Optimizing Hierarchical Access in OLAP Environment

Lipyeow Lim [#1], Bishwaranjan Bhattacharjee [#2]

[#]*IBM T. J. Watson Research Center*
19 Skyline Dr., Hawthorne, NY 10532, U.S.A.
[1]liplim@us.ibm.com
[2]bhatta@us.ibm.com

***Abstract*— In Online Analytic Processing (OLAP) deployments, different users, lines of businesses and business units often create adhoc aggregation hierarchies tailor-made for specific reporting or analytical applications. As a result, a large number of these application specific hierarchies accumulate over time. System administrators typically are not able to optimize all these hierarchical accesses by hand due to the large number of hierarchies. However, many optimization opportunities exist due to the significant amount of overlap between some hierarhies. In this paper, we sketch a novel method for optimizing OLAP aggregation queries using precomputed aggregates on other overlapping hierarchies. Our method detects common sub-structures among hierarchies and provides a rewriting algorithm to exploit any precomputations on these shared sub-structures.**

I. Introduction

Data warehouses and on-line analytical processing (OLAP) have gained widespread use in almost all parts of an enterprise for decision support and business performance monitoring. A typical deployment uses the two-tier data warehousing approach [1]. The data repository tier consists of one or more DBMS (eg. IBM DB2 UDB) that extracts, transforms and cleans the data from multiple sources. The data access tier consists of one or more data marts (eg. Hyperion Essbase, Cognos etc) through which users access subsets of the data for subject-specific OLAP. An OLAP data mart consists of a fact table, which can be thought of as a materialized view of the data in the data repository, and several dimensions, each of which can be associated with multiple complex and unbalanced hierarchies. Different users in the enterprise typically define their own application-specific hierarchies within the same data mart. For example, within the same OLAP data mart for sales transactions, an accountant might define a hierarchy for aggregating sales transactions across all the business units in the enterprise, and a marketing executive might define another hierarchy to aggregate sales transactions by geographical regions. In many real use-cases, a set of enterprise-wide primary hierarchies is defined on each common fact table. Different users, lines of business, and business units then define their own application-specific hierarchies such that the leaf nodes of each application-specific hierarchy point back to elements in the primary hierarchies. Figure 1 illustrates an example of a primary hierarchy and two application specific hierarchies defined on it.

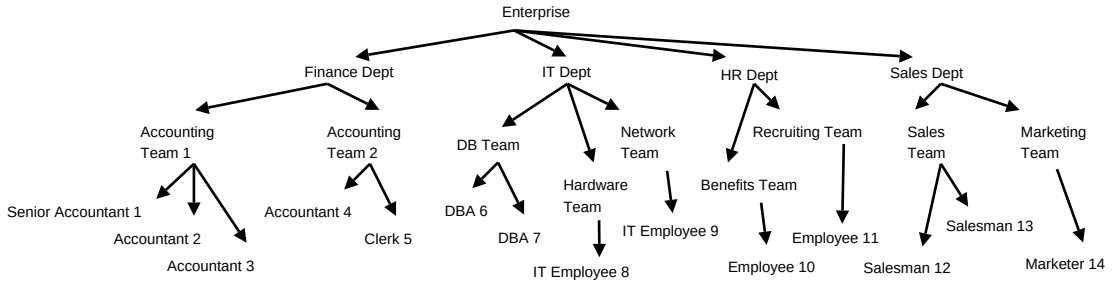
In the OLAP environment, user queries are defined against a small number of application-specific hierarchies. These queries are then translated into an equivalent query on the fact table automatically. Query optimization usually involves precomputing certain aggregates on some hierarchies.

Example 1 (Precomputing aggregates) *Consider a query that aggregates the expenses in the fact table (Figure 1(d)) according to the geographical hierarchy in Figure 1(b) and reports the expenses by continent (North America, Europe, and Asia). To speedup the query, the administrator can specify that expense aggregates for certain countries or certain continents be precomputed and stored in the OLAP environment, so that the query optimizer can exploit them.*

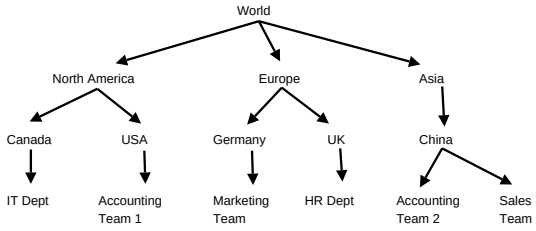
In practice, there can be hundreds of application-specific hierarchies. Precomputing aggregates for every internal node of every application-specific hierarchy is not feasible because of limited storage resources. Moreover, the OLAP administrator typically creates precomputed aggregates only for a few important hierarchies. Many hierarchies have no precomputed results associated with them. The question is: can we optimize access on the hierarchies on which no precomputed aggregations have been created ? Given that there are overlaps in the application-specific hierarchies, such optimization should be possible in principle. If precomputed aggregates have been created on a hierarchy H_1 and H_1 overlaps with hierarchy H_2, a query written on hierarchy H_2 should be able to exploit the precomputed aggregates on the overlapping portion of H_1 and H_2.

Example 2 (Exploiting overlapping hierarchies) *Consider the same query in Example 1 that reports the total expenses by continent (North America, Europe, and Asia) according to the geographical hierarchy in Figure 1(b). Further suppose no precomputed aggregates exists for the geographical hierarchy, but precomputed aggregates exists for the projects in the project hierarchy of Figure 1(c). The query should be able to exploit the precomputed aggregates for the* `Project 1` *node of the project hierarchy, because the aggregate is equivalent to the* `Asia` *or* `China` *node in the geographical hierarchy.*

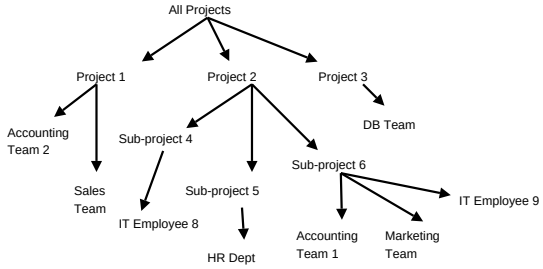
Unfortunately, current OLAP environments are not able to rewrite queries to exploit precomputed aggregates across different hierarchies, because the query optimizer does not know



(a) Primary hierarchy for the organization dimension.



(b) Application-specific geography hierarchy



(c) Application-specific project hierarchy

TransactionID	EmployeeID	Time	ExpenseType	Amount
10001	DBA 1	2007-05-01-2059	Harddisk	1000
10002	Accountant 2	2007-05-03-0759	Stationary	100
10003	DBA 1	2007-05-03-0859	Software	2000
...	...	...	...	...

(d) The expense transactions fact table.

Fig. 1. Primary organization hierarchy and hierarchies specific to expense accounts applications. The geographical hierarchy supports aggregations of expenses by geographical regions or the business units. The project hierarchy supports aggregations of expenses according to projects. Note that the leaf nodes of the application-specific hierarchies point to nodes in the primary hierarchy.

which hierarchies overlap with each other.

Our contributions. In this paper, we propose a method for optimizing aggregation queries on a set of hierarchies using precomputed aggregates from another set of hierarchies. Our method exploits the fact that most industrial hierarchies contain a significant amount of overlap and that these overlaps result in equivalences that can be used in query rewriting. Our method consists of two phases. In the off-line phase, we propose an algorithm to discover overlapping relationships in the hierarchies of the OLAP environment. These relationships are then stored in the catalog tables of the OLAP server. In the on-line phase, we propose a query rewrite algorithm that leverages the overlapping relationships discovered in the off-line phase to rewrite the OLAP queries.

II. Our Approach

Our approach to optimize aggregation queries on a set of hierarchies using precomputed aggregates from another set of hierarchies consists of two phases.

The off-line phase scans all the application-specific hierarchies in the OLAP environment in order to discover equivalences or overlaps between each pair of hierarchies. Naturally, only application-specific hierarchies of the same dimension can have equivalences, so comparisons of application hierarchies across different dimensions are not necessary. The result of the off-line phase is a list of node pairs for each pair of hierarchy. A node pair (u, v) for hierarchy pair (i, j) denotes that the node identifier (node ID) u from hierarchy i is equivalent to node ID v from hierarchy j in the sense that an aggregation based on the sub-tree rooted at u will yield the same result as an aggregation based on the sub-tree rooted at v. These equivalence pairs can be stored in a database catalog table for use by the optimizer.

The on-line phase rewrites a given query using the equivalence pairs from the off-line phase. An OLAP query is typically generated by a graphical user interface based on an application-specific hierarchy. The particular type of query we are concerned with are aggregations of a fact table column according to all the leaf labels of some sub-tree of the application-specific hierarchy. Note that queries consisting of more complex expressions can often be reduced to several of such simple aggregation sub-queries. Our rewriting algorithm proceeds as follows. The leaf nodes associated with the query are merged according to the associated hierarchy into sub-trees. A greedy set cover algorithm is used to find the set of sub-trees that cover the query. We then check the equivalence pairs from the off-line phase to find equivalent sub-trees in other hierarchies. Moreover, we only want equivalent sub-trees whose results have been materialized or cached in the OLAP environment. The output of the on-line phase consists of a

list of equivalent sub-trees for each sub-tree associated with the query. If each list contains more than one sub-tree, the optimizer can pick one using heuristics or cost-based metrics.

III. Related Work

In traditional OLAP environments where the dimension hierarchies are regular balanced trees, the precomputed aggregates are stored in a summary table or materialized view. Much research has been done on view selection and query rewriting to exploit views. Those previous work focus mainly on flat relational views [2], regular and balanced hierarchies [3], [4], [3], and cube views. The proposed techniques do no extend easily to the irregular hierarchies that are supported by the current state-of-the-art dedicated OLAP servers.

Many relational systems also support materialized views. Here, frequently used queries are precomputed and stored so as to improve query performance. When one has many such materialized views defined on the system, the optimizer has the task of selecting the appropriate materialized view to answer a query or a sub-query of the query [5], [6]. The on-line phase of our approach differs from previous rewriting algorithm mainly in that we are dealing with overlapping sub-trees as opposed to overlapping sub-queries.

The problem addressed by the off-line phase of our approach resembles the problem of finding common sub-trees addressed by [7]; however, our definition of "common sub-tree" is quite different Zaki's and his techniques do not extend easily to our problem.

IV. Conclusion

In this paper, we highlight an important and practical problem when data warehousing and OLAP are deployed in the industry: Multiple users create multiple ad hoc application-specific hierarchies and data warehousing systems fail to exploit query optimization opportunities across hierarchies. To the best of our knowledge, this problem has not been described in previous literature at all. To address this problem, we propose a solution that discovers and stores cross-hierarchy relationships in an OLAP environment, and that rewrite queries using the discovered relationships, so that precomputed aggregates can be better exploited. Our method addresses a limitation of current OLAP environments to exploit precomputed aggregates across different hierarchies.

References

[1] L. Gong, M. Olivas, C. Posluszny, D. Venditti, and G. McMillan, "Deliver an effective and flexible data warehouse solution, Part 2: Develop a warehouse data model," *IBM Developerworks*, July 2005, http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0507gong/.

[2] W. Xu, D. Theodoratos, and C. Zuzarte, "Computing closest common subexpressions for view selection problems," in *DOLAP '06: Proceedings of the 9th ACM international workshop on Data warehousing and OLAP*. New York, NY, USA: ACM Press, 2006, pp. 75–82.

[3] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson, "The treescape system: Reuse of pre-computed aggregates over irregular olap hierarchies," in *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K.-Y. Whang, Eds. Morgan Kaufmann, 2000, pp. 595–598.

[4] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson, "Extending practical pre-aggregation in on-line analytical processing," in *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, Eds. Morgan Kaufmann, 1999, pp. 663–674.

[5] C.-S. Park, M.-H. Kim, and Y.-J. Lee, "Rewriting olap queries using materialized views and dimension hierarchies in data warehouses." in *ICDE*, 2001, pp. 515–523.

[6] C.-S. Park, M.-H. Kim, and Y.-J. Lee, "Finding an efficient rewriting of olap queries using materialized views in data warehouses," *Decision Support Systems*, vol. 32, no. 4, pp. 379–399, 2002.

[7] M. J. Zaki, "Efficiently mining frequent trees in a forest," in *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM Press, 2002, pp. 71–80.