SQLite RDBMS Extension for Data Indexing Using B-tree Modifications

Anton Rigin

Faculty of Computer Science
National Research University – Higher School of Economics
Moscow, Russia
amrigin@edu.hse.ru

The B-tree structure has several modifications which are, by default, not supported in the popular open-source RDBMS SQLite. In the scope of this work an extension for SQLite is developed, based on some modifications of the B-tree structure. The modifications of the base structure were developed as a C++ library. The library is connected to the SQLite using the C-C++ cross-language API. The SQLite extension also implements the novel algorithm for selecting the best possible index structure (B-tree or one of its modifications) for some table of a database. The algorithm uses the multiclass linear classification. In addition, the plugin for a SQLite GUI manager is developed.

Keywords—B-tree; data indexing; SQLite; DBMS; RDBMS; multiway tree

I. INTRODUCTION

Last decades, the amount of data volume is growing substantially which exposes the well-known problem of big data [1].

Many companies and laboratories need to collect, store and process big data. There exist many algorithmic and software solutions to cope with these problems. One of these solutions is using indices which are usually represented by data structures such as hash tables and trees.

Using indices creates a new problem – when data are stored on slow carriers, it is more efficient to load data batches from a storage instead of splitting to individual elements. Multiway trees solve this problem. One type of them is a B-tree which was initially described by Bayer and McCreight in 1972 [2]. The B-tree also has several modifications. In this paper, the following B-tree modifications are considered: B^+-tree [3], B^*-tree [4] and B^{*+}-tree (the latter is developed by the author of this paper data structure, which combines the main B^+-tree and B^*-tree features) [5].

One of the popular open-source relational database management systems (RDBMS) is SQLite [6]. It is used in mobile phones, computers and many other devices. However, this RDBMS does not support using B^+-tree or B^*-tree as data index structures by default.

The main goals of the work are the following:

- to add B-tree modifications such as B^+-tree, B^*-tree and B^{*+}-tree to SQLite;

- to develop an algorithm that would allow selecting the most appropriate indexing data structure (B-tree, B^+-tree, B^*-tree or B^{*+}-tree) when a user creates a table;

- to create a plugin for one of the SQLite GUI managers which will allow users to visualize index structure and to output main index metrics.

The work includes linking of B-tree modifications from a C++ library (developed by the author of this work previously) to SQLite using a C-C++ cross-language API, developing an algorithm for selecting an indexing data structure and creating a plugin for one of the SQLite GUI managers.

The rest of the paper is organized as follows. Firstly, B-tree, B^+-tree, B^*-tree and B^{*+}-tree are shortly described. After this, the SQLite, its indexing algorithms and extensions are presented. Then, the B-tree modifications C++ library, connecting it to the SQLite RDBMS and the development of algorithm of selecting the best index structure for table are discussed, after which the SQLite GUI managers are described. After this, the main points of the paper are summarized in conclusion and used references are presented.

II. B-TREE AND ITS MODIFICATIONS

A. B-tree

The B-tree is a multiway tree. It means that each node may contain more than one data key. Furthermore, each node except of the leaf nodes contains more than one pointer to the children nodes. If some node contains k keys than it contains exactly $k + 1$ pointers to the children nodes [2].

The B-tree depends on its important parameter which is called B-tree order. The B-tree order is such a number t that:

- for each non-root node, the following is true: $t - 1 \leq k \leq 2t - 1$, where k is the number of keys in the node [2];

- for root node in the non-empty tree the following is true: $1 \leq k \leq 2t − 1$, where k is the number of keys in the node [2];
- for root node in the empty tree the following is true: $k = 0$, where k is the number of keys in the node [2].

B-tree operations complexities are the following (t is the tree order, n is the tree total keys count):

- for the searching operation: time complexity is $O(tlog_t n)$, memory usage is $O(t)$ and disk operations count is $O(log_t n)$ [2];
- for the nodes split operation (the part of the insertion operation): time complexity is $O(t)$, memory usage is $O(t)$ and disk operations count is $O(1)$ [2];
- for the insertion operation (includes the nodes split operation): time complexity is $O(tlog_t n)$, memory usage is $O(tlog_t n)$ for simple recursion and $O(t)$ for tail recursion and disk operations count is $O(log_t n)$ [2];
- for the deletion operation: time complexity is $O(tlog_t n)$, memory usage is $O(tlog_t n)$ for simple recursion and $O(t)$ for tail recursion and disk operations count is $O(log_t n)$ [2].

B-tree is usually used as the data index [2].

The example of B-tree is shown on the Fig. 1.

Fig. 1. The B-tree example, tree order $t = 6$.
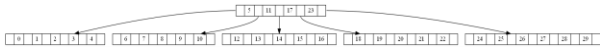
B. B-tree Modifications

B+-tree is the B-tree modification in which only leaf nodes contain real keys (real data), other nodes contain router keys for searching real keys. Leaf nodes in B+-tree contain $t \leq k \leq 2t$ keys, where t is the tree order, the rules for other nodes are the same as in B-tree [3]. Keys deletion in B+-tree is expected to be faster than in B-tree since it is always performed on the leaf nodes.

B*-tree is the B-tree modification in which each node (except of the root node) is filled at least by 2/3 not 1/2 [4]. Keys insertion in B*-tree is expected to be faster than in B-tree.

B*+-tree is the B-tree modification developed by the author of this paper which combines the main B+-tree and B*-tree features together. In this data structure only leaf nodes contain real keys (real data) as in B+-tree and each node (except of the root node) is filled at least by 2/3 as in B*-tree.

III. SQLITE AND ITS EXTENSIONS

The SQLite is the popular open-source C-language library which implements the SQLite relational database management system (RDBMS) [6]. The SQLite default index algorithms are hash-table and B-tree [7]. The SQLite does not implement B+-tree and B*-tree based indices.

Nevertheless, SQLite supports loading its extensions at run-time, which can add new functionality to the SQLite [8]. For example, it can be a new index structure implementation. One of such extensions is the R-tree. The R-tree is a B-tree modification which allows to index geodata. It is loaded by the SQLite as the extension and delivered together with the SQLite RDBMS default build [9].

IV. B-TREES MODIFICATIONS C++ LIBRARY

The B-tree modifications C++ library was developed by the author of this paper previously. It contains B-tree, B+-tree, B*-tree and B*+-tree implementations written in C++ [5].

In the current work this library is connected to the SQLite as the run-time loadable extension. For this goal the C-C++ cross-language API is implemented. It is possible to do using the *extern "C" { … }* C++ statement. The other tasks are to implement base SQLite extension's methods and to use Makefiles to make this extension run-time loadable correctly. The extension provides module [10] for creating virtual tables (tables which encapsulate callbacks instead of simple reading from database and writing to database) based on this module [11].

V. ALGORITHM OF SELECTING THE BEST INDEX STRUCTURE

In this work an algorithm for selecting the best index structure for a table is developed in the following way.

The algorithm considers B-tree and its modifications (B+-tree, B*-tree and B*+-tree) for using as an index and is based on the mathematical statistics and machine learning methods. It uses the multiclass linear classification (1). The table rows count (*rows_count*), table columns count (*columns_count*) and index column data type size (*index_size*) serve as appropriate features for this model. Also, other features can be used.

$$y_{i} = w_i^1 rows\_count + w_i^2 columns\_count + w_i^3 index\_size + \beta_i, \quad i = 1, 2, 3, 4 \quad (1)$$

i from (1) matches the B-tree ($i = 1$), B+-tree ($i = 2$), B*-tree ($i = 3$) or B*+-tree ($i = 4$) classifier. Each classifier checks the probability that this classifier's tree should be chosen for the given table against other classifiers' trees. The i-th classifier's tree is chosen as index structure for the table if and only if: $y_i = max\{y_1, y_2, y_3, y_4\}$.

The classifiers are trained using the Python 2 and the Sci-Kit Learn library. For each example from training dataset the class of the index with the smallest performance parameters (time complexity and memory usage) weight sum is chosen. After training the classifiers' coefficients are copied to the C-language SQLite extension code.

VI. SQLITE GUI MANAGERS

There exist several SQLite managers with graphical user interface (GUI). Examples are SQLiteManager [12], SQLite Expert [13] and SQLiteStudio [14]. The SQLiteManager [12] and SQLite Expert Professional [15] support developing

plugins on the Lua language. Also, the SQLiteStudio supports writing plugins on the Qt C++ platform.

The plugin for one of these or other SQLite managers will be developed. The plugin will support visualization of the B-tree modifications based index for the given table which uses this index and outputting the main metrics.

VII. CONCLUSION

The big data problem currently affects the world. There are many mathematical and software solutions for collecting, storing and processing big data including the data indexing. Many of the index data structures are tree-based ones such as B-tree and its modifications. B-tree is used as an index structure in many DBMSs including the popular open-source RDBMS SQLite. However, the SQLite does not support its modifications which may be more appropriate for some tasks than the original B-tree. In the current work this problem is elaborated.

Firstly, the B-trees modifications C++ library is connected to the SQLite as extension using C-C++ cross-language API. After this, the algorithm of the best index selecting is developed using the linear classification. In the completion, the plugin for one of the SQLite managers will be developed which will support visualization of the B-tree modifications based index and outputting the important metrics' values.

This work tests new data indexing approaches using the SQLite as an example. The results of the work can be used by researchers and professors in this field and their students. The SQLite B-tree modifications extension can be used by all the developers who use this DBMS.

REFERENCES

[1] J. Manyika *et al.*, "Big data: The next frontier for innovation, competition, and productivity," McKinsey Global Institute, May 2011. Accessed: Jan. 20, 2019. [Online]. Available: https://www.mckinsey.com/~/media/McKinsey/Business%20Functions/ McKinsey%20Digital/Our%20Insights/Big%20data%20The%20next%2 0frontier%20for%20innovation/MGI_big_data_exec_summary.ashx

[2] R. Bayer and E. McCreight, "Organization and maintenance of large ordered indexes," *Acta Informatica*, vol. 1, no. 3, pp. 173 – 189, 1972.

[3] K. Pollari-Malmi. (2010). B$^+$-trees [PDF paper]. Available: https://www.cs.helsinki.fi/u/mluukkai/tirak2010/B-tree.pdf

[4] "B*-tree." NIST Dictionary of Algorithms and Data Structures. Available: https://xlinux.nist.gov/dads/HTML/bstartree.html (accessed Dec. 24, 2018).

[5] A. Rigin, "On the Performance of Multiway Trees in the Problem of Structured Data Indexing," (in Russian), coursework, Dept. Soft. Eng., HSE, Moscow, Russia, 2018.

[6] "SQLite Home Page." SQLite.org. Available: https://www.sqlite.org/ (accessed Jan. 20, 2019).

[7] "Architecture of SQLite." SQLite.org. Available: https://www.sqlite.org/arch.html (accessed Jan. 20, 2019).

[8] "Run-Time Loadable Extensions." SQLite.org. Available: https://www.sqlite.org/loadext.html (accessed Jan. 20, 2019).

[9] "The SQLite R*Tree Module." SQLite.org. Available: https://www.sqlite.org/rtree.html (accessed Jan. 20, 2019).

[10] "Virtual Table Object." SQLite.org. Available: https://www.sqlite.org/vtab.html (accessed Jan. 21, 2019).

[11] "The Virtual Table Mechanism Of SQLite." SQLite.org. Available: https://www.sqlite.org/vtab.html (accessed Jan. 21, 2019).

[12] "SQLiteManager." SQLabs. Available: https://www.sqlabs.com/sqlitemanager.php (accessed Jan. 26, 2019).

[13] "SQLite Expert." SQLite Expert. Available: http://www.sqliteexpert.com/ (accessed Jan. 26, 2019).

[14] "SQLiteStudio". SQLiteStudio. Available: https://sqlitestudio.pl/ (accessed: Jan. 26, 2019).

[15] "SQLite Expert Features comparison". SQLite Expert. Available: http://www.sqliteexpert.com/features.html (accessed Jan. 26, 2019).

[16] "Writing new SQLiteStudio plugins". Github.com. Available: https://github.com/pawelsalawa/sqlitestudio/wiki/Writting_plugins (accessed Jan. 26, 2019).

Word Count: 1605