FQLite - Official User Documentation

Author: *Dirk Pawlaszczyk*
Last updated: 19/08/2024

Dirk Pawlaszczyk, Forensics Expert, Professor for Cybersecurity and Digital Forensics,
Hochschule Mittweida – University of Applied Sciences, pawlaszc@hs-mittweida.de

---

Introduction

Forensic SQLite Data Recovery Tool (FQLite) is tool that helps in recovering data from SQLite database files. FQLite is for users and forensic investigator who want to analyze an SQLite database in a forensically sound manner. Hence, it is a useful tool for digital forensic investigators who need to analyze 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.

It is written with the Java programming language. In this handbook, 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 analyze an entire SQLite database.

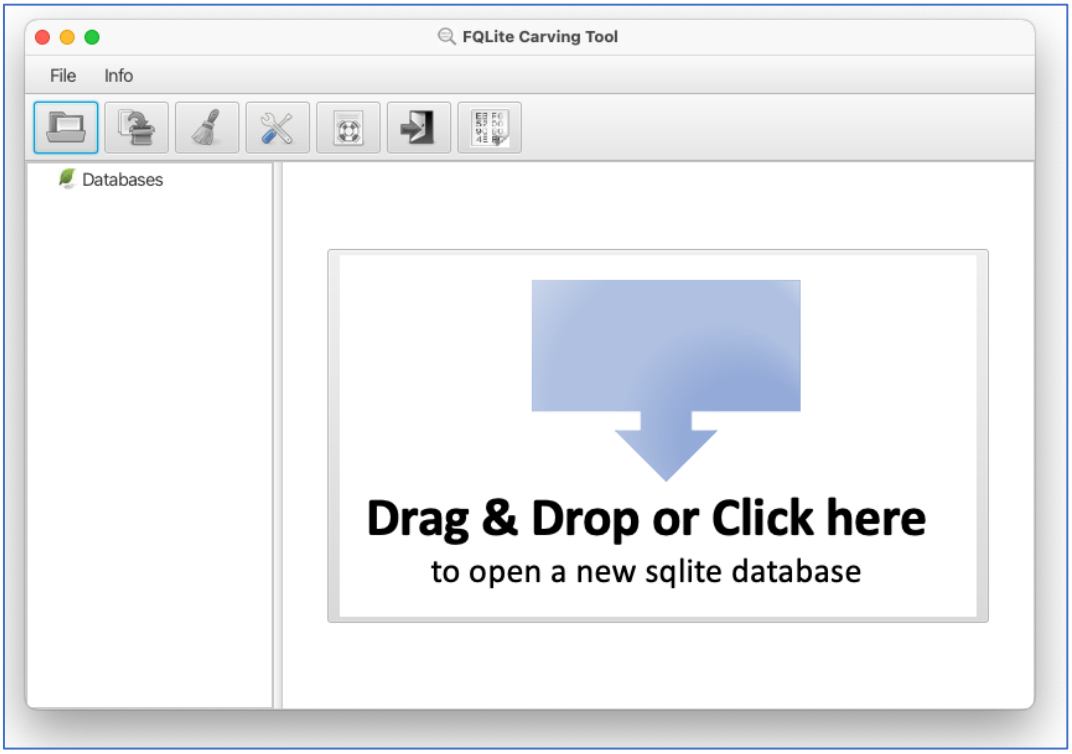


Figure: The Application Window after Startup

How to start? - the Main Window

To start *FQLite* on Windows run the batch file run.bat files with double-click. After starting FQLite, the user is first presented with the main window. Above the actual view area there is a toolbar 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 also not modified at any time. This distinguishes the program from a conventional viewer. Furthermore, when analyzing a new database, the program always automatically searches for possibly deleted content.

File Type Support

SQLite was developed for the analysis of 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.

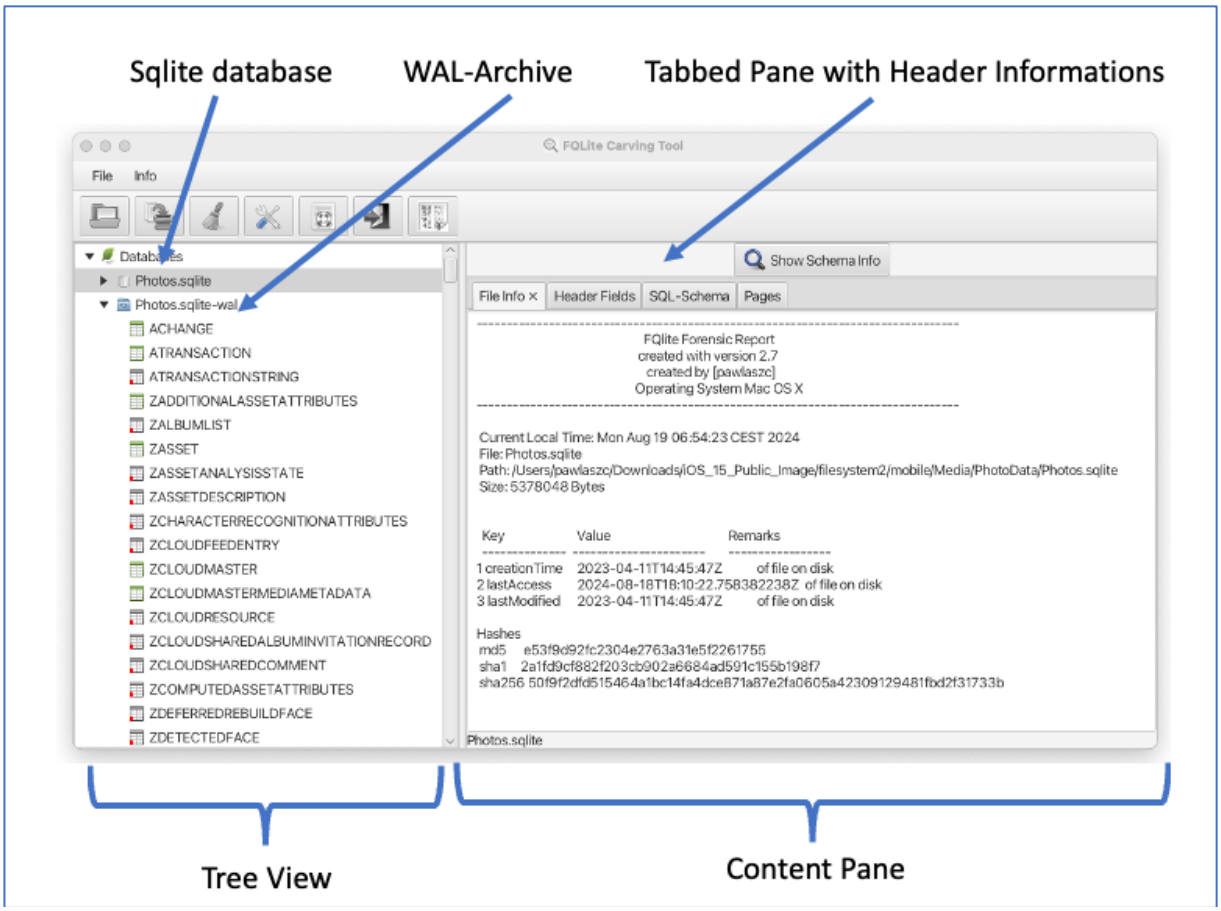


Figure: The Application Window and its divisions

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. A database and its tables are represented as nodes of the tree. The database file is represented 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. As soon as you select a database or a table in the tree view, the view of the Content Pane is automatically updated. 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 database, (4) calling up the user manual and exiting the application (5).

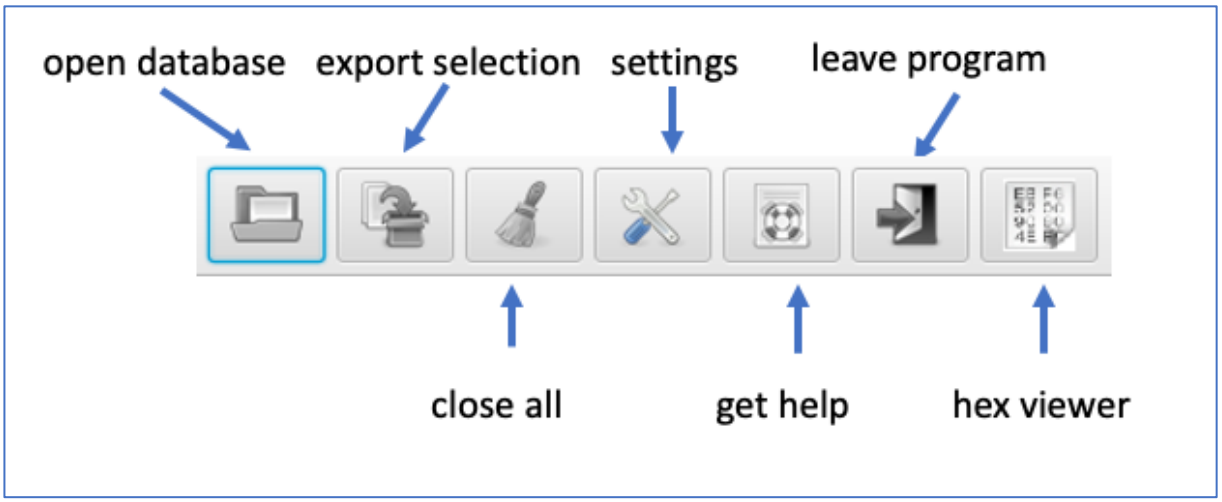


Figure: The FQLite Toolbar and the meaning of icons

Types of Tables

SQLite supports several types of tables, each serving different purposes within a database. Here are the primary types of tables available in SQLite:

1. **Ordinary (or Regular) Tables**:
    - These are the most common type of tables used in SQLite. They store data in rows and columns, where each column has a specific data type. Ordinary tables are persistent and can be queried, updated, and deleted like any other table in a relational database.
2. **Index Tables:**
    - In SQLite, an index table refers to the internal data structure used to store and manage indexes. An index in a database is a separate data structure that SQLite uses to speed up the retrieval of rows from a table
3. **Virtual Tables**:
    - Virtual tables allow you to create a table interface over data that is not actually stored in the SQLite database file. Instead, the data may come from other sources like files, external databases, or in-memory structures. SQLite provides various modules for virtual tables, such as FTS4 for full-text search or R*Tree for spatial indexing.
4. **WITHOUT ROWID Tables**:
    - These are a special type of table in SQLite where the table does not have a built-in ROWID column. Instead, one or more columns of the table are used as the primary key. WITHOUT ROWID tables can be more efficient in terms of space and performance when a natural primary key exists.

5. **System Tables**:
    - SQLite also has system tables like *__SQLiteMaster* (which stores metadata about the schema), *sqlite_sequence* (which stores information about AUTOINCREMENT values), and others. These tables are used internally by SQLite and can be queried to obtain information about the database schema.

FQLite offers support for all these different types of tables. In SQLite, **free list pages** are pages within the database file that are not currently being used to store data but are available for future use. These pages are part of the database's internal management system, helping to efficiently handle storage and reduce fragmentation. When a row is deleted or a table is dropped, the pages that were used to store that data are added to the free list. These pages are then available for reuse the next time SQLite needs to store new data. The content of the free page list is provided in FQLite via the *__FREELIST* table. This is not a real table. Rather, all released pages and data records are displayed here (if they exist).

The Header-Fields Tab

The header of an SQLite database file is a critical part of the database structure, containing essential metadata that defines the database format and guides how the SQLite engine interacts with the database. The header is located at the very beginning of the database file and consists of 100 bytes.

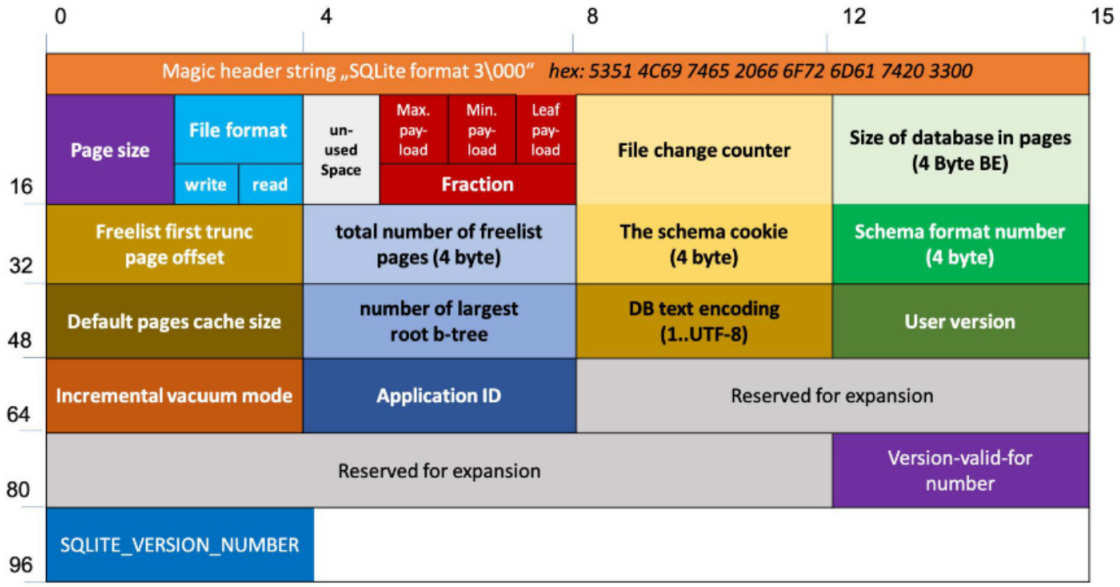


Figure: The SQLite Database Header Format

The SQLite database file header is meticulously organized to store essential metadata, providing information on the file format, page size, encoding, and other parameters. This structure is crucial for SQLite to efficiently manage and access the database, ensuring compatibility and reliability across different versions and environments.

Here's a breakdown of the organization of the SQLite database file header:

[1] Magic Header String (Offset 0, 16 bytes)
   a. Content: "SQLite format 3\000"
   b. Description: This string identifies the file as an SQLite 3 database. The string is exactly 16 bytes long, followed by a null terminator (\000).

[2] Page Size (Offset 16, 2 bytes)
   a. Content: 2-byte unsigned integer
   b. Description: Defines the database page size in bytes. It can range from 512 to 65536 bytes. The default page size is typically 4096 bytes. If the value is 1, it indicates a page size of 65536 bytes.

[3] File Format Write Version (Offset 18, 1 byte)
   a. Content: 1-byte unsigned integer
   b. Description: Indicates the file format version used for writing. Typically, this is set to 1 for the standard version.

[4] File Format Read Version (Offset 19, 1 byte)
   a. Content: 1-byte unsigned integer
   b. Description: Indicates the file format version used for reading. Like the write version, this is typically set to 1 for the standard version.

[5] Reserved Space per Page (Offset 20, 1 byte)
   a. Content: 1-byte unsigned integer
   b. Description: Specifies the number of bytes reserved at the end of each page for extensions. Generally set to 0, indicating no reserved space.

[6] Maximum Embedded Payload Fraction (Offset 21, 1 byte)
   a. Content: 1-byte unsigned integer
   b. Description: This value defines the maximum fraction of a database page that can be used for storing data in B-tree cells. The default value is 64, corresponding to 64/255 or about 25%.

[7] Minimum Embedded Payload Fraction (Offset 22, 1 byte)
   a. Content: 1-byte unsigned integer
   b. Description: This value specifies the minimum fraction of a database page that should be used for B-tree cell data. The default value is 32, corresponding to 32/255 or about 12.5%.

[8] Leaf Payload Fraction (Offset 23, 1 byte)

a. Content: 1-byte unsigned integer
b. Description: Defines the fraction of a database page that should be used for storing data in leaf nodes of B-trees. The default value is 32.

[9] File Change Counter (Offset 24, 4 bytes)
a. Content: 4-byte unsigned integer
b. Description: A counter that increments whenever the database file is modified. It helps in detecting changes to the database.

[10] Database Size in Pages (Offset 28, 4 bytes)
a. Content: 4-byte unsigned integer
b. Description: Indicates the total number of pages in the database file, including the freelist pages.

[11] First Freelist Trunk Page (Offset 32, 4 bytes)
a. Content: 4-byte unsigned integer
b. Description: The page number of the first freelist trunk page. If there are no freelist pages, this is 0.

[12] Total Number of Freelist Pages (Offset 36, 4 bytes)
a. Content: 4-byte unsigned integer
b. Description: The total number of freelist pages in the database.

[13] Schema Cookie (Offset 40, 4 bytes)
a. Content: 4-byte unsigned integer
b. Description: A version number for the schema format. This is used to detect changes to the schema of the database.

[14] Schema Format Number (Offset 44, 4 bytes)
a. Content: 4-byte unsigned integer
b. Description: Indicates the schema format used by the database. Typical values are 1, 2, 3, or 4, with 4 being the most recent format.

[15] Default Page Cache Size (Offset 48, 4 bytes)
a. Content: 4-byte unsigned integer
b. Description: Specifies the default number of pages in the page cache. If the value is 0, the cache size is determined by the application.

[16] Largest B-tree Page Number (Offset 52, 4 bytes)
a. Content: 4-byte unsigned integer
b. Description: The page number of the largest root B-tree page in the database.

[17] Database Text Encoding (Offset 56, 4 bytes)
a. Content: 4-byte unsigned integer
b. Description: Specifies the text encoding used in the database. The possible values are 1 (UTF-8), 2 (UTF-16le), and 3 (UTF-16be).

[18] User Version (Offset 60, 4 bytes)

a. Content: 4-byte unsigned integer
b. Description: This is a value that can be set and used by the user. SQLite does not modify this field; it's available for user-defined purposes.

[19] Incremental Vacuum Mode (Offset 64, 4 bytes)
a. Content: 4-byte unsigned integer
b. Description: Indicates whether incremental vacuum mode is enabled (1) or not (0).

[20] Application ID (Offset 68, 4 bytes)
a. Content: 4-byte unsigned integer
b. Description: An application-specific identifier that can be used to identify the database file format in custom applications.

[21] Reserved for Expansion (Offset 72, 20 bytes)
a. Content: 20 bytes reserved
b. Description: These bytes are reserved for future use by SQLite. They should be set to zero.

[22] Version Valid-for Number (Offset 92, 4 bytes)
a. Content: 4-byte unsigned integer
b. Description: A value used to determine whether the database needs to be checked for integrity following a hot journal rollback.

[23] SQLite Version Number (Offset 96, 4 bytes)
a. Content: 4-byte unsigned integer
b. Description: Indicates the version of SQLite that last wrote to the database file. The value is in the format 3008007 for SQLite version 3.8.7, for example.

There is information on FQLite itself at http://www.sqlite.org, which includes the official language overview.

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: The "Header Fields" – Tab of a data base (example)

The SQL-Schema Tab

The schema definition in SQLite serves as the blueprint for the structure and organization of the database. It outlines the various elements within the database and defines how data is stored, accessed, and manipulated.

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: The Schema-Definition Tab

The special schema tab allows you to view the tables, indices and trigger definitions available within the currently selected database.

The Pages Tab

The *pages* tab can be used to find out which database page is currently occupied by which table. You can also easily view the offset of each page and the page type.

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

page	offset	type of page	table	signature
1	0	database header	null	null
2	4096	pointer map	null	null
3	8192	interior table b-tree page	ZADDITIONALASSETATTRIBUTES	[INT, INT, INT, INT, INT, INT, INT, INT, INT, INT, INT, INT, INT, IN...
4	12288	leaf table b-tree page	Z_1KEYWORDS	[INT, INT]
5	16384	leaf index b-tree page	sqlite_autoindex_Z_1KEYWORDS_1	[INT]
6	20480	leaf table b-tree page	ZALBUMLIST	[INT, INT, INT, INT, INT, TEXT]
7	24576	interior table b-tree page	ZASSET	[INT, INT, INT, INT, INT, INT, INT, INT, INT, INT, INT, INT, INT, IN...
8	28672	leaf table b-tree page	Z_3SUGGESTIONSBEINGREPRESENTATIVEASSE...	[INT, INT]
9	32768	leaf index b-tree page	sqlite_autoindex_Z_3SUGGESTIONSBEINGREPRE...	[INT]
10	36864	leaf table b-tree page	Z_3SUGGESTIONSBEINGKEYASSETS	[INT, INT]
11	40960	leaf index b-tree page	sqlite_autoindex_Z_3SUGGESTIONSBEINGKEYAS...	[INT]
12	45056	leaf table b-tree page	Z_3MEMORIESBEINGREPRESENTATIVEASSETS	[INT, INT]
13	49152	leaf index b-tree page	sqlite_autoindex_Z_3MEMORIESBEINGREPRESEN...	[INT]
14	53248	leaf table b-tree page	Z_3MEMORIESBEINGMOVIECURATEDASSETS	[INT, INT]
15	57344	leaf index b-tree page	sqlite_autoindex_Z_3MEMORIESBEINGMOVIECU...	[INT]
16	61440	leaf table b-tree page	Z_3MEMORIESBEINGCURATEDASSETS	[INT, INT]
17	65536	leaf index b-tree page	sqlite_autoindex_Z_3MEMORIESBEINGCURATED...	[INT]
18	69632	leaf table b-tree page	Z_3MEMORIESBEINGEXTENDEDCURATEDASSETS	[INT, INT]
19	73728	leaf index b-tree page	sqlite_autoindex_Z_3MEMORIESBEINGEXTENDED...	[INT]
20	77824	leaf table b-tree page	Z_3MEMORIESBEINGUSERCURATEDASSETS	[INT, INT]
21	81920	leaf index b-tree page	sqlite_autoindex_Z_3MEMORIESBEINGUSERCUR...	[INT]

Figure: The *Pages*-Definition Tab

Hex-Viewer

FQLite makes it possible to check every match with a Hex-Viewer. 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 view that then opens, the relevant header of the data set is highlighted in color. A record contains a header and a body, in that order. The header begins with a single *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.

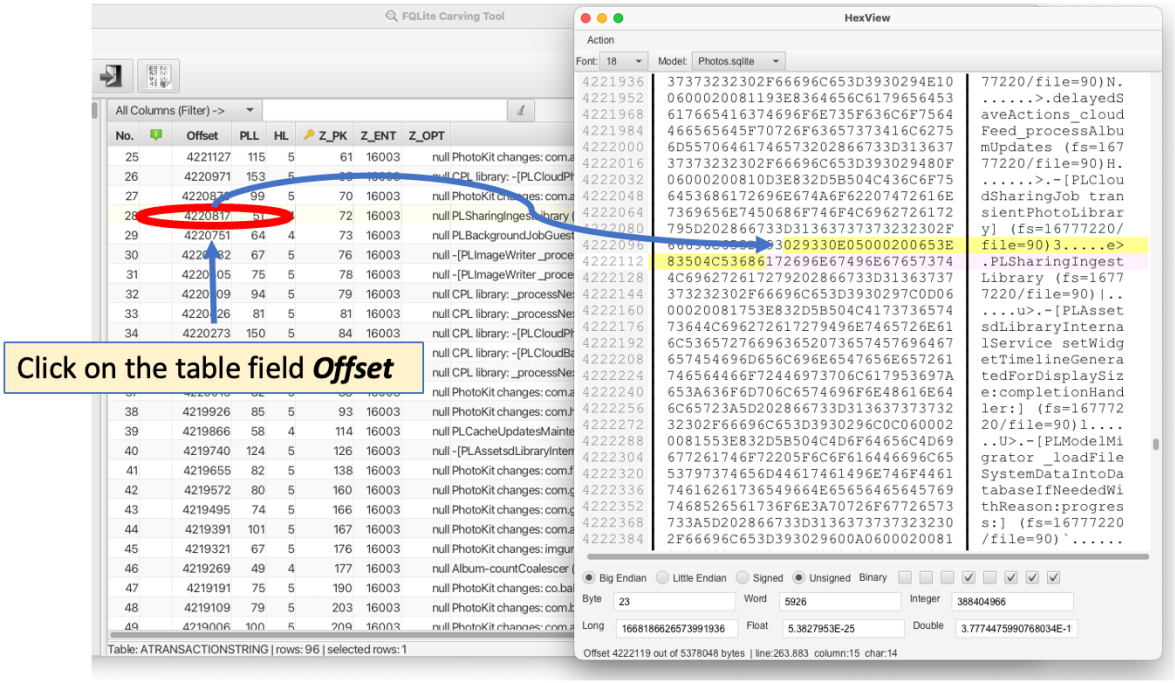
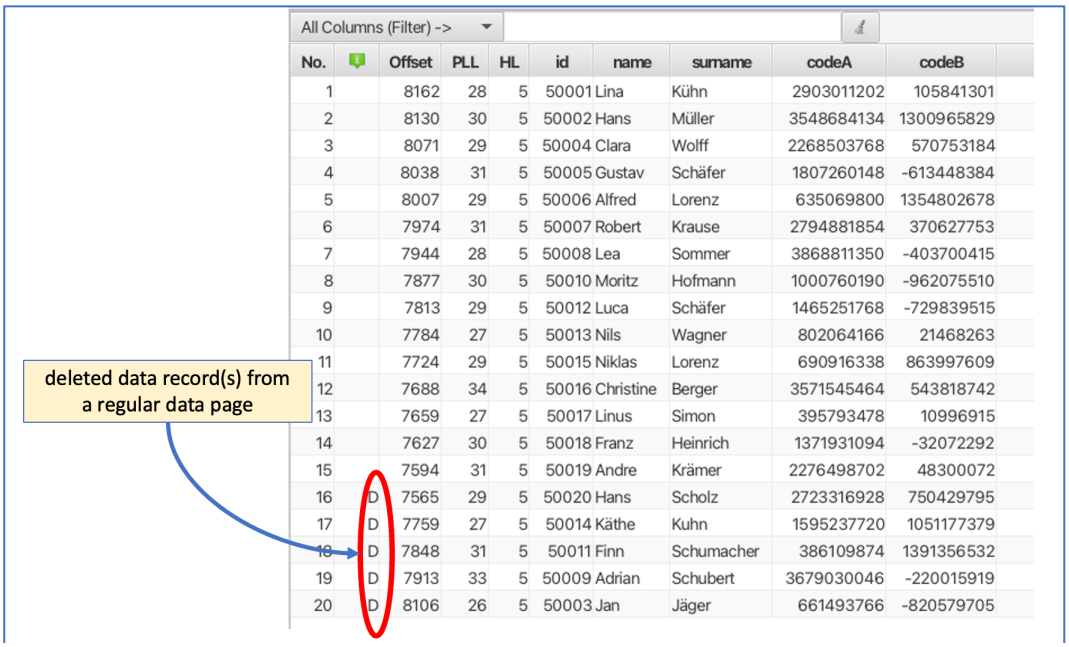


Figure: Jump to offset in hex viewer from a table view

View Deleted Content

As already mentioned, FQLite is capable of restoring deleted records in whole or in part. This is the great strength of the application. 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. 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.

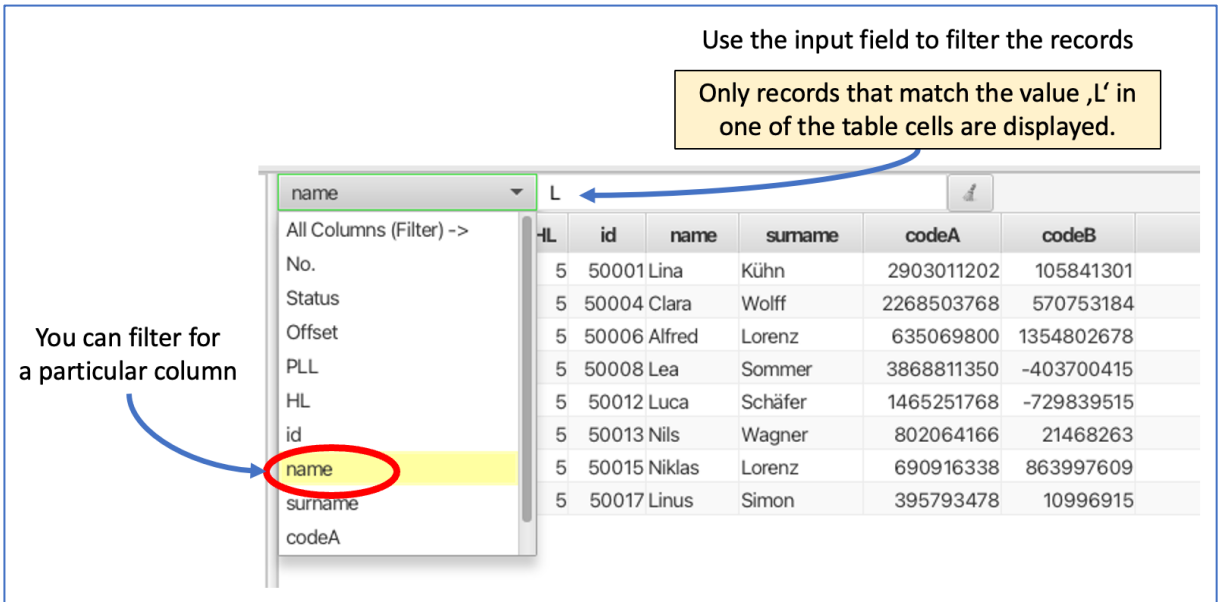


No.		Offset	PLL	HL	id	name	surname	codeA	codeB
1		8162	28	5	50001	Lina	Kühn	2903011202	105841301
2		8130	30	5	50002	Hans	Müller	3548684134	1300965829
3		8071	29	5	50004	Clara	Wolff	2268503768	570753184
4		8038	31	5	50005	Gustav	Schäfer	1807260148	-613448384
5		8007	29	5	50006	Alfred	Lorenz	635069800	1354802678
6		7974	31	5	50007	Robert	Krause	2794881854	370627753
7		7944	28	5	50008	Lea	Sommer	3868811350	-403700415
8		7877	30	5	50010	Moritz	Hofmann	1000760190	-962075510
9		7813	29	5	50012	Luca	Schäfer	1465251768	-729839515
10		7784	27	5	50013	Nils	Wagner	802064166	21468263
11		7724	29	5	50015	Niklas	Lorenz	690916338	863997609
12		7688	34	5	50016	Christine	Berger	3571545464	543818742
13		7659	27	5	50017	Linus	Simon	395793478	10996915
14		7627	30	5	50018	Franz	Heinrich	1371931094	-32072292
15		7594	31	5	50019	Andre	Krämer	2276498702	48300072
16	D	7565	29	5	50020	Hans	Scholz	2723316928	750429795
17	D	7759	27	5	50014	Käthe	Kuhn	1595237720	1051177379
18	D	7848	31	5	50011	Finn	Schumacher	386109874	1391356532
19	D	7913	33	5	50009	Adrian	Schubert	3679030046	-220015919
20	D	8106	26	5	50003	Jan	Jäger	661493766	-820579705

Figure: Viewing deleted content inside FQLite

Filter Content

Records in the table can be easily filtered using the Filter input field located above the table. Searching by means of regular expressions is not supported at the moment. However, this feature is planned for one of the next versions.



HL	id	name	surname	codeA	codeB
5	50001	Lina	Kühn	2903011202	105841301
5	50004	Clara	Wolff	2268503768	570753184
5	50006	Alfred	Lorenz	635069800	1354802678
5	50008	Lea	Sommer	3868811350	-403700415
5	50012	Luca	Schäfer	1465251768	-729839515
5	50013	Nils	Wagner	802064166	21468263
5	50015	Niklas	Lorenz	690916338	863997609
5	50017	Linus	Simon	395793478	10996915

Figure: filtering table values in FQLite (example)

Browse for Location info

In many cases, tables contain location information. These are usually represented by latitude and longitude values that are located in two neighboring columns. From version 2.7.1, FQLite provides a function to view the position directly in the browser.

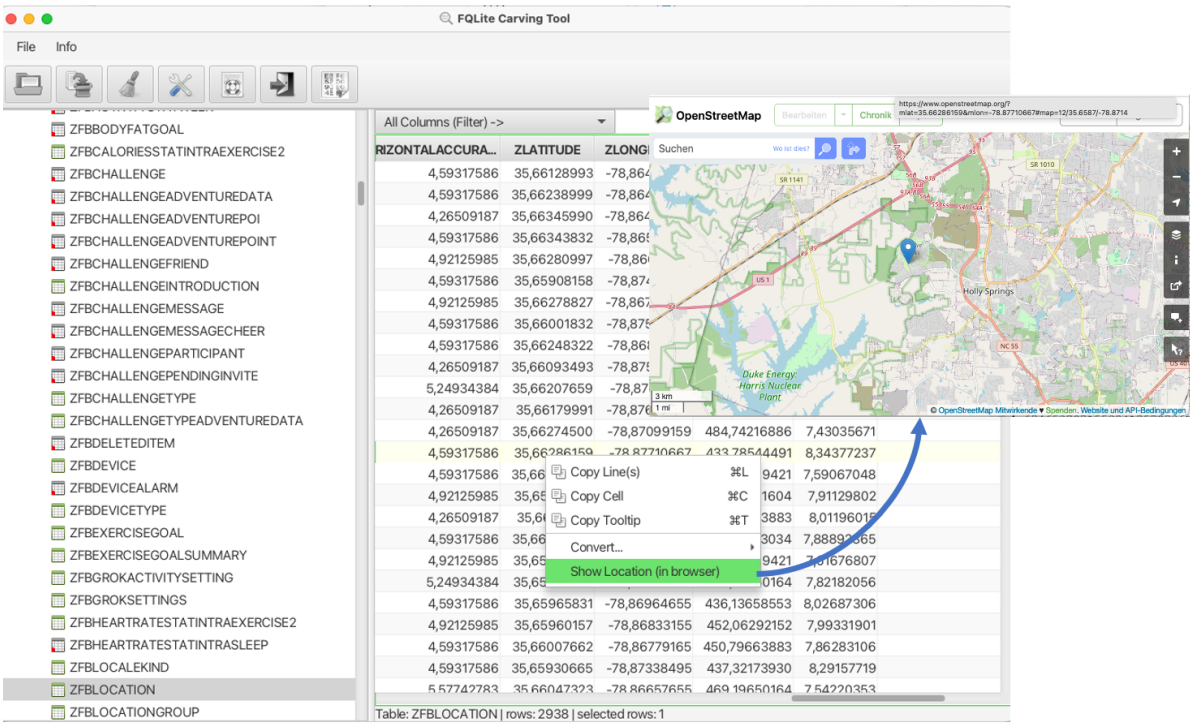


Figure: Preview for geo locations in FQLite

Preview Images and other binary Formats

FQLite supports a preview function for common binary formats such as .png, .jpg, .gif, .bmp, .pdf, .tiff. To activate this, it is usually sufficient to place the mouse pointer over the cell where the binary information is located. For special file formats for which there is no built-in viewer, the standard viewer provided by the operating system is used instead. This is the case, for example, for binary formats like High Efficiency Image Container Format (*heic*) or Tag Image File Format *(tiff)*.

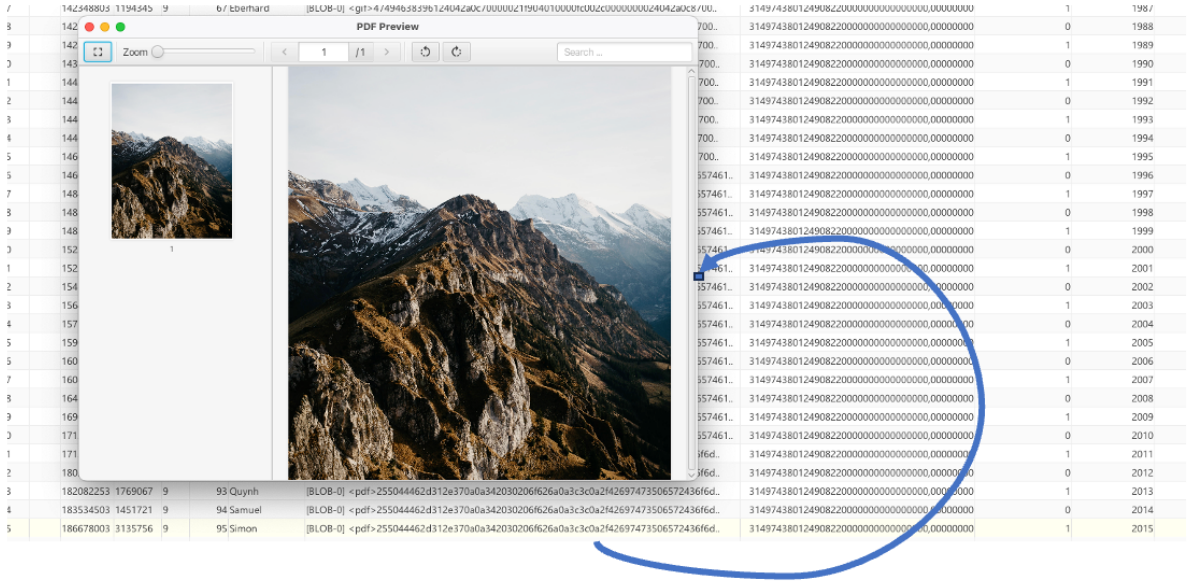


Figure: Preview for embedded PDF-Documents with the build in viewer

7891944	1920188	9	16	Chrissa2	[BLOB-0] <png>89504e470d0a1a0a0000000d494844520000035e0000047c08060000008d1b45..
7891447	1645118	9	17	Chrissa3	[BLOB-0] <png>89504e470d0a1a0a0000000d494844520000035e0000047c08060000008d1b45..
7890950	1350541	9	18	Chrissa4	[BLOB-0] <png>89504e470d0a1a0a0000000d494844520000035e0000047c08060000008d1b45..
12448384	1558408	9	19	Chrissa5	[BLOB-0] <png>89504e470d0a1a0a0000000d494844520000035e0000047c08060000008d1b45..
13770992	1321472	9	20	Hope	[BLOB-0] <png>89504e470d0a1a0a0000000d4948445200000 0005e3aea..
14954535	1182545	9	21	Maksym	[BLOB-0] <png>89504e470d0a1a0a0000000d4948445200000 0000cb3d63..
17759717	2795366	9	22	Maksymsirman	[BLOB-0] <png>89504e470d0a1a0a0000000d4948445200000 0000c40e3c..
17758021	811904	8	23	Navil	[BLOB-0] <png>89504e470d0a1a0a0000000d4948445200000 0000689fde..
17756912	483957	8	24	Navi2	[BLOB-0] <png>89504e470d0a1a0a0000000d4948445200000 0000625782..
20795940	1742640	9	25	Alain	[BLOB-0] <png>89504e470d0a1a0a0000000d4948445200000 5dc000003e80806000000c40e3c..
22727261	1924675	9	26	Wolfgang	[BLOB-0] <png>89504e470d0a1a0a0000000d4948445200000 3e8000005dc0806000000b198a4..
22726764	1252472	9	27	Laura	[BLOB-0] <png>89504e470d0a1a0a0000000d4948445200000 3760000052f0806000000fdfc77..

Preview Function for BLOBs: .jpeg, .bmp, .gif, .png

For certain important binary formats such as Apple's property list or serialized Java objects, a preview is also offered via the tooltip of the table cell.

Tooltip Decoding of a bplist BLOB (example)

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 artifacts to specific events. A time-value can be represented as a string, in integer or even as float value. It is not always immediately obvious that a cell value is a time specification.

FQLite attempts to visualize potential timestamp information on a best-guess basis. Numerical values are converted into the UTC time format and displayed if they lie within a corresponding value range.

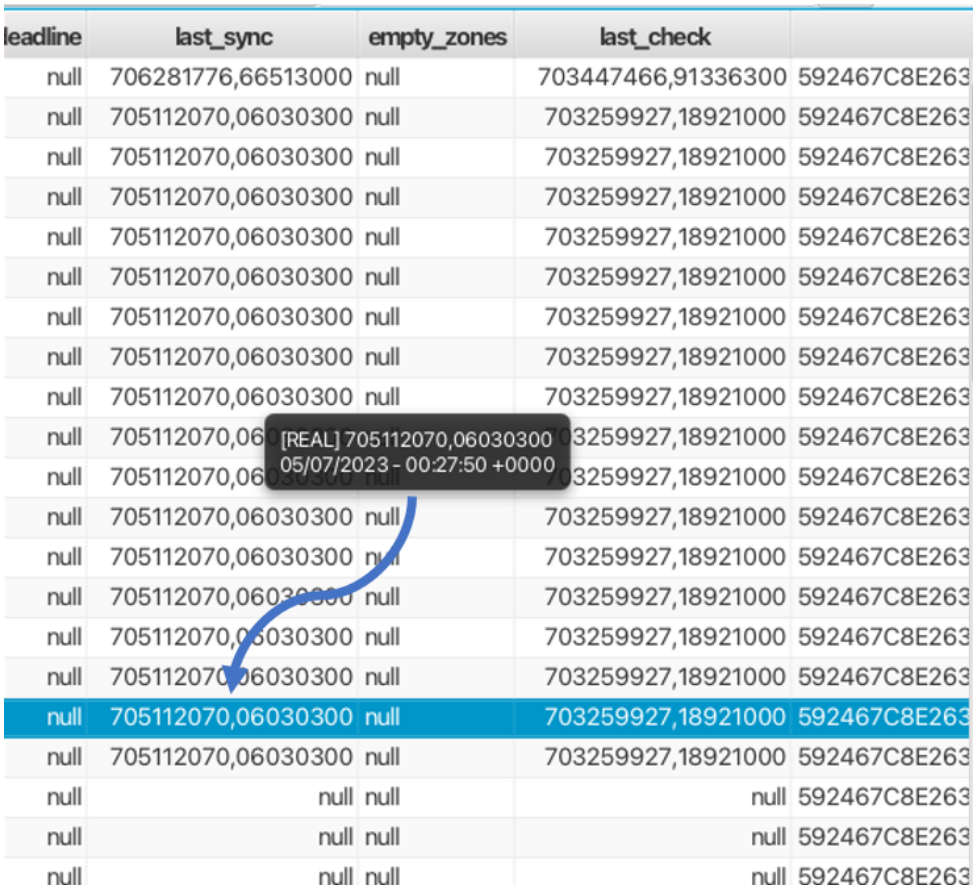


Figure: automatic time stamp detection in FQLite

Export data to file system or clipboard

For data extraction, a simple copy option to the clipboard is offered in the table overview. This can be called up via the context menu (can be called up 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 called up via the context menu or alternatively via 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.

Need more Information?

There is information on FQLite itself at

https://www.staff.hs-mittweida.de/~pawlaszc/fqlite/,

which includes the official language overview. You can find the latest information and news about the tool on the project website. There you will also find references to videos and further sources. You check out the source code you can download the Github repository from this link:

https://github.com/pawlaszczyk/fqlite

References

[1] SQLite Consortium, “Official SQLite database project website," 2024. [Online]. Available: https://www.sqlite.org/.

[2] D. Pawlaszczyk, "SQLite," in Mobile Forensics - The File Format Handbook, https://doi.org/10.1007/978-3-030-98467-0_5 , Springer Cham, 2022.

[3] D. Pawlaszczyk, "Java Serialization," in Mobile Forensics - The File Format Handbook", https://doi.org/10.1007/978-3-030-98467-0_7, Springer Cham, 2022.