International Journal of Computer Trends and Technology
ISSN: 2231-2803 / https://doi.org/10.14445/22312803/IJCTT-V67I2P101

Volume 67 Issue 2, 1-6, February 2019
© 2019 Seventh Sense Research Group®

Original Article

SQLite Database and its Application on Embedded Platform

Y.V. Sai Bharadwaj, Sai Bhageerath Yarrapatruni, Prasada Rao YVSSSV

School of Computer Science and Information Technology, University of Hyderabad
National Institute of Technology, Warangal
NRI Institute of Technology, Guntur, AP.

Abstract - *SQLite is an embedded database system popularly used in mobile devices, remote sensing, control systems, information appliances, etc. It has become the focal point for development works in related areas. Reliability, robustness, security, fast and high efficiency, flexibility, and so on are unique among many other embedded databases. This paper outlines the basic features, structure, and key technologies of main embedded databases and analyses the characteristics, architecture, and interface functions of SQLite. Also, a detailed porting process from SQLite to the ARM-Linux platform is discussed, and a development case on the home gateway over ARM-Linux is examined.*

Keywords - *Embedded Database; SQLite; Porting; ARM-Linux; Home Gateway*

I. INTRODUCTION

With the advancement in information technology, a new development space for embedded database technology has opened up according to the requirements of the mobile database system. With the advancement of smart machines like Mobile Devices, Information Appliances, etc., embedded database technology has entered into the application domain from its earlier research vertical. Numerous variants of embedded systems and embedded database technology are being worked on. Because of the overreach of mobile devices, the demand for real-time data processing and data management has become the need of the hour. This demand has drawn an efficient embedded database application into an accelerating development phase.

Recently, many database products have appeared in the international arena, such as SQL Server, Oracle, IBM DB2, Sybase, etc. However, the embedded systems' power utilization and memory consumption restrict these products from their full potential, especially when the volume of mobile users becomes elephantine. The traditional databases utilize enormous space and are much slower in real-time, while on the other hand, embedded databases show exceptional performance in the concerned vertical. Many embedded databases have surfaced in both the domestic and international markets, which has eased the development of various embedded devices. Domestic products include TinyDB, ZODB are developed in python, and International products like OracleLite of Oracle, DB2 and DB2 Satellite of IBM, and Sybase iAnywhere of Sybase.

Embedded databases based on the Linux open-source platform like Berkeley DB, Mysql, SQLite, etc., that provide API's which are easy to use, aid the management and access of data and encourage the development of open-source software. Adding to the emergence of research and application in some verticals like mobile devices, Communication Systems, Automatic Test Systems (ATS), etc., the development of efficient embedded databases will become further important.

In response to the requirement emerging in this hot reached vertical, major database vendors have developed their products (embedded databases). Each has its technical specifications and characteristics. Among these, SQLite has strongly attracted most developers due to its lightweight, portability, no copyright constraints, and so on.

II. FEATURES, FRAMEWORK, ARCHITECTURE, AND THE KEY TECHNOLOGIES OF EDS

A. Features of Embedded Database Systems

The architecture defines an embedded database system [1]. It is a DMS that supports mobile computing, is integrated with specific Mobile Application Software and embedded Operating systems, and runs on an embedded or mobile device. It is also known as an embedded mobile database as it involves mobile computing. Embedded database technology combines several fields like the real-time system, mobile communication, database, distributed computing, etc. This has developed into a new area of research in database technology. The structure of an Embedded System is shown in Fig.1.

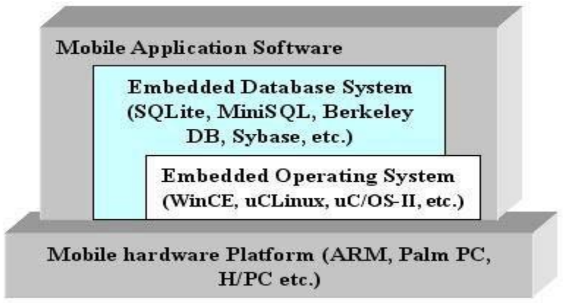
Mobile Application Software
Embedded Database System (SQLite, MiniSQL, Berkeley DB, Sybase, etc.)
Embedded Operating System (WinCE, uCLinux, uC/OS-II, etc.)
Mobile hardware Platform (ARM, Palm PC, H/PC etc.)

Fig. 1 Embedded System Framework

An Embedded database, like middleware, is bound to be restricted by various factors like speed, application, and other resources. An Embedded database system is very much similar to a traditional database system, which can be a hierarchical, network, relational, and object-oriented database. The typical distinction between the embedded database and a traditional database is that: an embedded database is procedure-driven, whereas a traditional database is response-driven (i.e., in an embedded database, a respective API is called to access data, while in a traditional database engine response approach is used). An embedded database can't be considered a small-scale traditional database, as an embedded database system and a traditional database system are not the same in the running environment and have their characteristics [2][3][4].

(1) Less space and memory requirement

The scale of the data structure (such as data tables and records) is limited, and less disk space is consumed. The memory utilization of an embedded system is also very little, so an embedded database can run infinite memory space.

(2) Reliability, robustness, and security

As an embedded database is generally used in a mobile environment, reliability, data integrity, and security are necessary.

(3) Interoperability and portability

Generally, an embedded database is specific to the application platform and the operating system, so to ensure hassle-free communication with other databases, interoperability is of great concern in the development process; at the same time, portability plays a major role in an embedded database (as OS and hardware platform used in various embedded systems are different), which should be considered during the design phase.

(4) Scalability

An embedded database must be scalable to reduce disk space utilization.

B. The Architecture of EDS

Similar to a traditional database, there are three levels in an embedded DB: the internal level, conceptual level, and external level [5], which is shown in Fig.2.

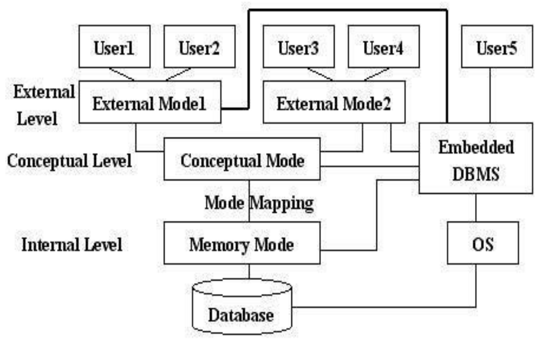

Fig. 2 Architecture of Embedded database System

The conceptual level provided a logical illustration of DB. The external level is an interface between the database system and users. Based on the structure of the external level, programmers store and operate data and are not concerned with the logical illustration. The internal (memory) level defines the physical structure like the data types, indexes, methods of data control, and so on. The conceptual level is a middleware and restricts both the external and internal levels.

Based on the need for Real-Time, the conceptual level can be bypassed (if real-time demand is high) and directly deal with the physical data at the internal (memory) level.

Embedded Database Management System is a software system that manages the data in the embedded database system. It consists of three program modules: the language compiler processing programs, the system running control program, and the service program. All the operations on embedded DB are executed through the Embedded DBMS. The three modules are described here [5]:

(1) Language compiler processing program: It includes language processing and compiler processing of DML at all levels.

(2) The system control program: It is used to control the processes in the program that include real-time scheduling, data access, control and storage into the database, concurrent control program, etc.

(3) Service-oriented program: It included the error recovery process, DB organization process, etc.

C. The Key Technologies of EDS

The design of an embedded database is balanced with the application system (Front end) and the data, which is a sub-set of the dataset (in the back end server). So, for a robust, reliable, and secured embedded database, various key technologies must be considered during the design phase [6].

(1) Replication and Synchronization

Data replication is employed when mapping with the DB at the back end server to access data at any time. This replication of data necessitates synchronization between front-end applications and backend servers.

(2) Backup recovery

The backup process in an embedded DB is different than that of a usual DB. It follows the procedure and is not independent of the service.

(3) Transaction processing

Processing of transactions in an embedded DBS is rationalized in the front-end. As accessing required data does not necessitate hitting the backend DB every time, this helps reduce the number of hits onto the backend server, saves time, and improves performance. But the entire application system may need to be combined with the characteristics of a mobile computing environment to control and deal with transactions.

(4) Security

Granting access control is the utmost need for embedded devices that is highly portable and mobile. The database system is precise with authority access.

(5) Quick start-up of the system

There is a greater risk of fault occurrence in an embedded system, so a Quick start-up is ensured to list every process running; this helps in the case of non-correction of Software Error.

(6) Real-time processing

To avoid processing delay, real-time processing is to be employed, and this forms the backbone requirement of an embedded system.

III. DATA STRUCTURES, ARCHITECTURE OF SQLITE, AND ITS API FUNCTIONALITIES

A. Features and Data Structure of SQLite

1. Features of SQLite

SQLite is an open-source light-weighted DB. It is written in C language. Compared to the traditional databases like Oracle, SQL Server, etc., SQLite does not require additional components (as it encompasses a complete embedded DB engine). It is suited for embedded application development and possesses many benefits; and hence it is preferred over many other embedded DB systems in the research and development domain. The main features are as follows [7][8][9]:

(1) The open library of this embedded DB is implemented in not more than 30,000 lines of C code. In addition to this, its DB is compatible with binary formats and scales up to 2 terabytes in size. As this is open-sourced, it minimizes the production costs.

(2) Autoconfiguration and execution is an enabling feature of SQLite, as it does not require any thread to start and stop. Neither does it require creating DB nor distributing access authority by the admin. At the time of system collapse, the restoration process involved in SQLite is automatic. Also, it provides an easy-to-use API and accesses the DB directly through API functions. It also supports advanced languages.

(3) SQLite can directly access the database files on the hard disk without calling an additional service. Interoperability enables the use of the same database files on different machines.

(4) SQLite does not differentiate data type, i.e., it can assign any data to any column of any table, no matter how the data is declared on the hard disk.

(5) SQLite enforces ACID properties and does not leave the system vulnerable during the unplanned collapse.

(6) SQLite is quick, scalable, and with high throughput. As it is platform-independent, it can be used on various embedded OS, like uC Linux, Windows CE, etc.

(7) It supports major programming languages, including C/C++, PHP, Perl, etc. These languages interact with the DB using an API call.

(8) SQLite employees more than 90% test coverage (i.e., test cases covering the complete application code).

2. Data Structure of SQLite

SQLite has several data structures to use in the design of the program. They are as follows:

(1) In the program design, SQLite * XXX defines a data structure pointing to a DB. SQLite is a data structure pointing to a database.

(2) In the program design, sqlite_stmt *XXX defines a handle pointing to some SQL statements. Sqlite_stmt is a data structure that includes byte codes compiled by SQL statements and all the necessary resources which execute the byte codes.

(3) Sqlite_value is used to hold values of dynamic data type.

B. Analysis of SQLite Architecture

1. SQLite Architecture

Modular design is facilitated for up-gradations, and the complete DB is divided into several modules. The 8 primary subsystems are depicted in Fig 3[8]. The architecture of SQLite constitutes three parts: Core, SQL Compiler, and BackendBackend. In addition to this, Accessories are also included.

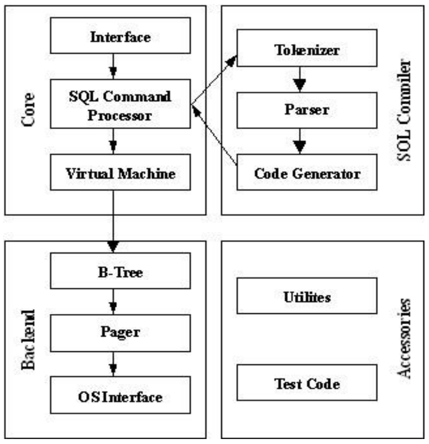

Fig. 3 SQLite Architecture

The topmost level of SQLite interface is implemented in the ANSI C library; even though API's in different languages are used, the C library is executed at the bottom. The received SQL statements from the interface are decomposed into various identifiers by the tokenizer. Then the parser (lemon analyzer) recomposes the identifiers and inputs the result to the code generator to produce VM code. The VM carries the VM code and finally completes the task of the SQL statement.

SQL command processor has three independent components: Tokenizer, Parser, and Code Generator. The interface transfers these strings containing SQL statements to the tokenizer. The tokenizer's job is to break the original string up into tokens and pass these broken tokens one by one to the parser. The Tokenizer and Parser transform the SQL statements into data structures handled at the bottom layer. The parser generator produces a parse tree by recombining the tokens and transferring the parse tree to the code generator.

VM is the most important component of the internal structure of SQLite, which is called the Virtual Database Engine (VDBE), which is an engine designed to deal with library files. It performs operations like data manipulation and acts as an interface between the client and backend storage. VDBE instructions are designated to complete various database operations, like insertion, deletion, querying, transaction processing, etc. [9]. It can also perform stack operations. The instruction sets of Virtual Database Engine analyze any SQL statement, first transforms it into corresponding virtual machine statements, and then complete the tasks given by the SQL statements.

The sole work of the Code generator is to convert the parse tree into assembly language and then transfer it to the virtual machine for execution.

The Virtual Database Engine executes the series of instructions of the mini-program (In assembly code) one by one and fulfills the request as specified in the SQL statement.

BackendBackend includes three parts, as depicted in Fig.3 [10][11]. The role of the B-tree and pager is to transfer the data blocks to the OS interface. B-tree is tasked to order the data blocks to ensure the relation among the data pages is easy to locate. The data blocks are stored in the form of B-Tree in the discs. The task of the pager is for data management, collapse recovery, etc.; this requires collaboration with the OS interface.

OS interface provides a unified interface for porting onto OS. As different OS use different methods to lock files, the task of OSI is to shield out these differences and provide an abstract layer for the other components of SQLite.

The architecture of SQLite, as depicted in Fig 3, also includes Accessories, namely Utilities and Test code.

Utilities mainly perform no data type string comparison, allocate memory, LEX functions, symbol table storage, functionalities of printing, and the random number function of SQLite.

Test Code: More than half of SQLite functions can be tested if regression testing is counted on. The fault recovery mechanism is simulated during the system recovery using os_test.c.

2. SQL statement Execution using SQLite

SQLite is used to implement SQL statements; this process involves three stages: Prepare, Step and Finalize, which is shown in Fig.4.

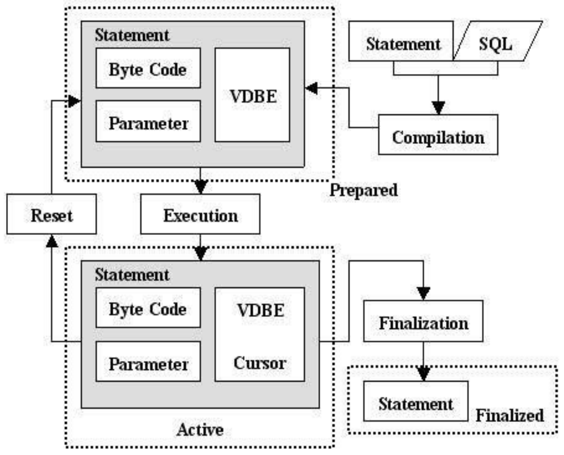

Fig. 4 SQL statement Execution Flow

(1) Preparation stage: Virtual Database Engine byte code is formed by compiling the SQL statements using the parser, tokenizer, and code generator in the API function written in C. The compiler associates with the sqlite_prepare() function and generates sqlite_stmt. sqlite_stmt includes Byte code, all the necessary resources required by executing SQL statements, etc.

(2) Execution stage: Virtual DB Engine implements the Byte-codes included in the sqlite_stmt step-by-step. This step-by-step process is completed by the sqlite_step().

(3) Finalization stage: The execution of the SQL statement is still under process by the Virtual DB Engine. Resources are released during this time. sqlite_finalize() determines this process.

As shown in the above diagram, prepared, active, and finalized represent the three stages of the implementation. They imply the following: prepared means the Virtual DB Engine Byte Codes are prepared by compiling all the SQL statements. Active means that the obtained Byte Codes are executed step by step using the sqlite_step() function. Finalized means that all the related resources are released after the end of the execution process.

C. SQLite API Functionalities

SQLite uses many API functions, which perform various basic functions on the database; these are used for the establishment of the table, querying,

modification, insertion, deletion, sorting, etc. The core functions of API are four [12] which are listed herewith: sqlite_open (), sqlite_exec (), sqlite_close (), and sqlite_errcode(), which are used to execute SQL and acquire data, and realize efficient data storage and management. In addition, it is extensible, allowing the programmers to custom functions and pass them on in the form of Callback.
sqlite_open(): This establishes the SQLite engine, where access mode and filename are provided. Then a callback function is implemented by SQLite for each record from the DB.
sqlite_exec(): This deals with the result fetched after executing the SQL statement.
sqlite_close (): This function closes the DB files and releases the SQLite engine. The following descriptions are the composition of four API functions.

(1) Sqlite_open ():

This function is used to open the database and establish the SQLite engine. There are various standards for writing this function based on the UTF (Unicode Transformation Format) used. In the case of UTF-8 encoding format, we use:
*intsqlite_open(const char *filename, sqlite3 **ppDb);*

But for UTF-16 encoding format we use:
*int sqlite3_open16 (const void *filename, sqlite3 **ppDb)*

The parameters used are FILENAME (name of the file) and PATH. For permission on access mode, we use two additional parameters:
*int sqlite3_open_v2 (const char *filename, sqlite3 **ppDb, int flags, const char *zVfs)*

ACCESS MODE determines (Read-only/Read-Write/Create). If the file exists, SQLite_OK is returned by the OPEN function, and ppDb parameter returns a legal database handle. If the file does not exist, it will establish the DB connection with the SQLite engine, and abnormal code is returned.

(2) Sqlite_close():
*intsqlite_close (sqlite3 *db)*
This function shuts down the already opened DB connection with the SQLite engine. The db parameter determines the handle of the DB being closed.

(3) Sqlite_exec ():
*intsqlite_exec (sqlite * db,char* sql,int(*Callback)(void*,int,char **,char**),void *parg,char **errmsg);*
SQL queries are dealt with this function which includes five parameters: database structure pointer, SQL statement string, the first pointer point to Callback functions, the first parameter pointer of Callback, and the pointer to error string.

(4)
*intsqlite_errcode(sqlite*db):*

This Error handling function returns the error code. But for obtaining the error information we use:
*const char * sqlite_errmsg(sqlite *); [for UTF-8]*
*const char * sqlite_errmsg16(sqlite *)[for UTF-16]*

IV. PORTING EDS (SQLITE) ONTO ARM-LINUX PLATFORM

Being an open-sourced platform and advantaged with the lightweight feature, SQLite has been the most widely used EDS. It operates on three levels: external level, logic level, and internal (storage) level. SQLite is ported onto the ARM-Linux platform to make DB applications run on the ARM-Linux platform (only then is application development possible).

The porting of SQLite is possible and easy due to its good characteristics, framework, and architecture. And as SQLite is developed in the C language, it is highly portable. For this, we perform cross-compiling of the source code on various platforms. For porting SQLite onto the ARM-Linux platform, corresponding ARM-Linux tools are needed, and these tools are installed in **usr / local / arm-lnx / bin**/path. The bottom OS is the ARM-Linux, and porting is done using 3 tools: arm-lnx-GCC, arm-lnx-ar, and arm-lnxranlib. The specific course on porting is [12][13]:

(1) Use the command "echo PATH" to see if *arm-lnx*-GCC has been included in the PATH.

(2) Download the source code package from http://www.sqlite.org/, then decompress the package. After the decompression process, the source code and a few attached files are available in the un-zipped SQLite directory.

(3) A few necessary docs are to be edited/modified to ensure that SQLite runs on the ARM-Linux platform. Rename *Makefile. Linux*-GCC to *Makefile*, and then edit the files as below { **MODIFIED TO** is denoted as **[::-::]** }

*TOP = .. / sqlite***[::-::]***read: TOP =.;*
TCC = gcc- O6 **[::-::]** *read:TCC = armlnx-gcc-O6 ;*
AR = ar cr **[::-::]** *read: AR = arm-lnx-ar cr;*
RANLIB = ranlib **[::-::]** *read: RANLIB = arm- lnx -ranlib ;*
MKSHLIB = gcc-shared **[::-::]** *read:MKSHLIB= arm- lnx -gcc-shared.*

Delete the tclsqlite.o output file in the program file, as the TCL language binding with SQLite is not supported on the ARM platform.

(4) Save all the modified files, and later run the commands ***make install*** to generate *SQLite.h, libsqlite.* Header files.

(5) Later, on the ARM-Linux platform, run the SQLite; for this, copy it to the ARM board, and test a program, and if the result is correct, it implies that SQLite has been ported successfully onto the ARM

platform. After this, application developments can be carried on this platform using SQLite.

V. CONCLUSION

With the rising demand for intelligent appliances, the need for an embedded database has become the focal point of study. SQLite is a C language library and is open-sourced and contains a small core; DB is a simple file; the DB files are easy to move and cross-platform and are very suitable for building an EDS. Due to its unique advantages, SQLite has become the mainstream database in embedded systems.

In this paper, we have made a comprehensive analysis of the basic features, framework, architecture, and key technologies of EDS, and discussed the key interface functions of SQLite, and then the porting of SQLite to the ARM-Linux platform was detailed; later discussed on the application of SQLite in-home gateway using the API provided by SQLite. With enormous advantages, SQLite will have widespread use in fields like remote control, intelligent appliances, home medical equipment, etc.

REFERENCES

[1] Chunyue Bi et al., "Research and Application of SQLite Embedded Database Technology," WSEAS TRANSACTIONS on COMPUTERS, Issue 1, Volume 8, January 2009, pp. 83-91.

[2] B. Schneier, A. Shostack. Breaking up is hard to do: Modeling Security Threats for Smart Cards. USENIX Symposium on Smart Cards, 1999.

[3] R. Munz: Usage Scenarios of DBMS, Keynote, 25th International Conference on Very Large Data Bases, Edinburgh, UK, 1999, http://www.dcs.napier.ac.uk/~vldb99/Industrial SpeakerSlides/SAPVLDB.pdf

[4] Jiang MF et al., "Discovering Structure from Document Databases," Lecture Notes in Computer Science, vol 1574. Springer

[5] Rick F, et al., Introduction to SQL: Mastering the Relational Database Language, Addison Wesley, 2006.

[6] Hector.Garcia-Molina, et al., Database System Implementation, Prentice-Hall, 2001.

[7] Mike Owens and Grant Allen, "The Definitive Guide to SQLite," Apress 2nd edition, 2006.

[8] SQLite homepage [EB/OL], http://www.sqlite.org.

[9] Oracle Berkeley DB SQL API vs. SQLite API- Integration, Benefits and Differences, Oracle White paper, Nov 2016, pp. 1-11.

[10] Mike Owens, Embedding SQL Database with SQLite, Linux Journal, June 2003.

[11] P. Bohannon, D. Lieuwen, R. Rastogi, A. Silberschatz, S. Seshadri, S. Sudarshan, "The Architecture of the Dalí Main-Memory Storage Manager," Multimedia Tools and Applications, vol. 4, no. 2, pp. 115-151, 1997.

[12] Michael A.Olson, Selecting and implementing an EDS, IEEE Computer, 2000,33(7). 27-34.

[13] Ling-yun, Li-Jun, The Development and Application of Script Program Based on SQLite, China Academic Journal Electronic Publishing House, 1994-2008.

[14] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012-2017, Cisco, February 2013.

[15] Chaudhuri, S., Narasayya, V., "AutoAdmin 'WhatIf' Index Analysis Utility," Proceedings of the ACM SIGMOD Conference, Seattle, 1998.

[16] Rick F, van der Lans. Introduction to SQL: Mastering the Relational Database Language, Addison Wesley, 2006.