Litebase: A Database System for Mobile Devices

Juliana C. Imperial[1], Sérgio Lifschitz[2], Guilherme C. Hazan[1], Bruno Muniz[1]

[1]TotalCross Global Mobile Platform – Fortaleza – CE – Brazil

[2]Departamento de Informática – PUC-Rio, Rio de Janeiro, RJ – Brazil

{juliana.imperial,guich,bruno}@totalcross.com, Sergio@inf.puc-rio.br

Abstract. *Litebase is a database system developed to run on TotalCross, a mobile development platform for smartphones. This paper describes the database system features, currently work, and future enhancements so that it is reliable, fast in all operations, has an efficient memory management, and an efficient storage space usage. The first requirement is challenging because applications might be killed by the device system anytime, and the last three requirements are necessary since even the modern and powerful nowadays devices are not as powerful as computers.*

1. Introduction

Nowadays, the number of mobile devices, such as smartphones, is very huge and may overcome the number of computers (desktops, notebooks, and netbooks) in this year. Therefore, it is interesting to invest in program development for these devices, such as Androids, iPhones, iPads, and Windows Phones [Morgan Stanley 2010].

In order to facilitate the programming for mobile devices with different hardware and operational systems, TotalCross (totally-cross platform) [TotalCross 2014] was created. It is a program development platform for mobile devices which focus on portability, so that the same program can be executed in different platforms without changing its code, and without needing third-party software or hardware to create an executable to the target platform.

TotalCross is currently used on thirty four thousand devices and is present in 17 countries around the world, being used by huge enterprises. Its current version runs on Java, Windows XP/Vista/7/8, Androids, iOSs, Linux, and Windows Phone 8. Previous TotalCross versions used to run on Palm, BlackBerry, Windows CE/Mobile, and previous desktop Windows versions, which are not supported anymore because these platforms were discontinued.

A TotalCross programmer only needs to know to program in Java, since TotalCross uses one subset of this widely used language and some additional libraries. When running a TotalCross program on Java, it uses the Java virtual machine (VM), whereas on the other devices, the TotalCross VM is used.

Since its first version, TotalCross comes with a portable relational database system called Litebase [Litebase 2014][1].

[1] Link for a demonstration video: http://www.julianaimperial.com.br/demo.rar.

Litebase implements an almost standard SQL subset. It was written in such a way that a database created on one device can be used on another one without any conversion. Although it is embedded in TotalCross, it is also possible to use it in applications without user interface, but the use of TotalCross is necessary. However, on Java, a programmer can use it without using TotalCross, using it as a server.

Litebase was designed to be fast and to have an efficient memory management and storage space usage. In the past these requirements were even more necessary as the old devices where it used to run, such as Palms and BlackBerries, had little RAM memory, small storage space, and very slow processors. Nowadays, these requirements are still necessary as the modern devices are not as fast as computers.

In spite of being used by many customers, Litebase still lacks much functionality that widely used database systems implement, such as transactions (which is being implemented) and concurrency. All of them are in the roadmap to be implemented in the future to improve the system.

The rest of this paper is organized as follow: section two describes Litebase features, section three describes Litebase limits and limitations, section four shows some implementation details, section five compares Litebase with SQLite, a widely used database system on mobile devices, and the last section concludes this work and shows its development roadmap.

2. Litebase Features

Litebase has the following features:

- SQL used keywords are treated as reserved words.
- Can issue error messages in English or Portuguese, and it is easy to add another language to it. However, internal error messages from the underlining operational system are issued in the language used by the device.
- Supports case-insensitive strings using the keyword `nocase`.
- Strings can be stored in unicode (default) or ascii to save disk space.
- Supports cryptography, shuffling table files data so that they become unreadable.
- Supports `like`, some SQL data and aggregate functions, `order` and `group by`.
- Supports simple and composed primary keys and indices. There are no foreign keys.
- Supports null and default values.
- Supports inner joins, there are no outer joins yet.
- Supports adding columns, and column and table rename using `alter table`.
- Supports multiple connections and multiple threads (with some limitations because the system doesn't support concurrency yet) with the same or different databases.
- Has error and operation logging.
- Has a specific configuration for huge-data loading in a table in order to speed up the total time taken for all operations in the database. When transaction implementation

is finished, this will not be necessary anymore because the flushes are also only done when the memory cache is full.

- Eases the synchronization task from the device to the server, a very important feature to mobile applications. Therefore, this is one of the most important Litebase features. The database system provides a way to mark a row as synchronized, new, updated, or deleted, without additional columns in a table. It also provides an easy way to search this information and send it to another database system on a server.
- The deletion of records and index keys are always logical. When the programmer issues a purge operation in a Litebase table, all the wasted space (due to deletions and updates), are recovered, and the indices, rebuilt.
- Each table has at least two files: one for row (fixed-length records) data, and another for the strings and blobs stored in the table. Moreover, each table index, including primary keys, is stored in a different file.
- The string types in Litebase are `char` and `varchar` and they are treated in the same way by the database system.
- Index files have a cache of 512 bytes (maximum node size), whereas the table files have a cache of at least 2048 bytes. This is greater if the system must read more than this size at once, for example, when a blob, string, or table record to be read is greater than that. Those values were found empirically.
- When the index is created or re-created, all internal nodes are full so that the queries will be faster. This is done by sorting the keys before inserting them.

3. Litebase Limits and Limitations

Litebase has the following limits and limitations:

- The first data of a row is always an internal 4-bytes field called rowid, which can only be used for searches, but can never be updated. rowid has a unique value for each line and is never reused, even if a row is deleted. The first two more significant bits of the four bytes store information for synchronization, indicating if a row is new, updated, deleted (a removed row is never marked as synchronized), or synchronized. The other bits represent a natural number with 30 bits, which can have more than 500 million different values (beginning with 1). This value is huge even for a database in modern mobile devices and does not to be increased soon.
- Each table can have at most 255 columns including rowid, a limit never reached by well-designed databases.
- The maximum number of parameters in a prepared statement is 254.
- A where clause can use at most 32 indices.
- The maximum number of simple indices is given by the number of table columns, while the maximum number of composed indices is also 32.
- The maximum number of columns used in a where or having clause is also 254, where this number takes into consideration repeated values. That is, if a where clause uses only the column `A` repeated 255, an exception will be raised. This

limitation would not be important if Litebase supported the IN operator. Without it, the programmer must use an OR with all the possible values of the column.

- Since Litebase transactions are still being implemented, during normal usage, all table modifications are flushed immediately, since the database system does not have transactions yet. This guarantees that no data will be lost. However, when the programmer uses the mode to load huge amounts of data in a table, the flush is only done when necessary, when the file caches are full. Therefore, if a TotalCross program using Litebase in this way is stopped suddenly (due to a crash or the system closes it), to be used again the table must pass through a recovery process, where a corrupted row is marked as deleted and has its rowid set to 0, and all indices are rebuilt.
- The currently maximum blob size is 10 Mb. Even this small size might be too huge and cause out of memory problems. In the future it is intended to increase this size and enable to use streams to store and query blobs, so that it is not necessary to have all its content in memory. Streams will also be used to improve some internal operations in order to improve memory management.
- The maximum size of the file that stores strings and blobs is 2 Gb, because the value stored as the string or blob position in the record has four bytes (`int`). To increase the file size, this value must be changed to be stored in 8 bytes (`long`).
- There can only be 65535 nodes in a Litebase index. If the index is created on a `short` (two bytes) column, it will be able to hold about two million keys if they are added after the index creation, and about four million if the index is created or recreated. This seems to be a very large number for a mobile device. However, if the key is of a larger type, or the index is composed, the number of keys it can hold will be much smaller and this might turn a specific index usage impossible if the table is big enough. This limit will be increased soon.
- There are no triggers, views, sub-queries, and constrains.

4. Implementation Details

Litebase has two different codes which do the same work: one implemented in Java to run on the Java VM and another one which has only a very small part in Java and all the rest in C to run on the TotalCross VM on the supported devices. The parts in Java are only to load the Litebase native library and to deal with some synchronization problems when trying to load it. This way, almost all Litebase execution is native, without using the TotalCross VM, improving its performance.

Although Litebase uses native code, it was written in order to make it as platform independent as possible, using TotalCross internal native functions to deal with platform dependent details, such as file treatment. Therefore, it is very easy to port Litebase to new mobile platforms that support code written in C.

The only parts of Litebase code that are platform dependent are the ones that deal with paths (on Windows Phone 8, for instance, path strings use unicode characters while the others use ascii), and when it has to deal with a POSIX limitation concerning the maximum number of files that can be opened at the same time.

The problem with the approach of having many files in a database is that iOSs, Androids, and Linux, have a maximum number of files that can be opened at the same time by all currently running programs. On iOSs, this number is very small, about 256, whereas on Androids, it's about 1024. To solve this problem, Litebase will only open 128, 512, and 1024 files on iOSs, Androids, and Linux, respectively, so that Litebase won't exhaust the system resources. If the programmer opens too many tables so that it's necessary to use too many files at the same time, Litebase will close the last used files when it needs to open another one and re-opens it again if needed without any extra effort from the programmer.

5. Comparison with SQLite

SQLite [SQLite] is a widely used database system and adopted as a standard on Androids and iOSs. The current TotalCross version also has an embedded SQLite so that the programmers have another option to store data. SQLite has many sponsors, has many more features and supports much more of the SQL standard than Litebase. However, Litebase has some advantages:

- Litebase can issue error messages in English or Portuguese and it is easy to add another language to it whereas SQLite only supports English.
- Litebase cryptography is free although it is not strong.
- Using Litebase it is possible to log all used SQL commands and some other operations such as purge and use the log files to reproduce a specific Litebase usage in order to find bugs in the database system or in the TotalCross program.
- Strings can be stored in ascii to save the storage space.
- Litebase is strong-typed, which is less error-prone than the weak-typed SQLite, where any type of data can be inserted in any another type of column (for instance, inserting a blob in an integer field).
- In SQLite the synchronization task must be programmed using an extra field, for instance, whereas Litebase provides it. Moreover, deletions will have to be delayed because the deleted data space might be reused at any time.
- SQLite does not provide a way to drop all indices in a single command, whereas Litebase supports `drop index * on table_name`.
- There is no way to add or drop a primary key on an already created table in SQLite, whereas Litebase supports these commands via `alter table`.
- Litebase result sets hold the number of rows that result from a query. Moreover, it is possible to navigate to any row of the result set. This is useful when loading data to a grid from a result set dynamically so that only the results shown on screen are loaded. When going backward on the grid, it is only necessary to navigate the result set backwards. It is also possible to jump to absolute and relative positions in it. SQLite only allows going forward. To go backward, it must be reset.
- Litebase does not support concurrency yet, but since all table are stored in different files, when writing data it will not be necessary to issue an exclusive lock in the entire database such as in SQLite. Litebase locks will be in table level.

- Litebase is faster than SQLite for huge-data load. A simple sample inserting 50.000 rows of a simple table with only one string field (using the configuration for huge-data load on Litebase and a transaction on SQLite), where half of the insertions use a prepared statement, takes 7 % less time on Android devices. For huge-data load, the difference of time taken with each database system can be significant.

Nevertheless, SQLite has the following advantages:

- There is no wasted time to check types when inserting data.
- The space used to store data is smaller because each column data only needs the number of bytes used to represent the current data. For instance, a column declared as `int` will always need four bytes in Litebase even if it is one or null whereas in SQLite one will only need one byte and null will not need any byte at all.
- There are more people working on it, and using and sponsoring it. So, it is more reliable and new features are implemented much faster.

Finally, the TotalCross software LaudoMovel [LaudoMovel] has versions using SQLite and Litebase their performance seemed to be the same.

6. Conclusion, Currently, and Future Work

The number of devices using TotalCross is increasing and will increase even more as it is ported to new mobile platforms such as FirefoxOS and MacOS. Therefore, the number of Litebase users will also increase, turning necessary the implementation of new features in Litebase to satisfy the programmers' needs.

A feature that is being currently implemented is transactions, so that the recovery process will not be needed anymore. The transactions will deal with that. Some future enhancements include concurrency implementation, stronger cryptography, table data compression, outer joins, subqueries, more column types such as `byte` and `decimal`, providing a JDBC API, etc.

References

LaudoMovel Site (2014), https://caixa.laudomovel.com.br/.

Litebase Companion v 2.8 (2014), http://www.totalcross.com.br.

Morgan Stanley Internet Trends Report (2010), http://pt.slideshare.net/alfonsogu/crecimiento-del-uso-de-internet-en-el-mvil.

SQLite Online Documentation (2014), http://www.sqlite.org/docs.html.

TotalCross Companion v 3.0 (2014), http://www.totalcross.com.br.

Ramakrishnan, R. and Gehrke, J. (2003), Database Management Systems, MacGraw-Hill, 3rd edition.