

SQLite

SQLite es una biblioteca en lenguaje C que implementa un motor de base de datos SQL pequeño, rápido, autónomo, de alta fiabilidad y completo. SQLite es el motor de base de datos más utilizado en el mundo. Está integrado en todos los teléfonos móviles y en la mayoría de las computadoras, y se incluye en innumerables aplicaciones que las personas usan

SQLite es una biblioteca en proceso que implementa un motor de base de datos transaccional de SQL auto contenido, **sin servidor**, de configuración cero. El código para SQLite está en el dominio público y, por lo tanto, es de uso gratuito para cualquier propósito, comercial o privado.

SQLite es un motor de base de datos SQL incorporado. A diferencia de la mayoría de las otras bases de datos SQL, SQLite no tiene un proceso de servidor separado. SQLite lee y escribe directamente en archivos de disco ordinarios.

¿Pueden varias aplicaciones o varias instancias de la misma aplicación acceder a un solo archivo de base de datos al mismo tiempo?

Varios procesos pueden tener la misma base de datos abierta al mismo tiempo. Múltiples procesos pueden estar haciendo un SELECT al mismo tiempo. Sin embargo, solo un proceso puede realizar cambios en la base de datos en cualquier momento. SQLite utiliza bloqueos de lectura / escritura para controlar el acceso a la base de datos.

SQLite permite que múltiples procesos tengan el archivo de base de datos abierto a la vez, y que múltiples procesos lean la base de datos a la vez. Cuando cualquier proceso quiera escribir, debe bloquear todo el archivo de la base de datos durante la actualización. Pero eso normalmente solo toma unos pocos milisegundos. Otros procesos solo esperan que el escritor termine y luego continúe con su negocio. Otros motores de base de datos SQL incorporados normalmente solo permiten que un solo proceso se conecte a la base de datos a la vez.



Small. Fast. Reliable.
Choose any three.

Home About Documentation Download License Support Purchase

Search

What Is SQLite?

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. [More Information...](#)

The SQLite [file format](#) is stable, cross-platform, and backwards compatible and the developers pledge to keep it that way through at least 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].

SQLite [source code](#) is in the [public domain](#) and is free to everyone to use for any purpose.

Latest Release

Version [3.27.2](#) (2019-02-25). [Download](#) [Prior Releases](#)

Common Links

- Features
- When to use SQLite
- Frequently Asked Questions
- Getting Started
- Prior Releases
- SQL Syntax
 - Pragma
 - SQL functions
 - Date & time functions
 - Aggregate functions
 - Window functions
 - JSON functions
- C/C++ Interface Spec
 - Introduction
 - List of C-language APIs
- The TCL Interface Spec
- Commit History
- Report a Bug
- News

Ongoing development and support of SQLite is made possible in part by [SQLite Consortium](#) members, including:



Small. Fast. Reliable.
Choose any three.

Home About Documentation Download License Support Purchase

Search

About SQLite

SQLite is an in-process library that implements a [self-contained, serverless, zero-configuration, transactional](#) SQL database engine. The code for SQLite is in the [public domain](#) and is thus free for use for any purpose, commercial or private. SQLite is the [most widely deployed](#) database in the world with more applications than we can count, including several [high-profile projects](#).

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database [file format](#) is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between [big-endian](#) and [little-endian](#) architectures. These features make SQLite a popular choice as an [Application File Format](#). SQLite database files are a [recommended storage format](#) by the US Library of Congress. Think of SQLite not as a replacement for [Oracle](#) but as a replacement for [fopen\(\)](#).

SQLite is a compact library. With all features enabled, the [library size](#) can be less than 600KiB, depending on the target platform and compiler optimization settings. (64-bit code is larger. And some compiler optimizations such as aggressive function inlining and loop unrolling can cause the object code to be much larger.) There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in low-memory environments. Depending on how it is used, SQLite can be [faster than direct filesystem I/O](#).

SQLite is [very carefully tested](#) prior to every release and has a reputation for being very reliable. Most of the SQLite source code is devoted purely to testing and verification. An automated test suite runs millions and millions of test cases involving hundreds of millions of individual SQL statements and achieves [100% branch test coverage](#). SQLite responds gracefully to memory allocation failures and disk I/O errors. Transactions are [ACID](#) even if interrupted by system crashes or power failures. All of this is verified by the automated tests using special test harnesses which simulate system failures. Of course, even with all this testing, there are still bugs. But unlike some similar projects (especially commercial competitors) SQLite is open and honest

Executive Summary

- [Full-featured SQL](#)
- [Billions and billions of deployments](#)
- [Single-file database](#)
- [Public domain source code](#)
- All source code in one file ([sqlite3.c](#))
- [Small footprint](#)
- Max DB size: [140 terabytes](#) (2⁴⁷ bytes)
- Max row size: [1 gigabyte](#)
- [Faster than direct file I/O](#)
- [Aviation-grade quality and testing](#)
- [Zero-configuration](#)
- [ACID transactions, even after power loss](#)
- [Stable, enduring file format](#)
- [Extensive, detailed documentation](#)
- [Long-term support](#)

Actividad 2 - Ejerc... SQLite Frequently... SQLite Home Page... About SQLite... Google Traductor... Acerca de SQLite... Google Traductor... + -

https://www.sqlite.org/faq.html#q5

equal to one another numerically (see the previous question); hence the values are not unique.

(5) Can multiple applications or multiple instances of the same application access a single database file at the same time?

Multiple processes can have the same database open at the same time. Multiple processes can be doing a SELECT at the same time. But only one process can be making changes to the database at any moment in time, however.

SQLite uses reader/writer locks to control access to the database. (Under Win95/98/ME which lacks support for reader/writer locks, a probabilistic simulation is used instead.) But use caution: this locking mechanism might not work correctly if the database file is kept on an NFS filesystem. This is because fcntl() file locking is broken on many NFS implementations. You should avoid putting SQLite database files on NFS if multiple processes might try to access the file at the same time. On Windows, Microsoft's documentation says that locking may not work under FAT filesystems if you are not running the Share.exe daemon. People who have a lot of experience with Windows tell me that file locking of network files is very buggy and is not dependable. If what they say is true, sharing an SQLite database between two or more Windows machines might cause unexpected problems.

We are aware of no other *embedded* SQL database engine that supports as much concurrency as SQLite. SQLite allows multiple processes to have the database file open at once, and for multiple processes to read the database at once. When any process wants to write, it must lock the entire database file for the duration of its update. But that normally only takes a few milliseconds. Other processes just wait on the writer to finish then continue about their business. Other embedded SQL database engines typically only allow a single process to connect to the database at once.

However, client/server database engines (such as PostgreSQL, MySQL, or Oracle) usually support a higher level of concurrency and allow multiple processes to be writing to the same database at the same time. This is possible in a client/server database because there is always a single well-controlled server process available to coordinate access. If your application has a need for a lot of concurrency, then you should consider using a client/server database. But experience suggests that most applications need much less concurrency than their designers imagine.

When SQLite tries to access a file that is locked by another process, the default behavior is to return SQLITE_BUSY. You can adjust this behavior from C code using the [sqlite3_busy_handler\(\)](#) or [sqlite3_busy_timeout\(\)](#) API functions.

(6) Is SQLite threadsafe?

[Threads are evil.](#) Avoid them.

SQLite is threadsafe. We make this concession since many users choose to ignore the advice given in the previous paragraph. But in order to be thread-safe, SQLite must be compiled with the SQLITE_THREADSAFE preprocessor macro set to 1. Both the Windows and Linux precompiled binaries in the distribution are compiled this way. If you are unsure if the SQLite library you are linking against is compiled to be threadsafe you can call the [sqlite3_threadsafe\(\)](#) interface to find out.

multiple Remarcar todo Sensible a mayúsculas Palabras completas (W) 1 de 11 coincidencias

08:52 a. m. 05/03/2019

Albar De La Torre García

Ivan Gamboa Ultreras