

Dynamic Detection and Administration of Materialized Views based on the Query Graph Model

Andreas Luebcke, Ingolf Geist
Otto-von-Guericke-University Magdeburg
School of Computer Science
Institute for Technical and Business Information Systems
D-39016 Magdeburg, Germany
P.O. Box 4120
{andreas.luebcke, ingolf.geist}@ovgu.de

Ronny Bubke (Alumni)
Otto-von-Guericke-University Magdeburg
School of Computer Science
Institute for Technical and Business Information Systems
D-39016 Magdeburg, Germany
P.O. Box 4120

Abstract

Database Management Systems (DBMS) are complex software systems which are used in an increasing number of applications. To obtain the best performance for business relevant applications, e.g., Decision Support Systems (DSS). DBMS have to be tuned and monitored. In order to reduce the costs researchers and DBMS vendors have focused on self-management techniques for recent years. In particular, materialized views were used to pre-compute results in Online Analytical Processing (OLAP) systems to improve the performance, e.g., the query processing time. In this work, we propose an automatic selection of materialized views based on statistics about a workload that is represented in the Query Graph Model (QGM). Our approach allows the mapping of workload queries into a map graph and materializes views by using access statistics and cost functions. We evaluate our approach with the TPC-H benchmark and compare it to a commercial DBMS.

1. Introduction

Just today Database Systems (DBS) become more and more complex. Hence, it will be more difficult to obtain the best performance in critical applications. In or-

der to simplify administration and tuning of DBMS, software vendors [1, 18] and researchers have focused on self-management techniques for several years. Our approach is located in this area, too. It tries to reach the optimal performance at any time, e.g., index tuning [6], materialized views [7], query plans [14], and SQL optimization [8].

In this work, we focus on the selection and management of Automatic Summary Tables (AST) [12]. ASTs are used to improve the query performance of frequently occurring queries by pre-computing complete operations, e.g., joins, aggregations. The original queries are rewritten to use ASTs. Further performance improvement can be achieved by creation of additional indexes over ASTs [14]. Unfortunately, ASTs cause costs in two ways. First, they consume disk space. Second, ASTs have to be updated if the corresponding base data is modified.

This paper focuses on the automatic recommendation of ASTs based on a given workload. The workload and the gathered statistics are represented in a map graph. The Query Graph Model (QGM) was developed for optimization using ASTs [17].

2. Related Work

For an all-embracing solution for self-tuning a DBS using ASTs, several problems have to be considered. At first,

we have to find the optimal selection of ASTs. At second, we have to decide which AST has to be used to compute a query. Hence, an efficient administration of the ASTs is needed for the QGM. Related work about these problems can be found in literature [7, 11, 13]. The presented approaches in [3, 4, 10, 16] aim at dynamical administration of ASTs. The differences between them are discussed in [17]. The fundamentals of cost models and cache strategies are described in [15], respectively the Query Graph Model [5].

The QGM is a directed acyclic graph with a root node and a kind of notation to map SQL queries. The mapping is similar to conventional query plans [14]. The nodes are called *boxes* in the QGM and the data flow is represented by edges between the *boxes*. The QGM comprises two types of *boxes*, *SELECT* and *GROUP BY* boxes. Using these box types, the functionality of SQL can be mapped, e.g., selections, projections, joins, all scalar expressions, groupings. An Example for a query is shown in **Figure 1**.

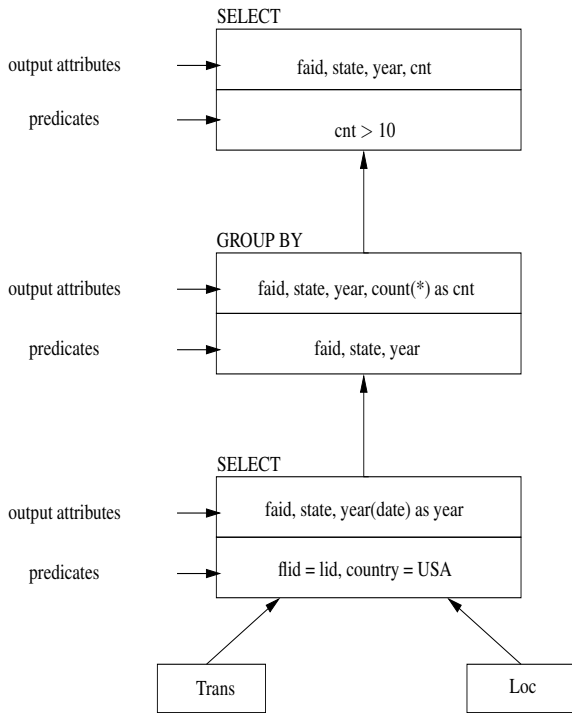


Figure 1. Example query as QGM graph [17]

3. Dynamic Creation and Administration of Materialized Views

We use the QGM representation of queries and adopt the proposed techniques of the WATCHMAN approach [15]. We assume each query of a workload is represented in the QGM.

The QGM is extended by a third box type *TABLE* which represents the base tables of the system and contains all attributes of them. We store the size and the cardinality of each output attribute. Furthermore, we define common features for all types of *boxes*, i.e., tuple count (TpCnt), tuple size (TpSz), operation process costs (OPC), and reference counter (RefCnt).

If the workload contains all base tables then the map graph (MG) also contains all base tables, i.e., the MG is the set of all query trees from the workload mapped into the QGM. Hence, it is necessary to match the *query boxes* to existing *MG boxes*. The query and the MG will be compared in a bottom up way. If a *query box* does not match to the *MG boxes*, we have to insert this *query box* as a *child box* into the MG. Restructuring of the MG is necessary because as many queries as possible should use the same sub-graph. This requirement guarantees a performant administration by using the MG.

Dynamic query processing replaces queries, but we do not only want to replace them. In addition, we want to use common parts and find similarities to encapsulate them into a new *box*. With the merging process, we can reduce the complexity of the MG.

Changing workloads are not optimal represented by static AST selection. Hence, we use aging algorithms to estimate a more exact profit for the profit calculation. The weighted assumption is a capable solution to assess a workload. Large QGM graphs can be avoided using a lower limit of the reference counter.

We select the ASTs to materialize using a profit function based on our defined common features which is defined as follows:

$$profit(b) = \frac{OPC(b)}{TpCnt(b) * TpSz(b)} * RefCnt(b)$$

4. Experimental Evaluation

We evaluated our approach via an test application (written in Java). It was restricted to parts of DBMS which were needed for the evaluation. We compared our results to the IBM DB2 Design Advisor (version 9.1).

First, we discuss the overhead of our approach. Basically, our approach has to guarantee that the profit of resulting operations is bigger than their costs. We assumed the costs for AST selection, merging, and restructuring are at most $O(n^2)$ with n the maximum number of *parent boxes*.

We selected eight appropriate queries from the TPC-H workload, and additionally we derived five queries from one of the eight queries to achieve a more general set of predicates. The modified queries still share common predicates. All experiments were performed on an AMD Athlon 1700XP with 1GB RAM. The resulting execution time was

about 437 seconds without any ASTs. In contrast, the Design Advisor estimated six ASTs and the execution time was about 67 seconds. Our system estimated eight ASTs for this workload and the execution time was approximately five seconds.

Compared to IBM DB2 Design Advisor, the very serious improvement of the execution time is caused by the much more specific definition for ASTs of our approach.

5. Conclusion

The present work addresses the problem of automatic selection and administration ASTs. All queries and statistics are represented in an MG based on the QGM. The MG will be constantly updated for the workload.

We have developed algorithms for the merging of queries. These algorithms are evaluated for common and specific views without general assumptions like predicate reduction. We classify the predicates and assign their reference value after their usage. Hence, we have the possibility to map combined regions of a workload to one materialization because the estimation is not only possible on the query level but also on a subset of predicates. Each cluster can be estimated based on its profit derived from the reference value, calculation costs, and size of the relation. Hence, AST definitions can not only be created based on actual queries but also on a subset of predicates. The AST selection algorithms and caching strategies work on the basis of the MG. The evaluation showed similar and better results than a state-of-the-art self-tuning advisor (DB2 Design Advisor).

In future work, we will work on the following research problems. Indexes can be derived perfectly from the MG to obtain automatic *Index Selection*. The *parent boxes* of a materialized *box* contain all predicates which are necessary to create indexes for the view. In our test setup, we use a static decrementation factor, a static upper and lower reference level. For dynamic workloads, the number of *boxes* is changing permanently. Hence, the estimations of materialized views will not be optimal, i.e., the *Static Parameters* have to be dynamically. We assume further optimizations in single parts of the algorithms, e.g., in particular the merge pattern, the profit function, and the merging of *boxes* with no common predicate intersection.

References

[1] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In *The VLDB Journal*, pages 496–505, 2000.

[2] I. M. Author. Some related article I wrote. *Some Fine Journal*, 99(7):1–100, January 1999.

[3] Z. Bellahsene and P. Marot. Materializing a set of views: Dynamic strategies and performance evaluation. In *IDEAS*, pages 424–430, 2000.

[4] R. G. Bello, K. Dias, A. Downing, J. James J. Feenan, J. L. Finnerty, W. D. Norcott, H. Sun, A. Witkowski, and M. Ziauddin. Materialized views in oracle. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 659–664, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[5] D. Bonchev and D. H. R. (Eds.). Chemical graph theory: Introduction and fundamentals. *of Mathematical Chemistry Series. London*, 1, 1990. London, UK: Gordon and Breach Science Publishers.

[6] S. Chaudhuri, M. Datar, and V. Narasayya. Index selection for databases: A hardness study and a principled heuristic solution. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1313–1323, 2004.

[7] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *11th Int. Conference on Data Engineering*, pages 190–200, Los Alamitos, CA, 1995. IEEE Computer Soc. Press.

[8] A. B. Danchenkov and D. K. Bursleson. *Oracle Tuning: The Definitive Reference*. Rampant Techpress, 2006.

[9] A. N. Expert. *A Book He Wrote*. His Publisher, Erewhon, NC, 1999.

[10] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environments. In *The VLDB Journal*, pages 358–369, 1995.

[11] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 205–216, New York, NY, USA, 1996. ACM.

[12] W. H. Immon. *Building the Date Warehouse (4th Edition)*. Wiley, 2005.

[13] I. S. Mumick, D. Quass, and B. S. Mumick. Maintenance of data cubes and summary tables in a warehouse. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 100–111, New York, NY, USA, 1997. ACM.

[14] G. Saake, A. Heuer, and K.-U. Sattler. *Datenbanken: Implementierungstechniken*. mitp-Verlag/Bonn, 2 edition, 2005.

[15] P. Scheuermann, J. Shim, and R. Vingralek. WATCHMAN: A data warehouse intelligent cache manager. In *The VLDB Journal*, pages 51–62, 1996.

[16] D. Srivastava, S. Dar, H. V. Jagadish, and A. Y. Levy. Answering queries with aggregation using views. In *The VLDB Journal*, pages 318–329, 1996.

[17] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh, and M. Urata. Answering complex sql queries using automatic summary tables. *SIGMOD Rec.*, 29(2):105–116, 2000.

[18] D. C. Zilio, J. Rao, S. Lightstone, G. Lohman, A. Storm, C. Garcia-arellano, and S. Fadden. Db2 design advisor: Integrated automatic physical database design. In *In VLDB*, pages 1087–1097, 2004.