Red Brick Vista™: Aggregate Computation and Management

Latha S. Colby Richard L. Cole Edward Haslam Nasi Jazayeri
Galt Johnson William J. McKenna Lee Schumacher David Wilhite

Red Brick Systems, Inc.
485 Alberto Way
Los Gatos, CA 95032

{colby,rickcole,haslam,nasi,galt,bmckenna,lees,dwilhite}@redbrick.com

Abstract

Aggregate query processing in large data warehouses is computationally intensive. Precomputation is an approach that can be used to speed up aggregate queries. However, in order to make precomputation a truly viable solution to the aggregate query processing problem, it is important to identify the best set of aggregates to precompute and to use these precomputed aggregates effectively. The Red Brick® aggregate computation and management system (Red Brick Vista) provides a complete server-integrated solution to these problems.

1 Introduction

Aggregation computation is an important component of decision support applications. Aggregates are typically computed on points in dimensions, e.g., sum of Sales grouped by some combination of points in the dimensions Time, Product, and Location. In relational databases, the queries that compute these aggregates are typically resource intensive. A natural solution is precomputation, i.e., materializing views, which can give dramatic performance gains. However, to be truly effective, this approach must provide additional DBMS functionality for both database administrators (DBAs) and users.

Specifically, there must be (1) an intelligent means for DBAs to determine a relatively small set of precomputed aggregates to create, since precomputing and maintaining aggregates for all combinations of points in all dimensions is impractical in a large data warehouse, and (2) a means for rewriting user queries to use this small set of precomputed aggregates without requiring the users to manually rewrite queries or to change their queries when aggregates are dropped or added.

Red Brick Vista is an aggregate processing and management system that provides a complete solution addressing these needs, and, to our knowledge, is the first system supporting precomputed aggregates (materialized aggregate views) that is fully integrated into a relational database server. This system has three primary components:

- The Advisor: This subsystem suggests what aggregates to precompute, and determines the effectiveness of existing aggregates based on an analysis of query histories.
- The Metadata Layer: This component stores information about precomputed aggregate definitions and dimensional hierarchies, and tracks the state of each precomputed aggregate in relation to its base tables. The dimensional hierarchies allow the Vista system to deliver multidimensional database (MDDB) functionality, such as rollups from aggregates on points in a dimension to other points of coarser granularity in the same dimension.
- The Transparent Query Rewriter: This component is an "Aggregate Navigator", to use Kimball's [4] terminology, which transforms queries on base tables to ones involving precomputed aggregates. The rewrites are performed transparently, thus insulating users from details of aggregate navigation. It uses a novel cost-based algorithm to rewrite a far wider spectrum of queries than is possible using existing commercial or research algorithms.

Unlike other products, the Red Brick Vista system is tightly integrated in the Red Brick Warehouse server (rather than implemented as a middleware product).

This greatly increases the scope for performance optimizations and the ability to share metadata with the server.

In this paper, we give an overview of aggregate processing in decision support environments. We then discuss the three main components of Vista, and show how they are integrated in Red Brick's relational database server to deliver an effective solution to the aggregate computation problem. We also point out the unique aspects of the Red Brick solution that address issues not solved by existing products, nor addressed by current database research.

2 Aggregate Computation in Decision Support Environments

In decision support environments, a standard model of data is that of *facts* associated with points in a *dimension space*. In a retailing environment, for example, each sale occurs at a particular `time`, in a particular `store`, and is of a particular `product`. In this example, each `sales` event is a fact and occurs at a point in the 3-dimensional space (`product`, `store`, `time`). Each dimension usually forms a hierarchy: `product` may be a 2-level hierarchy, for example, with `product-type` at the finest level of granularity and `product-category` at the coarsest level. Multidimensional data models distinguish between points in dimensions (e.g., `product-type`) and attributes of these points (e.g., `product-color`). Aggregates are typically computed on points in the dimension space, possibly with constraints placed on dimensional attributes. The `CUBE` operator is defined in [1] as a relational operator computing a table containing the set of all aggregates grouped on all combinations of a set of columns specified by a user query.

Aggregate processing in relational databases typically involves retrieving qualifying fact records based on dimension constraints, grouping the records by values for points in specified dimensions, and applying aggregate functions to each group. Even with a highly efficient query processing subsystem, aggregate queries accessing billions of fact and associated dimension records will often be very expensive to compute.

Precomputation can give dramatic performance gains in aggregate processing, and aggregate results at one granularity can often be used to compute aggregates at coarser granularities, eliminating the need to precompute all possible aggregates. This reuse is commonly referred to as *rollup*. In Gray et al. [1], `ROLLUP` is a SQL extension that allows users to limit the number of aggregates computed along points in a dimension.

Consider the following example schema consisting of a fact table `sales` and three (denormalized) dimension tables `period`, `product` and `store`.

```
sales(perkey, prodkey, storekey, dollars)

period(perkey, day, month, year)

product(prodkey, pname, type,
        category, category-desc)

store(storekey, sname, city)
```

The set of underlined columns for each table represents the primary key for that table. Now suppose that we have the following precomputed aggregate containing total sales by `type` and `day`:

```
select product.type, period.day,
       sum(dollars)
from sales, product, period
where sales.perkey = period.perkey
  and sales.prodkey = product.prodkey
group by product.type, period.day;
```

An example of a simple rollup would be in using the above precomputed aggregate to answer a query requesting the total sales by `product.type`. This first type of rollup has been discussed in database research literature [2, 5] and is supported by the Red Brick Vista system as well as some other aggregate processing systems implemented on relational DBMSs.

A more advanced type of rollup is one that uses information about hierarchies to answer queries aggregated on a column that is not present in the precomputed aggregate but is related to a column in the aggregate via a hierarchical (many-to-one) relationship. An example of this sophisticated type of rollup would be one where the above precomputed aggregate is used to answer a query requesting the total sales by `product.category` and `period.day`, assuming that there is a many-to-one relationship between `type` and `category`. In other words, `type` and `category` are columns representing points in the `product` dimension where a single `category` contains multiple `types`.

As we will show, Vista effectively supports this second type of rollup by maintaining metadata about dimensional hierarchies, and uses this metadata to apply novel, cost-based rewrites to aggregate queries. By supporting this type of rollup, Vista helps avoid having to create and manage all possible aggregates, and bridges the gap between relational DBMS aggregate processing and MDDB aggregate processing.

3 Components of Red Brick Vista

This section gives an overview of each Vista component, and describes interactions between components.

3.1 The Advisor

The Vista Advisor features a logging mechanism that examines each user aggregate query submitted. If no precomputed aggregates exist, the logging mechanism will record which, if any, "candidate" precomputed aggregates would be useful to answer the query. The generated candidate is typically not an exact match of the query, but rather one which can be used to efficiently answer (via rollups) a broad range of aggregate queries involving the same dimensions as the user query (or some subset of these dimensions). In this way, the Vista Advisor helps minimize the number of aggregates that need to be stored. In other words, based on actual query histories, the Advisor allows intelligent materialization (precomputation) of a subset of all possible aggregates (i.e., aggregates grouped on all combinations of dimensions and points along dimensions). To our knowledge, this method of candidate generation is unique to the Vista Advisor.

If precomputed aggregates exist, and the Transparent Query Rewriter transforms the query to use an existing aggregate, the logging mechanism records information about the usage of the aggregate. If the logging subsystem can suggest a more effective (nonexistent) aggregate as a candidate for precomputation, this candidate aggregate will also be logged.

An important feature of the Advisor is its relational SQL-based interface, which is made possible by having the Advisor integrated with the Red Brick Warehouse server. A DBA can use SQL commands to access Advisor-specific tables and obtain information on the utilization of existing aggregates, and on the candidate aggregates that should be created. The Advisor uses an algorithm partly based on the work described in [3]. Some of the significant extensions to the algorithm include new techniques to limit the set of aggregates that need to be analyzed to a small subset of the set of all possible aggregates. In addition, while analyzing existing and potential candidate aggregates, the advisor uses logged reference counts and the rewritability of one aggregate by another (using the Transparent Query Rewriter).

3.2 The Metadata Layer

The Metadata Layer stores information about precomputed aggregate definitions and about how logical dimensional hierarchies are represented in the database. A precomputed aggregate is comprised of a precomputed aggregate view definition and a table containing the precomputed data. The view definition establishes the semantic link between base (detail) tables and the table containing the precomputed results. The precomputed aggregate table could be populated either when the precomputed view is defined or could be an already populated table. This allows a user's existing precomputed aggregate tables to be seamlessly integrated into the Red Brick Vista system.

A dimensional hierarchy is specified using a `create hierarchy` statement. For example, consider the following hierarchy definition:

```
create hierarchy type-to-category
  (from product(type) to product(category));
```

This (Red Brick) SQL command creates metadata representing the fact that `type` and `category` columns are points in the `product` dimension, and that aggregates (such as `Sum(dollars)`) grouped on the `type` column can be used to rollup to (compatible) aggregates on the `category` column. Another way to interpret the `create hierarchy` statement is as a declaration of a functional dependency between two columns.

When denormalized (non-3NF) dimension tables exist (for performance reasons), the `create hierarchy` statement references columns from a single table. Hierarchies can also be specified between columns of different tables. Suppose that our example schema had normalized dimension tables. The `product` dimension may be represented in two tables as follows:

```
product(prodkey, pname, type, category-key)
class(category-key, category, category-desc)
```

In this case, the hierarchy between `type` and `category` would be defined as follows:

```
create hierarchy type-to-category
  (from product(type) to class(category));
```

Since these columns are in different tables, a join is required to perform this rollup, and therefore the Vista Metadata Layer requires a foreign key-primary key join constraint to exist between the `product` and `class` tables. The existence of these join constraints allows the Vista Rewriter to perform the second class of rollups while avoiding "double counting".

The Metadata Layer also infers implicit hierarchies (functional dependencies), such as those between a primary key column and another column of the same table, and dependencies implied by transitiv-

ity. These explicit and implicit hierarchies allow the Query Rewriter to answer a large class of queries using a small set of precomputed aggregates.

3.3 The Transparent Query Rewriter

This Red Brick Vista component intercepts aggregate queries and attempts to rewrite each query to use precomputed aggregates. This rewriting is done on a block-by-block basis, and includes correlated subqueries as well as blocks in union/intersect/except queries. The Rewriter uses a cost-based algorithm to choose among potential rewrites. The Rewriter is able to perform both types of rollups, but its novel use of metadata to perform the second class of rollups is what distinguishes it from existing products and research. Since the Rewriter is able to rollup efficiently from finer granularity aggregates to coarser granularity aggregates in the same dimension, the total number of precomputed aggregates can be minimized. These rollups can be performed when aggregates are grouped on both non-key and key columns corresponding to points in a dimension. Also, since rewrites are transparent, user queries do not have to be changed if aggregates are dropped from, or added to, the database.

4 Summary

The Red Brick Vista system alleviates many of the common problems surrounding the creation and use of precomputed aggregates to improve query performance in relational database systems. The Advisor helps the DBA with creating and evaluating the optimal set of precomputed aggregates to satisfy a system's unique performance and space requirements. End-users and applications can continue to query the database as they always have and the transparent query rewriter transforms the queries to utilize the existing aggregates. As in the case of indexes, the DBA has the freedom to tune the database's aggregate performance without affecting the way queries are submitted. Finally, all aggregate-related metadata is integrated into the database system's catalog including the new OLAP-like intra-dimensional hierarchy relationships. Red Brick Vista is currently being Beta tested and will be generally available in Red Brick Warehouse 5.1.

References

[1] Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F., and Pirahesh, H. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery 1* (1997), 29–53.

[2] Gupta, A., Harinarayan, V., and Quass, D. Aggregate-query processing in data warehousing environments. In *Proceedings of the 21st VLDB Conference* (Zurich, Switzerland, 1995).

[3] Harinarayan, V., Rajaraman, A., and Ullman, J. D. Implementing data cubes efficiently. In *Proceedings ACM SIGMOD International Conference on Management of Data* (Montreal, Canada, June 1996).

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

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