[QW91] Xiaolei Qian and Gio Wiederhold. Incremental recomputation of active relational expressions. *IEEE Transactions on Knowledge and Data Engineering*, pages 337–341, 1991.

[Rou91] Nick Roussopoulos. The incremental access method of View Cache: Concept, algorithms, and cost analysis. *ACM Transactions on Database Systems*, 16(3):535–563, September 1991.

[RSU95] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In [POD95], pages 105–112.

[SF90] Arie Segev and Weiping Fang. Currency-based updates to distributed materialized views. In [DE90], pages 512–520.

[SI84] Oded Shmueli and A. Itai. Maintenance of Views. In *Proceedings of ACM SIGMOD 1984 International Conference on Management of Data*, pages 240–255, 1984.

[Sig86] Carlo Zaniolo, editor. *Proceedings of ACM SIGMOD 1986 International Conference on Management of Data*, Washington, D.C., May 28-30 1986.

[Sig95] Michael Carey and Donovan Schneider, editors. *Proceedings of ACM SIGMOD 1995 International Conference on Management of Data*, San Jose, CA, May 23-25 1995.

[SP89] Arie Segev and Jooseok Park. Updating distributed materialized views. *IEEE Transactions on Knowledge and Data Engineering*, 1(2):173–184, June 1989.

[SR88] Jaideep Srivastava and Doron Rotem. Analytical modeling of materialized view maintenance. In *Proceedings of the Seventh Symposium on Principles of Database Systems (PODS)*, pages 126–134, Austin, TX, March 21-23 1988.

[ZGHW95] Yue Zhuge, Hector Garcia-Molina, Joachim Hammer, and Jennifer Widom. View maintenance in a warehousing environment. In [Sig95].

[ZHKF95] G. Zhou, R. Hull, R. King, and J-C. Franchitti. Using object matching and materialization to integrate heterogeneous databases. In *Proc. of* 3^{rd} *International Conference on Cooperative Information Systems*, pages 4–18, 1995.

This article was processed using the LaTeX macro package with LLNCS style

[DE90] *Proceedings of the Sixth IEEE International Conference on Data Engineering*, Los Angeles, CA, February 5-9 1990.

[DEB95] Jennifer Widom, editor. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, 18(2), June 1995.

[DJLS95] Shaul Dar, H.V. Jagadish, Alon Y. Levy, and Divesh Srivastava. Answering SQL queries with aggregation using views. Technical Memorandum, AT&T Bell Laboratories, 1995.

[GHQ95] Ashish Gupta, Venkatesh Harinarayan, and Dallan Quass. Generalized projections: A powerful approach to aggregation. In Umeshwar Dayal, Peter M.D. Gray, and Shojiro Nishio, editors, *Proceedings of the 21st International Conference on Very Large Databases*, Zurich, Switzerland, September 11-15 1995.

[GJM94] Ashish Gupta, H. V. Jagadish, and Inderpal Singh Mumick. Data integration using self-maintainable views. Technical Memorandum, AT&T Bell Laboratories, November 1994.

[GL95] Timothy Griffin and Leonid Libkin. Incremental maintenance of views with duplicates. In [Sig95].

[GM95] Ashish Gupta and Inderpal Singh Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. In [DEB95], pages 3–19.

[GMR95] Ashish Gupta, Inderpal Singh Mumick, and Kenneth A. Ross. Adapting materialized views after redefinitions. In [Sig95].

[GMS93] Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *Proceedings of ACM SIGMOD 1993 International Conference on Management of Data*, Washington, DC, May 26-28 1993.

[Han87] Eric N. Hanson. A performance analysis of view materialization strategies. In Umeshwar Dayal and Irv Traiger, editors, *Proceedings of ACM SIGMOD 1987 International Conference on Management of Data*, pages 440–453, San Francisco, CA, May 27-29 1987.

[HGW+95] Joachim Hammer, Hector Garcia-Molina, Jennifer Widom, Wilburt Labio, and Yue Zhuge. The Stanford Data Warehousing Project. In [DEB95], pages 41–48.

[JMS95] H. V. Jagadish, Inderpal Singh Mumick, and Avi Silberschatz. View maintenance issues in the chronicle data model. In [POD95], pages 113–124.

[LHM+86] Bruce Lindsay, Laura Haas, C. Mohan, Hamid Pirahesh, and Paul Wilms. A snapshot differential refresh algorithm. In [Sig86], pages 53–60.

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In [POD95], pages 95–104.

[NY83] J. M. Nicolas and K. Yazdanian. An Outline of BDGEN: A Deductive DBMS. In *Information Processing*, pages 705–717, 1983.

[NYT94] New York Times. Bug in chemical bank's ATM software, February 18 1994. Front page article.

[Pai84] R. Paige. Applications of finite differencing to database integrity control and query/transaction optimization. In H. Gallaire, J. Minker, and J. Nicolas, editors, *Advances in Database Theory*, pages 170–209, New York, 1984. Plenum Press.

[POD95] *Proceedings of the Fourteenth Symposium on Principles of Database Systems (PODS)*, San Jose, CA, May 22-24 1995.

- Given a set of queries over the transactional data, what aggregate views must be materialized, so that the queries can be answered even after some of the transactional data has been discarded?
- A user of the database system may not know about all the views being materialized by the system. How can a query over base data be automatically optimized to use the materialized views instead [RSU95, LMSS95, CKPS95, DJLS95, GHQ95]. For instance, consider a query in a retailing application that wants to compute the number of items sold for each item. A query optimizer can optimize this query to access a materialized view that stores the number of items sold for each item and store, and avoid access to a much larger sales-transactions table.

Implementations: The applications are forcing a closer look at implementation issues surrounding materialized views and view maintenance. When are materialized views maintained – before the transaction that updates the base relation commits (immediate maintenance), or after the transaction commits (deferred maintenance [LHM+86, SP89, SF90])? Is view maintenance part of the transaction or not? What transaction consistency guarantees are possible when the view maintenance is deferred? Several implementation efforts are under way. There are two efforts at AT&T Bell Laboratories, one investigating how to implement materialized views inside an object-oriented database, while the second tries to implement materialized views on top of an existing relational database. A project at Columbia University is implementing view adaptation [GMR95] on top of Sybase. The WHIPS project at Stanford is attempting to integrate data from multiple sources into a data warehouse using materialized views [HGW+95].

Acknowledgements

I thank Dallan Quass and Timothy Griffin for comments on a draft of this paper, and Ashish Gupta for exploring several of these topics with me.

References

[BC79] Peter O. Buneman and Eric K. Clemons. Efficiently monitoring relational databases. *ACM Transactions on Database Systems*, 4(3):368–382, September 1979.

[BCL89] J. A. Blakeley, N. Coburn, and P. Larson. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Transactions on Database Systems*, 14(3):369–400, September 1989.

[BLT86] J. A. Blakeley, P. Larson, and F. W. Tompa. Efficiently Updating Materialized Views. In [Sig86], pages 61–71.

[BM90] Jose A. Blakeley and Nancy L. Martin. Join index, materialized view, and hybrid hash join: A performance analysis. In [DE90], pages 256–263.

[CKPS95] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. Optimizing queries with materialized views. In *Proceedings of the Eleventh IEEE International Conference on Data Engineering*, Taipei, Taiwan, March 6-10 1995.

Data Warehousing:

- What should the view language support to enable an integration of data from different sources? Outer-joins [GJM94] and general matching [ZHKF95] have been proposed.
- While the materialized views are available for view maintenance in a data warehouse, access to the remote databases may be restricted or expensive. It is thus beneficial if the view maintenance can proceed without accessing the remote databases. A view that can be so maintained is called self-maintainable in [GJM94] and autonomously computable in [BCL89]. When is a view self-maintainable? What is the impact of foreign keys, functional dependencies, and aggregation on self-maintainability?
- The remote databases need to communicate their updates to the warehouse. Depending on the capabilities of the remote databases, they may send the actual changes, the SQL insert/delete/update statements, or database snapshots. Can data replication facilities in database products be used to send a log of changes? How does one run view maintenance in each scenario?
- When a view is not self-maintainable, what are the other views that can be materialized at the warehouse so that the collection of views can be *self-maintained*?
- When a view is not self-maintainable, and a remote database needs to be accessed to maintain it, how do we guarantee consistency of the materialized view? [ZGHW95]
- A materialized view in a warehouse may be modified directly at the warehouse, either to make annotations or to clean up erroneous data. Thus, the materialization is no longer derived exactly according to the definition of the view. How does one maintain such views without losing the warehouse-specific modifications?

Banking and Retailing:

- How do you deal with aggregation during view maintenance? [GMS93, GL95]
- In a batch environment, where updates to the base relation and hence view maintenance are done in batch, a desirable goal is to minimize the amount of time for which the view is unavailable for queries? It is thus important that the view be write locked by the view maintenance process for the minimum possible time. Can we implement view maintenance algorithms such that they compute the update to the materialized view separately from the materialization, without locking the data, and then quickly apply the update to the materialization, holding the lock for the short duration needed to apply the update.
- In a transactional environment, the view must be maintained after each transaction, so the view maintenance algorithm must be very efficient. What is the class of views that can be maintained in a high throughput transaction system? [JMS95]

transmitted to the warehouse. Incremental view maintenance techniques can be used to maintain the materialized views in response to these modifications.

Banking and Retailing: Banking, retailing, and billing systems deal with a continuous stream of transactional data. One characteristic of such data is that it can get very large, and it can be beyond the capacity of any database system to even store, far less access, all of this data for answering queries. Consequently, the data is stored for only a specific period of time (this can vary from a few weeks to a few months), and even then it typically forms the largest data set in the system.

Decision support applications typically involve aggregate queries over the data. For example, one may be interested in total cash withdrawals from each branch of a bank on each of the last 30 days. Transactional applications also may need to access aggregate data. For example, a withdrawal transaction would like to know the balance in the customer's account, and the balance may be obtained by adding up the credits and debits from *all* the past banking transactions in the account.

Applications that need a fast response time and high throughput on queries with aggregation build summary tables by aggregating the current transactional data sets along several dimensions, and store the summary tables as if they were base data. By asking queries against the summary tables whenever possible, the response time and throughput for queries is improved. Similarly, summaries that encapsulate the interesting features of old transactional data (such as the account balance) are defined, and stored as base data. One of the problems with this approach is that the applications also have the burden of maintaining the summary tables after every update to the transactional data.

Materialized Views can be used to move the functionality of summary tables into the database system, and out of the application code. The maintenance of summary tables is then automated. Consequently, the maintenance process can be more efficient, as well as error-free (an error in the maintenance code was responsible for Chemical bank ATM withdrawals causing incorrect updates to customer balances on February 18, 1994, leading to several bounced checks and frustrated customers [NYT94]). More summary tables can be defined, and each summary table can be defined quickly. Further, since the system knows that the summary tables are really views, even queries that reference the base transactional data can be optimized to reference the summary tables instead.

3 Problems

We give a sampling of new view materialization/maintenance problems due to these application domains. Citations to the recent work addressing some of the problems are included.

the base relations are called *incremental view maintenance* algorithms. A classification and survey of several view maintenance algorithms appears in [GM95].

Materialized views were investigated in the 1980's as a tool to speed up queries on views. Several view maintenance algorithms were proposed [NY83, Pai84, SI84, BLT86, LHM+86, BCL89, SP89, QW91], analytical models for the cost of view maintenance were developed [Han87, SR88, SF90], and the impact of materialized views on the performance of queries was studied [Han87, BM90]. The use of view maintenance algorithms in maintaining integrity constraints was also recognized [BC79]. However, implementations of materialized views in research prototypes and commercial systems were missing (with the sole exception of ADMS [Rou91]). The benefit of materialized views to applications was never proven, and no killer application came up to force the implementation of materialized views.

2 Applications for Materialized Views

Over the last couple of years, there has been a renewed interest in materialized views in both the research and industrial communities. Several factors are contributing to this:

- New and novel applications for materialized views and view maintenance techniques are emerging. These include data warehousing, mobile systems, data visualization, data replication, and distributed CD-ROM services. More applications remain to be discovered !
- A new look at some traditional applications such as banking, retailing, and billing shows that these can be managed more efficiently, and with fewer errors, using materialized views.
- There is an interest from the database vendors to implement materialized views, and a desire by large database customers for functionality that can be obtained through materialized views.
- Disk prices have come down dramatically, so the disk space needed to store materialized views is cheaper.

We discuss a couple of these applications, and some of the problems the applications present.

Data Warehousing: A database system that collects and stores data from several databases is often described as a data warehouse. The warehouse usually integrates data from multiple sources, and provides a different way of looking at the data than the databases being integrated, so that copying each database into the warehouse is wasteful.

Materialized views provide a framework within which to collect information into the warehouse from several databases without copying each database in the warehouse. Queries on the warehouse can then be answered using the materialized views without accessing the remote databases. Provisioning (modification of the data) still occurs on the remote databases. The modifications are then

The Rejuvenation of Materialized Views

(Extended Abstract)

Inderpal Singh Mumick

AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974, USA.
mumick@research.att.com

Abstract. This is a short summary of a talk presented at the sixth International *Conference on Information Systems and Management of Data* (CISMOD 95) held in Bombay in November 1995.
The summary describes some of the applications that are causing a renewed interest in materialized views, the problems in supporting the applications, and sketches how the current work in this area is addressing some of the problems.

1 Introduction

What is a view? A view is a derived relation defined in terms of base (stored) relations. A view thus defines a function from a set of base tables to a derived table; this function is typically recomputed every time the view is referenced.

What is a materialized view? A view can be materialized by storing the tuples of the view in the database. Index structures can be built on the materialized view. Consequently, database accesses to the materialized view can be much faster than recomputing the view. A materialized view is thus like a cache – a copy of the data that can be accessed quickly.

Why use materialized views? Like a cache, a materialized view provides fast access to data; the speed difference may be critical in applications where the query rate is high and the views are complex so that it is not feasible to recompute the view for every query.

What is view maintenance? Just as a cache gets *dirty* when the data from which it is copied is updated, a materialized view gets dirty whenever the underlying base relations are modified. The process of updating a materialized view in response to changes to the underlying data is called view maintenance.

What is incremental view maintenance? In most cases it is wasteful to maintain a view by recomputing it from scratch. Often it is cheaper to use the heuristic of inertia (only a part of the view changes in response to changes in the base relations) and thus compute only the changes in the view to update its materialization. Algorithms that compute changes to a view in response to changes to