Holistic Indexing: Offline, Online and Adaptive Indexing in the Same Kernel

Eleni Petraki
CWI Amsterdam
petraki@cwi.nl
Expected Graduation Date: 2016
Supervised by Stratos Idreos

ABSTRACT

Proper physical design is a momentous issue for the performance of modern database systems and applications. Nowadays, a growing amount of applications require the execution of dynamic and exploratory workloads with unpredictable characteristics that change over time, e.g., social networks, scientific databases and multimedia databases. In addition, as most modern applications move to the big data era, investing time and resources in building the wrong set of indexes over large collections of data can severely affect performance.

Offline, online and adaptive indexing are three distinct approaches to the problem of automating the physical design choices. Offline indexing is best in static environments with stable workloads. Online indexing is best in relatively dynamic environments where the query workload can be monitored. Adaptive indexing is best in fully dynamic environments where no idle time or workload knowledge may be assumed. We observe that these three approaches are complementary, while none of them can satisfy the needs of modern applications in isolation.

We envision a new index selection approach, *holistic indexing* that excels its predecessors by combining the best features of offline, online and adaptive indexing while overcoming their weaknesses. The main goal is the creation of a database kernel that can autonomously create partial indexes which are continuously refined during query processing as in adaptive indexing but at the same time the system continuously detects any opportunity to improve the physical design offline; whenever any idle time occurs it tries to exploit knowledge gathered during query processing to refine existing indexes further or create new ones. We sketch the research space and the new challenges such a direction brings.

Categories and Subject Descriptors: H.2 [DATABASE MANAGEMENT]: Physical Design - Access Methods
H.3 [INFORMATION STORAGE AND RETRIEVAL]: Content Analysis and Indexing - Indexing methods

General Terms: Algorithms, Performance, Design

Keywords: Holistic Indexing, Self-organization

SIGMOD/PODS'12 PhD Symposium, May 20, 2012, Scottsdale, AZ, USA.

1. INTRODUCTION

As we increase our ability to collect more and more data, the usage of data goes beyond the strict and static query processing paradigm of the past. Nowadays, modern applications such as scientific data management, social networks, web logs, etc. bring new challenges. In particular, the database size grows quickly to unprecedented amounts, while the query processing patterns follow an exploratory behavior. For example, an astronomer browses huge piles of scientific data in search for interesting patterns in the stars while a log analyst browses Terabytes of daily logs in search for interesting usage patterns.

The Problem. Physical design, i.e., proper index selection has always been one of the predominant performance factors for data management systems (DBMS). In the new era of dynamic and exploratory environments, physical design becomes especially hard given the instability of the workload and the continuous stream of big data; a physical design choice is not necessarily correct or useful for long stretches of time while at the same time workload knowledge is scarce given the exploratory user behavior.

State of the Art. In the past, the choice of the proper index collection was assigned to database administrators (DBAs). However, as applications became more and more complex index selection became too complex for human administration. Today, all major commercial DBMSs offer auto-tuning tools to accomplish automatic index selection [1, 6, 17]. There are three indexing approaches concerning when they perform the workload analysis and when they build the physical design. *Offline indexing* assumes enough workload knowledge and idle time to built the physical design before queries arrive to the system [1, 2, 3, 5, 6, 17]. *Online indexing* makes a step towards more dynamic environments by allowing for continuous monitoring and periodically evaluating the physical design [4, 15, 16]. *Adaptive indexing* is a fully dynamic approach as it assumes no workload knowledge and requires no idle time; indexes are built continuously, partially and incrementally as part of query processing [7, 8, 10, 11, 12, 13, 14] .

Observation. Although the above indexing methods share a common objective, they target different environments; they are complementary. Offline indexing fits best in static environments where the workload is known a priori while online indexing assumes that it can properly monitor and predict the workload. Adaptive indexing does not try to predict the workload; it only reacts on a single query at a time. Similarly, offline indexing assumes enough idle time to built the physical design a priori while online indexing interleaves index creation with queries. Adaptive indexing does not require or exploit any idle time.

Motivation. Modern applications with fully dynamic and exploratory workloads can benefit from all indexing approaches. None of them in isolation can provide a system that adapts instantly like

Indexing	Statistical analysis	Exploitation of a-priori idle time	Exploitation of idle time during workload execution	Incremental indexing	Workload
Offline	√	√	×	×	static
Online	√	×	√	×	dynamic
Adaptive	×	×	×	√	dynamic
Holistic	√	√	√	√	dynamic

Table 1: Features of offline, online, adaptive and holistic indexing.

adaptive indexing; and at the same time it can monitor the workload and observe long term patterns like online indexing; and at the same time it can exploit idle time and workload knowledge to built indexes that are anticipated to be useful like offline indexing.

For example, assume a scenario from the domain of scientific databases in astronomy. As new Terabyte of data arrive daily, there will be a standard set of queries which the scientists want to always run and a given minimum time which they are willing to wait for this tuning to take place. For those queries an offline indexing -like approach is useful. However, not all patterns are known a priori as the scientists will now start exploring the data. As queries arrive which are not covered by the existing indexes, the system starts building partial indexes and incrementally refining them as more queries arrive like adaptive indexing. At the same time it continuously monitors the query patterns for overall patterns like online indexing.

Vision. To address the needs of modern exploratory applications we introduce a novel indexing approach, *Holistic Indexing*. Holistic indexing targets exactly the motivation as described in the previous paragraph; it continuously monitors and analyzes the workload, instantly adjusting to changes in workload characteristics; it can exploit any idle time or any workload knowledge as it appears. Indexes are partial and incremental and are built and refined as part of query processing but also during idle time. There is no external tool or human administration; the continuous indexing properties are embedded in the database kernel and are part of the query processing and storage engine.

The new opportunities brought by holistic indexing are visualized in Table 1 and Figure 1. In offline indexing, statistical analysis of a given workload and index building take place before the workload execution. During workload processing there are idle time intervals that are not exploited. In online indexing, statistical analysis is continuous and triggers physical design reorganization. The physical design is reorganized on-the-fly or during idle time. Both offline and online indexing refer to full indexing, i.e., indexes on tables cover all the rows equally even if some rows are needed often and some never. On the other hand, adaptive indexing reacts to workload changes instantly and refines the physical design with partial and incremental indexing during query processing. It does not collect any statistical information and it does not exploit the idle time intervals between query processing.

Holistic indexing aims to monitor the workload continuously and exploit the collected information to refine the physical design both during query processing *and* during idle time. In contrast to offline and online indexing, and similar to adaptive indexing, in holistic indexing indexes are built partially and incrementally. In this paper, we sketch the research space towards realizing database kernels with holistic indexing.

Paper Structure. The rest of the paper is structured as follows. Section 2 provides background discussion and an overview of related work. Then, Section 3 sketches the research path and the new challenges brought by holistic indexing. Section 4 presents a brief proof of concept experimental analysis and then Section 4 concludes the paper.

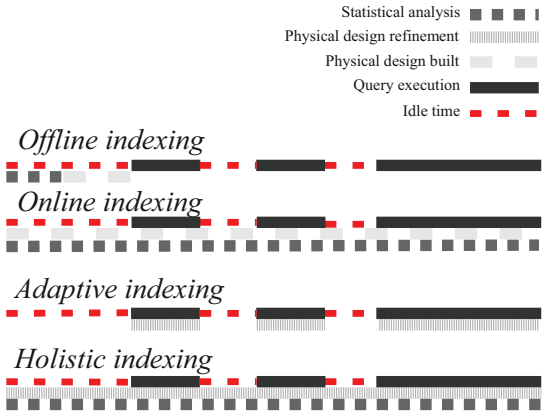

Figure 1: Query sequence evolution with indexing.

2. RELATED WORK

The last fifteen years, there has been extensive work on automated physical database design. The literature can be separated into three areas; offline, online and adaptive indexing. In this section, we briefly sketch the main concepts in each area and we highlight the main differences with our vision towards holistic indexing.

Offline Indexing. One of the earliest seminal works on automated physical database design appeared in 1997 [5]. Among the novel techniques that this paper introduced are the "what-if" API and the dependence on the optimizer. The former refers to hypothetical candidate indexes that instead of being materialized, they are simulated. The latter refers to the use of optimizer cost estimations to decide an appropriate set of indexes. In general an offline auto-tuning tool takes as input a known representative workload W of queries and examines various combinations of candidate indexes. The optimizer costs help determine the expected cost to run W with a candidate index configuration. In the end, the tool outputs a configuration which the database administrator may choose to implement.

Since then, several research efforts have helped to push the state of the art further in offline indexing [1, 2, 3, 6, 17]. The fundamental limitation of offline indexing appears when we cannot safely predict or know the workload a priori.

Online Indexing. Online indexing addresses this limitation of offline indexing. The main idea is that the system continuously monitors the workload and the performance and tries to periodically reevaluate the physical design as opposed to making all decisions a priori.

System COLT was one of the first online indexing approaches [16]. COLT monitors the workload continuously, collecting statistics. Periodically, after specific epochs, e.g., every N queries, the physical design is reconsidered and new indexes may be created or old ones are dropped. Similarly, a later approach on online indexing improves by requiring less calls to the optimizer to obtain cost estimations [4]. Moreover [4] takes into consideration additional overheads for online index tuning, e.g., index interactions. Soft in-

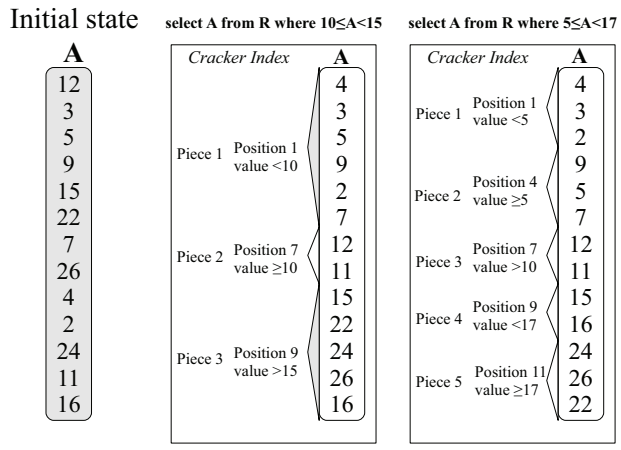

Figure 2: Adaptive indexing.

dexes push the state of the art even further by reducing the index creation cost [15]. The idea is that the scan for building an index online is shared with possible concurrent queries on the same data.

The fundamental limitation of online indexing is that it imposes significant overhead during query processing, i.e., the queries that happen to arrive during the tuning period face a significant penalty. Thus, online indexing is best for relatively dynamic scenarios where offline indexing cannot cope. At the same time the workload should not change very often causing continuous and costly physical design adjustments.

Adaptive Indexing. Adaptive indexing, the most recent indexing approach comes to tackle the problem of fully dynamic and unpredictable workloads [7, 8, 9, 10, 11, 12, 13, 14]. Database cracking was the first adaptive indexing approach [12] proposed in the context of column-stores. Indexes are built partially and incrementally during query processing and according to query predicates. A simplified example is shown in Figure 2; with every query the underlying storage changes, adapting to the queries. Each query is treated as a hint on how the data should be stored and triggers a clustering of the underlying columns. Future queries exploit past clustering but also introduce new clusters. With more queries arriving, columns become more and more structured, with smaller and smaller pieces. As the partitioning information increases, performance improves. The incremental indexing actions are fully integrated with query processing and are performed by the select operators. Database cracking relies on the bulk processing property of modern column-stores to consume and cluster a full column at a time and in one go [12]. Cracking has been extended to support updates [11], multi attribute queries [13], partial cracking [13], concurrency control [7], partition-merge -like logic [9, 14]. In addition, [8] shows how to benchmark adaptive indexing techniques while [10] shows how to be robust on query workloads via stochastic cracking.

The main limitation of adaptive indexing is that it does not exploit any available knowledge or slack time; the whole design is based on instant reaction to incoming queries. However, in modern applications such as social networks or web logs, we may have bursts of queries followed by long stretches of idle time.

Holistic indexing. Our vision, holistic indexing, comes to address all limitations mentioned for the current approaches while the goal is that at the same time we maintain all the nice properties. The system monitors the workload continuously like online indexing but also instantly adapts like adaptive indexing. It can use offline idle time like offline indexing but similar to adaptive indexing it does not have to necessarily create full/complete indexes; it can rely on incremental adaptive indexes which it continuously refines either triggered by queries or by statistics collection and idle time. In addition, similar to adaptive indexing methods, our goal is to create a new kernel that integrates this functionality in its engine as opposed to relying on external tools; this allows for performance enhancements which are hard otherwise.

3. HOLISTIC INDEXING

The previous sections gave the basic background and motivation for holistic indexing. In this section, we give a first sketch of the main challenges and research paths towards realizing the holistic indexing vision.

Main Parameters. Workload knowledge and idle time are the two critical parameters that have a decisive effect on physical database design. Knowledge, or statistical information, about the workload is used to determine the set of relevant indexes that will have the maximum benefit. Idle time is needed in order to perform any workload analysis and critically in order to actually prepare the physical design.

Existing Cases. There are two cases that have been studied in the past, namely the two extreme cases; (a) when there is enough idle time and workload knowledge which is the case where offline indexing excels and (b) when there is no idle time or workload knowledge which is when adaptive indexing excels.

New Challenges. Despite the extensive research on the index selection problem, there are still cases that are not covered adequately by the literature. These cases are the main focus in our research and occur in dynamic and exploratory environments where the mix of idle time and workload knowledge is not always clearly distinguished as in the past. We discuss these cases below.

Some Idle Time, and Enough Knowledge. A natural case to consider is when we simply do not have enough idle time to create all necessary indexes. That is, even if our workload knowledge is adequate such that we know exactly which indexes we would like to create, the available time is not enough to actually materialize these indexes. Still, though, some idle time is available and we would like to exploit it. Such cases may typically occur at initialization time, i.e., before the first few queries arrive. However, this scenario is also valid for online analysis, i.e., the system monitors the workload and acquires a good understanding about the workload patterns. Then, an amount of idle time appears because there is a pause in incoming queries. We may know exactly how much idle time we have or, more typically, we will not know. The question that arises then is what we can do during this idle time to exploit the available knowledge such that we are better prepared for future queries.

Spread Resources with Adaptive Indexes. Naturally, one approach would be to choose and build one index at a time, i.e., starting with the index that is expected to have the biggest global impact in the expected workload; once idle time is over, we stop indexing. Such approaches can be quite easily implemented by modern offline analysis tools. However, by investing on only a few or even on only one index may have a minimal impact to the workload. For example, if 5 minutes of idle time appear, then this is not enough to build even a single index in a reasonably sized database. Instead, we investigate the option of exploiting partial and incremental indexes inspired by adaptive indexing approaches. For example, assume that instead of building a single index at 100%, holistic index may choose to build 20 indexes at 5% each.

With the ability to build partial and incremental indexes, the question is how to spread the idle time and the resources to the various candidate indexes. Assuming that the amount of idle time available will not typically be known, then it makes sense to spread the resources across multiple indexes, instead of concentrating on

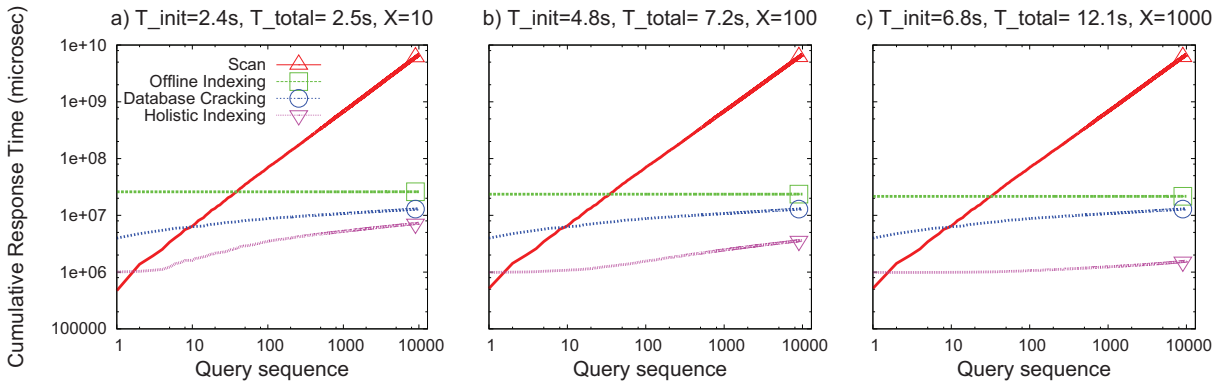

Figure 3: Gains of holistic indexing during a query sequence (*X*: index refinements in each idle time window, Time_sort=28.4s).

only one index at a time. In technical terms this can be translated in applying random database cracking actions on specific columns, i.e., if a given collection of columns is known to be relevant to the workload, then we can start applying random cracking actions in a round robin fashion. A more sophisticated approach can even rank the columns depending on the frequency of appearance in the workload, their size, etc.

Modeling. Our approach is to create a cost model which continuously monitors several parameters and can give us the answer to the question: "if we detect a couple of idle milliseconds on which column should we apply a random crack action?". We observe that once columns are cracked enough such that pieces fit into the CPU caches, then performance does not further improve by extra index refinement. By monitoring how many pieces exist in each crack column and thus knowing the average size of pieces in each column, we know how far away we are from the optimal for each index. This allows the creation of a ranking scheme which is continuously maintained during query processing and during auxiliary tuning actions; it is always up to date and ready to reveal what the next tuning action should be if some idle time appears.

Continuous Tuning. The approach described above leads to a continuous tuning scheme, i.e., if queries do not trigger adaptive indexing, idle time is detected and the system uses statistics to continue triggering adaptive indexing -like actions.

No Knowledge. A special case of the above scenario is that there is no workload knowledge available. For example, such a scenario can occur immediately when we initiate a system with some data. Then, holistic indexing can use catalog information and start to randomly spread tuning actions across many columns.

No Time. Another special case is when we keep collecting good statistics but no idle time appears such as we can exploit it. As in original adaptive indexing, holistic indexing continuously refines the physical design, i.e., during select operators of running queries but now a new opportunity appears. Holistic indexing targets to exploit knowledge to improve even more; even when no idle time appears. For example, assume a query q with a select operator on a given column A which leads into an index refinement action by cracking based on predicate p of q. Holistic indexing may choose to introduce further random cracks on column A if it detects that this column and this value range is rather hot for this part of the workload, e.g., because more than n queries cracked this column in the past.

Putting it All Together. Overall our goal is to create a continuous tuning system where numerous partial and incremental indexes co-exist and are continuously refined whenever such a chance occurs during query processing or during idle time. The challenges imposed towards realizing this vision spread across multiple areas. For example, there are algorithmic issues to tackle regarding how to properly apply multiple tuning actions in one go over a single index such that we can exploit a big chunk of workload knowledge efficiently. Similarly, there are database architecture issues to properly design a system that can continuously detect and exploit idle time as well as to continuously monitor index refinement actions. Our goal is to design and implement such a system on top of modern column-stores.

4. HOLISTIC INDEXING OPPORTUNITIES

In this section, we present a brief experimental analysis to demonstrate the strong potential benefits of holistic indexing. We compare holistic indexing with database cracking, offline indexing and with plain scans. All experiments and design are on top of the open source column-store, MonetDB. For the proof of concept's analysis, our holistic indexing implementation is a hand-tuned variation of the database cracking module in MonetDB, where we manually induce random auxiliary cracking actions during idle time. In addition, we artificially induce and control idle time.

We use a 3.40 GHz Intel(R) Core(TM) i7-2600 processor equipped with 16 GB RAM. The operating system is Fedora 16. The experiments are on a relational table that consists of 10 attributes $[A_1,...,A_{10}]$ each containing 10^8 uniformly distributed integers in $[1, 10^8]$. Each query is a select project query with selectivity 1%. The value range requested by each query is random. All queries are of the following form:

$$\text{select } A_i \text{ from R where } A_i \geq \text{low and } A_i < \text{high}$$

Exp1:Single Column Experiment. Our first experiment aims to show the strong potential in exploiting idle time with holistic indexing. The query workload consists of 10^4 queries. We manually enforce an amount of idle time T before the first query arrives. After the first query, we again manually enforce idle time every 10^2 queries. For ease of presentation and experimentation, in this proof of concept analysis we assume as idle time the time needed to apply X random index refinement actions.

Figure 3 shows the results. On the x-axis queries are ranked in execution order while the the y-axis represents the cumulative response time as the query sequence evolves. Each graph in Figure 3 uses a different X and thus a different amount of idle time. For example, the experiment for Figure 3(a) enforces 10 random actions in each idle time window which results in a total amount of 2.5 seconds idle time across the whole query sequence.

The performance of plain scans, i.e., when no indexing is available, remains stable regardless of the amount of idle time as there is no way to exploit this free time. The same is true for adaptive indexing; database cracking is only triggered by incoming queries. Cracking continuously improves performance but only as long as

Indexing	X=10	X=100	X=1000
Scan	6746 s	6746 s	6746 s
Offline	28.5 s	28.5 s	28.5 s
Adaptive	13 s	13 s	13 s
Holistic	7.3 s	3.6 s	1.6 s

Table 2: Exp1: Overall gains of holistic indexing in total time. *X*: index refinements in each idle time window.

queries arrive to trigger index refinement. Offline indexing on the other hand improves when more idle time is available but does so only marginally as, in general, it can exploit only the idle time, which appears before the very first query. In Figure 3 it takes 28 seconds to completely sort the respective column, i.e., to create the complete index assuming a priori workload knowledge. However, the time T_{init}, i.e., the a priori idle time is much smaller and thus queries start arriving before the index is ready and have to wait for indexing to finish.

Contrary to other approaches, holistic indexing demonstrates a strong potential by managing to exploit all idle time in a way that can benefit future queries, i.e., by forcing auxiliary index refinements actions. As the available idle time grows, i.e., from Figure 3(a) to Figure 3(b) and then to Figure 3(c) holistic indexing improves performance even more. The total impact of holistic indexing is also shown more clearly in Table 2 which reports the total time needed to run all 10^4 queries for each case.

Exp2:Multi-Column Experiment. The previous experiment demonstrated that exploiting idle time can bring significant improvements. Our next proof of concept experiment demonstrates a more realistic scenario where multiple columns are indexed. This time the workload is known and ideally we would like to index all 10 columns of our table, creating 10 single column indexes. However, the restriction in our scenario is that the idle time available a priori is enough to build only 2 of those indexes fully. After this time, queries on all 10 columns arrive (in a round robin fashion).

Figure 4 shows the results. Offline indexing exploits the idle time to build two full indexes. In this case, the choice is random as all columns have exactly the same weight in the expected workload. Thus, when queries arrive, the system can exploit efficient binary searches for the select operators in the query plans of the indexed columns while for the rest of the columns it has to rely on plain scans. In other words only 20% of the queries can exploit indexing. For example, the first two queries in Figure 4 are on the indexed attributes and thus they enjoy good performance. However, as of the third query, queries on non indexed attributes arrive and the cumulative response time grows quickly.

On the contrary, with holistic indexing we build all 10 indexes a priori, but we apply only 100 index refinement actions on each one. Specifically, we apply 100 random cracking actions on each index. The set up of the experiment is done such that the amount of time needed to sort 2 columns with offline full indexing is the same as the time needed to run 100 cracking actions on each one of the 10 columns. In this case this time is 55 seconds which is assumed to be the idle time in this proof of concept experiment. By exploiting this idle time holistic indexing can optimize all needed columns and thus every single query in the workload may benefit instantly as opposed to only 20% of the queries with full indexing. In addition, with every query the indexes are further refined. In this way, in the end of the query sequence holistic indexing has materialized a 2 orders of magnitude benefit over full indexing. Only for the first two queries holistic indexing is slower because all queries so far are on the fully indexed attributes. However, once all workload patterns appear holistic indexing quickly gains over full indexing.

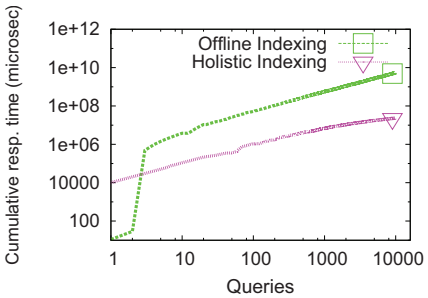

Figure 4: Gains of holistic indexing with multiple indexes.

5. CONCLUSIONS

As we enter an era with more and more dynamic and exploratory applications over big data, proper physical design choices become even more crucial towards achieving good performance and resource management. Holistic indexing aims to design a new database kernel that continuously tunes, both during query processing and during idle time. Every opportunity to improve the physical design is exploited by continuously tuning incremental indexes and continuously maintaining statistics about the system usage. Holistic indexing is inspired by the strong literature and experiences in the indexing research field but at the same time it brings a new mentality regarding how and when we tune. As such it creates a plethora of interesting research problems in the indexing and database architecture fields.

6. REFERENCES

[1] S. Agrawal, S. Chaudhuri, L. Kollar, A. P. Marathe, V. R. Narasayya, and M. Syamala. Database Tuning Advisor for Microsoft SQL Server 2005. In *VLDB*, 2004.

[2] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in sql databases. In *VLDB*, 2000.

[3] N. Bruno and S. Chaudhuri. Automatic Physical Database Tuning: A Relaxation-based Approach. In *SIGMOD*, 2005.

[4] N. Bruno and S. Chaudhuri. An online approach to physical design tuning. In *ICDE*, 2007.

[5] S. Chaudhuri and V. Narasayya. An efficient cost-driven index selection tool for Microsoft SQL Server. In *VLDB*, 1997.

[6] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zaït, and M. Ziauddin. Automatic sql tuning in oracle 10g. In *VLDB*, 2004.

[7] G. Graefe, F. Halim, S. Idreos, H. Kuno, and S. Manegold. Concurrency control for adaptive indexing. In *PVLDB*, 2012.

[8] G. Graefe, S. Idreos, H. A. Kuno, and S. Manegold. Benchmarking adaptive indexing. In *TPCTC*, 2010.

[9] G. Graefe and H. A. Kuno. Self-selecting, self-tuning, incrementally optimized indexes. In *EDBT*, 2010.

[10] F. Halim, S. Idreos, P. Karras, and R. H. C. Yap. Stochastic database cracking: Towards robust adaptive indexing in main-memory column-stores. In *PVLDB*, 2012.

[11] S. Idreos, M. Kersten, and S. Manegold. Updating a Cracked Database. SIGMOD 2007.

[12] S. Idreos, M. Kersten, and S. Manegold. Database Cracking. In *CIDR*, 2007.

[13] S. Idreos, M. L. Kersten, and S. Manegold. Self-organizing tuple reconstruction in column-stores. In *SIGMOD*, 2009.

[14] S. Idreos, S. Manegold, H. A. Kuno, and G. Graefe. Merging what's cracked, cracking what's merged: Adaptive indexing in main-memory column-stores. *PVLDB*, 4(9):585–597, 2011.

[15] M. Lühring, K.-U. Sattler, K. S. 0002, and E. Schallehn. Autonomous management of soft indexes. In *ICDE Workshops*, 2007.

[16] K. Schnaitter et al. COLT: Continuous On-Line Database Tuning. In *SIGMOD*, 2006.

[17] D. C. Zilio, J. Rao, S. Lightstone, G. M. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden. DB2 Design Advisor: Integrated Automatic Physical Database Design. In *VLDB*, 2004.