Dottorato di Ricerca in Management and Information Technology
XV ciclo Nuova Serie
Università di Salerno

Reducing the View Selection Problem through Code Modeling: Static and Dynamic approaches

Valentina Indelli Pisano

December 2016

Chairman:	Supervisor:
Prof. A. De Lucia	Prof.ssa G. Tortora

Abstract

Data warehouse systems aim to support decision making by providing users with the appropriate information at the right time. This task is particularly challenging in business contexts where large amount of data is produced at a high speed. To this end, data warehouses have been equipped with Online Analytical Processing tools that help users to make fast and precise decisions through the execution of complex queries. Since the computation of these queries is time consuming, data warehouses precompute a set of materialized views answering to the workload queries.

This thesis work defines a process to determine the minimal set of workload queries and the set of views to materialize. The set of queries is represented by an optimized lattice structure used to select the views to be materialized according to the processing time costs and the view storage space. The minimal set of required Online Analytical Processing queries is computed by analyzing the data model defined with the visual language CoDe (Complexity Design). The latter allows to conceptually organize the visualization of data reports and to generate visualizations of data obtained from data-mart queries. CoDe adopts a hybrid modeling process combining two main methodologies: user-driven and data-driven. The first aims to create a model according to the user knowledge, requirements, and analysis needs, whilst the latter has in charge to concretize data and their relationships in the model through Online Analytical Processing queries.

Since the materialized views change over time, we also propose a dynamic process that allows users to *(i)* upgrade the CoDe model with a context-aware editor, *(ii)* build an optimized lattice structure able to minimize the effort to recalculate it, and *(iii)* propose the new set of views to materialize. Moreover, the process applies a Markov strategy

to predict whether the views need to be recalculate or not according to the changes of the model. The effectiveness of the proposed techniques has been evaluated on a real-world data warehouse. The results revealed that the Markov strategy gives a better set of solutions in term of storage space and total processing cost.

Contents

Title Page **i**

Abstract **iii**

Contents **v**

List of Figures viii

List of Tables x

1 Introduction **1**

1.1 Motivation 2

1.2 Thesis Outline 4

2 Related Work **5**

2.1 VSP in a static environment 5

2.1.1 Deterministic approaches 5

2.1.2 Randomized approaches 8

2.1.3 Hybrid approaches 9

2.1.4 Query rewriting 9

2.1.5 Discussion 10

2.2 VSP in a dynamic environment 11

2.2.1 Cache updating techniques 11

2.2.2 Incremental view maintenance 13

2.2.3 Discussion 16

2.3 Data warehouse model evolution 17

2.3.1 Schema evolution 18
2.3.2 Schema versioning 22
2.4 Discussion 24

3 Conceptual Organization of Report Visualization: The CoDe Paradigm 25
3.1 The Graphic Language CoDe 26
3.2 The CoDe process 31
3.2.1 CoDe Modeling phase 32
3.2.2 OLAP operation pattern definition 32
3.2.3 OLAP Operation phase 36
3.2.4 Report visualization phase 37

4 The VSP problem 38
4.1 Problem Formulation 38
4.1.1 Cost model 39
4.2 Data structure for the view selection problem 39
4.2.1 AND / OR Graph 40
4.2.2 Multi-View Processing Plan (MVPP) 41
4.2.3 Lattice 41

5 A static approach to VSP problem 43
5.1 Code Modelling 43
5.2 OLAP Operation Pattern Definition 45
5.2.1 Eligible patterns generation 46
5.2.2 Selection of one OLAP eligible pattern 47
5.2.3 Creation of the lattice structure 50
5.3 OLAP Operation Optimization 51
5.3.1 Minimum spanning tree generation 51
5.3.2 Heuristic and views selection 52

6 A dynamic approach to VSP problem **55**

6.1 The CoDe model evolution . . . 55

6.1.1 The CoDe Dynamic modeling . . . 56

6.1.2 The optimized lattice creation . . . 67

7 A probabilistic model to improve the dynamic approach **70**

7.1 Markov analysis and view selection . . . 70

8 Case Study **73**

8.1 Static approach . . . 73

8.2 Dynamic approach . . . 79

8.2.1 Addition of a component to the model . . . 82

8.3 Probabilistic approach . . . 86

9 Conclusions **89**

9.1 Thesis Summary . . . 89

9.2 Perspectives . . . 91

10 Appendix **92**

References **94**

List of Figures

3.1 *Resources* information item and its representation in CoDe. 26
3.2 Definition of the term All_Sales. 27
3.3 Example of AGGREGATION function. 27
3.4 Example of SUM function. 28
3.5 Example of NEST function. 28
3.6 Example of UNION function. 29
3.7 Example of SHARE function. 29
3.8 Example of Relation, ICON and COLOR. 30
3.9 The CoDe process. 32
3.10 CoDe model for the data-mart *Sales* 33
3.11 A multidimensional cube (a) and OLAP operations (b) performed to extract the report in Fig. 3.1(a). 34
3.12 The operation pattern for the *CoDe Term*. 35
3.13 The *Resources.Energy* report. 36
3.14 Graphical representation of the data-mart *Sales* 37

4.1 An example of AND-OR view graph. 40
4.2 An example of MVPP. 41
4.3 An example of Data Cube Lattice. 42

5.1 Dimensional Fact Model of the *Sales* data-mart. 44
5.2 The OLAP Operation Pattern Definition phase. 46
5.3 The prefix tree. 49

5.4 The OLAP Operation Optimization phase. 51

6.1 The Code Dynamic process. 56
6.2 Report on the Feb 24 1997 for the product family drink, food and non-consumable. 57
6.3 CoDe Model for the information item Food D24. 57
6.4 Addition of a new component. 58
6.5 Addition of SUM function. 58
6.6 Addition of EQUAL function. 58
6.7 Addition of an item for the EQUAL function. 59
6.8 Addition of the item $Total_Profit$. 59
6.9 Addition of another component to the item. 60
6.10 Error message. 60
6.11 Code Model with the $Total_Profit$ item. 60
6.12 Addition of the NEST function. 61
6.13 Final CoDe model. 61
6.14 Example of classifier method. 66

7.1 The Markov optimization. 71

8.1 The CoDe model. 73
8.2 The lattice structure with the OLAP operations and the corresponding MST (dashed arrows). 77
8.3 CoDe model for the data-mart *Sales* concerning the cost and profit of the food category in the WA state for the day 24 (a), and its graphical representation (b). 79
8.4 The lattice structure with the OLAP operations and the corresponding MST (dashed arrows) constructed on the CoDe model in Fig. 8.3(a). 81
8.5 The CoDe model with the new component $FoodD25$. 83
8.6 The lattice structure with the OLAP operations and the corresponding MST (dashed arrows) constructed on the CoDe model in Fig.8.5. 85

List of Tables

2.1 Static view selection approaches 10

2.2 Dynamic view selection approaches 17

3.1 Graphical representation of a report. 26

3.2 Summary of the mapping among Code syntax and OLAP operation patterns. 34

5.1 Coefficients to compute V_s and P_c. 50

6.1 Context-sensitive grammar. 62

6.2 Application of the min-max technique. 67

8.1 Eligible OLAP operation patterns for the term *Drink*. 74

8.2 Eligible OLAP operation patterns for SUM_1. 74

8.3 Vocabulary table. 75

8.4 Switchable strings and OLAP unique operation patterns. 76

8.5 Processing time and storage space of adopted algorithms applied on the lattice structure of Fig. 8.2. 78

8.6 Comparison of processing times of two consecutive executions. 78

8.7 Algorithms evaluation on the entire *Sales* data-mart by using the CoDe model in Fig.3.10. 78

8.8 Switchable strings and OLAP unique operation patterns for the CoDe model in Fig. 8.3(a). 80

8.9 Processing time and storage space of adopted algorithms applied on the lattice structure of Fig. 8.4. 82

8.10 Switchable strings and OLAP unique operation patterns for the CoDe model in Fig. 8.5. 83
8.11 The SEQ_MATCH execution. 84
8.12 The SEQ_MATCH output. 84
8.13 Algorithms evaluation by using the CoDe model in Fig. 8.5 86
8.14 The VN_1 vector 86
8.15 The VN_i vector 87
8.16 The VN_{i+1} vector 87
8.17 Markov Algorithm evaluation 88
8.18 Algorithms evaluation corresponding to the CoDe model in Fig.8.5. 88

10.1 The $Q' * V'$ Matrix 92
10.2 The $Q * V$ Matrix 93

Chapter 1

Introduction

In recent years, the use of decision support systems based on data warehouses is widely increasing in different application domains, such as marketing, business research, demographic analysis, security, and medical field. A data warehouse (DW) is an integrated collection of information extracted from distributed and heterogeneous database systems. Differently from a database, which is a planned collection of information, usually stored as a set of related lists of similar items designed to handle transactions, a DW collects and stores integrated sets of historical data organised in a way to make analysis fast and easy. According to [32], a DW is a subject-oriented, integrated, time-varying, non-volatile collection of data that is used primarily in organizational decision making. Moreover, these data are redundantly stored, cleaned from inconsistencies, and transformed for optimal access and performance. To this end, with respect to a relational database, a DW environment can include an extraction, transformation, and loading (ETL) solution, data mining capabilities, client analysis tools, and so forth. For example, in the business world, a DW for market research might incorporate sales information, such as the number of sold products, their price, the number of customers, information about the delivered invoices. By combining all of these information in a DW, a company manager can analyse gains, trends about sold items, and takes decisions about new marketing strategies. As a consequence, a DW can be used as a decision support system ensuring the user to get the appropriate data at the right time. Therefore, in a context like the business world, where a large volume of data is produced, the speed with which the information are computed represents a crucial

aspect. In order to satisfy the need of business managers and to help them to make fast and precise decisions, DWs have to include efficient On-line Analytical Processing (OLAP) tools able to process complex queries [70].

With the widespread use of DWs, the size and complexity of OLAP queries have considerably increased, so the query processing costs impact on the performance and the productivity of decision support systems. Moreover, the execution of high frequency queries on-the-fly every time is expensive, produces wasted effort, and makes the data warehousing extremely slow. Thus, improving the performances of data warehousing processes is one of the most crucial aspect to boost the productivity of companies.

1.1 Motivation

In the last two decades several approaches have been proposed to speed up the data warehousing process, such as advanced indexes, parallel query processing, and materialized views [2, 18, 21, 54, 69]. The latter is the most common investigated approach in the literature. A *materialized view* is an 'information of interest' used by business managers to takes advantageous decisions. In particular, the materialized views are queries that instead of being computed from scratch are already calculated, stored, and maintained.

On the one hand, materialising the views every time requires a large amount of memory, and on the other hand, not materialising any view requires lots of redundant on-the-fly computations. Thus, it is important to identify the set of views to materialize with the lowest query processing costs and storage space. In the literature this issue has been investigated in many studies [1, 29, 62, 73] and it is known as the *view selection problem* (VSP). Formally, given a database schema and a query workload, the VSP is the problem of selecting an appropriate and minimal set of materialized views under fixed constraints (i.e., the storage space) [42]. A *query workload* is a set of queries that corresponds to the user requests submitted to the DW [51, 39, 26]. In order to define the set of workload queries and the minimal set of materialized views a set-up phase on the DW is required. In particular, to select the right set of workload queries it is required an analysis phase that can be performed by calculating the frequent data usage on the DW. Furthermore, also to select the appropriate set of views it is required a training phase on the DW. This phases are time consuming, increases the overall costs and produces wasted time.

In the literature several approaches has been proposed to mitigate the VSP by selecting minimal set of materialized views under fixed constraints [52, 48, 42, 27, 53]. However, to our knowledge, no research tries to avoid the overhead caused by such set-up phases.

To address this issue we use CoDe that let us to know a priori the data of interest and the relationships among them in order to answer to the user requests and optimize the views selection process.

The language *CoDe* [56, 55], by exploiting the business manager knowledge, allows specifying the relevant data, the relationships among them, and how such information should be represented. CoDe is a visual language that allows to conceptually organize the visualization of reports, it adopts an hybrid modeling process combining two main methodologies: *user-driven* and *data-driven*. The first one aims to create a model according to the user knowledge, requirements, and analysis needs, whilst the latter has in charge to organize data and their relationships in the model through OLAP queries.

In this thesis we present a basic approach that exploits the CoDe modeling language to find the set of workload queries and to mitigate the VSP. This approach extends the CoDe process by enabling the selection of the minimal number of required OLAP queries, compact them, and create a lattice structure avoiding the explosion of the number of nodes. The *lattice* is a directed acyclic graph (DAG), where the nodes represent the views (or queries), while the edges represent the derivability relations between views. This representation allows queries to be answered from the result of other queries, optimizing the query processing costs.

However, the nature of the selected views is uncertain because DW schemas, and their data, can change frequently. As a consequence, the model and the views have to be upgraded or constructed from scratch, requiring the maintenance of the schema. To this end, we extend our previous approach by proposing dynamic process that (i) allows managers to upgrade the CoDe model with a new context-aware editor, (ii) builds an optimized lattice structure that minimizes the re-computation of the views, and (iii) proposes the new set of views to materialize.

Nevertheless, the re-computation of the views, each time the model changes, produces an overhead in the data warehouse process. Thus, we exploit a Markov strategy to predict if a new set of views improves the performances. In particular, we adapt a probabilistic approach to the CoDe dynamic process that exploits the impact frequency of the OLAP

queries on the possible new views, and suggests the subset of views to pick up for materialization, and those to be replaced.

In order to validate the static, dynamic, and probabilistic approaches we have analysed their performances on the Foodmart DW [45]. Foodmart maintains information about a franchising of big supermarkets located in the United States, Mexico, and Canada. In particular, the data-mart *Sales* has been selected to analyse the sales of these stores, the customers information, and the products assortment [31, 67].

1.2 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 outlines related work. Chapter 3 provides an overview of the CoDe modeling, its syntax and semantic. Chapter 4 defines the view selection problem, while Chapters 5, 6, and 7 describe the optimization approach to define a workload query and to select the materialized views by exploiting static, dynamic and probabilistic methodologies, respectively. Chapter 8 provides the details of an experimentation and a discussion of the obtained results. Final remarks are discussed in Chapter 9.

Chapter 2

Related Work

The view selection problem (VSP) has been the subject of a considerable amount of previous work. In the following we first discuss the most popular approaches defined in literature that select materialized views to speed up decision support queries in static environments, then the ones for dynamic environments. Finally, we also consider the approaches addressing the schema evolution problem.

2.1 VSP in a static environment

The VSP plays a central role in the design and query of a DW [70]. Many approaches have been proposed to address this problem, such as deterministic, genetics, hybrid, and query rewriting algorithms [22, 42, 27].

2.1.1 Deterministic approaches

Harinarayan *et al.* in [29] presented a greedy algorithm to select a set of materialized views using a constraint on the maximum number of views to materialize and a framework lattice to express dependencies between such views. The lattice data cube is a directed acyclic graph (DAG) whose nodes represent the views (or query) characterized by the attributes of the group-by clause, while edges denote relations of derivability between views (see Section 4.2.3). The benefit of this representation is that some queries can be answered from the result of others optimizing the query processing costs. The idea of this algorithm is to select the views to materialize in the direction of maximizing the total benefits, ending after k iterations and returning the k selected views to be materialized. The total benefit

is denoted by the expression $B(v, S)$, where v is the view to choose to take into account those materialized S. The greedy algorithm returns a selection of views in the lattice that approximates very well the optimal solution, representing a lower-bound of 63% compared to the optimal solution. However, they did not take into account the view maintenance costs and the storage space constraints. Moreover, the lattice representation is suitable and easy to implement in low dimensional deterministic cases but the main disadvantage of this representation is that the number of nodes in the lattice structure is exponential relative to the number of dimensions. Our first goal was to develop a framework lattice more scalable than theirs to select the views taking into account this information and making our solution closer to the real problem.

In [62] the VSP is addressed by exploiting a greedy algorithm which picks the views focusing its choice on a benefit metric, such metric is based on the probability which each view being queried. In particular, they propose PBS-U that picks aggregates in order of their probability weighted size. They try to minimize the following ratio: $\mid v \mid /p_v$, where $\mid v \mid$ is the size of aggregate view v, and p, is the frequency with which v is queried. Moreover, they examine the materialized view selection problem when subsets of aggregates can be computed using chunks [17]. The idea of chunks is motivated by MOLAP system which use multi dimensional arrays to represent the data. Instead of storing a large array in simple row or column major order they are divided into ranges, and chunks are created based on this division. They show that the benefit of the views selected by PBS using chunks can be greater than the benefit of the optimal set of views selected without chunk based pre computation. However, this solution is not suitable for us because we need to know a priori the frequency with which the view is expected to be queried.

Yang *et al.* [71] proposed a heuristic algorithm which utilizes a Multiple View Processing Plan (MVPP) to obtain an optimal materialized view selection. The MVPP is a DAG representing a query processing plan (or strategy) to obtain the response views of these queries in a warehouse (see Section 4.2.2). Basically, the execution plan of a query can be more than one and among them we can find the optimal one. So a MVPP can be obtained by considering all the optimal plans of each query. The proposed algorithm takes in input a MVPP with the following parameters: the set of nodes, the set of the directed edges that indicate the order relation between nodes, the processing and maintenance costs, the access and upgrade frequencies. The authors use a tree structure which every node is a

potential candidate to the materialization. If a node is considered a good candidate, the savings will calculate taking into account the materialization costs and subtracting the cost maintenance. If the result is a positive value, the node is added to the tree, otherwise is deleted with all its descendants from the candidate set. In particular the following heuristic was adopted to reduce the search space: if the views v_1 and v_2 are related where v_1 is the son of v_2 and if the materialization of v_1 returns no benefit, then v_2 is not considered. In this way, they assert to achieve the best combination of performance and maintenance costs. However, this algorithm did not consider the storage space constraints. Moreover, the MVPP representation is suitable for depicting relationships among queries to the base relations through intermediate and shared temporary views. From the MVPP graph, the size of intermediate views can be found or computed easily and provided as input to the view selection for materialization algorithm. But the cost involved in generation of an MVPP graph from the query workload of a data warehouse is high when the query processing plan changes and input workload is very large.

A framework to solve the views selection problem is presented by H. Gupta in [23] that exploit AND-OR Views Graph. A AND-OR graph is one of the most common DAG used in the literature and it is the union of all possible execution plans of each query. It is composed of two types of nodes: operation node, and equivalence node (see Section 4.2.1). The author presents greedy heuristics that in polynomial time make the views selection by using AND-OR graphs in order to minimize the total processing cost of the workload queries. Gupta does not consider the maintenance views cost but his heuristic takes into account a constraint on the storage space.

This work was extended by the same Gupta H. with the collaboration of Mumick I.S. [24], [25]. In these articles they take into account the maintenance views cost and develop an algorithm that minimize the total views maintenance and the total processing costs, taking into account a limited storage space. Proofs are presented to show that this algorithm is guaranteed to provide a solution that is fairly close to the optimal solution. However, the graph generation process becomes costly for complex and huge query workloads.

Other deterministic approaches are represented by the works [57, 44, 3]. The authors Roy et al. in [57], introduce the benefits and significant improvements in multi-query optimization techniques used in conjunction with a greedy heuristic. This heuristic allows to iteratively pick from an AND-OR graph the set of views to materialize, which minimize

the processing cost of the queries that make up the workload. This study was extended by H. Mistry et al. in [44] in order to optimize the maintenance costs of materialized views. This article in addition to the speed of query processing workload based on the selected view, it shows an algorithm that creates an efficient maintenance plan materialized view, which uses the expressions in common between the different expressions of maintained views. However, these works are studying solutions which do not take account of any constraints on resources.

The view selection algorithm proposed by X. Baril et al. [3] is based on the idea of a view level in the query graph. Thus, each view has a level associated, starting from the root that his level is one, and so on in ascending order. The framework used for the view representation is a variant of an AND-OR graph, called Multi-View Materialization Graph (MVMG), it allows expressing common sub expressions of aggregate SQL queries. The approach deal with the view selection problem under a storage space constraint and split it in two phases. The first one performs a local optimization heuristic, calculating for each view in MVMG the local benefit to materialize it, and pre-selecting a set of candidates views that reduce the processing costs, without increasing significantly the maintenance costs. Such heuristic calculates the total costs for each level of the graph MVMG and selects the views belonging to the level which has the minimum total cost, in term of development and maintenance, of all the views that compose it. The second phase selects from the set of view candidates for each query, the views to materialize that maximize the overall benefits taking into account a space constraint. The proposed algorithm has polynomial complexity and improves the performance of the algorithm proposed in [71], because the latter provides a solution based on MVPP which tends to materialize the views closer to the leaf node (base relations), making the processing and maintenance costs higher. The difference to our proposed work is that they based them approach on the use of SQL queries and not OLAP queries.

2.1.2 Randomized approaches

Genetic algorithms generate solutions using techniques inspired by the natural selection laws and biological evolution. The base strategy starts with an initial random population of solutions (chromosomes) and generates new populations mixing randomly (crossover) and changing (mutation) the best solutions that are evaluated through a function called *fitness*,

then stops it execution when there is no improvement to the fitness function evaluated on the current population. Some genetic solutions were proposed, such as [25], they use a MVPP framework to represent the set of views that concur to materialization. The set of materialized views is chosen in terms of reduction on the processing costs and maintenance costs. However, for the random characteristics, these algorithms do not give an optimal solution. A solution to this problem can be to add a penalty value to the fitness function to ensure that non-optimum solutions are discarded, anyway this should be investigated.

2.1.3 Hybrid approaches

Other approaches are represented by the hybrid algorithms. Zhang *et al.* in [73] applied their hybrid solution combining greedy and genetic algorithms to solve three types of problems. The first one addressed the optimization of queries, the second concerned the choice of the best execution plan for each query and the third was about the views selection problem. However, such algorithms are characterized by a high computational complexity, which makes them not a good choice.

2.1.4 Query rewriting

Another different approach to the view selection problem is the rewritten of query (Query Rewriting). This strategy not only selects the views to materialize, but rewrites completely the query workload based to optimize its processing time. So the input to the view selection problem is no longer a multi-query DAG, but the definition of the same queries. The problem is modelled as a research problem by exploiting a set of transformation rules, which detect and use common sub expressions between the query workload and ensure that each of them can be answered using only materialized views.

The work proposed by Park C.S. et al. in [51] introduces an algorithm for the OLAP query rewrite that significantly improves the utilization of materialized views with respect to most of the approaches proposed in the literature and analysed in the survey [27], because it considers the characteristics of the DW and the OLAP query. The authors start by defining a 'Normal Form' of typical OLAP queries expressed in SQL and based on a structure, very similar to the one proposed in [29], called Dimension Hierarchies Lattice (DH), and present the conditions under a materialized view can be used in the rewriting of a given OLAP query. In particular, these conditions are specified by the partial

ordering relations expressed by the lattice structure between the components (selection attributes and aggregation) of their normal forms. The query rewrite method consists of three major steps. The first step select the views to materialize through a greedy algorithm of quadratic complexity compared to the number of views in the lattice structure. The second step generates the query blocks for materialized views using query regions on the lattice structure, that represent areas that share the predicates of the query selection. Finally, the third step integrates the query blocks in a compact final rewrite of all the query workload. However the drawback of the query rewriting is that the number of possible rewritings of a query using views is exponential in the size of the query.

2.1.5 Discussion

Based on the selected works, we observe that the problem that affecs the deterministic and heuristic algorithms is the scalability. Thus, these methods are effective only with a small number of views. To overcome this problem several randomized and evolutionary algorithms have been introduced. However, they have limitations as well. Genetic Algorithms (GA) are able to perform better in multi-directional search over a set of candidate views in the search space. Thus, such algorithms can provide effective search performance and find a solution near a global optimum in the view selection problem. Moreover, a limitation of the evolutionary algorithms is that it is hard to acquire good initial solutions, and therefore in the view selection problem, GA-based approaches converge slowly. To summarize this section we propose the following table 2.1.

Table 2.1: Static view selection approaches

	Pro	**Cons**
Deterministic	Polynomial complexity	Non-optimum solutions
Randomized	Find a point near the global optimum	There is no guaranteed convergence to global minimum and the convergence is usually slow
Hybrid	Best set of solutions	High computational complexity
Query rewriting	Compute the set of materialized views and also find a complete rewriting of the queries over it	High computational complexity

2.2 VSP in a dynamic environment

Static view selection algorithms still suffer from a variety of problems, first of all they rely on a pre-compiled query workload, and may not perform well for ad-hoc queries. Second, the space maintenance, and time constraints may change over time while the materialized views set is fixed, once selected. Finally, the space and the maintenance constraints are usually unable to be minimized at the same time. Thus, to adapt the problem to an actual one, monitoring and reconfiguration should be performed. The DW has a dynamic nature, and since it supports the decision making process then its data or even its schema have to be changed. Consequently, the materialized views defined upon such DW have to be updated. Thus, in the view selection process, these changes should be taken into account and dynamic view selection techniques should be investigated [37]. In the literature several works has been outlined such as [60, 38, 17, 15] that concerns cache updating algorithms and [50, 75, 59, 47, 19] that concerns incremental views maintenance algorithms.

2.2.1 Cache updating techniques

With the caching strategies, the cache is initially empty and data are inserted or deleted from the cache during the query processing. Materialization could be performed even if no queries have been processed and materialized views have to be updated in response of changes on the base relations [42]. Caching can be divided into physical and semantic. The first one refers to the mechanism employed in operating systems and traditional relational databases, where some physical storage unit such as a page or a tuple is kept in cache. Semantic caching keeps track of the semantic description of the cached data [60, 38, 17, 15] and takes advantage of high level knowledge about the data being cached. We take into account only the semantic caching that is referred to views or queries, since the cache manager knows both the data and their query expressions.

WATCHMAN [60] is a cache manager for OLAP queries. It is based on two algorithms for cache replacement and for cache admission and perform a simple 'hit or miss' strategy that relies on temporal locality of queries to gain benefits. The admission and cache replacement algorithms are denoted as LNC-A (Least Normalized Cost Admission) and LNC-R (Least Normalized Cost Replacement). LNC-A and LNC-R aim at minimizing the execution time of queries that miss the cache instead of minimizing the hit ratio. Usually

the criterion for deciding which query to cache is based upon its probability of reference in the future, however such probability is not precise and based on the past reference pattern by assuming that these patterns are stable. Both algorithm use a profit metric, reported in the following, based for each retrieved set on its average rate of reference, its size, and execution cost of the associated query.

The algorithm LNC-R in order to capture the actual execution costs (or savings) of a retrieved set it uses other additional parameters in addition to the reference pattern: λ_i which is the average rate of reference to query Q_i, S_i that is the size of the set retrieved by query the Q_i, and c_i the cost of execution of query the Q_i. LNC-R aims at minimizing the cost savings ratio (CSR) defined as:

$$CSR = \frac{\sum_i c_i h_i}{\sum_i c_i r_i} \tag{2.1}$$

where h_i is the number of times that references to query Q_i were satisfied from cache, and r_i is the total number of references to query Q_i.

The algorithm LNC-A prevents caching of retrieved sets which may cause response time degradation. Thus, it should cache a retrieved set only if it improves the overall profit. In particular, given a set C of replacement candidates for a retrieved set RS_i, the procedure decides to cache RS_i only if RS_i has a higher profit than all the retrieved sets in C. The profit is defined as follow:

$$profit(C) = \frac{\sum_{RS_j \in C} \lambda_j c_i}{\sum_{RS_j \in C} S_j} \tag{2.2}$$

where $RS_i = \frac{c_i}{s_i}$.

DynaMat [38]is a system that dynamically materializes information at multiple levels of granularity in order to match the workload but also takes into account the maintenance costs, the time to update the views, and the space availability. DynaMat constantly monitors incoming queries and materializes the best set of views subject to the space constraints, it work is performed in two phases. The first one is the 'on line' phase where the system answers queries posed to the warehouse using a Fragment Locator to determine whether or not already materialized results can be efficiently used to answer the query, a cost model authors have defined to perform this phase. A Directory Index supports sub linear search in order to find candidate materialized views. Then the result is computed and given to

the user and it is tested by the Admission Control Entity which decides whether or not it is beneficial to store it in the Pool. During the on-line phase, the goal of the system is to answer as many queries as possible from the pool and at the same time DynaMat will quickly adapt to new query patterns and efficiently utilize the system resources. The second is an 'off line' phase, during which the updates are stored in the warehouse and the materialized results in the Pool are refreshed. DynaMat is more flexible than WATCHMAN, but it does not allow combinations of cached views to answer queries.

Deshpande et al. in [17] propose to use *chunks*, which are organized in a hierarchy of aggregation levels. Chunk caching is a kind of semantic caching specific to chunk based organization [74]. Chunks have finer granularity than views or tables and are thus more flexible and may be more efficient in answering overlapping queries mainly involving aggregations. A multidimensional query is then decomposed to chunks at the same aggregation level, with missing chunks computed from raw data. This work is further extended in [16] to allow aggregation from lower level cached chunks. As metrics of caching Deshpande et al. in [17] utilize the CLOCK algorithm which discards the oldest data cached and that is an efficient approximation of LRU.

2.2.2 Incremental view maintenance

The incremental maintenance of materialized views is a well studied problem, and efficient maintenance algorithms are used to reduce the huge amounts of data transfer at runtime. When updates occur to a database there are two distinct execution strategies to update all affected materialized views whether incrementally or not. In particular, the first is the *immediate update*, where all affected views are immediately updated. This strategy creates an overhead for the processing of the up dates but minimizes the query response time for queries accessing the view. The second is the *deferred update*, where all affected views stay outdated until an access to them is made. This strategy avoids the system overhead associated with immediate up date propagation but slows down query evaluation for queries accessing outdated views. Both immediate and deferred maintenance guarantee that the view is consistent with the underlying database at the time the view is accessed.

In [50], authors propose the EVE system(Evolvable View Environment). They define that each component is composed by attribute, relation or condition that has attached two evolution parameters. The *dispensable parameter* specifies if or not the component

must be present in any evolved view selection. The *replaceable parameter* specifies if the component could be replaced or not in the view evolution process. Then define the *pc containment constraint* in order to describe if a view is equivalent, subset or superset of an initial view whilst the evolution parameters allow the user to specify criteria based on which view will evolve. In the EVE system they use several algorithms such as the POC algorithm [49]. It uses containment constraint information for replacing the deleted relation with another relation such that the redefined view satisfies the evolution parameters. However, this algorithm has two major drawbacks. The first is that it can be applied only if the relation is still available even after the evolution. Secondly it is composed by intermediate steps that could considerably increase the size of the intermediate views requiring unnecessarily overhead in term of source IO time and overall computation time. Thus, they propose other strategies such as: the re-materialization strategy, that computes the view from scratch given its new definition. SYNCMAB strategy that uses the containment information between the relation and its replacement given by a PC constraint and apply defined maintenance strategies. SYNCMAA strategy that applies specialized techniques to compute the view when the relation is not available. Finally, redefinition strategy that apply maintenance techniques for view redefinition for each change necessary to obtain the new definition of a view.

Another incremental maintenance technique is outlined in [75] that illustrates in terms of obtaining modification information from different sources, then ranks them by ascending order, next inserts them into the message queue, then removing modification information from the message queue and finally carry on incremental changes and modification operations. The experimental result also shows the cost reduction in view maintenance.

Ghosh et al. in [19] exploit the linear regression on attributes to find the co-relations between such attributes. They adopt an incremental view maintenance policy based on attribute affinity to update the materialized views at run time without using extra space and minimizing the data transfer between the secondary memory and primary memory. They exploit an Attribute Affinity Matrix (AAM) to classifies views taking into account the relations between the attributes of each view. The last column of AAM represents the total deviation of each attribute from the other attributes. Moreover they define the Important Attribute and Affinity Matrix (IAAM) that store information about the materialized view set. The first row of IAAM represents the number of occurrences of each attributes and

the second row represents the total deviation of corresponding attribute.

The attributes that have no relations in common with the other attributes are labelled as 'unmatched'. When a view is selected the method keeps an amount of space to store the views that have attributes labelled as 'unmatched'. Each time a query is submitted, such procedure looks for a materialized view, if the view is found, the *total use* of each attribute for each involved query, is incremented. Instead, if no view is identified that can respond to the request, a view with the highest number of related attributes with the involved query is kept. For the 'unmatched' attributes, the procedure can calculates the 'total use' value of the attribute in the secondary memory, if it has a value greater than the attributes belonging to the view already materialized, then such attribute is merged with the attributes of the materialized view. Otherwise, it can checks if there is a tie between the attribute occurrences, and use the AAM to break the tie. Then the 'total deviation' of the attributes (the one in the main and the other in the secondary memory) is verified. If the 'total deviation' attribute in secondary memory is less than the attribute in main memory, then the attribute is added, although they do not merge. But, if there is no space left, the system calculates the important attributes in the views. The attribute that has the highest number of occurrences and least amount of total deviation is considered as the most important attribute. With respect others methodologies this method instead of replacing the entire materialized views it replaces the attributes only from the primary memory. Indeed, it reduces the data transfer between primary and secondary memory, which has better time complexity over the existing system which replaces the entire materialized view.

Ghosh et al. in [20] proposed an approach that exploits a Markov strategy in order to select the set of views to materialize on large amount of data transfer. Their solution replaces the unused materialized views from primary memory with new views from secondary memory that are likely to be used frequently. The method is divided in two phases: the *Initial Probability* and the *Stable Probability*.

Initial Probability. The first step takes in input a set of materialized views stored in the secondary memory and a set of queries computing such views. A $(m * n)$ *Hit Matrix* (VHM) is created, based on the input, where m denotes the number of distinct queries and n the number of views. If queries are using an initial view V_i and continue hitting in it the corresponding cell value of V_i will be marked as 'HIT'. The moment the query misses

V_i and hits in another view V_j the iteration stops and calculate the probability of the next query will hit in V_j. A *query that hits in a view* means that such query is responded by such view and the view is a complete or partial answer to such query. This process continues for each view located in the secondary memory.

Steady State Probability Calculation. This step crates a $(n * n)$ *Initial Probability Matrix* where each cell contains the probability that queries hit in a specific view for the first time, then the probability that the next query will hit in that particular view and also the probability that the next query will hit the other views. The future state of the system is calculated by the Markov analysis. Successively, after the computation of the Initial Probability Matrix then it is introduced a transition matrix T that calculates the future states of the system $VNvn(i)$ by multiplying present state with Initial Probability Matrix. Where VN is the probability of hit at present, vn the initial starting state, and i is the i-th future period. For example, by supposing to have three views stored in secondary memory, the T matrix has the following structure $[V1v1(1)V2v1(1)V3v1(1)]$, where the probability of a query hitting in the 1st view for the first time, given that the query hits in the 1st view is 1. So the value of T will be [1.00.00.0]. Then T will be multiplied with Initial Probability Matrix and this operation will be repeated until a steady state probability is achieved. Successively, when the step converges then the views that has the highest probability are transferred from secondary memory to primary memory. Their solution is different from ours because we considerate that views are stored in secondary memory, then they did not take into account the storage space constrains.

2.2.3 Discussion

All the cache updating techniques restrict the query language to a relatively simple class so that the interrelationship modelling between candidate views or between views and queries is usually analogous to some geometric relationship that is efficient to reason about. In principle, the relationship between views and queries can be handled using the techniques of answering queries using views [64, 27]. Another most significant issue in view caching is the admission and replacement control, i.e., how to decide which view is admitted and which view is replaced. If space allows, caching data is in general beneficial. The situation is more complicated, if the free space is not sufficient for the new view. The benefits of use the cache methods are low overhead, efficiency and flexibility, but the limitation regards

Table 2.2: Dynamic view selection approaches

	Pro	**Cons**
Incremental view maintenance	A view can be exploited even before it is fully materialized	Do not concentrate on reducing time complexity as well as space complexity
Cache updating	Low overhead, efficiency and flexibility	Admission and replacement control if there is not enough space for the new view

the admission and replacement control if there is not enough space for the new view.

The incremental view maintenance techniques deal with different methodologies of materialized view maintenance but do not concentrate on reducing time complexity as well as space complexity. However, reduction of these two types of complexity is inherently necessary as these systems deal with huge amount of data at run-time.

To summarize this section we propose the following table 2.2.

2.3 Data warehouse model evolution

One of the key points for the success of data warehouse process is the design of the model according to the available data sources and analytical requirements. However, as the business environment evolves, there may be some changes in the content, the structure of the data sources, and the analysis requirements. The main purpose of a DW is to provide analytical support for decision making, and a DW scheme and its data can evolve at the same time. In particular in a DW scheme can be added or removed dimensions, measures, levels. Differently data can be inserted, deleted and updated in the DW.

For example, Kimball et al. [36] introduced three types of 'slowly changing dimensions', which consist of three possible ways to deal with volumes changes. The basic assumption is that an identifier can not change, but the descriptors do. The first way is to update the attribute value but in this case, the historicization of changes is not available. Then, this solution if the updated attribute is involved to perform the analysis, has bad consequences on the coherence analysis. The second type allows keeping all the versions of the attribute value creating another valid record for a period of time. The disadvantage of this approach is the loss of a comparison between all versions. This is due because the links between the

evolutions are not preserved even if the information is stored. The last type is the creation of another descriptor to track the old value in the same record, and then we maintain the link between the two versions. However, if there are several evolutions, there is a problem to consider the different versions with changes of different attributes not present at the same time. Kimball's study takes into account the needs of most users, and stresses the need to keep track of the history and the links between evolutions. In fact, the main goal of a DW is to support a correct analysis over the time and to ensure good decisions. This goal mainly depends on the capacity of the DW to be a good mirror of reality.

To the best of our knowledge, the model evolution of the data warehouse problem can be classified into two different approaches, that are: schema evolution [1, 2, 3, 4], and schema versioning [5, 6].

2.3.1 Schema evolution

This approach, also named model evolution, focuses on dimensions updates [30, 6], facts and attributes updates [8], and instances updates [46].

In [30], authors proposed a formal model of dimension updates that include the definition of primitive operators to perform these updates and a study of the effect of these updates. Those operators are:

- *Generalize operators*: allow the creation of new level to roll up a pre-existent level. Authors use the example of the dimension 'store' to which they defined a new level 'type of store' that generalizes the dimension 'store'

- *Specialize operators*: add a new level to a dimension. Authors specialize the dimension 'day' with the level 'hour', and then the level 'hour' specializes the dimension 'day'.

- *Relate levels operators*: define a roll up function between two independent levels of the same dimension. Authors defined a relation between the level 'category' and the level 'brand'. Those two levels were independent.

- *Unrelated levels operators*: delete a relation between two levels. Authors deleted the relation between the levels 'company', 'category' and the level 'brand'.

- *Delete level operator*: delete a level then to define new relations between levels. Authors deleted the level 'branch' then a direct relation between the levels 'category' and 'item' was defined.

- *Add instance operator*: add an instance to a level in the dimension. Authors added the instance item five to the level 'item'.

- *Delete instance operator*:delete an instance of a level. Authors deleted the instance item four of the level 'item'.

After defining operators to manage dimension updates, they proposed some data cube adaptation after the Delete level update, Add level update, Delete Instance update, Add Instance update by computing for each cube view an expression to maintain it.

In [6] authors proposed an extension to the work presented in [30] and defined the Warehouse Evolution System (WHES) a prototype to support dimensions and cubes update. In fact, they extended the SQL language and create the Multidimensional Data definition Language (MDL). The latter allows defining operators to support evolution of dimensions and cubes. Where the cube is the fact table and the axis is the dimension in the relational schema. For dimensions update, authors defined the following operators:

- *Create Dimension*: that allows the creation of a new dimension (with its name, its properties and its levels).

- *Drop Dimension*: it allows the removing of an existing dimension (with its name, its properties and its levels).

- *Rename Dimension*: it allows the update of the name of a given dimension.

- *Add Level*: this operator allows the insertion of a new level to a given dimension.

- *Delete Level*: this operator allows the removing of a level from a given dimension.

- *Rename Level*: it allows changing the name of a given level.

- *Add Property*: that add a property or an attribute to a given dimension or a given level.

- *Delete Property*: that delete a property from a dimension or from a level.

While, for cube updates, authors defined the following operators:

- *Create Cube*: this operator creates of a new cube.
- *Drop Cube*: this operator deletes of a given cube.
- *Rename Cube*: this operator allows changing of the name of a given cube.
- *Add Measure*: this operator allows the insertion of a measure to a given cube.
- *Delete Measure*: this operator deletes of a measure from a given cube.
- *Rename Measure*: it changes the name of a given measure.
- *Add Axis*: this operator allows the insertion of an axis of analyse to a given cube.
- *Delete Axis*: it the deletion of a given axis of analyse from a cube.

In [8], authors defined a formal framework to describe evolutions of multidimensional schemas and instances. The framework is based on a formal conceptual description of a multidimensional schema and a corresponding schema evolution algebra. This formal description constitutes the data model. That was defined as follows: a MD model η is a 6 tuple $(F, L, A, gran, class, attr)$ where F is a finite set of fact names, L is a finite set of dimension level names, A is a finite set of attributes names, $Gran$ is a function that associates a fact with a set of dimension level names, $Class$ is a relation defined on the level name, $Attr$ is a function mapping an attribute to a given fact or to a given dimension level. After defining the data model, authors presented a set of formal evolution operations, listed in the following, and grouped by those which have effect on the model and those which have no effect on the model. The following evolution operations have no effects on the model:

- *Insert level*: it consists on extending the MD model by a new dimension level. This operation has no effects on instances.
- *Delete level*: it consists on deleting a dimension level from an MD model but the deleted dimension must not be connected to the fact. This operation has no effects on instances.

- *Insert attribute*: it consists on creating new attribute without attaching it to a dimension level or fact. This operation has no effects on instances.
- *Delete attribute*: it consists on deleting an attribute which is a disconnected attribute. This operation has no affects on instances.
- *Insert classification relationship*: it consists on defining a classification relationship between two existing dimension levels. This operation has no effect on instances.
- *Delete classification relationship*: it consists on deleting a classification relationship without deleting the corresponding dimension levels. This operation has no effect on instance.

The following evolution operations have effects on the model:

- *Connect attribute to dimension level*: it consists on connecting an existing attribute to a dimension level. This operation has an effect on the instance. In fact, it should define a new function for each new attribute to assign an attribute value to each member of the corresponding level.
- *Disconnect attribute from dimension level*: it consists on disconnecting an attribute from a dimension level. This operation has an effect on the instance since it should eliminate the deleted attribute functions.
- *Connect attribute to fact*: it consists on connecting an existing attribute to a fact. This operation has an effect on the instance. In fact, it should define a function that maps coordinates of the cube to measures.
- *Disconnect attribute from fact*: it consists on disconnecting an existing attribute from a fact. This operation has an effect on instance. In fact, it should delete the function that maps coordinates to measures.
- *Insert fact*: it consists on extending the MD model by a new fact and without attaching dimension levels to this fact. It should define dimensions for this fact separately. This operation has no effect on the instance but has an effect on the MD model since it should define a new function that associates a fact with a set of dimension level names.

- *Delete fact*: it consists on removing an existing fact from the MD model but this fact must not be connected to any dimension and do not contain any attributes. This operation has no effect on the instance but has an effect on the MD model since the name of the deleted fact will be removed from the finite set of fact names.
- *Insert dimension into fact*: it consists on inserting a dimension at a given dimension level into an existing fact. This operation has as an effect the computing of the new fact.
- *Delete dimension*: it consists on deleting a dimension which is connected to a fact from it. This operation has as an effect the deleting of the function that maps coordinates of the cube to measures.

In [46] authors propose an approach to querying a multi version data warehouse. They extended a SQL language and built a multi version query language interface (MVDW) with functionalities that express queries to address several DW versions, present their partial results annotated with version and meta-data information and, if possible, integrate partial results into a single homogeneous result set. The meta-data information allows to appropriately analyse the results under schema changes and dimension instance structure changes in DW versions.

2.3.2 Schema versioning

This approach, also named temporal modelling, focuses on keeping different versions of a given DW [4, 10].

In [4] authors define the concept of schema versioning, that consists in keeping the history of all versions by temporal extension or by physical storing of different versions. They distinguish two types of versions: a real versions and an alternative version. The real versions support changes related to external data sources (changes in the real world) but the alternative versions support changes caused by the *what-if analysis*. Maintaining real and alternative versions of the whole data warehouse allows them to run queries that span multiple versions and compare various factors computed in those versions and to create and manage alternative virtual business scenarios required for the what-if analysis. To illustrate the two different type of version they consider an example of a police data warehouse, storing information about committed violations and tickets given to drivers, in

given locations (cities located in provinces) at given periods of time. Violations are organized into severity groups that define minimum and maximum fines allowed for violations. As real version, they presented the case of changing the borders of regions (i.e. the city Konin moved from the region A to the region B). Assuming that the police may analyse those data in order to find out how many violations were committed in a set of given cities at certain periods of time. Cities are grouped into administrative regions, whereas violations are organized into groups. In this case an old DW version would store data before an administrative-territorial change, and a new DW version would store data after that change. As alternative version, authors presented a virtual scenario. Assuming that a certain percent of fines paid by drivers in a city feeds the local budget, the police may investigate how the budget would increase if they moved a violation from the group of ordinary violations to a group of more severe ones. In order to create such a simulating environment, a data warehouse must be able to create alternative versions of schema and data and manage these versions. Moreover, in this scenario a new version of fact data will also be created from the previous version, so the decision maker can compare the real situation with the virtual one. In their proposed approach every version has a valid time so as time constraints on versions. They stored in a given DW version only data that are new or changed, in a given version and other data related to a parent version, and then shared by its child versions. To model this, a prototype multi version DW was implemented in visual C++.

In [10], provided a new conceptual model to track history but also to compare data, mapped into static structure. In order to keep the links between members versions, they introduce the concepts of Mapping Relationships and Confidence Factor that are used to build a Multi version Fact Table. Then to modify the structure of the temporal multidimensional schema, they provide to four basic operators: Insert, Exclude, Associate and Reclassify. For each change, a new version was defined in order to keep trace and to respect the definition of a DW (time variant). Each version is valid within a time valid interval. This solution was developed with the visual basic interface on the commercial OLAP environment.

2.4 Discussion

All the approaches we have analysed, select, fully or partially updates the materialized view set during a maintenance downtime in a warehousing environment. This set-up phase has two drawbacks because is time consuming and increases the overall costs, then since data warehouse is used as decision support system then such phase can be a disadvantage because the user need particular information as quickly as possible. Indeed, in this thesis we present a process that avoid the set-up phase by exploiting CoDe that allow the user to know a priori the data of interest and the relationships among them 3.

Chapter 3

Conceptual Organization of Report Visualization: The CoDe Paradigm

This chapter outlines a logic paradigm to conceptually organize relevant data and the relationships among them, and proposes a methodology to design visual representations of such relationships given in tabular form by exploiting a visual language named CoDe (*Co*mplexity *De*sign) [14].

The CoDe-based graph composition modeling allows to visualize relationships between information in the same image following the definition of *efficiency* of a visualization given by Bertin [7]: *"The most efficient (graphic) construction are those in which any question, whatever its type and level, can be answered in a single instant of perception, that is, in a single image"*. This representation named *CoDe model* can be considered a high-level cognitive map of the complex information underlying the ground data. The choice of the final visualization layout in terms of standard graphs is left to a visualization interface which provides the necessary implementation constructs.

Information extracted in tabular form using the OLAP operations [12] is visually represented by different graphs that are suitably aggregated to simultaneously visualize the data values and their interrelationships through the CoDe process. With this approach, conceptual links between data become evident improving both the understanding and the management of information stored in the DW.

Expressiveness of the CoDe language is guaranteed by the natural paradigm choice of the *First Order Logic* (FOL) [58].

3.1 The Graphic Language CoDe

In this section we introduce the CoDe syntax and some useful concept in order to outline the CoDe process.

A *report* is a *double-entry table* (see Table 3.1), where *title_name* denotes a single information item, each C_i denotes a *category* and $value_i$ its corresponding value. The tuple *[*$value_1$, ..., $value_n$*]* is referred to as *data series*.

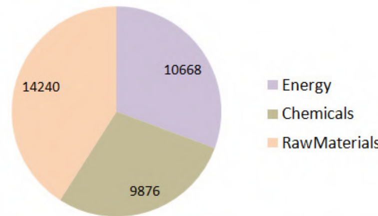
Table 3.1: Graphical representation of a report.

title_name		
C_1	...	C_n
$value_1$	...	$value_n$

Following [7], a *graph* is the visualization of the information carried out by a report. Thus, we call *information item* a graph and its associated report.

Resources (Companies)		
Energy	Chemicals	RawMaterials
10668	9876	14240

(a) *Resources* report.



(b) *Resources* visualization.

$Companies[Energy, Chemicals, RawMaterials]$

(c) CoDe term representing *Resources* item.

Figure 3.1: *Resources* information item and its representation in CoDe.

Fig. 3.1 shows an information item describing the resources consumption (in terms of energy, chemicals and raw-materials) used by a company. In particular, Fig. 3.1(a) depicts the *Resources* report whereas Fig. 3.1(b) shows its corresponding graph in terms of a Pie

standard graph. According to the FOL paradigm, CoDe represents the information item *Resources* as the logical term shown in Fig. 3.1(c).

Summarizing the CoDe language allows to organize visualizations involving more than one type of graphs that have to be composed and aggregated. Through the CoDe model the user can visually represent information items and their interrelationships at different levels of abstraction keeping consistency between items and ground data. The CoDe model can thus be considered a conceptual map of the complex information underlying the visualization of the ground data. The syntax of the CoDe language consist of *Terms*, *Functions* and *Relations*. A *Term* is an array of components, is identified by a name and has associated data extracted from data-mart. In Fig. 3.2, components in the square brackets represent members or dimensional attributes, whilst the name can specify measures and/or hierarchies. In particular, the name All_Sales indicates the measure Sales with the maximum level of aggregation, and the three components denotes the members belonging to the dimensional attribute *Product Family*. Practically, it corresponds to the total sales for the three members.

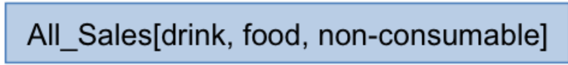


Figure 3.2: Definition of the term All_Sales.

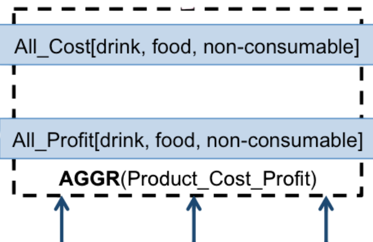


Figure 3.3: Example of AGGREGATION function.

A *Function* is adopted to link terms provided as input by defining constraints and correspondences among their components. The function *AGGREGATION* is used to group several terms having the same components into a single term preserving the original data values for each component. The output term includes both the involved terms and the

AGGR label. In Fig. 3.3 we show such function (with label ProductFamily_Sales_1997) that allows to group the sales of the drink, food and non-consumable components for the four quarters in the year 1997.

The function SUM_i has two input terms. As a pre-condition the value of i-th component in the second term is the sum of the data series in the first one. Fig. 3.4 shows three function SUM_1, SUM_2 and SUM_3 associate the sum of all the values of data series drink, food and non-consumable to the term All_Sales.

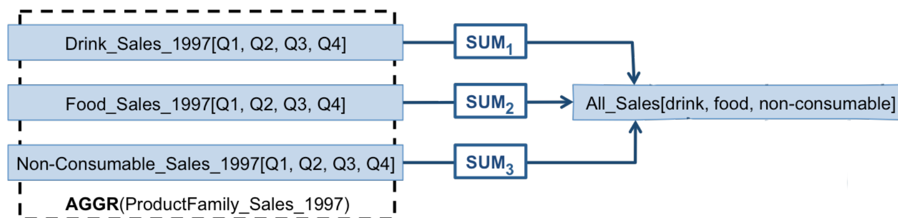


Figure 3.4: Example of SUM function.

The function $NEST_i$ has a symmetric definition with respect to the SUM_i function. It applies to two input terms where one component in a report has a value aggregated from data in the other one.

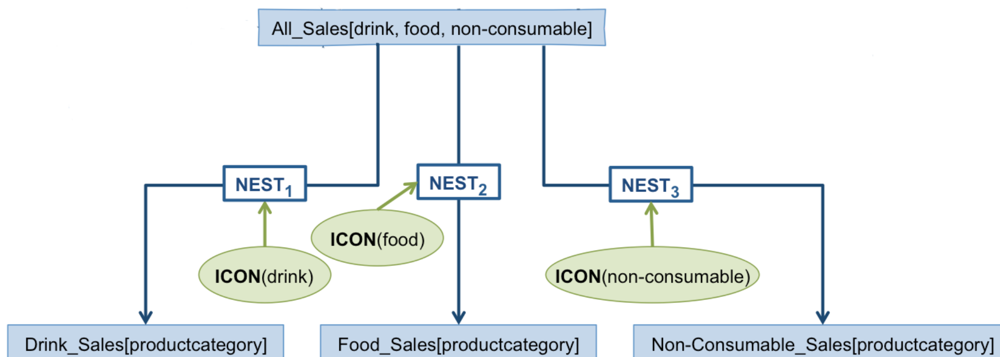


Figure 3.5: Example of NEST function.

In Fig. 3.5 is represented such function, in this example the term All_Sales is given in input to the functions $NEST_1$, $NEST_2$ and $NEST_3$ in order to detail the sales of its three components drinks, food and non-consumable distinct by Product Category. The details are represented by the Drink_Sales, Food_Sales and Non-Consumable_Sales terms. In

addition, to each nest function is applied $ICON$ that adds an icon in the generated report.

The three terms Drink_Sales, Food_Sales and Non-Consumable_Sales on the right side of Fig. 3.10 represent the data series (distinct by product family) containing the values of total incomes made in the stores of three different American states, such as California (CA), Oregon (OR) and Washington (WA).

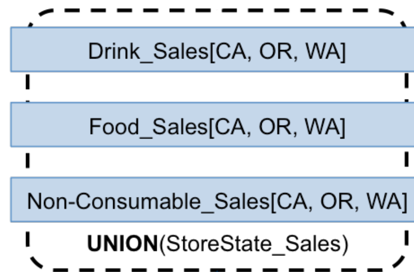


Figure 3.6: Example of UNION function.

The $UNION$ function is the same as the aggregation function, but different sets of components are allowed in the input terms. The representation of the output CoDe term is denoted by the UNION label. In Fig. 3.6, the three reports Drink_Sales, Food_Sales and Non-Consumable_Sales has been aggregated in the term StoreState_Sales by using this function.

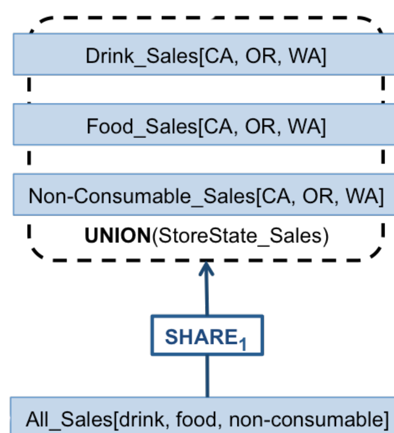


Figure 3.7: Example of SHARE function.

The function $SHARE_i$ shares all the n components of the input term on the i-th position of the other n output ones, respectively. In the example in Fig. 3.7 this function is defined between All_Sales and StoreState_Sales.

A *Relation* is a logic connection existing between two terms. An example is shown in the top right part of Fig. 3.8 where the term *All_Sales(CA, OR, WA)* is related with the term *StoreState_Sales* through a thick arrow. The first term represents the report relative to the total receipts of the sales made on any family of products from the shops located in the CA, OR and WA states.

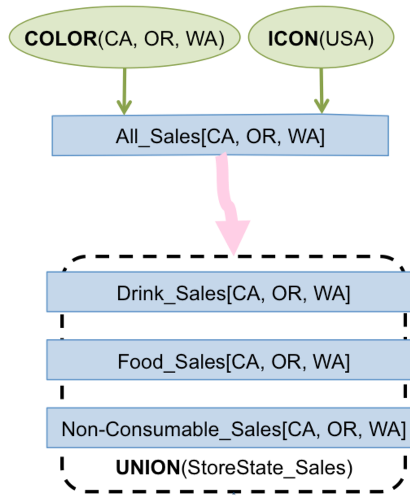


Figure 3.8: Example of Relation, ICON and COLOR.

The output is a graphical representation of the input terms augmented with links among correspondences of each couple of components. Moreover, $ICON$ and $COLOR$ are applied on All_Sales(CA, OR, WA). In particular, the ICON(USA) displays in the final visualization the total incomes of the sales on the United States geographical map, and COLOR(CA, OR, WA) highlights the three states with different colours on such map.

The two terms *All_Profit* and *All_Cost* represent profits and costs respect with the total sales for each product family. These data series are aggregated with the $AGGREGATION$ function, which groups them for each product family. The recursive relation applied on the aggregation, adds to the final visualization the total incomes related to the sales for the product families drink, food and non-consumable.

The function $EQUAL_i$ has two input terms $T_1[D_1, ..., D_i, ..., D_h]$ and $T_2[C_1, ..., C_j, ..., C_n]$,

where the i-th component of term T_1 is equal and has the same associated value of the j-th component of T_2. At the top side of the Fig. 3.10 the three functions $EQUAL_{1,1}$, $EQUAL_{2,2}$, and $EQUAL_{3,3}$ and the $AGGREGATION$ named *Product_ Cost_ Profit*, define an identity between the value of the i-th component of All_Sales and the i-th component given in output by the recursive *Relation* applied on the $AGGREGATION$ function ($\forall i = 1, 2, 3$).

Finally, the bidirectional relation between the two terms *All_ Sales* produces two different visual representations and one-to-one graphical links among the their components.

Further details on the CoDe visual language syntax are available in [56].

3.2 The CoDe process

Different approaches have been presented to perform data visualization [65, 41]. They allows to view the data with different types of visual representations and to switch from a display to another, but they maintain the visualizations separated and not connected to each other. The conceptual visualization obtained in the *CoDe process* through the CoDe model represents information and their relationships at different levels of abstraction in the same visualization.

The CoDe process is composed of four phases as detailed in the following and showed in Fig. 3.9:

1. *Code Modeling.* It produces as output the cognitive map describing information items and their relationships. This phase is performed by the company manager, which is the expert of the specific domain.

2. *OLAP operation pattern definition.* It is used to define the sequences of operations needed to extract all the information.

3. *OLAP operation.* The OLAP operation patterns are mapped into OLAP queries, which are used to extract in a tabular form the information from the data-mart.

4. *Report Visualization.* It produces the final visualization that represents all data as a single image [7].

In Fig. 3.9 rounded rectangles represent process phases, whilst rectangles represent intermediate artifacts produced at the end of each phase.

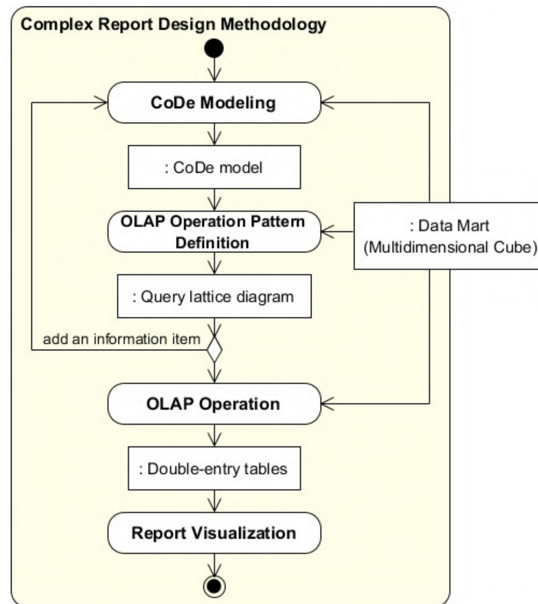


Figure 3.9: The CoDe process.

3.2.1 CoDe Modeling phase

The first phase concerns the CoDe model generation. The business manager produces a model composed of the main concepts and relations that represents the cognitive map of information to be visualized. In Fig. 3.10 we show an example of CoDe model.

3.2.2 OLAP operation pattern definition

The **OLAP Operation Pattern Definition** and **OLAP Operation** phases dynamically generate reports corresponding to information items of the CoDe model. These reports are extracted from a data mart represented as a multidimensional cube.

The construction of a report from a data mart maps the cube dimensions on a structure composed by one horizontal axis (corresponding to the components in the report) and one or more vertical axes (corresponding to the data series values). The resulting report is extracted by applying a combination of selection and/or aggregation *slicing/dicing/pivoting/rolling/drilling* dimensional operators (i.e., the OLAP operations that allow multi-dimensional data analysis) [12]. We define *operation pattern* the combination of OLAP operations to be performed. Operation patterns are expressed considering only meta-data

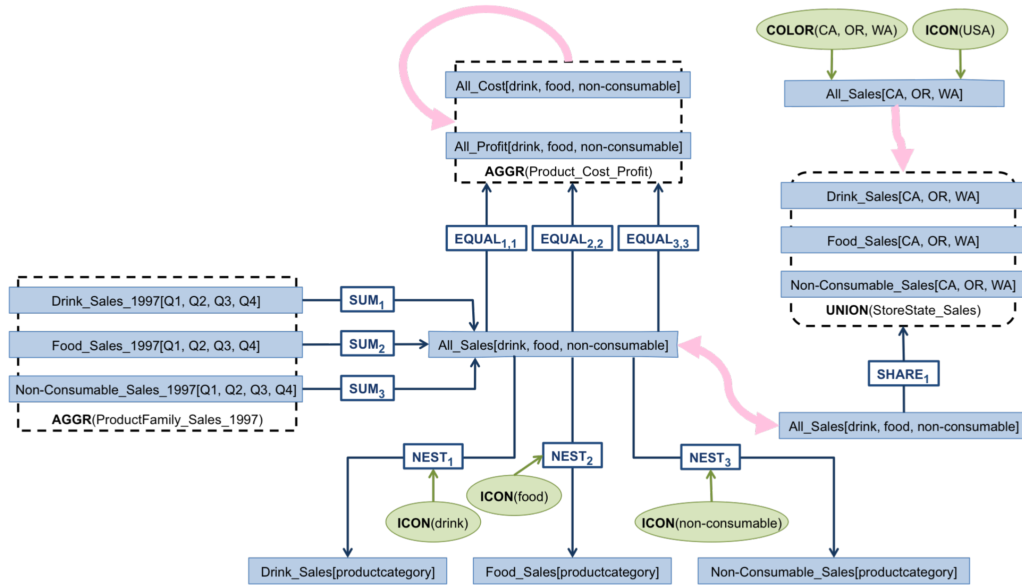


Figure 3.10: CoDe model for the data-mart *Sales*

of the data mart. The actual execution of operation patterns to extract data is performed during the OLAP Operation phase at the end of the design process.

The operation pattern to extract information for any *CoDe Term* is shown in the first row of Table 3.2, where each label represents a single OLAP operation, whilst the symbol h (resp. v) in the parentheses denotes the horizontal (resp. vertical) axis on which the operation is performed (the multiplicity is expressed by the $*$ symbol). The parameter m after the symbol *;* represents the set of members (separated by the comma) on which the OLAP operation is performed. In particular, $pivoting(h \cup v; m)$ is used to rotate the dimensional members m on any single dimension, $rollup(h; m)/rollup(v; m)$ or $drilldown(h; m)/drilldown(v; m)$ are performed on the horizontal/vertical axis in order to decrease or increase the details of the set m, respectively, $dicing(h; m)$ is performed on the horizontal axis to select a subset of dimensional members m and to exclude the others (if present), and $slicing(v; m)$ is executed on the vertical axis to reduce the number of selected dimensional members to the ones in m.

Five steps labeled from *(a)* to *(e)* are used to build the report. In particular: *(a)* $pivoting(h \cup v)$ is used to rotate the cube on any dimension, *(b)* $rolling(h)$ or $drilling(h)$ are performed on the horizontal axis in order to decrease or increase the details of the report categories, respectively, *(c)* $dicing(h)$ is performed on the horizontal axis to select

Table 3.2: Summary of the mapping among Code syntax and OLAP operation patterns.

CoDe syntax	**OLAP operation pattern**
CoDe Term	$pivoting(h \cup v)^*[rollup(h)\vert drilldown(h)]^*dicing(h)[rollup(v)\vert drilldown(v)]^*slicing(v)$
SUM_i	$rollup(h)^*pivoting(h_i \cup v)dicing(h_1 \ldots h_i \ldots h_n)$
$NEST_i$	$slicing(h_1 \ldots h_{i-1}h_{i+1} \ldots h_n)drilldown(h_i)^*$
$EQUAL_{ij}$	$slicing(h_1 \ldots h_{i-1}h_{i+1} \ldots h_n)pivoting(h_j \cup v)dicing(h_1 \ldots h_j \ldots h_m)$
$SHARE_i$	$\forall j = 1, \ldots, n \quad slicing(h_1 \ldots h_{j-1}h_{j+1} \ldots h_n)pivoting(h_i \cup v)]dicing(h_1 \ldots h_i \ldots h_m)$
AGGREGATION / UNION	$pivoting(h \cup v)^*[rollup(h)\vert drilldown(h)]^*dicing(h)\ [rollup(v)\vert drilldown(v)]^*slicing(v)$
RELATION	$\forall j = 1, \ldots, n \quad slicing(h_1 \ldots h_{j-1}h_{j+1} \ldots h_n)\ drilldown(h_i)^*$

To ease the readability the OLAP operation patterns, we have omitted the set of members on which the OLAP operations is performed.

a subset of dimensional attributes and to exclude the others, *(d)* $rolling(v)$ or $drilling(v)$ are similar to *(b)* but they are performed on the vertical axis, and *(e)* $slicing(v)$ is executed on the vertical axis to reduce the number of selected dimensions.

Fig. 3.11(a) shows the multidimensional cube (with four dimensions: *Companies*, *Resources*, *Locations*, and *Pollution*) providing information about the production by companies located in Italy with respect to resources employed and pollution produced, whereas Fig. 3.11(b) shows the application of a specific instance of the operation pattern given in Fig. 3.12 to extract the report displayed in Fig. 3.1(a).

The first OLAP operation is *pivoting*, which rotates the cube to place the *Resources*

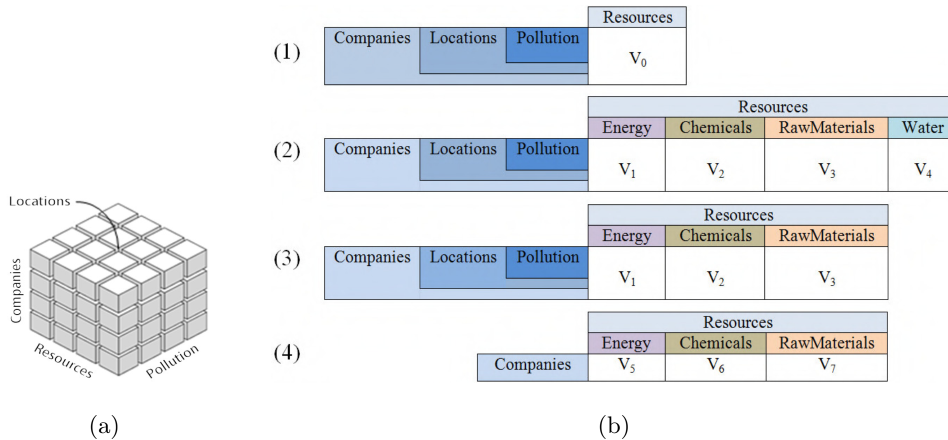


Figure 3.11: A multidimensional cube (a) and OLAP operations (b) performed to extract the report in Fig. 3.1(a).

OLAP operation pattern for the *CoDe Term*
$pivoting(h \cup v; m)^*[rollup(h; m)\|drilldown(h; m)]^*$
$dicing(h; m)[rollup(v; m)\|drilldown(v; m)]^* slicing(v; m)$

Figure 3.12: The operation pattern for the *CoDe Term*.

dimension on the horizontal axis. Remaining dimensions of the cube are considered on the vertical axis (see subfigure *(1)* of Fig. 3.11(b)). A *drilling* operation is performed on the horizontal axis (i.e., *Resources*) in order to increase the details (see subfigure *(2)*). A *dicing* operation on the horizontal axis is then performed to select a subset of *Resources* attributes by excluding the *Water* attribute (see subfigure *(3)*). Since *Companies* represents the final dimension attribute to be computed, *rolling* or *drilling* operations are not needed on the vertical axes (i.e., *Companies*, *Locations* and *Pollution*) to aggregate data or to increase the details, respectively. Finally, a *slicing* operation on the vertical axis reduces the dimensions to *Companies* (see subfigure *(4)*) and produces the report.

A careful reader could observe that the order of OLAP operations performed to obtain the same report may not be unique. The OLAP operation patterns allow to organize the visualization at a suitable abstract level without taking into account the ground data. In other words, the effective queries are a consequence of the visualization design phase, and not vice versa, since they are applied when the CoDe model design activity ends.

The OLAP Operation Pattern Definition phase produces a set of operation patterns able to generate reports corresponding to CoDe terms defined in the CoDe model. Since OLAP operations can share data, we can find a partial order between these operation patterns. We thus organize the set of operation patterns as a *query lattice* structure [40,29] to determine in what order OLAP operations have to be executed.

More precisely, considering a multidimensional cube C, with d dimensions, let us denote with op_A and op_B two OLAP operation patterns applied on C that respectively provide two dimensional attribute tuples $A = (a_1, a_2, ..., a_d)$ and $B = (b_1, b_2, ..., b_d)$, where each a_i and b_i are attributes in the hierarchy for the i-th dimension. Then, a partial ordering $\preceq$ on the set Q of the OLAP operation patterns can be defined by setting $op_A \preceq op_B$

if and only if for any i, $1 \leq i \leq d$, the dimensional attribute a_i in A can be computed by OLAP operations given in op_A applied on the attributes in B.

The partial ordering $\preceq$ allows to define a *lattice* on the set Q since any couple of OLAP operation patterns has a least upper bound and a greatest lower bound. Moreover

Resources.Energy (Companies)			
Diesel	Electricity	Fuel	Methane
762	5715	423	3768

Figure 3.13: The *Resources.Energy* report.

the empty operation pattern, which corresponds to the overall DW is the top element with respect to the partial ordering relation $\preceq$ [29].

As an example, let $op_{Resources}$ and op_{Energy} be the two OLAP operation patterns respectively providing report *Resources* in Fig. 3.1(a) and report *Resources.Energy* in Fig. 3.13. The latter specifies resources employed by companies in terms of energy consumption. The related attribute tuples are in the following hierarchical relation:

$$\begin{aligned} &(Companies, \mathbf{Energy}, none, none) \preceq_d \\ &(Companies, \mathbf{Resources}, none, none) \end{aligned} \tag{3.1}$$

Since *Pollution* and *Locations* dimensions are not considered in the reports we specify *none* attributes in the relation (3.1).

The operation pattern $op_{Resources}$ is described in Fig. 3.11(b). In order to define the operation pattern op_{Energy} we can start from the attribute tuple provided by $op_{Resources}$. A *dicing* operation allows to select *Energy* from *Resources* attributes, then a *drilling* operation increases details providing the actual data of the *Resources.Energy* report.

We thus can assert that $op_{Energy} \preceq op_{Resources}$ and that the computation of op_{Energy} made from the results of $op_{Resources}$ reduces the number of OLAP operations performed on the multidimensional cube.

It is worth noting that CoDe modeling does not negatively affect the OLAP query implementation, thus it does not degrade the performances. Moreover, the CoDe modeling of graph composition exploiting the query lattice structure allows to optimize the OLAP operations.

3.2.3 OLAP Operation phase

The **OLAP Operation** phase extracts data from the multidimensional cube through the OLAP operation patterns organized in the *lattice* structure. OLAP processing could

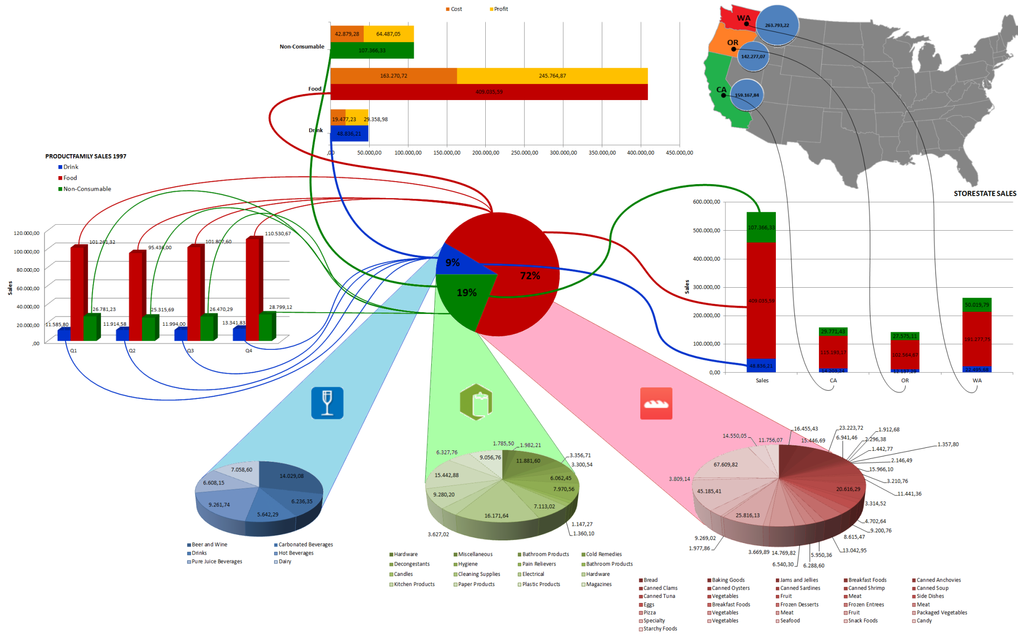


Figure 3.14: Graphical representation of the data-mart *Sales*

be slow so the use of the *lattice* structure improves the performance reducing the total number of OLAP operations to be performed. In fact, by exploiting the partial ordering relation between OLAP operation patterns we compute the results of some OLAP operation patterns starting from the results of another one. Thus we find the minimal set of OLAP operations generating the required OLAP operation patterns. This set allows to reduce the access to raw data in order to provide the required reports [29].

3.2.4 Report visualization phase

The **Report Visualization** phase displays the visualization of extracted reports and their relationships according to the CoDe model. In Fig. 3.14 we show the generated report from the CoDe model of Fig.3.10. In particular, the visualization is implemented taking into account the units of related data series in order to preserve the ratio between the quantitative data extracted from the multidimensional cube. During this phase the designer selects the type of standard graph to draw the reports and places them in specific locations of the drawing area. Moreover, additional information labels or visual symbols can improve visualization details.

Chapter 4

The VSP problem

In the early eighties, researchers started to investigate the issue about the optimization of the data warehouse process. Some of the most crucial aspect in very large databases is to reduce query response time, and improve the DW performance and scalability, for efficiently supporting the process of decision making, as it is a key to gain competitive advantage for business company. In order to speed up the DW process the use of materialized views is a common technique [5] as access to a set of materialized views is much faster than recomputing it. The appropriate set of view is the set with the lowest query processing cost, but there are other parameters to take into account. One of them can be howmany viewsw can be materialized. Materializing all the workload queries give the lowest query processing cost but the highest view maintenance cost because these views have to be maintained in order to keep them consistent with the source data. Another parameter to take into account is the storage space occupied by these view. Indeed, the set of views that answer to the workload queries can be too large for the available storage space. Thus, there is a need for selecting a set of views to materialize by considering all of three parameters: query processing cost, view maintenance cost and storage space. The problem of choosing which views to materialize by choosing the right trade off between these three parameters is known as the view selection problem.

4.1 Problem Formulation

The problem of view selection (VSP) is known to be NP-complete by a reduction from minimum set cover [34] and it can be formulated as follows. Given a query workload $Q = \{q_1, q_2, \ldots, q_q\}$ defined over the lattice L composed of n nodes (i.e., $v_1, \ldots, v_n$), the

problem is to select an appropriate set of views to materialize $M = \{mv_1, mv_2, \ldots, mv_m\}$ (with $m \leq n$) such that Q is answered starting from M with the lowest processing cost under a limited amount of resources, e.g., storage space [42].

4.1.1 Cost model

The cost model is an important issue for the view selection process [13]. The main objective in view selection problem is the minimization of the weighted query processing cost, defined by the formula:

$$Query\ Processing\ Cost = \sum_{Q_i \in Q} f_{Qi} * Q_c(Q_i, M) \tag{4.1}$$

where f_{Q_i} is the query frequency of the query Q_i and $Q_c(Q_i, M)$ is the processing cost corresponding to Q_i given a set of materialized views M.

Because materialized views have to be kept up to date, the view maintenance cost has to be considered. This cost is weighted by the update frequency indicating the frequency of updating materialized views. The view maintenance cost is computed as follows:

$$View\ Maintenance\ Cost = \sum_{V_i \in M} f_u(V_i) * M_c(V_i, M) \tag{4.2}$$

where $f_u(V_i)$ is the update frequency of the view V_i and $M_c(V_i, M)$ is the maintenance cost of V_i given a set of materialized views M . The cost model is extended for distributed setting by taking into account the communication cost which is the cost for transferring data from its origin to the node that initiated the query. Given a query Q_i which is asked at a node N_j and denoting by V_k a view used to answer Q_i, the communication cost is zero if V_k is materialized at N_j.

4.2 Data structure for the view selection problem

In view selection problem, data structures are used as support to represent the view selection. In the following subsections we present some of the most commonly directed acyclic

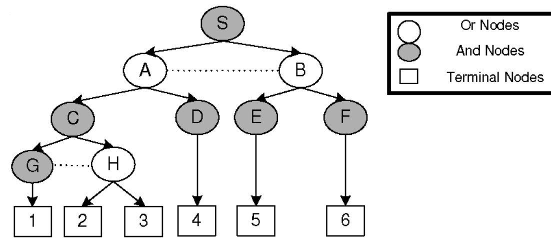


Figure 4.1: An example of AND-OR view graph.

graphs (DAG) used in the literature to represent all the views [29, 71, 72, 23, 57] and select the set of views to materialize.

A *Directed Acyclic Graph* is a finite directed graph with no directed cycles, which is formed by a collection of vertices and directed edges, with each edge directed from one vertex to another, such that there is no way to start at any vertex V and follow a sequence of edges that can loops back to V again. Equivalently, a DAG is a directed graph that has a topological ordering where every edge is directed from earlier to later in the sequence

4.2.1 AND / OR Graph

The AND-OR view graph described by Roy [57, 23] is a Directed Acyclic Graph (DAG) composed of two types of nodes: Operation nodes and Equivalence nodes. An operation node represents an algebraic expression (i.e. Select-Projection-Join). An equivalence node represents a set of logical expressions that are equivalent (i.e., that give the same result). The AND-OR view graph is structured as follows. An operation node has only equivalence nodes as children and an equivalence node has only operation nodes as children. The root nodes are the query results and the leaf nodes represent the base relations.

An example of AND-OR view graph is shown in Figure 4.1, where circles represent operation nodes and boxes represent equivalence nodes. An AND view graph is a graph with only one way to answer a query. On the contrary an OR view graph is an AND-OR view graph in which any node is an equivalence node that can be computed from any one of its children. Summarizing, if there is only one way to answer a query or update the graph is said AND View Graph, and we have a single execution plan for each query, otherwise, if there are multiple ways for every query we talk about OR View Graph.

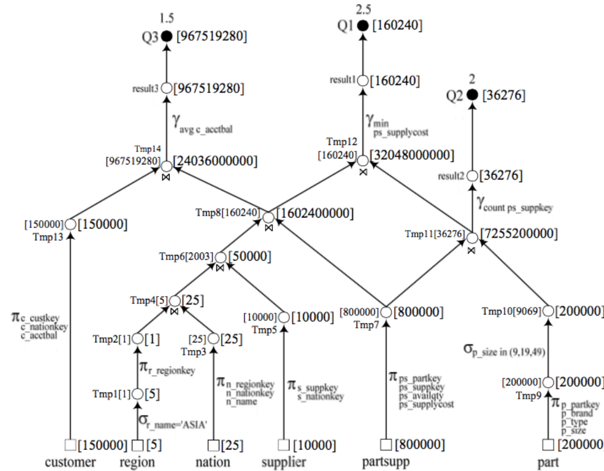


Figure 4.2: An example of MVPP.

4.2.2 Multi-View Processing Plan (MVPP)

The MVPP defined by Yang et al. in [72, 71] is a directed acyclic graph where the root nodes represent the queries, while the leaf nodes are the relations, and all other intermediate nodes represent algebraic expressions as selection, projection, join or aggregation views that contribute to the construction of a given query. Any vertex which is an intermediate or a final result of a query is denoted as a view. The cost for each operation node is usually labelled at the right hand side of each node. The query access frequencies arelabelledd on the top of each query node An example of MVPP is shown in Figure 4.2.

The difference between an MVPP representation and an AND-OR graphs is that an intermediate node in the MVPP exclusively represents an algebraic operation.

4.2.3 Lattice

On-line analytical processing (OLAP) systems builds data cubes with multiple dimensions. Data cubes are made up of two elements: dimensions and measures that represent the actual data values. Most OLAP systems can build data cubes with many more dimensions. Harinarayan and al. [29] propose a structure to model such multiple dimensions data cube called Lattice. The Data Cube Lattice is a DAG whose nodes represent queries (or views) and the edges represent the derivability relation between views. Such relation is denoted

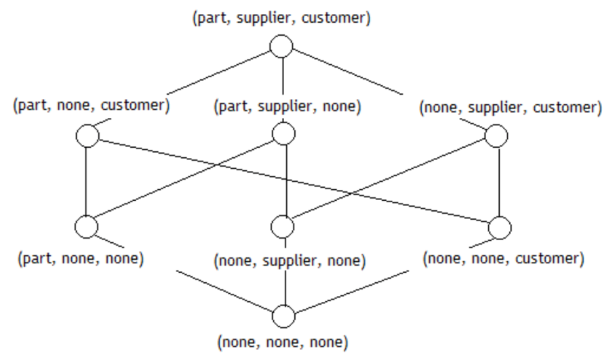


Figure 4.3: An example of Data Cube Lattice.

by the $\leqslant$ operator. For example by considering two queries Q_1 and Q_2. $Q_1 \leqslant Q_2$ can be defined if Q_1 can be answered using only the results of Q_2. It is said that Q_1 is dependent on Q_2. The dimension of the data cube consists of more than one attribute and the dimensions are organized as hierarchies of these attributes. An example of lattice composed by eight views is shown in Figure 4.3.

Chapter 5

A static approach to VSP problem

In this chapter we consider a set of heuristic and algorithms that exploits the CoDe modeling to find the set of workload queries, to mitigate the view selection problem (VSP) and finally to select the set of views to materialize through two different approaches defined in [29, 62].

The optimization approach is composed of three phases:

A. *Code Modeling.* It produces as output a model describing information items and their relationships. This phase is performed by the company manager that is the expert of the specific domain.

B. *OLAP Operation Pattern Definition.* It is used to define the sequences of operations needed to extract all the information.

C. *OLAP Operation Optimization.* In order to speed-up the data extraction, this phase selects the set of views to be materialized and maps the OLAP operation patterns into OLAP queries, which are used to extract information from the data-mart.

The extracted information is used to display the final report taking into account data series and their relationships [7] according to the CoDe process.

5.1 Code Modelling

The CoDe Modeling phase produces a CoDe model (see Fig. 3.10), describing information items and their relationships. The CoDe model is related to the Dimensional Fact Model of the Sales data-mart showed in Fig. 5.1.

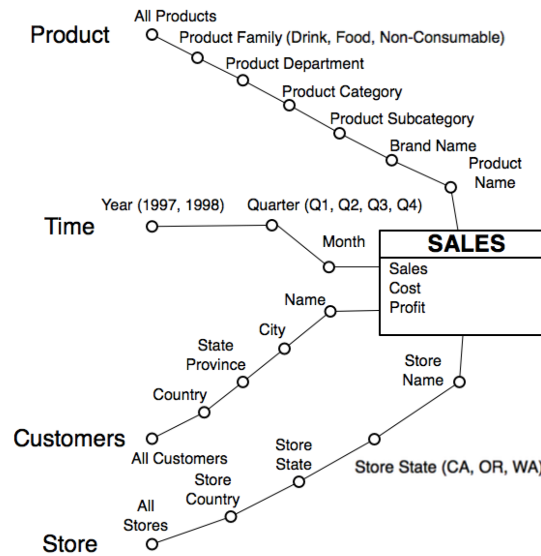


Figure 5.1: Dimensional Fact Model of the *Sales* data-mart.

It consists of a fact schema SALES with measures Sales, Cost, and Profit, and dimensions Customers, Store, Product and Time. The maximum level of aggregation (named ALL_{dw}) is represented by ALL for Products, ALL for Stores ed ALL for Customers, otherwise in the absence of an ALL level, as for the Time dimension, the members at the top are all those contained in the Year level. The company manager, as the expert of the specific domain, generates the CoDe model showed in Fig. 3.10 associated to the Sales data-mart. In particular, Drink, Food and Non-Consumable are referred to the income perceived in the four quarters (labelled with Q1, Q2, Q3, Q4) in the 1997 (measure Store Sales) for the respective products. To these terms is applied the function AGGREGATION (with label PRODUCT FAMILY SALES 1997), which allows grouping the reports drink, food and non-consumable for the four quarters in the year 1997. The term Total_Sales correspond to the total sales for each item family in the year 1997. The three function SUM_1, SUM_2 and SUM_3 associate the sum of all the values of the data series Drink, Food and Non-Consumable to the term Total_Sales. The latter is given in input to the functions $NEST_1$, $NEST_2$ and $NEST_3$ in order to detail the values of its three components Drinks, Food and Non-Consumable in the cumulative data set. The functions give in output the new three reports Drink_Category, Food_Category and Non-Consumable_Category. In addition, to Total_Sales is applied ICON that represents the three components with an icon identi-

fying the final display of the report produced by CoDe. The tree terms Drink_StoreState, Food_StoreState and Non-Consumable_StoreState on the right side of Fig. 3.10 represent the data series containing the values of the total incomes made in the stores of three different American states, such as California (CA), Oregon (OR) and Washington (WA) in the year 1997, always distinct by product family. These reports have been aggregated using the UNION function, instead the bidirectional function LINKS relates them with the Sales_StoreState term which represents the report relative to the total receipts of the sales made on any family of products from the shops located in California, Oregon and Washington. Moreover, ICON and COLOR are applied on the Sales_StoreState term, in particular ICON(USA) displays in the final visualization the total incomes of the sales on the geographical map of the USA, and COLOR(CA, OR, WA) highlights the three states indicated in parentheses on such map. The $SHARE_1$ function defined between Total_Sales and Drink_StoreState, Food_StoreState and Non-Consumable_StoreState builds a complex graphic in the final visualization. The two terms Total_Profit and Total_Cost represent the profits and the costs respect with the total sales for each product family in the 1997. These data series are aggregated with the function AGGREGATION, which groups the profits and the costs for each product family. The recursive LINK relation add to the final visualization the total incomes (measure Store Sales) related to the sales for the product families Drink, Food and Non-Consumable. Finally, the three functions $EQUAL_{1,1}$, $EQUAL_{2,2}$, and $EQUAL_{3,3}$ and the AGGREGATION named PRODUCT COST AND PROFIT, define an identity between the value of the i-th component of Total_Sales and the i-th component given in output by the LINK relationship on the AGGREGATION function.

5.2 OLAP Operation Pattern Definition

This phase takes in input the CoDe model in Fig. 3.10, computes the sequence of operation patterns and then provides as output a lattice representing the set of candidate views to materialize in the DW. Such phase is composed of three steps (see Fig. 5.2).

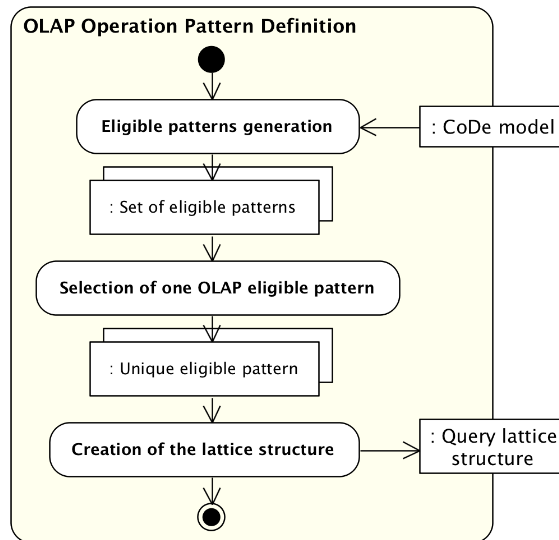


Figure 5.2: The OLAP Operation Pattern Definition phase.

5.2.1 Eligible patterns generation

Starting from the CoDe model, this phase selects the attributes in the DFM on which a finite sequence of OLAP operations has to be executed (named *eligible pattern*). The eligible pattern is determined taking into account the OLAP operation patterns for each term and for every term derived from the functions and/or relations in the CoDe model. Two examples of eligible pattern generation for the $CoDe\ Term$ and for the SUM_i function are provided in Algorithm 1 and Algorithm 2, respectively. The Algorithm 1 generates the eligible pattern for the $CoDe\ Term$. In particular, it computes the set of attributes to perform the OLAP operation by exploiting the input/output attributes of a term (lines 2-9), while at line 10 the eligible pattern is created by following the OLAP pattern defined for that term. The symbol ! denotes that the OLAP operation produces the same result independently from the order of dimensional members on which it is applied, while the symbol ∪ denotes the operations that can be executed in any order. Finally, at lines 11-12 the computed dimensional members and the eligible pattern are associated to the term. Similarly, the Algorithm 2 computes the set of attributes needed to perform the OLAP operations by exploiting the input/output attributes of the terms involved in the SUM function.

5.2.2 Selection of one OLAP eligible pattern

The previous phase generates a set of eligible patterns for each term, function and relation in the CoDe model. This phase adopts the heuristic strategy to select one eli-

Algorithm 1 Eligible pattern for the *CoDe Term*.

Require: The CoDe model

1. **for all** *term* in the CoDe model **do**
2. $cols_1$={set of measures: Sales, Cost, Profit}
3. $rows_1$={set of dimensional members that belongs to ALL_{dw}}
4. $cols_2$={set of dimensional members used to compute the column attributes of the *term* (i.e., components)}
5. $rows_2$={set of measures/hierarchies used to compute the row attributes of the *term*}
6. $pivoting_{v/h}$={set of dimensional members of $cols_1/rows_1$ present in $rows_2/cols_2$ to compute the *vertical/horizontal* pivoting}
7. $slicing_v$={set of dimensional members of $rows_1$ not present in $rows_2$ to compute the vertical slicing}
8. $dicing_h$={set of dimensional members used to compute the col attributes of the *term*}
9. $drilling_{h/v}$={set of dimensional members represented by ancestors recursively computed of the dimensional members in $cols_2/rows_2$ present in $cols_1/rows_1$}
10. $eligible = [pivoting(h; pivoting_h) \cup pivoting(v; pivoting_v)]!$ $[drilldown(h; drilling_h)]![dicing(h; dicing_h)]$ $[drilldown(v; drilling_v)]![slicing(v; slicing_v)]!$
11. $term.rows/cols = eligible.rows/cols$
12. $term.eligible_pattern = eligible.olap_pattern$
13. **end for**

Algorithm 2 Eligible pattern for the SUM_i function.

Require: The SUM_i function between the T_1 and T_2 terms in the CoDe model

1. $cols_1/rows_1$={set of dimensional members used to compute $T_1.cols/T_1.rows$}
2. $cols_2/rows_2$={set of dimensional members used to compute $T_2.cols/T_1.rows$}
3. $rolling_h$={set of dimensional members of $cols_1$ to be aggregated to obtain the set of attributes of $cols_2$}
4. $pivoting_i$={the i-th attribute of $rows_2$}
5. $pivoting_v$={set of dimensional members of $rows_1$}
6. $dicing_h$={set of dimensional members of $cols_2$}
7. $eligible = [rolling(h; rolling_h)]!$ $[pivoting(h; pivoting_i) \cup pivoting(v; pivoting_v)]!$ $[dicing(h; dicing_h)]$
8. $SUM_i.eligible_pattern = eligible.olap_pattern$

Algorithm 3 LCP(S, root).

Require: $S = \{s_1, s_2, \ldots, s_n\}$, root

1. **for all** s_i in S with 0 characters **do**
2. addLabelNode(root, "s_i")
3. remove s_i from S
4. **end for**
5. **if** $|S| == 1$ **then**
6. addLabelNode(root, "s_1"(s_1))
7. remove s_1 from S
8. **end if**
9. **if** $|S| == 0$ **then**
10. **return**
11. **end if**
12. V = sort(unique_characters(S))
13. i = 1
14. **while** $|S| \geq 1$ **do**
15. $c = V[i]$
16. S_c = select strings in S starting with c
17. remove S_c from S
18. **if** $|S_c|$ >1 **then**
19. node=createChild(root)
20. setLabelEdge(root, node, c)
21. **else**
22. node=root
23. **end if**
24. remove the first character c from the strings in S_c
25. $LCP(S_c$, node)
26. i++
27. **end while**
28. **return**

gible pattern from each set. The selected one is the longest common prefix (LCP) of OLAP operations also considering their permutation. This strategy is implemented in the Algorithm 3. In order to simplify the description of the algorithm, we represent each OLAP operation in the eligible pattern with a unique label. For example, let the eligible pattern $[pivoting(h; m_1, m_2, ..., m_n)]!slicing(v; m_k)$ and the labels $a = pivoting(h; m_1)$, $b = pivoting(h; m_2)$, $c = pivoting(h; m_n)$, $d = slicing(v; m_k)$, thus, the eligible pattern is represented as the string $\{a, b, c\}, d$ where the a, b, c labels can be swapped.

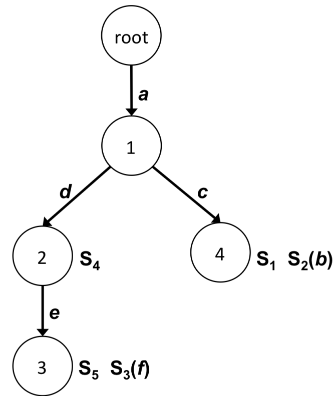


Figure 5.3: The prefix tree.

The Algorithm 3 builds a prefix tree and adopts a greedy strategy to selects the prefixes, through a breadth-first search on the maximum number of strings exchangeable that share a prefix, and a depth-search on the maximum length of the common prefixes. The *LCP* algorithm is used by Algorithm 4 to determine one of OLAP eligible patterns, split it into a sequence of OLAP operations, associate a unique label to each single OLAP operation, and finally select the longest common prefix. As an example, given the sets of eligible patterns: $S_1 = \{a, c\}; S_2 = \{a, b, c\}; S_3 = \{a, d\}, e, f; S_4 = \{a, d\}, f; S_5 = \{a, d\}, e$. The algorithm *LCP* generates the prefix tree in Fig. 5.3, obtaining the set of unique eligible patterns $S_1 = a, c; S_2 = a, c, b; S_3 = a, d, e, f; S_4 = a, d, e; S_5 = a, d, f$.

Algorithm 4 Selection of OLAP eligible patterns.

Require: Set E of all eligible patterns

1. map each OLAP operations in E with an unique character
2. root = create a new node
3. *LCP*(E, root)
4. **for all** string $s_i \in E$ **do**
5. prefix = concatenation of characters on the path from the root to the node labeled with s_i
6. remove from the selected string s_i the characters in *prefix*
7. build the unique OLAP eligible pattern by concatenating prefix with the string s_i
8. **end for**

5.2.3 Creation of the lattice structure

A lattice structure is created by using the OLAP eligible patterns. The lattice is a directed acyclic graph and is defined as follow: *i)* Each node in the lattice represents a view that has to be computed on the DW. *ii)* Let u and v two views, each edge between u and v represents an OLAP operation that applied on u computes v. *iii)* There exists a partial order $\preceq$ between views in the lattice: $v \preceq u$ if and only if v can be computed starting from u *(dependency)*. *iv)* There is the aggregated view "ALL" in the lattice, upon which every view is computed. The ancestors of v is the set $\{s|s \preceq v\}$. Thus, v is computed starting from any of its ancestors, i.e., the views it transitively depends on, applying the sequence of OLAP operations specified on the edges among any s and v. To tackle the state-space explosion problem [68], the views of the lattice structure are merged by exploiting the common prefixes of unique eligible patterns.

Table 5.1: Coefficients to compute V_s and P_c.

OLAP Operation	$\mathbf{Rows_v}$	$\mathbf{Cols_v}$	**Coefficient** $\mathbf{m_o}$
pivoting(h) or pivoting(v)	$rows$	$cols$	$2.5 * 10^{-3}$
slicing(h)	1	$cols$	$5.0 * 10^{-3}$
slicing(v)	$rows$	1	
dicing(h)	$rows - 1$	$cols$	
dicing(v)	$rows$	$cols - 1$	
rollup(h)	$rows/2$	$cols$	$10.0 * 10^{-3}$
rollup(v)	$rows$	$cols/2$	
drilldown(h)	$2 * rows$	$cols$	
drilldown(v)	$rows$	$2 * cols$	

The coefficient o is $5 * 10^{-3}$ and it is fixed for all the OLAP operations (m_o and o are expressed in secs.).

To define the processing cost of a view, we compute a cost model by considering the processing cost of each single OLAP operation that produces that view. This processing cost (i.e., P_c) is a linear function applied on the view size (i.e., V_s) and it is expressd by the formula $P_c = m_o * V_s + o$, where m_o is a multiplying coefficient depending on the OLAP operation and o is a fixed cost (e.g., the overhead of running a query on a negligible DW size). These two coefficients have been empirically determined by executing all the

five OLAP operations on different DWs. The view size V_s is computed by multiplying the $Rows_v$ and $Cols_v$ coefficients (i.e., the number of rows and the columns obtained from the OLAP operation). Table 5.1 summarizes the coefficients for computing the view size and processing costs in terms of OLAP operations.

5.3 OLAP Operation Optimization

This phase takes as input the lattice and gives as output the set of views to be materialized. Figure 5.4 details the two steps composing this phase.

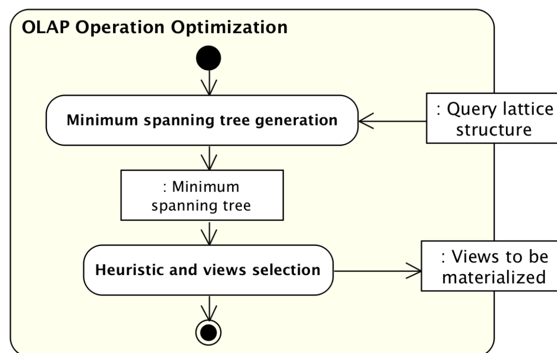


Figure 5.4: The OLAP Operation Optimization phase.

5.3.1 Minimum spanning tree generation

The problem of view selection (*VSP*) is known to be NP-complete by a reduction from minimum set cover [34] and it can be formulated as follows. Given a query workload $Q = \{q_1, q_2, \dots, q_q\}$ defined over the lattice L composed of n nodes (i.e., $v_1, \dots, v_n$), the problem is to select an appropriate set of views to materialize $M = \{mv_1, mv_2, \dots, mv_m\}$ (with $m \leq n$) such that Q is answered starting from M with the lowest processing cost under a limited amount of resources, e.g., storage space [42]. Since the lattice structure does not have an unique path from the root to each node, a MST is computed to select the paths that starting from the root node generate the views (answering the workload query Q) with the lowest processing costs. This is performed by considering as weights the set of P_c required to compute OLAP operations among each couple of directly connected nodes in the lattice. Then, each *node* in the MST has associated a couple of weights that

represent the view processing cost and the view size. These values are computed as follows: *i)* $T(root, node)$ is the processing cost of all the OLAP operations from the root to *node*, and is the given by:

$$\sum_{op \in path(root,node)} P_c(op) \tag{5.1}$$

where *path* computes the sequence of OLAP operations among two nodes in the MST. *ii)* Starting from the rows and columns of the view represented by the root of MST, $S(node)$ is computed as the product of $Rows_{node}$ and $Cols_{node}$. These values are updated taking into account the OLAP operations present in the path from root to *node* and coefficients in Table 5.1.

5.3.2 Heuristic and views selection

On the MST, the *VSP* can be reduced to an optimization problem where we are interested to minimize the following objective function:

$$\min \sum_{q_j \in Q} T(mv_r, v_j) \tag{5.2}$$

$$where \sum_{mv_i \in M} Size(mv_i) \leq S \quad and \quad r \in 1, \ldots, m$$

and $T(mv_r, v_j)$ is the processing cost of the view v_j that answers the query q_j, starting from the materialized view mv_r and S is the available total storage space. The problem can be approached by following two steps: *1)* Calculate the processing cost and the view size for each node in MST where $M = \emptyset$ (i.e., no materialization has been computed); *2)* Add in M the set of nodes present in the MST that minimizes the objective function and respects the space constraints [31, 67].

Many algorithms have been proposed to select properly the set M [1, 29, 25, 11, 2, 61]. Most of all focused on the concept of a *benefit* metric and differ from each others in the definition of this metric. Informally, the benefit to materialize a view is the savings we obtain choosing to materialize such view instead of another one. Given an instance of the VSP problem, Shukla *et al.* [62] introduce a benefit metric called *Average Query Cost* (*AvQC*) computed as follow:

$$\sum_{j=1}^{n} p_j \cdot T(mv_j, v_j) \tag{5.3}$$

where $p_1, \ldots, p_n$ represent the probabilities that queries $q_1, \ldots, q_n$ occur. These queries are answered by the views $v_1, \ldots, v_n$, starting from the materialized view $mv_1, \ldots, mv_n$. This approach differs from ours because the authors want to minimize the ratio between the size of the query to pick up, and the probability of its occurrence. In the case the materialized views have not an equal probability of being queried, the user has to assign such probabilities. However, the frequency (i.e., the probability) a materialized view is expected to be queried is not always a priori known, and this frequency may change during the DW process.

Differently, Harinarayan *at al.* define in [29] two different benefit metrics taking into account the processing cost and the space occupied by the materialized views. The first one is defined as follows and is used in the Algorithm 5.

Let v a view in the MST, for each view $w \preceq v$ (i.e., w in the MST that covers v):

$$Inv(v, M) = \sum_{w \preceq v} I(v, w, M) \tag{5.4}$$

$$where\ I(v, w, M) = \begin{cases} T(u, w) - T(v, w) \\ \qquad if\ T(u, w) > T(v, w) \\ 0 \qquad otherwise \end{cases}$$

and u is the materialized view in M with the lowest cost that is covered by w. Summarizing, the cost of evaluating w by using v is compared wrt. The cost of evaluating w by using a materialized view u. If v helps (i.e., the cost of v is less than the cost of u), then the

Algorithm 5 HRU_T

Require: The MST, k

1. $M = \{root\}$
2. **for** $i = 0$ **to** k **do**
3.   **if** $\exists v \in MST \backslash M$ that maximizes $Inv(v, M)$ **then**
4. $M = M \cup v$
5.   **end if**
6. **end for**
7. **return** M

Algorithm 6 HRU_S

Require: The MST, s

1. $M = \{root\}$
2. **while** $s > 0$ **do**
3.   **if** $\exists v \in MST \backslash M$ that maximizes $Inv(v, M)$ and $s - Size(v) > 0$ **then**
4. $M = M \cup v$
5. $s = s - Size(v)$
6.   **end if**
7. **end while**
8. **return** M

difference represents part of the benefit of v in case it is selected as a materialized view. The total benefit $Inv(v, M)$ is the sum over all views that cover v. The algorithm HRU_T maximizes the benefit (line 3), by adding the selected view in the set M (line 4) until the fixed limit (i.e., k) on the number of view to materialize is reached. At line 7 the algorithm returns the set M containing the selected view to materialize.

The second benefit metric, is defined as follows and is used in the Algorithm 6.

$$InvS(v, M) = \frac{Inv(v, M)}{Size(v)} \tag{5.5}$$

The metric considers the view space occupied by M, and is calculated as the ratio between the investment to compute v and all its descendant, and the space to materialize it. The algorithm HRU_S maximizes the total benefit (line 3), by adding the view v in M (line 4) as long as the upper-bound of the disk space (i.e., s) is not been reached. The algorithm does not consider the space occupied by the root which is always materialized. At line 8 the algorithm returns the set M containing the selected view to materialize.

Chapter 6

A dynamic approach to VSP problem

The static selection of views contradicts the dynamic nature of a decision support system. Indeed, a company manager looks for data and trends, that are information that changes overtime, thus a static selection of views can quickly become outdated. The DW administrator should monitor these trends and re-calibrate the DW scheme and its data, then he should check the consistency of the set of materialized views as it can evolve, consequently. This task can be hard and time consuming. To mitigate this situation we exploit the CoDe process to handle the model evolution shown in the following Chapter.

6.1 The CoDe model evolution

In this section we present a dynamic process by exploiting the CoDe paradigm. This process focus to add new items or new components to the CoDe model by optimizing the static CoDe process proposed in 5. The two new phases are showed in Fig. 6.1. **The CoDe Dynamic modeling** phase allows the manager to modify (add, remove or update) the information items present in the CoDe model with the help of a semi-automated editor presented in the following Section 6.1.1.

The optimized lattice creation step optimizes the construction of the lattice structure exploiting shared paths through the SEQ_MATCH algorithm, which selects the subsequence of stings with the most occurrences in common, in order to classify them and build the lattice structure.

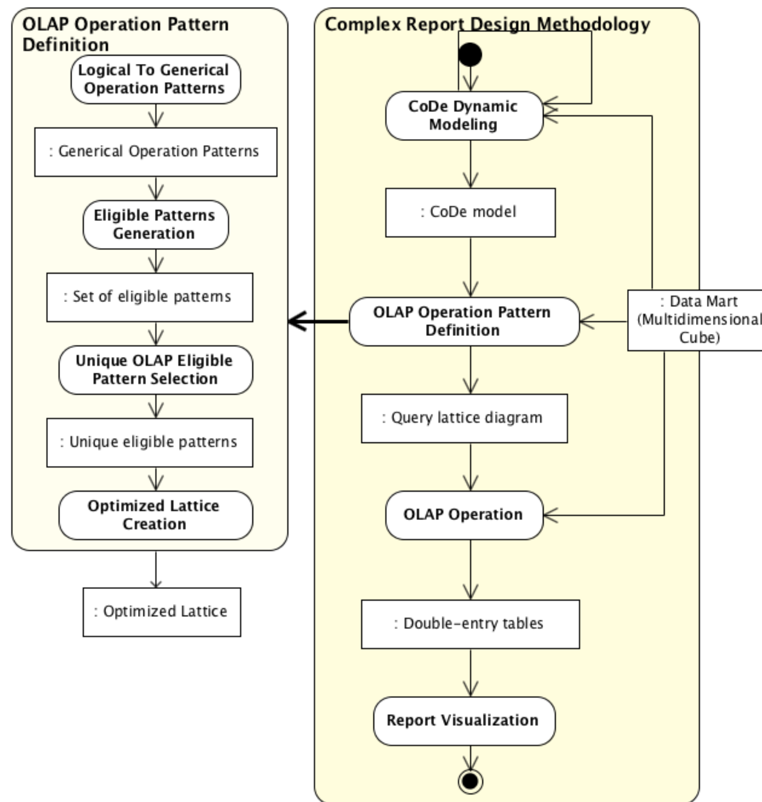


Figure 6.1: The Code Dynamic process.

6.1.1 The CoDe Dynamic modeling

The CoDe Dynamic modeling phase exploits a context-sensitive editor based on a context-sensitive grammar to support the manager in designing step by step a new model starting from the old one. The goal of such editor is to help the manager in making the best choice by listing all the possible options, sorted exploiting the min-max strategy and a check validation function. In the next subsections all the listed techniques are described.

The context-sensitive editor

In order to introduce the editor and show the evolutionary process, we take into account the Sales data-mart (presented in Chapter 5, Fig. 5.1) and analyses the sales of the day 24 February 1997. In Table 6.2 are shown the actual data extracted from the FOODMART DW, while Fig. 6.3 shows the CoDe model that we want to update.

The editor takes in input the model, that have only one information item named

	Store Sales
Product Family	24
Drink	
Food	36.02
Non-Consumable	17.99

Figure 6.2: Report on the Feb 24 1997 for the product family drink, food and non-consumable.

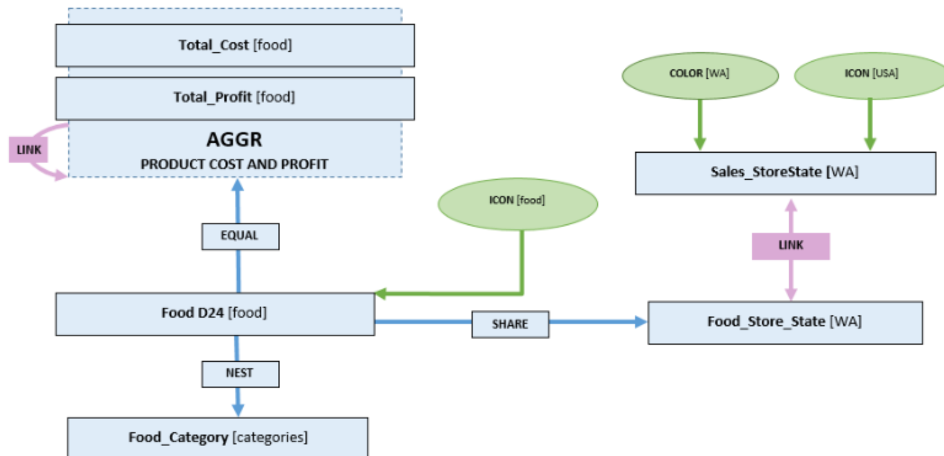


Figure 6.3: CoDe Model for the information item Food D24.

$FoodD24[Food]$, then it shows a graphic window (see Fig. 6.4) containing all the possible suggestions to build dynamically the model. Each option has a priority value, given from the min-max strategy, for example the addition of a new component has a higher priority with respect to the addition of a new information item.

The manager selects the new component $FoodD25[Food]$ that is added to the model, then the editor presents the next option window, shown in Fig. 6.5.

Every time a new item is added to the model the check validation function is called. At this point, the next choice is to add the SUM function that automatically generates the $Total_Sales[Food]$ information item. The obtained model is shown in Fig. 6.6.

Figure 6.4: Addition of a new component.

Figure 6.5: Addition of SUM function.

Figure 6.6: Addition of EQUAL function.

The manager adds the EQUAL function (Fig.6.7) and by selecting the $Total_Sales[Food]$ information item the editor allows him to create a new item. Then the item named $Total_Profit$ and a component named $Food$ are added to the CoDe model (see Fig. 6.8).

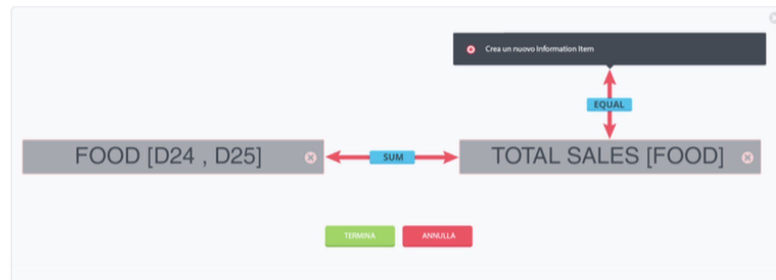


Figure 6.7: Addition of an item for the EQUAL function.

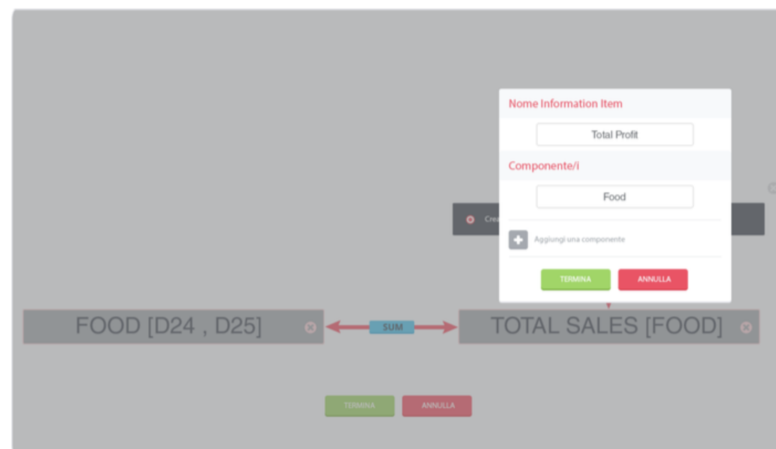


Figure 6.8: Addition of the item $Total_Profit$.

If the manager tries to add another component to $Total_Profit$ (as we can see in Fig. 6.9), the editor does not allow this action showing an error message (Fig. 6.10). The generated model is show in Fig. 6.11.

By selecting the item $Total_Sales[Food]$ the manager can apply the NEST function. The editor shows the following window Fig.6.12 to allow him to choose on which item (hierarchically inferior with respect the $Total_Sales[Food]$ item) apply this function. Once selected the NEST element, the action is performed and the component is created. Fig.

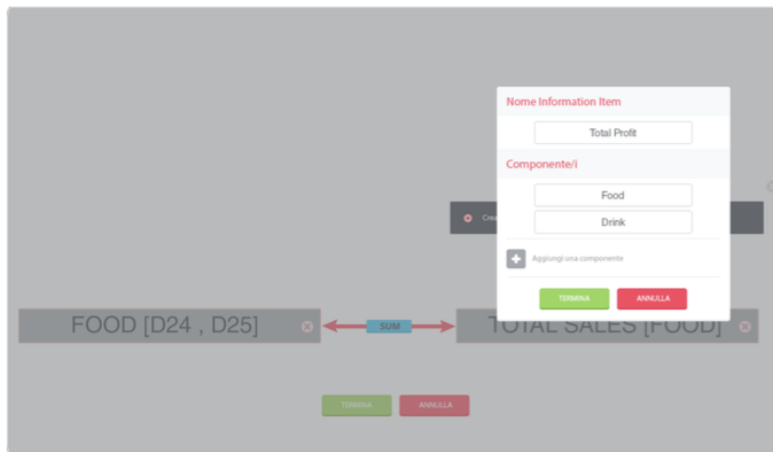


Figure 6.9: Addition of another component to the item.

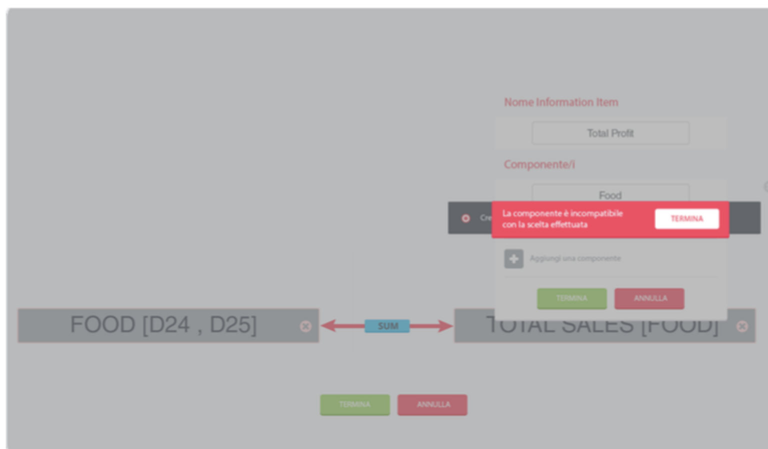


Figure 6.10: Error message.

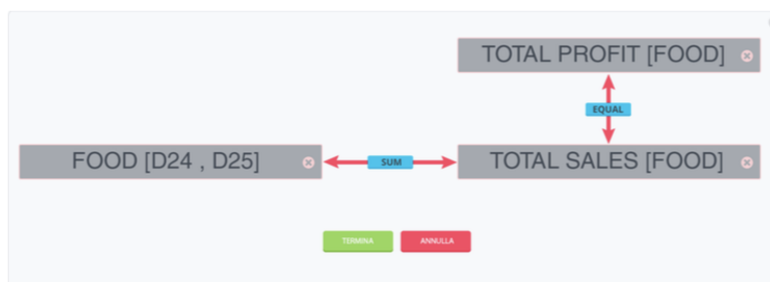


Figure 6.11: Code Model with the $Total_Profit$ item.

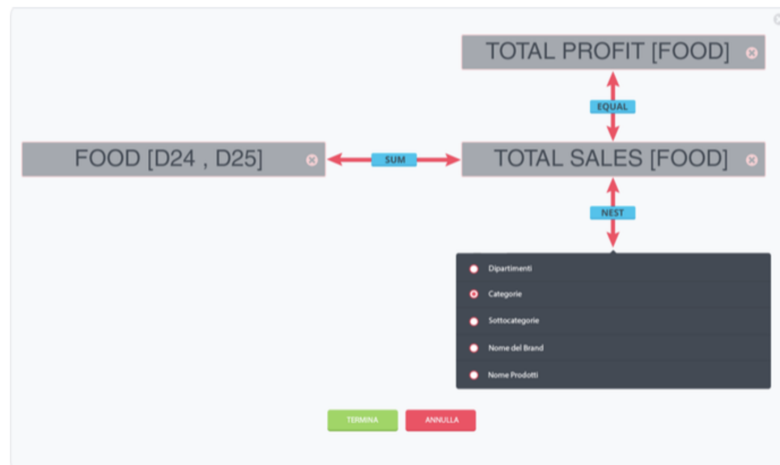


Figure 6.12: Addition of the NEST function.

Figure 6.13: Final CoDe model.

6.13 shows the final CoDe model that will be given in input to the OLAP operation pattern definition phase. It is worth to notice that in the control window, the manager is always allowed to terminate or cancel the design phase.

Context-sensitive grammar definition

In this subsection we introduce the context-sensitive grammar that is built according to the syntax of CoDe paradigm. A context-sensitive grammar is needed to build the model, by providing the right syntax rules. For example, if the manager wants introduce the SUM function, the grammar leads the editor to build the model in conformity with the SUM rules defined in Chapter 3.

The production 1 and 2 create an Information Item and a component that are associated

Table 6.1: Context-sensitive grammar.

1 =	$A \rightarrow InformationItem$
2 =	$B \rightarrow Element$
3 =	$C_1 \rightarrow A[B_i]\|B\| = 1$
4 =	$C_n \rightarrow A[B_1 \dots B_n]\|B\| > 1$
5 =	$AB \rightarrow C_1$
6 =	$AB_1B_2 \dots B_n \rightarrow C_n$
7 =	$C_1? \cup \dots \cup ? \dots C_1 \rightarrow AGGR_1$
8 =	$C_nU? \cup \dots \cup ? \dots C_n \rightarrow AGGR_n$
9 =	$C_1\|C_n \dots C_1\|C_n \rightarrow UNION_n$
10 =	$D_1 \rightarrow C_1$
11 =	$D_n \rightarrow C_n$
12 =	$G \rightarrow ?B_1 + \dots + ?B_n$
13 =	$SUM \rightarrow A[G]$
14 =	$T \rightarrow [SUM_1 \dots SUM_n]$
15 =	$H_i \rightarrow ?B.children$
16 =	$NEST \rightarrow A[H_i]$
17 =	$D_n \rightarrow NEST$
18 =	$SHARE_i \rightarrow B_i \triangleright C_i$
19 =	$D_i \rightarrow SHARE_i$
20 =	$EQUAL \rightarrow ?B_i = B_j$

to the symbols A and B respectively. The production 3 and 4 build a complete item, the production 3 indicates that the item must contain only one component, while the meaning of the index n indicates that the production 4 can contain two or more components. The production 5 links the symbol A to the symbol B to generate a new Information Item named C_1, production 6 specifies the same but for n components. The productions 7 and 8 are associated with the $AGGREGATION$ function. The production 7 combines the all non-terminals C_1 and aggregates it with the non-terminal $AGGR1$. After each symbol of union $\cup$ is presents the symbol ? that indicates the validation function. Such function checks that each non-terminal C_1 presents in the model has the same component. The production 8 performs the same tasks as the 7 but for n components. The production 9 performs the UNION function, each non-terminal C_i appears as the OR of C_1 and C_n. This allows the function of joining items that have different number of components, but this characteristic

cannot be checked by the validation function. The next two productions, 10 and 11, running a transitive operation by associating the non-terminal C to the non-terminal D. The first one for one component, while the latter is performed for n components. This will be useful to associate the results of the productions to the non-terminal D. The productions 12, 13 and 14 are related to the SUM function. In the production 12 all components B_i present in an information item C_i are summed and the result is associated in the non-terminal G. In this case the validation function checks if each value is numeric. The non-terminal G contains the SUM of all components that belong to an item. Production 13 add this value to an item of a new component then associated to the non-terminal SUM. Production 14 inserts all the results of the all SUM function present in the model in a new item. The production 15 associates to the non-terminal H the children of a component B. The children have a lower hierarchical level with respect the component. The validation function ? on the B component, performs a check if there are lower hierarchical levels and if not H has an empty value. In the production 16 all the components H calculated in the previous step for the component B are associated with a new item. Even in this case a validation function is performed on the value of H, because if has an empty value then the NEST cannot be applied. Production 17 stores the value of the NEST function to the item D_n. Production 18 applies the SHARE function by associating each component to a new information item. The result is stored into the new item D_n (production 19). Finally, the last production defines the EQUAL function, that performs a graphic comparison between two components (B_i and B_j). The validation function checks if these two components occupy the same memory area. It is worth to notice that the ICON and COLOR operator are not been included into the grammar because they are visual operators that enhance the final report and do not manipulate any data series.

Check validation function definition

In the last decades computer systems are evolving by becoming more complex, thus a main challenge is to provide formalisms, techniques, and tools that ensure an efficient design of correct and well-functioning systems despite their complexity. Model checking [9] is a formal automatic verification technique which allows for desired properties of a given system to be verified on the basis of a suitable model through systematic inspection of all states of the model. The system validation is performed in three phases, described in the

following:

- *Modeling phase*: the extraction of the model and its properties definition.
- *Running phase*: run the model checker to check the validity of the property in the system model, the reachability tree generation.
- *Analysis phase*: if the properties are satisfied or violated.

Modeling. The inputs to the model checker are a model of the system under consideration and a formal characterization of the property to be checked. The model describes the behaviour of systems, usually a finite-state automata is used to check if a property is correct or not. States include information about the current values of variables, the previously executed statement. Transitions describe how the system evolves from one state into another.

Running phase. The model checker first has to be initialized by appropriately setting the various options then the actual model checking takes place. This is basically done with an algorithmic approach in which the validity of the property under consideration is checked in all states of the system model.

Analysis phase. This phase concerns analysing the results. So a property can be valid or not. In case the property is valid, the following property can be checked, or, in case all properties have been checked and are all valid, is concluded that the model is correct. Whenever a property is falsified, the negative result may have different causes, this implies the understanding of the cause, a correction of the model, and the restarted of the verification process.

In our solution the states of the finite-state automata are associated with the non-terminals of the grammar. The model checking has been exploited for the following functions:

- NEST
- EQUAL
- AGGREGATION

The validation function on the NEST function checks if the items involved are at the same hierarchical level, while the EQUAL and AGGREGATION function check if they belong to

the same dimension and if they respect the rules defined in the CoDe paradigm (Chapter 3).

Data mining, its techniques and the descriptive statistics

The data extraction or data mining (DM) [66,28], is an information technology whose goal is to find useful information in large collections of data, information that would otherwise remain unknown. In recent years, the sector is undergoing a strong expansion due to the increase in available databases and the interest of the companies that are discovering the potential and the results that are obtained with the use of this discipline. The data mining are generally divided into two broad categories:

- *Predictive use*: the aim of this analysis is to predict a particular attribute (objective function) from known attributes (predictors).
- *Descriptive use*: the goal is to identify recurring patterns (frequent pattern), groups of similar data (cluster), anomalies or sequential patterns that characterize the analysed data.

Following the stage of DM is necessary to use post-processing techniques that allow validating and display the results obtained. The DM provides different types of analysis:

- *Predictive analysis*: this analysis has as the goal to build a predictive model from a set of known attributes. There are two different techniques: classification, used in order to predict the value of discrete variables and the regression, used for continuous variables.
- *Associative analysis*: this analysis is used to identify frequent patterns that describe particular characteristics of the data. The identified patterns are generally expressed in the form of association rules.
- *Based on cluster analysis*: this analysis aims to identify data in a number of groups, called clusters, in which the data are very similar within the same group and significantly different between different clusters.
- *Analysis of anomalies*: this analysis is responsible for identifying small groups of data whose characteristics are significantly different from the others.

The classification technique. Among the DM techniques we take into account the classification technique. The classification technique [35] has the goal of identifying the value assumed by the attribute of an object, starting from the known values of other attributes. This supervised method uses a set of data where the values to predict are known. The data are divided into two groups called the training set and test set. Usually about the 70% of them become part of the training set and the remaining 30% instead set up the test. Through the training set phase, the ranking algorithm tries to deduce the rules to determine the value of the attribute sought, starting from the values of available attributes. When the rules have been identified, the actual validity is checked in order to calculate the percentage of the right predicted values. In our study we predict if, given a new input information, it constitutes a new item, or it can be added as a component of the item that already exists, by using our data we can create the training and the test set then we added to these data an attribute, named *label* that allows classifying the new data in a category. After the phase of creating the rule, there is the validation phase, which consists in check whether or not rules are valid. Fig. 6.14 shows an example of a classification method. In particular, when the item $FoodD24$ and $DrinkD25$ are given in input and they are not present as information items in the model, such method adds two new items to the model. Differently when the item $FoodD25$ is given in input and there is another item with the same label $FOOD$ because a new component is added to the model.

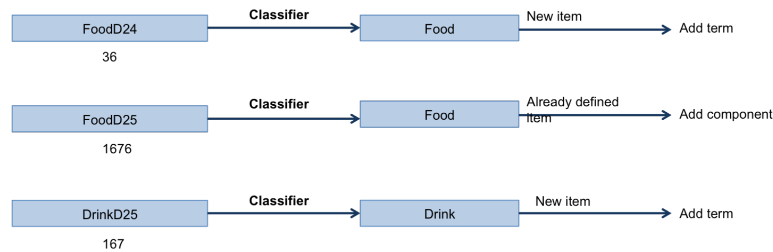


Figure 6.14: Example of classifier method.

The Descriptive statistics. The descriptive statistics [33] is a set of techniques used to describe the basic features of the data collected in an experiment. Such techniques provide a summary of the collected measurements. With descriptive statistics it is allowed to describe, represent and summarize, what is observed or what the data show in their essential features (also named population). The descriptive statistics uses measures to describe

Day	Drink	Food	Non cons.
X_{24}	0	36	17
Y_{25}	167	1676	509

Table 6.2: Application of the min-max technique.

a data set, the most commonly measures are: measures of central tendency (i.e. mean, median and mode) and measures of variability or dispersion (i.e. the standard deviation (or variance), the minimum and maximum values of the variables, kurtosis and skewness). Contrariwise to represent a data set, the most commonly graphical representations used, are: the dotplot, frequency, frequency histogram, absolute frequency, relative frequency, cumulative frequency, boxplot, and probability plot.

In this thesis we exploit the descriptive statistics to properly sort the different options (i.e. add a component, add a function, etc). For example, assuming to take into account two days, 24 and 25 and the three family products Drink, Food e Non Consumable. Assuming to have in the model only the item $FoodD24$, we want to know if we should add a new item (i.e., $NonConsumableD24$) or a new component (i.e., $FoodD25$). The symbol X_{24} represents the day 25 while the symbol Y_{25} the day 25. Applying the min-max technique, the results are shown in Table 6.1.1 that shows the values of each product for the days 24 and 25, respectively. If we analyse the two items $FoodD25$ and not $ConsumableD24$, the first has value of 1676 while the second has value of 17. Thus, according to the min-max technique, the addition of a new component in the model has higher priority with respect to the addition of the new item. Thus, the manager will visualize as first choice, the *Adding a new component* option.

6.1.2 The optimized lattice creation

The *optimized lattice creation* phase is part of the CoDe dynamic process. Thus, after the manager has generated the new CoDe model, we have to reapply the CoDe process, in particular the eligible patterns generation and the selection of the unique eligible patterns.

Successively, we apply such phase in order to minimize the changes on the multidimensional lattice exploiting the previous one.

The lattice optimization phase defines the algorithm SEQ_MATCH that takes in input the old lattice, the old vocabulary table, the new OLAP operations and the new vocabulary table. SEQ_MATCH compares the old and new OLAP operations in order to distinguish three types of states:

- **equal** when the old OLAP operation has the same elements as compared to the sequence of the new OLAP operation. For example: the string **S6** (Equal [24]) = a t m n d with the string **S16** (Equal [24-25]) = a t m n d).
- **similar** when the new OLAP operation has at least one common value starting from the beginning of the sequence. For example: the string **S3** (Total Profit [24]) = a b d e n f g h i j k l and the string **S14** (Total Profit [24-25]) = n a b d e f g h i j k).
- **different** when the first value of the new OLAP operations sequence is different with respect to all old OLAP operations.

The Algorithm 6.1.2 starts by comparing all the new OLAP operation patterns with the old OLAP operation patterns, and makes a recursive call by returning if is there are equal, similar, or different paths. If the selected new pattern is labelled as *different* a new path to the root is added, while if the new pattern is labelled as *similar* the procedure search which is the shared path and add only the remaining different nodes, and finally if the new pattern is labelled as *equal* all the paths will reuse.

Algorithm 7 The SEQ_MATCH Algorithm.

```
Require: V_old, OLAP.opold, V_new, OLAP.opnew
 1. NEW_S =
 2. OLD_S =
 3. n = number of operations in OLD_S =
 4. i = number of operations in NEW_S =
 5. S_i[] = newS_i[n]
 6. for i = 0 to length(NEW_S), i++ do
 7.     for j = 0 to length(OLD_S), j++ do
 8.         SEQ_MATCH_INT(NEW_S[i], OLD_S[j])
 9.         if NEW_S[i] ==' SIMILAR' then
10.             S_i[j]= associated value for similar
11.         end if
12.     end for
13.     a = max(S_i)
14.     index[i] = position(a)
15.     if NEW_S[i] ==' EQUAL' then
16.         return OLD_S[i]
17.     end if
18.     if NEW_S[i] ==' DIFFERENT' then
19.         continue;
20.     end if
21. end for
22. SEQ_MATCH_INT(x : nsequence, y : msequence)
23. LIST_SEQ = emptylist
24. while i > 0 and j > 0 do
25.     if x[i] = y[i] then
26.         LIST_SEQ = x[i] + LIST_SEQ
27.         i = i + 1
28.         j = j + 1
29.     else
30.         return LIST_SEQ
31.     end if
32. end while
33. if length(LIST_SEQ) == n and m then
34.     return 'EQUAL'
35. end if
36. if length(LIST_SEQ) >= 1 then
37.     return 'SIMILAR'
38. end if
39. if length(LIST_SEQ) == 0 then
40.     return 'DIFFERENT'
41. end if
42. return
```

Chapter 7

A probabilistic model to improve the dynamic approach

Since the DW schemes and their data can frequently be changed, the re computation of the associated materialized views does not always guarantee performance improvements in the DW process. This is due to the overhead of the process for generating the new minimal set of views. This chapter presents a solution to alleviate this problem, based on the Markov strategy proposed in [20] and adapted on the CoDe dynamic process. The Markov solution has been chosen because it keeps all the historical information about the gains computed on the previous selected views. In particular, it allows to choose whether or not recalculate the set of views taking into account if there is a gain in computing such new set. Fig. 7.1 show in which point of the CoDe dynamic process the proposed solution, named *Markov analysis and views selection*, is collocated.

7.1 Markov analysis and view selection

The *Markov analysis and view selection* phase is applied every time the manager updates the CoDe model, by adding or removing items, components or functions. The Markov strategy through a probability calculus, selects the set of views to materialize taking into account the impact frequency over the time of the new OLAP queries on the possible set of views to materialize. Thus, the new set of views will be materialized only if the performance will increase with respect to the cost in term of updating time. In particular, we adapt the procedure proposed by [20] and define an algorithm that classifies the views by their importance by computing two probabilities: the *Initial Probability* and the *Stable*

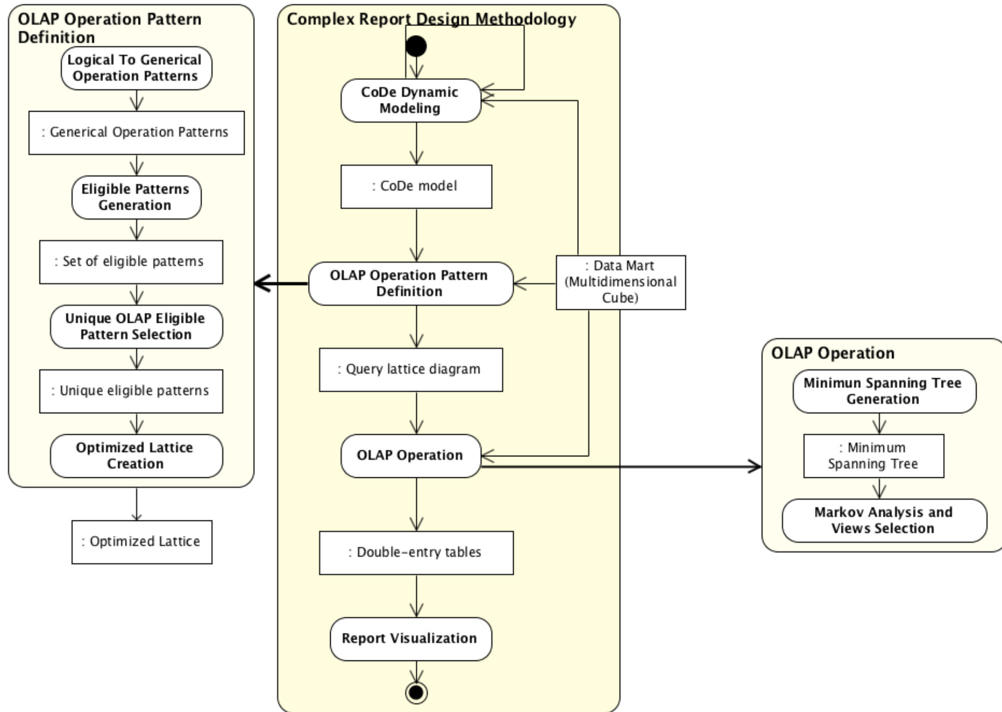


Figure 7.1: The Markov optimization.

Probability, and then selects the views to materialize, as shown in Algorithm 8. In particular, the algorithm takes as input the set of the OLAP patterns Q', the set of views V' computed on a CoDe model, the set of the new OLAP patterns Q, and the new set of views V computed by re-applying the CoDe process on the updated model where items, components or functions have been added or removed. Such algorithm at lines 1-2, builds two *Initial Probability* matrices $Q'_m * V'_n$ and $Q_i * V_j$. Each cell in the matrices is computed by dividing the number of labels (i.e., each OLAP operation pattern and each views is represented by a set of label), calculated on the views and the OLAP patterns that match, by the maximum length of an OLAP pattern. At line 3, the algorithm calculates a new $V * V$ matrix, where its values are probability distributions representing how the materialization of a specific view affects the computation of the other views. The number of rows and columns of the $V * V$ is the number of the views present in the lattice. Such $V * V$ matrix is computed by considering two cases as follows:

Case 1. If items, components or functions have been added in the CoDe model the algorithm calculates the $Q_m * V_n$ starting from the $Q_i * V_j$ matrix by removing the rows

corresponding to the new views and the columns corresponding to the new OLAP patterns. Then, it multiplies the $Q'_m * V'_n$ matrix by the transposed sub matrix $Q^T_m * V_n$ and it adds to the obtained $V * V$ matrix the values of the rows and the columns previously removed.

Case 2. If items, components or functions have been deleted, the algorithm removes from the $Q'_m * V'_n$ matrix the rows corresponding to the views not present in the new lattice and the columns corresponding to the OLAP patterns not present in the workload query set. The $V * V$ matrix is obtained by the product with the new $Q'_m * V'_n$ matrix and the transposed matrix $Q^T_i * V_j$.

Algorithm 8 The Markov algorithm.

Require: $Q', V', Q, V, \varepsilon$

1. $Probability_Matrix_{old} = Initial_Probability_Calculation(Q', V')$
2. $Probability_Matrix_{new} = Initial_Probability_Calculation(Q, V)$
3. $Previous_Transition_Matrix = Initial_Probability_Matrix(Q'_m * V'_n, Q_i * V_j)$
4. $i = 1$
5. $VN_i = \{set\ of\ views\ already\ materialized\}$
6. **repeat**
7. $\quad VN_{i+1} = Previous_Transition_Matrix * VN_i$
8. $\quad lms = leastMeanSquare(VN_{i+1}, VN_i)$
9. $\quad i + +$
10. **until** $lms \geq \varepsilon$
11. **return** $selectViews(VN_i)$

Let S the set of materialized views, the algorithm at line 4, computes the *Impact Probability Vector* VN_1. Such vector contains a value for each view in the lattice. If that view has already been materialized this value is $1/Size(S)$, otherwise is zero. At line 7, the algorithm multiplies the $V * V$ matrix by the vector VN_i until a steady state probability is achieved. This steady state is reached when the least mean square [63] between two consecutive Impact Probability Vectors (i.e., VN_i and VN_{i+1}) will be minor with respect to a fixed threshold ε. Finally, at line 11, the algorithm selects the views in the vector which values are greater than a threshold and taking into account a limit on the storage space. The threshold is computed as the average of the values in the vector.

Chapter 8

Case Study

In this chapter we present several case studies for the three different approaches. We have evaluated the processing time of each operation and the storage space value with the Saiku 2.4 suite [43] installed on laptop with a 2.93 GHz i3 processor, 4 GB of RAM and Windows 7. Then in order to obtain clean values, to avoid the caching effect during the execution of multiple queries we cleaned the memory cache before of each execution.

8.1 Static approach

In this case study, we show the static optimization process on the CoDe model of Fig. 8.1.

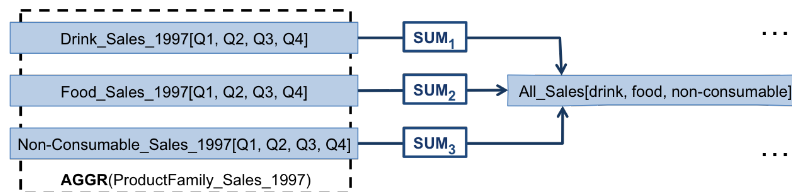


Figure 8.1: The CoDe model.

In such model the terms *Drink*, *Food* and *Non-consumable* representing the data series for each product family sold in the Foodmart stores which are respectively drinks, food products and not edible products. These data series are referred to the sales in four quarters (i.e., *Q1*, *Q2*, *Q3*, *Q4*) of the 1997, and on them is applied an aggregation function (i.e., $AGGR$) that allows grouping the sales of drink, food and non-consumable for four quarters. The *All_ Sales* term represents the cumulative data series of the total sales made in the 1997 for each product family, and the SUM_1, SUM_2, and SUM_3 functions are used to

Table 8.1: Eligible OLAP operation patterns for the term *Drink*.

1)	[pivoting(h; [Time].[1997],[Time].[1998])]!
2)	drilldown(h; [Time].[1997])
3)	dicing(h; [Measures].[Sales], [Time].[1997].[Q1], [Time].[1997].[Q2], [Time].[1997].[Q3], [Time].[1997].[Q4])
4)	drilldown(v; [Product].[All Products])
5)	[slicing(v; [Store].[All Stores], [Customers].[All Customers], [Product].[All Products].[Food], [Product].[All Products].[Not-Consumable])]!

Table 8.2: Eligible OLAP operation patterns for SUM_1.

1)	[rollup(h; [Time].[1997].[Q1], [Time].[1997].[Q2], [Time].[1997].[Q3],[Time].[1997].[Q4])]!
2)	[pivoting(v; [Time].[1997])∪ pivoting(h; [Product].[All Products].[Drink])]!
3)	dicing(h; [Measures].[Sales], [Product].[All Products].[Drink], [Product].[All Products].[Food], [Product].[All Products].[Non-Consumable])

map the sum of data series *Drink*, *Food* and *Non-consumable*, respectively. The first step of the *OLAP Operation Pattern Definition* phase generates the eligible patterns for the four terms *Drink*, *Food*, *Non-Consumable*, *All_Sales*, and for the functions *AGGR ProductFamily_Sales_1997*, SUM_1, SUM_2, and SUM_3. Tables 8.1 and 8.2 show the outputs obtained for the term *Drink* and the function SUM, respectively.

The second step selects the unique OLAP patterns, decomposing the OLAP operation patterns of each dimensional members and by renaming them with a unique labels. The output is a vocabulary table shown in Table 8.3.

The set *Ss* of switchable strings replacing each OLAP operation with a label is built, and

Table 8.3: Vocabulary table.

a =	pivoting(h; [Time].[1997])
b =	pivoting(h; [Time].[1998])
c =	drilldown(h; [Time].[1997])
d =	dicing(h; [Measures].[Sales])
e =	dicing(h; [Time].[1997].[Q1])
f =	dicing(h; [Time].[1997].[Q2])
g =	dicing(h; [Time].[1997].[Q3])
h =	dicing(h; [Time].[1997].[Q4])
i =	drilldown(v; [Product].[All Products])
j =	slicing(v; [Store].[All Stores])
k =	slicing(v; [Customers].[All Customers])
x =	rollup(h; [Time].[1997].[Q4])
l =	slicing(v; [Product].[All Products].[Food])
m =	slicing(v; [Product].[All Products].[Non-Consumable])
n =	slicing(v; [Product].[All Products].[Drink])
o =	pivoting(h; [Product].[All Products])
p =	drilldown(h; [Product].[All Products])
q =	dicing(h; [Product].[All Products].[Drink])
r =	dicing(h; [Product].[All Products].[Food])
s =	dicing(h; [Product].[All Products].[Non-Consumable])
t =	slicing(v; [Time].[1998])
u =	rollup(h; [Time].[1997].[Q1])
v =	rollup(h; [Time].[1997].[Q2])
w =	rollup(h; [Time].[1997].[Q3])
y =	pivoting(v; [Time].[1997])
z =	pivoting(h; [Product].[All Products].[Food])
$\boldsymbol{\alpha}$ =	pivoting(h; [Product].[All Products].[Drink])
$\boldsymbol{\beta}$ =	pivoting(h; [Product].[All Products].[Non-Consumable])

for each switchable string in *Ss*, the OLAP operation patterns is computed by following the path on the prefix tree (see Table 8.4). However, since the *SUM* functions do not share the first input term then the OLAP unique operation, patterns are casually computed respecting the dimensional operation order. The output of this step is shown in the right side part of the Table 8.4.

Table 8.4: Switchable strings and OLAP unique operation patterns.

S_1-Drink =	{a, b} c d e f g h i {j, k, l, m}	a b c d e f g h i j k l m
S_2-Food =	{b, a} c d e f g h i {n, m, j, k}	a b c d e f g h i j k m n
S_3-Non-Consumable =	{b, a} c d e f g h i {j, n, k, l}	a b c d e f g h i j k l n
S_4-All Sales =	o p d q r s {t, j, k}	o p d q r s t j k
S_5-AGGR =	{a, b} c d e f g h i {j, k}	a b c d e f g h i j k
S_6-SUM$_1$ =	{u, v, w ,x} {y, α} d q r s	u v w x y α d q r s
S_7-SUM$_2$ =	{u, v, w, x} {y, z} d q r s	u v w x y z d q r s
S_8-SUM$_3$ =	{u ,v ,w ,x} {y, β} d q r s	u v w x y β d q r s

The last step aims to build the lattice structure (see Fig. 8.2). In such structure, the edges represent OLAP operations, the nodes correspond to the generated views, and each view has a path which starts from the root. The *OLAP Operation Optimization* phase generates the MST from the lattice structure. Figure 8.2 shows the MST with the view space and the processing cost computed for each node in according to the proposed cost model, the nodes coloured in grey represent the views of the workload. Once generated the MST, the views to materialize by applying the solution proposed by [29, 62] are selected. The HRU_T procedure (with $k = 3$) selects the views: *root*, v_3, v_9 and v_{18}. The HRU_S procedure with storage space $s = 81$ (given multiplying 3 by the average size of views in the MST), selects the views: *root*, v_9, v_{11}, v_{23}. Finally, the *AvQC* algorithm proposed by Shukla *et al.* [62], by assuming that all aggregates have an equal probability of being queried, selects the views: *root*, v_{10}, v_{23}, v_{24}.

Table 8.5 summarizes the results of the three algorithms. In particular, *Size(M)* indicates the size occupied by the materialized views, *Time(V, M)* indicates the total processing time to produce all the views V in the MST, whilst *Time(Q, M)* is the processing time of the materialized views that answer to the workload queries. The algorithm HRU_S reduces the total processing time to answer the workload queries and the used storage space (78MB wrt. HRU_T that uses 171MB). On the contrary, HRU_T has a better performance in term of processing time when all the views have to be calculated. Moreover, *AvQC* has similar results in term of storage space and worst processing time wrt. HRU_S.

Table 8.6 shows the results of the three algorithms when two consecutive executions are performed. In particular, the processing times are improved in the range of 34-36% in the second execution. Moreover, HRU_S gives better performance than the HRU_T, reducing its processing time to 0.34 seconds. The overall improvement of HRU_S is 62% after the

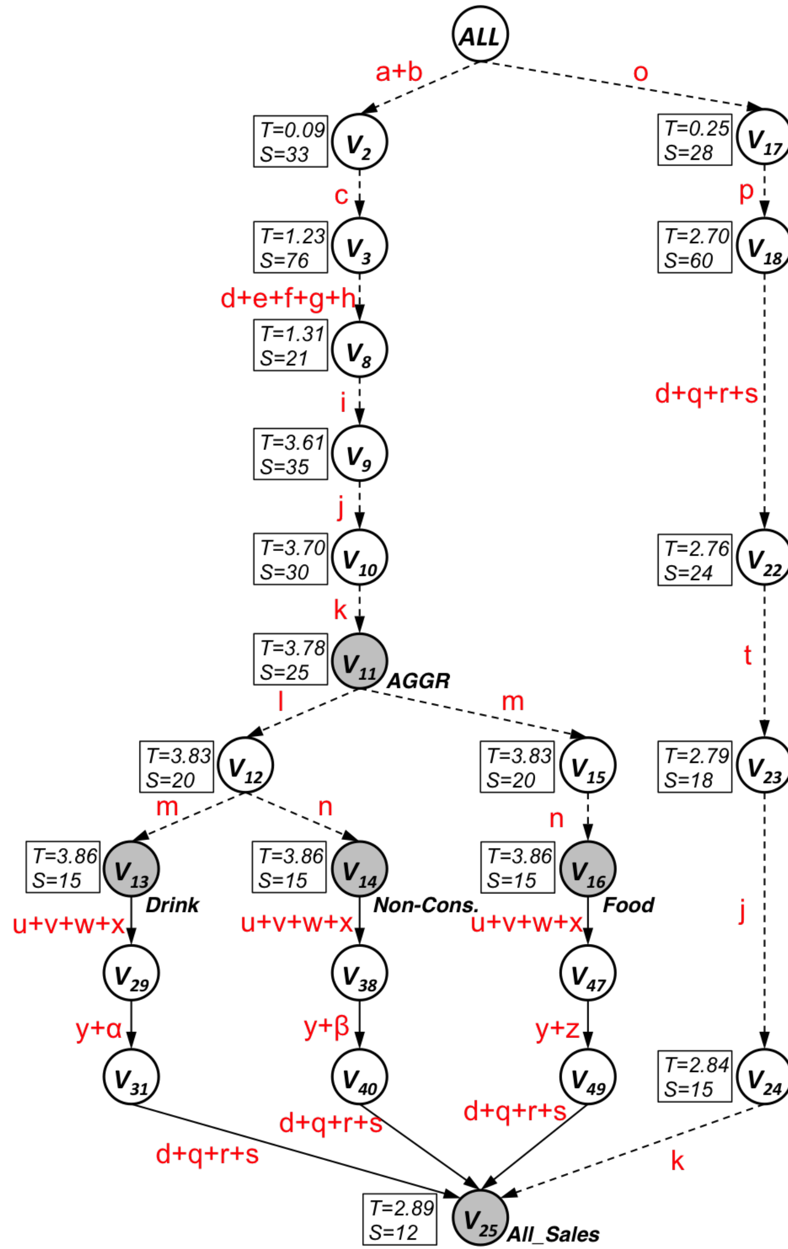


Figure 8.2: The lattice structure with the OLAP operations and the corresponding MST (dashed arrows).

first execution and 98% after the second one, comparing with the algorithm that does not materialize views. The improvement of HRU_S is of 5%wrt. the two other algorithms.

To demonstrate the scalability of the proposed process on the entire *Sales* data-mart, we have used the whole CoDe model in Fig. 3.10 obtaining comparable results. Indeed,

Table 8.5: Processing time and storage space of adopted algorithms applied on the lattice structure of Fig. 8.2.

	No Mat.	**AvQC**	$\mathbf{HRU_T}$	$\mathbf{HRU_S}$
Selected views	*root*	*root*, v_{10}, v_{23}, v_{24}	*root*, v_3, v_9, v_{18}	*root*, v_9, v_{11}, v_{23}
Size(M)	-	66MB	171MB	78MB
Time(V, M)	47.19s	16.55s	8.66s	15.49s
Time(Q, M)	18.25s	7.38s	7.42s	6.91s

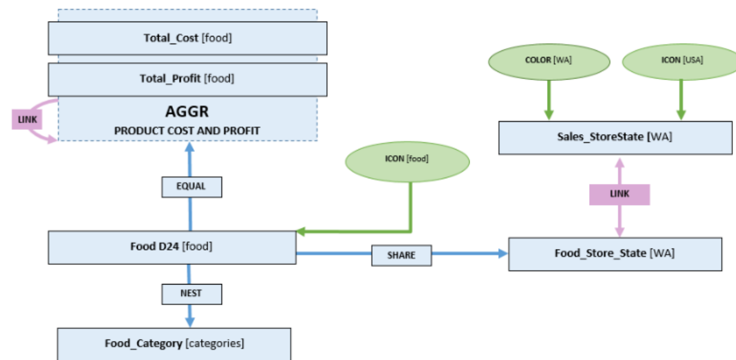
Table 8.6: Comparison of processing times of two consecutive executions.

	No Mat.	**AvQC**	$\mathbf{HRU_T}$	$\mathbf{HRU_S}$
$\mathbf{1^{st}}$ **execution-Time(Q, M)**	18.25s	7.38s	7.42s	6.91s
$\mathbf{1^{st}}$ **execution-Profit**	-	59%	59%	62%
$\mathbf{2^{nd}}$ **execution-Time(Q, M)**	18.25s	1.09s	1.11s	0.34s
$\mathbf{2^{nd}}$ **execution-Profit**	-	93%	93%	98%

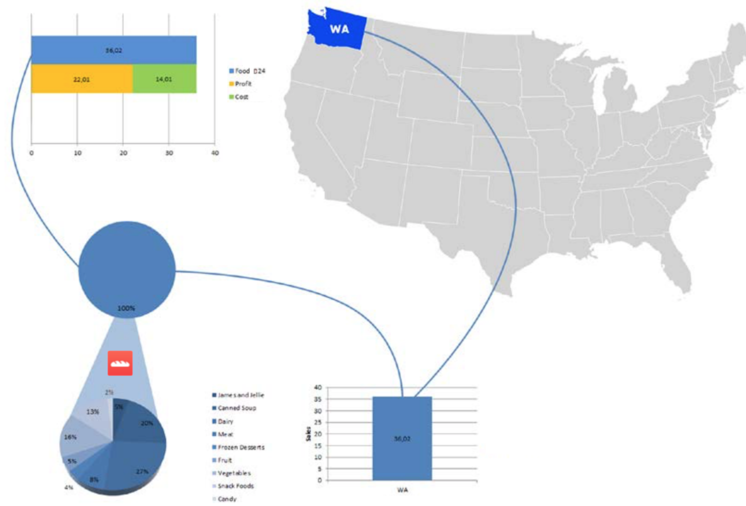
Table 8.7: Algorithms evaluation on the entire *Sales* data-mart by using the CoDe model in Fig.3.10.

	No Mat.	$\mathbf{HRU_T}$	$\mathbf{HRU_S}$
$\mathbf{Size_p(M)}$	-	327MB	177MB
$\mathbf{Time_t(V, M)}$	95.05s	45.41s	32.28s
$\mathbf{Time_t(Q, M)}$	38.10s	18.75s	15.16s
Profit	-	51%	60%

the algorithm HRU_S (with $s = 180$) reaches an improvement of 60% wrt the algorithm that does not materialize views and reduces the processing time obtaining an improvement of 9% with respect the algorithm HRU_T exploiting the materialized views when workload queries have to be answered. In addition, HRU_S uses less storage space than the other algorithm. In conclusion, the results assess that the algorithm HRU_S maintains good performance also on a complex CoDe model.



(a)



(b)

Figure 8.3: CoDe model for the data-mart *Sales* concerning the cost and profit of the food category in the WA state for the day 24 (a), and its graphical representation (b).

8.2 Dynamic approach

In order to show the dynamic process, the optimization process has been applied on the CoDe model of Fig. 8.3(a). Successively, we provide to add a new component to the model and present the dynamic optimization process.

In particular, this model regards the cost and profit of the food category of markets in the Washington state (i.e., WA), but taking into account the fixed day (i.e., February 24, 1997) for the product family food sold in the Foodmart stores. The NEST function

increases the level of detail of the product *Food*, it is applied on the $Total_Sales$ term and give as output the report $Food_Category$. The visual operator ICON represents the component $Food_Category$ with a highlighted icon that is showed in the final report. The $Food_Store_State$ term contains the total incomes of the all sales, subdivided by product type, made in the Washington (WA) stores. While the $Sales_Store_State$ term represents the total incomes of the all sales for any family products, in Washington (WA). These two terms are connected by the function LINK. The visual operators ICON and COLOR are applied on the $Sales_StoreState$ term. The ICON operator is used to show on the USA map, for any states take into account, the total incomes for any family products. The COLOR (WA) operator highlights such states. The AGGREGATION function joins the two data series $Total_Profit$ and $Total_Cost$, representing the profit from sales and the cost of the sold goods, respectively. The LINK function add to the final visualization the $Store_Sales$ value (given by $Total_Cost + Total_Profit$). Finally, the SHARE function is defined between the $FoodD24$ and $Food_Store_State$ terms, the function builds a CoDe complex term by sharing the *Food D24* component on the respective connected report. The generated report is shown in Fig. 8.3(b).

The first step of the *OLAP Operation Pattern Definition* phase generates the eligible patterns for all the terms and functions, then the second step selects the unique OLAP patterns by generating the set *Ss* of switchable strings (see Table 8.8). The last step aims to build the lattice structure shown in Fig.8.4. The nodes coloured in grey represent the views of the workload.

The first step of the *OLAP Operation Optimization* phase generates the MST from the lattice structure. Figure 8.4 shows the MST with the view space and the processing

Table 8.8: Switchable strings and OLAP unique operation patterns for the CoDe model in Fig. 8.3(a).

S_1-Food[**food**] =	a b c d e {f g h i j k l}	a b c d e f g h i j k l
S_2-TotalCost[**food**] =	a b m d e {f g h i j k l}	a b m d e f g h i j k l
S_3-TotalProfit[**food**] =	a b n d e {f g h i j k l}	a b n d e f g h i j k l
S_4-Aggr.**CostandProfit** =	a b m n d e {f g h i j k l}	a b m n d e f g h i j k l
S_5-FoodStoreState[**food**] =	o p c q e r {g s h i j l}	o p c q e r g s h i j l
S_6-Equal =	{a t} m n d	a t m n d
S_7-SalesStoreState[**food**]) =	o p c q e r {g s h i j l}	o p c q e r g s h i j l
S_8-Link =	p	p

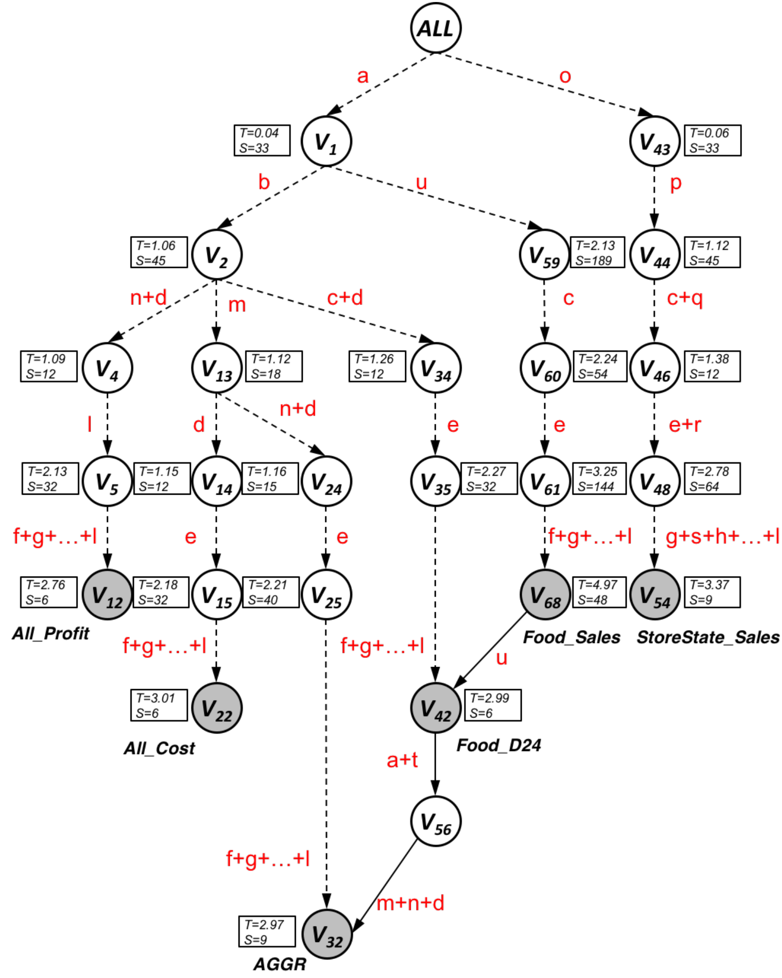


Figure 8.4: The lattice structure with the OLAP operations and the corresponding MST (dashed arrows) constructed on the CoDe model in Fig. 8.3(a).

cost computed for each node in according to the proposed cost model. Finally, the views to materialize by applying the solution proposed by [29, 62] are selected. The HRU_T procedure (with $k = 3$) selects the views: *root*, v_2, v_{59} and v_{48}. The HRU_S procedure with storage space $s = 114$, selects the views: *root*, v_{22}, v_{42}, v_{12}, v_{13}, v_{54}, v_{46}, v_2, v_{32}. The $AvQC$ procedure, by assuming that all aggregates have an equal probability of being queried, selects the views: *root*, v_{12}, v_{22}, v_{42}, v_{32}, v_{54}, v_4, v_{14}, v_{34}, v_{46}.

The HRU_T procedure (with $k = 3$) selects the views: *root*, v_2, v_{48} and v_{59}. The HRU_S procedure with storage space $s = 114$ selects the views: *root*, v_2, v_{12}, v_{13}, v_{22}, v_{32}, v_{42}, v_{46}, v_{54}. Finally, the algorithm proposed by Shukla *et al.* [62], by assuming that all aggregates

Table 8.9: Processing time and storage space of adopted algorithms applied on the lattice structure of Fig. 8.4.

	No Mat.	**AvQC**	**$\mathbf{HRU_T}$**	**$\mathbf{HRU_S}$**
Selected views	*root*	*root*, v_2, v_{12}, v_{34}, v_{43}	*root*, v_2, v_{48}, v_{59}	*root*, v_2, v_{12}, v_{13}, v_{22}, v_{32}, v_{42}, v_{46}, v_{54}
Size(M)	-	96MB	298MB	111MB
Time(V, M)	48.70s	21.11s	25.75s	17.16s
Time(Q, M)	24.07s	17.03s	16.89s	15.50s
Profit	-	29%	30%	36%

have an equal probability of being queried, selects the views: *root*, v_2, v_{12}, v_{34}, and v_{43}.

Table 8.9 summarizes the obtained results of the three algorithms. In particular, the algorithm HRU_S reduces the total processing time to answer the workload queries and the used storage space (111MB wrt. HRU_T that uses 298MB). In this case HRU_T has a worse performance in term of processing time when all the views have to be calculated. Moreover, *AvQC* has better results in term of storage space but worst processing time wrt. HRU_S. It is worth noting that the obtained results are worse considering the ones obtained on the previous CoDe model (i.e., 36% vs. 62%), due the presence of a higher number of leaf nodes onto the MST corresponding to the workload queries wrt the MST depicted in Fig. 8.2. This aspect highlights how the results are affected by the lattice structure.

8.2.1 Addition of a component to the model

The component added to the CoDe model is $FoodD25$ which is at the same hierarchical level as $FoodD24$. It represents the data series for the product family food in the February 25, 1997. The generated model is shown in Fig.8.5, where the SUM function is applied to the item $FOOD[D24, D25]$ and its results are reported in the item $Total_Sales$.

After the CoDe model generation, the CoDe Dynamic process provides the *OLAP Operation pattern definition* phase (as specified in Chapter 6). The first step generates the OLAP eligible patterns for the CoDe terms and functions, while the second step selects the unique OLAP patterns, decomposing the OLAP operation patterns of each dimensional members by renaming them with a unique labels and generates the set of *switchable strings* (showed in Table 8.10). Successively, the optimized lattice creation step calls the

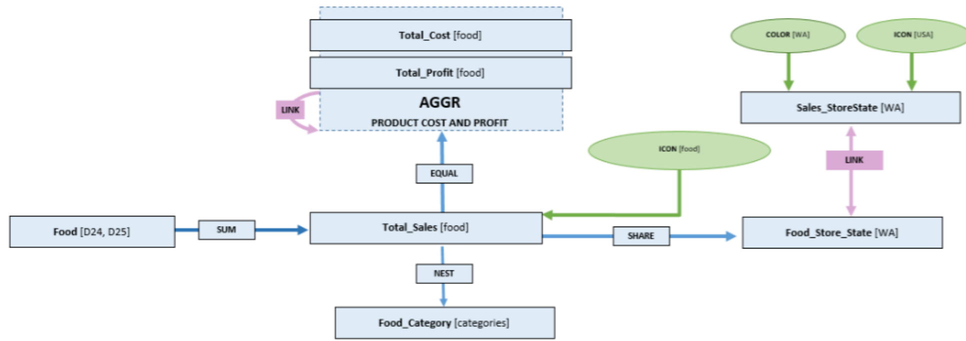


Figure 8.5: The CoDe model with the new component $FoodD25$.

Table 8.10: Switchable strings and OLAP unique operation patterns for the CoDe model in Fig. 8.5.

S_1-**Food**[**food**] =	a b c d e {f g h i j k l}	a b c d e f g h i j k l
S_2-**TotalCost**[**food**] =	a b m d e {f g h i j k l}	a b m d e f g h i j k l
S_3-**TotalProfit**[**food**] =	a b n d e {f g h i j k l}	a b n d e f g h i j k l
S_4-**Aggr**.**CostandProfit** =	a b m n d e {f g h i j k l}	a b m n d e f g h i j k l
S_5-**FoodStoreState**[**food**] =	o p c q e r {g s h i j l}	o p c q e r g s h i j l
S_6-**Equal** =	{a t} m n d	a t m n d
S_7-**SalesStoreState**[**food**]) =	o p c q e r {g s h i j l}	o p c q e r g s h i j l
S_8-**Link** =	p	p
S_9-**FoodCategory**[**food**] =	a u c e {f g h i j k l}	a u c e f g h i j k l
S_{10}-**Nest**[**food**]) =	u	u
S_{11}-**Food**[**24** – **25**]) =	v w c x y r {f g s z}	v w c x y r f g s z
S_{12}-**TotalSales**[**24** – **25**] =	a b c d e {f g h i j k}	a b c d e f g h i j k
S_{13}-**Sum**[**24** – **25**] =	{α β} {t a} c d	α t a c d
S_{14}-**FoodCategory**[**24** – **25**] =	a u c e {f g h i j k}	a u c e f g h i j k
S_{15}-**Nest**[**24** – **25**] =	u	u
S_{16}-**FoodStoreState**[**24** – **25**] =	o p c q e r {g s h i j k z}	o p c q e r g s h i j k z
S_{17}-**TotalCost**[**24** – **25**] =	a b m d e {f g h i j k}	a b m d e f g h i j k
S_{18}-**TotalProfit**[**24** – **25**] =	a b n d e {f g h i j k}	a b n d e f g h i j k
S_{19}-**Agg**.**Cost** & **Profit**[**24** – **25**] =	a b m n d e {f g h i j k}	a b m n d e f g h i j k
S_{20}-**Equal**[**24** – **25**] =	{a t} m n d	a t m n d
S_{21}-**SalesStoreState**[**24** – **25**] =	o p c q e r {g s h i j k z}	o p c q e r g s h i j k z
S_{22}-**Link**[**24** – **25**] =	p	p

SEQ_MATCH algorithm 6.1.2. Such algorithm builds Table 8.11 where rows correspond to the new paths and columns represent the paths belonging to the initial CoDe model. Each cell is computed by matching the strings, the value 0 indicates that the strings have

Table 8.11: The SEQ_MATCH execution.

	$\mathbf{S_1}$	$\mathbf{S_2}$	$\mathbf{S_3}$	$\mathbf{S_4}$	$\mathbf{S_5}$	$\mathbf{S_6}$	$\mathbf{S_7}$	$\mathbf{S_8}$	$\mathbf{S_9}$	$\mathbf{S_{10}}$
$\mathbf{S_{11}}$	0	0	0	0	0	0	0	0	0	0
$\mathbf{S_{12}}$	11	2	2	2	0	1	0	0	1	0
$\mathbf{S_{13}}$	0	0	0	0	0	0	0	0	0	0
$\mathbf{S_{14}}$	1	1	1	1	0	1	0	0	10	0
$\mathbf{S_{15}}$	0	0	0	0	0	0	0	0	0	1
$\mathbf{S_{16}}$	0	0	0	0	0	11	0	11	11	0
$\mathbf{S_{17}}$	2	11	2	3	0	1	0	0	1	0
$\mathbf{S_{18}}$	2	2	11	2	0	1	0	0	1	0
$\mathbf{S_{19}}$	2	3	2	12	0	1	0	0	1	0
$\mathbf{S_{20}}$	1	1	1	1	0	5	0	0	1	0
$\mathbf{S_{21}}$	0	0	0	0	11	0	11	11	0	0
$\mathbf{S_{22}}$	0	0	0	0	11	0	11	11	0	0

Table 8.12: The SEQ_MATCH output.

paht	$\mathbf{S_{11}}$	$\mathbf{S_{12}}$	$\mathbf{S_{13}}$	$\mathbf{S_{14}}$	$\mathbf{S_{15}}$	$\mathbf{S_{16}}$	$\mathbf{S_{17}}$	$\mathbf{S_{18}}$	$\mathbf{S_{19}}$	$\mathbf{S_{20}}$	$\mathbf{S_{21}}$	$\mathbf{S_{22}}$
string match	different	similar to S_1	different	similar to S_9	equal S_{10}	similar to S_5	similar to S_2	similar to S_3	similar to S_4	equal S_6	similar to S_5	similar to S_5

no labels in common, while a value greater than one indicates that there is at least one common value. Then, for each row we search the grater value and we associate one of the three different states (i.e., different, equal to, similar to) with the correspondent column, as show in Table 8.12. For example, to the S_{11} and S_{13} strings is assigned the label *Different* because all the values of the two rows are zero. Then, to the S_{20} string is assigned the label *Equal to* S_6 because its path match with the string S_6. Finally,y to the remaining paths are assigned the label *Similar to*. Once we obtained a mapping between the new and the old paths the optimized lattice creation step generates the lattice structure, shown in Fig. 8.6. The *OLAP Operation Optimization* phase generates the MST from the lattice structure. Figure 8.6 shows the MST (dashed arrows) with the view space and the processing cost computed for each node in according to the proposed cost model, the nodes coloured in grey represent the workload query.

Finally, the views to materialize are selected, by applying the solution proposed by [29, 62]. The HRU_T procedure (with $k = 3$) selects the views: *root*, v_2, v_{59} and v_{48}. The

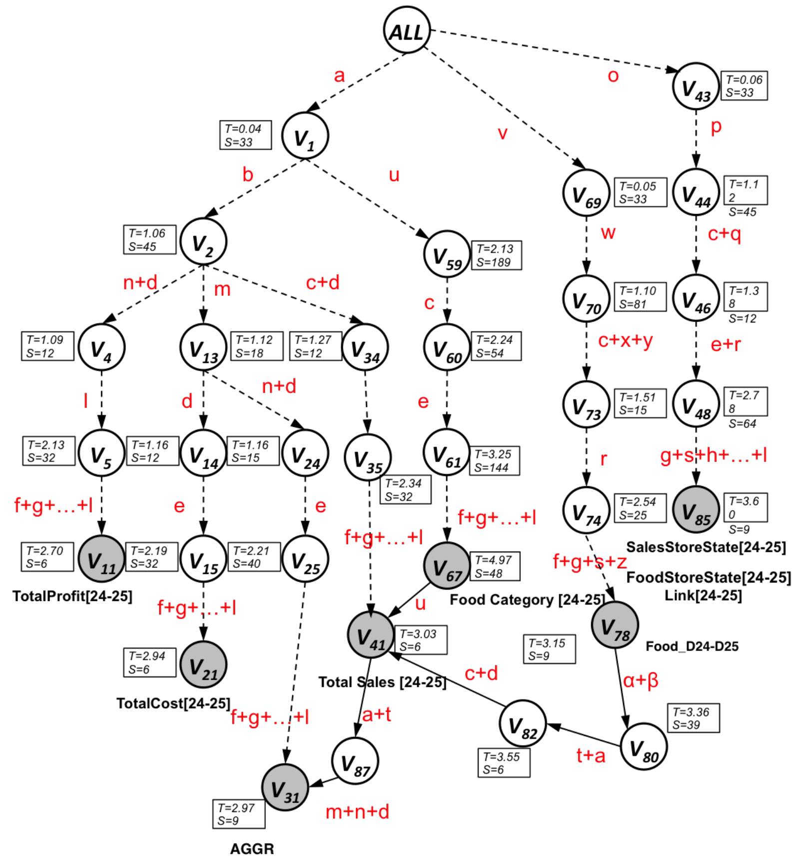


Figure 8.6: The lattice structure with the OLAP operations and the corresponding MST (dashed arrows) constructed on the CoDe model in Fig.8.5.

HRU_S procedure with storage space $s = 111$ (given multiplying 3 by the average size of views in the MST), selects the views: $root$, v_{41}, v_{21}, v_{11}, v_{13}, v_{85}, v_{78}, v_{46}, v_2. Finally, the algorithm proposed by Shukla *et al.* [62], by assuming that all aggregates have an equal probability of being queried, selects the views: $root$, v_{11}, v_{21}, v_{41}, v_{31}, v_{78}, v_{85}, v_4, v_{14}, v_{34}, v_{46}, v_{24}, v_{73}.

Table 8.13 summarizes the results of the three algorithms. In particular, the greedy algorithm HRU_S ($S = 111$) reduces its processing time with respect HRU_T (20.06s wrt. 20.18s). The $AvQC$ has better performance with respect the two algorithms in term of processing time (19.77s) and profit (15% wrt. 14% of others two algorithms).

Table 8.13: Algorithms evaluation by using the CoDe model in Fig. 8.5

	No Mat.	AvQC	HRU_T	HRU_S
Selected views	-	root, v_{11}, v_{21}, v_{41}, v_{78}, v_{85}, v_4, v_{14}, v_{34}, v_{46}, v_{24}	root, v_2, v_{59}, v_{48}	root, v_{41}, v_{21}, v_{11}, v_{13}, v_{85}, v_{78}, v_{46}, v_2
Size(M)	-	108	298	111
Time(V, M)	57.29	26.78s	33.52s	26.03s
Time(Q,M)	23.36s	19.77s	20.18s	20.06s
Profit	-	15%	14%	14%

Table 8.14: The VN_1 vector

V1	V2	V4	V5	V11
0	1/8	0	0	1/8
V13	V14	V15	V21	V24
1/8	0	0	1/8	0
V25	V31	V34	V35	V41
0	0	0	0	1/8
V59	V60	V61	V67	V59
0	0	0	0	0
V87	V31	V69	V70	V73
0	0	0	0	0
V74	V78	V80	V82	V41
0	1/8	0	0	0
V43	V44	V46	V48	V85
0	0	1/8	0	1/8

8.3 Probabilistic approach

In this case study, we show the probabilistic approach on the two CoDe models of Fig. 6.3 and Fig.8.5.

The Markov algorithm builds the $Q' * V'$ matrix (shown in Chapter 10, Table10.1), where the rows represent the view of the MST (see Fig. 8.4) and columns represent the unique OLAP patterns (see Table. 8.8) of the CoDe model of Fig.6.3. Then it builds the $Q * V$ matrix (shown in Chapter 10, Table10.2), where the rows represent the view of the MST (see Fig. 8.6) and columns represent the unique OLAP patterns (see Table. 8.10) of the CoDe model of Fig. 8.5 on which the item $FoodD25$ has been added. Such matrices are obtained by dividing the number of labels (calculated on the views and OLAP patterns that match) by the value of the OLAP pattern with the maximum length.

Table 8.15: The VN_i vector

V1	V2	V4	V5	V11
0.0071	0.0040	0.022	0.009	0.033
V13	V14	V15	V21	V24
0.006	0.001	0.0014	0.036	0.0064
V25	V31	V34	V35	V41
0.006	0.04	0.0009	0.0002	0.034
V59	V60	V61	V67	V59
0.0029	0.01	0.0069	0.005	0.002
V87	V31	V69	V70	V73
0.007	0.0002	0.0083	0.0002	0.0052
V74	V78	V80	V82	V41
0.0014	0.0033	0.0073	0.002	0.0082
V43	V44	V46	V48	V85
0.0002	0.005	0.027	0.007	0.009

Table 8.16: The VN_{i+1} vector

V1	V2	V4	V5	V11
0.0069	0.0039	0.019	0.0088	0.032
V13	V14	V15	V21	V24
0.0059	0.0009	0.0011	0.034	0.0062
V25	V31	V34	V35	V41
0.006	0.039	0.0004	0.0001	0.033
V59	V60	V61	V67	V59
0.0029	0.009	0.0069	0.005	0.002
V87	V31	V69	V70	V73
0.0069	0.0001	0.0082	0.0001	0.0053
V74	V78	V80	V82	V41
0.0012	0.0032	0.0072	0.0019	0.0081
V43	V44	V46	V48	V85
0.0001	0.0049	0.026	0.0069	0.008

Successively, the Markov Algorithm 8 builds the Initial Probability Matrix $V * V$ that is given multiplying $Q' * V'$ and the transposed sub matrices of $Q * V$. Then, it creates the Impact Probability Vectors VN_1 shown in Fig. 8.14, that represents the set of view already materialized for the CoDe model in Fig. 6.3 and multiplies it by the $V * V$ matrix. The obtained vector VN_2 is multiplied by the $V * V$ matrix until it converges. The vector will converge when the least mean square between VN_{i+1} and VN_i will be minor with respect

Table 8.17: Markov Algorithm evaluation

	No-Mat	Markov
Views	-	$root$, v_{11}, v_{21}, v_{31}, v_{41}, v_{46}
Size	-	35
Time(Q,M)	20.07s	12.94s
Profit	-	36%

Table 8.18: Algorithms evaluation corresponding to the CoDe model in Fig.8.5.

	No-Mat	AvQC	$\mathbf{HRU_S}$	$\mathbf{HRU_T}$	Markov
Views	-	$root$,v_12, v_{22},v_{42}, v_{32},v_{54},v_4,v_{14}, v_{34},v_{46}	$root$,v_{22},v_{42},v_{12}, v_{13},v_{54},v_{46},v_2, v_{32}	$root$, v_2, v_{59},v_{48}	$root$, v_{11}, v_{21}, v_{31}, v_{41}, v_{46}
Size	-	84	111	289	35
Time(Q,M)	20.07s	14.93s	16s	16.89s	12.94s
Profit	-	26%	21%	16%	36%

the threshold $\varepsilon = 0.05$. The two vectors are showed in Fig.8.15 and Fig.8.16. Finally, the Markov algorithm selects the new set of views to materialize: $V_{11}, V_{21}, V_{31}, V_{41}$ and V_{46}. Table 8.17 highlights the processing time and the storage space values whit respect to the approach without materialization.

In Table 8.18 we summarize the results obtained from the four algorithms (i.e., HRU_S, HRU_T, $AvQC$, $Markov$) by applying the dynamic CoDe process on the CoDe model of Fig.8.5. As we can see the $AvQC$ and the $Markov$ procedure have better performance in term of total processing time to answer the workload queries ($14.93s$ and $12.94s$). Moreover, the Markov algorithm selects a set of views that occupies less space than the other solutions, it takes up the 42% less than the $AvQC$ procedure. Thus, the Markov algorithm gives a better set of solutions.

Chapter 9

Conclusions

A data warehouse is a read-only analytical database that is used as the foundation of a decision support system. In particular, users can analyse situations and make decisions through the execution of complex queries. Since the computation of these queries is time consuming, data warehouses pre compute a set of materialized views answering to the workload queries. In order to define the right set of workload queries and the minimal set of precomputed views, an analysis phase on the data warehouse is needed. However, this approach requires a time consuming set-up phase that increases the overall costs.

CoDe is a visual language that represents high level information exploiting a CoDe model. The company manager expert of a specific domain, through such model, designs what information have to be visualized.

9.1 Thesis Summary

In this thesis we propose three different approaches that exploit the CoDe modeling language to find the set of workload queries that answers the user requests and mitigates the problem to find a minimal set of views to materialize. The proposed approaches are summarized below.

A static approach to the VSP problem. We have proposed a set of heuristics and algorithms that allows the company manager to design the CoDe model, to choose the workload query, and to find the right set of views to materialize [31, 67]. In particular, we presented:

- an algorithm to generate the eligible patterns for all the CoDe terms and functions;

- an algorithm to select the minimal number of required OLAP queries;
- a heuristic to create a lattice structure that allows obtaining all the possible paths that answer the workload queries;
- an algorithm to map the lattice structure into a MST in order to avoid the explosion of the number of nodes;
- three greedy algorithms to select the set of views to materialize (i.e., HRU_T, HRU_S, $AvQC$).

A dynamic approach to the VSP problem. Since the information in DW changes overtime, we take into account the problem of the CoDe model evolution. In particular, we proposed:

- a context-aware editor, based on a context sensitive grammar, that supports the manager in the specification of the model by suggesting the items to add, remove, or replace;
- a validation function that checks the syntax of the CoDe functions;
- a min-max strategy that sorts all the possible actions the manager can perform to update the CoDe model;
- an algorithm that exploits historical information to compute or update the lattice structure.

A probabilistic approach to the VSP problem. In order to reduce the overhead for generating a new minimal set of views, we have adapted a Markov strategy into our CoDe dynamic process, that exploits a small set of historical information concerning the costs to materialize the views. In particular, the proposed algorithm identifies the views to be materialized by evaluating the impact frequency of the OLAP queries on them.

The proposed approaches have been evaluated on a real DW. In particular, the static approach showed an improvement on the processing time in the range of 36-62% for the algorithm HRU_S with respect to the solution which does not perform any materialization, and 7% with respect to an approach that exploits the materialized views maximizing the benefit per unit space based on their probability to be queried (i.e., $AvQC$). In the case of

two consecutive executions, the algorithm HRU_S reaches an improvement of at least 98% after the second execution. This value is quite constant in the successive iterations. By considering a whole CoDe model, the results confirm the ones obtained on a sub-part of the model. In particular, the algorithms HRU_T and HRU_S reach an improvement of 51% and 60%, respectively. When a new component is added, the $AvQC$ outperforms the other two algorithms in term of processing time ($19.77s$). Finally, by applying the probabilistic approach, the $AvQC$ and the $Markov$ procedure achieve better performance with respect to the other algorithms, in term of total processing time to answer the workload queries ($14.93s$ and $12.94s$). Moreover, the Markov algorithm selects a set of views that occupies less space than the other solutions taking up the 42% less than the $AvQC$ procedure. Thus, the Markov algorithm returns a better set of solutions.

9.2 Perspectives

To consolidate the results presented in this paper we plan to test the proposed process on larger DWs. Moreover, in the future, we plan to take into account other techniques to sorts all the possible options in the context-aware editor with respect to the min-max strategy and focus our study towards other probabilistic techniques to mitigate the problem to find a minimal set of views to materialize taking into account the previously selected views. In addition, we shall consider adding new functionalities based on data mining techniques, which allow investigating the CoDe model and help the company manager to easily perform statistical analysis and to find patterns on selected data.

Chapter 10

Appendix

Table 10.1: The $Q' * V'$ Matrix

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
V1	1/13	1/13	1/13	1/13	0	1/13	0	0	1/13	0
V2	1/13	1/13	1/13	1/13	0	0	0	0	0	0
V4	1/13	1/13	2/13	2/13	0	2/13	0	0	0	0
V5	1/13	1/13	1/13	1/13	1/13	0	1/13	0	1/13	0
V12	11/13	11/13	12/13	12/13	6/13	3/13	6/13	0	9/13	0
V13	0	1/13	0	1/13	0	1/13	0	0	0	0
V14	1/13	1/13	1/13	1/13	0	1/13	0	0	0	0
V15	1/13	1/13	1/13	1/13	1/13	0	1/13	0	1/13	0
V22	11/13	11/13	12/13	12/13	6/13	3/13	6/13	0	9/13	0
V24	1/13	1/13	2/13	2/13	0	2/13	0	0	0	0
V25	1/13	1/13	1/13	1/13	1/13	0	1/13	0	1/13	0
V32	11/13	12/13	12/13	13/13	6/13	4/13	6/13	0	9/13	0
V34	2/13	1/13	1/13	1/13	1/13	1/13	1/13	0	1/13	0
V35	1/13	1/13	1/13	1/13	1/13	0	1/13	0	1/13	0
V42	12/13	11/13	11/13	11/13	7/13	2/13	7/13	0	10/13	0
V56	1/13	1/13	1/13	1/13	0	2/13	0	0	1/13	0
V43	0	0	0	0	1/13	0	1/13	0	0	0
V44	0	0	0	0	1/13	0	1/13	1/13	0	0
V46	1/13	0	0	0	2/13	0	2/13	0	1/13	0
V48	1/13	1/13	1/13	1/13	2/13	0	2/13	0	1/13	0
V54	7/13	6/13	6/13	6/13	12/13	0	12/13	1/13	7/13	0
V59	0	0	0	0	0	0	0	0	1/13	1/13
V60	1/13	0	0	0	1/13	0	1/13	0	1/13	0
V61	1/13	1/13	1/13	1/13	1/13	0	1/13	0	1/13	0
V68	10/13	9/13	10/13	9/13	7/13	1/13	7/13	0	11/13	1/13

To ease the readability we show the transposed matrix.

Table 10.2: The $Q * V$ Matrix

V1	V2	V4	V5	V11	V13	V14	V15	V21	V24	V25	V31	V34	V35	V41	V59	V60	V61	V67	V59	V87	V31	V69	V70	V73	V74	V78	V80	V82	V41	V43	V44	V46	V48	V85	
S1	1/13	1/13	0	1/13	6/13	0	0	1/13	6/13	0	1/13	6/13	2/13	1/13	6/13	0	1/13	1/13	6/13	0	1/13	1/13	0	0	1/13	0	2/13	0	1/13	2/13	0	0	1/13	1/13	4/13
S2	1/13	1/13	0	1/13	6/13	1/13	0	1/13	6/13	0	1/13	6/13	1/13	1/13	6/13	0	0	1/13	6/13	0	1/13	2/13	0	0	0	0	2/13	0	1/13	1/13	0	0	0	1/13	4/13
S3	1/13	1/13	2/13	1/13	6/13	0	0	1/13	6/13	2/13	1/13	6/13	1/13	1/13	6/13	0	0	1/13	6/13	0	1/13	2/13	0	0	0	0	2/13	0	1/13	1/13	0	0	0	1/13	4/13
S4	1/13	1/13	2/13	1/13	6/13	1/13	0	1/13	6/13	2/13	1/13	6/13	1/13	1/13	6/13	0	0	1/13	6/13	0	1/13	3/1/133	0	0	0	0	2/13	0	1/13	1/13	0	0	0	1/13	4/13
S5	0	0	0	1/13	3/1/133	0	0	1/13	3/1/133	0	1/13	3/1/133	1/13	1/13	3/1/133	0	1/13	1/13	3/1/133	0	0	0	0	0	1/13	0	2/13	0	0	1/13	1/13	1/13	2/13	2/13	4/13
S6	1/13	1/13	2/13	0	0	1/13	0	0	0	2/13	0	0	1/13	0	0	0	0	0	0	0	1/13	3/1/133	0	0	0	0	0	0	1/13	1/13	0	0	0	0	0
S7	0	0	0	1/13	3/1/133	0	0	1/13	3/1/133	0	1/13	3/1/133	1/13	1/13	3/1/133	0	1/13	1/13	3/1/133	0	0	0	0	0	1/13	1/13	2/13		0	1/13	1/13	1/13	2/13	2/13	4/13
S8	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1/13	0	0	0
S9	1/13	1/13	0	1/13	6/13		0	1/13	6/13	0	1/13	6/13	1/13	1/13	6/13	0	1/13	1/13	6/13	0	1/13	0	1/13		1/13	0	2/13		1/13	1/13	0	0	1/13	1/13	5
S10	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	1/13	0	0	0	0	0	0	0	0	0	0	0	0
S11	0	0	0	0	2/13	0	0	0	2/13	0	0	2/13	1/13	0	2/13	0	1/13	0	2/13	0	0	0	0	1/13	3/1/133	1/13	3/1/133	1/13	0	1/13	0	0	1/13	2/13	2/13
S12	1/13	1/13	1/13	1/13	6/13	0	1/13	1/13	6/13	1/13	1/13	6/13	2/13	1/13	6/13	0	1/13	1/13	6/13	0	1/13	1/13	0	0	1/13	0	2/13		1/13	2/13	0	0	1/13	1/13	4/13
S13	1/13		1/13	0	0	0	1/13	0	0	1/13	0	0	2/13	0	0	1/13	1/13	0	0	1/13	2/13	1/13	0	0	1/13	0	0	1/13	2/13	2/13	0	0	1/13	1/13	0
S14	1/13	1/13	0	1/13	6/13	0	0	1/13	6/13		1/13	6/13	1/13	1/13	6/13	0	1/13	1/13	6/13	0	1/13	0	1/13	0	1/13	0	2/13		1/13	1/13	0	0	1/13	1/13	5
S15	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	1/13	0	0	0	0	0	0	0	0	0	0	0	0
S16	0	0	0	1/13	3/1/133	0	0	1/13	3/1/133	0	1/13	3/1/133	1/13	1/13	3/1/133	0	1/13	1/13	3/1/133	0	0	0	0	0	1/13	1/13	2/13	1/13	0	1/13	1/13	1/13	2/13	2/13	5
S17	1/13	1/13	1/13	1/13	6/13	1/13	1/13	1/13	6/13	1/13	1/13	6/13	1/13	1/13	6/13	0	0	1/13	6/13	0	1/13	2/13	0	0	0	0	2/13	0	1/13	1/13	0	0	0	1/13	5
S18	1/13	1/13	1/13	1/13	6/13	0	1/13	1/13	6/13	1/13	1/13	6/13	1/13	1/13	6/13	0	0	1/13	6/13	0	1/13	2/13	0	0	0	0	2/13	0	1/13	1/13	0	0	0	1/13	5
S19	1/13	1/13	2/13	1/13	6/13	1/13	0	1/13	6/13	2/13	1/13	6/13	1/13	1/13	6/13	0	0	1/13	6/13	0	1/13	3/1/133	0	0	0	0	2/13	0	1/13	1/13	0	0	0	1/13	5
S20	1/13	1/13	2/13	0	0	1/13	0	0	0	2/13	0	0	1/13	0	0	0	0	0	0	0	2/13	3/1/133	0	0	0	0	0	0	2/13	1/13	0	0	0	0	0
S21	0	0	0	1/13	3/1/133	0	0	1/13	3/1/133	0	1/13	3/1/133	1/13	1/13	3/1/133	0	1/13	1/13	3/1/133	0	0	0	0	0	1/13	1/13	2/13	1/13	0	1/13	1/13	1/13	2/13	2/13	6/13
S22	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1/13	0	0	0

References

[1] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB)*, pages 496–505. Morgan Kaufmann Publishers Inc., 2000.

[2] Elena Baralis, Stefano Paraboschi, and Ernest Teniente. Materialized views selection in a multidimensional database. In *VLDB*, volume 97, pages 156–165, 1997.

[3] Xavier Baril and Zohra Bellahsene. Selection of materialized views: A cost-based approach. In *Proceedings of the 15th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 665–680. Springer, 2003.

[4] B Bebel, J Eder, C Koncilia, et al. Creation and nlanagenlent of ve sions in muhiversion data warehouses. In *ACM SAC*, volume 2004, 2004.

[5] Randall G. Bello, Karl Dias, Alan Downing, James J. Feenan, Jr., James L. Finnerty, William D. Norcott, Harry Sun, Andrew Witkowski, and Mohamed Ziauddin. Materialized views in oracle. In *Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB)*, pages 659–664. Morgan Kaufmann Publishers Inc., 1998.

[6] Edgard Benitez-Guerrero, Christine Collet, and Michel Adiba. The WHES approach to data warehouse evolution. *E-Gnosis*, 2, 2004.

[7] Jacques Bertin. Semiology of graphics: diagrams, networks, maps. 1983.

[8] Markus Blaschka, Carsten Sapia, and Gabriele Hofling. On schema evolution in multidimensional databases. In *Proceedings of the 1st International Conference on Data Warehousing and Knowledge Discovery (DAWAK)*, pages 153–164. Springer, 1999.

[9] Bruno Blaskovic, Petar Kneievic, and Mirko Randic. Model checking approach for communication procedures validation. In *Proceedings of International Conference on Trends in Communications (EUROCON)*, volume 2, pages 532–535. IEEE Press, 2001.

[10] Mathurin Body, Maryvonne Miquel, Yvan Bédard, and Anne Tchounikine. A multi-dimensional and multiversion structure for OLAP applications. In *Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP (DOLAP)*, pages 1–6. ACM Press, 2002.

[11] G.K.Y. Chan, Q. Li, and L. Feng. Optimized design of materialized views in a real-life data warehousing environment. *International Journal of Information Technology*, 7(1):30–54, 2001.

[12] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Rec.*, 26(1):65–74, March 1997.

[13] Rada Chirkova, Alon Y Halevy, and Dan Suciu. A formal perspective on the view selection problem. *The International Journal on Very Large Data Bases (VLDBJ)*, 11(3):216–237, 2002.

[14] Maria I Ciuccarelli, Paolo Sessa and Maurizio Tucci. CoDe: A graphic language for complex system visualization. In *Proceedings of Italian Association for Information Systems (ItAIS)*, 2010.

[15] Shaul Dar, Michael J Franklin, Bjorn T Jonsson, Divesh Srivastava, Michael Tan, et al. Semantic data caching and replacement. In *Proceedings of the 22th International Conference on Very Large Data Bases*, pages 330–341. Morgan Kaufmann Publishers Inc., 1996.

[16] Prasad M Deshpande and Jeffrey F Naughton. Aggregate aware caching for multi-dimensional queries. In *Proceedings of the 7th International Conference on Extending Database Technology (EDBT)*, pages 167–182. Springer, 2000.

[17] Prasad M Deshpande, Karthikeyan Ramasamy, Amit Shukla, and Jeffrey F. Naughton. Caching multidimensional queries using chunks. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 259–270. ACM Press, 1998.

[18] Chandrashekhar A Dhote and MS ALi. Materialized view selection in data warehousing. In *Proceedings of the 4th International Conference on Information Technology (ITNG)*, pages 843–847. IEEE Press, 2007.

[19] Partha Ghosh and Soumya Sen. Dynamic incremental maintenance of materialized view based on attribute affinity. In *Proceedings of International Conference on Data Science & Engineering (ICDSE)*, pages 12–17. IEEE Press, 2014.

[20] Partha Ghosh and Soumya Sen. Materialized view replacement using markov's analysis. In *Proceedings of IEEE International Conference on Industrial Technology (ICIT)*, pages 771–775. IEEE Press, 2014.

[21] Jonathan Goldstein and Perke Larson. Optimizing queries using materialized views: a practical, scalable solution. In *ACM SIGMOD Record*, volume 30, pages 331–342. ACM Press, 2001.

[22] Rajib Goswami, Dhruba Kr Bhattacharyya, Malayananda Dutta, and Jugal K Kalita. Approaches and issues in view selection for materialising in data warehouse. *International Journal of Business Information Systems*, 21(1):17–47, 2016.

[23] Himanshu Gupta. Selection of views to materialize in a data warehouse. In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 98–112. Springer, 1997.

[24] Himanshu Gupta and Inderpal Singh Mumick. Selection of views to materialize under a maintenance cost constraint. In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 453–470. Springer, 1999.

[25] Himanshu Gupta and Inderpal Singh Mumick. Selection of views to materialize in a data warehouse. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 17(1):24–43, 2005.

[26] Dirk Habich, Wolfgang Lehner, and Michael Just. Materialized views in the presence of reporting functions. In *Proceedings of the 18th International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 159–168. IEEE CS Press, 2006.

[27] Alon Y Halevy. Answering queries using views: A survey. *The International Journal on Very Large Data Bases (VLDBJ)*, 10(4):270–294, 2001.

[28] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[29] Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Implementing data cubes efficiently. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 205–216, 1996.

[30] Carlos A Hurtado, Alberto O Mendelzon, and Alejandro A Vaisman. Maintaining data cubes under dimension updates. In *Proceedings of 15th International Conference on Data Engineering (ICDE)*, pages 346–355. IEEE Press, 1999.

[31] Valentina Indelli Pisano, Michele Risi, and Genoveffa Tortora. Exploiting CoDe modeling for the optimization of OLAP queries. In *Proceedings of the 11th International Conference on Digital Information Management (ICDIM)*. IEEE CS Press, 2016.

[32] William H Inmon. *Building the data warehouse*. John wiley & sons, 2005.

[33] Seema Jaggi. Descriptive statistics and exploratory data analysis. *Indian Agricultural Statistics Research Institute*, pages 1–18, 2003.

[34] Howard Karloff and Milena Mihail. On the complexity of the view-selection problem. In *Proceedings of the 18th Symposium on Principles of Database Systems (PODS)*, pages 167–173, 1999.

[35] Gopalan Kesavaraj and Sreekumar Sukumaran. A study on classification techniques in data mining. In *Proceedings of 4th International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, pages 1–7. IEEE Press, 2013.

[36] Ralph Kimball and Margy Ross. *The data warehouse toolkit: The complete guide to dimensional modeling*. John Wiley & Sons, 2011.

[37] Donald Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys (CSUR)*, 32(4):422–469, 2000.

[38] Yannis Kotidis and Nick Roussopoulos. Dynamat: a dynamic view management system for data warehouses. In *ACM SIGMOD Record*, volume 28, pages 371–382. ACM Press, 1999.

[39] Wolfgang Lehner, Wolfgang Hummer, and Lutz Schlesinger. Processing reporting function views in a data warehouse environment. In *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, pages 176–185, 2002.

[40] Jia Liu, Yongwei Wu, and Guangwen Yang. Optimization of data retrievals in processing data integration queries. In *Proceedings of the International Conference on Frontier of Computer Science and Technology (FCST)*. IEEE CS Press, 2009.

[41] J. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1137–1144, 2007.

[42] Imene Mami and Zohra Bellahsene. A Survey of View Selection Methods. *SIGMOD Record*, 41(1):20–29, 2012.

[43] Meteorite Saiku. *Open Source Analysis Suite*. http://analytical-labs.com.

[44] Hoshi Mistry, Prasan Roy, S Sudarshan, and Krithi Ramamritham. Materialized view selection and maintenance using multi-query optimization. In *ACM SIGMOD Record*, volume 30, pages 307–318. ACM Press, 2001.

[45] Mondrian Pentaho. *Foodmart*. http://mondrian.pentaho.com.

[46] Tadeusz Morzy and Robert Wrembel. On querying versions of multiversion data warehouse. In *Proceedings of the 7th ACM International Workshop on Data Warehousing and OLAP (DOLAP)*, pages 92–101. ACM Press, 2004.

[47] Stephan Muller, Lars Butzmann, Kai Howelmeyer, Stefan Klauck, and Hasso Plattner. Efficient view maintenance for enterprise applications in columnar in-memory databases. In *Proceedings of the 17th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 249–258. IEEE Press, 2013.

[48] Surendra Nahar, Sartaj Sahni, and Eugene Shragowitz. Simulated annealing and combinatorial optimization. In *Proceedings of the 23rd ACM/IEEE design automation conference*, pages 293–299. IEEE Press, 1986.

[49] Anisoara Nica and Elke A Rundensteiner. Using containment information for view evolution in dynamic distributed environments. In *Database and Expert Systems Ap-*

plications, 1998. Proceedings. Ninth International Workshop on, pages 212–217. IEEE, 1998.

[50] Anisoara Nica and Elke A Rundensteiner. View maintenance after view synchronization. In *Proceedings of International Symposium on Database Engineering and Applications (IDEAS)*, pages 215–223. IEEE Press, 1999.

[51] Chang-Sup Park, Myoung-Ho Kim, and Yoon-Joon Lee. Rewriting OLAP queries using materialized views and dimension hierarchies in data warehouses. In *Proceedings of the 17th International Conference on Data Eng. (ICDE)*, pages 515–523, 2001.

[52] Jiratta Phuboon-ob and R Auepanwiriyakul. Analysis and comparison of algorithm for selecting materialized views in a data warehousing environment, 2006.

[53] Jiratta Phuboon-ob and Raweewan Auepanwiriyakul. Selecting materialized views using two-phase optimization with multiple view processing plan. *World Academy of Science, Engineering and Technology*, 27, 2007.

[54] Paulraj Ponniah. *Data warehousing fundamentals: A comprehensive guide for IT professionals.* John Wiley & Sons, 2004.

[55] Michele Risi, Maria I. Sessa, Genoveffa Tortora, and Maurizio Tucci. Visualizing information in data warehouses reports. In *Proceedings of the Symposium on Advanced Database Systems (SEBD)*, pages 246–257, 2011.

[56] Michele Risi, Maria Immacolata Sessa, Maurizio Tucci, and Genoveffa Tortora. CoDe modeling of graph composition for data warehouse report visualization. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):563–576, 2014.

[57] Prasan Roy, Srinivasan Seshadri, S Sudarshan, and Siddhesh Bhobe. Efficient and extensible algorithms for multi query optimization. In *ACM SIGMOD Record*, volume 29, pages 249–260. ACM Press, 2000.

[58] Stuart J Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2009.

[59] Shinichirou Saeki, Subhash Bhalla, and Masaki Hasegawa. Parallel generation of base relation snapshots for materialized view maintenance in data warehouse environment.

In *Proceedings of International Conference on Parallel Processing Workshops*, pages 383–390. IEEE Press, 2002.

[60] Peter Scheuermann, Junho Shim, and Radek Vingralek. Watchman: A data warehouse intelligent cache manager. In *Proceedings of the 22th International Conference on Very Large Data Bases*, pages 51–62. Morgan Kaufmann Publishers Inc., 1996.

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

[62] Amit Shukla, Prasad Deshpande, and Jeffrey F. Naughton. Materialized view selection for multidimensional datasets. In *Proceedings of the 24rd Intlernational Conference on Very Large Data Bases (VLDB)*, pages 488–499, 1998.

[63] DG Simpson. Introduction to rousseeuw (1984) least median of squares regression. In *Breakthroughs in Statistics*, pages 433–461. Springer, 1997.

[64] Divesh Srivastava, Shaul Dar, HV Jagadish, and Alon Y Levy. Answering queries with aggregation using views. In *VLDB*, volume 96, pages 318–329, 1996.

[65] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002.

[66] Kurt Thearling. An introduction to data mining. *Direct Marketing Magazine*, pages 28–31, 1999.

[67] Valentina Indelli Pisano, Michele Risi, Genoveffa Tortora. How reduce the view selection problem through the code modeling. *Journal on Advances in Theoretical and Applied Informatics (JADI)*, 2(2):19–30, 2016.

[68] Antti Valmari. The state explosion problem. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the Volumes Are Based on the Advanced Course on Petri Nets*, pages 429–528. Springer-Verlag, 1998.

[69] Sirirut Vanichayobon and Gruenwald Le. Indexing techniques for data warehouses queries. *The University of Oklahoma, School of Computer Science, Technical Report*, 1999.

[70] Jennifer Widom. Research problems in data warehousing. In *Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM)*, pages 25–30. ACM Press, 1995.

[71] Jian Yang, Kamalakar Karlapalem, and Qing Li. Algorithms for materialized view design in data warehousing environment. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB)*, pages 136–145. Morgan Kaufmann Publishers Inc., 1997.

[72] Jian Yang, Kamalakar Karlapalem, and Qing Li. A framework for designing materialized views in data warehousing environment. In *Proceedings of the 17th International Conference on Distributed Computing Systems*, pages 458–465. IEEE Press, 1997.

[73] Chuan Zhang, Xin Yao, and Jian Yang. An evolutionary approach to materialized views selection in a data warehouse environment. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 31(3):282–294, 2001.

[74] Yihong Zhao, Prasad M Deshpande, and Jeffrey F Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *ACM SIGMOD Record*, volume 26, pages 159–170. ACM Press, 1997.

[75] Lijuan Zhou, Qian Shi, and Haijun Geng. The minimum incremental maintenance of materialized views in data warehouse. In *Proceedings of the 2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR)*, volume 3, pages 220–223. IEEE Press, 2010.