International Journal of Allied Practice, Research and Review
Website: www.ijaprr.com (ISSN 2350-1294)

Sqlite- A Better Choice as an Embedded Database for Smart Phones

Iqra Jan[1] and Junaida Shafi[2]
[1]CSE Department, Kurukshetra University, Kurukshetra,, Haryana
[2] CSE Department, Kurukshetra University, Kurukshetra, Haryana

Abstract - Nowadays, embedded database technology has become a very popular research field; various embedded database soft ware's have been launched in the market. Each of these products have different characteristics based on their specific technical specifications. SQLite, an embedded database, is very popular open source embedded database software used for local/client storage. It is widely deployed database engine in smart phones. SQLite is embedded in every android device and it provides support to database operations in android via android inbuilt APIs. This paper describes the SQLite embedded database, its features, users and its architecture.

Keywords: ***SQLite; Embedded Systems; smart phone; database; server less***

I. Introduction

An embedded database system can be defined as a database management system integrated with application software that would be able to access stored data such that the database system remains hidden from the end user of an application and that requires little or no ongoing maintenance. This means that an embedded database does not run in a separate process, instead embedded databases are directly linked to an application requiring access to stored data. This is in contrast to conventional database management systems (such as SQL server), which runs as a separate process and the application connects to the database management system using APIs (Application programming interfaces)

Embedded databases have become the focus of study currently, with the popularity of intelligent appliances, the formation of mobile computing environment, and the rise of mobile commerce. The key benefit of embedded database systems lies in their availability and simplicity of administration. Also there is no need to take extraordinary permissions to connect to the database process, as the data is kept in ordinary files in the user's space.

So far many different types of embedded databases have been developed, among them some of the popular embedded database products are: - Advantage database Server (from Sybase), Infinity DB from Boiler Bay Inc, Informix Dynamic server (from IBM), Extensible Storage Engine (from Microsoft), InnoDB from Oracle Corporation, Solid DB, SQLServer Compact from Microsoft Corporation, etc and SQLite which is my area of study. All these products have their own characteristics and uses. However SQLite has strongly attracted majority of developers for its advantages as simplicity, efficiency, reliability, lightweight, independence etc.

SQLite is one of the popular open source embedded database. SQLite is local database management system used to store data for individual applications and devices. It emphasizes economy, efficiency, reliability, independence and simplicity. Unlike client server database management system, the SQLite engine has no separate processes through which an application communicates. The SQLite library is associated with an application program and thus becomes an essential part of an application program. The application program uses the functionality of SQLite through simple function calls that result in reduction of latency in database access. Using function calls within a single process is more efficient than inter process communication. SQLite uses relational model and is thus also called relational database management system. It is most widely deployed database engine used today by several browsers, operating systems and embedded systems. It was designed by D.Richard Hipp in the year 2000. In August 2000 SQLite version 1 was released with storage based on GNU Database Manager. After that SQLite 2.0 was released that replaced GNU database with a custom B-tree implementation, thus adding transaction capability. Then SQLite 3.0 added manifest typing, internationalization and other major improvements. SQLite has bindings for large number of languages like C, C++, C#, PHP, Visual basic etc.

Following is the list of different SQLite versions used currently and the different android versions for which they are suitable:

SQLite 3.8.6:
- 22-5.1-Lollipop

SQLite 3.8.4.3:
- 21-5.0-Lollipop
- 20-Android L Developer Preview

SQLite 3.7.11:
- 19-4.4-KitKat
- 18-4.3-Jelly Bean
- 17-4.2-Jelly Bean
- 16-4.1-Jelly Bean

SQLite 3.7.4:
- 15-4.0.3-Ice Cream Sandwich
- 14-4.0-Ice Cream Sandwich
- 13-3.2-Honeycomb
- 12-3.1-Honeycomb
- 11-3.0-Honeycomb

SQLite 3.6.22:
- 10-2.3.3-Gingerbread
- 9-2.3.1-Gingerbread
- 8-2.2-Froyo

SQLite 3.5.9:
- 7-2.1-Eclair
- 4-1.6-Donut
- 3-1.5-Cupcake

II. Features of SQLite

SQLite is self contained: SQLite requires very minimum support from operating system or from external libraries. Because of this feature SQLite is well suited for embedded devices that lack the support and infrastructure of a desktop computer and thus becomes suitable for use within applications that need to run without alteration on a large variety of computers of varying configurations.

SQLite is server less: Client/Server SQL database engines like SQL Server, Oracle, and MySQL etc are implemented as a separate server process. Using inter-process communication, programs that wish to access the database communicate with the server to send requests to the server and to receive results back from the server. On the other hand, SQLite does not work this way. Unlike client/server SQL database engines, in SQLite, the process that wishes to read and write the database directly from the database files on disk. No intermediatory server process is required. This feature has both merits as well as demerits. The main merit is that there is no overhead of installing, configuring, initializing, managing, setting up and troubleshooting a separate server process. There is no need for an administrative support for the programs that use SQLite for setting up the database engine before they are run. Program with the ability to access the disk is able to use SQLite database also. In contrast, a database engine that uses a server can offer better protection from errors in the client application. SQLite has the ability to control database access with more precision and thus allow for better concurrency and grain locking.

SQLite is zero configuration databases: SqLite doesn't require any installation before it is used. No "setup" procedure is required. There is no need to start, stop, or configure a server process. An administrator doesn't need to create a new database instance or assign access permissions to users. No configuration files are used in SQLite. Nothing needs to be done to tell the system that SqLite is running. No actions are required to recover after a system crash or power failure. Nothing needs to be troubleshooted.

SQLite is Transactional: A transactional database is one in which all changes and queries appear to be atomic, consistent, isolated and durable. SQLite is ACID compliant.

Atomicity: Everything in the transaction succeeds lest it is rolled back.

Consistency: A transaction cannot leave the database in an inconsistent state.

Isolation: One transaction cannot interfere with another.

Durability: A completed transaction persists, even after applications restart.

SQLite's database files are cross-platform: we can use a SQLite database file created on any laptop in our laptop as if it has been created on our laptop. It might be useful to draw the database file from the device (or our emulator) and run queries from within our development machine, especially if we would like to make use of tools with a graphical user interface.

SQLite has no fixed column length: Unlike other relational database management systems e.g., SQL Server, Oracle etc, in SQLite any TEXT value is of any length. There is no restriction on the size of TEXT. If we want to enforce restrictions, we have to add them through coding. SQLite won't help us. This feature is useful if we large sized strings or numbers.

All data is stored in one single database file: In SQLite all database contents are stored in a single file. Main data, indices, triggers and any Meta data needed by SQLite are contained in a single file. Journal file which is used during transactions is added in newer versions.

III. Well Known Users of SQLite

- SQLite is used by Adobe as the application file format for their Photoshop Light room product. SQLite is also used for the Adobe Integrated Runtime (AIR). Acrobat Reader also uses SQLite.

- SQLite is being used in the flight software for the A350 XWB family of aircraft as confirmed by Airbus.

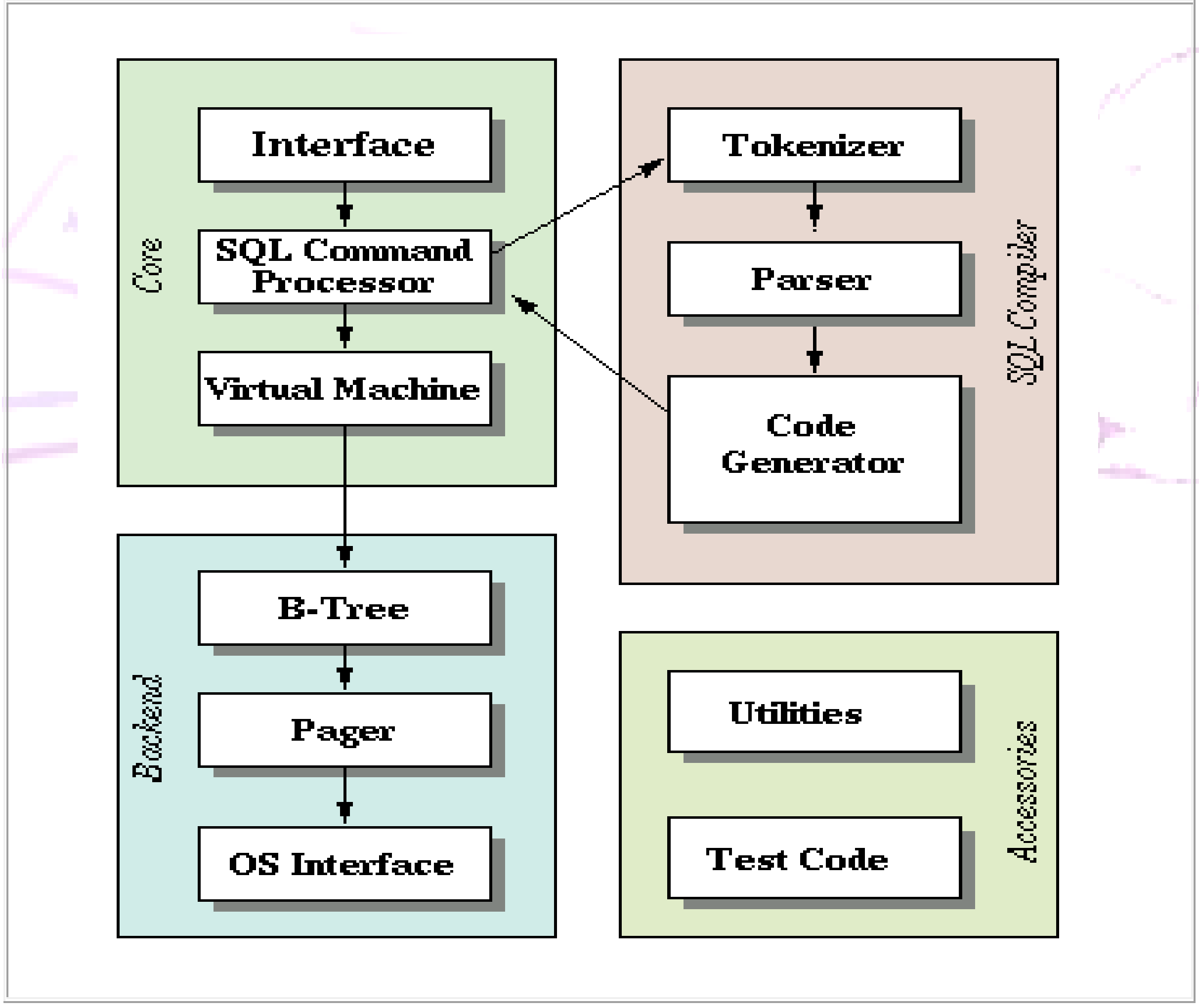
- SQLite is used by Apple for many functions within Mac OS X, including Apple mail, Safari, and in Aperture. Apple uses SQLite in the iPod touch and in the iPhone and in iTunes software also.

- SQLite is used as the primary data store on the client side for Drop box synchronization and file archiving service.

- SQLite is used as a primary meta-data storage format for the Firefox web browser and the Thunderbird Email Reader by Mozilla.
- Flame, a malware spy program is reported to make intense use of SQLite.
- SQLite is used by McAfee in its antivirus programs.
- There are several sightings of SQLite in the Skype client for Mac OS X and Windows.
- Google uses SQLite in their Desktop for Mac, in Google Gears, in the Chrome Web Browser, and in the Android cell-phone operating system.

IV. SQLite Architecture

Block Diagram Of SQLite



- **Interface:**

  SQLite interface is the point of communication for other programs and scripts. A large amount of the public interface to the SQLite library is implemented by functions found in the **main.c**, **legacy.c**, and vdbeapi.c source files while some routines are spread about in other files where they can have access to data structures with file scope. For example the sqlite3_get_table () routine is implemented in **table.c**. sqlite3_mprintf () is found inprintf.c. sqlite3_complete () is in **tokenize.c**. The Tcl interface is implemented by **tclsqlite.c** etc.

- **Tokenizer:**

Tokenizer breaks the original string into tokens and passes those tokens to the parser one by one. String to be executed; containing SQL statements is passed to the tokenizer by an interface. The tokenizer is hand-coded in C in the file **tokenize.c**. Parser for SQLite is generated by the Lemon parser generator which is faster than YACC and BISON.

- **Parser:**

The parser assigns meaning to the tokens based on their context. Lemon LALR (1) parser generator is used to generate the parser for SQLite. Lemon works in the same way as YACC or BISON, but it uses a different input syntax which is less error-prone. Lemon also generates a thread-safe and reentrant parser. For preventing the memory leakage when syntax errors are encountered lemon defines the concept of a non-terminal destructor. The source file that drives Lemon is found in **parse.y**.

- **Code Generator:**

After assembling the tokens into complete SQL statements, parser calls the code generator to produce virtual machine code to perform the work requested by SQL statements. Code generator includes followingfiles: **attach.c**, **auth.c**, **build.c**, **delete.c**, **expr.c**, **insert.c**, **pragma.c**, **select.c**, **trigger.c**, **update.c**, **vacuum.c** and **where.c**.

Expr.c handles code generation for expressions. **Where.c** handles code generation for WHERE clauses on SELECT, UPDATE and DELETE statements.

The files **attach.c**, **delete.c**, **insert.c**, **select.c**, **trigger.c** **update.c** and **vacuum.c** handle the code generation for SQL statements with the same names. All other SQL statements are coded out of **build.c**. The **auth.c** file implements the functionality of **sqlite3_set_authorizer ()**.

- **Virtual Machine:**

Virtual machine executes the program generated by code generator. Virtual machine implements an abstract computing engine specially designed for manipulating the database files. Virtual machine consists of a stack that is used for intermediate storage. Each program instruction contain an opcode and up to three additional operands. The virtual machine is itself completely contained in a single source file **vdbe.c**. The virtual machine also has its own header files: **vdbe.h** that defines an interface between the virtual machine and the rest of the SQLite library and **vdbeInt.h** which defines structure of the virtual machine. The **vdbeaux.c** file contains utilities used by the virtual machine and interface modules used by the rest of the library to construct virtual machine programs. The **vdbeapi.c** file contains external interfaces to the virtual machine such as the **sqlite3_bind_...** family of functions. Strings, BLOBs, integers, and floating point numbers are stored in an internal object named "Mem" which is implemented by **vdbemem.c**.

Callbacks to C-language routines are used to implement SQLite. Built-in SQL functions are also implemented the same way. The majority of built-in SQL functions such as **coalesce ()**, **count ()**, **substring ()** etc can be found in **func.c**. Date and time conversion functions are found in **date.c**.

- **B-Tree:**

B-Tree implementation is used to maintain SQLite databases on disk. B-tree implementation is found in the **btree.c** source file. The interface to the B-tree subsystem is defined by the header file **btree.h**. A separate B-tree is used for each table and index in the database. All B-trees are stored in the same disk file.

- **Page Cache:**

  The page cache is used for reading, writing, and caching the fixed-size chunks of information from the disk requested by B-tree module. The default size of chunk is 1024 bytes but can vary between 512 and 65536 bytes. The page cache also provides the rollback and atomic commit abstraction and performs locking of the database file. Particular pages from the page cache are requested by B-tree. B-tree notifies the page cache when it wishes to modify pages or commit or rollback changes. The page cache handles all the messy details of making sure the requests are handled quickly, safely, and efficiently. Code used for implementing the page cache is contained in the single C source file **pager.c**. The interface to the page cache subsystem is defined by the header file **pager.h**.

- **OS Interface:**

  SQLite uses an abstraction layer to interface with the operating system to provide portability between POSIX and win32 operating systems. The interface to the OS abstraction layer is defined in **os.h header file**. Each supported operating system has its own implementation like **os_unix.c** for UNIX, **os_win.c** for Windows, and so forth. Each of these operating-specific implements has its own header file: **os_unix.h**, **os_win.h**, etc.

- **Utilities:**

  Memory allocation and case less string comparison routines are located in **util.c**. Hash tables found in **hash.c** maintain symbol tables used by the parser. The **utf.c** source file contains Unicode conversion subroutines. SQLite has its own private implementation of **printf ()** (with some extensions) in **printf.c** and its own random number generator in **random.c**.

- **Test Code:**

  If we count regression test scripts, more than half the total code base of SQLite is devoted to testing. There are many **assert ()** statements in the main code files. In additional, the source files **test1.c** through **test5.c** together with **md5.c** implement extensions used for testing purposes only. The **os_test.c** backend interface is used to simulate power failures to verify the crash-recovery mechanism in the pager.

V. Conclusion

Some of these options are Phone’s SD card, Phone’s internal memory etc. Sometimes, however, we need to be able to carry out complex operations on persistent data. Modern e-business is based on mobile applications. Recent advances in device technology and connectivity have forced different organizations and vendors to allow the users to access their applications via their smart phones also. This requires the synchronization of data between the smart phones and the actual server. This is where a smart phone needs to have a database, but the database should be light weight, requiring no configuration and so on. All of these features and many more suitable for smart (android) phones are provided by SQLite. SQLite consumes very little memory in android phones. SQLite memory footprint starts at about 50 kilobytes. In android phones memory per process is limited compared to desktop systems so database should be such that doesn’t add too much burden to memory consumption of our android app, so using SQLite is the better choice. Furthermore, SQLite is server less system that makes the handling of database much easier as there is no need for configuration files or the commands used to configure the server. SQLite is focused on simplicity. If we are looking for portability, simplicity, speed and small memory footprint—Sqlite is ideal. SQLite is very simple and fast open source SQL engine.

VI. References

[1] Jay A. Kreibich “Using SQLite”, O Reilly August 2010, ISBN: 978-0-596-52118-9.

[2] Kiran Dhokale, Nandeo Bange, Shelke Pradeep, Sachin Malave “Implementation of SQL Server based on SQLite Engine on Android Platform”, International Journal of Research in Engineering and Technology, Volume:03, Issue:04 ,April 2014.

[3] Kun Yue, Linying Jiang, Lin Yang, Heming Pang “Research of Embedded Database SQLite Application in Intelligent Remote Monitoring Systems”, IEEE 2010.

[4] Chunyue Bi “Research and Application of SQLite Embedded Database Technology”, Issue 1, Volume 8, January 2009 ISSN: 1109-2750.

[5] Jaroslav Pokorny “Databases in the 3rd Millennium: Trends and Research directions”, Journal of System Integration 2010.

[6] Margo Seltzer, Michael Olson “Challenges in Embedded Database System Administration”, “Proceedings of the Embedded Systems Workshop”, Cambridge, Massachusetts, USA, March 29–31, 1999.

[7] Sunguk Lee, “Creating and Using Database for Android Applications”, “International Journal of database Theory and Application”, vol.5, No.2, June 2012.

[8] Lv Junyan, Xu Shigno, Li Yijie, “Application Research of Embedded Database SQLite”, International Forum on Information Technology and Applications, 2009.

[9] Ming Xu, XinChun Yin, Jing Rong, “Researchment and Realization Based on Android Database Application Technology”, Proceedings of 2nd International Symposium on Computer, Communication, Control and Automation (ISCCCA-13), 2013.

[10] Weibo Li, Hong Yang, Ping He, “The research and application of embedded mobile database”, International Conference on Information Technology and Computer Science, 2009.