SQLite
Documentation
Not logged in
2015-11-19 07:58
HomeFilesTimelineBranchesTagsTicketsWikiLoginMore...

SQLite Source Repository

This repository contains the complete source code for the SQLite database engine. Some test scripts are also include. However, many other test scripts and most of the documentation are managed separately.

If you are reading this on a Git mirror someplace, you are doing it wrong. The official repository is better. Go there now.

Compiling

First create a directory in which to place the build products. It is recommended, but not required, that the build directory be separate from the source directory. Cd into the build directory and then from the build directory run the configure script found at the root of the source tree. Then run "make".

For example:

```
tar xzf sqlite.tar.gz    ;#  Unpack the source tree into "sqlite"
mkdir bld                ;#  Build will occur in a sibling directory
cd bld                   ;#  Change to the build directory
../sqlite/configure      ;#  Run the configure script
make                     ;#  Run the makefile.
make sqlite3.c           ;#  Build the "amalgamation" source file
make test                ;#  Run some tests (requires Tcl)
```

See the makefile for additional targets.

The configure script uses autoconf 2.61 and libtool. If the configure script does not work out for you, there is a generic makefile named "Makefile.linux-gcc" in the top directory of the source tree that you can copy and edit to suit your needs. Comments on the generic makefile show what changes are needed.

Using MSVC

On Windows, all applicable build products can be compiled with MSVC. First open the command prompt window associated with the desired compiler version (e.g. "Developer Command Prompt for VS2013"). Next, use NMAKE with the provided "Makefile.msc" to build one of the supported targets.

For example:

```
mkdir bld
cd bld
nmake /f Makefile.msc TOP=..\sqlite
nmake /f Makefile.msc sqlite3.c TOP=..\sqlite
nmake /f Makefile.msc sqlite3.dll TOP=..\sqlite
nmake /f Makefile.msc sqlite3.exe TOP=..\sqlite
nmake /f Makefile.msc test TOP=..\sqlite
```

There are several build options that can be set via the NMAKE command line. For example, to build for WinRT, simply add "FOR_WINRT=1" argument to the "sqlite3.dll" command line above. When debugging into the SQLite code, adding the "DEBUG=1" argument to one of the above command lines is recommended.

SQLite does not require Tcl to run, but a Tcl installation is required by the makefiles (including those for MSVC). SQLite contains a lot of generated code and Tcl is used to do much of that code generation. The makefiles also require AWK.

Source Code Tour

Most of the core source files are in the **src/** subdirectory. But src/ also contains files used to build the "testfixture" test harness; those file all begin with "test". And src/ contains the "shell.c" file which is the main program for the "sqlite3.exe" command-line shell and the "tclsqlite.c" file which implements the bindings to SQLite from the Tcl programming language. (Historical note: SQLite began as a Tcl extension and only later escaped to the wild as an independent library.)

Test scripts and programs are found in the **test/** subdirectory. There are other test suites for SQLite (see How SQLite Is Tested) but those other test suites are in separate source repositories.

The **ext/** subdirectory contains code for extensions. The Full-text search engine is in **ext/fts3**. The R-Tree engine is in **ext/rtree**. The **ext/misc** subdirectory contains a number of smaller, single-file extensions, such as a REGEXP operator.

The **tool/** subdirectory contains various scripts and programs used for building generated source code files or for testing or for generating accessory programs such as "sqlite3_analyzer(.exe)".

Generated Source Code Files

Several of the C-language source files used by SQLite are generated from other sources rather than being typed in manually by a programmer. This section will summarize those automatically-generated files. To create all of the automatically-generated files, simply run "make target_source". The "target_source" make target will create a subdirectory "tsrc/" and fill it with all the source files needed to build SQLite, both manually-edited files and automatically-generated files.

The SQLite interface is defined by the **sqlite3.h** header file, which is generated from src/sqlite.h.in, ./manifest.uuid, and ./VERSION. The Tcl script at tool/mksqlite3h.tcl does the conversion. The manifest.uuid file contains the SHA1 hash of the particular check-in and is used to generate the SQLITE_SOURCE_ID macro. The VERSION file contains the current SQLite version number. The sqlite3.h header is really just a copy of src/sqlite.h.in with the source-id and version number inserted at just the right spots. Note that comment text in the sqlite3.h file is used to generate much of the SQLite API documentation. The Tcl scripts used to generate that documentation are in a separate source repository.

The SQL language parser is **parse.c** which is generate from a grammar in the src/parse.y file. The conversion of "parse.y" into "parse.c" is done by the lemon LALR(1) parser generator. The source code for lemon is at tool/lemon.c. Lemon uses a template for generating its parser. A generic template is in tool/lempar.c, but SQLite uses a slightly modified template found in src/lempar.c.

Lemon also generates the **parse.h** header file, at the same time it generates parse.c. But the parse.h header file is modified further (to add additional symbols) using the ./addopcodes.awk AWK script.

The **opcodes.h** header file contains macros that define the numbers corresponding to opcodes in the "VDBE" virtual machine. The opcodes.h file is generated by the scanning the src/vdbe.c source file. The AWK script at ./mkopcodeh.awk does this scan and generates opcodes.h. A second AWK script, ./mkopcodec.awk, then scans opcodes.h to generate the **opcodes.c** source file, which contains a reverse mapping from opcode-number to opcode-name that is used for EXPLAIN output.

The **keywordhash.h** header file contains the definition of a hash table that maps SQL language keywords (ex: "CREATE", "SELECT", "INDEX", etc.) into the numeric codes used by the parse.c parser. The keywordhash.h file is generated by a C-language program at tool mkkeywordhash.c.

The Amalgamation

All of the individual C source code and header files (both manually-edited and automatically-generated) can be combined into a single big source file **sqlite3.c** called "the amalgamation". The amalgamation is the recommended way of using SQLite in a larger application. Combining all individual source code files into a single big source code file allows the C compiler to perform more cross-procedure analysis and generate better code. SQLite runs about 5% faster when compiled from the amalgamation versus when compiled from individual source files.

The amalgamation is generated from the tool/mksqlite3c.tcl Tcl script. First, all of the individual source files must be gathered into the tsrc/ subdirectory (using the equivalent of "make target_source") then the tool/mksqlite3c.tcl script is run to copy them all together in just the right order while resolving internal "#include" references.

The amalgamation source file is more than 100K lines long. Some symbolic debuggers (most notably MSVC) are unable to deal with files longer than 64K lines. To work around this, a separate Tcl script, tool/split-sqlite3c.tcl, can be run on the amalgamation to break it up into a single small C file called **sqlite3-all.c** that does #include on about five other files named **sqlite3-1.c**, **sqlite3-2.c**, ..., **sqlite3-5.c**. In this way, all of the source code is contained within a single translation unit so that the compiler can do extra cross-procedure optimization, but no individual source file exceeds 32K lines in length.

How It All Fits Together

SQLite is modular in design. See the architectural description for details. Other documents that are useful in (helping to understand how SQLite works include the file format description, the virtual machine that runs prepared statements, the description of how transactions work, and the overview of the query planner.

Unfortunately, years of effort have gone into optimizating SQLite, both for small size and high performance. And optimizations tend to result in complex code. So there is a lot of complexity in the SQLite implementation.

Key files:

- **sqlite.h.in** - This file defines the public interface to the SQLite library. Readers will need to be familiar with this interface before trying to understand how the library works internally.
- **sqliteInt.h** - this header file defines many of the data objects used internally by SQLite.
- **parse.y** - This file describes the LALR(1) grammer that SQLite uses to parse SQL statements, and the actions that are taken at each step in the parsing process.
- **vdbe.c** - This file implements the virtual machine that runs prepared statements. There are various helper files whose names begin with "vdbe". The VDBE has access to the vdbeInt.h header file which defines internal data objects. The rest of SQLite interacts with the VDBE through an interface defined by vdbe.h.
- **where.c** - This file analyzes the WHERE clause and generates virtual machine code to run queries efficiently. This file is sometimes called the "query optimizer". It has its own private header file, whereInt.h, that defines data objects used internally.
- **btree.c** - This file contains the implementation of the B-Tree storage engine used by SQLite.
- **pager.c** - This file contains the "pager" implementation, the module that implements transactions.
- **os_unix.c** and **os_win.c** - These two files implement the interface between SQLite and the underlying operating system using the run-time pluggable VFS interface.
- **shell.c** - This file is not part of the core SQLite library. This is the file that, when linked against sqlite3.a, generates the "sqlite3.exe" command-line shell.
- **tclsqlite.c** - This file implements the Tcl bindings for SQLite. It is not part of the core SQLite library. But as most of the tests in this repository are written in Tcl, the Tcl language bindings are important.

There are many other source files. Each has a suscinct header comment that describes its purpose and role within the larger system.

Contacts

The main SQLite webpage is http://www.sqlite.org/ with geographically distributed backup servers at http://www2.sqlite.org/ and http://www3.sqlite.org/.

This page was generated in about 0.02s by Fossil version 1.34 [3967d043e8] 2015-11-17 23:22:46