SQLite

SQLite, in a nutshell, is a database with limited capabilities when compared to more sophisticated databases. SQLite, on the other hand, does not require any additional setup or software. It is highly dependent on the file system. To put it another way, all data is saved on a disc as a file. These files are very small in size and can easily be copied and backed up.

As of now, Our website don't use database that heavily. So this is the most appropriate choice among many alternatives.

According to SQLite developers

> SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite is the most used database engine in the world. SQLite is built into all mobile phones and most computers and comes bundled inside countless other applications that people use every day.
>
> The SQLite file format is stable, cross-platform, and backwards compatible and the developers pledge to keep it that way through the year 2050. SQLite database files are commonly used as containers to transfer rich content between systems [1] [2] [3] and as a long-term archival format for data [4]. There are over 1 trillion (1e12) SQLite databases in active use [5].

History

D. Richard Hipp designed SQLite in the spring of 2000 while working for General Dynamics on contract with the United States Navy. Hipp was designing software used for a damage-control system aboard guided-missile destroyers, which originally used HP-

UX with an IBM Informix database back-end. SQLite began as a Tcl extension.

The design goals of SQLite were to allow the program to be operated without installing a database management system or requiring a database administrator. Hipp based the syntax and semantics on those of PostgreSQL 6.5. In August 2000, version 1.0 of SQLite was released, with storage based on gdbm (GNU Database Manager). In September 2001, SQLite 2.0 replaced gdbm with a custom B-tree implementation, adding transaction capability. In June 2004, SQLite 3.0 added internationalization, manifest typing, and other major improvements, partially funded by America Online.

In 2011, Hipp announced his plans to add a NoSQL interface (managing documents expressed in JSON) to SQLite databases and to develop UnQLite, an embeddable document-oriented database.

SQLite is one of four formats recommended for long-term storage of datasets approved for use by the Library of Congress.

Design

Unlike client–server database management systems, the SQLite engine has no standalone processes with which the application program communicates. Instead, the SQLite library is linked in and thus becomes an integral part of the application program. Linking may be static or dynamic. The application program uses SQLite's functionality through simple function calls, which reduce latency in database access: function calls within a single process are more efficient than inter-process communication.

SQLite stores the entire database (definitions, tables, indices, and the data itself) as a single cross-platform file on a host machine. It implements this simple design by locking the entire database file during writing. SQLite read operations can be multitasked, though writes can only be performed sequentially.

Due to the server-less design, SQLite applications require less configuration than client–server databases. SQLite is called zero-conf because it does not require service management (such as startup scripts) or access control based on GRANT and passwords. Access control is handled by means of file-system permissions given to the database file itself. Databases in client–server systems use file-system permissions that give access to the database files only to the daemon process.

Another implication of the serverless design is that several processes may not be able to write to the database file. In server-based databases, several writers will all connect to the same daemon, which is able to handle its locks internally. SQLite, on the other hand, has to rely on file-system locks. It has less knowledge of the other processes that are accessing the database at the same time. Therefore, SQLite is not the preferred choice for write-intensive deployments. However, for simple queries with little concurrency, SQLite performance profits from avoiding the overhead of passing its data to another process.

SQLite uses PostgreSQL as a reference platform. "What would PostgreSQL do" is used to make sense of the SQL standard. One major deviation is that, with the exception of primary keys, SQLite does not enforce type checking; the type of a value is dynamic and not strictly constrained by the schema (although the schema will trigger a conversion when storing, if such a conversion is potentially reversible). SQLite strives to follow Postel's rule.

Features

SQLite implements most of the SQL-92 standard for SQL, but lacks some features. For example, it only partially provides triggers and cannot write to views (however, it provides INSTEAD OF triggers that provide this functionality). Its support of ALTER TABLE statements is limited.

SQLite uses an unusual type system for a SQL-compatible DBMS: instead of assigning a type to a column as in most SQL database systems, types are assigned to individual values; in language terms it

is dynamically typed. Moreover, it is weakly typed in some of the same ways that Perl is: one can insert a string into an integer column (although SQLite will try to convert the string to an integer first, if the column's preferred type is integer). This adds flexibility to columns, especially when bound to a dynamically typed scripting language. However, the technique is not portable to other SQL products. A common criticism is that SQLite's type system lacks the data integrity mechanism provided by statically typed columns in other products. The SQLite web site describes a "strict affinity" mode, but this feature has not yet been added. However, it can be implemented with constraints like CHECK(typeof(x)='integer').

Tables normally include a hidden rowid index column, which gives faster access. If a database includes an Integer Primary Key column, SQLite will typically optimize it by treating it as an alias for rowid, causing the contents to be stored as a strictly typed 64-bit signed integer and changing its behavior to be somewhat like an auto-incrementing column. Future[when?] versions of SQLite may include a command to introspect whether a column has behavior like that of rowid to differentiate these columns from weakly typed, non-autoincrementing Integer Primary Keys.

Full support for Unicode case-conversions can be optionally be enabled through an extension.

Several computer processes or threads may access the same database concurrently. Several read accesses can be satisfied in parallel. A write access can only be satisfied if no other accesses are currently being serviced. Otherwise, the write access fails with an error code (or can automatically be retried until a configurable timeout expires). This concurrent access situation would change when dealing with temporary tables. This restriction is relaxed in version 3.7 when write-ahead logging (WAL) is turned on, enabling concurrent reads and writes.

Version 3.6.19 released on October 14, 2009 added support for foreign key constraints.

SQLite version 3.7.4 first saw the addition of the FTS4 (full-text search) module, which features enhancements over the older FTS3 module. FTS4 allows users to perform full-text searches on documents similar to how search engines search webpages. Version 3.8.2 added support for creating tables without rowid, which may provide space and performance improvements. Common table expressions support was added to SQLite in version 3.8.3. 3.8.11 added a newer search module called FTS5, the more radical (compared to FTS4) changes requiring a bump in version.

In 2015, with the json1 extension and new subtype interfaces, SQLite version 3.9 introduced JSON content managing.

As of version 3.33.0, the maximum supported database size is 281 TB.

Development and Distribution

SQLite's code is hosted with Fossil, a distributed version control system that is itself built upon an SQLite database.

A standalone command-line program is provided in SQLite's distribution. It can be used to create a database, define tables, insert and change rows, run queries and manage an SQLite database file. It also serves as an example for writing applications that use the SQLite library.

SQLite uses automated regression testing prior to each release. Over 2 million tests are run as part of a release's verification. Starting with the August 10, 2009 release of SQLite 3.6.17, SQLite releases have 100% branch test coverage, one of the components of code coverage. The tests and test harnesses are partially public-domain and partially proprietary.