Materialized Views and Data Warehouses

Nick Roussopoulos
Department of Computer Science
and
Institute of Advanced Computer Studies
University of Maryland
nick@cs.umd.edu

Abstract

A data warehouse is a redundant collection of data replicated from several possibly distributed and loosely coupled source databases, organized to answer OLAP queries. Relational views are used both as a specification technique and as an execution plan for the derivation of the warehouse data. In this position paper, we summarize the versatility of relational views and their potential.

1 Views

The importance of the "algebraic closedness" of the relational model has not been recognized enough in its 27 years of existence. Although a lot of energy has been consumed on dogmatizing on the "relational purity", on its interface simplicity, on its mathematical foundation, etc., there has not been a single paper with a central focus on the importance of relational views, their versatility, and their yet-to-be exploited potential.

What is a relational view? Is it a program? Is it data? Is it an index? Is it an OLAP aggregate? It is all these. And a lot more. Below I summarize the most important uses, techniques, and benefits pertaining to views. Note that the cited work here is not meant to be exhaustive but representative and easily accessible from my short-term memory.

Proceedings of the 4th KRDB Workshop
Athens, Greece, 30-August-1997

(F. Baader, M.A. Jeusfeld, W. Nutt, eds.)

http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-8/

2 The Multifaceted Form of Views

Relational views have several forms:

- *pure program:* an unmaterialized view is a program specification, "the intension", that generates data. Query modification [Sto75] and compiled queries [ABC$^+$76] were the first techniques exploiting views– their basic difference is that the first is used as a macro that does not optimize until run-time while the second stores optimized execution plans. Such a view form is a pure program with no extensional attachments. Each time the view program is invoked, it generates (materializes) the data at a cost that is roughly the same for each invocation.

- *derived data:* a materialized view is "the extension" of the pure program form and has the characteristics of data like any other relational data. Thus, it can be further queried to build *views-on-views* or collectively grouped [Pap94] to build *super-views*. The derivation operations are attached to materialized views. These procedural attachments along with some "delta" relational algebra are used to perform incremental updates on the extension.

- *pure data:* when materialized views are converted to snapshots, the derivation procedure is detached and the views become pure data that is not maintainable (pure data is at the opposite end of the spectrum from pure program).

- *pure index:* view indexes [Rou82b] and View-Caches [Rou91] illustrate this flavor of views. Their extension has only pointers to the underlying data which are dereferenced when the values are needed. Like all indexing schemes, the importance of indexes lies in their organization, which facilitates easy manipulation of pointers and efficient single-pass dereferencing, and thus avoids thrashing.

- *hybrid data & index:* a partially materialized view [BR96] stores some attributes as data while the rest are referenced through pointers. This form combines data and indexes. B-trees, Join indexes [Val87], star-indexes [RBS96] and most of the other indexing schemes belong to this category, with appropriate schema mapping for translating pointers to record field values. Note that in this form, the data values are drawn directly from the underlying relations and no transformation to these values is required[1].

- *OLAP aggregate/indexing:* a data cube [GBLP96] is a set of materialized or indexed views [GHRU96, RKR97]. They correspond to projections of the multi-dimensional space data to lesser dimensionality subspaces and store aggregate values in it. In this form, the data values are aggregated from a collection of underlying relation values. Summary tables and Star Schemas [RBS96] belong in this form (the latter belongs here as much as in the previous category).

Each of these forms is used by some component of a relational system. Having a unified view of all forms of relational views is important in recognizing commonalities, re-using implementation techniques, and discovering potential uses not yet exploited.

3 Discovery and Re-use of Views

RDBMSs do nothing else but generate or access materialized views 24 hours a day whether these are predefined views, results of compiled queries, ad hoc queries, or even *materialized view fragments* [RCK+95], i.e., temporary results generated during the execution of a larger query. Unfortunately, commercial RDBMSs discard these views immediately after they are delivered to the user or to a subsequent execution phase. The cost for generating the views is for one-time-use only instead of being *amortized* over multiple and/or shared accesses [Rou91].

Caching query (intermediate) results for speeding up intra- and inter-query processing has been studied widely [Fin82, LY85, Rou91, Sel87, Jhi88, DR92, AL80, Rou82b, Rou91, Sel88, Jhi88, RK86b, BALT86, Fin82, LY85, DR92, HS93, RK86b, Han87a, Han87b, JMRS93]. The goals of these studies range from improving query optimization and processing to supporting rules in active databases, to query processing in client-server and distributed/replicated database architectures, to handling time queries, to obtaining efficient update dissemination, to avoiding expensive computations of external predicates, etc. All these techniques have one common underlying theme: *the re-use of views* to save cost.

Amortization and re-use of views can only be possible if they can be discovered by the query optimizer[2] which decides to *plug-in* those views which reduce the cost of the query. The benefits are multiplied in a multi-user environment with a lot of shared access to views. Despite this, only the ADMS prototype has extended the query optimizer and its cost model [CR94a] to include in its plan selection materialized views, ViewCaches, and incremental access methods and a tailored buffer manager , as well as a tailored buffer manager designed to support these access methods [CR93]. However, both IBM and Microsoft plan to incorporate similar constructs in their DB2 and Sequel Server future releases.

The most common technique for discovering views (in any of its forms) is *subsumption* [Rou82a, Fin82, LY85, Rou91, BJNS94]. Subsumption in its most general form is an undecidable problem, but for the most common queries can be reduced to an NP-complete problem. For simple conjunctive query views, it further reduces to polynomial-time and very efficient algorithms [Rou82a, CR94a].

In a data warehouse where query execution and I/O are magnified, the mandate for re-use cannot be ignored. Furthermore, in an OLAP environment, (unlike OLTP), updates come in bulk rather than a few-at-a-time, making incremental update techniques more effectively amortized [RKR97]. Therefore, query optimizers based on materialized view fragments are a necessity. At this point, data warehouses rely solely on users' memory for re-using precomputed summary tables. This severely limits their performance potential.

4 Processing of Views

Now let's examine view processing for all the view forms except for the pure data (snapshots) which are not maintainable. View processing involves view scanning, incremental update, or both applied simultaneously. Scanning and incremental update of views imply special locks, locking protocols [RES93], authorization [RB85], and consistency protocols for asynchronous updates from multiple sources [ZGMHW95].

[1] This is how the indexed form is almost exclusively used although there is no intrinsic reason for not applying a transformation function, other than the identity one, to the underlying values before indexing them- e.g., calibrate the values before entered in a B-tree.

[2] the user cannot be aware of views generated by the system and other users.

I will concentrate here on performance issues.

View scanning in the pure program view form is typically the same as re-execution of the query that created the view. There is no performance benefit for unmaterialized views other than predicting re-execution cost more accurately after the first time. The performance is bad but predictable. Scanning a materialized view has a cost that depends on the ratio of the useful tuples in it to answer a given query, called *density* of the view. For a 100% ratio, scanning a materialized view is optimal because it has all the data for answering the query compacted in a tight storage space. If the density is low, the noise can be more than the amount of useful data. For the index view form, scanning cost can range from near optimal, when the pointers are aligned and point to a tight space, to very high, when pointer dereferencing causes thrashing (similar to unclustering indexes in RDBMSs or in OODBMSs). For this reason, in the index form, it is very important that the pointers be well organized [RK87, AR90] and use a tailored buffer manager [CR93] which avoids thrashing caused by the multi-dimensionality of the view. ViewCache uses a form of puzzle-shaped packed R-trees [RL85] and tailored cache replacement strategies. Cubetrees [RKR97] utilize multi-dimensional compressed and packed R-trees [RL85]. Again, for performance, the organization is the only thing that matters.

Incremental update techniques for views are mature, as they go back for more than a decade of research [BLT86, RK86b, Rou87, Rou91, RES93]. The same techniques were the foundation for the management of replicated data [RK86a, RK86b] which found its way to the log-based replication tools of commercial RDBMSs.

Incremental update of a view depends again on its underlying form. In its un-materialized form the cost of an incremental update is the cost of re-execution. For other forms we must distinguish two cases. The first case occurs when the incremental update is done in real-time during the query execution. In this case, the update is combined with scanning and therefore, the cost of incremental update is subsumed by the scanning cost. This was the main objective of the one-pass incremental update algorithms of ViewCache. The subsumed cost savings are significant and this was shown by comparing worst case analysis estimations against actual timed experiments [RES93]. This was especially true for views-on-views because of the elimination of storing and accessing intermediate results.

The second case is when the incremental update of a view is done at times other than scanning. This is the typical case in a data warehouse where updates from multiple sources are applied asynchronously either when they arrive or at scheduled (often off-line) times. The benefit of combining scanning and updating is not a factor any more. Therefore, minimal dereferencing is a good target optimization. Partially materialized views [BR96] which materialize only the subset of the attributes useful for the incremental update, *outer-joins* instead of joins, or other appropriate attribute caching techniques [Sta89] are best suited. On the other hand, fully materialized views are cumbersome and generate a lot of unnecessary I/O and data movement for just updating views that are to be used in the future.

It should be mentioned here that the issue of *self-maintenance* [GJM96] of views is important. However, the additional information necessary for the incremental update and its storage organization must be well designed since this affects performance. For example, the storage organization of the *deltas* may have an equivalent adverse effects to thrashing if their tuples are scattered in an unclustered space.

5 Data Mining within Views

Views which are materialized or partially materialized contain valuable information in them such as value distributions and other statistical information that are much more accurate than the clumsy ones maintained by the DBMS statistics utility which is run once in a while. Having accurate value distributions avoids those unrealistic assumptions and guestimates based on the uniformity assumption. Since RDBMSs do nothing else but generate materialized views, a smarter system can extract this valuable *meta-data* and, with a query feedback mechanism [CR94b], maintain precise statistics. This was shown to incur no additional I/O cost and have negligible CPU only overhead for a big return, as the error in the estimation is very close to none.

6 Harvesting the Executions of Views

Consider a view as a program again. Whether we execute it to materialize it or to incrementally update it, the system's behavior can be observed during this execution and be used to adapt resource allocation during subsequent executions. For example, buffer page fault behavior during a view's execution can be accurately predicted using regression from a few executions of it [CR93]. Using this information, buffer allocation is done much more efficiently using a marginal gain technique [FNS91]. Clearly, buffers are some of the most important resources to be managed, but other resources such as locks, logs, threads, etc., can be observed during view execution and used to adapt strategies for improving performance.

Repeated materialization of views can also be harvested to obtain patterns of access and use them for *just-in-time* page prefetching from disk. This not only enhances buffer management, but more importantly, reduces context switching overhead. Similar techniques have been engaged successfully in OS studies [PG96] where the patterns are much less predictable than re-materializing a view.

7 Conclusion

Views are the most important asset of the relational model. They provide a uniform conceptual and implementation model of relational programs, derived data, indexes, and aggregated derived data. I can think of a very few things that are so elegant and practical too.

These are my views on views. And, like most views, they are evolving and incrementally updating with time. I am impressed with views' versatility, resilience, and refusal of retiring [Mat96]. As for the yet-to-be discovered uses of views that I promised at the beginning of the paper, I remind you that it is just a speculative view on my part.

References

[ABC+76] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, Jim Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson. "System R: Relational Approach to Database Management". *ACM Transactions on Database Systems*, 1(2):97–137, June 1976.

[AL80] M.E. Adiba and B. G. Lindsay. "Database Snapshots". In *Procs. of 6th VLDB*, 1980.

[AR90] A. Amir and N. Roussopoulos. "Optimal View Caching". *Information Systems*, 15(2):169–171, 1990.

[BALT86] José A. Blakeley, Per Åke Larson, and Frank Wm. Tompa. "Efficiently Updating Materialized Views". In Carlo Zaniolo, editor, *Proc. of the ACM SIGMOD Conference*, pages 61–71, Washington, D.C., May 1986.

[BJNS94] M. Buchheit, M.A. Jeusfeld, W. Nutt, and M. Staudt. "Subsumption Between Queries to Object-Oriented Databases". In *Proc. of the International Conference on Extending Database Technology (EDBT)*, Cambridge, UK, March 1994.

[BLT86] J.A. Blakeley, P.A. Larson, and F.W. Tompa. "Efficiently Updating Materialized Views". In *Proc. of the 1986 ACM SIGMOD International Conference*, pages 61–71, August 1986.

[BR96] Lars Baekgraard and Nick Roussopoulos. "Efficient Refreshment of Data Warehouse Views". Technical Report CS-TR-3642, Dept. of Computer Science, Univ of Maryland, College Park, MD, May 1996.

[CR93] C.M. Chen and N. Roussopoulos. "Adaptive Database Buffer Allocation Using Query Feedback". In *Procs. of the 19th Int. Conf. on Very Large Data Bases*, Dublin, Ireland, 1993.

[CR94a] Chungmin Melvin Chen and Nick Roussopoulos. "The Implementation and Performance Evaluation of the ADMS Query Optimizer: Integrating Query Result Caching and Matching". In *Proc. of the International Conference on Extending Database Technology (EDBT)*, pages 323–336, Cambridge, UK, March 1994.

[CR94b] C.M. Chen and N. Roussopoulos. "Adaptive Selectivity Estimation Using Query Feedback". In *Procs. of the ACM SIGMOD Intl. Conf. on Management of Data*, Minneapolis, Minnesota, 1994.

[DR92] A. Delis and N. Roussopoulos. "Evaluation of an Enhanced Workstation-Server DBMS Architecture". In *Procs. of 18th VLDB*, 1992.

[Fin82] S. Finkelstein. "common Expression Analysis in Database Applications". In *Procs. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 235–245, 1982.

[FNS91] C. Faloutsos, R. Ng, and T. Sellis. "Predictive Load Control for Flexible Buffer Allocation". In *VLDB Conf. Proceedings*, pages 265–274, Barcelona, Spain, September 1991. Also available as UMIACS-TR-91-34 and CS-TR-2622.

[GBLP96] J. Gray, Bosworth, Layman, and Piramish. "Data Cube: A Relational Aggregation". In *Proc. of the 12th Int. Conference on Data Engineering*, New Orleans, February 1996. IEEE.

[GHRU96] H. Gupta, V. Harinarayan, Rajaraman, and J. Ullman. "Index Selection for

OLAP". In *Proc. of of VLDB*, Bombay, India, August 1996.

[GJM96] Ashish Gupta, H. V. Jagadish, and Inderpal Singh Mumick. "Data Integration using Self-Maintainable Views". In *Proc. of the International Conference on Extending Database Technology (EDBT)*, pages 140–144, Avignon, France, mar 25-26 1996.

[Han87a] E. Hanson. "A Performance Analysis of View Materialization Strategies". In *Proc. of SIGMOD Conference on Management of Data*, pages 440–453. ACM, 1987.

[Han87b] Eric N. Hanson. "A Performance Analysis of View Materialization Strategies". In *Proc. of the ACM SIGMOD Conference*, pages 440–453, Brighton, England, September 1987.

[HS93] J.M. Hellerstein and M. Stonebraker. "Predicate Migration: Optimizing Queries with Expensive Predicates". In *Procs. of ACM-SIGMOD*, 1993.

[Jhi88] A. Jhingran. "A performance Study of Query Optimization Algorithms on a Database System Supporting Procedures". In *Procs. of the 14th Intl. Conf. on VLDB*, 1988.

[JMRS93] C. S. Jensen, Leo Mark, Nick Roussopoulos, and Timos Sellis. "Using Differential Techniques to Efficiently Support Transaction Time". *VLDB Journal*, 2(1):75–111, 1993.

[LY85] P.-Å. Larson and H. Z. Yang. "Computing Queries from Derived Relations". In *Procs. of the 11th Intl. Conf. on VLDB*, pages 259–269, 1985.

[Mat96] In Inderpal Mumick and Ashish Gupta, editors, *Proceedings of the Workshop on Materialized Views: Techniques and Applications*, Montreal, Canada, jun 7 1996.

[Pap94] Y. Papakonstantinou. "Computing a Query as a Union of Disjoint Horizontal Fragments". Technical report, Department of Computer Science, University of Maryland, College Park, MD, 1994.

[PG96] R. Hugo Patterson and Garth A. Gibson. "Exposing I/O Concurrency with Informed Prefetching". In *IPDIS*, 1996.

[RB85] N. Roussopoulos and C. Bader. "Dynamic Access Control for Relational Views". *Information Systems*, 10(3):361–369, August 1985.

[RBS96] Inc. Red Brick Systems. "Star Schemas And STARjoin Technology". Technical report, 1996. White Paper.

[RCK+95] N. Roussopoulos, C.M. Chen, S. Kelley, A. Delis, and Y. Papakonstantinou. "The ADMS Project: Views "R"Us". *Data Engineering Bulletin*, 18(2):19–28, June 1995.

[RES93] Nick Roussopoulos, Nikos Economou, and Antony Stamenas. "ADMS: A Testbed for Incremental Access Methods". *IEEE Transactions on Knowledge and Data Engineering*, 5(5):762–774, October 1993.

[RK86a] N. Roussopoulos and H. Kang. "Preliminary Design of *ADMS*±: A Workstation-Mainframe Integrated Architecture for Database Management Systems". In *Proc. 12th International Conference on VLDB*, Kyoto, Japan, August 1986.

[RK86b] N. Roussopoulos and H. Kang. "Principles and Techniques in the Design of *ADMS*±". *Computer*, 19(12):19–25, 1986.

[RK87] N. Roussopoulos and H. Kang. "A Pipeline N-way Join Algorithm based on the 2-way Semijoin Program". *IEEE Trans. on Knowledge and Data Engineering*, 3(4):486–495, December 1987.

[RKR97] Nick Roussopoulos, Yannis Kotidis, and Mema Roussopoulos. "Cubetree: Organization of and Bulk Updates on the Data Cube". In *Proc. of the ACM SIGMOD Conference*, Tucson, Arizona, May 13-15 1997.

[RL85] N. Roussopoulos and D. Leifker. "Direct Spatial Search on Pictorial Databases Using Packed R-Trees". In *Proc. ACM SIGMOD*, pages 17–31, Austin, Texas, May 1985.

[Rou82a] N. Roussopoulos. "The Logical Access Path Schema of a Database". *IEEE Trans. on Software Engineering*, SE-8(6):563–573, 1982.

[Rou82b] N. Roussopoulos. "View Indexing in Relational Databases". *ACM TODS*, 7(2):258–290, 1982.

[Rou87] Nick Roussopoulos. "Overview of ADMS: A High Performance Database Management System". In *Fall Joint Computer Conference*, Dallas, Texas, October 25-29 1987.

[Rou91] N. Roussopoulos. "The Incremental Access Method of View Cache: Concept, Algorithms, and Cost Analysis". *ACM Transactions on Database Systems*, 16(3):535–563, September 1991.

[Sel87] T. Sellis. "Efficiently Supporting Procedures in Relational Database Systems". *Proc. ACM SIGMOD*, May 1987.

[Sel88] T. K. Sellis. "Intelligent Caching and Indexing Techniques for Relational Database Systems". *Inform. Systems*, 13(2), 1988.

[Sta89] Antonios G. Stamenas. "High Performance Incremental Relational Databases". Technical report, UMIACS-TR-89-49, CS-TR-2245, Department of Computer Science, University of Maryland, College Park, MD 20742, May 1989.

[Sto75] M. Stonebraker. "Implementation of Integrity Constraints and Views by Query Modification". In *Proceedings of the ACM SIGMOD Intl. Conf. on the Management of Data*, pages 65–78, San Jose, CA, 1975.

[Val87] Patrick Valduriez. "Join Indices". *ACM Transactions on Database Systems*, 12(2):218–246, June 1987.

[ZGMHW95] Yue Zhuge, Hector Garcia-Molina, Joachim Hammer, and Jennifer Widom. "View Maintenance in a Warehousing Environment". In Michael J. Carey and Donovan A. Schneider, editors, *Proc. of the ACM SIGMOD Conference*, pages 316–327, San Jose, California, May 1995.