SQLite,
A Simple, Embeddable
Relational Database Engine

John H. Harris

1 February 2007

http://UnencumberedDesign.com
JHHarris@valley.net

Overview

SQLite Characteristics

SQLite for Systems

SQLite in Applications

Advanced Features

SQLite Characteristics

History

May 2000. D. Richard Hipp releases version 1.0 alpha.

Sep 2001. Version 2.0.

Oct 2003. I discover SQLite (2.8.7).

Sep 2004. Stable version 3.0.7 released.

Dec 2005. Last version 2 release: 2.8.17.

Oct 2006. Added full text search in version 3.3.7.

SQLite Features

- Small. ($<$ 250 kB). No dependencies.
- Simple. No administrative overhead. Untyped. Single data file with a portable format.
- True relational database. Rich subset of SQL92. ACID transactions.
- APIs in C, C++, Tcl. Many others (Python, ODBC, Java, Perl, PHP, etc.) available from second sources.
- Portable. Widely used, stable, solid.
- High performance. Fast. Large databases (10^{41} bytes).
- Open source (public domain).

SQLite Limitations

- No access control (i.e., *embeddable*).
- Foreign key constants must be implemented with triggers.
- No type checking.
- No scaled integers. No date and time types.
- No nested transactions.
- Other minor SQL limitations.

Where Does SQLite Fit In?

MySQL. Very high-performance server. Fashionable.

PostgreSQL. High-performance sever. Full SQL implementation. Safer data. Better ad hoc queries.

Firebird. Simpler, smaller server or bigger, fancier embeddable db. Limited APIs.

SQLite. Small, simple, embeddable (no access control), but true relational model, SQL.

Berkeley DB. Embeddable. Transaction-based, but not relational (no SQL). Rumored to be vulnerable to power failures.

Metakit. Embeddable. Not relational (no SQL). No transactions.

How Fast?

In tests of SQLite 2.7.6 versus PostgreSQL 7.1.3
and MySQL 3.23.41.

- SQLite was significantly faster (as much as 10 or 20×) than the default PostgreSQL for most common operations.
- SQLite was often faster (sometimes more than 2×) than MySQL for most common operations.
- SQLite did not execute `create index` or `drop table` as fast as the others.
- SQLite works best with operations grouped into a single transaction.

Who Uses SQLite?

America On Line, Apple Computer (OS X, Safari, Mail, etc.), Cisco Systems?, Google?, Mozilla, Palm OS, PHP 5, Sun Microsystems (Solaris), Trolltech (KDE), Unencumbered Design.

SQLite for Systems

The Missing UNIX Tool

- High performance.
- Safer than UNIX tools for related tables.
- Great way to learn SQL.
- System databases are more accessible.

Getting Started

From BASH,

```
sqlite3 my.db
```

From the SQLite shell prompt:

```
create table poot (a integer, b text);
insert into poot values (1, 'I did it.');
select * from poot;
.exit
```

Investigating a SQLite DB File

Use the sqlite3 admin tool to discover the schema:

```
select * from sqlite_master;
```

You get a result set:

`type`	Type of object (`table`, `index`, `view`).
`name`	Name of object.
`table_name`	Object's associated table.
`root_page`	Object's B-tree root page.
`sql`	Object's defining SQL code.

Troubleshooting a SQLite DB File

Use the sqlite_analyzer program to get memory use stats:

`sqlite_analyzer` *dbFile*

Use the `vacuum` command to collect garbage.

Attaching to a System DB File

Sqlite can join multiple db files:

```
attach database blort.db ;
```

Then join tables across databases:

```
select uid, name, address
  from blort.users, main.users
  where blort.users.id = uid;
```

SQLite in Applications

Why Embed a Database?

- You need a database.
- A *simple* back end for a web server.
- Improve performance for client-server applications.
- Persistent data. No need to "save."
- Implement infinite undo, redo.
- Transactions protect data from corruption.

Why a *Relational* Database?

- Normalized data structures are stable.
- Normalized data structures minimize redundancy, and hence, bugs.
- Simple data integrity protects data from GUI bugs.
- Simple, declarative data integrity code protects data from GUI bugs.
- Ad hoc queries are a powerful diagnostic tool.

Getting Started, from Tcl

From tclsh,

```
package require sqlite3
sqlite3 db my.db

db eval {create table poot (a integer, b text);
   insert into poot values (1, 'I did it.')}

db eval {select * from poot}
exit
```

Advanced Features

Full Text Searches. Like Google searches.

Define Functions with Tcl.

Database in RAM. Fast.

Full Text Searches

```
create virtual table photos using fts1 ( id, content );
```

Insert your data, then

```
select * from photos where content match 'fox owl';

select * from photos where content match 'fox OR moose';

select * from photos where content match '"bird dog"';
```

Defining Functions in Tcl

To convert data formats:

```
proc iso8601 { t } {
   clock format $t -format %Y-%m-%dT%T }
db function iDate iso8601
db eval {select iDate(modified)}
```

To executing data as code:

```
db function tcl eval
db eval {select tcl('set x 5')}
```

Database in RAM

Just leave off the file name.

+ It's fast—no seeking.

– Data must me small enough to fit in RAM.

– *Not* persistent.

Resources

- SQLite web site, `http://www.sqlite.org/`
- Michael Owens. *The Definitive Guide to SQLite*. 2006. Berkeley CA: Apress.
- Any introductory SQL book.
- D. Richard Hipp, HWACI Applied Software Research, (704) 948 4565, DRH@hwaci.com.
- John H. Harris, Unencumbered Design, (802) 649 8130, JHHarris@valley.net.

rev 0a