The TreeScape System: Reuse of Pre-Computed Aggregates over Irregular OLAP Hierarchies

Torben Bach Pedersen Christian S. Jensen

Department of Computer Science,
Aalborg University, DK–9220 Aalborg Ø, Denmark,
{tbp,csj}@cs.auc.dk

Curtis E. Dyreson

School of Information Technology,
Bond University, Gold Coast, QLD 4229,
Australia, cdyreson@bond.edu.au

Abstract

We present the TreeScape system that, unlike any other system known to the authors, enables the reuse of pre-computed aggregate query results for irregular dimension hierarchies, which occur frequently in practice. The system establishes a foundation for obtaining high query processing performance while pre-computing only limited aggregates. The paper shows how this reuse of aggregates is enabled through dimension transformations that occur transparently to the user.

1 Introduction

In order to improve query performance, modern On-Line Analytical Processing (OLAP) systems use a technique known as *practical pre-aggregation*, where *select* combinations of aggregate queries are materialized and re-used when computing other aggregates; full pre-aggregation, where all combinations of aggregates are materialized, is infeasible, as it typically causes a blowup in storage requirements of 200–500 times the size of the raw data [3, 5]. Normally, practical pre-aggregation requires the dimension hierarchies to be regular, i.e., to be balanced trees, but this is quite often not the case in real-world systems.

The TreeScape system presented here enables practical pre-aggregation even for irregular hierarchies, based on techniques described previously by the authors [4]. We show how to achieve practical pre-aggregation through transformations of the dimensions and how the transformations can be accomplished transparently to the user. The system enables the achievement of fast query response time while saving huge amounts of storage compared to current OLAP systems and techniques. The prototype implementation of TreeScape demonstrates that these benefits may be achieved with standard technology. While this demonstration uses a particular RDBMS, it's ODBC driver, and particular relational OLAP tool, TreeScape is not dependent on any specific suite of products[1], making the solution flexible and useful.

Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000.

2 Normalizing Hierarchies

We use a small case study concerning patients and their diagnoses for illustrating the workings of the system. Diagnoses have three different levels of precision, depending on how accurate a patient's condition can be described. The most precise diagnoses are *low-level diagnoses*, which are grouped into *diagnosis families*, which, in turn, are grouped into diagnosis groups. The example data consists of 9 diagnoses and their hierarchical relationship, along with patient counts. The data can be seen in Table 1 and to the left in Figure 1.

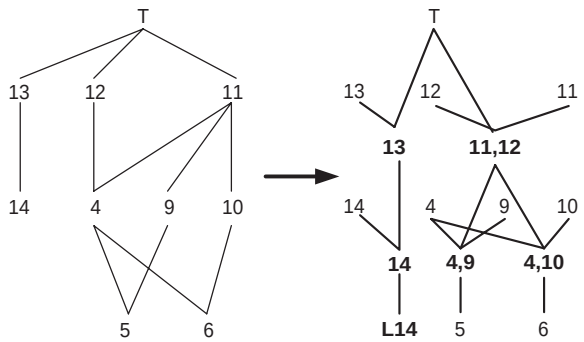

Figure 1: Dimension Transformations

The hierarchy is irregular. For example, it is unbalanced because the diagnosis "Lung cancer" (14) has no low-level diagnoses associated with it. The hierarchy is non-strict because, e.g., diagnosis 4 ("Diabetes during pregnancy")

[1] The solution assumes that an ODBC interface is available for the RDBMS, a requirement that is met for all commercial RDBMSs.

ID	Text	Type
4	Diabetes during pregnancy	Family
5	Insulin dependent diabetes during pregnancy	Low-Level
6	Non insulin dependent diabetes during pregnancy	Low-Level
9	Insulin dependent diabetes	Family
10	Non insulin dependent diabetes	Family
11	Diabetes	Group
12	Pregnancy related	Group
13	Cancer	Group
14	Lung cancer	Family

Diagnosis

ParentID	ChildID
4	5
4	6
9	5
10	6
11	9
11	10
12	4
13	14

Grouping

DiagID	Count
5	1

Patient

Table 1: Case Study Tables

has several parents. As a result, problems occur when pre-aggregated data at lower levels is used to compute new values at higher levels. For example, if we pre-aggregate the number of patients at the low-level diagnosis level and want to aggregate to the diagnosis family level, we cannot deduce what the value should be for "Lung Cancer" (14). If we pre-aggregate at the diagnosis family level, patients with diagnoses 5 or 6 will be counted for both of the diagnoses 4 and 9, and 4 and 10, respectively, leading to wrong results when we aggregate to the diagnosis group level.

A solution of the problems with pre-aggregation is to render the hierarchies well-behaved by *normalizing* them. Informally, the normalization process introduces new *placeholder* values where the hierarchy is unbalanced, and introduces *fused* values that represent *sets of* parent values when child values have multiple parents. The result of normalizing the hierarchy described above is seen to the right in Figure 1. For example, value "L14" representing "Lung Cancer" at the low-level diagnosis level, and value "4,9" representing the set of diagnoses $\{4, 9\}$ are introduced by the normalization. In the figure, all boldface values and links have been added by the normalization process. The normalization technique is described in detail elsewhere [4].

The normalized hierarchy supports practical pre-aggregation. For example, it is possible to store counts of patients at the low-level diagnosis level, and then re-use these to compute the counts for diagnosis families and diagnosis groups. With the example data (one patient with diagnosis 5), this will only require the storage of the one value versus six values being required for *full* pre-aggregation (one value for low-level diagnosis 5, two values for diagnosis families 4 and 9, two values for diagnosis groups 11 and 12, and one value for $\top$, which represents the total for all diagnoses).

The example is somewhat indicative of the storage savings achieved within a single dimension. When several dimensions are combined, the total space saved (with respect to full pre-aggregation) is the product of the savings in each dimension, resulting in savings factors of 100 or more in practice. The savings occur because of *multidimensional sparseness* [3, 5], the phenomenon of the multidimensional space being very sparse for the lower levels in the dimensions, while quickly becoming more dense at higher levels. The query response time using the normalization approach will not be quite as fast as using full pre-aggregation, but will most likely be comparable, i.e., within an order of magnitude. This is much faster than computing the results from the base data, as would be required with *no* pre-aggregation.

3 System Architecture

While the hierarchy transformations enable practical pre-aggregation, they also have the undesired side-effect of introducing new values into the hierarchies that are of little meaning to the users. Thus, the transformations should remain invisible to the users. This is achieved by working with two versions of each user-specified hierarchy and by using a query rewrite mechanism. This is described in detail in Section 4. The overall system architecture is seen in Figure 2.

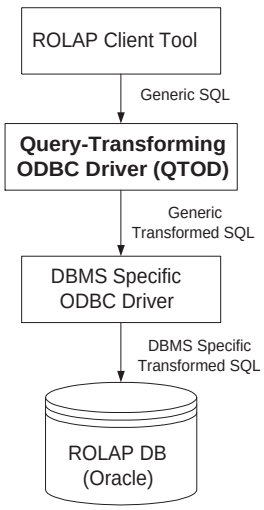

Figure 2: System Architecture

The ROLAP client tool , in this case the ROLAP tool Synchrony which originated from Kimball's Startracker tool [1], makes SQL requests to the ROLAP database, in this case the Oracle8 RDBMS, using the ODBC standard. We have implemented a special, query-transforming ODBC driver (QTOD) that, based on case-specific metadata, trans-

DiagID	Lowlevel	Family	Group
5	Insulin dependent diabetes during pregnancy	Diabetes during pregnancy	Diabetes
5	Insulin dependent diabetes during pregnancy	Diabetes during pregnancy	Pregnancy related
5	Insulin dependent diabetes during pregnancy	Insulin dependent diabetes	Diabetes
6	Non insulin dependent diabetes during pregnancy	Diabetes during pregnancy	Diabetes
6	Non insulin dependent diabetes during pregnancy	Diabetes during pregnancy	Pregnancy related
6	Non insulin dependent diabetes during pregnancy	Non insulin dependent diabetes	Diabetes
100	!Lowlevel!Lung Cancer	Lung cancer	Cancer

Table 2: DDiagnosis Dimension Table

forms the SQL requests into requests that hide the transformations from the users, returning the query results that the user would expect based on the original hierarchies. A transformed request is submitted to the OLAP DB using an RDBMS-specific ODBC driver. The QTOD component is common to all RDBMSs, so Oracle8 may be replaced by another RDBMS such as IBM DB2, Informix, or MS SQL Server. Another ROLAP tool may also be used, making the solution quite general and flexible.

We have chosen to base the prototype on an RDBMS (Oracle8) since RDBMSs are the most commonly used platform for Data Warehouse and OLAP applications. Additionally, the major RDBMSs now, like dedicated multidimensional DBMSes (MDDBs), use pre-aggregated data for faster query responses [6]. However, the approach could also be implemented using multidimensional technology, e.g., based on the Microsoft OLE DB for OLAP standard [2].

The transformation algorithms are implemented in Oracle's PL/SQL programming language. The transformations are relatively fast, taking at most a few minutes, even for large dimensions. Once the dimension hierarchies have been transformed, the QTOD transforms queries and results between the original and transformed hierarchies. The QTOD is a thin layer and adds very little overhead to queries. It is implemented using GNU Flex++/Bison++ scanner/parser generators and the MS Visual C++ compiler.

4 Implementation Specifics

Studies have shown that queries on a data warehouse consist of 80% *navigational* queries, which explore the dimension hierarchies, and 20% *aggregation* queries, which aggregate the data at various levels of detail [1]. These two types of queries are treated differently to give the user the illusion that the dimension hierarchies have their original form.

The multidimensional data is captured in a *star schema* [1]. The dimension table for the Diagnosis dimension is given in Table 2, which has one column for the low-level diagnosis ID in addition to columns for the textual descriptions of low-level diagnoses, diagnosis families, and diagnosis groups.

The hierarchy captured in the table is *partially normalized*, i.e., placeholder values have been introduced to balance the hierarchy (but it remains non-strict). Specifically, the "!Lowlevel!Lung Cancer" placeholder value has been inserted into the Low-level Diagnosis level. We prefix such values with a "!" and their level to indicate that they are inserted by the transformation process. Note the multiple occurrences of lower-level values caused by the non-strictness of the hierarchy. This is the table that will be used for user navigation in the hierarchy. Its name is prefixed with a "D" to distinguish it from another "Diagnosis" dimension table (described below), to be used for aggregation queries.

We now describe how to achieve transformation transparency for *navigational queries*. The query below retrieves all low-level diagnosis names.

```
SELECT DISTINCT Lowlevel
FROM Diagnosis
```

Navigational queries issued by ROLAP tools have exactly this format. The query is transformed by the QTOD into the query below, which operates against the table DDiagnosis. The transformed query returns the result seen in Table 3.

```
SELECT DISTINCT Lowlevel
FROM DDiagnosis
WHERE Lowlevel NOT LIKE '!%'
```

Lowlevel
Insulin dependent diabetes during pregnancy
Non insulin dependent diabetes during pregnancy

Table 3: Navigational Query Result

Due to the use of `DISTINCT` as a quantifier, duplicates are not returned. The `NOT LIKE` predicate removes the placeholder values inserted into the hierarchy to balance it, which in this case is the value "!Lowlevel!Lung Cancer." As desired, the result is unaffected by the translations.

For *aggregation queries*, it is also possible to achieve transformation transparency, although this is more difficult. For dimensions with non-strictness, a special dimension table is introduced that holds only the part of the normalized hierarchy that does *not* contain non-strictness. In the normalized hierarchy to the right in Figure 1, this part is the Low-level Diagnosis category and the two special categories introduced by the normalization process to hold *sets of diagnosis families* and *sets of diagnosis groups*, respectively. This part of the hierarchy is implemented in the Diagnosis dimension table seen in Table 4.

DiagID	Lowlevel	Family	Group
1000020	!Low-level Diagnosis!Lung cancer	14	13
5	Insulin dependent diabetes during pregnancy	4,9	11,12
6	Non insulin dependent diabetes during pregnancy	4,10	11,12

Diagnosis

Group	SGroup
Cancer	13
Diabetes	11,12
Pregnancy Related	11,12

SGroup

Table 4: Dimension and Group Tables for Aggregation

The "Lowlevel" column contains the normal textual diagnosis description, whereas the special "Family" and "Group" columns contain comma-separated ordered lists of the IDs of the sets of values that are represented by the column values. For example, value "4,9" represents the set $\{4, 9\}$.

We need to capture the remaining part of the hierarchy, which consists of non-strict mappings from a "set-of-X" category to the "X" category, e.g., the mapping of the "set-of-Diagnosis Group" category to the "Diagnosis Group" category to the right in Figure 1, which maps $\{13\}$ to 13 (Cancer) and $\{11, 12\}$ to 11 (Diabetes) and 12 (Pregnancy Related). This is done by introducing a special table for each such mapping, named by the category prefixed with an "S" (for Set-of). For example, for the Diagnosis Group category, table "SGroup" in Table 4 maps sets of diagnosis groups to the individual diagnosis groups in the sets. The "Group" column represents the diagnosis group, while the "SGroup" column represents the associated set of diagnosis groups.

With these tables available, it is possible to obtain transformation transparency for aggregation queries. A ROLAP aggregation query has the format of the query below that computes the number of patients per diagnosis group.

```
SELECT Diagnosis.Group, SUM(Patient.Count)
FROM Diagnosis,Patient
WHERE Diagnosis.DiagID=Patient.DiagID
GROUP BY Diagnosis.Group
```

This is transformed into the query given next.

```
SELECT SGroup.Group, SUM(QQQQQQQ.Count)
FROM Sgroup,
 (SELECT Diagnosis.Group,
         SUM(Patient.Count) AS Count
  FROM Diagnosis,Patient
  WHERE Diagnosis.DiagID=Patient.DiagID
  GROUP BY Diagnosis.Group) QQQQQQQ
WHERE QQQQQQQ.Group=SGroup.SGroup AND
      SGroup.SGroup NOT LIKE '!%'
GROUP BY SGroup.Sgroup
```

The transformed aggregation query has two parts. The nested table expression computes the number of patients per *set of diagnosis group*, making this available via correlation name QQQQQQQ. This part of the hierarchy is a balanced tree, so the RDBMS can safely use pre-aggregated data for optimizing the query performance. The result of the nested table expression is used in the outer query, which aggregates the last part of the way up to the diagnosis groups using the "SGroup" table. The outer query also filters out any placeholder values inserted by the normalization process (prefixed with a "!"). As a result, the client OLAP tool will retrieve the expected result.

Good query performance without the use of excessive storage for pre-aggregated data is obtained by using practical pre-aggregation for the "nice" part of the hierarchy captured in the "Diagnosis" dimension table. The query transformation exemplified here can be performed for all ROLAP aggregation queries, making the solution quite general.

5 Demonstration

Based on concrete data from a real-world case study, the demonstration will initially show snapshots that illustrate the hierarchy normalization process. Next, query processing will be demonstrated by means of concrete navigational and aggregation queries. This includes a description of how the queries are transformed to hide the hierarchy transformations from the user, as well as the evaluation of the queries on concrete data. Finally, the demonstration will compare the query execution times for the queries and the amount of storage required for pre-aggregated data with the two alternatives to our approach, namely *no* pre-aggregation, which gives very long query response times, and *full* pre-aggregation, which requires unrealistically large amounts of storage for pre-aggregated data.

Supporting material in the form of slides and posters will be used in the demonstration.

References

[1] R. Kimball. *The Data Warehouse Toolkit*. Wiley Computer Publishing, 1996.

[2] Microsoft Corporation. OLE DB for OLAP Version 1.0 Specification. Microsoft Technical Document, 1998.

[3] The OLAP Report. *Database Explosion*. <www.olapreport.com/DatabaseExplosion.htm>. Current as of February 18, 2000.

[4] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Extending Practical Pre-Aggregation in On-Line Analytical Processing. In *Proc. VLDB*, pp. 663–674, 1999. Extended version available as TR R-99-5004, Dept. of Comp. Sci. Aalborg University, <www.cs.auc.dk/~tbp/articles/R995004.ps>, 1999.

[5] A. Shukla et al. Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies. In *Proc. VLDB*, pp. 522–531, 1996.

[6] R. Winter. Databases: Back in the OLAP game. *Intelligent Enterprise Magazine*, 1(4):60–64, 1998.