Adaptive Indexing: Fundamental and Hybrid Approaches

Pavlo Shevchenko
Otto-von-Guericke-University, Magdeburg
pavlo.shevchenko@st.ovgu.de

Therefore, even as an index to a book
So to his mind was young Leander's look

Christopher Marlowe
Hero and Leander, 1593

***Abstract*—What did I do in a nutshell?**

***Keywords*—Database indexing, database cracking, adaptive merging, hybrid approaches**

I. Introduction

The concept of indexes was widely used in bibliothecography throughout the history of mankind. Indexes designed to help the reader find information quickly and easier found their way into the databases. Introduction of such intuitive concept has helped to improve the speed of data retrieval. In the early stages of database technology data and indices were stored on disk in a form of B-Trees.

However, over the years, as in-memory databases (IMDB) found their way to the market, need for more lightweighted index structures that could utilize the fact that data is always in main memory has become evident. At first, precalculated index in a form of a sorted column with pointers to the original records in the table that was stored in main memory has become a concept that helped to decrease the response time for answering a query. However, continuously growing amount of data, more sophisticated queries and unpredictable behavior of a user have turned advantages of indexes into disadvantages. Storing large columns in a main memory has become a problem, need to store indexes to numerous columns has even increased that problem. As it was hard to predict which column is specificly interesting for the user, new indexes have to be calculated online as queries referring to new columns arrive.

All these issues motivated developing of advanced indexing techniques as partial and soft indexes. Both these concepts will be presented in details in Background section. Creators of soft indexes [2], Lühring, Sattler, Schmidt and Schallehn, have predicted the future of database indexes. They realised that index cannot remain static after being created, it requires to be continuously tuned as some parameters of database system are changing. Such parameters include workload, size and distribution of data, schema and infrastructure changes. This leads to a conclusion that the best index is a result of query processing with all these parameters kept in mind.

This concept known as adaptive indexing will be presented in this short paper. Some fundamental approaches (e.g. database cracking [3] and adaptive merging [5]) will be discussed in Sections 2 and 3 respectively. Further approaches that try to combine the advantages of both methods will be introduced in Section 5. Presented approaches will be evaluated in Section 7. The work will be concluded by presenting related work in Section 8 and proposing directions for further reasearch in Section 9.

II. Background

As mentioned before, two indexing methods: *partial indexes* and *soft indexes* were developed to tackle some problems caused by traditional indexes. In this section, both of these approaches and their influence on adaptive indexing will be presented.

A. Partial Indexes

As its name suggests, partial index, also known as filtered index, is an index which contains only part of the indexed column that satisfies some condition. Such condition is usually expressed in a form of an interval known as *inclusion interval*. Depending on selectivity of a condition, the amount of rows in the indexed column can be decreased. This allows to reduce the amount of memory needed to store the index. Creation of such index may be specified as follows:

$$CREATE\ INDEX\ ON\ relation(column)name$$
$$WHERE\ condition$$

Clearly, selecting the right condition is a key element for creation of partial index. Condition can be found in several ways:

- user input:

  Obviously, user himself can specify the insertion interval as he/she may possess some additional knowledge about the data. However, one of the major principles of modern DBMS is removing any low-level data management aspects from the user of the system. As a result, some database tuning is needed.

- index creation as side-effect of query processing [9]:

  Even though user has been removed from tuning process, he/she can still indirectly participate in finding a suitable condition. The fact that user queries some relation (column) while applying some selection condition, leads to an assumption that data satisfying query condition is of some interest for the user and may be queried again in the future. So, taking query condition from the user as a qualification for partial index is not a bad idea after all. Continuous refinement of a condition and index ad hoc may provide an optimal partial index for given relation. This concept was later used in database cracking.

- statistical approach [10]:

  Since index creation in a way described above may increase query processing time, statistical information, like the percentage of queries that access each column or distribution of values iin each column, may be used to perform index creation after data loading and before queries are processed.

B. Soft Indexes

Soft indexes proposed in [2] is one of the first attempts to pass responsibility for index creation from DBA to DBMS. Index management is carried out in three major steps:

- Observation
- Prediction
- Reaction

At first, statistical information about current state of the system (workload, data, schema, infrastructure) is collected (Observation). Afterwards, in order to derive an optimal index configuration, a list of index candidates sorted by some criteria is analyzed and top *k* candidates are chosen for materilization in the Reaction phase. During the latter depending on results of Prediction phase, some indexes will be created or deleted.

Meaning. Combination of advantages of soft and partial indexes resulted in further approaches known as adaptive indexing. On-the-fly index creation based on query workload and storing only subset of rows in an index motivated the first method for adaptive indexing, called *database cracking*, which will be presented in the next section.

III. Database Cracking

After presenting the necessary background about partial and soft indexes, overview of one of the main adaptive indexing approaches, database cracking, will be provided. In the first section (3.A) motivation and main idea of database cracking will be presented, followed by an example. In conclusion, the approach will be analyzed regarding its performance.

A. Motivation and Basic Idea

Motivation.Database cracking is pursuing the goal to provide fast access to the data and the self-organized behavior of database system. It is based on idea that index maintenance is a byproduct of query processing, not updates. In contrast to soft indexes, discussed earlier, database cracking achieves its goal not by tuning system's configuration and choosing the best fitting indexes based on numerous parameters, but by using continuous physical reorganization (cracking the database into manageable pieces).

Main Idea. Here the steps of the method will be presented, followed by explanations:
Assumptions: given a column oriented database with column A; range query q in a form of either $c \leq R.A$ or $c_1 \leq R.A \leq c_2$

1) The first time query on attribute A arrives, copy of the column is created and cracked (notation: A_{crk}).
   This allows to leave original column intact and fast reconstruction of records. Cracking is performed by partitioning values in A_{crk} as followed:
   - q = $c \leq R.A$
     A_{crk} consists of two partitions, where all elements before some position p are smaller than c and all elements after position p are larger.
   - q = $c_1 \leq R.A \leq c_2$
     A_{crk} consists of three partitions, where all elements before some position p_1 are smaller than c_1, all elements after some position p_2 are larger than c_2 and all elements between p_1 and p_2 are bigger than c_1 and smaller than c_2.
2) Refine Further queries on attribute A arrive
   Partition A_{crk} with regard of earlier cracking. E.g. $q_1 = 5 \leq R.A \leq 10$. After cracking, A_{crk} will contain three partitions as described earlier. When next query arrives $q_2 = 7 \leq R.A$, then cracking should be performed only on part of A_{crk} between positions p_1 and p_2, as it is known that all elements before position p_1 satisfy the query and all elements after p_2 don't.
3) Cracking Index
   In order to quickly find the partition to refine, each cracker column is supplied with *cracker index* which stores how values are currently distributed in the cracker column. Each node of a tree contains value v, its position p in a cracker column and whether v is left or right inclusive regarding p.
4) Query for new column B arrives
   Perform steps 1 through 3 for new column B and store previous results.

B. Cracking Algorithms

C. Analysis

IV. Adaptive Merging

V. Hybrid Approaches

VI. Evaluation

Point out complementary nature of cracking and merging. Compare to other hybrid approaches. Speculate on future and usage of the methods

VII. Related Work

Some research on related work has to be done.

VIII. Conclusions

What did I find out?

Acknowledgement

I thank M.Sc. Gabriel Campero Durand of Otto-von-Guericke-University, Magdeburg for providing insight and expertise to start this research and for his enormous support throughout the whole process of research, writing and evaluation of this scientific work. I would also like to show my gratitude to the DBSE Research Group of Otto-von-Guericke-University, Magdeburg for making this work possible and organising "Seminar on Modern Software Engineering and Database Concepts", during which this research took place.

References

[1] Knight, G. Norman. Indexing, the art of a guide to the indexing of books and periodicals. No. 029.5 K55. 1979.

[2] Lühring, Martin, et al. "Autonomous management of soft indexes." Data Engineering Workshop, 2007 IEEE 23rd International Conference on. IEEE, 2007.

[3] Idreos, Stratos, Martin L. Kersten, and Stefan Manegold. "Database Cracking." CIDR. Vol. 7. 2007.

[4] Schuhknecht, Felix Martin. "Closing the circle of algorithmic and system-centric database optimization: a comprehensive survey on adaptive indexing, data partitioning, and the rewiring of virtual memory." (2016).

[5] Graefe, Goetz, and Harumi Kuno. "Self-selecting, self-tuning, incrementally optimized indexes." Proceedings of the 13th International Conference on Extending Database Technology. ACM, 2010.

[6] Idreos, Stratos, et al. "Merging what's cracked, cracking what's merged: adaptive indexing in main-memory column-stores." Proceedings of the VLDB Endowment 4.9 (2011): 586-597.

[7] Pirk, Holger, et al. "Database cracking: fancy scan, not poor man's sort!." Proceedings of the Tenth International Workshop on Data Management on New Hardware. ACM, 2014.

[8] Schuhknecht, Felix Martin, Alekh Jindal, and Jens Dittrich. "The uncracked pieces in database cracking." Proceedings of the VLDB Endowment 7.2 (2013): 97-108.

[9] Stonebraker, Michael. "The case for partial indexes." Sigmod Record 18.4 (1989): 4-11.

[10] Seshadri, Praveen, and Arun Swami. "Generalized partial indexes." Data Engineering, 1995. Proceedings of the Eleventh International Conference on. IEEE, 1995.