Aggregate Maintenance for Data Warehousing in Informix Red Brick® Vista™

Craig J. Bunger* Latha S. Colby* Richard L. Cole* William J. McKenna†
Gopal Mulagund‡ David G. Wilhite*

Informix Software Inc.
485 Alberto Way
Los Gatos, CA 95032

Abstract

Dramatic performance gains can be realized in decision support DBMSs by leveraging precomputed aggregates (i.e., materialized aggregate views). Informix Red Brick Vista provides a comprehensive solution for aggregate computation and management in data warehousing environments through the effective use of precomputed aggregates. Vista's transparent rewrite, advisor, and maintenance components allow users to maximize the benefits of precomputed aggregates while minimizing the costs associated with creating and maintaining them. In this paper, we provide an overview of Vista's maintenance subsystem. We describe optimizations used to make maintenance highly efficient, and discuss how Vista's focus on star schemas and data warehousing environments influenced the maintenance subsystem.

1 Introduction

Informix Red Brick Decision Server is a specialized relational database engine that has been designed and optimized to meet the requirements of large data warehouses. Vista is a comprehensive solution for aggregate computation and management that is integrated in Informix Red Brick Decision Server. Aggregate query processing can be computationally intensive in large data warehousing environments. A standard model of data in such environments is that of *facts* associated with points in a *dimension space*. Aggregates are typically computed on points in the dimension space, possibly with constraints placed on dimensional attributes. Even with a highly efficient query processing subsystem, aggregate queries on large data sets can often be very expensive to compute. Vista provides an effective solution for achieving exceptional query performance through the use of precomputed aggregates while minimizing the costs associated with creating and maintaining such aggregates. Vista's query rewrite component automatically rewrites queries on detail data to use precomputed aggregates thus allowing end users, query tools, and application programmers to derive the performance advantages of aggregates without having to change queries and applications. The Advisor component of Vista helps maximize the potential benefit of aggregates while minimizing the associated costs by performing an analysis of query logs to determine where aggregates can improve warehouse performance. The transparent aggregate maintenance feature of Vista, introduced in Informix Red Brick Decision Server 6.10, further reduces the costs associated with maintaining aggregates. Maintenance of aggregates is seamlessly integrated with operations that modify the underlying detail tables.

A precomputed aggregate in Vista is composed of a view defining an aggregate query over tables related via foreign-key/primary-key relationships, and a table (an aggregate table) containing the precomputed result of the query. Aggregate tables are physically no different than regular tables and all of Red Brick's indexing, segmentation and other physical design techniques available to regular tables are applicable to aggregate tables as well. For example, a good aggre-

*Email: {craigb,colby,rickcole,dwilhite}@informix.com; †Work performed while employed at Informix, currently at Oracle: bmckenna@oracle.com; ‡Work performed while employed at Informix, currently at Aztec Software & Technology Services Ltd: gopalm@aztec.soft.net

Proceedings of the 27th VLDB Conference, Roma, Italy, 2001

gate design strategy [5], involves creating referential relationships between an aggregated fact table and a dimension or an aggregated (derived) dimension table. For large data warehouses, the efficiency of query processing is an important quality even on aggregated data. In order to further improve the performance of queries that reference these aggregate tables (either directly or through Vista rewrites), a DBA might choose to create indexes on these tables. For example, a STARindex^{TM} could be created on the aggregated fact table. An aggregate query directed against these related aggregates can thus benefit from Red Brick's STARjoin^{TM} and other efficient query processing techniques.

Techniques for rewriting queries, selecting aggregates, and maintaining aggregates have been proposed in the literature (e.g., [4], [7], [8]) and implemented by DBMS vendors (e.g.[1], [2], [3], [6], [10]). In this paper, we highlight some of the features of Red Brick Vista's aggregate maintenance strategy that are designed to satisfy the unique requirements of data warehouses.

2 Maintenance Functionality Overview

Vista's aggregate maintenance subsystem, like the other components of Vista, is fully integrated into the Red Brick server and, more importantly for data warehousing applications, with Red Brick's bulk loader.

Aggregates can be maintained in immediate mode, i.e., within the same transaction that updates detail tables, using either incremental or rebuild (full recomputation) strategies. Maintenance is performed automatically when server-based DML operations (insert/delete/update) and time-cyclic data roll-off/on operations are performed on the detail table. Red Brick's high speed parallel loader, the Table Management Utility (TMU), communicates with the server (via shared memory) to maintain the appropriate aggregates when data is inserted or modified in a batch load. Maintenance is immediate in this case also because the loader and server run in the same transaction guaranteeing that the data is "query ready" at the end of bulk loads that are typical of data warehousing processes. Both incremental and rebuild maintenance strategies are utilized. Vista's incremental maintenance is designed to handle aggregates defined using any combination of aggregation functions (`sum`, `count`, `min`, `max`, `sum-distinct`, and `count-distinct`) and joins under all types of modifications (server-based or loader-based) to detail data. Aggregates may also be maintained in deferred mode via rebuild from detail tables or other finer-granularity aggregates.

A metadata component manages information pertinent to precomputed views such as their definition and information about the schema and constraints of the aggregates and underlying detail tables. The integration of Vista with the server (and the loader) ensures that the validity of aggregates, i.e., their status reflecting whether or not they are "in-synch" with detail tables, is correctly tracked by the metadata layer and communicated to the rewrite and maintenance components. An aggregate may become invalid if, for example, automatic maintenance is turned off during a load. Only valid aggregates are considered for incremental maintenance and for use in maintaining other coarser-granularity aggregates during immediate maintenance. Invalid aggregates can be rebuilt in deferred mode and, as in the case of immediate maintenance, the maintenance subsystem considers finer-granularity aggregates from which to rebuild related coarser-granularity ones. Foreign-key/primary-key relationships and dimensional hierarchies are used in determining an optimal maintenance strategy as explained in the next section.

Aggregates can also be maintained when versioning (Red Brick's data warehouse specific concurrency mechanism [9]) is enabled. Versioning in Red Brick is designed to handle bulk loads concurrently with queries. Unlike traditional OLTP systems, loads in data warehousing environments involve large amounts of data and require maintaining integrity constraints across multiple tables, building indexes and maintaining aggregate tables. Therefore, versioning at the row or other finer-granularity level is not very useful when queries and loads involve large sections of data. Versioning in Red Brick is at a level that allows queries to see consistent snapshots (older revisions) of tables, indexes, and aggregates while detail table, index and aggregate maintenance operations are being performed. With versioning, the TMU's bulk-loads can be applied in periodic-commit mode where the load can be specified to commit in certain (time or row) intervals. At each commit interval the changes to both the detail table and aggregates are committed.

3 Maintenance Techniques

Vista's star-schema focused maintenance algorithms coupled with the server's core execution engine help achieve superior performance during maintenance operations. When a detail table is modified, the subsystem determines which aggregates need maintenance and generates a plan that executes the detail table modifications and maintains the aggregate tables in a single transaction.

One of the distinguishing features of Vista's maintenance system is dynamic plan selection. Dynamic plan selection allows accurate selection of maintenance strategies based on the actual intermediate results of maintenance processes. The maintenance subplan for each aggregate table includes, in general, a plan based on incremental maintenance and a plan based on rebuilding the aggregate. The rebuild plan might be based on rebuilding the aggregate from either the detail tables or from other finer-granularity aggregates. Parent-child relationships between coarser and finer-

granularity aggregates are derived from functional dependencies. Both implicit functional-dependencies and those based on user-defined hierarchies are considered in determining parent-child relationships between aggregates. An implicit functional dependency is one that can be inferred based on implicit schema properties such as foreign-key/primary-key relationships. User-defined hierarchies are metadata that the user provides expressing functional dependencies among columns. The system considers functional dependencies that can be inferred (transitively) from the set of implicit and user-defined hierarchies while constructing a parent-child dependency graph.

The overall maintenance plan is thus based on an intelligent ordering of plans for maintaining all the aggregates and alternative plans for maintaining each aggregate. The actual choice of a plan for a particular aggregate is made at execution-time based on run-time statistics about delta-rows, detail tables and any parent aggregate tables. The choice of incremental vs rebuild plan is also influenced by the type (view definition) of the aggregate and the type of modifications to the detail table. For example, an incremental maintenance plan for a view with only `count(*)` aggregation functions would not require any groups to be recomputed from detail data. Hence, even if the detail data modifications result in modifications to a large number of rows in the aggregate table, incremental maintenance might be preferable to rebuild. Conversely, an aggregate table containing `max` may require some groups to be recomputed if the changes to the underlying tables are not pure inserts. Thus, rebuilding might be preferable under the same sets of modifications (as compared to the previous aggregate). The presence of a finer-granularity parent aggregate may also influence whether or not an aggregate is maintained incrementally. If the size of the parent aggregate is significantly smaller than that of the detail tables then the aggregate may be maintained by rebuilding from its parent under the same set of modifications for which incremental maintenance might have been a better choice if the parent aggregate were not present. The maintenance subsystem automatically decides on the appropriate maintenance strategy based on these factors.

Figure 1 shows an example of a high-level plan for maintaining two aggregates `Sales-by-qtr` and `Sales-by-year` when the detail fact table is modified. In this example, the aggregates are assumed to be computing aggregations on the `Sales` fact table grouped by the time dimension columns `qtr` and `year`, respectively. Additionally, we assume that a user-defined hierarchy (functional dependency) exists from `qtr` to `year`. The delta rows would have already been produced by either the server or the TMU prior to running the maintenance plan (but in the same transaction as the maintenance plan).

The root of the plan has a Sequence operator (Node 1). A Sequence operator runs its inputs to completion in order (left to right in the plan shown). The input plan rooted at Node 2 represents the maintenance plan for the `Sales-by-qtr` aggregate while the one rooted at Node 8 represents the maintenance plan for the `Sales-by-year` aggregate. Node 2 is a Choice operator. A Choice operator runs its *preliminary* inputs (in this case, a single input marked P0) before making a decision about which *choice* inputs (marked C0 and C1) to run. Only 1 choice input is executed. Therefore, the Choice operator can be used to make run-time optimizations. In this plan, statistics about the delta rows and the `Sales` table are gathered (Node 3) and examined at run-time to determine whether to execute the incremental plan rooted at Node 4 or the rebuild plan rooted at Node 5. The plan rooted at Node 8, the maintenance plan for `Sales-by-year`, is similar to the maintenance plan for `Sales-by-qtr`. In the rebuild plan rooted at Node 11 the `Sales-by-year` aggregate table is recomputed from the previously maintained `Sales-by-qtr` aggregate table utilizing the hierarchy from `qtr` to `year`.

Integration with the server's core execution engine also allows the maintenance subsystem to leverage parallelism and various high performance operators and optimizations available in the server, e.g., Red Brick's unique STARjoin algorithm. The incremental maintenance algorithm is designed to minimize detail-table accesses and repeated computations of sub-expressions. For example, the algorithm is based on a data-flow model that intelligently partitions incremental changes into groups of rows based on whether they can be be inserted, deleted, or updated directly in the aggregate table, or require new aggregation values to be computed from detail data. Techniques for detecting common subplans and generating star-schema based access methods further improve the efficiency of any detail table access necessary for computing new aggregation values. For example, recomputing a particular group for incremental maintenance might involve identifying all the fact table rows referencing dimension rows corresponding to the values in the group. The maintenance algorithm could use a STARindex on the detail tables, if an appropriate one existed, to efficiently identify the fact table rows in the group. The recompute part of an incremental plan might itself contain different execution choices depending on the presence of STAR or other indexes such as TARGET^{TM} indexes. Subplans of the maintenance plan that are common to alternative choice plans are coalesced.

The coupling of the TMU's load process with the server enables all of the same maintenance techniques and optimizations used during server-based DML to be leveraged during bulk-loads. A single bulk-load could involve both appends and modifications, i.e., a mix of inserts and updates, to a table. The TMU first performs the detail-table load including all referential-

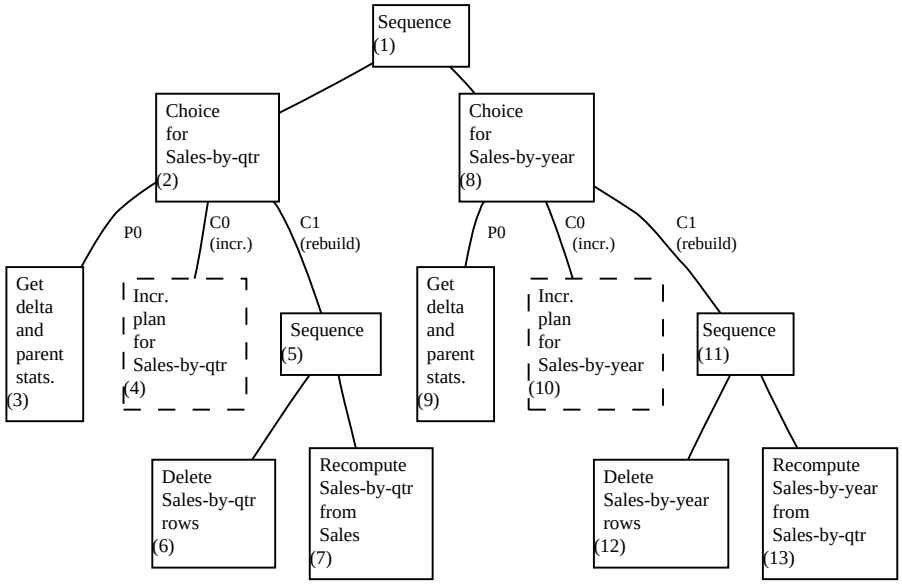

Figure 1: An example maintenance plan

integrity checking and index maintenance before invoking aggregate maintenance (within the same transaction) through the server. The detail table load could involve data that has not been cleaned, i.e., might contain data with duplicates or might contain data representing a series of changes to a given row. For example, a load could insert a row and subsequently modify that same row. Therefore, the set of changes made by a load represented as sets of inserts and deletes may not be minimal (e.g., the sets may not have an empty intersection). However, the incremental maintenance algorithm can handle changes that are not necessarily minimal, for aggregates containing any combination of the previously mentioned aggregation functions, without requiring any pre-processing or compressing of deltas.

4 Conclusions

In keeping with Red Brick's tradition of providing key technologies that are driven by the unique requirements of data warehousing, the features of Vista's maintenance subsystem both support and leverage the properties of data warehouse focused star schemas and dimensional hierarchies. Integration of the maintenance subsystem with the server's core execution engine and the TMU help achieve superior performance during maintenance operations and ensure that detail and aggregate data are query-ready at the end of warehouse-style bulk loads.

5 Acknowledgements

The authors are grateful to all the members of the Red Brick development team at Informix who contributed to the work described in this paper.

References

[1] S. Agrawal, S. Chaudhuri, and V. Narasayya. Automated selection of materialized views and indexes for SQL databases. In *Proceedings of the 26th VLDB Conference*, 2000.

[2] R. G. Bello, K. Dias, A. Downing, J. Feenan, J. Finnerty, W. D. Norcott, H. Sun, A. Witkowski, and M. Ziauddin. Materialized views in Oracle. In *Proceedings of 24th VLDB Conference*, 1998.

[3] L. S. Colby, R. L. Cole, E. Haslam, N. Jazayeri, G. Johnson, W. J. McKenna, L. Schumacher, and D. Wilhite. Red Brick Vista: Aggregate computation and management. In *Proceedings of the 14th ICDE Conference*, 1998.

[4] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proceedings of ACM SIGMOD*, 1996.

[5] R. Kimball. Aggregate navigation with (almost) no metadata. *DBMS*, August 1996.

[6] W. Lehner, R. Sidle, H. Pirahesh, and R. Cochrane. Maintenance of cube automatic summary tables. In *Proceedings of ACM SIGMOD*, 2000.

[7] I. S. Mumick, D. Quass, and B. S. Mumick. Maintenance of data cubes and summary tables in a warehouse. In A. Gupta and I. S. Mumick, editors, *Materialized Views: Techniques, Implementations, and Applications*, chapter 25. The MIT Press, 1999.

[8] D. Srivastava, S. Dar, H. V. Jagadish, and A. Y. Levy. Answering queries with aggregation using views. In *Proceedings of the 22nd VLDB Conference*, 1996.

[9] R. Taylor. Concurrency in the data warehouse. In *Proceedings of the 26th VLDB Conference*, 2000.

[10] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh, and M. Urata. Answering complex SQL queries using automatic summary tables. In *Proceedings of ACM SIGMOD*, 2000.