# Reminiscences on Influential Papers

This issue's contributors chose papers that address challenges at the heart of database systems: physical design tuning for index selection and transaction isolation levels. Both contributions emphasize the elegant, modular, and long-lasting design choices of the respective work. Enjoy reading!

While I will keep inviting members of the data management community, and neighboring communities, to contribute to this column, I also welcome unsolicited contributions. Please contact me if you are interested.

Pınar Tözün, *editor*
IT University of Copenhagen, Denmark
pito@itu.dk

---

**Renata Borovica-Gajic**
University of Melbourne, Australia
renata.borovica@unimelb.edu.au

Surajit Chaudhuri and Vivek Narasayya.
***An Efficient, Cost-Driven Index Selection Tool for Microsoft SQL Server.***
In Proceedings of the International Conference on Very Large Data Bases (VLDB), pages 146–155, 1997.

Pondering on which paper impacted you most is hard - so when Pınar invited me to contribute to this column, I had a hard time picking it. Yet, I immediately knew which thread of papers had the greatest impact on my work. Thus, as is only natural, I chose the first paper from this thread that started an avalanche of amazing work in physical design tuning of databases through the AutoAdmin project.

The AutoAdmin project started in mid-1996 at Microsoft, with the goal of developing technology that makes database systems more self-tuning and self-managing. The task was a tall order at the time, when the database administrators were deeply involved in performance tuning of databases, and when very little automation had seen the light in production systems. Yet, the problem of choosing the right physical design was critical, since together with the execution engine and the optimizer, the physical database design determines the efficiency of executed queries, and hence ultimately it affects the overall user experience when using a database. The VLDB 1997 paper was the first paper of this line of work, and it looked at recommending indexes (a.k.a. index selection) as one (important) aspect of physical database design.

The index selection problem had been studied since the early 70's and the importance of the problem was well recognized at the time. The goal of automated index selection is to automatically pick a set of indexes, referred to as a configuration, estimated to (maximally) boost the performance of the given workload, often under memory budget constraints. While the index selection problem was proven to be NP-complete (through the work of Shapiro in 1983), the VLDB 1997 paper presented a practical, efficient, and elegant solution to this rather challenging problem. And this is what struck me the most about this paper. The authors had taken an extremely challenging problem, broke it down into small pieces, and proposed a set of elegant techniques to address all the challenges efficiently and effectively, while keeping the entire architecture fully modularized. This end-to-end systems approach not only resulted in an extremely clean design, but also opened doors for future improvements, since any component could easily be replaced by another, more efficient algorithm in the future. Thanks to this modularized design, over a dozen of papers had appeared in the following decade, many leveraging the proposed architecture and adding new dimensions to the problem (e.g., considering materialized views, partitioning, statistics, etc).

So, let's dive into the architecture. The index selection tool proposed in this paper consists of candidate index selection module, configuration enumeration module, "what-if" index creation module, cost evaluation module and multi-column index generation module.

The index selection module proposes a set of candidate indexes for the given workload. Since the number of possible index candidates is too large (exponential in the number of columns and tables), the authors proposed a heuristic of determining the best configuration for each query independently, and only consider indexes belonging to one or more of such best configurations as a candidate index set. This neat trick allowed them to use their own tool to generate candidate indexes, since each individual query could be considered as a workload consisting of that single query - which is yet another example of the elegance of the approach.

Next, out of N candidate indexes chosen as the overall candidate set, the goal of configuration enumeration modules is to pick the best K. Exhaustively enumerating all possible combinations is again too large, and the authors employed a greedy incremental approach where in each round the algorithm selects an optimal configuration of the size M, where M <= K, greedily adding the next most beneficial index to the existing configuration until M == K or until no further cost reduction is possible.

Probably the most influential component is the "what-if" API in the query optimizer. The "what-if" component stood the test of time and remained a critical component of many subsequent tools as well as served as an independent component for manual performance tuning of databases employed by database administrators. The "what if" API simulates the presence of different physical design structures without materializing them. When the index selection tool needs to evaluate the cost of a workload, it simulates the presence of the configuration by loading the catalog tables of a database system with metadata and statistical information about defined structures. It then optimizes the queries from the workload in a no-execute mode, in which the optimizer returns a plan and a cost estimate for each query without executing them. Using the query optimizer's cost estimates as the basis for the physical design tool has several advantages. First, it can guarantee that any proposed index, if materialized, will actually be used by the optimizer. Second, it is much more efficient than paying the cost to materialize candidate indexes. Finally, as the query optimizer's cost model evolves over time the tool will just benefit from those improvements, which is yet another forward-looking aspect of this paper.

Still, calling the optimizer to cost all possible configurations across the entire workload may be too expensive. To reduce the number of optimizer calls, the authors introduce a concept of atomic configurations and show that it is sufficient only to evaluate all atomic configurations across the workload, as the cost of all other configurations could be derived from atomic configurations without requiring any additional optimizer calls. On top of reducing the number of atomic configurations to cost, the authors propose a way of reducing the cost of evaluating atomic configurations by costing only a subset of relevant indexes from the configuration whose columns are part of the query set. By caching the results of the optimizer calls for atomic configurations, optimizer invocations for other configurations for the same query could be eliminated (often even by orders of magnitude).

Finally, the authors proposed a search algorithm to incrementally examine the space of multi-column indexes. The approach the authors employed is to iteratively expand the space of multi-column indexes by choosing only the winners of one iteration and augmenting the search space of the next iteration by expanding such winners with an additional column. This heuristic allows for a structured and tractable exploration of what would be an enormous space of alternative choices.

The impact of this paper was manyfold. It was the first approach that looked at creating an automated tool for index selection. This work formed the basis of the Index Tuning Wizard (ITW) that shipped in Microsoft SQL Server 7.0, and many commercial vendors have followed suit, using somewhat similar techniques. The paper rightfully received the 10-Year Best Paper Award at VLDB 2007 due to its novelty, clean architecture but also the broad impact it had on the research community and database vendors at large. As the work progressed, so did the product, resulting in a fully-fledged Database Tuning Advisor (DTA) that shipped as part of Microsoft SQL Server 2005. DTA went beyond index selection and supported selection of materialized views, and horizontal partitioning. Today's SQL Server DTA also supports selection of partial indexes and columnstore indexes.

On a personal note, I first discovered this paper more than a decade ago when embarking on a PhD journey, and immediately appreciated its elegant architecture, and pragmatic approach to solving challenging problems. This pragmatic approach stayed with me throughout my professional life, for which I will be forever grateful to the authors. Finally, while we as a research community naturally evolve and sometimes outgrow research problems, the problem of automated physical design tuning is more important than ever, considering the strong presence of cloud platforms where the workload complexity and the absence of on-premise database administrators make such tools a necessity. When we will be able to completely solve this problem is hard to say, since as authors conclude in their 10-year paper summary (published in VLDB 2007) "it will probably be impossible to make database systems self-tuning by a single architectural or algorithmic breakthrough" and that "demand for self-manageability could lead to development of newer structured store that is built grounds-up with self-manageability as a critical requirement". Almost two decades later, we are seeing first strides towards making database systems self-driving from the ground up. And I am sure that the next two decades will be as exciting, with fully adaptive and self-driving systems becoming common.

---

**Bailu Ding**
Microsoft Research Redmond, USA
bailu.ding@microsoft.com

Atul Adya, Barbara Liskov, and Patrick E. O'Neil.
***Generalized Isolation Level Definitions.***
In Proceedings of the 16th International Conference on Data Engineering (ICDE), pages 67-78, 2000.

When I first started working on transaction processing, one of the first papers I read was "Generalized Isolation Level Definitions" by Adya, Liskov, and O'Neil. This paper argues that isolation levels should be a logical property of transactions, rather than being defined based on how transactions are implemented in a locking-based concurrency control scheme, as was the case in earlier ANSI-SQL 92 standards. The authors propose a new way to define transaction isolation levels based on the dependencies of transactions, which decouples the abstraction of isolation levels from their implementation. This definition can be applied to different concurrency control schemes, such as optimistic concurrency control or multi-version concurrency control. I was impressed by this work for its elegance and practicality. The technique of analyzing isolation levels based on transaction dependency graphs has also become a crucial tool used in my later work on relaxed concurrency control for transaction processing, such as in watermarking [1] and transaction reordering [2].

[1] Bailu Ding, Lucja Kot, Alan Demers, and Johannes Gehrke."Centiman: Elastic, High Performance Pptimistic Concurrency Control by Watermarking." In Proceedings of the Sixth ACM Symposium on Cloud Computing, pages 262-275, 2015.

[2] Bailu Ding, Lucja Kot, and Johannes Gehrke. "Improving Optimistic Concurrency Control through Transaction Batching and Operation Reordering." In Proceedings of the VLDB Endowment 12.2, pages 169-182, 2018.