



Numéro d'ordre : 174-2012 Année 2012

Université Claude Bernard Lyon 1
Laboratoire d'InfoRmatique en Image et Systèmes d'information
École Doctorale Informatique et Mathématiques de Lyon

Thèse de l'Université de Lyon

Présentée en vue d'obtenir le grade de Docteur,
spécialité Informatique

par

Haitang FENG

Data Management in Forecasting Systems: Optimization and Maintenance

---

Thèse soutenue le 17 octobre 2012 devant le jury composé de :

Mme.	Anne Doucet	Professeur à l'Université Paris 6	Rapporteuse
M.	François Pinet	Chargé de Recherche à l'IRSTEA de Clermont-Ferrand	Rapporteur
M.	Jérôme Darmont	Professeur à l'Université Lumière Lyon 2	Examinateur
M.	Didier Donsez	Professeur à l'Université Grenoble 1	Examinateur
M.	Mohand-Saïd Hacid	Professeur à l'Université Lyon 1	Directeur
M.	Nicolas Lumineau	Maître de Conférences à l'Université Lyon 1	Co-encadrant
M.	Richard Domps	Président d'Anticipeo	Co-encadrant

Laboratoire d'InfoRmatique en Image et Systèmes d'information
UMR 5205 CNRS - Lyon 1 - Bât. Nautibus
69622 Villeurbanne Cedex - France

Acknowledgment

During the three years of my PhD, I have realized that even the hardest work can be achieved by discussions, collaborations and support. It is a great pleasure to thank those who have helped me.

First of all, I wish to express my gratitude to my supervisors. Pr. Mohand-Saïd Hacid, the chief supervisor, was abundantly helpful and offered invaluable assistance, support and guidance. His experience on the thesis subject and on the guidance of PhD students was a treasure to me. Moreover, his passion of work was, is and will be my long-lasted motivation for work. To me, it was also very fortunate have another research supervisor, Dr. Nicolas Lumineau. He guided me tightly through three years: from the analysis, the conception, the implementation, the experimentation up to the final stage. Our discussions about the research and the life will not be forgotten. I would also like to deeply thank Mr. Richard Domps, the thesis promoter and also the industrial supervisor, who has chosen me for this thesis and has always given his confidence to me. He provides me with the best work condition, which is so appreciated.

I wish to thank Professors Anne Doucet and François Pinet for their interest in my research work and for their willingness to be the reviewers of this thesis. I also want to thank Professors Jérôme Darmont and Didier Donsez who have agreed to participate in the jury board. I look forward to hearing valuable remarks.

A special thank goes to M. Alain Malacchina, my technical manager in Anticipeo. His industrial and technical point of view helped me keep a balance between academic research and the real world. His attitude towards work set me an example and gave me a lot of positive energy.

I would like to thank all the members of database group, for both friendly and scientific discussions. Of course, I will not forget Ho Lee, who shared the same office with me and supported and advised me for two long years. His encouragement is an important part of the success of this thesis. I also thank other members of LIRIS that I was lucky to know for their encouragement.

I also wish to thank my friends out of the laboratory, in particular, Lina Lu, Marie-France Eymard, Alain Iametti, Lorène Leonidas, Dominika Grabowska, Nadia Derbas and Xiaohuli Zhang, for sharing the happiness and sadness of my life.

I owe my deepest gratitude to my parents and my family for their endless love,

support, trust, and presence. I cannot be here without all of you.

Last but not least, I want to thank specially my husband Zhan Zhang for his support and love. I feel so fortune to be loved by this kind and warmhearted man.

Thank you all!

Haitang Feng

Abstract

In daily life, more and more forecasting systems are used to determine what the future holds in many areas like climate, weather, traffic, health, finance, and tourism. These predictive analytics systems support three functionalities: prediction, visualization and simulation based on modifications. A specific problem for forecasting systems is to ensure data consistency after data modification and to allow updated data access within a short latency.

Forecasting systems are usually based on data warehouses for data storage, and OLAP tools for historical and predictive data visualization. Data that are presented to and modified by end users are aggregated data. Hence, the research issue can be described as the propagation of an aggregate-based modification in hierarchies and dimensions in a data warehouse environment. This issue corresponds to a view maintenance problem in a data warehouse. There exists a great number of research works on view maintenance problems in data warehouses. However, they only consider updates on source data or evolution of the structure of dimensions and hierarchies. To our knowledge, the impact of aggregate modifications on raw data was not investigated. In addition, end users perform the modification interactively. The propagation of the modification should be efficient in order to provide an acceptable response time.

This "Conventions Industrielles de Formation par la REcherche (CIFRE)" thesis is supported by the "Association Nationale de la Recherche et de la Technologie (ANRT)" and the company Anticipeo. The Anticipeo application is a sales forecasting system that predicts future sales in order to help enterprise decision-makers to draw appropriate business strategies in advance. By the beginning of the thesis, the customers of Anticipeo were satisfied by the precision of the prediction results, but there were unidentifiable performance problems.

During the working period, the work can be divided into two parts. In the first part, in order to identify the latency provenance, we performed an audit on the existing application. The result of audit showed the database may be the main source of latency. We proposed a methodology relying on different technical approaches to improve the performance of the application. Our methodology covers several aspects from hardware to software, from programming to database design. The response time of the application has been significantly improved. However, there was still a situation which cannot be solved by these technical solutions. It concerns the propagation of an aggregate-based modification in a data warehouse. The second part of our work consists in the proposi-

tion of a new algorithm (PAM - Propagation of Aggregate-based Modification) with an extended version (PAM II) to efficiently propagate an aggregate-based modification. The algorithms identify and update the exact sets of source data and other aggregates impacted by the aggregate modification. The optimized PAM II version achieves better performance compared to PAM when the use of additional semantics (e.g., dependencies) is possible. The experiments on real data of Anticipeo proved that the PAM algorithm and its extension bring better performance when treating a backward propagation.

Keywords: OLAP, Data warehousing, Decision support systems, Optimization and performance, view materialization

Résumé

De nos jours, de plus en plus de systèmes prévisionnels sont utilisés pour fournir des indications sur un phénomène dans le futur, que ce soit dans le domaine météorologique, des transports, de la santé, des finances, du tourisme... Ces systèmes d'analyse prédictive ont souvent trois fonctionnalités: la prédiction, la visualisation et la simulation par modification des résultats. Un problème spécifique pour les systèmes prévisionnels est de maintenir la cohérence des données après leur modification et de permettre un accès aux données mises à jour avec une latence faible.

Les systèmes prévisionnels reposent généralement sur des architectures de type entrepôts de données pour le stockage des données et sur les outils OLAP pour la visualisation de données historiques et prédictives. Les données présentées aux utilisateurs finaux et modifiées par ces derniers sont des données agrégées. Par conséquent, la problématique de recherche peut être décrite comme la propagation d'une modification faite sur un agrégat à travers des hiérarchies et des dimensions dans un environnement d'entrepôt de données. Cette problématique relève de la maintenance des vues dans un entrepôt de données. Il existe un grand nombre de travaux de recherche sur les problèmes de maintenance de vues dans les entrepôts de données. Cependant, ils ne considèrent que des mises à jour sur les données sources ou l'évolution de la structure des dimensions et des hiérarchies. A notre connaissance, l'impact de la mise à jour d'un agrégat sur les données de base n'a pas été exploré. En outre, les utilisateurs finaux effectuent des modifications de façon interactive à travers une interface. La propagation de la modification doit être efficace afin de fournir un temps de réponse acceptable.

Cette thèse CIFRE (Conventions Industrielles de Formation par la REcherche) est soutenue par l'ANRT (Association Nationale de la Recherche et de la Technologie) et l'entreprise Anticipeo. L'application Anticipeo est un système prévisionnel de ventes, qui prédit des ventes futures afin d'aider les décideurs d'entreprise à tirer des stratégies commerciales appropriées à l'avance. Au début de ce travail de thèse, les clients d'Anticipeo ont été satisfaits par la précision des résultats de la prédiction, mais il y avait des problèmes de performance non identifiés.

Ce travail de thèse comporte deux parties. Dans la première partie, afin d'identifier la provenance de la latence, nous avons effectué un audit sur l'application existante. Le résultat de l'audit a montré que la base de données pouvait être la source principale de la latence. Nous avons proposé une méthodologie s'appuyant sur différentes approches et techniques pour améliorer les performances d'une application. Notre méthodologie

couvre plusieurs aspects allant du matériel au logiciel, de la programmation à la conception de base de données. Le temps de réponse de l'application a été amélioré de façon significative. Cependant, il y avait encore une situation qui ne pouvait pas être résolue par ces solutions techniques. Il s'agit de la propagation d'une modification effectuée sur un agrégat dans un entrepôt de données. La deuxième partie de notre travail consiste en la proposition d'un nouvel algorithme (PAM - Propagation de modification basée sur un agrégat) avec une version étendue (PAM II) pour propager efficacement une modification effectuée sur un agrégat. Les algorithmes identifient et mettent à jour les ensembles exactes de données sources et d'autres agrégats influencés par la modification d'agrégat. La version optimisée PAM II réalise une meilleure performance par rapport à PAM quand l'utilisation d'une sémantique supplémentaire (par exemple, les dépendances) est possible. Les expériences sur des données réelles d'Anticipeo ont montré que l'algorithme PAM et son extension apportent de meilleures performances dans la propagation des mises à jour.

Mots-clefs: OLAP, Entrepôt de données, Systèmes d'aide à la décision, Optimization et performance, matérialization des vues

Contents

Acknowledgment iii

Abstract v

Résumé vii

Contents ix

List of figures xiii

List of tables xv

1 Introduction 1
- 1.1 General context . . . 1
  - 1.1.1 Forecasting systems . . . 2
  - 1.1.2 Sales forecasting systems . . . 4
  - 1.1.3 Applications of sales forecasting systems . . . 4
- 1.2 Problem statement and motivations . . . 5
- 1.3 Contributions . . . 8
- 1.4 Organization of the manuscript . . . 9

2 State of the art 11
- 2.1 Data generation . . . 12
- 2.2 Data storage . . . 13
- 2.3 Data visualization . . . 17
  - 2.3.1 On-Line Analytical Processing (OLAP) . . . 17
  - 2.3.2 View maintenance . . . 21
- 2.4 Data simulation . . . 22
- 2.5 Synthesis . . . 24

3 Aggregate-based modification: impact management 25
- 3.1 Notations and definitions . . . 26
- 3.2 Current solution: principles and limitations . . . 28

3.3 Proposed algorithm . . . 29
3.3.1 PAM Algorithm . . . 30
3.3.1.1 Description of the algorithm . . . 30
3.3.1.2 Time complexity . . . 33
3.3.2 PAM II Algorithm . . . 34
3.3.2.1 Description of the algorithm . . . 34
3.3.2.2 Time complexity . . . 35
3.3.3 Other aggregate functions . . . 36

4 Experimental evaluation and validation **39**
4.1 Presentation of the experimental environment . . . 40
4.2 Evaluation of different methods with two dimensions . . . 40
4.2.1 Current solution . . . 41
4.2.2 PAM algorithm . . . 41
4.2.2.1 Validation . . . 41
4.2.2.2 Complexity . . . 42
4.2.3 PAM II algorithm . . . 46
4.2.3.1 Validation . . . 46
4.2.3.2 Complexity . . . 47
4.2.4 Comparison of different methods . . . 49
4.3 Evaluation of different methods with three dimensions . . . 50

5 Context of this work: Anticipeo **55**
5.1 Sales forecasting systems . . . 56
5.2 Presentation of the Anticipeo application . . . 58
5.2.1 Application process . . . 58
5.2.2 User interface . . . 59
5.2.3 Data features . . . 60
5.2.4 Main manipulations . . . 62
5.2.5 Problem statement . . . 63
5.3 Optimization guideline . . . 64
5.3.1 Hardware and application programming analysis . . . 64
5.3.2 Database management system configuration . . . 65
5.3.3 Additional materialized views . . . 66
5.3.4 Database design . . . 69
5.4 Implementation and optimization results . . . 72
5.4.1 Observations on current implementation of the application . . . 72
5.4.2 Diagnosis of latency provenance . . . 73
5.4.3 Database management system configuration . . . 75
5.4.4 Selection of materialized views . . . 76
5.4.5 Database schema modification . . . 77
5.5 Overall optimization result . . . 81
5.6 Recommendations . . . 82

6 Conclusion and future work **85**
6.1 Contributions 86
6.2 Future work 87

Bibliography **89**

Author's publications **99**

List of Figures

1.1 Example of a fact table with different hierarchies of three dimensions which are used to analyze raw data 6

2.1 Methodology tree for forecasting [Armoo] 13
2.2 Data Warehousing Architecture [CD97] 15

3.1 Example of data modification on an aggregated level in a dimension-hierarchy structure and the impact of the modification 27
3.2 The database schema for meta-table storing dependency information . . . 35

4.1 Comparison of evaluation time using the current solution, the PAM and PAM II algorithms 50
4.2 Comparison of evaluation time using the current solution, the PAM and PAM II algorithms in a three-dimensional schema 53

5.1 The data process of the Anticipeo application 58
5.2 An example of forecasting sales trend presentation 59
5.3 Hierarchy organization for dimension customer and dimension product . 60
5.4 SQL query for the example of an electrical appliance manufacturer 63
5.5 MySQL server architecture and main system variables selected for the tuning 65
5.6 Illustration of hierarchy C1 and hierarchies P1, P2 in the notion of query dependence 67
5.7 Illustration of lattice constructed by the dependence information of the hierarchy C1 and the hierarchies P1, P2 68
5.8 Star schema for the Anticipeo application 72
5.9 SQL query for the example of an electrical appliance manufacturer on the star schema 72
5.10 Illustration of dimension-hierarchies about the database DB1 of the Anticipeo application 76
5.11 Modified star schema for the Anticipeo application 80
5.12 SQL query for the example of an electrical appliance manufacturer on modified star schema 80

List of Tables

3.1 Algorithm PAM for the update propagation of an aggregate modification 31
3.2 Temporary table ΔX created to store impacted raw tuples 32
3.3 Algorithm PAM II for the update propagation of a modification 36

4.1 Number of raw tuples involved by the aggregate modification on the appropriate level of hierarchies in the two-dimensional schema 41
4.2 Evaluation time of updating the whole schema following an aggregate modification by using the current solution in a two-dimensional data warehouse 41
4.3 Evaluation time of updating the whole schema following an aggregate modification by using our PAM algorithm in a two-dimensional data warehouse 42
4.4 Estimated evaluation time of updating the whole schema following an aggregate modification by using our PAM algorithm in "*DB_twice*" of two dimensions 43
4.5 Observed evaluation time of updating the whole schema following an aggregate modification by using our PAM algorithm in "*DB_twice*" of two dimensions 43
4.6 Percent difference between the estimated result and the observed result in "*DB_twice*" of two dimensions 44
4.7 Evaluation time of propagating modifications to all hierarchies using PAM algorithm under different dimension/hierarchy schemas with two dimensions 45
4.8 Evaluation time per level of propagating modifications to all hierarchies using PAM algorithm under different dimension/hierarchy schema with two dimensions 45
4.9 Evaluation time of updating the whole schema following an aggregate modification by using our derived PAM II algorithm in a two-dimensional data warehouse 47
4.10 Estimated evaluation time of updating the whole schema following an aggregate modification by using our PAM II algorithm in "*DB_twice*" of two dimensions 48

4.11 Observed evaluation time of updating the whole schema following an aggregate modification by using our PAM algorithm in "*DB_twice*" of two dimensions 48
4.12 Percent difference between the estimated result and the observed result in "*DB_twice*" of two dimensions 48
4.13 Evaluation time of propagating modifications to all hierarchies using PAM II algorithm under different dimension/hierarchy schemas with two dimensions 49
4.14 Evaluation time per level of propagating modifications to all hierarchies using PAM II algorithm under different dimension/hierarchy schemas with two dimensions 49
4.15 Number of raw tuples involved in the modification of each test 51
4.16 Evaluation time of updating the whole schema following an aggregate modification by using the current solution in a three-dimensional data warehouse 52
4.17 Evaluation time of updating the whole schema following an aggregate modification by using our PAM algorithm in a three-dimensional data warehouse 52
4.18 Evaluation time of updating the whole schema following an aggregate modification by using our PAM II algorithm in a three-dimensional data warehouse 53

5.1 Schema for one of the materialized views used for efficient data display . 61
5.2 Cost of every query/view in the example lattice 68
5.3 Benefits of each view in three rounds and possible choices for the materialization at each round 69
5.4 Schema for the first materialized view: hierarchical information 70
5.5 Schema for the second materialized view: sales information 70
5.6 Experimental databases characteristics and results 73
5.7 Average time distribution on application level and on DBMS level for the execution of different user manipulations 74
5.8 Performance comparison between different values chosen for each MySQL system variable 75
5.9 Result of theoretical gain of implementing Greedy Algorithm 77
5.10 Evaluation of queries on actual schema and on new data schema using two materialized views (Two MVs schema) 78
5.11 Evaluation of queries on actual schema and on star schema 79
5.12 Evaluation of queries on actual schema, on star schema with time dimension and on star schema without time dimension 81

Chapter 1

Introduction

Contents

1.1 General context 1
1.1.1 Forecasting systems 2
1.1.2 Sales forecasting systems 4
1.1.3 Applications of sales forecasting systems 4
1.2 Problem statement and motivations 5
1.3 Contributions 8
1.4 Organization of the manuscript 9

A Forecasting system is a specific application consuming a large number of historical data to produce predictive data reflecting the future [FSd]. A specific issue facing data management in forecasting systems is the latency of accessing aggregated data, when their results may be updated. Latency is critical when data visualization is performed via on-line applications. This problem motivates this research. The general objective of this work is to improve the performance of forecasting systems like Anticipeo.

We introduce the context of the thesis on three levels: general forecasting systems, sales forecasting systems and applications of sales forecasting systems. We then describe the problems, research issues and show our motivations. We give a sketchy presentation of our two main contributions, an optimization guideline and a novel algorithm with an extended version. Finally, we summarize the organization of this manuscript.

1.1 General context

In this section, we introduce general forecasting systems, sales forecasting systems and the concrete case of a sales forecasting system, the Anticipeo application, and some other

applications of sales forecasting systems.

1.1.1 Forecasting systems

A forecasting system comprises techniques or tools that are mainly used for analysis of historical data, for selection of the most appropriate modeling structure for the computation of forecasts, for model validation, for development of forecasts, and for monitoring and adjustment of forecasts [FSd].

In daily life, different forecasting systems are used in many areas. They help their users to achieve their objectives. We introduce some important forecasting systems in the following paragraphs.

Environmental forecasting is one of the most frequently and earliest used forecasting application. Many countries and trans-boundary agencies achieve predictions with derived statistical models specific to their domains. For instance, the European Center for Medium-Range Weather Forecasts [Eur] is an intergovernmental organization supported by 34 states established in 1975. It provides operational medium- and extended-range forecasts and a state-of-the-art super-computing facility for scientific research. The National Centers for Environmental Prediction [Nat] of the United States is another example of environmental forecasting systems. The nine centers provide national and global weather, water, climate and space weather guidance, forecasts, warnings and analyses to their partners and external user communities. These products and services are based on a service-science legacy. Environmental forecasting systems respond to user needs to protect life and property, enhance the nation's economy and support the nation's growing need for environmental information.

Another well-known forecasting system is used for the traffic estimation and prediction. Singapore is the first country in the world that implemented the practical application of congestion pricing in 1975. Thanks to technological advances in electronic toll collection, detection, and video surveillance, Singapore upgraded its system in 1998 [Min]. In order to improve the pricing mechanism and to introduce real-time variable pricing, Singapore's Land Transport Authority, together with IBM, ran a pilot from December 2006 to April 2007, with a traffic estimation and prediction tool, which uses historical traffic data and real-time feeds with flow conditions from several sources. The objective is to be able to predict the levels of congestion over preset durations (from ten minutes up to an hour) in advance [IBM07]. Traffic forecasting systems help improve traffic conditions and reduce travel delays by facilitating the utilization of available transportation facilities.

Other forecasting systems appeared more recently to respond to new demands. Tourism forecasting systems provide forecasts of tourism demand, which are prerequisites to the decision-making process in the organizations of the private or public sector, involved in the tourism industry, helping decision-makers to plan more effectively and efficiently [PMN*03]. Stock forecasting systems [SC09] and sales forecasting systems are among useful financial forecasting systems for investors and enterprise managers to reduce logistics cost and to improve the income of enterprises.

The field of forecasting is concerned with approaches to determining what the future holds. It is also concerned with the proper presentation and use of forecasts. The terms "forecast", "prediction", "projection", and "prognosis" are typically used interchangeably. Often forecasts are of future values of a time-series. For example, the number of babies that will be born in a year, or the likely demand for compact cars. Alternatively, forecasts can be of one-off events such as the outcome of a union-management dispute or the performance of a new recruit. Forecasts can also be of distributions, such as the locations of terrorist attacks or the occurrence of heart attacks among different age cohorts. The field of forecasting includes the study and application of judgment as well as of quantitative (statistical) methods[ACGG04].

The basic functionalities a forecasting system supports are: computation, visualization and modification. The first functionality, computation of forecasts, uses specific methods (typically statistical models) to derive forecasts. The second functionality, visualization of computed forecasts, uses OLAP (online analytical processing) tools to visualize data stored in data warehouses. However, the third functionality, modification of computed forecasts during visualization, is a specific problem which is not well investigated in the data warehousing domain. In forecasting systems, source data are composed of historical data and predictive data. Unlike historical data which represent achieved facts and do not evolve over time, predictive data can be dynamic and can be updated. Experts of the domain could make some modifications to adjust computed forecasts to some specific situations. They could also make some simulations in order to visualize an objective. These modifications occur on summarized data and should be propagated to raw data (computed forecasts) and then to other summarized data. This process implies two directions of modifications. However, the work in the data warehouse domain focuses only on propagating source data modification to summarized data, which are usually considered as materialized views.

Our research targets all quantitative-measurement forecasting systems. The actual research experiments are carried on a sales forecasting system, the Anticipeo application.

1.1.2 Sales forecasting systems

A sales forecasting system, also called a business forecasting system is a forecasting system that can compute achievable sales revenue, based on historical sales data, analysis of market surveys and trends, and salespersons' estimates [SFd].

The goal is to predict the forthcoming stages of sales of any company or organization. The sales forecasting is one of the most difficult areas of management, where a lot of experience and knowledge is required for accurate prediction [eSa]. It is done through detailed analysis of all the available information regarding the different aspects of sales. This future prediction will help the company to calculate profits, to make decisions on investments, and to launch new products and services. The implementation of sales forecasting systems will help the company to improve the methods in targeting new customers, thereby giving greater sales output, and supreme customer service. It will also help to attain maximum efficiency through proper scheduling of its various activities. An effective sales forecast can have a positive impact on: financing and valuation, inventory management, order management, sales headcount capacity planning, sales revenue, visibility into sales activities [Gilo6] The sales forecasting process is managed by a point person which can be: a sales or financial analyst; a sales operations manager; a sales finance manager or similar other positions. The other intended users of the forecast can be people of other departments than sales or marketing as discussed above.

Hence the design of a sales forecasting system should consist of four phases: (1) data collection, (2) sales forecasts generation, (3) result revision, and (4) result presentation/visualization.

1.1.3 Applications of sales forecasting systems

Many business intelligence (BI) tools provide the possibility to perform simulations based on their historical data. In the BI tool survey 2012 of Passionned Group [1], 16 most used BI tools are analyzed based on 103 criteria. Those tools are widely used for reporting, dashboarding and analysis. In the simulation part, they often combine with what-if analysis including sensitive analysis and goal seeking analysis (definitions can be found in Section 2.4). However, those tools do not consider the updates/modifications of a specific result, which is a core functionality of sales forecasting systems.

There are also some proper solutions for forecasting. Besides the forecasting mod-

[1] Passionned Group is an analyst and consultancy company, based in The Netherlands, specializing in Business Intelligence and Data Integration. They offer in-depth and vendor independent research and strategic consulting. Read more, see http://www.businessintelligencetoolbox.com/

ules included in BI tools, some companies are specialized in this field, for example, ForecastPro [For], GMDH Shell [GMD], MJC^2 [MJC], etc. These solutions focus on the utilization of different forecasting methods, such as time series analysis, to get accurate forecasts. They give the possibility to modify variables to adjust projection, but not the ability of goal simulations.

The Anticipeo application is a concrete case of a sales forecasting system. It performs sales forecasting monthly for its customer companies or organizations. The customer companies or organizations provide Anticipeo with their new sales informations. After data cleansing and formatting, Anticipeo integrates the data into the system as historical data information. Then sales forecasts are generated using the pattern models predefined by statisticians of Anticipeo and chosen on the fly depending on the characteristics of the sales data. The result is first presented to the key person of the sales forecasting process, who revises the forecasting result and makes some corrections if necessary, e.g., for some planned promotions. Finally, the sales forecasting result is made accessible to the users. The presentation and the visualization of the result follows hierarchies defined by Anticipeo together with customer companies or organizations during the design phase of the application. For example, the sales can be analyzed by the purchasers' geographical distribution or by the benefit margin of the products.

The performance problem of this application resides at two levels: the sales generation and the result presentation. We consider the sales generation as a black box. The goal of this work is the optimization of the result presentation/visualization.

There are two kinds of visualizations: (1) the visualization of information pre-computed and stored, which is immediate when requested, and (2) the visualization of information calculated on the fly. Due to the quantity of manipulated information, the visualization methods should be optimized to speed up both kinds of visualizations: (1) the methods to keep the pre-computed and stored information up to date and (2) the methods to calculate information on the fly.

1.2 Problem statement and motivations

In predictive analytics systems, results are presented in the form of hierarchies to provide aggregated information at different levels of knowledge. Technically speaking, the visualization of results uses OLAP tools to visualize data stored in a data warehouse. However, a specific functionality of predictive analytics systems is the modification of computed forecasts during the visualization. This problem is not well investigated in

the data warehouses. In predictive analytics systems, source data are composed of historical data and predictive data. Unlike historical data which represent achieved facts and do not evolve over the time, predictive data could be not static and can be updated. These adjustments occur on summarized data and should be propagated to raw data (computed forecasts) and then to other summarized data. This procedure implies two directions of modifications. However, the work in data warehouse domain focuses mainly on propagating source data modification to summarized data.

To clearly define our problem, we first review how dimensions, hierarchies and the basic data schema are used by visualization tools of OLAP systems [CCS93, Inm05, KR02].

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. The values within a dimension can be further organized in a containment type hierarchy to support multiple granularities.

In the example shown in Figure 1.1, we present the dimension-hierarchy data model used in a sales forecasting system. This dimension-hierarchy data model is based on

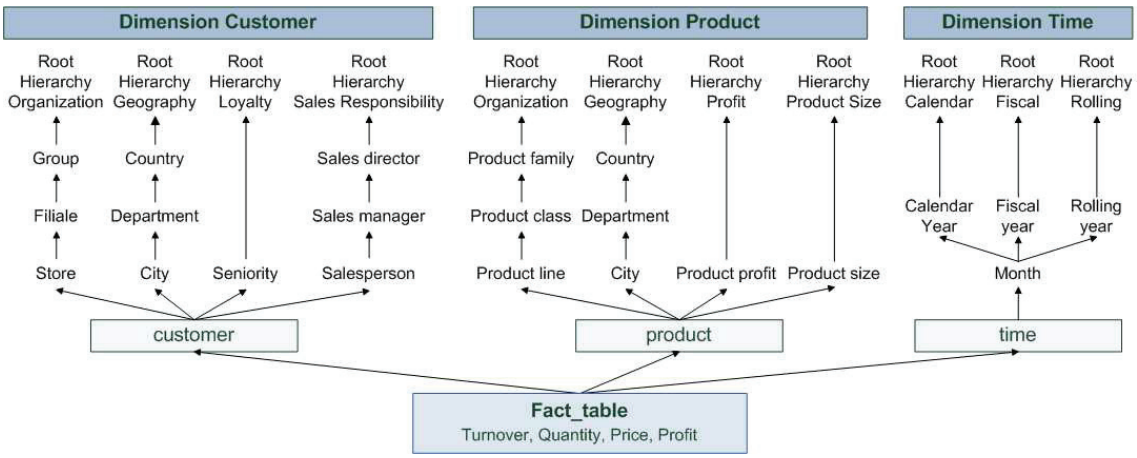


Figure 1.1: Example of a fact table with different hierarchies of three dimensions which are used to analyze raw data

one fact table and three different dimensions. The fact table contains four measures: *turnover*, *quantity*, *price*, and *profit*. We would like to mention that in the fact table of a forecasting system, there are not only "facts", which are achieving results, but also predictions. The three dimensions refer to *customer*, *product* and *time*. Each dimension has its hierarchies to describe the organization of data. The customer dimension has 4 hierarchies, the product dimension has 4 hierarchies and the time dimension has 3 hierarchies. For instance, the second hierarchy, "Hierarchy Geography", of customer di-

mension is a geographical hierarchy for analyzing sales by area of sales. Customers are grouped by city for level 1, by department for level 2 and by country for level 3. Base sales are aggregated at each level according to this geographical organization when one analyzes the sales through this hierarchy.

Regarding the visualization, OLAP systems employ materialized views to store fictive information in order to avoid extra response time. In the example of sales, fictive customers and fictive products are added to represent elements in superior hierarchy levels, such as the creation of a fictive customer for the city of Lyon, a second fictive customer for the Rhône department 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 fictive customers. Finally, all the 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.

However, the visualization in a forecasting system is not the last operation as in other OLAP systems. The systems only produce an initial version of the sales forecasts, which are then reviewed by experienced salespersons. Salespersons check these mathematically generated sales forecasts, take into account some issues not considered by the system and perform some necessary adjustments. For example, promotional offers can lead to higher turnover during the concerned period, but can also cause a decrease in turnover for the next few days because of the carried inventory. Salespersons should make some modifications for these two periods. In other cases, sales managers can also perform some modifications in order to simulate a new marketing target. They make an estimation on a level of one hierarchy and analyze the modification impacts on other levels, e.g., the detailed customer level, to decide whether the target is achievable. The fact that this update takes place on an aggregated level constitutes the major specific feature of sales forecasting systems. Compared to traditional OLAP systems in which source data are considered to be static, data in sales forecasting systems could 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. The problem we need to deal with can be generalized to how to efficiently update aggregated data through a dimension-hierarchy structure.

1.3 Contributions

At the beginning of this work, we were aware that the problem we were facing to is concerned with the performance of visualization of a sales forecasting system. However, we did not have, at our disposal, enough information to point the source of the problem. The first issue is then how to proceed in order to identify the problem.

To tackle the above mentioned problem, we define the scenario of utilization of this application. We take one kind of typical users of the application, the sales manager, because this is the only user type which has access to all functionalities of the application. We then simulate his routine work. Thus, we define four main usages of the application in our scenario.

We perform an audit of the existing application: at the hardware level and at the software level. We collected information about the performance of the hardware using the system activity report. The last one shows that the hardware is sufficient for the execution of this application. Regarding the software level, we set time line points in the application. We calculated the execution time for all functions invoked by the scenario. We filtered the functions by choosing those functions whose execution time exceeds our defined threshold. For those functions, more time line points are added to observe the main latency block(s). As there was an optimization work already carried out on the the application code, our observation result shows that nearly all the time is spent on the database part: database access and the query execution. The programming part of the application is already correctly optimized. The result leads us to a conclusion that the main latency is from the database and optimization should focus on this part. Thus, we focused on how to reduce the execution time of database queries.

In view of the system being already operational, we considered solutions which need less modifications of the actual application. We first introduced materialized views for the multidimensional visualization. We implemented the basic greedy algorithm to choose the most valuable views to materialize. The experimentation showed a significant improvement of the query execution time by using these materialized views. A deeper understanding of the application led us to acknowledge that the visualization part of system should be considered as a data warehousing and reporting system. The adoption of a star schema for this part of the system might be a better choice in terms of performance. We redesigned the database by changing the actual schema to a star schema. The modification proves that a star schema is a better solution for the visualization part.

So far, the response time of the application is significantly improved. We defined a

methodology about how to improve the performance of an application when the cause is unknown. Our methodology covers several aspects from hardware to software, from the programming to the database design.

However, there is still a situation that could not be solved by these technical solutions. It concerns the propagation of a summarized sales modification, more generally, the propagation of the impact of an aggregate modification in a data warehouse. A modification performed on an aggregate needs to be propagated to raw data and also to other aggregates computed from the same raw data. In traditional data warehouses, data are considered to be non-volatile. Data in the data warehouse are rarely over-written or deleted. Once committed, the data are static, read-only, and retained for future reporting. Nevertheless, the backward propagation is widely employed in predictive analytics systems. We need a solution to efficiently support this requirement which is not well considered in data warehouses so far.

The system, Anticipeo, already implemented a naïve solution to this problem. When the value of an aggregate is modified, all the precomputed aggregates are destroyed and then recomputed from scratch. This solution is expensive because it recomputes all the aggregates even though they are not impacted by the modification. We propose an PAM algorithm (Propagation of aggregate-based modification), which identifies the exact sets of concerned raw data and aggregates to update. The update is performed by using a temporary table of raw data impacted by the modification. We also propose an optimized version of PAM that achieves better performance when the use of additional semantics (e.g., dependencies) is possible. The PAM algorithm and its extension are proved to bring much better performance when treating a backward propagation. They significantly reduce the response time of the application when modifications take place.

Our work consists in the proposition of a methodology of different technical approaches to improve the performance of the application. We also propose an algorithm with an extended version to efficiently propagate an aggregate-based modification.

1.4 Organization of the manuscript

In this first chapter, we introduced the objective of this work and our contributions. We first stated the context by referring to three levels: forecasting systems, sales forecasting systems and applications of sales forecasting systems. We described the motivations and the research issues in this last context. Then, we have shown the two main

contributions of this work: (1) the proposition of a methodology of different technical approaches to improve the performance of the application, and (2) the proposition of an algorithm together with its extension to efficiently propagate aggregate-based modifications. Chapter 2 describes the state of the art of technologies related to this work. We present the prediction methods used in the data generation phase. We survey data warehouses, which are used as data storage in forecasting systems. We then discuss OLAP tools and view maintenance issues for data visualization. We introduce data simulation methods before relating this work to existing solutions. In Chapter 3, three algorithms are presented. We first present the current solution by explaining the principles and its limitations. We describe our proposed algorithm PAM and its extended version PAM II. We also discuss the time complexity of these two algorithms. Chapter 4 shows the experimental results performed on real data. We validate the algorithms and the estimated time complexity under two data schemas: one displaying two dimensions, and the other one based on three dimensions. We also compare the results of the three algorithms in the same scenario of tests to show the improvement achieved by our algorithms. In Chapter 5, we provide more details regarding the context of this work. We describe the application process, the user interface, the data features, the main manipulations and we state the performance problem. We propose a general methodology, considered as a guideline, which includes various technical approaches to improve the performance of the application. We conclude and present some future work in Chapter 6.

Chapter 2

State of the art

Contents

2.1 Data generation 12
2.2 Data storage 13
2.3 Data visualization 17
2.3.1 On-Line Analytical Processing (OLAP) 17
2.3.2 View maintenance 21
2.4 Data simulation 22
2.5 Synthesis 24

In forecasting systems, historical data are usually stored in relational databases to compute predictive data, which are also stored in relational databases during the prediction phase. Regarding the presentation phase, data, including raw data and aggregated data, are stored in data warehouses, i.e., multidimensional databases. The presentation and the analysis of these data employ OLAP tools. During the simulation, aggregated data and raw data are updated. View maintenance solutions are considered with OLAP tools to provide a visualization of updated data within a short latency.

In this state-of-the-art chapter, we present related works in relation with data processing in forecasting systems. We first introduce some notions of forecasting systems and the main methods to generate forecasting data. We present approaches to data storage. Then we introduce the visualization techniques and the optimizations of data visualization: OLAP and view maintenance issues in relational databases and data cubes. We describe similar solutions to forecasting systems, such as the simulation in BI, i.e., what-if analysis. We point out the differences between the existing approaches and ours and highlight the features of our work.

2.1 Data generation

A forecast is a prediction of what might happen in the future. It is based on past information and an analysis of expected environment conditions. For example, an earthquake in the southwest of the United States in the next 15 days is a forecast issue from an environmental forecasting system. A saturation of 3 hours for the morning of the July 14th 2012 near Valence in France is a forecast for the traffic.

Forecasting is a collection of methods for generating forecasts. The steps of forecasting can be summarized as: to determine the use of the forecast, to select the items to be forecasted, to determine the time horizon of the forecast, data collection, data reduction, to select the forecasting models, to make the forecast and forecast evaluation.

Forecasting is relevant to many activities [Kus99]. Governments need to forecast unemployment, interest rates, expected revenues from income taxes to formulate policies. Companies need to forecast demand, sales, consumer preferences in strategic planning. Banks/investors/financial analysts need to forecast financial returns, risk or volatility, market timing. University administrators need to forecast enrollments to plan facilities and faculty recruitment. Retail stores need to forecast demand to control inventory levels, hire employees and provide training. Sport organizations need to project sports performance, crowd figures, club gear sales, revenues, etc., in the coming season.

There is a number of forecasting methods. Figure 2.1 depicts the methodology tree for forecasting [Arm00]. It classifies all the possible types of forecasting methods into categories and shows how they relate to each other. Dotted lines represent possible relationships. Forecasting methods can be classified as either subjective or objective [ACGG04] [MWH98b]. Subjective (judgmental) methods include expert opinions, and the intentions and expectations of customers, for example, the Delphi method [RW99]. They are widely used for important forecasts. They are also used in situations where there is no history to apply statistical methods. Objective (statistical) methods include extrapolation (such as moving averages [Mov], linear regression against time, or exponential smoothing [Nat11]) and econometric methods [JD07] (typically using regression techniques [Hof93] to estimate the effects of causal variables). In [AG05], J.S. Armstrong and K.C. Green conclude that in situations where there are sufficient data, we should use quantitative methods including extrapolation, quantitative analogies, rule-based forecasting, and causal methods. Otherwise, we should use methods that structure judgment including surveys of intentions and expectations, judgmental bootstrapping, structured analogies, and simulated interaction. Managers' domain knowledge should be incorporated into statistical forecasts. To improve forecasting accuracy, we

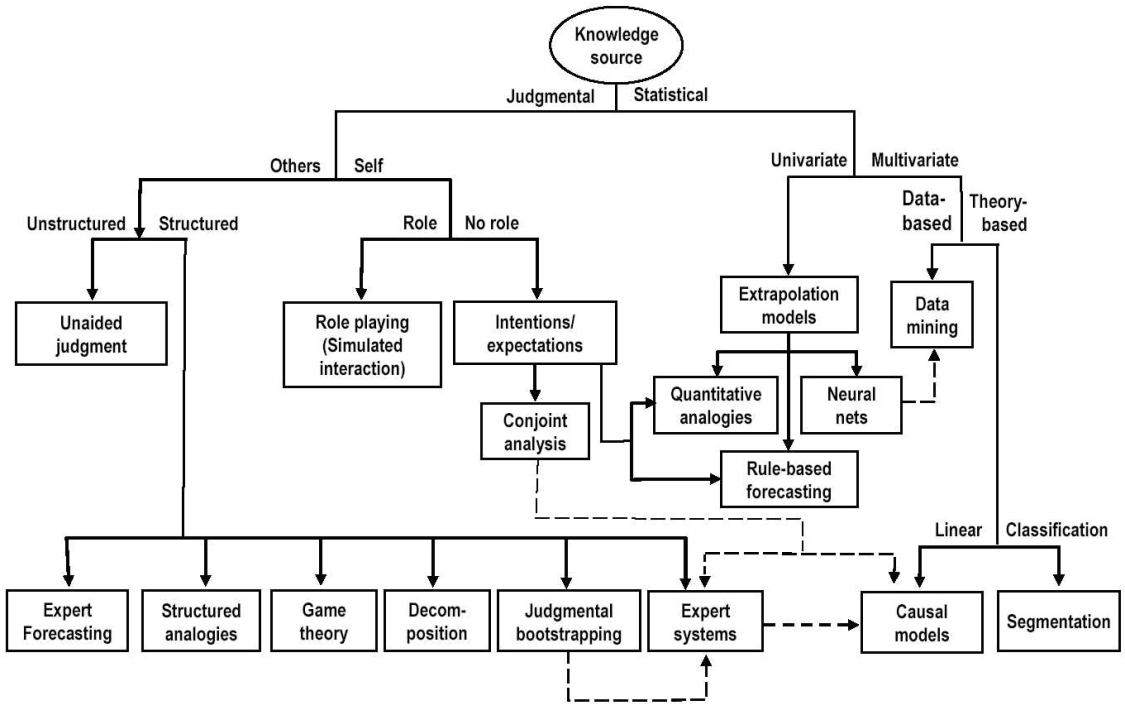


Figure 2.1: Methodology tree for forecasting [Armoo]

can combine forecasts derived from methods that differ substantially and draw from different sources of information. When feasible, five or more methods can be used, including Delphi and prediction markets. The most common approach in business is judgmentally adjusted statistical forecasting.

2.2 Data storage

Databases and database theory have been around for a long time. Early renditions of databases centered around a single database serving every purpose known to the information processing community, from transaction to batch processing to analytical processing. In most cases, the primary focus of the early database systems was operational, usually transactional, processing. More and more, people are interested in getting *information* from the raw *data* in order to improve their *knowledge* (see [Ack89] for the differences between data, information, knowledge and wisdom). In recent years, a more sophisticated notion of the database has emerged. The modern way to build systems is to separate the operational from the informational or analytical processing and data. Here arise data warehousing and decision support systems (DSS). Since the 90s, data warehousing technologies have been successfully deployed in many industries [CD97]: manufacturing (for order shipment and customer support), retail (for user profiling and

inventory management), financial services (for claims analysis, risk analysis, credit card analysis, and fraud detection), transportation (for fleet management), telecommunications (for call analysis and fraud detection), utilities (for power usage analysis), and healthcare (for outcomes analysis).

A data warehouse is defined as: "a data warehouse is a subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management's decisions" [Inm05]. It can also be defined as follows: "A data warehouse is a copy of transaction data specifically structured for query and analysis" [KR02]. A data warehouse is not a decision support system, it is an organized collection of large amounts of structured data [Pow02]. A data warehouse contains granular corporate data. Data in the data warehouse can be used for many different purposes. Typically, the data warehouse is maintained separately from the organization's operational databases. To successfully build a data warehouse, some requirements have to be fulfilled: (1) it must make an organization's information easily accessible; (2) it must present the organization's information consistently; (3) it must be adaptive and resilient to change; (4) it must be a secure bastion that protects our information assets; (5) it must serve as the foundation for improved decision making; and (6) the business community must accept the data warehouse if it is to be deemed successful.

There are four levels of data in the architectural environment: the operational level, the atomic (or the data warehouse) level, the departmental (or the data mart) level, and the individual level [Inm05]. These different levels of data are the basis of a larger architecture called the Corporate Information Factory (CIF) [IIS01]. The operational level of data holds application-oriented primitive data only and primarily serves the high-performance transaction-processing community. The data warehouse level of data holds integrated, historical primitive data that cannot be updated. In addition, some derived data is found there. The departmental or data mart level of data contains derived data almost exclusively. The departmental or data mart level of data is shaped by end-user requirements into a form specifically suited to the needs of the department. Finally, the individual level of data is where much heuristic analysis is performed.

The typical architecture of a data warehouse [CD97] (shown in Figure 2.2) is designed by respecting these levels of data. It includes tools for extracting data from multiple operational databases and external sources; for cleaning, transforming and integrating this data; for loading data into the data warehouse; and for periodically refreshing the warehouse to reflect updates at the sources and to purge data from the warehouse onto slower archival storage. In addition to the main warehouse, there may be several departmental data marts. Data in the warehouse and data marts are stored and managed

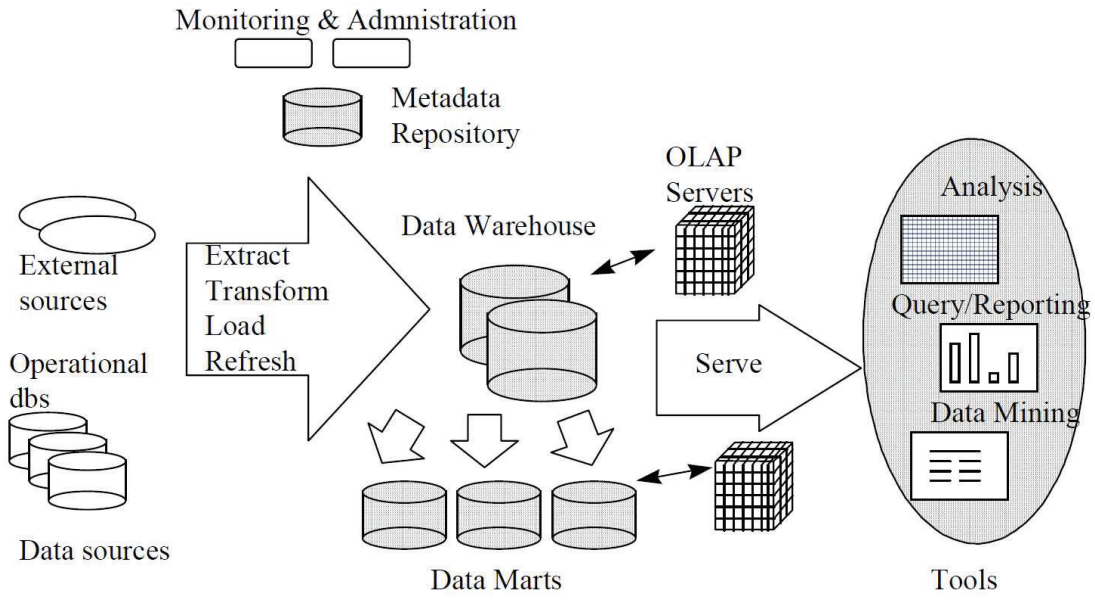


Figure 2.2: Data Warehousing Architecture [CD97]

by one or more warehouse servers, which present multidimensional views of data to a variety of front end tools: query tools, report writers, analysis tools, and data mining tools. Finally, there is a repository for storing and managing metadata, and tools for monitoring and administering the warehousing system.

Back End. Data warehousing systems use a variety of data extraction and cleaning tools, and load and refresh utilities for populating warehouses.

Data Cleansing: Since a data warehouse is used for decision making, it is important that the data in the warehouse are correct. However, since large volumes of data from multiple sources are involved, there is a high probability of errors and anomalies in the data. Therefore, it is necessary to detect data anomalies and correct them. Some examples of data cleansing are: inconsistent field lengths, inconsistent descriptions, inconsistent value assignments, missing entries and violation of integrity constraints.

Load: After extracting, cleaning and transforming, data must be loaded into the warehouse. Additional preprocessing may still be required: checking integrity constraints; sorting; summarization, aggregation and other computation to build the derived tables stored in the warehouse; building indices and other access paths; and partitioning to multiple target storage areas. Typically, batch load utilities are used for this purpose. In addition to populating the warehouse, a load utility must allow the system administrator to monitor status, to cancel, suspend and resume a load, and to restart after failure with no loss of data integrity.

Refresh: Refreshing a warehouse consists in propagating updates on source data to cor-

respondingly update the raw data and derived data stored in the warehouse. There are two sets of issues to consider: when to refresh, and how to refresh. Usually, the warehouse is refreshed periodically (e.g., daily or weekly). Only if some OLAP queries need current data (e.g., up to the minute stock quotes), it is necessary to propagate every update.

Conceptual Model. A popular conceptual model that influences the front-end tools, database design, and the query engines for OLAP is the *multidimensional* view of data in the warehouse. In a multidimensional data model, there is a set of numeric *measures* that are the objects of analysis. Examples of such measures are sales, budget, revenue, inventory, ROI (return on investment). Each of the numeric measures depends on a set of *dimensions*, which provide the context for the measure. For example, the dimensions associated with a sale amount can be the customer name, product name, the date when the sale was performed and the amount. The dimensions together are assumed to *uniquely* determine the measure. Thus, the multidimensional data model consider, a measure as a value in the multidimensional space of dimensions. Each dimension is described by a set of attributes. For example, the Product dimension may consist of four attributes: the category and the industry of the product, the year of its introduction, and the average profit margin. The attributes of a dimension may be related via a hierarchy of relationships. In the above example, a product name "LG 47LM7600" is related to both the category attribute "TV" and the industry attribute "Electronics".

Different architectural alternatives exist for the implementation of a data warehouse. Many organizations want to implement an integrated enterprise warehouse that collects information about all subjects (e.g., customers, products, sales, assets, personnel) spanning the whole organization. Building an enterprise warehouse is a long and complex process, requiring extensive business modeling and may take many years to accomplish. Some organizations are settling for data marts instead, which are departmental subsets focused on selected subjects (e.g., a marketing data mart may include customer, product and sales information). These data marts enable faster roll out, since they do not require enterprise-wide consensus, but they may lead to complex integration problems in the long run. Data warehouses and data marts differ in scope only. This means that they are built using the same methods and procedures, so the process is the same, while only their intended scope varies.

Front End. Front end tools implement typical analytical operations such as rollup (increasing the level of aggregation) and drill-down (decreasing the level of aggregation or increasing detail) along one or more dimension hierarchies, slice_and_dice (selection and projection), and pivot (re-orienting the multidimensional view of data). There are a

variety of data mining tools that are often used as front end tools to data warehouses, such as Microsoft Excel Spreadsheet [spr] (still the most compelling front-end application), MicroStragery [mica], Business Objects [bo], Cognos [cog], SAS [sas], etc.

Designing and rolling out a data warehouse is a complex process. It consists in the following activities [KR02].

- Define the architecture, do capacity planning, and select the storage servers, database and OLAP servers, and tools.
- Integrate the servers, storage, and client tools.
- Design the warehouse schema and views.
- Define the physical warehouse organization, data placement, partitioning, and access methods.
- Connect the sources using gateways, ODBC drivers, or other wrappers.
- Design and implement scripts for data extraction, cleaning, transformation, load, and refresh.
- Populate the repository with the schema and view definitions, scripts, and other metadata.
- Design and implement end-user applications.
- Roll out the warehouse and applications.

2.3 Data visualization

2.3.1 On-Line Analytical Processing (OLAP)

Analytical processing refers to using the computer to produce an analysis for management decision, usually involving trend analysis, drill-down analysis, demographic analysis, profiling, and so forth [Pow10].

The Relational Model is a foundation for relational database management system (DBMS) design, that provides interesting facilities for storage, update and retrieval of data. However, most notably lacking has been the ability to consolidate, view, and analyze data according to multiple dimensions, in ways that make sense to one or more specific enterprise analysts at any given point in time. This requirement is called ”multidimensional data analysis“. A more generic name for this type of functionality is OLAP [CD97], wherein multidimensional data analysis is one of its characteristics.

It is important to distinguish the capabilities of a data warehouse from those of an OLAP system. OLAP is a technology, while the data warehouse is an architectural infrastructure, and a symbiotic relationship exists between the two [OLA97]. In contrast to a data warehouse, which is usually based on relational technology, OLAP uses a multidimensional view of aggregate data to provide quick access to strategic information for further analysis. In the normal case, the data warehouse serves as a foundation for the data that will flow into the multidimensional DBMS, feeding selected subsets of the detailed data into the multidimensional DBMS where it is summarized and otherwise aggregated.

In [CCS93], Codd et al. describe OLAP characteristics:

Dynamic Data Analysis: Once data has been captured in a database, the analytical process of synthesizing the data into information can start. Dynamic data analysis can provide an understanding of the changes occurring within a business enterprise, and may be used to identify candidate solutions to specific business challenges as they are uncovered, and to facilitate the development of future strategic and tactical formulae.

Four Enterprise Data Models: The used data models fall into four categories: the categorical model, the exegetical model, the contemplative model, and the formulaic model. The categorical model is employed in static data analysis to describe what has gone on before by comparing historical values or behaviors which have typically been stored in the enterprise database. Moving along the continuum, the exegetical model reflects what has previously occurred to bring about the state which reflected by the categorical model. The third model, the contemplative model, indicates what outcomes might result from the introduction of a specific set of parameters or variances across one or more dimensions of the data model. This type of analysis is significantly more dynamic. The fourth data model, the formulaic model, is the most dynamic and requires the highest degree of user interaction and associated variable data consolidation. This data model indicates which values or behaviors across multiple dimensions must be introduced into the model to influence a specific outcome.

Common Enterprise Data: The data required for Online Transaction Processing (OLTP) [OLT] systems is the same data which is required for OLAP. The nature of the transactions differs, as does the need for the data to be strictly up-to-date, but both types of processing take place against the same data stores.

Synergistic Implementation: During the years, the requirement for OLAP has been realized by relational DBMS and the concomitant end-user tools. Only in the recent years the requirement for OLAP has become evident and understood. Since the end-user has become very comfortable with the interface to the spreadsheet, the approach was to add

the function to the spreadsheet product.

OLAP server technology is the key to high performance analytical use of large databases. Its added intelligence about the structure and organization of the data, as compared to flat, detailed relational tables, makes an OLAP server more responsive to end user requests, while also eliminating SQL-style queries. An OLAP server may physically stage the processed multidimensional information to deliver consistent and rapid response times to end users, or it may populate its data structures in real-time from relational or other databases, or it may offer a choice of both. Users of data warehouses work in a graphical environment and data are usually presented to them as a multidimensional "data cube" whose 2-D, 3-D, or even higher-dimensional sub cubes they explore trying to discover interesting information [HRU96]. Each cell of the data cube is a view consisting of an aggregation of interest, like total sales. The values of many of these cells are dependent on the values of other cells in the data cube. The values in each cell of this data cube are some "measures" of interest.

The cube data can be divided into three different types - meta-data, detail data and aggregate data. No matter what storage is used, the meta-data will always be stored on the OLAP server but storage of the detail data and aggregate data will depend on the specified storage mode [Ars]. A partition can use one of three basic storage modes: multidimensional OLAP (MOLAP), relational OLAP (ROLAP) and hybrid OLAP (HOLAP). The storage mode of a partition affects the query and processing performance, storage requirements, and storage locations of the partition and its parent measure group and cube [Micb]. The choice of storage mode also affects processing choices.

MOLAP. The MOLAP storage mode causes the aggregations of the partition and a copy of its source data to be stored in a multidimensional structure in the OLAP server. After processing, once the data from the underlying relational database is retrieved, there is no connection to the relational data stores. So if there are any subsequent changes in the relational data after processing, then they will not reflect in the cube unless the cube is reprocessed and hence the MOLAP is called off-line data-set mode. Since both the detail and aggregate data are stored locally on the OLAP server, the MOLAP storage mode is very efficient and provides the fastest query performance.

ROLAP. The ROLAP storage mode causes the aggregations of the partition to be stored in indexed views in the relational database that was specified in the partition's data source. In comparison with MOLAP, ROLAP does not pull data from the underlying relational database source to the OLAP server but rather both cube detail data and aggregation stay at the relational database source. In order to store the calculated aggregation, the database server creates additional database objects (indexed views). In other

words, the ROLAP mode does not copy the detail data to the OLAP server, and when a query result cannot be obtained from the query cache the created indexed views are accessed to provide the results.

HOLAP. The HOLAP storage mode combines attributes of both MOLAP and ROLAP. Like in the case of MOLAP, in HOLAP the aggregations of the partition are stored in a multidimensional structure in the OLAP server. HOLAP does not store a copy of the source data. For queries that access only summary data in the aggregations of a partition, HOLAP is the equivalent of MOLAP. Queries that access source data, for example, if one wants to drill down to an atomic cube cell for which there is no aggregation data, data must be retrieved from the relational database and will not be as fast as they would be if the source data were stored in the MOLAP structure. With HOLAP storage mode, users will typically experience substantial differences in query times depending upon whether the query can be resolved from cache or aggregations versus from the source data itself.

The multidimensional data model described above is implemented directly by MOLAP servers. However, when a relational ROLAP server is used, the multidimensional model and its operations have to be mapped into relations and SQL queries. Entity Relationship (ER) diagrams and normalization techniques are popularly used for database design in OLTP environments. However, the database designs recommended by ER diagrams are inappropriate for decision support systems where efficiency in querying and in loading data (including incremental loads) are important.

Most data warehouses use a star schema to represent the multidimensional data model. The database consists of a single fact table and a single table for each dimension. Each tuple in the fact table consists of a pointer (foreign key - often uses a generated key for efficiency) to each of the dimensions that provide its multidimensional coordinates, and stores the numeric measures for those coordinates. Each dimension table consists of columns that correspond to attributes of the dimension. The hierarchies are contained in the individual dimension tables. No additional tables are needed to hold hierarchical information. The traditional ER model has an even and balanced style of entities and complex relationships among entities, the dimensional model is very asymmetric [BHS*98].

Sometimes, the dimension tables have the hierarchies broken out into separate tables. This is a more normalized structure, but leads to more difficult queries and slower response times. This structure increases the number of joins and can slow queries. Since the purpose of an OLAP system is to improve response time of decision querying,

snowflaking is usually not productive. Some people try to normalize the dimension tables to save space. However, in the overall scheme of the data warehouse, the dimension tables usually only account for about 1% of the total storage [Utl02]. Therefore, any space savings from normalizing, or snowflaking, are negligible. In [AV98], Adamson et al. present concrete solutions for different target business.

2.3.2 View maintenance

Materialized views have been recognized as effective objects in databases to improve query evaluation. In [GM95], Gupta and Mumick have described materialized views, their applications, and the problems and techniques for their maintenance. A view is a derived relation defined in terms of base (stored) relations. A view can be materialized by storing the answer to the underlying query in the database. Index structures can be built on the materialized view. A materialized view is thus like a cache - a copy of the data that can be accessed quickly. Just as a cache gets dirty when the data from which it is copied is updated, a materialized view gets dirty whenever the underlying base relations are modified. The process of updating a materialized view in response to changes to the underlying data is called view maintenance. In most cases it is wasteful to maintain a view by recomputing it from scratch. Often it is cheaper to use the heuristic of inertia (only a part of the view changes in response to changes in the base relations) and thus compute only the changes in the view to update its materialization. Algorithms that compute changes to a view in response to changes to the base relations are called incremental view maintenance [LSK01].

Materialized views have different applications. In data warehousing, materialized views can be used to precompute and store aggregated data such as sum of sales. Materialized views in these environments are typically referred to as summaries since they store summarized data. They can also be used to precompute joins with or without aggregations, such as the number of babies born between 2000 and 2010 by country. So a materialized view is used to eliminate overhead associated with expensive joins or aggregations for a large or important class of queries.

The materialized view maintenance problem has been widely discussed in data warehousing. Solutions about how to efficiently update materialized views in relational databases are introduced in this field. The combination of ”materialized view log“ and ”fast refresh“ applied in Oracle [Ora12] shows a good performance in certain contexts. Approaches to view maintenance in data warehouses are concerned with different directions. In [ZLE07], the authors propose “lazy” maintenance of materialized views. In

order to reduce the view maintenance cost, this paper suggests to postpone maintenance of a view until the system has free cycles or the view is referenced by a query rather than update materialized views when source data change. [MQM97, LL06] propose solutions of incremental view maintenance. These solutions create differential files, which keep the differences of the relevant tuples and calculate new views based on these differential files instead of calculating complete materialized views. [NLR98, CLR04] discuss multi-view maintenance and their consistency problems over distributed data sources. There exist many others (see the research-oriented bibliography on Data Warehouse and OLAP[1] and Jacob Hammer's web bibliography[2]). Some approaches dealing with view maintenance in OLAP were also proposed. Some of them focus on the evolution of the multidimensional structure [BMBT03, HMV99]. They discuss materialized views recomputation regarding changes to the axes of analysis, or dimensions. In [Bel02], the issues related to the evolution and maintenance of data warehousing systems, when underlying data sources change their schema capabilities were addressed. It considers the problem of invalidation of views due to schema changes arising on the data sources. Some approaches adapt materialized views after their redefinition according to user requirement changing over time [GMRR01, MD96]. They identify guidelines for users and database administrators that can be used to facilitate efficient view adaptation. Other works focus on the optimization of OLAP operators such as pivot and unpivot [CR05]. They propose rewriting rules, combination rules and propagation rules for such operators and also design a novel view maintenance framework for applying these rules to obtain an efficient maintenance plan.

However, the main context of these approaches is the propagation of updates occurring on sources to materialized views. In our context, the updates take place on summarized data, in other words, directly on materialized views. We need to propagate the modification to raw data and also to other materialized views. To the best of our knowledge, the problem of updating summaries and computing the effect on raw data has not been investigated so far.

2.4 Data simulation

In order to be able to evaluate beforehand the impact of a strategical or tactical move, decision makers need reliable previsional systems. What-if analysis partially satisfies this need by enabling users to simulate and inspect the behavior of a complex system

[1] http://lemire.me/OLAP/
[2] http://www.cise.ufl.edu/~jhammer/online-bib.htm

(i.e., the enterprise business or a part of it) under some given hypotheses, called scenarios [GRP06]. More pragmatically, what-if analysis measures how changes in a set of independent variables impact a set of dependent variables with reference to a given simulation model [Phi88]; such a model is a simplified representation of the business, tuned according to the historical enterprise data. The Microsoft Excel 2010 Help Document defines what-if analysis as a "process of changing the values in cells to see how those changes affect the outcome of formulas on the worksheet. For example, varying the interest rate that is used in an amortization table to determine the amount of the payments" [Win11]. The simplest type of what-if analysis is manually changing a value in a cell that is used in a formula to see the result. In [PVSV07], Papastefanatos et al. describe a general mechanism for performing what-if analysis for potential changes of data source configurations.

Pannell [Pan97] identifies the uses of the what-if analysis in decision making, communication, understanding systems and in model development. Based on his discussion, a model-driven DSS with appropriate analysis should help in 1) testing the robustness of an optimal solution, 2) identifying critical values, thresholds or break-even values where the optimal strategy changes, 3) identifying sensitive or important variables, 4) investigating sub-optimal solutions, 5) developing flexible recommendations which depend on circumstances, 6) comparing the values of simple and complex decision strategies, and 7) assessing the "riskiness" of a strategy or scenario.

Some experts use the terms sensitivity analysis and what-if analysis interchangeably [Pow04], even if in the decision support system literature and in common discourse, there is no agreement about the difference between what-if analysis and sensitivity analysis (see [Ale89] for more information about sensitivity analysis). There is an important difference between what-if analysis and simple forecasting. In fact, while forecasting is normally carried out by extrapolating trends out of the historical series stored in information systems, what-if analysis requires to simulate complex phenomena whose effects cannot be simply determined as a projection of past data, which in turn requires to build a simulation model capable of reproducing - with satisfactory approximation - the real behavior of the business.

As a part of what-if analysis, goal seeking analysis represents the ability to calculate a formula backward to obtain a desired input [OM10]. It is the process of finding the correct input when only the output is known [Goa]. For example, goal seeking helps a manager who wishes to determine what change would have to take place in the value of a specified variable in a specified time period to achieve a specified value for another variable.

There is a fundamental difference between our issue and what-if analysis. As defined above, what-if analysis changes variables' values to inspect the impacts. When variables' values are changed, a new calculation is required using the simulation model to evaluate the impact. In our work, decision makers perform changes in the forecasting results produced by simulation models. Nevertheless, propagating the modification does not require a new calculation with simulation models. The impact is directly evaluated at different levels of hierarchies in different dimensions regarding some predefined rules. As the forecasting results are stocked as materialized views, our issue is rather an issue of data consistency, in other words, maintenance of materialized views.

2.5 Synthesis

The motivation of our work comes from forecasting systems or, more generally, predictive analytics systems. In these systems, decision makers need to perform some goal simulations to validate or modify their strategical or tactical moves regarding the forecasting results. This work is not related to what-if analysis because the objective is not to modify values of parameters so as to project new forecasts, but to directly modify the results so as to inspect the impact in the whole hierarchies and dimensions. Among existing forecasting and simulation solutions, they rarely provide the possibility to modify directly the value of a forecast, which shows the needs of simulating aggregate value modification in a data warehouse environment.

As the underlying environment of predictive analytics systems is usually presented by data warehousing including OLAP, our research issue is on how to propagate an aggregate-based modification to all data, including raw data and summarized data in a data warehouse. Technically, the problem that we deal with can be related to the maintenance of materialized views. A lot of works in the data warehousing field are devoted to this problem. But their common point is that they consider only modifications taking place in source data. They do not take into account modifications in cells of a data cube. Moreover, to the best of our knowledge, no work has discussed how to distribute modifications on aggregated data over raw data.

In our work, we propose incremental view maintenance algorithms. Existing incremental view maintenance solutions often focus on insertion and deletion of tuples on raw data. Updates are considered as a sequence of a deletion and an insertion. However, in our specific context, data are only updated by simply changing their values. The combination of a deletion and an insertion costs too much to manage the physical storage and to maintain indexes on tables and materialized views.

Chapter 3

Aggregate-based modification: impact management

Contents

3.1 Notations and definitions 26

3.2 Current solution: principles and limitations 28

3.3 Proposed algorithm 29

3.3.1 PAM Algorithm 30

3.3.1.1 Description of the algorithm 30

3.3.1.2 Time complexity 33

3.3.2 PAM II Algorithm 34

3.3.2.1 Description of the algorithm 34

3.3.2.2 Time complexity 35

3.3.3 Other aggregate functions 36

In this chapter, we will present three algorithms used to propagate the impact of aggregate modification to raw tuples and to all other aggregates of all hierarchies of all dimensions in a data warehouse. Before presenting the algorithms, we introduce some notations and definitions employed in their description. The first algorithm that we present is a naïve solution used in the Anticipeo application so far. We discuss the principles of this algorithm and its limitations. We describe our algorithm, PAM (Propagation of Aggregate-based Modification) and we show its complexity. We also present an extended version of the PAM algorithm, which is designed to improve the performance of the PAM algorithm.

3.1 Notations and definitions

In the presentation of the algorithms, we use some notations and predicates. In this part, we first clarify some notions and introduce some definitions used in our context. In the following sections:

- $\mathbb{T}$ stands for all raw tuples
- $\mathbb{A}$ stands for all the aggregates in the materialized view
- α is a distributive aggregate function (e.g., SUM)
- A=α_T is an aggregate of A that employs the aggregate function α on a set of tuples $T \subseteq \mathbb{T}$

Definitions:
*Definition 3.1 (**tuple dependency**).* Given an aggregate A=α_T and a set of raw tuples T', A is said to depend on T' iff $T \cap T' \neq \varnothing$.

*Definition 3.2 (**tuple dependency predicate**).* **dep(A, T')** returns true if the aggregate A depends on the set of raw tuples T', false otherwise.

*Definition 3.3 (**impacted tuple**).* A tuple t is said to be impacted by the modification performed on the aggregate A=α_T iff A depends on the tuple t.

*Definition 3.4 (**aggregate dependency**).* An aggregate A=α_T is said to depend on the aggregate A'=$\alpha_{T'}$ iff A depends on T'.

*Definition 3.5 (**impacted aggregate**).* An aggregate A=α_T is said to be impacted by the modification on the aggregate A'=$\alpha_{T'}$ iff A depends on A'.

*Definition 3.6 (**aggregate impact predicate**).* **imp(A, A')** returns true iff the aggregate A is impacted by the modification of the aggregate A', false otherwise.

Let us show on an example how an aggregation-level modification can impact other data by using these definitions and predicates (see Figure 3.1).

In this example and for the sake of simplicity, we consider only two hierarchies respectively for the customer dimension and the product dimension. In the fact table, we

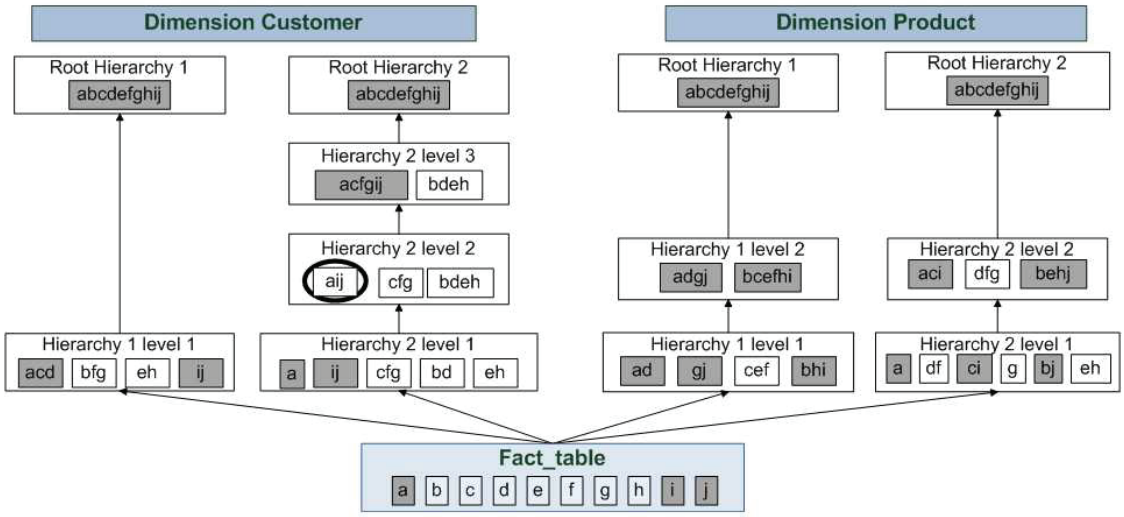


Figure 3.1: Example of data modification on an aggregated level in a dimension-hierarchy structure and the impact of the modification

consider only 10 raw tuples: named from *a* to *j*. Aggregates at superior hierarchy levels are presented by rectangles including the raw tuples which generate corresponding aggregates. For instance, the circled rectangle of level 2 of hierarchy 2 in the customer dimension represents the aggregate $\alpha_{\{a,i,j\}}$. This presentation denotes that the aggregate $\alpha_{\{a,i,j\}}$ depends on the set of raw tuples {a,i,j}. In the specific case of a sales forecasting system, the result val($\alpha_{\{a,i,j\}}$) of the aggregate $\alpha_{\{a,i,j\}}$ is the sum of the base sales *a*, *i* and *j*. Other aggregates are presented in the same manner. The root rectangles of every hierarchy stand for all the sales. The results of different root rectangles are the same because they stand for all the sales.

Figure 3.1 depicts the underlying data structure when the system presents the prediction result to sales managers. Sales managers analyze the sales and then decide to modify the value of an aggregate, for example the aggregate $\alpha_{\{a,i,j\}}$ (i.e., to evaluate beforehand the impact of a strategical or tactical move). As the aggregate $\alpha_{\{a,i,j\}}$ is generated from *a*, *i* and *j*, if its value is modified, the results of the three tuples should be updated afterwards. Meanwhile, these three tuples are also the raw tuples that are involved in the calculation of other aggregates in hierarchies of all dimensions, e.g., the aggregate $\alpha_{\{a,c,d\}}$ of level 1 of hierarchy 1 in the customer dimension and the aggregate $\alpha_{\{b,e,h,j\}}$ of level 2 of hierarchy 2 in the product dimension. Hence, all the aggregates containing any of these three tuples in their composition should be updated as well. These aggregates impacted by the modification on the aggregate $\alpha_{\{a,i,j\}}$ in this example are darkened in Figure 3.1.

3.2 Current solution: principles and limitations

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 already well defined. The current solution uses these methods to calculate new results. The steps of the current solution which consists in recomputing everything are the following:

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

To illustrate this process, consider the example shown in Figure 3.1. We assume the actual result of the aggregate $\alpha_{\{a,i,j\}}$ is 500 000 euros. The sales manager has a new marketing plan, estimated to achieve 600 000 euros sales. The result of $\alpha_{\{a,i,j\}}$ is updated, and the sales manager needs to evaluate the impact on other aggregates in order to determine whether this new plan is achievable in different angles. This example introduces two different values of the aggregate $\alpha_{\{a,i,j\}}$. We denote by val($\alpha_{\{a,i,j\}}$) the value before the modification and by val'($\alpha_{\{a,i,j\}}$) the value after the modification. In this example, val($\alpha_{\{a,i,j\}}$) = 500 000 and val'($\alpha_{\{a,i,j\}}$) = 600 000. Assume that the distribution of sales on raw tuples a, i and j is 100 000 euros, 200 000 euros and 200 000 euros, respectively. We then denote by val(t) the value of the attribute considered in the computation for a tuple. We have, in this example, val(a) = 100 000, val(i) = 200 000 and val(j) = 200 000. Here, we see that each of the raw tuples does not contribute equally to the result of the aggregate. We should consider the contribution of each raw tuple while calculating their new results.

*Definition 3.7 (**tuple weight**).* A tuple weight is a measure to evaluate the contribution of a tuple to the calculation of an aggregate. It does not depend neither on the value of the raw tuple nor on the value of the aggregate. A tuple weight could be defined as a constant or as a variable relating to some criteria. In this case, where the result of an aggregate is the simple sum of raw tuples, the tuple weight is defined as a variable and it can be determined as follows:

$weight(t, A) = \frac{val(t)}{val(A)}$,

where t is a tuple and A is an aggregate depending on t.

By considering our example, we have:

$$weight(a, \alpha_{\{a,i,j\}}) = \frac{val(a)}{val(\alpha_{\{a,i,j\}})} = \frac{100\ 000}{500\ 000} = 0.2$$

$$weight(i, \alpha_{\{a,i,j\}}) = \frac{val(i)}{val(\alpha_{\{a,i,j\}})} = \frac{200\ 000}{500\ 000} = 0.4$$

$$weight(j, \alpha_{\{a,i,j\}}) = \frac{val(j)}{val(\alpha_{\{a,i,j\}})} = \frac{200\ 000}{500\ 000} = 0.4$$

Please notice that the total weight of all the raw tuples composing an aggregate should be equal to 1. Then, the propagation of the modification using the current solution is processed as follows:

Step 1: calculation of new values of raw tuples

We aim to compute new values of each raw tuple impacted by the modification of the aggregate. Then, the formula to calculate the new result for a tuple t is:

$$\forall t \in T : val'(t) = val(t) + (val'(\alpha_T) - val(\alpha_T)) * weight(t, \alpha_T)$$

In our example, the new values for T={a,i,j} are:

$$val'(a) = 100\ 000 + (600\ 000 - 500\ 000) * 0.2 = 120\ 000$$

$$val'(i) = 200\ 000 + (600\ 000 - 500\ 000) * 0.4 = 240\ 000$$

$$val'(j) = 200\ 000 + (600\ 000 - 500\ 000) * 0.4 = 240\ 000$$

Step 2: recalculation of aggregated information

The second step consists in recomputing the aggregates of all levels for all hierarchies of all dimensions. We follow the same process as when the aggregates were previously created for the hierarchies, i.e., a new execution of the definition of the materialized views containing aggregates. For instance, the aggregate $\alpha_{\{a,c,d\}}$ is an aggregate of the raw tuples a, c and d; so its new result is calculated by summing the sales of a, c and d with their updated values.

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

3.3 Proposed algorithm

The current solution advocates the calculation of all the aggregates of all the hierarchies. However, this solution performs some useless work. If we look closely at the recomputed aggregates in Figure 3.1, only the dark ones are concerned with the modification and need to be updated, that is, **19** aggregates out of **33**. 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 raw tuples, we can identify the exact aggregates to modify and hence

avoid useless work.

Another drawback of the current solution is its heavy recomputing procedure. Operations of removing and adding aggregates ask for 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.

3.3.1 PAM Algorithm

In this section, we explain how the PAM algorithm (Propagation of Aggregate-based Modification) [FLHD12] identifies and updates the relevant sets of aggregates. We also present its utilization in more complex data schema with multiple hierarchies. The time complexity is also calculated to show its scalability.

3.3.1.1 Description of the algorithm

A coarse-grained description of our algorithm is composed of the following steps:

1. retrieval of participating raw tuples to the modified aggregate;
   creation of a temporary table for the raw tuples to be updated;
   and calculation of the differences for raw tuples resulting from the old values and the new ones
2. update of impacted raw tuples
3. identification of impacted aggregates;
   and update of impacted aggregates based on previously calculated differences of raw tuples

In the following, δ of a tuple or an aggregate stands for the difference of the value of a tuple or the result of an aggregate before and after modification.

The algorithm for the update propagation through a dimension-hierarchy architecture is shown in Table 3.1. The description of this algorithm uses the notations defined in Section 3.1. Line 1 to line 4 identify the raw tuples involved in the modification and calculate their differences. Line 5 allows to update these raw tuples. Line 6 to line 10 identify impacted aggregates and perform the update.

Let us take the previous example (Section 3.2) to illustrate the approach. A sales manager changes the sales of the aggregate $\alpha_{\{a,i,j\}}$ from 500 000 euros to 600 000 euros. Once the modification is confirmed, the system will proceed using the algorithm in Table 3.1.

Table 3.1: Algorithm PAM for the update propagation of an aggregate modification

Algorithm PAM (Propagation of Aggregate-based Modification)

Input: Schema S, aggregate A=α_T, the current result CR of T and the updated result UR of A

Output: An updated schema S′ of all hierarchies

Algorithm:

1: Calculate the modification of the aggregate A:
$\delta = UR - CR$

2: Retrieve participating raw tuples of A :
$T = \{x_1, x_2, ..., x_n\}$

3: Create a temporary table ΔX for T containing:
element identifier, keys of the dimensions and delta δ_i.

4: Calculate the difference for every raw tuple:
$\forall x_i \in T$: $\delta_i = \delta * weight(x_i)$
Add update attribute δ_i of table ΔX for each tuple x_i

5: Update all the impacted raw tuples:
$\forall bt_i \in T$: $val'(bt_i) = val(bt_i) + \delta_{bt_i}$

6: For each level of each hierarchy of each dimension

7: Identify impacted aggregates A′ in all aggregates $\mathbb{A}$:
$A' = \{A_i \in \mathbb{A} | imp(A, A_i)\}$

8: Calculate the difference for every aggregate:
$\forall A_i \in A'$: $\delta_{A_i} = \sum_{x_i \in \{t \in T | dep(A_i, t)\}} (\delta_{x_i})$

9: Update the impacted aggregates:
$\forall A_i \in A'$: $val'(A_i) = val(A_i) + \delta_{A_i}$

10: End for

Step 1: retrieval of the participating tuples to the aggregate, creation of a temporary table and calculation of differences

Retrieve the composition of the aggregate $\alpha_{\{a,i,j\}}$: sales of the aggregate $\alpha_{\{a,i,j\}}$ is the sum of a, i and j. Hence, the composing tuples are a, i and j.

Create a temporary table ΔX for the raw tuples that are identified.

Calculate the δ for the aggregate $\alpha_{\{a,i,j\}}$: δ = 600 000 - 500 000 = 100 000.

Calculate the difference for every tuple using the tuple weight.

$\delta_a = \delta * weight(a) = 100\,000 * \frac{100\,000}{500\,000} = 20\,000$

$\delta_i = \delta * weight(i) = 100\,000 * \frac{200\,000}{500\,000} = 40\,000$

$\delta_j = \delta * weight(j) = 100\,000 * \frac{200\,000}{500\,000} = 40\,000$

The resulting differences of raw tuples are added to the temporary table. This table also contains the dependency information to higher hierarchical levels (shown in Table 3.2).

Step 2: update of raw tuples

Update the raw tuples impacted by the aggregate modification. The new values of these

Table 3.2: Temporary table ΔX created to store impacted raw tuples

element identifier	customer key	product key	delta δ_x
a	customer_key$_a$	product_key$_a$	20 000
i	customer_key$_i$	product_key$_i$	40 000
j	customer_key$_j$	product_key$_j$	40 000

raw tuples are computed by their actual values and the differences calculated in step 1.

$$val'(t) = val(t) + \delta_t$$

In this case, a is updated to 100 000 + 20 000 = 120 000, i to 200 000 + 40 000 = 240 000 and j to 200 000 + 40 000 = 240 000.

Step 3: identification of impacted aggregates and update of impacted aggregates

Identify level by level all the aggregates impacted by the modification of the result of the aggregate $\alpha_{\{a,i,j\}}$ by using the dependencies between aggregates and registered raw tuples in the temporary table ΔX. In this case, we identify all the dark rectangles in Figure 3.1.

Propagate the changes to every impacted aggregate. Let us illustrate this issue with the customer dimension hierarchy 1. We loop for every level of the hierarchy. For level 1, two aggregates to be updated are identified: $\alpha_{\{a,c,d\}}$ and $\alpha_{\{i,j\}}$ because they have at least one of the registered raw tuples in their composition. The aggregate $\alpha_{\{a,c,d\}}$ depends on a, c and d and among these raw tuples, only one is registered in the table ΔX, namely, the raw tuple a. Hence, the value of $\alpha_{\{a,c,d\}}$ is changed only by adding δ_a (here 20 000).

$$\begin{aligned} val'(\alpha_{\{a,c,d\}}) &= val(\alpha_{\{a,c,d\}}) + \delta_a \\ &= val(\alpha_{\{a,c,d\}}) + 20\,000 \end{aligned}$$

The new value of the other aggregate $\alpha_{\{i,j\}}$ at level 1 is then

$$\begin{aligned} val'(\alpha_{\{i,j\}}) &= val(\alpha_{\{i,j\}}) + \delta_i + \delta_j \\ &= val(\alpha_{\{i,j\}}) + 40\,000 + 40\,000; \end{aligned}$$

The root aggregate $\alpha_{\{a,b,c,d,e,f,g,h,i,j\}}$ at level 2 of the same hierarchy can be calculated in a similar way with only the differences of depending raw tuples which are registered in ΔX, a, i and j:

$$\begin{aligned} &val'(\alpha, \{a,b,c,d,e,f,g,h,i,j\}) \\ &\quad = val(\alpha_{\{a,b,c,d,e,f,g,h,i,j\}}) + \delta_a + \delta_i + \delta_j \\ &\quad = val(\alpha_{\{a,b,c,d,e,f,g,h,i,j\}}) + 20\,000 + 40\,000 + 40\,000 \end{aligned}$$

Doing this way, we update only the aggregates impacted by the modification for hierarchy 1 of the customer dimension. The propagation to other hierarchies are processed in the same manner. Finally, we obtain updated data over the entire schema.

Application of PAM for multiple hierarchies

In the example that illustrates the PAM algorithm, the aggregate, subject to a modification, results from only one hierarchy. Meanwhile, a modification can take place on an aggregate resulting from multiple hierarchies, for example, the sales of the product category "office furniture" for the city of "Lyon". The PAM algorithm can also be applied to these cases when aggregates resulting from multiple hierarchies are subject to a modification. Compared with the cases in which one hierarchy is involved, only the queries in the identification of raw tuples are different. There are more restrictions when retrieving participating tuples. With one hierarchy, we select raw tuples whose hierarchical classification is the modified aggregate regarding the hierarchy. With multiple hierarchies, we select raw tuples whose every hierarchical classification corresponds to the modified aggregate. In the example of the sales of the product category "office furniture" for the city of "Lyon", the impacted raw tuples are the intersection of raw tuples belonging to the product category "office furniture" and the ones corresponding to "Lyon".

3.3.1.2 Time complexity

In order to determine the scaling ability of the PAM algorithm, we evaluate its performance by estimating the time complexity.

Let n be the number of raw tuples impacted by the aggregate modification, k the total number of levels for all hierarchies and m the average number of aggregates to be updated in a given level. We assume that all tables used in the algorithms are correctly indexed and the optimization engine of the database management system performs a hash search. Let t_i be the time unit consumed by the actions carried out in line i of the algorithm given in Table 3.1, then line 1 is considered to consume time t_1, line 2 uses $n * t_2$ and so forth. The total time required to run this algorithm can be estimated as:

$$\begin{aligned} T &= t_1 + n * t_2 + n * t_3 + n * t_4 + n * t_5 + k * (n * t_7 + n * t_8 + m * t_9) \\ &= t_1 + n * t_2 + n * t_3 + n * t_4 + n * t_5 + k * n * t_7 + k * n * t_8 + k * m * t_9 \\ &= t_1 + n * (t_2 + t_3 + t_4 + t_5 + k * t_7 + k * t_8) + k * m * t_9 \end{aligned}$$

Suppose the unit time t_u is the same, then

$$\begin{aligned} T &= t_u + n * (t_u + t_u + t_u + t_u + k * t_u + k * t_u) + k * m * t_u \\ &= (2 * n * k + m * k + 1) * t_u \end{aligned}$$

Subsequently the time complexity of the PAM algorithm is estimated. In practical cases, as the value of n is much larger than m, the time complexity can be approximated by **O(k*n)**. We see that O(k*n) is polynomial in k and n, hence the PAM algorithm is a polynomial time algorithm.

3.3.2 PAM II Algorithm

In a second stage, we propose the PAM II algorithm, which is an extended version of PAM algorithm. The PAM II algorithm uses supplementary semantics (e.g., dependencies between raw tuples and aggregates) in order to improve the performance when propagating the aggregate modification. In the following paragraphs, we will describe the PAM II algorithm and show the difference between the PAM algorithm and its extension.

3.3.2.1 Description of the algorithm

In the PAM algorithm, we notice that we perform a loop on each level of each hierarchy to identify the aggregates to update. It means that we have one SQL query per level per hierarchy to execute. For the example of hierarchies shown in Figure 1.1, we have to execute 17 queries for customer dimension, 12 queries for product dimension and 7 queries for time dimension. If these similar queries can be grouped into a single query, the execution will be accelerated.

The dependencies between aggregates and raw tuples are already fixed when the dimensional schema is determined. The idea of this derivative is to provide direct access from all aggregates to raw tuples by employing meta-tables which contain their dependency information. In addition, the temporary table ΔX (Table 3.2) contains the keys of the dimensions (one key per dimension). If the identification of aggregates through dependency information by providing raw tuples' information is possible, we can reduce the size of this temporary table by not storing the keys of the dimensions.

The meta-tables are persistent tables and are created when the dimension schema is determined. They need to be maintained up-to-date afterwards when the schema is modified. One meta-table is created for one materialized view to limit the size of the meta-table for the sake of future efficient search. There are two attributes in these tables: keys of aggregate and keys of their depending raw tuples. Figure 3.2 depicts the database schema of how the meta-table "dependency_info" links materialized views and the fact table sales in a sales forecasting system.

The general approach of the PAM algorithm II remains the same as the PAM algorithm. We first identify and update involved raw tuples and then identify and update impacted aggregates by an intermediate temporary table. Nonetheless, the detailed processing of the creation of temporary table and the identification of aggregates is not the same. Since the dependency information already exists in the database, we do not need to store the keys of the dimensions in the temporary table. The temporary table has now

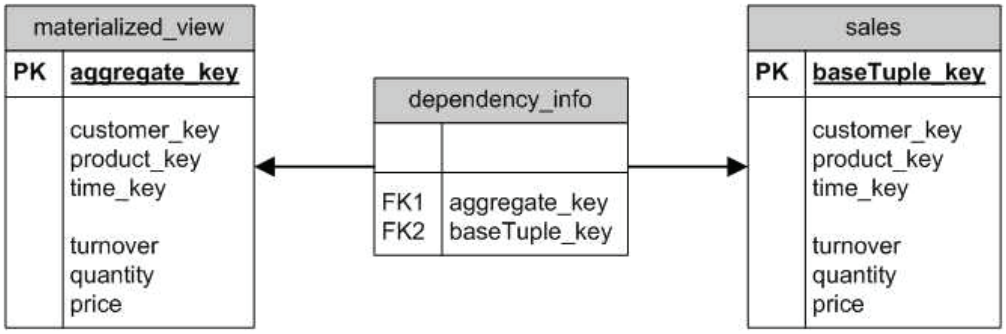


Figure 3.2: The database schema for meta-table storing dependency information

only two attributes: the element identifier and the delta for this element. The size of this temporary table is reduced. Regarding the identification of the impacted aggregates, instead of running through the dimension tables to identify impacted aggregates level by level, we can identify them directly through the dependency meta-table at one time.

Compared to the original algorithm PAM described in Table 3.1, the changes of the derived algorithm PAM II mainly target the lines 3, 6 and 10. The instruction given in line 3 creates a temporary table with less attributes than the one created by the original algorithm. For the update part of the aggregate, it is not necessary any more to loop through the dimensions and levels to perform the aggregate updates because we can identify all the aggregates at one time by dependency information in the meta-table. Line 6 and line 10 which intended to loop on levels of hierarchies are removed for the improved algorithm. The PAM II algorithm is shown in Table 3.3.
There are some further advantages with the meta-tables. These tables give direct dependency information between aggregates and raw tuples. This can serve not only the aggregates, which can be directly deduced from raw tuples via dimension hierarchy structure, but also the aggregates satisfying some specific conditions, e.g., the sum of sales for retail stores whose turnover is more than 100 000 euros. Hence, the PAM II algorithm can be applied more widely to any similar domain that needs to update raw tuples and other materialized views from an aggregate modification.

3.3.2.2 Time complexity

The performance of the PAM II algorithm is also calculated to determine its scalability.

Consider n to be the number of raw tuples that are impacted by the aggregation modification, k to be the total number of levels for all hierarchies and m to be the total number of aggregates that are influenced by the modification in the entire schema. We use the same method of the PAM algorithm to estimate the time complexity of the PAM

Table 3.3: Algorithm PAM II for the update propagation of a modification

Algorithm PAM II (Propagation of Aggregate Modification - II)

Input: Schema S, aggregate A=α_T, the current result CR of T, dependency meta-table D and the updated result UR of A
Output: An updated schema S′ of all hierarchies
Algorithm:
1: Calculate the modification of the aggregate A:
$\delta = UR - CR$
2: Retrieve participating raw tuples of A :
$T = \{x_1, x_2, ..., x_n\}$
3: Create a temporary table ΔX for T containing:
element identifier and delta δ_i.
4: Calculate the difference for every raw tuple:
$\forall x_i \in T$: $\delta_i = \delta * weight(x_i)$
Add update attribute δ_i of table ΔX for each tuple x_i
5: Update all the impacted raw tuples:
$\forall bt_i \in T$: $val'(bt_i) = val(bt_i) + \delta_{bt_i}$
6: Identify impacted aggregates A′ in all aggregates 𝔸:
$A' = \{A_i \in \mathbb{A} | imp(A, A_i)\}$
7: Calculate the difference for every aggregate:
$\forall A_i \in A'$: $\delta_{A_i} = \sum_{x_i \in \{t \in T | dep(A_i, t)\}} (\delta_{x_i})$
8: Update the impacted aggregates:
$\forall A_i \in A'$: $val'(A_i) = val(A_i) + \delta_{A_i}$

II algorithm. The total time required to run this algorithm is:

$$T = t_1 + n * t_2 + n * t_3 + n * t_4 + n * t_5 + n * t_6 + k * n * t_7 + m * t_8$$

In practice, as the value of n is much larger than m, the time complexity can be approximated by **O(k*n)**. We see that O(k*n) is polynomial in k and n, hence the PAM II algorithm is a polynomial time algorithm.

3.3.3 Other aggregate functions

Generally, the aggregate functions are divided into three classes [GCB*97]: distributive, algebraic and holistic. Distributive aggregate functions can be computed by partitioning their input into disjoint sets, aggregating each set individually and obtaining the final result by further aggregating the partial results. Among the aggregate functions, COUNT, SUM, MIN and MAX found in standard SQL, belong to this category. For example, COUNT can be computed by summing partial counts. Algebraic aggregate functions can be expressed as a scalar function of distributive aggregate functions. AVERAGE, for example, is an algebraic function since it can be expressed as SUM / COUNT. Holistic aggregate functions (e.g., MEDIAN) cannot be computed by dividing the input into parts.

We have introduced the PAM algorithm and its extension PAM II by using the aggregate function SUM. These algorithms are also applicable with other aggregate functions, except that in this work, we do not consider the holistic aggregate functions.

COUNT. Actually, the result of COUNT for higher hierarchical levels is the sum of the partial results corresponding to lower hierarchical levels. The PAM and PAM II algorithms for the COUNT aggregate function are similar to the algorithms used for the SUM function. We identify raw tuples involved in the calculation of the modified aggregate, which is the result of COUNT. We calculate the delta for each of those raw tuples and update them. Then, we identify aggregates impacted by this modification and update those aggregates. The only difference is the calculation of the delta δ for each raw tuple. The numbers used in SUM can be decimal numbers, but the result of COUNT should only contain natural numbers. We slightly modify the calculation mechanism in step 1, the calculation of delta, of the PAM and PAM II algorithms by adding a prune phase to the temporary table ΔX. Once the delta δ of each raw tuple is calculated by their contribution weight of the result, it will be rounded to integer if necessary. The rules are the following:

- if δ is an integer, it will be recorded as such.
- if δ is a decimal, the sum of fractional part of all decimal δ is 1, so
  - the raw tuple having the biggest fractional part will get 1.
  - in the case of equality for fractional part, the raw tuple having the biggest integer part will get 1.
  - in the case of equality for both integer and fractional parts, the first raw tuple registered in the table ΔX will get 1.

AVG. We assume that if a view contains the AVG aggregate function, the materialized view will contain instead the SUM and COUNT functions. The PAM and PAM II algorithms for AVG aggregate function are then reduced to the combination of algorithms for SUM and COUNT functions. The only difference is that, for the function AVG, we lightly modify the structure of the temporary table ΔX. Instead of storing one column for the delta δ, two columns are created: one for storing the delta δ_{sum} of SUM, and the other one for storing the delta δ_{count} of COUNT. The propagation of the aggregate modification, i.e. update of raw tuples involved and update of impacted aggregates,

is processed with the modification of results of SUM and COUNT functions. The algorithms remain globally the same.

MAX and MIN. The above functions, SUM, COUNT, AVG, generate new tuples. However, the aggregate functions MAX and MIN do not generate new tuples. Their results correspond to selected raw tuples. When the result of MAX or MIN is modified, it is the value of the raw tuple (or raw tuples in the case of equality) that is modified. We do not need to identify raw tuples involved in the modification, because they are already known. We assume that we store the MAX/MIN raw tuple(s) and their followers in the materialized views. The PAM and PAM II algorithms only need to identify the impacted aggregates, whose underlying modified raw tuple(s) are the same as those of MAX/MIN or as their followers. When the value of a MAX or a MIN raw tuple is modified, we compare directly the follower with the new value. If the result after modification is bigger than the follower in the case of MAX or smaller than the follower in the case of MIN, the aggregate result does not need to be updated. If not, we replace the MAX/MIN tuple by its follower. The followers' information needs to be updated consequently.

Chapter 4

Experimental evaluation and validation

Contents

4.1 Presentation of the experimental environment 40
4.2 Evaluation of different methods with two dimensions 40
4.2.1 Current solution 41
4.2.2 PAM algorithm 41
4.2.2.1 Validation 41
4.2.2.2 Complexity 42
4.2.3 PAM II algorithm 46
4.2.3.1 Validation 46
4.2.3.2 Complexity 47
4.2.4 Comparison of different methods 49
4.3 Evaluation of different methods with three dimensions 50

In this chapter, we will discuss the performed experiments in order to evaluate the proposed algorithms. We first introduce the experimental environment: hardware and software platforms. Experiments are then divided into two parts. The first part is in a two-dimensional data schema and the second part is in a three-dimensional data schema. We describe database and data schema for each schema. We evaluate the current solution, our proposed algorithms, PAM (Propagation of Aggregate-based Modification) and PAM II in the two data schemas. We also validate the estimation of the time complexity of our PAM and PAM II algorithms. Finally, we compare the results of different solutions and demonstrate the improvements of performance achieved by the PAM algorithm and its extension PAM II.

4.1 Presentation of the experimental environment

The main technical features of the server on which we run the evaluation are: two Intel Quad core Xeon-based 2.4 GHz, 16 GB RAM and one SAS disk of 500 GB, 15000 rotations per second. The operating system is a 64-bit Linux Debian system using the 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, out of which 50% is used in the computation engine, 45% for result visualization and 5% for the web framework. The problem we deal with is concerned with the result visualization. Our test only focuses on the data used by the update: one fact table and dimension tables.

4.2 Evaluation of different methods with two dimensions

In this first data schema, there are only two dimensions: *customer* and *product*. The fact table containing the keys of the dimensions and forecasts measures has about 300 MB, with 257.8 MB of data and 40.1 MB of indexes. There are 688 419 raw tuples in this fact table. As we know, materializing all aggregates of a data cube is not applicable in a real application. In this experiment, we materialized aggregates resulting from one hierarchy of one dimension, that represents 6 861 aggregates. The customer dimension table contains 5240 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) (ref. see Section 1.2 for the definition of fictive customer and fictive product).

Each of these dimension tables is composed of 4 hierarchies. It presents a similar structure to the one depicted in Figure 1.1 with different numbers of levels in each hierarchy (from 2 to 4 levels). Note that the time dimension is investigated within the fact table for some performance issues [Fen11, FLHD11] (see Section 5.4.5 for more explications). Hence, only two explicit dimensions are materialized in dimension tables.

In this section, we will show the evaluation results of different methods in a two-dimensional environment The objective of the evaluation is to show the time of updating the whole schema using the current solution and our PAM and PAM II algorithms. We demonstrate the benefits brought by our algorithms. We also validate the estimation of their complexity. Different tests are performed with respect to the place of modification. This refers to aggregate modifications which take place on each level of 3 hierarchies, which have 2, 3 and 4 levels, respectively. In our evaluation, we modify one aggregate from each level of each of these 3 hierarchies to compare the evaluation time resulting

from the current solution and from our approaches. The number of raw tuples involved in the aggregate modification is shown in Table 4.1. In other words, this is the number of tuples stored in the temporary table for the PAM and PAM II algorithms.

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
	level 1	level 2	level 1	level 2	level 3	level 1	level 2	level 3	level 4
Number	64 308	688 419	61 567	61 580	688 419	4 739	50 071	262 771	688 419

Table 4.1: Number of raw tuples involved by the aggregate modification on the appropriate level of hierarchies in the two-dimensional schema

4.2.1 Current solution

We first perform tests with the current solution. The result is shown in Table 4.2. In this table, we see that when the modification occurs at level 1 of the Hierarchy H1, it takes 0.9 second to perform the step 1, to update raw tuples and 179.5 seconds to perform step 2, to delete and reconstruct all the aggregates. The total time spent for the update of the entire schema caused by this modification is 180.4 seconds. This table shows time spent for updates of the whole schema when modifications occur at different level of different hierarchies. We notice that the time devoted to step 2 stays almost the same for different hierarchies. That is because it is concerned with the destruction and the recomputation of the whole schema each time. This operation is also the source of the latency of the current solution.

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
(seconds)	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*	179.5	182.1	185.7	181.4	188.4	181.1	179.6	179.9	176.6
Total	180.4	190.0	186.6	182.4	195.9	181.2	180.4	182.8	184.4

* Step 1: updating raw tuples;
* Step 2: deleting outdated aggregates and constructing updated aggregates

Table 4.2: Evaluation time of updating the whole schema following an aggregate modification by using the current solution in a two-dimensional data warehouse

4.2.2 PAM algorithm

4.2.2.1 Validation

The same tests are performed with our PAM algorithm. The result is shown in Table 4.3. We take the same modification example introduced within the current solution. When

we modify an aggregate at level 1 of the Hierarchy H1, it takes 0.3 second to perform stage 1, to create a temporary table containing raw tuples information; 1.0 second to perform stage 2, to update raw tuples and 4.4 seconds to perform stage 3, to propagate modifications to all impacted aggregates. In total, we spend 5.8 seconds to update the entire schema.

If we analyze the results of different levels of one hierarchy, we can see that they globally correspond to our estimation of first time complexity criterion, i.e., number of raw tuples involved in a modification. When a modification occurs in a high level, the number of raw tuples involved in the modification may be large. Then, the execution of the algorithm takes more time. In contrast, a modification on a low level impacts less raw tuples and thus less time is required to update the whole schema. That is why in this table, we note that the time consumed to deal with a higher level is greater than the time required to deal with lower levels of the same hierarchy.

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
(seconds)	level 1	level 2	level 1	level 2	level 3	level 1	level 2	level 3	level 4
Step 1*	0.3	3.0	0.3	0.3	2.6	0.05	0.3	1.4	3.0
Step 2*	1.0	8.1	0.9	0.9	7.9	0.1	0.8	3.3	8.3
Step 3*	4.4	47.2	4.4	4.4	49.4	0.5	3.5	17.9	47.4
Total	5.8	58.3	5.6	5.6	59.8	0.7	4.5	22.6	58.7

* Step 1: creating a temporary table of four attributes;
* Step 2: updating raw tuples;
* Step 3: propagating modifications to impacted aggregates

Table 4.3: Evaluation time of updating the whole schema following an aggregate modification by using our PAM algorithm in a two-dimensional data warehouse

4.2.2.2 Complexity

We estimated that the time complexity of the PAM algorithm is polynomial to the number of tuples involved in the modification and to the total number of levels of all hierarchies. In the following paragraphs, we validate our estimation of the time complexity with some experiments.

Complexity wrt the number of tuples involved

To validate this estimation, we compare the estimated evaluation time and the observed one on a twice bigger database. In the remaining of the chapter, we will call this twice bigger database "*DB_twice*". The number of raw tuples is twice the number of raw tuples in the fact table introduced in Section 4.1. The dimension and the hierarchy structure stays the same. All the aggregates use twice the number of raw tuples. As the

time complexity is estimated to be polynomial to the number of tuples involved in the modification, the evaluation time should double when the number of tuples involved is doubled. The estimation on this twice bigger table is then twice the result in the original database (shown in Table 4.3). This estimation result is calculated and shown in Table 4.4.

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
(seconds)	level 1	level 2	level 1	level 2	level 3	level 1	level 2	level 3	level 4
Step 1*	0.7	6.0	0.6	0.6	5.2	0.1	0.6	2.7	6.0
Step 2*	2.0	16.3	1.8	1.7	15.7	0.2	1.6	6.6	16.6
Step 3*	8.8	94.3	8.8	8.7	98.8	1.0	6.9	35.8	94.8
Total	11.5	116.6	11.2	11.0	119.7	1.3	9.0	45.1	117.4

* Step 1: creating a temporary table of four attributes;
* Step 2: updating raw tuples;
* Step 3: propagating modifications to impacted aggregates

Table 4.4: Estimated evaluation time of updating the whole schema following an aggregate modification by using our PAM algorithm in "*DB_twice*" of two dimensions

We then perform real experiments in "*DB_twice*". We modify the same aggregates as we did in the original database. The modifications of different tests also take place at each level of each of the three hierarchies. The observed result of real experiments is shown in Table 4.5.

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
(seconds)	level 1	level 2	level 1	level 2	level 3	level 1	level 2	level 3	level 4
Step 1*	0.7	5.8	0.6	0.6	5.7	0.1	0.5	2.3	5.8
Step 2*	1.8	17.1	1.6	1.6	15.3	0.2	1.3	6.2	16.6
Step 3*	9.0	95.1	8.7	8.8	100.1	0.9	7.0	36.5	96.2
Total	11.5	118.0	10.9	11.0	121.1	1.1	8.9	45.0	118.5

* Step 1: creating a temporary table of four attributes;
* Step 2: updating raw tuples;
* Step 3: propagating modifications to impacted aggregates

Table 4.5: Observed evaluation time of updating the whole schema following an aggregate modification by using our PAM algorithm in "*DB_twice*" of two dimensions

To compare the estimated and observed results, we compute their percent difference. The percent difference is a mathematical measure generally used to compare two different values of the same property. The percent difference between two numbers is the difference between them expressed as a percent change with respect to the numbers. The formula to calculate the percent difference between two values is given below [Per]:

$PercentDiff = [(|Value\ 1 - Value\ 2|)/Value\ 2] * 100,$

where Value 1 refers to observed value and Value 2 refers to accepted value.

In this case, the estimated result in Table 4.4 is considered as accepted values (Value 2) and the observed values in Table 4.5 are considered as observed values (Value 1). We calculate the percent difference between these values and the result is shown in Table 4.6.

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
(seconds)	level 1	level 2	level 1	level 2	level 3	level 1	level 2	level 3	level 4
Estimated result	11.5	116.6	11.2	11.0	119.7	1.3	9.0	45.1	117.4
Observed result	11.5	118.0	10.9	11.0	121.1	1.1	8.9	45.0	118.5
Percent difference	0.2%	1.2%	2.9%	0.4%	1.2%	13.1%	1.6%	0.3%	1.0%

Table 4.6: Percent difference between the estimated result and the observed result in "*DB_twice*" of two dimensions

According to the calculation, we find that the percent difference is between 0.2% and 2.9%, except the case of a modification on level 1 of hierarchy H3. This exception makes a percent difference of 13.1%. This exception can be explained by the too short evaluation time of this modification. As this aggregate update leads only to a small modification, the estimated evaluation time is only 1.3 second and the observed evaluation time is only 1.1 second. These values are so small that other factors could have a more significant influence on the evaluation time, like CPU process/thread priority or memory and disk activities. To be general, we do not take this exception into consideration. For all other cases, the two results are very close. The fact that the algorithm is polynomial to the number of tuples involved in the modification of the PAM algorithm is shown by using "*DB_twice*".

Complexity wrt the total number of levels of all hierarchies

The second criterion influencing the time complexity is the total number of levels for all hierarchies. We estimate that the evaluation time is polynomial to this number. To validate this estimation, we perform different experiments in the original database copy. Every time we fix the raw tuples and the aggregate, subject to modification, and we redefine the dimensions and the hierarchies to generate different schemas. Different dimension/hierarchy schemas give different numbers of levels for all hierarchies. For example, we consider a new dimension/hierarchy schema with two customer hierarchies and one product hierarchy. We have in total 8 levels for the three hierarchies. In another example, we consider a schema with three customer hierarchies and two product hierarchies which creates 16 levels in total for the five hierarchies. As the number

of raw tuples involved in the modification does not change, the time spent in Step 1, the creation of temporary table and in Step 2, the update of raw tuples do not change (see Table 4.3 for the explanations of the three Steps). We compare only the Step 3, the propagation of updates to all hierarchies. The actual Anticipeo's dimension/hierarchy schema has 22 levels for all hierarchies. The experiments are performed on different schemas with 8, 12, 16, 20 and 22 levels respectively. The evaluation of the modification on each level of each hierarchy in different dimension/hierarchy schemas is shown in Table 4.7.

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
(seconds)	level 1	level 2	level 1	level 2	level 3	level 1	level 2	level 3	level 4
8 levels	1.6	16.1	1.9	1.9	19.3	0.2	1.3	6.2	16.3
12 levels	2.5	25.5	2.6	2.6	27.6	0.3	1.8	9.7	26.0
16 levels	3.2	33.6	3.3	3.3	36.9	0.4	2.5	12.9	33.8
20 levels	4.2	43.5	4.0	4.0	44.4	0.5	3.2	16.7	43.8
22 levels	4.4	47.2	4.4	4.4	49.4	0.5	3.5	17.9	47.4

Table 4.7: Evaluation time of propagating modifications to all hierarchies using PAM algorithm under different dimension/hierarchy schemas with two dimensions

To confirm the fact that the algorithm is polynomial to the total number of levels of all hierarchies, we calculate the evaluation time per level for different cases. We divide the evaluation time by the corresponding number of levels of hierarchies. The evaluation time per level is shown in Table 4.8.

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
(seconds)	level 1	level 2	level 1	level 2	level 3	level 1	level 2	level 3	level 4
8 levels	0.197	2.008	0.234	0.232	2.407	0.022	0.163	0.770	2.034
12 levels	0.211	2.128	0.214	0.217	2.301	0.024	0.153	0.808	2.170
16 levels	0.199	2.102	0.203	0.209	2.306	0.023	0.156	0.805	2.113
20 levels	0.208	2.177	0.202	0.198	2.218	0.027	0.158	0.837	2.191
22 levels	0.200	2.144	0.201	0.198	2.245	0.024	0.157	0.813	2.155
Variance	$3.7 * 10^{-5}$	$4.1 * 10^{-3}$	$2.0 * 10^{-4}$	$2.0 * 10^{-4}$	$5.3 * 10^{-3}$	$3.5 * 10^{-6}$	$1.3 * 10^{-5}$	$5.8 * 10^{-4}$	$3.9 * 10^{-3}$
Standard deviation	$6.0 * 10^{-3}$	$6.4 * 10^{-2}$	$1.4 * 10^{-2}$	$7.3 * 10^{-2}$	$1.9 * 10^{-3}$	$3.6 * 10^{-3}$	$2.4 * 10^{-2}$	$2.4 * 10^{-2}$	$6.2 * 10^{-2}$

Table 4.8: Evaluation time per level of propagating modifications to all hierarchies using PAM algorithm under different dimension/hierarchy schema with two dimensions

In addition to the calculated evaluation time per level, we compute the variance [var] and the standard deviation [std] on the result. The (population) variance of a random variable is a non-negative number which gives an idea of how widely spread the values of the random variable are likely to be; the larger the variance, the more scattered the observations on average. The standard deviation is a measure of the spread or dispersion of a set of data. The variance and the standard deviation are widely used measures of

variability or diversity used in statistics and probability theory. The standard deviation is the (positive) square root of the variance. These measures show how much variation or "dispersion" exists from the average (mean, or expected value). They have some common properties [Sci12]. They are proportional to the scatter of the data (small when the data are clustered together, and large when the data are widely scattered). They are independent of the number of values in the data set (otherwise, simply by taking more measurements, the value would increase even if the scatter of the measurements was not increasing). In addition, they are independent of the mean (since now we are only interested in the spread of the data, not its central tendency). A low variance or a low standard deviation indicates that the data points tend to be very close to the mean, whereas a high variance or a high standard deviation indicates that the data points are spread out over a large range of values.

The variance and the standard deviation in Table 4.8 are very low. They demonstrate that the values of the evaluation time per level under different dimension/hierarchy schemas are very concentrated to their mean. We prove that the algorithm is polynomial to the total number of levels for all hierarchies of the PAM algorithm.

4.2.3 PAM II algorithm

4.2.3.1 Validation

To validate the PAM II algorithm, we perform the same tests described at the beginning of Section 4.2 with the PAM II algorithm as we did with the current solution and the PAM algorithm. The result is shown in Table 4.9. We take the same modification example introduced within the current solution and the PAM algorithm. When we modify an aggregate at level 1 of the Hierarchy H1, it takes 0.3 second to perform stage 1, to create a temporary table containing raw tuples information; 1.0 second to perform stage 2, to update raw tuples and 5.9 seconds to perform stage 3, to propagate modifications to all impacted aggregates. In total, we spent 7.2 seconds to update the entire schema. Compared to 5.8 seconds using the PAM algorithm, this extended version does not show much effect of performance improvement for low level modifications. High level modifications show that the PAM II algorithm is better when compared to the other solutions. For example, only 35.7 seconds are needed to propagate a modification occurring on level 2 of the hierarchy H1. Using the PAM algorithm, we should spend 58.3 seconds for the same operation.

The results of different levels of one hierarchy also confirm our estimation of first time complexity criterion, i.e., number of raw tuples involved in a modification. High

level modification takes more time as the number of raw tuples involved in the modification might be large. In contrast, a modification on a low level impacts less raw tuples and requires less time to update the whole schema. That is why in this table, we note the time consumed on a higher level is more important than the time required for a lower level of the same hierarchy.

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
(seconds)	level 1	level 2	level 1	level 2	level 3	level 1	level 2	level 3	level 4
Stage 1*	0.3	2.7	0.3	0.3	2.5	0.05	0.3	1.2	2.7
Stage 2*	1.0	3.9	0.9	0.9	3.5	0.1	0.7	2.6	3.5
Stage 3*	5.9	29.2	3.3	3.4	28.8	0.3	2.4	11.2	29.6
Total	7.2	35.7	4.5	4.6	34.8	0.4	3.4	15.1	35.9

* Stage 1: creating a temporary table of two attributes;
* Stage 2: updating raw tuples;
* Stage 3: propagating modifications to impacted aggregates

Table 4.9: Evaluation time of updating the whole schema following an aggregate modification by using our derived PAM II algorithm in a two-dimensional data warehouse

4.2.3.2 Complexity

We estimated the time complexity of the PAM II algorithm to be polynomial to the number of tuples involved in the modification and to the total number of levels of all hierarchies. In the following, we validate our estimation of the time complexity by conducting some experiments.

Complexity wrt the number of tuples involved

To validate this estimation, we compare the estimated evaluation time and the observed one in "*DB_twice*". Section 4.2.2.2 describes "*DB_twice*". Experiments are performed in this database.

The estimation evaluation time is the result in the original database (shown in Table 4.9) multiplied by 2. This estimation result is calculated and shown in Table 4.10.

We then perform real experiments in "*DB_twice*". The observed result is shown in Table 4.11.

To compare the estimated and observed results, we compute their percent difference shown in Table 4.12.

The percent difference shows that the two results are very close for all cases. The fact that the algorithm is polynomial to the number of tuples involved in the modification of the PAM II algorithm is shown by using "*DB_twice*".

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
(seconds)	level 1	level 2	level 1	level 2	level 3	level 1	level 2	level 3	level 4
Stage 1*	0.7	5.3	0.6	0.6	5.0	0.1	0.6	2.4	5.5
Stage 2*	2.0	7.8	1.8	1.8	7.1	0.2	1.4	5.3	7.1
Stage 3*	11.8	58.4	6.6	6.7	57.5	0.6	4.8	22.5	59.3
Total	14.5	71.5	9.0	9.1	69.5	0.9	6.8	30.1	71.8

* Stage 1: creating a temporary table of four attributes;
* Stage 2: updating raw tuples;
* Stage 3: propagating modifications to impacted aggregates

Table 4.10: Estimated evaluation time of updating the whole schema following an aggregate modification by using our PAM II algorithm in "*DB_twice*" of two dimensions

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
(seconds)	level 1	level 2	level 1	level 2	level 3	level 1	level 2	level 3	level 4
Stage 1*	0.7	5.5	0.6	0.6	4.7	0.1	0.5	2.1	5.3
Stage 2*	1.8	7.3	1.5	1.5	7.6	0.2	1.1	5.7	7.0
Stage 3*	12.4	60.5	6.2	6.3	58.6	0.6	4.4	23.6	59.5
Total	14.9	73.4	8.8	8.8	70.9	0.9	6.7	31.3	71.8

* Stage 1: creating a temporary table of four attributes;
* Stage 2: updating raw tuples;
* Stage 3: propagating modifications to impacted aggregates

Table 4.11: Observed evaluation time of updating the whole schema following an aggregate modification by using our PAM algorithm in "*DB_twice*" of two dimensions

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
(seconds)	level 1	level 2	level 1	level 2	level 3	level 1	level 2	level 3	level 4
Estimated result	14.5	71.5	9.0	9.1	69.5	0.9	6.8	30.1	71.8
Observed result	14.9	73.4	8.8	8.8	70.9	0.9	6.7	31.3	71.8
Percent difference	2.6%	2.6%	1.8%	3.3%	1.9%	0.4%	1.6%	3.8%	0.1%

Table 4.12: Percent difference between the estimated result and the observed result in "*DB_twice*" of two dimensions

Complexity wrt the total number of levels for all hierarchies

The second criterion influencing the time complexity is the total number of levels for all hierarchies. We estimate that the evaluation time is polynomial to this number. To validate this estimation, the experiments in the original database copy are performed on different schemas with 8, 12, 16, 20 and 22 levels respectively. The evaluation of the modification on each level of each hierarchy in different dimension/hierarchy schemas is shown in Table 4.13.

To confirm that the algorithm is polynomial to the total number of levels of all hi-

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
(seconds)	level 1	level 2	level 1	level 2	level 3	level 1	level 2	level 3	level 4
8 levels	4.9	16.0	2.2	2.3	15.8	0.2	1.6	6.8	16.3
12 levels	5.3	19.9	2.5	2.6	20.0	0.2	1.8	8.0	20.3
16 levels	5.5	23.6	2.9	2.9	23.1	0.3	2.2	9.6	24.2
20 levels	6.0	27.3	3.2	3.2	27.4	0.3	2.2	10.6	27.5
22 levels	4.4	47.2	4.4	4.4	49.4	0.5	3.5	17.9	47.4

Table 4.13: Evaluation time of propagating modifications to all hierarchies using PAM II algorithm under different dimension/hierarchy schemas with two dimensions

erarchies, we calculate the evaluation time per level for different cases. We divide the evaluation time by the corresponding number of levels of hierarchies. The evaluation time per level is shown in Table 4.14.

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
(seconds)	level 1	level 2	level 1	level 2	level 3	level 1	level 2	level 3	level 4
8 levels	0.617	1.995	0.277	0.282	1.978	0.024	0.205	0.847	2.042
12 levels	0.440	1.657	0.205	0.217	1.665	0.018	0.152	0.668	1.693
16 levels	0.343	1.475	0.181	0.183	1.443	0.017	0.135	0.597	1.512
20 levels	0.299	1.363	0.160	0.160	1.369	0.015	0.111	0.531	1.375
22 levels	0.268	1.327	0.153	0.150	1.307	0.014	0.110	0.510	1.347
Variance	$2.0 * 10^{-2}$	$7.5 * 10^{-2}$	$2.5 * 10^{-3}$	$2.8 * 10^{-3}$	$7.5 * 10^{-2}$	$1.7 * 10^{-5}$	$1.5 * 10^{-3}$	$1.8 * 10^{-2}$	$8.2 * 10^{-2}$
Standard deviation	$1.4 * 10^{-1}$	$2.7 * 10^{-1}$	$5.0 * 10^{-2}$	$5.3 * 10^{-2}$	$2.7 * 10^{-1}$	$4.1 * 10^{-3}$	$3.9 * 10^{-2}$	$1.4 * 10^{-1}$	$2.9 * 10^{-1}$

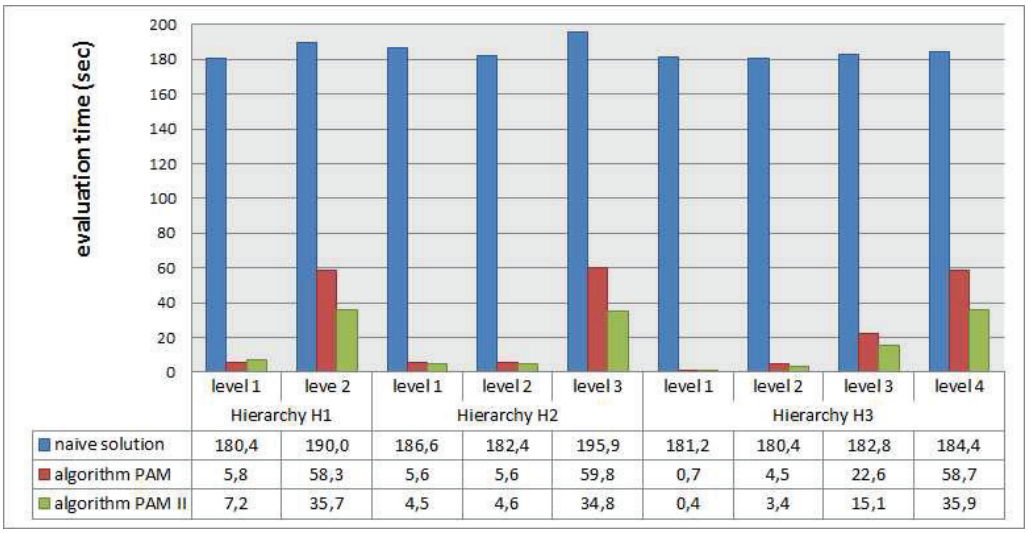
Table 4.14: Evaluation time per level of propagating modifications to all hierarchies using PAM II algorithm under different dimension/hierarchy schemas with two dimensions

The variance and the standard deviation in Table 4.14 are very low. They demonstrate that the values of the evaluation time per level under different dimension/hierarchy schemas are very concentrated to their mean. We prove that the algorithm is polynomial to the total number of levels for all hierarchies of the PAM II algorithm.

4.2.4 Comparison of different methods

We compare the total evaluation time using the three solutions in one chart shown in Figure 4.1.

Roughly speaking, the new algorithms display much better performance than the current solution. In most cases, the evaluation time is significantly reduced. For example, for the modification at level 1 of hierarchy H3, the propagation time is only 0.7 second using the PAM algorithm and 0.4 second using the PAM II algorithm. Compared to 181.2 seconds spent by the current solution, the gain of performance reaches 25786% and 45200% respectively. Even in the worst case where the root aggregate (the single aggregate at top level of every hierarchy) is subject to modifications, we get a nearly



	level 1	leve 2	level 1	level 2	level 3	level 1	level 2	level 3	level 4
	Hierarchy H1		Hierarchy H2			Hierarchy H3			
naive solution	180,4	190,0	186,6	182,4	195,9	181,2	180,4	182,8	184,4
algorithm PAM	5,8	58,3	5,6	5,6	59,8	0,7	4,5	22,6	58,7
algorithm PAM II	7,2	35,7	4,5	4,6	34,8	0,4	3,4	15,1	35,9

Figure 4.1: Comparison of evaluation time using the current solution, the PAM and PAM II algorithms

220% and 437% better performance using the PAM and PAM II algorithms. The result confirms that, instead of recalculating all the aggregates as the current solution does, our solutions are more efficient by identifying and updating the **exact** set of aggregates impacted by the modification.

Regarding the comparison between our algorithms, PAM and PAM II, PAM II shows an average of 40% better performance. In particular, higher levels benefit more from the existence of the meta-tables by avoiding complex joins. Nevertheless, we have sacrificed physical space. In this test, one meta-table is created to contain dependencies between raw tuples and hierarchical aggregates. There are 688 419 raw tuples and 6 861 aggregates in this test database. Even if the number of tuples in the meta-table is not the Cartesian product of raw tuples and aggregates, more precisely 688 419 * 6 861, there are still 17 711 504 tuples created in this meta-table. This represents 630 MB of data and 627 MB of indexes in terms of physical storage. For a database of 50 GB, the meta-table of 1.23 GB is relatively large. In addition, if other materialized views need to be updated in the same way, additional meta-tables should be created. Hence, when the physical storage is not a constraint, we recommend the PAM II algorithm. Otherwise, the PAM algorithm is a good candidate.

4.3 Evaluation of different methods with three dimensions

In the second data schema, we investigate the performance with three dimensions: *customer*, *product* and *time*. In Section 4.3, we introduce the fact that the time dimension of

this application is merged into the fact table. In this section, we make the time dimension explicit to create an environment of three dimensions with real data. The customer dimension table and the product dimension table are the same as the ones used in the schema with two dimensions. The time dimension table has 60 basic lines for 60 months and 13 fictive lines for corresponding different hierarchical years. The customer dimension and the product dimension are both composed of 4 hierarchies and the time dimension is composed of 2 hierarchies. The fact table containing the keys of the dimensions and forecasts measures has about 985.5 MB with 453 MB of data and 532.5 MB of indexes. There are 6 995 465 raw tuples in this fact table. Like in the experiment with two dimensions, we only materialized aggregates resulting from one hierarchy of one dimension, which represents 6 897 aggregates.

In this section, we will show the evaluation results of different methods in a three-dimensional environment. We also perform tests on each level of 3 hierarchies which have 2, 3 and 4 levels, respectively. In our evaluation, we modify one aggregate from each level of each of these 3 hierarchies to compare the evaluation time resulting from the current solution and from our approaches. The number of raw tuples involved in the modification is shown in Table 4.15.

	Hierarchy H1		Hierarchy H2			Hierarchy H3			
	level 1	level 2	level 1	level 2	level 3	level 1	level 2	level 3	level 4
Number	1 245 321	6 995 465	825 955	826 106	6 995 465	98 190	498 173	2 647 289	6 995 465

Table 4.15: Number of raw tuples involved in the modification of each test

Current solution

We first perform tests with the current solution. The result is shown in Table 4.16. As the second step of the solution consists in removing and constructing all aggregates, the time for each test stays almost the same. For the level 1 of the hierarchy H1, it takes a total of 220.1 seconds, corresponding to 12.5 seconds to update raw tuples involved in the modification and 207.7 seconds to reconstruct all aggregates in every hierarchy of every dimension.

PAM algorithm

The same tests are performed with our PAM algorithm. The result is shown in Table 4.17. We take the same modification example introduced within the current solution. When we modify an aggregate at level 1 of the Hierarchy H1, it takes 6.3 seconds to per-

(seconds)	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*	12.5	51.3	7.9	7.9	51.6	0.9	4.9	24.9	52.9
Step 2*	207.7	206.8	206.5	207.0	205.4	206.9	207.8	209.2	209.3
Total	220.1	258.1	214.4	214.9	257.0	207.8	212.7	234.1	262.3

* Step 1: updating raw tuples;
* Step 2: deleting outdated aggregates and constructing updated aggregates

Table 4.16: Evaluation time of updating the whole schema following an aggregate modification by using the current solution in a three-dimensional data warehouse

form step 1, to create a temporary table containing raw tuples information; 11.3 seconds to perform step 2, to update raw tuples and 38.8 seconds to perform step 3, to propagate modifications to all impacted aggregates. In total, we spend 56.4 seconds to update the entire schema.

(seconds)	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*	6.3	37.7	4.0	4.0	39.2	0.5	2.7	14.0	40.1
Step 2*	11.3	46.5	6.9	6.9	44.9	0.7	4.1	22.7	45.4
Step 3*	38.8	218.3	25.6	25.7	218.9	3.1	15.6	81.6	219.3
Total	56.4	302.5	36.5	36.7	303.1	4.2	22.3	118.3	304.8

* Step 1: creating a temporary table of four attributes;
* Step 2: updating raw tuples;
* Step 3: propagating modifications to impacted aggregates

Table 4.17: Evaluation time of updating the whole schema following an aggregate modification by using our PAM algorithm in a three-dimensional data warehouse

PAM II algorithm

The same tests are also performed with the extended PAM II algorithm. The result is shown in Table 4.18. We take the same modification example introduced within the current solution. When we modify an aggregate at level 1 of the Hierarchy H1, it takes 4.0 seconds to perform step 1, to create a temporary table containing raw tuples information; 10.9 seconds to perform step 2, to update raw tuples and 15.4 seconds to perform step 3, to propagate modifications to all impacted aggregates. In total, we spend 30.4 seconds to update the entire schema.

Comparison of the different methods

We compare the total evaluation time using the three solutions in a three-dimensional environment in one chart shown in Figure 4.2.

(seconds)	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*	4.0	24.7	2.9	4.2	24.8	0.3	1.5	9.5	24.5
Step 2*	10.9	45.5	7.1	7.1	46.1	0.7	4.1	24.1	45.3
Step 3*	15.4	67.2	8.8	10.1	69.8	1.1	7.5	44.0	65.6
Total	30.4	137.4	20.1	20.2	140.7	1.8	13.1	77.5	135.4

* Step 1: creating a temporary table of four attributes;
* Step 2: updating raw tuples;
* Step 3: propagating modifications to impacted aggregates

Table 4.18: Evaluation time of updating the whole schema following an aggregate modification by using our PAM II algorithm in a three-dimensional data warehouse

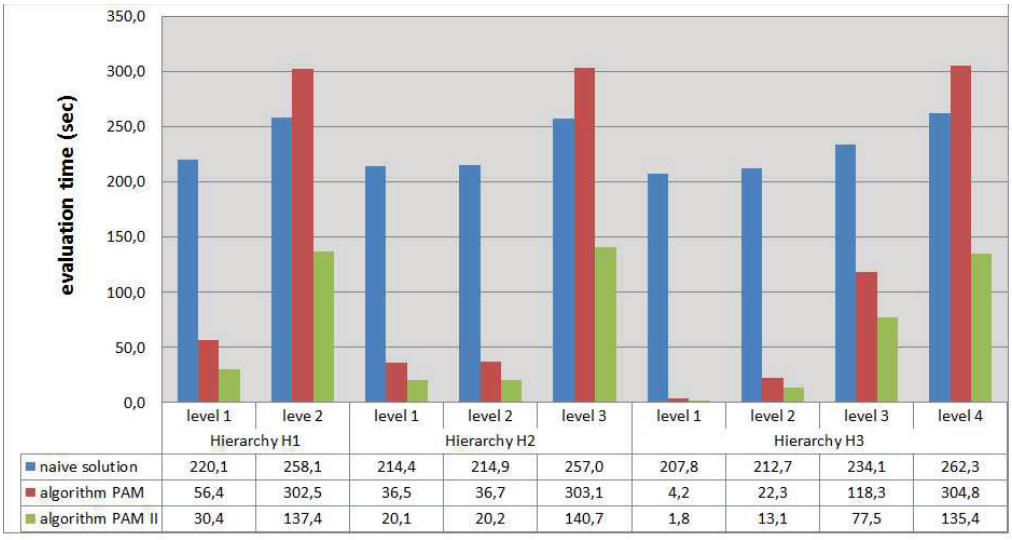


Figure 4.2: Comparison of evaluation time using the current solution, the PAM and PAM II algorithms in a three-dimensional schema

In most cases, proposed algorithms present a better performance than the current solution. We take the example of the level 1 of hierarchy H1, time spending to update the whole schema is reduced from 220.1 seconds using current solution to 56.4 seconds using PAM algorithm and 30.4 seconds using PAM II algorithm, which is a gain of 290% and 625% respectively for PAM and PAM II. In the case of modifying an aggregate, which impacts less raw tuples, the gain of PAM and PAM II is more important. As for the example of the level 1 of hierarchy H3, the gain of performance reaches 4827% and 11203% respectively.

However, we notice that in the worst case where the root aggregate (the single aggregate at top level of every hierarchy) is subject to modifications, the current solution of reconstructing all the aggregates is more efficient than the PAM algorithm. Applying the PAM algorithm on this data is not always optimal. Hence, when implementing the

PAM algorithm in the real application, we propose an alternative. As we mentioned previously, the PAM algorithm is linear to the number of raw tuples involved in an aggregate modification. We can compute the average time spent on a single raw tuple by dividing the total time by the number of raw tuples involved. In the case where the PAM algorithm is less efficient in time than the current solution, we switch to the current solution. The threshold is easy to determine. The execution time of the current solution is known, the average time spent on a single raw tuple by PAM is also known. Their division is the threshold under which PAM is more efficient. Therefore, when propagating an aggregate modification to the whole schema, we estimate the number of raw tuples that should be updated and make the decision of which solution to adopt.

In this schema, the meta-table of PAM II, which contains the dependencies between aggregates and raw tuples has 191 279 805 tuples. This represents 15 GB including 9.7 GB of data and 5.3 GB of indexes. In the case where the physical storage is not a constraint, the PAM II is the optimal solution.

Chapter 5

Context of this work: Anticipeo

Contents

5.1 Sales forecasting systems 56

5.2 Presentation of the Anticipeo application 58

5.2.1 Application process 58

5.2.2 User interface 59

5.2.3 Data features 60

5.2.4 Main manipulations 62

5.2.5 Problem statement 63

5.3 Optimization guideline 64

5.3.1 Hardware and application programming analysis 64

5.3.2 Database management system configuration 65

5.3.3 Additional materialized views 66

5.3.4 Database design 69

5.4 Implementation and optimization results 72

5.4.1 Observations on current implementation of the application . . . 72

5.4.2 Diagnosis of latency provenance 73

5.4.3 Database management system configuration 75

5.4.4 Selection of materialized views 76

5.4.5 Database schema modification 77

5.5 Overall optimization result 81

5.6 Recommendations 82

This CIFRE thesis is initiated by the Anticipeo company. The objective of this collaboration is to improve the performance of the main software, also called as the Anticipeo application. The Anticipeo application is a sales forecasting system that predicts future sales in order to help enterprise decision-makers to make appropriate business strategies in advance. This application provides a reliable result[1], but for some users' requests, the response time is not satisfactory. Hence, our work focuses on analysis of the performances problems and investigations of novel solutions to achieve the objective.

In this chapter, we introduce, first of all, sales forecasting systems. Then, we present the Anticipeo application with the process flow, the user interface, different functions, database information and its existing problems. We conduct an audit to diagnose this application. In compliance with the audit result, we propose an optimization guideline with a list of steps to follow. We show the results achieved by each optimization step as well as overall optimization results obtained when integrating all the proposals. Finally, some recommendations are given for the Anticipeo company to fulfill their needs in terms of performance.

5.1 Sales forecasting systems

Sales forecasting involves predicting the amount people will purchase, given the product features and the conditions of the sale. Sales forecasts help investors make decisions about investments in new ventures. It is essential for managing a business of any size. It runs a month-by-month prediction of the level of sales that are expected to achieve. Most businesses draw up a sales forecast once a year.

Sales forecasting is a self-assessment tool for a company [Viro9]. It allows to analyze the pulse of the business via reports composed of summaries and/or graphs. Implementing accurate sales forecasting systems could entail important benefits such as:

- Enhanced cash flow
- Knowing when and how much to buy
- In-depth knowledge of customers and the products they order
- The ability to plan for production and capacity
- The ability to identify the pattern or trend of sales

[1] Here, reliable stands for acceptable quality and acceptable approximation with results in the confidence interval.

- Determine the value of a business above the value of its current assets
- Ability to determine the expected return on investment

Profitability depends on (1) having a relatively accurate forecast of sales and costs; (2) assessing the confidence one can place in the forecast; and (3) properly using the forecast in the plan. There has been a long history in the field of sales forecasts computation research [App65]. The most popular theory is Reilly's law of retail gravitation [Rei31]. Many works were devoted to the improvement of this theory [GBGB05, GR06, LBYD08]. Other data models and methods were also proposed [Chao0, Hyno8, MWH98a]. Besides these studies concentrating on the techniques used in the computation of forecasts, other works focus on the systems or managerial approaches that are used [MK97, MMSG98]. Some other works consider qualitative factors as well as non-quantitative factors. The research work [KWW02] utilizes fuzzy logic which is capable of learning to learn the experts' knowledge regarding the effect of promotion on the sales.

Compared to other information systems, the design of sales forecasting systems, as other forecasting systems, should comply with some requirements:

- Limited life duration:
  Forecasting calculation is periodically triggered. Once the period passed, the forecasts are no longer exploitable. For example, the weather forecasts predicted for yesterday are not necessarily useful today.
- Non-reusable:
  The forecasting is a global calculation and it is almost impossible to reuse the former results to reduce future forecasting calculation. This means partial recomputation of precomputed results is not easy.
- Updates on built predictions:
  As forecasting information results from a computation process, there is often no need for updates. However, in some cases, the system needs some corrections for some unpredictable situations, e.g., the impact of a car crash for a traffic forecasting system.
- Timeliness:
  Forecasting systems should provide just in time responses for users. Decision makers devise corresponding plans of purchasing, manufacturing, logistics, etc., according to sales forecasting. The delay between achieved data entering and new data forecasting should be as short as possible.

5.2 Presentation of the Anticipeo application

The Anticipeo application is a sales forecasting system. It helps decision makers or executives to foresee the trends of the future market and adapt their business plan in time. It also provides them with the possibility to simulate future situations in order to achieve some desired objectives. In the following, we will present this application in detail.

5.2.1 Application process

The forecasting work is usually performed periodically. The interval of forecasting depends on the nature of application. In compliance with the normal business process, the Anticipeo application performs a monthly forecasting. The process flow of the Anticipeo application is shown in Figure 5.1.

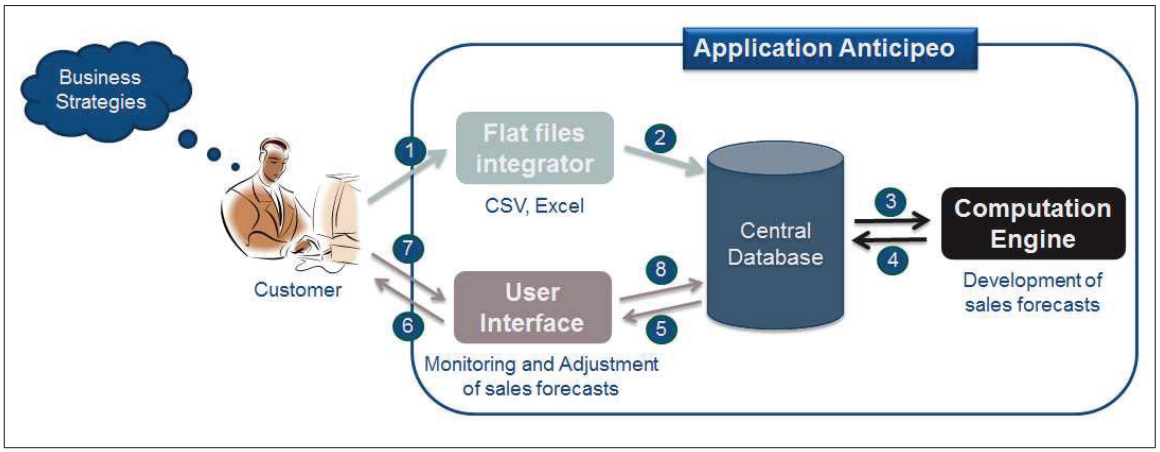


Figure 5.1: The data process of the Anticipeo application

Every month, customers provide Anticipeo with their new achieved sales in form of flat files, e.g., CSV (Comma-Separated Values) files or MS Excel [spr]. A CSV file contains the values in a table as a series of ASCII text lines organized so that each column value is separated by a comma from the next column's value and each raw starts a new line [csv]. Anticipeo integrates the data into the main database of the system. The computation engine then calculates sales forecasts based on the features of the historical data with appropriate statistical models. Once new forecasts are established, they are stored into the main database. Before presenting the result to the customer, the application prepares some information for analysis, which accelerates some time-consuming consultations. Now customers can navigate via a secured web service to the forecasting sales. They can also perform some modifications on the forecasts. In this case, the visualization generator will be launched to adjust the information for analysis. According to

final results, decision makers make their decision of strategic plans for next month(s).

Our work mainly focuses on the users' interaction part, i.e., the visualization of historical and forecasting data and the modification of forecasting data.

5.2.2 User interface

Figure 5.2 is the interface of a demonstration website of Anticipeo. It presents the results generated at the end of May 2011 for a selected category of product "Appareillages (APPA)".

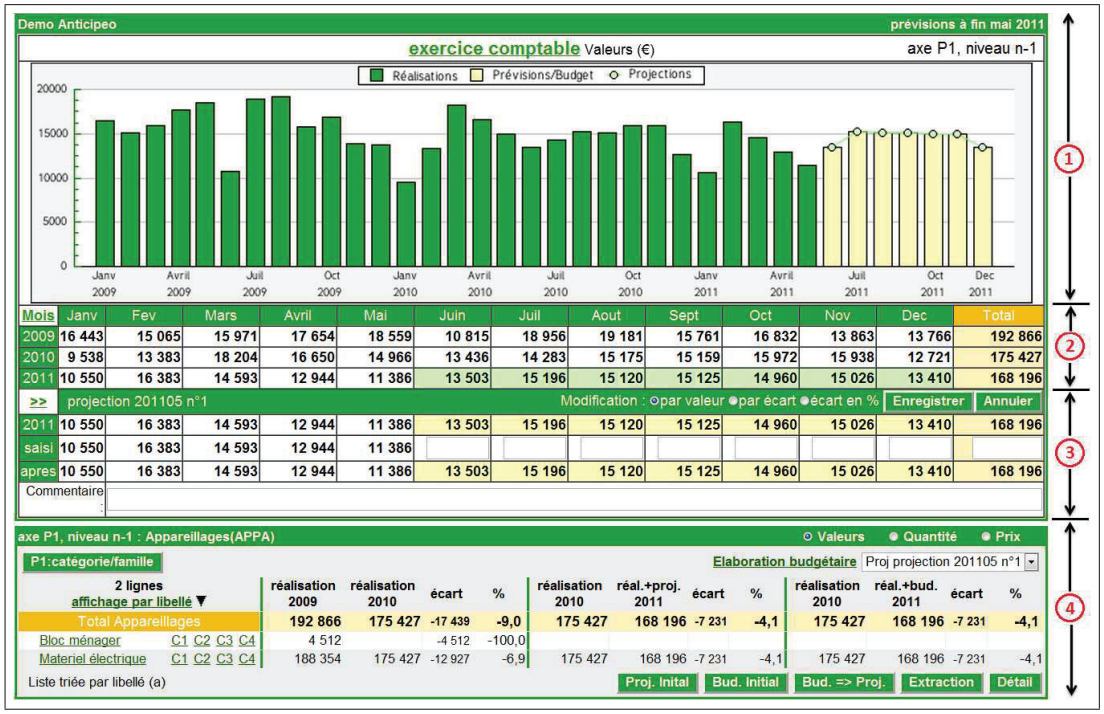


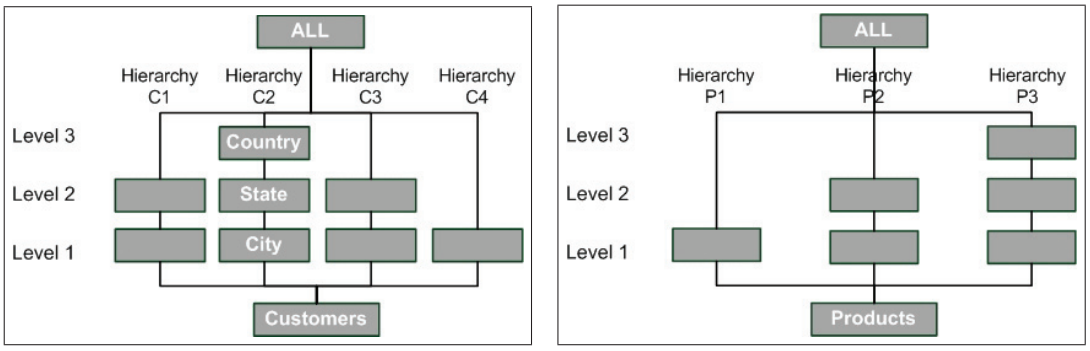
Figure 5.2: An example of forecasting sales trend presentation

The interface is composed of four blocks: a bar chart, a table of consultation, a table of modification (displayed when demanded) and a navigation table of the composing information for this selected category of product. Block 1 and block 2 show the sales information for a total of 3 years including sales history for the last 29 months and its trend for the next 7 months in the form of a bar chart and a table. In the bar chart, different colors and forms are used to express different information. Dark green bars represent historical sales; light yellow bars represent future predictive sales estimated by salespersons and sales managers while small light green circles represent forecasting sales computed by the Anticipeo application. By default, predictive sales estimated by salespersons are set to be equal to the computed forecasting sales. They can be modified

by salespersons afterwards. Block 3 provides the possibility of making these possible modifications. This block is not displayed by default. Block 4 gives more information about the composition of the sales of this selected category of product. It also allows to navigate these composing elements for more information.

5.2.3 Data features

Regarding the exploration part, results are displayed in hierarchies. In this application, three dimensions are defined: *customer* dimension, *product* dimension and *time* dimension. Since the sales are grouped by month, the time dimension is implicit. Only two dimensions are explored: customer dimension and product dimension. Each dimension is composed of several hierarchies. Figure 5.3 shows an example of sales hierarchy. In this example, we have four hierarchies shown in Figure 5.3(a) for the customer dimension: C1, C2, C3 and C4 and three hierarchies shown in Figure 5.3(b) for the product dimension: P1, P2 and P3. For instance, let the hierarchy C2 be a 3-level geographical distribution hierarchy. Customers are then analyzed by city for level 1, state for level 2 and country for level 3.



(a) Customer hierarchies (b) Product hierarchies

Figure 5.3: Hierarchy organization for dimension customer and dimension product

To show the sales trend, we need to store both 36-month historical data and 24-month predictive data. In addition, we need some meta-data to understand/interpret the displayed aggregated data. Hence, the latency to request all the information needed by an interface is too long to be accepted. To guarantee a quick access, materialized views are created, which contain all the information about customers, products, hierarchies, purchase dates and sales volumes. Actually, we have three materialized views (called $MV_{example}$ in the rest of this thesis) for three metrics: turnover, quantity and price. The main schema of $MV_{example}$ is shown in Table 5.1.

Attribute Name	Attribute Type
sales_key	bigint(20) unsigned
customer_key	bigint(20) unsigned
product_key	bigint(20) unsigned
customer_name	varchar(25)
product_name	varchar(25)
hierarchy_key	bigint(20) unsigned
parent_hierarchy_key	bigint(20) unsigned
dimension	char(2)
hierarchy_number	tinyint(3)
hierarchy_level	tinyint(3)
hierarchy_name	varchar(25)
achieved_sale_35	decimal(18,8)
... (other achieved sales)	decimal(18,8)
achieved_sale_0	decimal(18,8)
forecasting_sale_1	decimal(18,8)
... (other forecasting sales)	decimal(18,8)
forecasting_sale_24	decimal(18,8)
fiscal_year_sales_cumulation_1	decimal(18,8)
... (other fiscal year sales cumulations)	decimal(18,8)
fiscal_year_sales_cumulation_4	decimal(18,8)
... (other sales cumulations)	...

Table 5.1: Schema for one of the materialized views used for efficient data display

The materialized view is composed of five elements: identification for a sale (customer and product information), the sales volume during the last 36 months, the forecasting sales for the next 24 months, the hierarchy information, and some other aggregated sales information. We notice that the concept of time is expressed by the sales volume of each month and the sales cumulation in every fiscal year.

The size of a tuple in $MV_{example}$ can be easily estimated. We know that in MySQL[2] a *decimal* of 18 digits requires 9 bytes, a *bigint* 8 bytes and a *tinyint* 1 byte. So the total size of one tuple is 695 bytes if we take 25 bytes for the *varchar*(25). Then, we can estimate the size of the materialized view $MV_{example}$ if we know the volume of manipulated sales. For a user who achieves approximately 670000 sales per month, we get 465 650 000 bytes, which is about 445 MB. In reality, this view is much larger because organizations usually have some cumulations other than fiscal year cumulation, and there are also indexes employed to accelerate different queries.

[2]MySQL is the database management system used by Anticipeo to implement and manage forecasting data.

5.2.4 Main manipulations

We gather and analyze the workload of users to capture the main user manipulations of the application. According to the analyses, we notice four categories of typical queries, namely: (i) simple data retrieval (ii) updates on sales level (updates on raw data) (iii) calculation of aggregates for one level of one hierarchy (iv) data retrieval from multiple hierarchies.

Most of the time, users require information on sales at different levels on a single hierarchy. The data demanded have been precomputed and stored in materialized views, such as $MV_{example}$, so that the response to queries is fast. The **first** category of queries consists of simple data retrieval from materialized views.

During the salespersons' estimation of future sales, users need to modify forecasting results. The modification can occur on any level of any given hierarchy. When it happens, the system distributes the modification over the base sales level. The forecasting values of certain base sales are then modified. As other precomputed hierarchies are calculated by the same sales data, the system needs to recompute all the superior levels of hierarchies. These two manipulations constitute the second and the third categories of typical queries. Hence, the **second** and **third** categories of typical queries presented here are updates on sales level and construction of a certain level of a certain hierarchy on the fly, respectively.

The last manipulations through the interface consist in retrieving data across hierarchies of dimension. Unfortunately, it is almost impossible to build all the aggregates across hierarchies because of the tremendously high number of combinations. Even if, in this case, we consider only two hierarchies from two different dimensions at a time, this number is very important. Hence, the **fourth** category of queries consists in the computation of across-hierarchy information on the fly.

Example: an electrical appliance manufacturer needs sales information about a supermarket whose customer_key is equal to 500 and (s)he wants to display it in P2 hierarchy pattern (see figure 5.3(b)) in the turnover measure. The result will be displayed at level 2 of P2. The corresponding query for this demand is shown in figure 5.4. This query needs a projection that involves 64 sums, which consist of 36 sums for the historical sales (line 2), 24 sums for the forecasting sales (line 3) and 4 sums for the fiscal year sales cumulation (line 4). We need to do several self-join on the materialized view $MV_{example}$ to reach the level 2 of P2 (from line 6 to line 19).

```
1.  SELECT tab4.hierarchy_key, tab4.hierarchy_name,
2.         SUM(tab1.achieved_sale_35), ..., SUM(tab1.achieved_sale_0),
3.         SUM(tab1.forecasting_sale_1), ..., SUM(tab1.forecasting_sale_24),
4.         SUM(tab1.fiscal_year_sales_cumulation_1), ..., SUM(tab1.fiscal_year_sales_cumulation_4)
5.  FROM MV_example AS tab1, MV_example AS tab2, MV_example AS tab3, MV_example AS tab4
6.  WHERE tab1.hierarchy_level = -1          // level of detailed sales
7.  AND tab1.customer_key = 500
8.  AND tab1.dimension = 'p'                 // 'p' stands for product
9.  AND tab1.parent_hierarchy_key = tab2.hierarchy_key
10. AND tab2.dimension = 'p'
11. AND tab2.hierarchy_level = 0             // level for products
12. AND tab2.parent_hierarchy_key = tab3.hierarchy_key
13. AND tab3.dimension = 'p'
14. AND tab3.hierarchy_number = 2            // tree N°2
15. AND tab3.hierarchy_level = 1
16. AND tab3.parent_hierarchy_key = tab4.hierarchy_key
17. AND tab4.dimension = 'p'
18. AND tab4.hierarchy_number = 2
19. AND tab4.hierarchy_level = 2
20. GROUP BY tab4.hierarchy_key, tab4.hierarchy_name;
```

Figure 5.4: SQL query for the example of an electrical appliance manufacturer

5.2.5 Problem statement

There were performance problems with the Anticipeo application, an online user interactive application. For the main manipulations introduced in Section 5.2.4, only the execution time of the first category about simple retrieval of information is guaranteed with a quick access by using materialized views. The response time of other manipulations is difficult to be accepted by users. For example, across-hierarchical consultations (the fourth category of manipulations) on a database of 55 GB may take from 3 to 15 seconds depending on the hierarchical level from where users request information. Another example with an unacceptable latency is on the modification part. The time of a modification is mainly spent on the update of base sales and the update of the displaying hierarchical level. The evaluation on the same database of 55 GB takes approximately 35 seconds (4.3 seconds for base sales update and 31.4 seconds for the level construction) in the case of a modification at the level 2 of the hierarchy P2 (see Figure 5.3(b) for hierarchy information). The process of these manipulations should be optimized. However, at the beginning of our work, we did not know the source of the problems and thus how to solve the problems and improve the performance was not clear. In the following, we discuss different tracks that we took for the optimization and we show the achieved results.

5.3 Optimization guideline

The problem we face is a performance problem with respect to the exploration of a multidimensional database using a relational database. Bock and Schrage [BS02] have indicated that a number of factors affecting system response time are related to i) ineffective use of database management system tuning, ii) insufficient hardware platforms, iii) poor application programming techniques and iv) poor conceptual and physical database design.

In this section, we propose a guideline of optimization to diagnose the performance problem regarding these issues and to eventually provide better performance. Since the Anticipeo application is already operational, we should consider first solutions that require less effort for their implementation.

5.3.1 Hardware and application programming analysis

The first question we consider is whether the application is working in the appropriate environment. The execution environment refers to two levels: the hardware platform and the operating system supporting the utilization of the integrality of the hardware. The main technical characteristics of the server in use are: two Intel Quad core Xeon-based 2.4 GHz, 16 GB RAM and one SAS disk of 600 GB and 15000 rotations per minute. The operating system is a 64-bit Linux Debian system using EXT3 file system. Our focus in the audit is to inspect the hardware for three criteria: CPU, memory and disk I/O.

Regarding the application programming, we need to analyze whether the techniques of the application programming are efficient. An analytical result of time distribution (generalized and detailed) on different parts of the application is the objective of our audit. For the time distribution, we consider a separation of the execution time on the code itself and on the database including its access and the execution of queries. Especially, for queries that execute more than a given time threshold, we analyze their query execution plans. A query execution plan [Fri09] is the result of the query optimizer's attempt to calculate the most efficient way to implement the request represented by the query. Execution plans can tell how a query will be executed, or how a query was executed. They are, therefore, primary means of troubleshooting a poorly performing query. We can use the execution plan to identify the exact piece of SQL code that is causing the problem. For example, it may scan an entire table-worth of data while, with the proper index, it could simply backpack out only the rows needed. The scan method is displayed in the execution plan together with additional useful information.

5.3.2 Database management system configuration

We would like to know if the Data Base Management System (DBMS) is well tuned to support the workload of the application. In this context, Anticipeo uses MySQL to implement and manage forecasting data.

Figure 5.5 [Ora10] depicts the MySQL server architecture. With regard to the features of the Anticipeo application, the main MySQL system variables [Ora11] selected for the tuning are *innodb_buffer_pool_size*, *innodb_log_file_size*, *query_cache_size*, *innodb_flush_log_at_trx_commit*, *key_buffer_size*, etc.

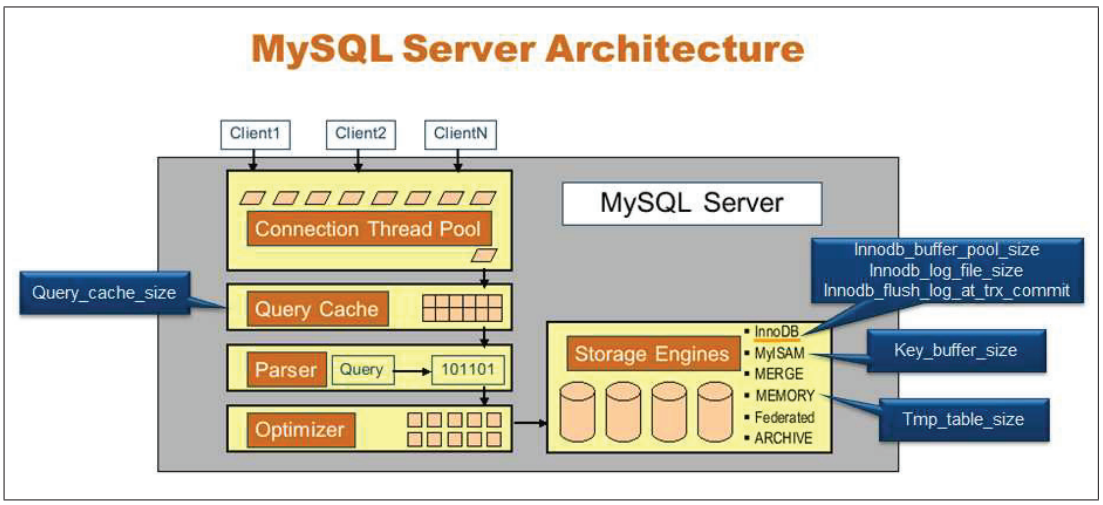


Figure 5.5: MySQL server architecture and main system variables selected for the tuning

- *innodb_buffer_pool_size*: memory buffer InnoDB to cache both data and indexes. The bigger the value is, the less disk I/O is needed.
- *innodb_log_file_size*: size of each log file in a log group. By default, a log group has two log files. The larger the value is, the less checkpoint flush activity is needed in the buffer pool which saves disk I/O.
- *query_cache_size*: cache for storing query results. The larger the value is, the more possibility to get the results directly from the cache without executing.
- *innodb_flush_log_at_trx_commit*: synchronization mode for transactions. 0 writes and synchronizes once per second. 1 forces synchronization to disk after every commit (ACID compliance). 2 writes to disk every commit but only synchronizes once per second.
- *key_buffer_size*: cache for storing indices. Increasing its value can get better index handling.

Some "blind" tuning has been done based on existing experimental results on different web services [Zaio7]. The actual configuration is 8 GB, 800 MB, 64 MB, 0 (zero), and 512 MB, respectively for the five system variables mentioned above. Additional benchmarking will be discussed in the following sections to determine whether the adopted configuration is efficient.

5.3.3 Additional materialized views

The fourth category of user manipulations presented in Section 5.2.4 is concerned with the visualization of achieved and forecasting results by considering different hierarchies from different dimensions. It is a typical data warehouse and OLAP problem (using relational databases). In this domain, one of the most used solutions is to select useful intermediate results and store them as materialized views. Many approaches have been proposed for the selection of materialized views.

The main idea is to use the greedy approaches [Gup97, HRU96, SDN98]. These solutions pre-process the most beneficial intermediate results in a limited-space hypothesis to avoid complex computations so as to enhance data access. Extensions of these solutions also consider the maintenance cost [BPT97] or large scale databases [GM99, KR99], or make the set of materialized views dynamic according to the workload [BKV06]. They have already been proved to bring significant improvements to data access.

In our case, we implement the classic greedy algorithm [HRU96]. This algorithm refers to the dependence relation in queries. We say $Q1 \preceq Q2$ if $Q1$ can be answered using only the results of $Q2$. We then say that $Q1$ is *dependent* on $Q2$. A lattice framework is used to express dependencies among queries (or views in this context). For elements a, b of the lattice, b is an *ancestor* of a, if and only if $a \preceq b$. Once the lattice is built, a space cost is associated to each element of the lattice. The cost is equal to the space occupied by the view from which the query is answered, which can also be expressed by the total number of tuples answers to the view. Without additional materialized views, only the raw data is stored in the database. All the views are evaluated on this table. The initial total cost of evaluating all the views is

$Cost = n * m - 1$,

where n is the number of views in the lattice and m is the number of tuples for the raw data. We do not count the view that contains only one tuple about the total result of all raw data.

We then compute the benefit of each view by considering how it can improve the total cost of evaluating views, including itself. Finally, the greedy algorithm selects the

most beneficial views to materialize with respect to the space limitation.

We describe this algorithm with an example. Consider the hierarchies illustrated in Figure 5.3. We take only the hierarchy C1 from the customer dimension and the hierarchies P1, P2 from the product dimension in this example. The hierarchies to explore are shown in Figure 5.6. Each element in this figure represents the condition by which the query performs the grouping operation. Hence, level 3 represents the sales of every single customer or every single product. Levels 1 and 2 represent the result of sales aggregated by the respective hierarchy level. Level 0 represents the aggregated result of all customers or all the products, which is also the coarsest level in the hierarchy. According to the dependency definition, we have $(None) \preceq (C11) \preceq (C12) \preceq (All_C)$, $(None) \preceq (P11) \preceq (All_P)$ and $(None) \preceq (P21) \preceq (P22) \preceq (All_P)$.

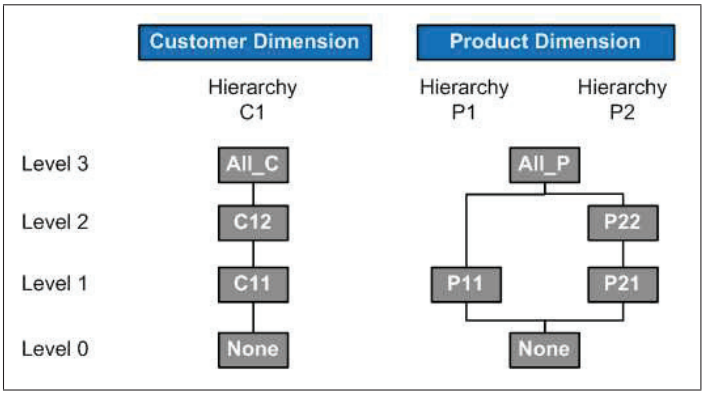


Figure 5.6: Illustration of hierarchy C1 and hierarchies P1, P2 in the notion of query dependence

We build the lattice by taking one element, which represents a grouping condition, from each dimension. The customer dimension has 4 elements and the product dimension has 5 elements. We then have 4 * 5 = 20 elements in the lattice shown Figure 5.7. In this lattice, we still can see the sketch of the hierarchies C1, P1 and P2. Blocs of different colors demonstrate the hierarchy C1. Inside every bloc, the hierarchies P1 and P2 are presented. We then evaluate the cost of every query which is presented by an element in the lattice. The cost is equal to the number of tuples returned by the query, in other words, the space occupied by the query if it is materialized.

Table 5.2 shows the cost of every view. Among these 20 views, the node All_C - All_P represents the sales information. This view should always be materialized. The view "None" is the sum of all the sales. There is only one tuple in this view and no view can be calculated from this view. We do not take this view into consideration when we search for the most beneficial views to materialize. We then compute the benefits of

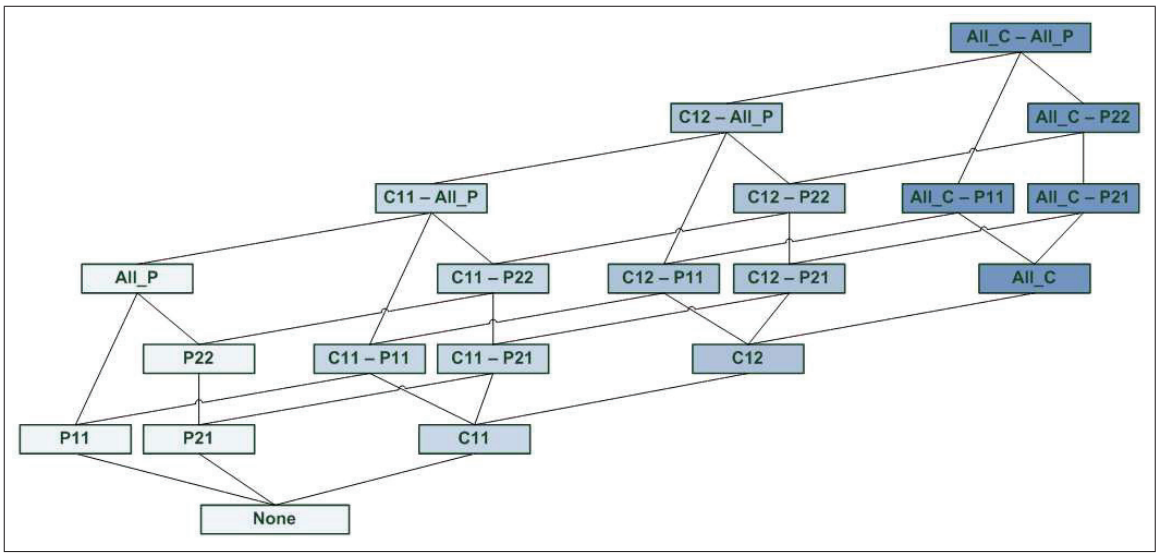


Figure 5.7: Illustration of lattice constructed by the dependence information of the hierarchy C1 and the hierarchies P1, P2

Node Name	Cost	Node Name	Cost
All_C - All_P	688419	C11 - All_P	18304
All_C - P11	55136	C11 - P11	4233
All_C - P22	60090	C11 - P22	679
All_C - P21	22193	C11 - P21	216
All_C	3519	C11	26
C12 - All_P	186935	All_P	4979
C12 - P11	4233	P11	42
C12 - P22	5629	P22	107
C12 - P21	1601	P21	25
C12	123	None	1

Table 5.2: Cost of every query/view in the example lattice

each of the remaining 18 views if they are subject to materialization. In this example, we will choose three views to materialize. The computation of benefits is then proceeded in three rounds. The benefits of views and the choice at the end of each round are depicted in Table 5.3.

According to the result of the computed benefits, we choose C12-All_P at the end of the first round; All_C-P11 after the second round and All_C-P22 after the third round. Compared to the initial evaluation based on the raw data containing 688 419 tuples for the 19 views, the total cost for evaluating all the views is reduced from 688 419*19=13 079 961 to 2 115 896 (the total cost of evaluating all the 19 views by 4 materialized views: raw data table and 3 selected views). This example shows that implementing a greedy algorithm could be a very interesting improvement for the user manipulation of

Node Number	Node Name	Choice 1	Choice 2	Choice 3
1	All_C - P11	5 066 264	2 057 360	
2	All_C - P22	5 026 632	2 017 728	2 017 728
3	All_C - P21	2 664 904	1 160 452	1 160 452
4	All_C	2 739 600	1 235 148	206 468
5	C12 - All_P	7 522 260		
6	C12 - P11	4 105 116	1 096 212	305 418
7	C12 - P22	4 096 740	1 087 836	1 087 836
8	C12 - P21	2 060 454	556 002	556 002
9	C12	2 064 888	560 436	165 039
10	C11 - All_P	6 701 150	1 686 310	1 159 114
11	C11 - P11	2 736 744	730 808	203 612
12	C11 - P22	2 750 960	745 024	745 024
13	C11 - P21	1 376 406	373 438	373 438
14	C11	1 376 786	373 818	110 220
15	All_P	6 834 400	1 819 560	1 292 364
16	P11	2 753 508	747 572	220 376
17	P22	2 753 248	747 312	747 312
18	P21	1 376 788	373 820	373 820

Table 5.3: Benefits of each view in three rounds and possible choices for the materialization at each round

across-hierarchy consultation.

We implemented the classic greedy algorithm on top of the Anticipeo application.

5.3.4 Database design

This application works on a large materialized view for results visualization. The advantage of merging all information in a same materialized view is obvious: we can omit time-consuming joins over tables containing millions of rows. However, it creates other problems, e.g., heavy work for queries which make several joins on the same materialized view to derive high-level aggregations. Our idea is to find a medium solution that can both avoid costly joins on different tables and reduce the time of self join of this view as well, which constitute the main source of time-consuming queries in the workload.

Two solutions are proposed to improve the database design. The first one can be quickly implemented, which does not require many modifications of the existing solution. The second one requires a lot of changes of the database design, but it brings a better performance compared to the first solution.

The first solution

The first solution is a naïve solution which modifies as minimum as possible the data design. The idea is to reduce the size of the materialized view which is involved in the

join operations. To do this, we break down the materialized view $MV_{example}$ into two. More precisely, we create a small view that contains the hierarchical information and a larger one that contains all the remaining attributes. Thus, the time-consuming self join is based only on the first small view, which can significantly reduce the amount of data involved in join operations. Table 5.4 and Table 5.5 describe the data schema of the two materialized views.

Attribute Name	Attribute Type
sales_key	bigint(20) unsigned
customer_key	bigint(20) unsigned
product_key	bigint(20) unsigned
customer_name	varchar(25)
product_name	varchar(25)
hierarchy_key	bigint(20) unsigned
parent_hierarchy_key	bigint(20) unsigned
dimension	char(2)
hierarchy_number	tinyint(3)
hierarchy_level	tinyint(3)
hierarchy_name	varchar(25)

Table 5.4: Schema for the first materialized view: hierarchical information

Attribute Name	Attribute Type
sales_key	bigint(20) unsigned
achieved_sale_35	decimal(18,8)
... (other achieved sales)	decimal(18,8)
achieved_sale_0	decimal(18,8)
forecasting_sale_1	decimal(18,8)
... (other forecasting sales)	decimal(18,8)
forecasting_sale_24	decimal(18,8)
fiscal_year_sales_cumulation_1	decimal(18,8)
... (other fiscal year sales cumulations)	decimal(18,8)
fiscal_year_sales_cumulation_4	decimal(18,8)
... (other sales cumulations)	...

Table 5.5: Schema for the second materialized view: sales information

Table 5.4 contains the customer and product information and their hierarchical information, while Table 5.5 contains the last 36 sales, the next 24 sales and the cumulations by year. After the fragmentation, the two materialized views have both the same number of tuples as $MV_{example}$, meanwhile the tuple size of the view shown in Table 5.4 is reduced. The tuple size of this view can be estimated in the same manner as the one introduced in Section 5.2.3 for the $MV_{example}$ (whose tuple size is estimated to 695 bytes).

The size of one tuple is 119 bytes if we take 25 bytes for the *varchar(25)*. We see the tuple size of the view involved in join operations is reduced from 695 bytes to 119 bytes.

We implemented this solution to see how much benefit we get by breaking down the materialized view into two.

The second solution

The second solution is to resort to the use of a star schema. The star schema and snowflake schema [VS99] (normalized version of a star schema) are widely used in the exploration of multidimensional data by OLAP tools. According to the literature, the star schema is more efficient than the snow schema in most of cases. The analyses of the data features and the manipulations of the Anticipeo application indicate that the star schema could be applied to our case to bring better performance.

Within the principles of the star schema, there are two types of tables: dimension tables and fact tables. In this case, we have three dimensions: *Customer* dimension, *Product* dimension and *Time* dimension. Three tables are created for each of the three dimensions respectively. Star schema employs denormalized tables for dimension tables, in which all the levels of every hierarchy are stored in its appropriate dimension table. One fact table is also created. This table contains foreign keys to each dimension and different measures: turnover, quantity and price. Figure 5.8 shows the star schema for the Anticipeo application.

Let us consider the query previously presented in Figure 5.4. This query is based on the large materialized views as $MV_{example}$. It returns the sales information about a supermarket whose customer_key is equal to 500 and displays the result at level 2 in product hierarchy 2 pattern in the measure of turnover. The equivalent query on the star schema is shown in Figure 5.9. Line 1 and line 2 perform a projection of some descriptive information and a sum of all sales. Line 3 to line 6 show the tables involved in this query and their relations. Line 7 gives the criteria of classification. We can see that the query is simplified. It targets three tables: fact_table, dim_product and dim_time. As the hierarchical information is already in the dim_product table, we do not need extra joins to get to level 2 of the hierarchy 2. The information at level 2 is directly reached by the attribute hierarchy2_level2 of the product dimension table. We have less tables involved and less join operations compared to the actual schema.

In order to evaluate this solution, we modify the data schema of the Anticipeo application to the one shown in Figure 5.8.

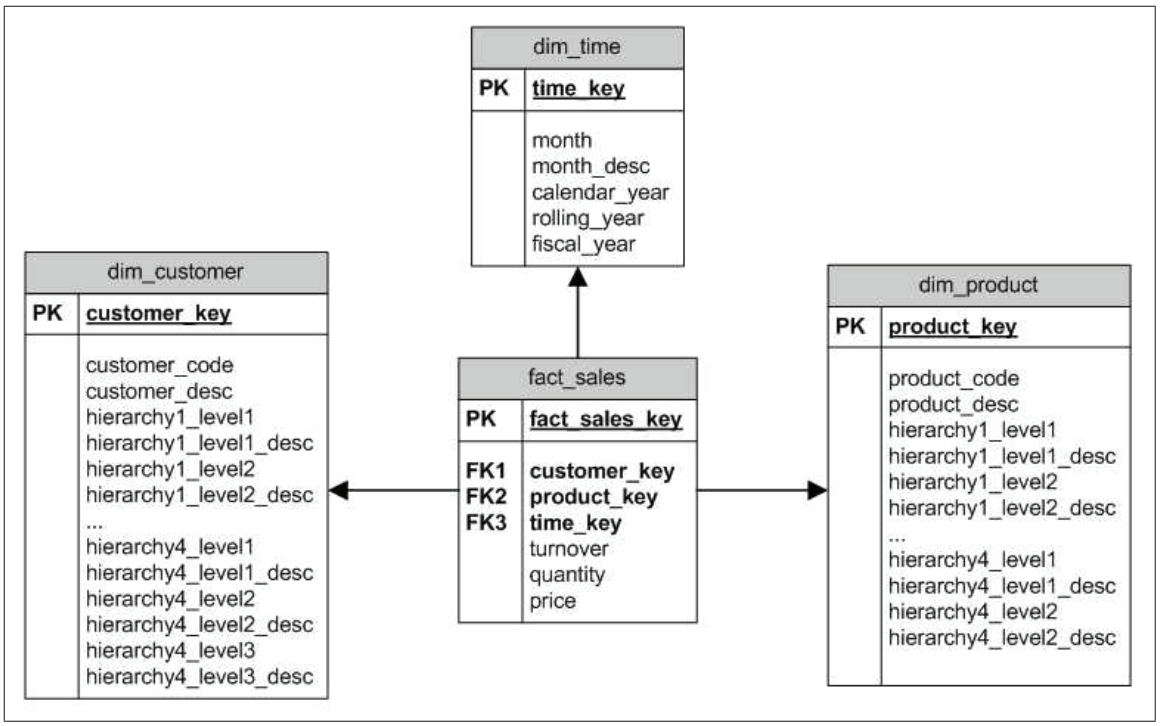


Figure 5.8: Star schema for the Anticipeo application

```
1.  SELECT fiscal_year, month,
2.          p.hierarchy2_level2, p.hierarchy2_level2_desc, SUM(turnover) AS total_turnover
3.  FROM fact_table f, dim_product p, dim_time t
4.  WHERE f.customer_key = 500
5.  AND f.product_key = p.product_key
6.  AND f.time_key = t.time_key
7.  GROUP BY fiscal_year, month, p.hierarchy2_level2;
```

Figure 5.9: SQL query for the example of an electrical appliance manufacturer on the star schema

5.4 Implementation and optimization results

Previous discussions target some main existing possibilities to improve the performance. In the following section, we present the experimental result of these solutions. Due to some specificities of sales forecasting systems, there may be some unexpected results after implementing these solutions. We then propose some suggestions for these situations. First of all, let us show some referential observations of the actual application on the experimental system described in Sections 5.3.1 and 5.3.2.

5.4.1 Observations on current implementation of the application

Experiments on different-size databases have been conducted to observe the behavior of the application implemented in the actual environment. We take the example of an

across-hierarchy visualization, which is described as the fourth category of user manipulation in Section 5.2.4. The corresponding query for this example is given in Figure 5.4. The descriptive information of different databases and the execution time of the visualization query are shown in Table 5.6.

	Database size (GB)	Sales number (SN)	$MV_{example}$ size (GB)	Across-hierarchy visualization response time (s) (AVRT)	AVRT/SN (ms)
DB1	55.8	679823	1.18	18.28	0.027
DB2	61.7	995211	2.42	21.89	0.021
DB3	68.9	1404267	3.27	43.31	0.031
DB4	81.7	2120115	4.92	41.65	0.020

Table 5.6: Experimental databases characteristics and results

The result shows that the query evaluation time is linear to the number of sales. This observation helps us to estimate the execution on a new database if we know the number of sales. These results also show one brake of the enterprise: in the case of a 55-GB database, it takes more than 18 seconds to answer an online user query. If the application considers larger databases, the response time can hardly be acceptable by the user.

5.4.2 Diagnosis of latency provenance

Two diagnoses are performed in this part to determine the latency provenance.

- Program level:
  We first conduct an analysis of the distribution of time in order to identify the latency provenance. The distribution is shown in Table 5.7.

  For the black box, the mathematical calculation of forecasting, the time spent on the execution of the application represents 30% of the total time, while the remaining 70% is used to access the database. Regarding the focus of our research, different kinds of visualizations and modifications (user interactive manipulations) during the navigation, the part of the time spent on the application represents less than 10% of the total time. The remaining time is due to the access to the database, the evaluation of queries, eventual updates of data and the return of query results. We can conclude here that the performance issues of the database might be the source of the performance problem.

User manipulation	Evaluation time	Time spent on application level	Time spent on DBMS level
Sales forecasts Calculation	> 2 hours	< 30%	> 70%
Simple data retrieval	< 10 ms	negligible	
Across-hierarchy visualization	> 60 sec	< 10%	> 90%
Modification of forecasts	> 50 sec	< 10%	> 90%

Table 5.7: Average time distribution on application level and on DBMS level for the execution of different user manipulations

We then extract slow queries captured by the database management system to determine whether these queries are well structured. We analyze the query execution plan for each query. The main purpose is to find whether the queries use indexes and if so, whether they use appropriate ones. Even if we noticed that for certain tables, there exist some unnecessary indexes, generally speaking, the query execution plan shows the slow queries are correctly optimized.

- Hardware level:
  In a second stage, we are interested in better knowing the system behavior.
  We use SAR (System Activity Report [3]), one of Linux performance monitoring tools to collect and analyze system activity information. The following paragraphs depict the observation on the actual system:
  - CPU
    CPU is idle during on average 89.42% of time.
    When the CPU works, it spends 7.38% of time to work on user processes, 0.31% of time on system processes and for 2.89% of time, it is idle during which there is an outstanding disk I/O request.
  - Memory
    Memory remains at a normal status without swapping activities.
    The measures of *swap in* and *swap out* in the report are both 0% during all the process time.
  - Disk
    Disk I/O also shows a normal activity.

[3]The SAR command collects, reports, or saves system activity information [God].

The average number of sectors read from the device is 2357 per second and the average number of sectors written to the device is 25668 per second during 5 hours of experimentation of simulating user's actions. 80% of users operations acting simple consultation demands about 3000 sectors read from the device per second (which represents about 1.5 MB/s data) and writes less than 100 sectors per second (about 0.04 MB/s). For the rest 20% usage, which refers to update or insert of data, we observe only 1% of time that we get more than 50000 sectors per second (24 MB/s), and never reach more than 100000 sectors (50 MB/s).
While according to the benchmarks [Sch07] performed on similar SAS disks, the minimum transfer rates of a SAS disk never fall below 68 MB/s, the previous result reveals that there are rare occurrences of device saturation.

The diagnosis leads us to the conclusion that the application performs well with the actual infrastructure for the program level as well as the hardware level. The latency observed on the application is not due to hardware issues, neither to program level issues. It is clearly due to the performance issues of the database.

5.4.3 Database management system configuration

According to the DBMS configuration recommendations [SZTZ08, Zai07], we determine some of the most important tuning variables in the case of the Anticipeo application. We set values above and below the current settings of these variables to see whether the current ones are optimized. The summarized results are shown in Table 5.8.

`innodb_buffer_pool_size`	Value	2 GB	4 GB	8 GB	12 GB
	Ratio	98.83%	100.22%	100.00%	97.60%
`innodb_log_file_size`	Value	128 MB	256 MB	800 MB	1 GB
	Ratio	99.86%	101.81%	100.00%	99.23%
`innodb_flush_log_at_trx_commit`	Value	0	1	2	Not
	Ratio	100.00%	101.32%	105.73%	relevant
`query_cache_size`	Value	32 MB	64 MB	128 MB	256 MB
	Ratio	102.94%	100.00%	111.76%	107.03%
`key_buffer_size`	Value	32 MB	128 MB	512 MB	1 GB
	Ratio	103.08%	103.82%	100.00%	100.01%
`tmp_table_size` (`max_heap_table_size`)	Value	32 MB	64 MB	128 MB	256 MB
	Ratio	109.87%	100.00%	102.21%	100.48%

Table 5.8: Performance comparison between different values chosen for each MySQL system variable

To evaluate the values of each variable, we give the ratio of the average execution

time[4] in comparison to current settings. The current setting of each variable is thus represented by 100%. We see that for most of the variables, the current value is already optimal. There are two variables *innodb_buffer_pool_size* and *innodb_log_file_size* that could improve the actual performance if they were defined with the setting of 12 GB and 1 GB, respectively.

We run an integral test with these new DBMS settings on the whole system and we obtain a result of 42% worse than the current setting. The reason is the excessively high value of *innodb_buffer_pool_size* for a machine that serves at the same time as a database and a web server. We reset this variable to 8 GB and then we run an integral test again. This time, we get a better performance of 7.32%.

The conclusion is that the system is running with an almost optimal configuration and by the DBMS configuration approach, we get 7.32% better performance.

5.4.4 Selection of materialized views

We implemented the classical greedy algorithm introduced in Section 5.3.3 on top of the database DB1 (information about DB1 is shown in Table 5.6). The dimension-hierarchies information about the database DB1 is illustrated in Figure 5.10.

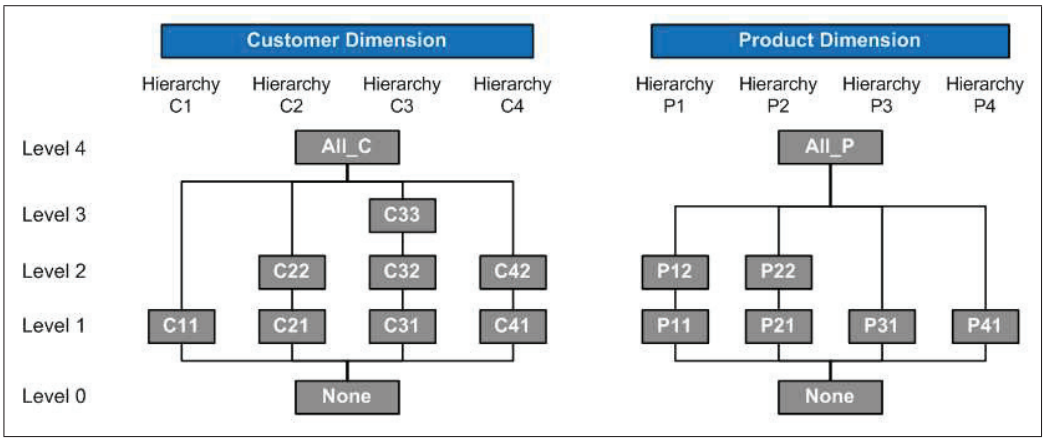


Figure 5.10: Illustration of dimension-hierarchies about the database DB1 of the Anticipeo application

The database DB1 has 2 dimensions: customer dimension and product dimension. Each dimension is composed of 4 hierarchies respectively. The customer dimension has a total of 10 elements and the product dimension has 8 elements. So the resulting lattice has a total of 10 * 8= 80 nodes. Among these 80 nodes, 79 views could be materialized to accelerate the across-hierarchy manipulations because the node presenting raw data is

[4]Here, the average execution time results from five executions of the same test. The objective is to get an average and more realistic estimation time.

already stored physically. We evaluate each view cost and search for the most beneficial views regarding their benefits. The gain can be calculated for different numbers of views allowed to be materialized (see Table 5.9).

	Number of tuples materialized	Total evaluating cost	Gain
no view materialized	0	53 706 017	
1 view materialized	59 561	35 718 419	33.49%
3 views materialized	141 115	16 396 767	69.47%
5 views materialized	240 841	8 037 913	85.03%
7 views materialized	433 146	6 495 694	87.91%

Table 5.9: Result of theoretical gain of implementing Greedy Algorithm

In this table, we see that when one additional view is allowed to be materialized, the view containing 59 561 tuples is chosen. The materialization of this view reduces the total cost of evaluating all the views from 53 706 017 to 35 718 419, which represents a gain of 33.49%. In the case that 3 views are allowed to be materialized, we reach a gain of 69.47%, nearly 70%. In terms of additional data, 141 115 tuples need to be materialized. Compared to the table of raw data containing 679 823 tuples, we need to materialize 20% more data. This is an interesting result for the Anticipeo application to improve the response time of the across-hierarchy manipulations.

Unfortunately, MySQL, the DBMS Anticipeo uses to manage data, supports neither materialized views nor automatic query rewriting by materialized views. We mention query rewriting because when we need to replace the raw data table by a materialized view to answer a query faster, a query rewriting process should be performed. Our results show the possibility of performance improvement if we had used another DBMS that supports automatic query rewriting. We could cite the Oracle relational database system.

5.4.5 Database schema modification

Evaluations described in Section 5.3.4 have been conducted on the same hardware environment and the actual DBMS configuration of the enterprise.

We instantiate the user manipulations presented in Section 5.2.4. Then we select the most important query of queries associated to each type of user manipulations to evaluate on both the actual data schema and the new data schema.

The first category of user manipulations is simple retrieval of precomputed aggregated sales. The execution time of these queries is only several milliseconds, which is

so small that we do not perform any evaluation on this first category. The second category is update of base sales. Query of type 1 is the query that performs this update on raw data. The third category consists of recomputation of aggregations requested. Query of type 2 is the query that computes the aggregated information displayed on the requested level of requested hierarchy after the update. Regarding the fourth category of user manipulations, we have initiated two queries of across-hierarchy consultation. Query of type 3 consists of a query which explores one customer hierarchy with the filter of a fixed product information while query of type 4 is a query which explores one product hierarchy with the filter of a fixed customer hierarchy. Every evaluation is performed several times to get an average result in order to reduce the impact of parameters that we cannot control.

Creating two separate materialized views

Table 5.10 shows the results performed using the actual schema and two separate materialized views. With the schema using two materialized views, an improvement of an average gain of 8.08%, 4.03%, 17.13% and 12.09%, respectively, is reached for the four query types.

Query	Schema	Average evaluation time (in seconds)	Gain
Query type 1	Actual schema	4.33	
	Two MVs schema	3.98	8.08%
Query type 2	Actual schema	15.70	
	Two MVs schema	15.07	4.03%
Query type 3	Actual schema	3.95	
	Two MVs schema	3.28	17.13%
Query type 4	Actual schema	3.25	
	Two MVs schema	2.86	12.09%

Table 5.10: Evaluation of queries on actual schema and on new data schema using two materialized views (Two MVs schema)

Using star schema

In a second stage, we make some more important changes to the data schema. We create three dimension tables, one fact table and then we integrate data in the fact table and dimension tables. In this case, there are three dimensions: *Customer*, *Product* and *Time*. The fact table stores foreign keys to dimension tables and sales measures, i.e., turnover, quantity and price.

Evaluations have been conducted on the actual schema and the star schema. We have obtained results shown in Table 5.11. The table reveals some unexpected results. Only

Query	Schema	Average evaluation time (in seconds)	Gain
Query type 1	Actual schema	4.33	
	Star schema	0.55	87.25%
Query type 2	Actual schema	15.70	
	Star schema	22.03	-40.30%
Query type 3	Actual schema	3.95	
	Star schema	4.45	-12.44%
Query type 4	Actual schema	3.25	
	Star schema	7.70	-136.80%

Table 5.11: Evaluation of queries on actual schema and on star schema

the query type 1 has some significant improvements. All other queries are less efficient using the star schema than using the actual schema. This result is due to the particularity of sales forecasting applications. In the forecasting applications, time is always requested by block, e.g., sales are displayed in months to show the sales trend from the past to the future in the Anticipeo application. In this case, the time dimension is rather an aggregation criterion than a condition of selection as in classical multidimensional queries. The fact of being an aggregation criterion, more precisely having *year* and *month* in *GROUP BY* in this query type, makes queries take more time to execute. Here, we need to design the star schema with adaptation to this particularity. We propose to "merge" the time dimension into the fact table. Then we have only two explicit dimension tables and the time dimension becomes implicit, which is hidden in the fact table.

Figure 5.11 depicts the modified star schema. In this schema, we do not have the time dimension anymore. In the fact table, we have 36 attributes for each measure of the 36 historical sales and 24 attributes for each measure of the next 24 sales. As there are three measures in this example, i.e., turnover, quantity and price, we have 36*3=108 attributes for historical sales and 24*3=72 attributes for predicting sales. In addition, cumulations by year are also added as attributes in this fact table. This pre-processing of grouping the sales in months and in years accelerates request response time.

The query example is shown in Figure 5.12 which performs the same tasks as the query shown in Figure 5.4 on the actual schema and the query shown in Figure 5.9 on the original star schema. Line 1 to line 4 perform a projection of some descriptive information, 36 sums on historical sales, 24 sums on forecasting sales and 4 sums on

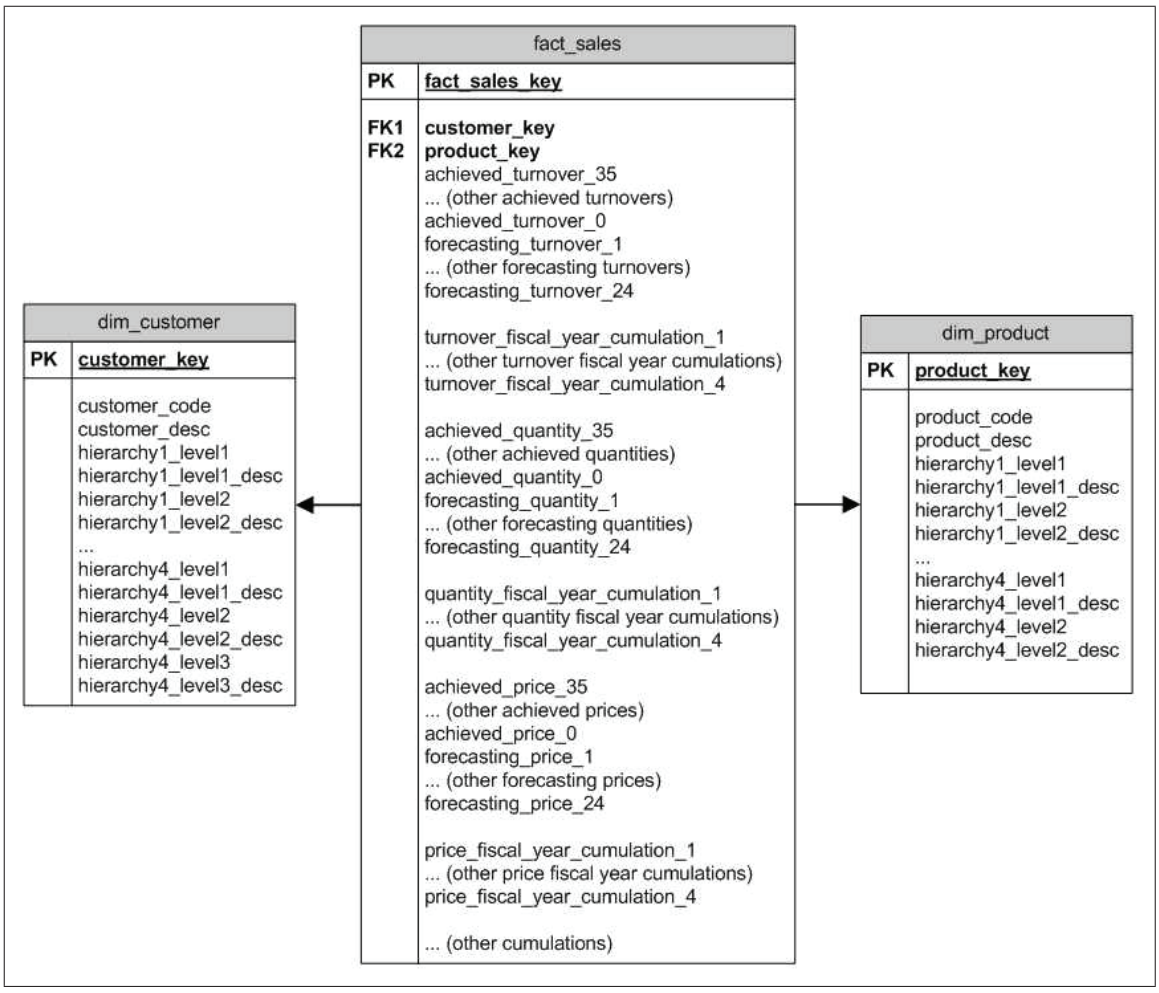


Figure 5.11: Modified star schema for the Anticipeo application

```
1.  SELECT p.hierarchy2_level2, p.hierarchy2_level2_desc,
2.         SUM(achieved_turnover_35), ... , SUM(achieved_turnover_0),
3.         SUM(forecasting_turnover1), ... , SUM(forecasting_turnover24),
4.         SUM(turnover_fiscal_year_cumulation_1), ... , SUM(turnover_fiscal_year_cumulation_4)
5.  FROM fact_table AS f, dim_customer AS c
6.  WHERE f.customer_key = 500
7.  AND f.product_key = p.product_key
8.  GROUP BY p.hierarchy2_level2;
```

Figure 5.12: SQL query for the example of an electrical appliance manufacturer on modified star schema

the annual cumulations. Line 5 to line 7 show the tables involved in this query and their relations. Line 7 gives the criteria of classification. This query returns the sales information about a supermarket whose customer_key is equal to 500 and displays the result at level 2 in product hierarchy 2 pattern in the measure of turnover. Compared to the query on the original star schema (Figure 5.9), it involves less tables, so naturally

less join operations and more important, less criteria in the *group by* operation.

We then compare the evaluation time of the 4 queries on this star schema without the time dimension, on the actual schema and on the original star schema. The results are shown in Table 5.12. This result reveals that the star schema without the explicit time dimension improves the response time of all the types of queries. Besides, it balances the inadequate utilization of the original star schema in this case by preprocessing the grouping by time (month and year).

Query (in seconds)	Schema	Average evaluation time	Gain
Query type 1	Actual schema	4.33	
	Without time DIM	2.32	46.44%
	With time DIM	0.55	87.25%
Query type 2	Actual schema	15.70	
	Without time DIM	9.90	36.94%
	With time DIM	22.03	-40.30%
Query type 3	Actual schema	3.95	
	Without time DIM	1.30	67.20%
	With time DIM	4.45	-12.44%
Query type 4	Actual schema	3.25	
	Without time DIM	1.32	59.36%
	With time DIM	7.70	-136.80%

Table 5.12: Evaluation of queries on actual schema, on star schema with time dimension and on star schema without time dimension

5.5 Overall optimization result

Facing with an operational application that has performance problems, we proposed a guideline with different optimization issues. We search for optimization tracks from the hardware, the DBMS tuning, the application programming and the conceptual and physical database design.

The observations on the activities of the hardware show that the application is well supported by the actual hardware platform. With an effort to the DBMS tuning, we get an improvement of approximately 7%. The diagnoses of the latency provenance reveal that the programs of the application are already correctly optimized. Among these optimization tracks, an adequate database design to a specific application seems crucial. It can significantly improve the performance. Based on the actual schema, we propose a solution of additional materialized views using the traditional greedy algorithm. This solution is shown to bring 70% better performance for one time-consuming type of user

manipulations when only 20% more information compared to the raw data is materialized. Despite the fact that this solution is not applicable on the MySQL DBMS, it stays an interesting suggestion for future database design of Anticipeo and for users of other DBMSs. The most significant improvement is the adoption of the star schema for this application. With this new data schema adapted to features of sales forecasting systems, which is a star schema without explicit time dimension, the visualization issues of the Anticipeo application are well supported. The visualization of a single hierarchy is almost immediate thanks to precomputed and stored aggregates at all levels of all hierarchies. The visualization of several hierarchies is now possible in less than 1.3 second on a database of 50 GB.

However, the specific operations of predictive data modifications on sales forecasting applications remain to be optimized. In order to display an interface of navigation like the one shown in Figure 5.2, two levels of aggregates should be recomputed. The first one is the aggregate involved in the modification (block 1, block 2 and block 3 of Figure 5.2). The second one is the level below (block 4 of Figure 5.2). At this level, the aggregate components of the modified aggregate should be recomputed. We need to execute one time the query type 1 in order to update raw data and two times the query type 2 in order to recompute the two levels of aggregates (see query type description in Section 5.4.5). On the example database of 50 GB, the evaluation of each query type on the new star schema without time dimension is shown in Table 5.12. A predictive data modification takes more than 22.12 $(= 2.32 + 9.90 + 9.90)$ seconds. This response time is not acceptable for interactive utilization of the application. That is why we launched a research work to this specific feature of sales forecasting systems and proposed our PAM and PAM II algorithms.

5.6 Recommendations

With respect to observations, benchmarking and experiments of our proposals, we propose some recommendations to improve the performance of the Anticipeo application. The actual infrastructure of the hardware platform is suitable for existing customer databases of Anticipeo. There is no need to switch to more powerful hardwares except for future eventually large scale customer databases. The database management system and application programs are correctly configured and optimized. Obviously, it seems interesting to change the actual data schema to a star schema without time dimension, which is our strong suggestion in this case. The change of data schema could lead to a series of modifications on actual programs. The schema in the first solution of data de-

sign with two separate materialized views could be applied as the transit schema before the system toggles to a completely new schema. Finally, to improve the performance of aggregate modification propagations, new programs should be developed using the PAM algorithm I or II. PAM II is recommended if there are enough physical spaces for additional data materialization. Otherwise, PAM I is an alternative.

Chapter 6

Conclusion and future work

This work is based on a real world and operational application tat displays some performance problems. Anticipeo, a sales forecasting application, provides its customers with satisfying sales forecasts precision, but the performance problem prevents the company from further collaborations with customers working on large databases. Improving the performance of the application is the main objective of the CIFRE thesis. An audit had been conducted on the application to diagnose the latency provenance. It covered different angles of possible latency provenance such as hardware platform, database management system tuning, application programming and database physical and conceptual design. Once having identified the latency, which is mainly related to the database, we proposed some underlying solutions: database management system better tuning, adding materialized views and revising the database design. These technical solutions helped the application to achieve a better performance. However, the problem of efficiently propagating an aggregate modification through a dimension-hierarchy structure still remains. Existing research work did not investigate this problem. We propose an algorithm named PAM to manipulate and solve this issue. The PAM algorithm identifies the raw tuples to update, calculates the delta of each raw tuple, then identifies and updates aggregates by raw tuples and the calculated delta. Moreover, an extended version of the algorithm is proposed to bring better performance by using additional semantics (i.e. dependencies). The efficiency of the PAM algorithm and its extension is proved by experiments on the data of the Anticipeo application.

At the end of the thesis, the performance of the Anticipeo application has been significantly improved. Most of the interactive user manipulations became almost immediate instead of seconds/minutes of waiting. There is some work to be completed afterwards. The new database conceptual design will be applied to the whole presentation layer of

the application. Programs related to the presentation layer need to be updated. New algorithms defined on this data schema will be implemented and will replace former solutions.

6.1 Contributions

The scientific contributions of this work can be summarized as follows:

1. The first contribution is the proposal of a guideline of optimizations for applications suffering from performance troubles. This is a general guideline considering four main performance issues. We have shown the process of the guideline in the case of the Anticipeo application. First, diagnosis on hardware platform, programming and SQL query execution are carried out. Second, database management system tuning takes place. Once identifying the bottleneck of the system, modifications on this part are implemented to remove the bottleneck. There can be several modifications to accomplish different purposes. Finally, measurement is performed again to validate the modifications.

2. The second contribution is the proposal of an algorithm which handles the problem of efficiently propagating an aggregate modification through a dimension-hierarchy structure. This algorithm is capable of identifying raw tuples, which are impacted by the modification of the aggregate. It calculates the delta of each raw tuple involved regarding the predefined decomposition rules. Other aggregates impacted by the modification are identified and they are updated according to the delta of raw tuples. The algorithm is shown to be more efficient than the current solution (which destroys and reconstructs all the aggregates from scratch) in most cases. The PAM algorithm can be applied to most distributive and algebraic aggregate functions, such as SUM, COUNT and AVG, although the MIN and MAX functions need some additional materialization information.

3. The third contribution is the proposal of an extension of the PAM algorithm. The dimension structure of the data warehouse is determined from the beginning of the database design. The relationship between aggregates and raw tuples is known. In the extension of the algorithm, this relationship, so called dependency of aggregates on raw tuples, is materialized. This provides a direct access from both sides: from an aggregate to its composing raw tuples and from a raw tuple to its contributing aggregates. Hence, the direct access enables a better performance of the extension.

Like the original algorithm, the extension is also applicable to most distributive and algebraic aggregate functions. In term of efficiency, the extension is shown to perform better compared to the current solution and the original PAM algorithm. Its scalability is also better than the original algorithm in spite of the fact that a remarkable amount of physical storage is required to ensure the efficiency.

6.2 Future work

For further work, we have identified some tracks:

1. We will take into consideration the scalability of the PAM algorithm and its derivative. We have shown in this work that the algorithms are polynomial in time. We are facing performance issues when databases reach a certain size. Our idea is essentially to decrease the number of raw tuples, which is the main criterion of time complexity. In order to do this, we will classify raw tuples into groups. Our algorithms will then handle groups of tuples instead of raw tuples. In this case, the time complexity will depend on the number of groups (which we expect to be less than the current one). For our algorithms, we need to identify dependencies between aggregates and groups and then adapt our algorithms to be able to manipulate groups instead of raw tuples.

   The research issue in this perspective is how to build significant groups. A group should consist of tuples that appear frequently in the same aggregates, which would allow us to calculate and store differential values for these groups of tuples. The concept of maximal rectangles in formal concept analysis [GW99, CR04, Wil09] seems to be possible directions. We will consider the raw tuples as the set of objects and the aggregates as the set of attributes. The maximal rectangles refer to our group of tuples.

2. Nevertheless, central databases have their limits. When dealing with very large databases, we should consider distributed solutions. Our algorithms should be revised to be applied to distributed databases [OV11, CFK*99]. These algorithms can be implemented on single machines, which represent the atomic units of a distributed database. Solutions for aggregating results from different machines should be provided to compute the final results.

3. The third perspective is to evaluate the performance of the propagation of the aggregate-based modification in a column-oriented database [SAB*05, OOCR09].

In a column-oriented database, tables are stored as sections of columns of data rather than as rows of data, as in most relational database management systems. Column-oriented databases show their efficiency when new values of a column are supplied for all the rows at once, because that column data can be written efficiently and replace old column data without touching any other columns for the rows. This sounds to be a possible infrastructure for forecasting systems. When modifying the result of an aggregate, only the values in the column need to be updated. Storing data in columns seems to be more appropriate in this case. Our objective is to evaluate this forecasting system on a column-oriented system and eventually to reveal new research issues when column-oriented systems face this update intended application.

Bibliography

[ACGG04] Armstrong S., Collopy F., Graefe A., Green K. C.: Answers to frequently asked questions (FAQ) in forecasting, 2004. http://repository.upenn.edu/marketing_papers/156/. Last updated 24 November 2004. 3, 12

[Ack89] Ackoff R. L.: From data to wisdom. *Journal Of Applied Systems Analysis 16*, 1 (1989), 3–9. 13

[AG05] Armstrong J. S., Green K. C.: *Demand Forecasting: Evidence-based Methods*. Monash Econometrics and Business Statistics Working Papers 24/05, Monash University, Department of Econometrics and Business Statistics, 2005. 12

[Ale89] Alexander E. R.: Sensitivity analysis in complex decision models. *Journal of the American Planning Association 55*, 3 (1989), 323–333. 23

[App65] Applebaum W.: Can store location research be a science? *Economic Geography 41*, 3 (July 1965), 234–237. 57

[Arm00] Armstrong J. S.: *Principles of Forecasting: A Handbook for Researchers and Practitioners*. International Series in Operations Research & Management Science. Kluwer Academic, 2000. xiii, 12, 13

[Ars] Arshad A.: Storage modes in ssas (molap, rolap and holap). http://www.sql-server-performance.com/2009/ssas-storage-modes/. 19

[AV98] Adamson C., Venerable M.: *Data warehouse design solutions*. Wiley Computer Publishing. Wiley, 1998. 21

[Bel02] Bellahsene Z.: Schema evolution in data warehouses. *Knowledge and Information Systems 4*, 3 (2002), 283–304. 22

[BHS*98] Ballard C., Herreman D., Schau D., Bell R., Kim E., Valencic A.: *Data Modeling Techniques for Data Warehousing*. IBM Corp., 1998. 20

[BKV06] Biren S., Karthik R., Vijay R.: A hybrid approach for data warehouse view selection. *International Journal of Data Warehousing and Mining 2* (2006), 1–37. 66

[BMBT03] Body M., Miquel M., Bedard Y., Tchounikine A.: Handling Evolutions in Multidimensional Structures. In *International Conference on Data Engineering (ICDE)* (2003), pp. 581–591. 22

[bo] Business Objects. Http://www.sap.com/sapbusinessobjects/. 17

[BPT97] Baralis E., Paraboschi S., Teniente E.: Materialized views selection in a multidimensional database. In *Very Large Data Bases (VLDB)* (1997), pp. 156–165. 66

[BS02] Bock D. B., Schrage J. F.: Denormalization guidelines for base and transaction tables. *SIGCSE Bulletin 34*, 4 (Dec. 2002), 129–133. 64

[CCS93] Codd E. F., Codd S. B., Salley C. T.: *Providing OLAP (On-Line Analytical Processing) to User-Analysis: An IT Mandate*, vol. 32. Codd & Date, Inc., 1993. 6, 18

[CD97] Chaudhuri S., Dayal U.: An overview of data warehousing and olap technology. *SIGMOD Record 26*, 1 (Mar. 1997), 65–74. xiii, 13, 14, 15, 17

[CFK*99] Chervenak A., Foster I., Kesselman C., Salisbury C., Tuecke S.: The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of network and computer applications 23* (1999), 187–200. 87

[Cha00] Chaman J. L.: Which forecasting model should we use? *Journal of Business Forecasting Methods & Systems 19*, 3 (2000), 2. 57

[CLR04] Chen S., Liu B., Rundensteiner E. A.: Multiversion-based view maintenance over distributed data sources. *ACM Transactions on Database Systems 29*, 4 (2004), 675–709. 22

[cog] Cognos. Http://www-01.ibm.com/software/analytics/cognos/. 17

[CR04] Carpineto C., Romano G.: *Concept Data Analysis: Theory and Applications*. Wiley, 2004. 87

[CR05] Chen S., Rundensteiner E. A.: Gpivot: Efficient incremental maintenance of complex rolap views. In *International Conference on Data Engineering (ICDE)* (2005), pp. 552–563. 22

[csv] Definition of CSV. Http://searchsqlserver.techtarget.com/definition/comma-separated-values-file. 58

[eSa] eSalesTrack: Sales forecasting. http://www.esalestrack.com/Article/sales-forecasting.html. Accessed on 13 August 2011. 4

[Eur] European Centre for Medium-Range Weather Forecasts:. http://www.ecmwf.int/. Accessed on 18 January 2012. 2

[Fen11] Feng H.: Performance problems of forecasting systems. In *Advances in Databases and Information Systems (ADBIS) (2)* (2011), pp. 254–261. 40

[FLHD11] Feng H., Lumineau N., Hacid M.-S., Domps R.: Data management in forecasting systems: Case study - performance problems and preliminary results. In *Bases de données Avancées (BDA)* (2011). 40

[FLHD12] Feng H., Lumineau N., Hacid M.-S., Domps R.: Hierarchy-based update propagation in decision support systems. In *Database Systems for Advanced Applications (DASFAA) (2)* (2012), pp. 261–271. 30

[For] ForecastPro:. http://www.forecastpro.com/. 5

[Fri09] Fritchey G.: *SQL Server Execution Plans*. Simple Talk Publishing, 2009. 64

[FSd] Definition of Forecasting System. Http://www.businessdictionary.com /definition/forecasting-system.html. Accessed on 24 August 2011. 1, 2

[GBGB05] Gonzalez-Benito O., Gonzalez-Benito J.: The role of geo-demographic segmentation in retail location strategy. *International Journal of Market Research 47* (2005), 295–305. 57

[GCB*97] Gray J., Chaudhuri S., Bosworth A., Layman A., Reichart D., Venkatrao M., Pellow F., Pirahesh H.: Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Mining and Knowledge Discovery 1*, 1 (1997), 29–53. 36

[Gil06] Gilmore Lewis LLC.: How to Develop an Effective Sales Forecast. White paper, Gilmore Lewis LLC., July 2006. Available online (9 pages), http://www.gilmorelewis.com/storage/salesforecast.pdf. 4

[GM95] Gupta A., Mumick I. S.: Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin 18*, 2 (1995), 3–18. 21

[GM99] Gupta H., Mumick I. S.: Selection of views to materialize under a maintenance cost constraint. In *International Conference on Database Theory (ICDT)* (1999), pp. 453–470. 66

[GMD] GMDH Shell:. http://www.gmdhshell.com/. 5

[GMRR01] Gupta A., Mumick I. S., Rao J., Ross K. A.: Adapting materialized views after redefinitions: techniques and a performance study. *Information Systems 26*, 5 (2001), 323–362. 22

[Goa] Definition of goal seeking. Http://www.answers.com/topic/goal-seeking#ixzz1zoeLY5U7. Accessed on 20 June 2012. 23

[God] Godard S.: sar(1) - linux man page. http://linux.die.net/man/1/sar. Accessed on 15 December 2010. 74

[GR06] Greistorfer P., Rego C.: A simple filter-and-fan approach to the facility location problem. *Computers & Operations Research 33* (2006), 2590–2601. 57

[GRP06] Golfarelli M., Rizzi S., Proli A.: Designing what-if analysis: towards a methodology. In *International Workshop on Data Warehousing and OLAP (DOLAP)* (2006), pp. 51–58. 23

[Gup97] Gupta H.: Selection of views to materialize in a data warehouse. In *International Conference on Database Theory (ICDT)* (1997), pp. 98–112. 66

[GW99] Ganter B., Wille R.: *Formal concept analysis: mathematical foundations*. Springer, 1999. 87

[HMV99] Hurtado C. A., Mendelzon A. O., Vaisman A. A.: Maintaining data cubes under dimension updates. In *International Conference on Data Engineering (ICDE)* (1999), pp. 346–355. 22

[Hof93] Hoffman M. S.: *The world almanac and book of facts 1993*. Pharos Books; 125 Annv edition, 1993. 12

[HRU96] Harinarayan V., Rajaraman A., Ullman J. D.: Implementing data cubes efficiently. In *Special Interest Group on Management Of Data (SIGMOD)* (1996), pp. 205–216. 19, 66

[Hyn08] Hyndman R.: *Forecasting with exponential smoothing: the state space approach*. Springer series in statistics. Springer, 2008. 57

[IBM07] IBM Press release: IBM and Singapore's Land Transport Authority Pilot Innovative Traffic Prediction Tool, August 2007. http://www-03.ibm.com/press/us/en/pressrelease/21971.wss. Accessed on 18 January 2012. 2

[IIS01] Inmon W., Imhoff C., Sousa R.: *Corporate Information Factory*. Wiley Computer Publishing. John Wiley & Sons, 2001. 14

[Inm05] Inmon W. H.: *Building the Data Warehouse (3rd Edition)*. John Wiley & Sons, Inc., New York, NY, USA, 2005. 6, 14

[JD07] Johnston J., Dinardo J. E.: *Econometric Methodes (4th Edition)*. McGraw-Hill, New York, NY, USA, 2007. 12

[KR99] Kotidis Y., Roussopoulos N.: Dynamat: A dynamic view management system for data warehouses. In *Special Interest Group on Management Of Data (SIGMOD)* (1999), pp. 371–382. 66

[KR02] Kimball R., Ross M.: *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling (Second Edition)*. John Wiley & Sons, Inc., 2002. 6, 14, 17

[Kus99] Kusters B.: The forecasting report: A comparative survey of commercial forecasting systems. *IT Research* (November 1999). 12

[KWW02] Kuo R. J., Wu P., Wang C. P.: An intelligent sales forecasting system through integration of artificial neural networks and fuzzy neural networks with fuzzy weight elimination. *Neural Network 15*, 7 (Sept. 2002), 909–925. 57

[LBYD08] Lv H., Bai X., Yin W., Dong J.: Simulation based sales forecasting on retail small stores. In *Winter Simulation Conference (WSC)* (2008), pp. 1711–1716. 57

[LL06] Leung C. K.-S., Lee W.: Efficient update of data warehouse views with generalised referential integrity differential files. In *British National Conference on Databases (BNCOD)* (2006), pp. 199–211. 22

[LSK01] Lee K. Y., Son J. H., Kim M. H.: Efficient incremental view maintenance in data warehouses. In *International Conference on Information and Knowledge Management (CIKM)* (2001), pp. 349–356. 21

[MD96] Mohania M., Dong G.: Algorithms for adapting materialised views in data warehouses. In *International Symposium on Cooperative Database Systems for Advanced Applications (CODAS)* (1996), pp. 309–316. 22

[mica] MicroStrategy. Http://www.microstrategy.com/. 17

[Micb] Microsoft: Partition storage modes and processing. SQL Server Books Online (BOL) / MSDN. http://msdn.microsoft.com/en-us/library/ms174915.aspx. 19

[Min] Ministry of Transport, Singapore Government: Electronic road pricing. http://app.mot.gov.sg/Land_Transport/Managing_Road_Use /Electronic_Road_Pricing.aspx. Accessed on 18 January 2012. 2

[MJC] MJC2:. http://www.mjc2.com/demand-forecasting-software.htm. 5

[MK97] Mentzer J. T., Kahn K. B.: State of sales forecasting systems in corporate america. *Journal of Business Forecasting* (1997), 6–13. 57

[MMSG98] Moon M. A., Mentzer J. T., Smith C. D., Garver M. S.: Seven keys to better forecasting. *Business Horizons* (1998), 44–52. 57

[Mov] Definition of Moving Average. http://www.businessdictionary.com/definition/moving-average.html. Accessed on 06 June 2012. 12

[MQM97] Mumick I. S., Quass D., Mumick B. S.: Maintenance of data cubes and summary tables in a warehouse. In *Special Interest Group on Management Of Data (SIGMOD)* (1997), pp. 100–111. 22

[MWH98a] Makridakis S. G., Wheelwright S., Hyndman R.: *Forecasting: methods and applications*. Wiley series in management. Wiley, 1998. 57

[MWH98b] Makridakis S. G., Wheelwright S. C., Hyndman R. J.: *Forecasting: Methods And Applications, 3rd edition*. Wiley series in management. Wiley, 1998. 12

[Nat] National Oceanic and Atmospheric Administration, US Government: National Centers for Environmental Prediction. http://www.ncep.noaa.gov/. Accessed on 18 January 2012. 2

[Nat11] National Institute of Standards and Technology (NIST) of America: Engineering Statistics Handbook, 2011. http://www.itl.nist.gov/div898/handbook/index.htm. Last updated 1 April 2011. 12

[NLR98] Nica A., Lee A. J., Rundensteiner E. A.: The cvs algorithm for view synchronization in evolvable large-scale information systems. In *International Conference on Extending Database Technology (EDBT)* (1998), pp. 359–373. 22

[OLA97] OLAP Council: OLAP Council White Paper, 1997. http://www.olapcouncil.org/research/whtpaply.htm. 18

[OLT] Definition of OLTP. Http://www.businessdictionary.com/definition/online-transaction-processing-OLTP.html. Accessed on 11 May 2012. 18

[OM10] O'Brien J., Marakas G.: *Management Information Systems*. McGraw-Hill Companies,Incorporated, 2010. 23

[OOCR09] O'Neil P., O'Neil E., Chen X., Revilak S.: Performance evaluation and benchmarking. Springer-Verlag, Berlin, Heidelberg, 2009, ch. The Star Schema Benchmark and Augmented Fact Table Indexing, pp. 237–252. 87

[Ora10] Oracle: MySQL performance tuning: top 5 tips. MySQL Webinar, April 2010. 65

[Ora11] Oracle: Server system variables, February 2011. http://dev.mysql.com/doc/refman/5.0/en/server-system-variables.html. 65

[Ora12] Oracle: Materialized view concepts and architecture, 2012. http://docs.oracle.com/cd/B10501_01/server.920/a96567/repmview.htm. Accessed on 01 June 2012. 21

[OV11] Ozsu M. T., Valduriez P.: *Principles of Distributed Database Systems*. Springer, 2011. 87

[Pan97] Pannell D. J.: Sensitivity analysis of normative economic models: theoretical framework and practical strategies. *Agricultural Economics 16*, 2 (May 1997), 139–152. 23

[Per] Calculate Percent Difference. Http://www.buzzle.com/articles/calculate-percent-difference.html. Accessed on 15 June 2012. 43

[Phi88] Philippakis A. S.: Structured what if analysis in dss models. In *Annual Hawaii International Conference on Decision Support and Knowledge Based Systems Track* (Los Alamitos, CA, USA, 1988), IEEE Computer Society Press, pp. 366–370. 23

[PMN*03] Petropoulos C., Metaxiotis K., Nikolopoulos K., Assimakopoulos V., Patelis A.: Sftis: a decision support system for tourism demand forecasting. *Journal of Computer Information Systems 44*, 1 (2003), 21–32. 3

[Pow02] Power D. J.: Is a Data Warehouse a DSS? What is a star schema? How does a snowflake schema differ from a star schema? *DSS News 3*, 4 (2002). 14

[Pow04] Power D. J.: *Decision Support Systems: Frequently Asked Questions*. iUniverse, 2004. 23

[Pow10] Power D. J.: What is analytical processing?, February 2010. http://dssresources.com/faq/index.php?action=artikel&id=201. Accessed 06 June 2012. 17

[PVSV07] Papastefanatos G., Vassiliadis P., Simitsis A., Vassiliou Y.: What-if analysis for data warehouse evolution. In *Data Warehousing and Knowledge Discovery (DaWaK)* (2007), pp. 23–33. 23

[Rei31] Reilly W. J.: The law of retail gravitation. *New York: published by the author* (1931). 57

[RW99] Rowe G., Wright G.: The Delphi technique as a forecasting tool: issues and analysis. *International journal of forecasting 15* (October 1999), 353–375. 12

[SAB*05] Stonebraker M., Abadi D. J., Batkin A., Chen X., Cherniack M., Ferreira M., Lau E., Lin A., Madden S., O'Neil E., O'Neil P., Rasin A., Tran N., Zdonik S.: C-store: a column-oriented dbms. In *Very Large Data Bases (VLDB)* (2005), VLDB Endowment, pp. 553–564. 87

[sas] SAS. Http://www.sas.com/. 17

[SC09] Schumaker R. P., Chen H.: A quantitative stock prediction system based on financial news. *Information Processing & Management 45* (Sept. 2009), 571–583. 3

[Sch07] Schmid P.: Seagate Savvio 15k.1: 15,000 RPM, October 2007. http://www.tomshardware.com/reviews/sas-hard-drives,1702-4.html. Accessed on 28 February 2011. 75

[Sci12] Science Buddies: Variance & standard deviation, 2012. http://www.sciencebuddies.org/science-fair-projects/project_data _analysis_ variance_std_deviation.shtml. Accessed on 03 April 2012. 46

[SDN98] Shukla A., Deshpande P., Naughton J. F.: Materialized view selection for multidimensional datasets. In *Very Large Data Bases (VLDB)* (1998), pp. 488–499. 66

[SFd] Definition of Sales forecast. Http://www.businessdictionary.com/definition /sales-forecast.html. Accessed on 15 August 2011. 4

[spr] SpreadSheet. Http://office.microsoft.com/en-us/excel/. 17, 58

[std] Definition of Standard deviation. Http://www.stats.gla.ac.uk/steps /glossary/presenting_data.html#standev. 45

[SZTZ08] Schwartz B., Zaitsev P., Tkachenko V., Zawodny J. D.: *High Performance MySQL, Second Edition*. O'Reilly Media, 2008. 75

[Utl02] Utley C.: Designing the star schema database. *Data Warehousing Resources* (2002), 1–13. 21

[var] Definition of Variance. Http://www.stats.gla.ac.uk/steps/glossary /probability_distributions.html#variance. 45

[Vir09] Virtual Advisor, Inc.: Conduct a sales forecast, 2009. http://www.vainteractive.com/inbusiness/editorial/sales/ibt/sales_fo.html. Accessed on 25 January 2012. 56

[VS99] Vassiliadis P., Sellis T. K.: A survey of logical models for olap databases. *SIGMOD Record 28*, 4 (1999), 64–69. 71

[Wil09] Wille R.: Restructuring lattice theory: An approach based on hierarchies of concepts. In *International Conference on Formal Concept Analysis (ICFCA)* (2009), Springer-Verlag, pp. 314–339. 87

[Win11] Winston W.: *Microsoft Excel 2010: Data Analysis and Business Modeling*. Microsoft Press Series. Microsoft Press, 2011. 23

[Zai07] Zaitsev P.: Innodb performance optimization basics, November 2007. http://www.mysqlperformanceblog.com/2007/11/01/innodb-performance-optimization-basics. 66, 75

[ZLE07] Zhou J., Larson P.-Å., Elmongui H. G.: Lazy maintenance of materialized views. In *Very Large Data Bases (VLDB)* (2007), pp. 231–242. 21

Publications

International conferences with reviewing committee

- **Feng H.**, Lumineau N., Hacid M.-S., Domps R.: Hierarchy-Based Update Propagation in Decision Support Systems. In *Database Systems for Advanced Applications (DASFAA) (2)*, pp. 261-271, 2012.
- **Feng H.**: Performance problems of forecasting systems. In *East-European Conference on Advances in Databases and Information Systems (ADBIS) (II)*, pp. 254-261, 2011.

National conference with reviewing committee

- **Feng H.**, Lumineau N., Hacid M.-S., Domps R.: Data management in forecasting systems: Case study - performance problems and preliminary results. In proceedings of *Bases de Données Avancées (BDA)*, 2011.

Title: Data Management in Forecasting Systems: Optimization and Update Issues

Abstract: In daily life, more and more forecasting systems are used to determine what the future holds in many areas like climate, weather, traffic, health, finance, and tourism. These predictive analytics systems support three functionalities: prediction, visualization and simulation based on modifications. A specific problem for forecasting systems is to ensure data consistency after data modification and to allow updated data access within a short latency.

Forecasting systems are usually based on data warehouses for data storage, and OLAP tools for historical and predictive data visualization. Data that are presented to and modified by end users are aggregated data. Hence, the research issue can be described as the propagation of an aggregate-based modification in hierarchies and dimensions in a data warehouse environment. This issue corresponds to a view maintenance problem in a data warehouse. There exists a great number of research works on view maintenance problems in data warehouses. However, they only consider updates on source data or evolution of the structure of dimensions and hierarchies. To our knowledge, the impact of aggregate modifications on raw data was not investigated. In addition, end users perform the modification interactively. The propagation of the modification should be efficient in order to provide an acceptable response time.

This "Conventions Industrielles de Formation par la REcherche (CIFRE)" thesis is supported by the "Association Nationale de la Recherche et de la Technologie (ANRT)" and the company Anticipeo. The Anticipeo application is a sales forecasting system that predicts future sales in order to help enterprise decision-makers to draw appropriate business strategies in advance. By the beginning of the thesis, the customers of Anticipeo were satisfied by the precision of the prediction results, but there were unidentifiable performance problems.

During the working period, the work can be divided into two parts. In the first part, in order to identify the latency provenance, we performed an audit on the existing application. The result of audit showed the database may be the main source of latency. We proposed a methodology relying on different technical approaches to improve the performance of the application. Our methodology covers several aspects from hardware to software, from programming to database design. The response time of the application has been significantly improved. However, there was still a situation which cannot be solved by these technical solutions. It concerns the propagation of an aggregate-based modification in a data warehouse. The second part of our work consists in the proposition of a new algorithm (PAM - Propagation of Aggregate-based Modification) with an extended version (PAM II) to efficiently propagate an aggregate-based modification. The algorithms identify and update the exact sets of source data and other aggregates impacted by the aggregate modification. The optimized PAM II version achieves better performance compared to PAM when the use of additional semantics (e.g., dependencies) is possible. The experiments on real data of Anticipeo proved that the PAM algorithm and its extension bring better performance when treating a backward propagation.

Keywords: OLAP, Data warehousing, Decision support systems, Optimization and performance, view materialization.