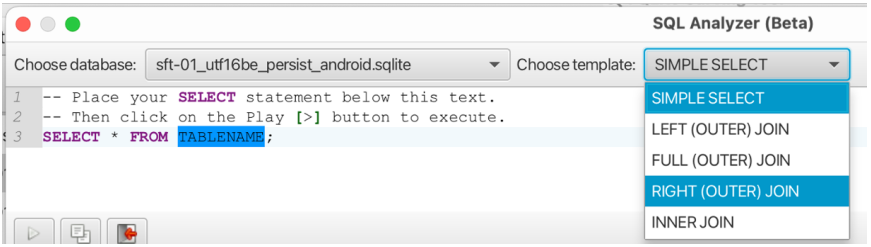


Figure 28: SQL-Analyzer Select-Templates

Epilogue

FQLite represents a powerful tool for SQLite database forensics and data recovery. Its comprehensive feature set, forensically sound methodology, and user-friendly interface make it an essential tool for digital forensic investigations. By understanding its capabilities and following proper forensic procedures, investigators can effectively recover critical data from SQLite databases while maintaining the integrity required for legal proceedings.

Remember that forensic analysis is both a technical and legal discipline. Always ensure you have proper authorisation, follow established procedures, and maintain the highest standards of professional conduct when using FQLite in investigative work.

This handbook is based on FQLite version information available as of September 2025. Always refer to the official project website and release notes for the most current information about features and capabilities.