Introduction of SQLite:

SQL Server is an RDBMS in which data is stored in the form of tables. Any number of tables can be created and they can be related to each other. In a mobile application development technology such as Flutter, SQL Server can be used by adding the sql\_conn plugin to the app. These days, though, when it comes to small Flutter applications, SQLite is chosen over SQL Server. SQLite is a free, self-contained, serverless, and lightweight RDBMS.

So, if you're building a mobile app with a small to medium-sized database and a need for local storage and offline capabilities, SQLite is generally a suitable and popular choice. However, for larger-scale applications with complex data needs, multi-user access, and centralized data management, SQL Server or other client-server databases may be more appropriate.

How SQLite works for Flutter?

Interface

Much of the C-language Interface is found in source files asmain.c, though some routines are scattered about in other files where they can have access to data structures with file scope.

To avoid name collisions, all external symbols in the SQLite library begin with the prefix sqlite3. Those symbols that are intended for external use (in other words, those symbols which form the API for SQLite) add an underscore, and thus begin with sqlite3\_.

Tokenizer

When a string containing SQL statements is to be evaluated it is first sent to the tokenizer. The tokenizer breaks the SQL text into tokens and hands those tokens one by one to the parser.

Parser

The parser assigns meaning to tokens based on their context. The parser for SQLite is generated using the Lemon parser generator. Lemon does the same job as YACC/BISON, but it uses a different input syntax which is less error-prone. Lemon also generates a parser which is reentrant and thread-safe.

Code Generator

After the parser assembles tokens into a parse tree, the code generator runs to analyze the parse tree and generate bytecode that performs the work of the SQL statement. The prepared statement object is a container for this bytecode. There are many files in the code generator. In these files is where most of the serious magic happens. expr.c handles code generation for expressions. where\*.c handles code generation for WHERE clauses on SELECT, UPDATE and DELETE statements.

The code generator sometimes called the query planner. For any particular SQL statement, there might be hundreds, thousands, or millions of different algorithms to compute the answer. The query planner is an AI that strives to select the best algorithm from these millions of choices.

Bytecode Engine

The bytecode program created by the code generator is run by a virtual machine.

The virtual machine itself is entirely contained in a single source file vdbe.c. which defines structures and interfaces that are private to the virtual machine itself.The vdbeaux.c file contains utilities used by the virtual machine and interface modules used by the rest of the library to construct VM programs.

SQLite implements SQL functions using callbacks to C-language routines. Even the built-in SQL functions are implemented this way.

B-Tree

An SQLite database is maintained on disk using a B-tree implementation found in the btree.c source file. Separate B-trees are used for each table and each index in the database. All B-trees are stored in the same disk file. The file format details are stable and well-defined and are guaranteed to be compatible moving forward.

Page Cache

The B-tree module requests information from the disk in fixed-size pages. The default page\_size is 4096 bytes but can be any power of two between 512 and 65536 bytes. The page cache is responsible for reading, writing, and caching these pages. The page cache also provides the rollback and atomic commit abstraction and takes care of locking of the database file.

OS Interface

In order to provide portability across operating systems, SQLite uses an abstract object called the VFS. Each VFS provides methods for opening, reading, writing, and closing files on disk, and for other OS-specific tasks such as finding the current time, or obtaining randomness to initialize the built-in pseudo-random number generator.

Utilities

Memory allocation, caseless string comparison routines, portable text-to-number conversion routines, and other utilities are located in util.c. Symbol tables used by the parser are maintained by hash tables found in hash.c.

Test Code

Files in the "src/" folder of the source tree whose names begin with test are for testing only and are not included in a standard build of the library.

As SQLite doesn't have web support. There is a package with all the support. You can use Hive DB instead.