Hierarchy-based Update Propagation in Decision Support Systems *

Haitang Feng[1,2], Nicolas Lumineau[1], Mohand-Saïd Hacid[1], and Richard Domps[2]

[1] Université de Lyon, CNRS
Université Lyon 1, LIRIS, UMR5205, F-69622, France
{firstname.lastname}@liris.cnrs.fr
[2] Anticipeo, 4 bis impasse Courteline, 94800, Villejuif, France
{firstname.lastname}@anticipeo.com

Abstract. Sales forecasting systems are used by enterprise managers and executives to better understand the market trends and prepare appropriate business plans. These decision support systems usually use a data warehouse to store data and OLAP tools to visualize query results. A specific feature of sales forecasting systems regarding future predictions modification is backward propagation of updates, which is the computation of the impact of modifications on summaries over base data. In Data warehouses domain, some methods propagate updates in hierarchies when data sources are subject to modifications. However, very few works have been performed so far regarding update propagation from summaries to data sources. This paper proposes an algorithm named PAM algorithm, to efficiently propagate modifications on summaries. Experiments on an operational application (Anticipeo[3]) have been performed to validate our algorithm.

Keywords: Aggregate update propagation, Data warehouses, Decision support systems, Dimension-Hierarchy

1 Introduction

A forecasting system is a set of techniques or tools mainly used for analysis of historical data, selection of most appropriate modeling structure, model validation, development of forecasts, and monitoring and adjustment of forecasts[4]. The most frequently used forecasting systems relate to domains like weather, traffic or sales.
A sales forecasting system (SFS), also called a business forecasting system is a kind of forecasting system allowing achievable sales revenue, based on historical

* Research partially supported by the French Agency ANRT (www.anrt.asso.fr) and by Anticipeo (www.anticipeo.com).
[3] See http://www.anticipeo.com
[4] See http://www.businessdictionary.com/definition/forecasting-system.html.

sales data, analysis of market surveys and trends, and salespersons' estimates[5]. To be effective, a SFS must adhere to some fundamental principles such as the use of a suite of time-series techniques [12].
The basic functionalities a SFS supports, are: computation, visualization and modification. The first functionality, computation of forecasts, uses specific methods (typically statistical models) to derive sales forecasts. Several predictive methods have been introduced in the domain of statistics (see, e.g., [10][11]). The second functionality, visualization of computed forecasts, uses OLAP (online analytical processing) tools to visualize data stored in a data warehouse (DW). The visualization methods are investigated, especially by resorting to "roll-up" and "drill-down" operators of OLAP or reporting tools of BI (business intelligence) (see, e.g., [8][5][13]). However, the third functionality, modification of computed forecasts during visualization, is a specific problem. In SFSs, source data are composed of historical data and predictive data while predictive data are not as stable as traditional source data that we employ in a traditional DW. Experienced salespersons and sales managers could make some modifications to adjust computed forecasts to some specific situations, e.g., occasional offers. These adjustments occur on summarized data and should be propagated to source data (facts and forecasts) and then to other summarized data. Some approaches dealing with view maintenance in OLAP were proposed. Some of them focus on the evolution of multidimensional structure (see, e.g., [1][9]) and others focus on the optimization of OLAP operators such as pivot and unpivot (see, e.g., [3]). Approaches to view maintenance in DWs were also investigated. They concern combined view updates (see, e.g. [14]), multi-view consistency over distributed data sources (e.g., [2]) and many others. The main context of these approaches is the propagation of source updates to materialized views. To the best of our knowledge, the problem of updating summaries and computing the effect on row data has not been investigated so far. The motivating case study we consider is a real sales forecasting system called Anticipeo. In this application, we are facing the optimization problem regarding update propagation from summarized data to base data and to other summarized data.
The rest of this paper is organized as follows : Section 2 defines the sales forecasts modification problem. Section 3 discusses the current and ad-hoc solutions to support this issue. In Section 4, we describe a novel algorithm based on intelligent exploitation of dimension-hierarchies. Section 5 describes the evaluation and the experimental results. We conclude in Section 6.

2 Problem Statement and Motivations

To clearly define our problem, we first review the use of dimensions (and hierarchies) and the basic data schema by visualization tools of OLAP systems [4]. OLAP systems employ multidimensional data models to structure "raw" data into multidimensional structures in which each data entry is shaped into a fact with associated measure(s) and descriptive dimensions that characterize the fact.

[5] See http://www.businessdictionary.com/definition/sales-forecast.html.

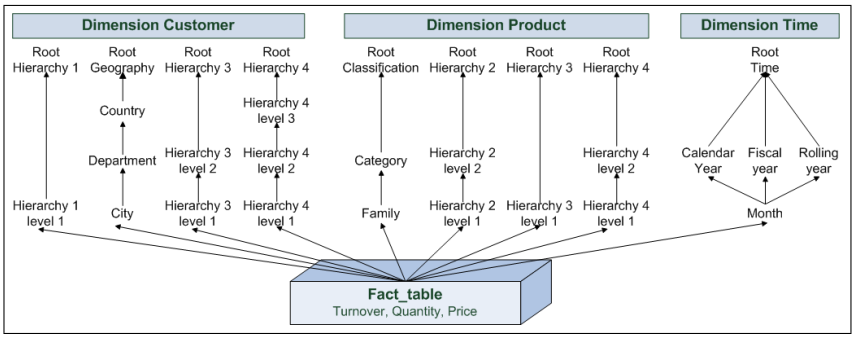

Fig. 1. Example of dimension-hierarchies and fact tables

The values within a dimension can be further organized in a containment type hierarchy to support multiple granularities. In our case, we are interested in quantitative measures, not qualitative measures.
In the example shown Fig. 1, we present the data model in a sales forecasting system. This data model is based on one fact table with three measures: *turnover*, *quantity* and *price*; and three different dimensions: *customer*, *product* and *time*. Each dimension has its hierarchies to describe the organization of data. Customer dimension has 4 hierarchies, product dimension has 4 hierarchies and time dimension has 3 hierarchies. For instance, the second hierarchy of customer dimension is a geographical hierarchy. Customers are grouped by city for level 1, by department for level 2 and by country for level 3. Base sales are aggregated on each level according to this geographical organization when one analyzes the sales through this hierarchy.
Regarding the visualization, the calculation of aggregated information needs to be performed on the fly. OLAP systems employ materialized views or fictive information to avoid extra response time. In the example of sales, fictive customers and fictive products are added to represent elements in superior hierarchy level, such as the creation of a fictive customer for the city of Lyon, a second fictive customer for the department Rhône and a third one for the country France. Thus, the system has three new entries in the customer dimension and accordingly some aggregated sales in the fact table regarding these newly created customers. Finally, all elements of every hierarchy level from every dimension are aggregated and added to the dimension and fact tables. This pre-calculation guarantees an immediate access to any direct aggregated information while users perform visualization demands.

Regarding forecasting systems, the visualization is not the last operation as in other OLAP systems. The sales forecasts produced by a system are a first version which are reviewed by experienced salespersons. Salespersons check these automatically generated sales forecasts, take into consideration some issues not considered by the system (e.g. promotional offers) and perform some necessary modifications. In other cases, salespersons can also perform some modifications

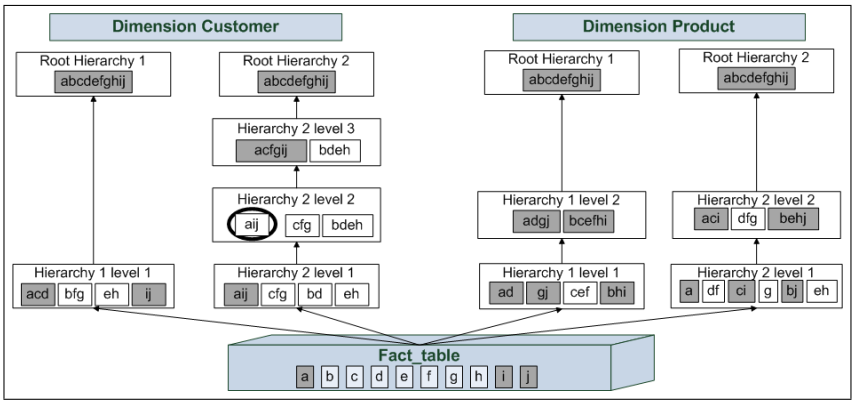

Fig. 2. Example of data modification on an aggregated level

on summaries in order to simulate a new marketing target. This update taking place on an aggregated level constitutes the major feature of sales forecasting systems. Compared to classical OLAP systems in which source data are considered as static, data in sales forecasting systems can be modified many times to obtain a final result.

Hence, sales forecasting systems need to have the ability to quickly react to data modification on an aggregated level. Fig. 2 is an example to show how an aggregation-level modification impacts all the data.

In this example and for the sake of simplicity, we consider only the first two hierarchies respectively from customer dimension and product dimension of the previous data model (see Fig. 1). In the fact table, we group all the tuples into 10 sets: named from a to j (For the following, these sets are named "base tuple"). Aggregates at superior hierarchical levels are presented by rectangles including the composing base tuples and are formalized by the function α with the composing base tuples in parameters: α(x,y,...). For instance, the circled rectangle, or the aggregate α(a,i,j) on level 2 of hierarchy 2 of the customer dimension containing aij means that the result of this aggregate is generated from the base tuples aij. In this specific case, the result of the aggregate α(a,i,j) is the sum of sales: a, i and j. Other aggregates are generated and presented in the same manner. The root rectangle of every hierarchy represents all the sales.

Figure 2 shows the underlying data structure when the system presents the prediction result to sales managers. Sales managers analyze the sales and then decide to correct the sales of the aggregate α(a,i,j) (i.e. to evaluate beforehand the impact of a strategical or tactical move). Let us see the impact of this modification. As the aggregate α(a,i,j) is generated from a, i and j, once its result is corrected, the results of the three tuples should be afterwards corrected. Meanwhile, these three tuples are also the base tuples which constitute other aggregates in other hierarchies of all dimensions. All aggregates containing any of these three tuples in its composition should be corrected as well. The aggregates impacted by the modification on the aggregate α(a,i,j) are darkened in Fig. 2.

Sales managers may perform the modifications many times to obtain a satisfying result. Sales forecasting systems should provide a short delay between the modification and the visualization of modified data. Hence, the problem we need to deal with is how to efficiently update aggregated data through a dimension-hierarchy structure.

3 Current Solution

A current solution consists in identifying approaches to similar problems and builds on the implemented solutions. In this system, methods to calculate the aggregates are well defined. The steps of the current solution which recomputes everything are as follows:

1. calculate the base tuples wrt the modification and rules,
2. recompute all the aggregates.

To illustrate this process, consider the example shown Fig. 2. The actual result of the aggregate α(a,i,j) is 500 000 euros, but the sales manager has a new marketing plan and (s)he estimates the sales can achieve 600 000 euros. Then (s)he does this modification which is then processed as follows:

Step 1: calculation of modified base tuples
The example in Fig. 2 indicates that the result of the aggregate α(a,i,j) is composed of three tuples: a, i and j; suppose that the distribution of sales is respectively 100 000 euros, 200 000 euros and 200 000 euros for the three tuples. When the sales of the aggregate aij raise to 600 000 euros, the modification is spread over these tuples with respect to the weight of every tuple according to the company's previously defined rules; the weight in this case is the actual sales. For a, i and j, their weights are respective 100 000, 200 000 and 200 000 (i.e. 1:2:2). The formula to calculate corrected result for a tuple t is

$$eval_{\alpha,T}(t) = val(t) + (val'(\alpha, T) - val(\alpha, T)) * \frac{weight(t)}{\sum_{t' \in T} weight(t')}$$

where α is an aggregation operator, T is a set of tuples, val returns current value of a tuple or an aggregate and val' returns new value of a tuple or an aggregate. The corrected results for T={a,i,j} are then:

$eval_{\alpha,T}(a) = 100000 + (600000 - 500000) * \frac{1}{1+2+2} = 120000$
$eval_{\alpha,T}(i) = 200000 + (600000 - 500000) * \frac{2}{1+2+2} = 240000$
$eval_{\alpha,T}(j) = 200000 + (600000 - 500000) * \frac{2}{1+2+2} = 240000$

Step 2: recalculation of aggregated information
The second step consists in recomputing the aggregates on higher levels. We follow the same process as when the aggregates are created using the hierarchy dependencies. For instance, the aggregate α(a,c,d) is an aggregate of the base tuples a, c and d; so its new result is calculated by summing the sales of a, c and d.

Following this straightforward solution, we can regenerate all the hierarchies of the whole schema with updated data.

4 Update Propagation Algorithm

The current solution advocates the calculation of all the aggregates of all the hierarchies. However, this solution does some useless work. If we closely look at the recomputed aggregates in Fig. 2, only the dark ones are concerned with the modification and need to be updated, that is, **18** aggregates out of **32**. Hence, the current solution leads to the calculation of **14** aggregates in vain. The key idea is thus to be able to identify and recompute only the concerned elements. By considering the dependencies between aggregates and base tuples, we can identify the exact aggregates to modify and hence avoid useless work.
Another drawback with the current solution is its heavy recomputing procedure. Operations of removing and adding aggregates demand heavy maintenance of index tables and physical storage. Nevertheless, our approach can keep the aggregates at their logical and physical location and avoid extra effort.
To summarize, our approach follows the following steps:

1. retrieval of participating tuples (from the fact table) to the aggregate
2. creation of a temporary table for the base tuples to be updated and calculation of the value difference for every base tuple
3. update of impacted base tuples
4. identification of impacted aggregates
5. update of impacted aggregates with the value differences of base tuples

The algorithm for the update propagation through a dimension-hierarchy architecture is shown Table 1. Line 1 to line 4 identify the base tuples involved in

Algorithm PAM (Propagation of Aggregate Modification)	
Input: Schema S, aggregate X, its actual result AR and its objective result OR	
Output: An updated schema S' of all hierarchies	
Algorithm:	
1:	Calculate the modification of the aggregate X: δ = OR - AR
2:	Retrieve participating base tuples of X : CX = $\{x_1, x_2, ..., x_n\}$
3:	Create a temporary table: ΔX = CX
4:	Calculate the differences for every base tuple: $\forall x_i \in \Delta$X: $\delta_i = \delta$ * weight_coeff$_i$ Add the result to ΔX
5:	Update all the base tuples impacted: $\forall e_i \in \Delta$X: value_of_t_i = value_of_t_i + δ_i
6:	Identify aggregates to update: A = {results after filtering by dependencies to ΔX}
7:	For A_i in A
8:	Retrieve its composition: $CA_i = \{a_1, a_2, ..., a_m\}$ Identify base tuples which are also in ΔX: U = ΔX $\cap$ CA_i Calculate new result of A_i: $\forall u_i \in U$ value_of_A_i = value_of_A_i + δ_{u_i}
9:	End for

Table 1. Algorithm for the update propagation

the modification and calculate their values in difference. Line 5 allows to update these base tuples. Line 6 to line 9 identify impacted aggregates and perform the update.
Let us take the previous example to illustrate the approach. A sales manager changes the sales of the aggregate α(a,i,j) from 500 000 euros to 600 000 euros. Once (s)he confirms the modification, the system will proceed using the algorithm in Table 1.

Step 1: retrieval of the participating tuples to the aggregate
Retrieve the composition of the aggregate α(a,i,j): sales of the aggregate α(a,i,j) is the sum of a, i and j. Hence, the composing tuples are a, i and j.

Step 2: creation of temporary table and calculation
Create a temporary table ΔX for the base tuples identified in step 1.
Calculate the δ for the aggregate α(a,i,j): δ = 600000 - 500000 = 100000
Considering W=$\sum_{t\in\{a,i,j\}} weight(t)$, calculate the differences in value for every tuple using the weight coefficient.

$\delta_a = \delta * \frac{weight(a)}{W} = 100000 * \frac{1}{1+2+2} = 20000$

$\delta_i = \delta * \frac{weight(i)}{W} = 100000 * \frac{2}{1+2+2} = 40000$

$\delta_j = \delta * \frac{weight(j)}{W} = 100000 * \frac{2}{1+2+2} = 40000$

The differences in value of base tuples are added to the temporary table. This table also contains the dependency information to higher hierarchical levels (shown Table 2).

Step 3: update of base tuples
Update the base tuples impacted by the aggregate modification (same procedure as in the current solution). The new values of these base tuples are computed by their actual values and the values in difference calculated in step 2. In this case, a is updated to 120 000, i to 240 000 and j to 240 000.

Step 4: identification of impacted aggregates
Identify all the aggregates concerned with the modification of the sales of the aggregate α(a,i,j) by using the links between aggregates and registered base tuples in the temporary table ΔX. In this case, we identify all the dark rectangles.

Step 5: update of impacted aggregates
Propagate the changes to every involved aggregate. Let us illustrate this issue with the customer dimension hierarchy 1. We loop for every level of the hierarchy.

element_identifier	customer_dim_link	product_dim_link	δ_x
a	customer_link$_a$	product_link$_a$	20000
i	customer_link$_i$	product_link$_i$	40000
j	customer_link$_j$	product_link$_j$	40000

Table 2. Temporary table ΔX created to store influenced base tuples

For level 1, two aggregates need to be updated: *acd* and *ij*. The aggregate *acd* is composed of *a*, *c* and *d* and among these base tuples, only one is registered in the table ΔX, namely, the base tuple *a*. Hence, the value of *acd* is changed only by adding the δ_a or 20 000.
$val'(\alpha, \{a, c, d\}) = val(\alpha, \{a, c, d\}) + \delta_a = val(\alpha, \{a, c, d\}) + 20\ 000$
Another element *ij* of level 1 and the root element *abcdefghij* on level 2 can be calculated in a similar way: $val'(\alpha, \{i, j\}) = val(\alpha, \{i, j\}) + 40\ 000 + 40\ 000$; $val'(\alpha, \{a, b, c, d, e, f, g, h, i, j\}) = val(\alpha, \{a, b, c, d, e, f, g, h, i, j\}) + 20\ 000 + 40\ 000 + 40\ 000$.
Doing this way, we updated only the aggregates impacted by the modification for hierarchy 1 of the customer dimension. The propagation in other hierarchies are processed in the same manner. Finally, we obtain updated data over the entire schema.

5 Experiments

The main technical characteristics of the server on which we run the evaluation are: two Intel Quad core Xeon-based 2.4GHz, 16GB RAM and one SAS disk of 500GB. The operating system is a 64-bit Linux Debian system using EXT3 file system. Our evaluation has been performed on real data (copy of Anticipeo database) implemented on MySQL.
The total size of the database is 50 GB of which 50% is used in the computation engine, 45% for result visualization and 5% for the application. Our test only focuses on the data used by the update: one fact table and 2 dimension tables: *customer* and *product*. The fact table is about 300 MB with 257.8MB of data and 40.1 MB of index. The customer dimension table contains 5420 real customers and 1319 fictive customers (6559 in total) and the product dimension table contains 8256 real products and 404 fictive products (8660 in total). Each of these dimension tables is composed of 4 hierarchies. It presents a similar structure to Fig. 1 with different number of levels in each hierarchy (from 2 levels to 4 levels). Note that the time dimension is merged in the fact table for some performance issues [6][7]. Hence, only two explicit dimensions are materialized in dimension tables.
The objective of the evaluation is to compare the time of the whole schema update using the current solution and our algorithm. The tests are performed on 3 hierarchies which have 2, 3 and 4 levels, respectively. In our evaluation, we modify one aggregate from each level of every hierarchy to compare the evaluation time resulting form the current solution and from our approach.
We first perform tests with the current solution. The result is shown Table 3. Step 2 and step 3 stays almost the same for different hierarchies because the whole schema is recomputed for each update.
The same tests are performed with our algorithm. The result is shown Table 4. The time spent on the base tuples update is almost the same as the current solution because they follow the same procedure.
We then compare the total evaluation time using the two solutions in one chart

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
	Level 1	Level 2	Level 1	Level 2	Level 3	Level 1	Level 2	Level 3	Level 4
Step 1*	0.9	7.9	0.9	1.0	7.5	0.08	0.8	2.9	7.8
Step 2*	0.1	0.2	0.1	0.2	0.1	0.2	0.1	0.2	0.2
Step 3*	179.4	181.9	185.6	181.2	188.3	180.9	179.5	179.7	176.4
Total	180.4	190.0	186.6	182.4	195.9	181.2	180.4	182.8	184.4

* Step 1: updating base tuples; Step 2: deleting aggregates; Step 3: reconstructing

Table 3. Evaluation time of an aggregate modification using the current solution

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
	Level 1	Level 2	Level 1	Level 2	Level 3	Level 1	Level 2	Level 3	Level 4
Step 1*	0.3	2.9	0.3	0.3	2.9	0.04	0.3	1.2	3.0
Step 2*	0.9	7.8	0.9	2.0	7.2	0.1	0.8	2.8	7.7
Step 3*	5.4	53.5	5.2	5.4	56.1	0.6	4.3	21.0	54.4
Total	6.6	64.2	6.4	6.7	66.2	0.7	5.4	25.0	65.1

* Step 1: creating temporary table; Step 2: updating base tuples; Step 3: propagating

Table 4. Evaluation time of an aggregate modification using our algorithm

shown Fig. 3. The scenario of modification on the root (top level of every hi-

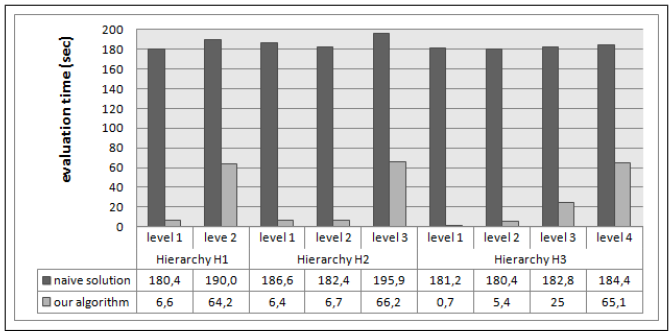

Fig. 3. Comparison of evaluation time using two solutions

erarchy) takes much more time than other levels and stays almost the same because we modify all the base tuples[6]. Even so, we get a nearly 200% better performance. In all other pratical cases, our algorithm presents more than 3000% better performance in average.

[6] The modification of the root aggregate is rare in the application. We perform this experiment to show the worst evaluation time we can have in our approach.

6 Conclusion

In this paper, we discussed the problem of efficiently propagate an aggregate modification through a dimension-hierarchy structure. Current solution naively recomputes all the aggregates of all the hierarchies, which is time-consuming and does not fulfill the performance needs. We proposed the algorithm PAM to reduce the modification cost. Our algorithm is based on the dependencies of aggregates and bases sales. It identifies the exact sets of aggregates to be updated and calculates at the mean time the value in difference for each aggregate. We proved that with our approach, the update propagation time can be reduced by more than 3000% compared to the current solution implemented in a real application.
For further work, we will take into consideration more factors that could affect the evaluation time, such as complexity of dimensions, complexity of hierarchies, complexity of queries, etc. And we will also investigate the update propagation of an aggregate which results from multi-hierarchies.

References

1. M. Body, M. Miquel, Y. Bedard, and A. Tchounikine. Handling Evolutions in Multidimensional Structures. In *ICDE*, pages 581–591, 2003.
2. S. Chen, B. Liu, and E. A. Rundensteiner. Multiversion-based view maintenance over distributed data sources. *ACM TODS*, 29(4):675–709, 2004.
3. S. Chen and E. A. Rundensteiner. Gpivot: Efficient incremental maintenance of complex rolap views. pages 552–563, 2005.
4. E. F. Codd, S. B. Codd, and C. T. Salley. Providing OLAP (On-Line Analytical Processing) to User-Analysis: An IT Mandate, 1993.
5. K. R. Divesh, K. A. Ross, D. Srivastava, and D. Chatziantoniou. Complex aggregation at multiple granularities. In *EDBT*, pages 263–277, 1998.
6. H. Feng. Performance problems of forecasting systems. In *ADBIS Proceddings II*, pages 254–261, 2011.
7. H. Feng. Data management in forecasting systems: Case study - performance problems and preliminary results. In *BDA*, 2011 (to appear).
8. H. Gupta. Selection of views to materialize in a data warehouse. In *ICDT*, pages 98–112, 1997.
9. C. A. Hurtado, A. O. Mendelzon, and A. A. Vaisman. Maintaining data cubes under dimension updates. In *ICDE*, pages 346–355, 1999.
10. Jain and Chaman. Which forecasting model should we use? *Journal of Business Forecasting Methods & Systems*, Fall 2000.
11. McKeefry and H. Lynne. Adding more science to the art of forecasting. *EBN*, March 2001.
12. J. T. Mentzer and C. C. Bienstock. The seven principles of sales-forecasting systems. *Supply Chain Management Review*, pages 76–83, 1998.
13. D. Theodoratos. Exploiting hierarchical clustering in evaluating multidimensional aggregation queries. In *DOLAP*, pages 63–70, 2003.
14. J. Zhou, P.-A. Larson, and H. G. Elmongui. Lazy maintenance of materialized views. In *VLDB*, page 231242, 2007.