Quality-Oriented Optimization of the Design of the Global Data Warehouse

Panos Vassiliadis, Spyros Ligoudistianos, Timos Sellis, Dimitri Theodoratos

National Technical University of Athens, Department of Electrical and Computer Engineering,
Computer Science Division, Iroon Polytechniou 9, 157 73, Athens, Greece
E-mail: {pvassil, spyros, timos, dth}@dbnet.ece.ntua.gr

Abstract. Decision analysts require from data warehouse systems to provide the highest level of data and service quality. The Data Staging Area (DSA) is the central layer of the data warehouse architecture that integrates data from the different enterprise sources. The DSA can be seen as a set of materialized views defined over relations that are stored in remote heterogeneous database systems. The DSA design problem is the problem of selecting an optimal set of views to materialize that answer a given set of queries. In this paper we extend previous approaches to the problem, by treating the problem formally, with respect to specific, objective quality factors. Moreover, by using already existing cost-based DSA design methods, we incorporate data quality factors in the cost formulae of these methods in order to be taken under consideration during the selection of the final data warehouse views.

1. Introduction

Data Warehouse (DW) systems are centralized repositories of reconciled data within an organization. To turn data into useful information for the users, high level quality of both data and services is required. Coherency, freshness, accuracy, accessibility, availability and performance are among the quality features required by data warehouse users. The central layer of the architecture, called the *Global Data Warehouse*, keeps a historical record of data that result from the transformation, integration, and aggregation of detailed data found in the data sources. Usually, a data store of volatile, low granularity data is used for the integration of data from the various sources: it is called *Data Staging Area* (DSA). The data staging area serves also as a buffer for data transformation and cleaning so that the data warehouse is populated with clean and homogeneous data.

The data staging area can be considered as a set of (possibly) distributed materialized views defined over the data provided by the data sources. These views are used to answer all the queries posed to the data staging area, thus avoiding going back to the original sources. When modifications occur to the sources, the materialized views have to be refreshed through insertions deletions and updates. For a given set of different source databases and a given set of queries that the data warehouse has to service, there is a number of alternative sets of materialized views that the administrator can choose to maintain. The problem of selecting a set of views to materialize in order to maximize the overall quality of the data staging area, is called *Quality-Oriented DSA Design Problem*.

In this paper we extend previous approaches to this problem [1,4,5,6,7,9,10,11,12,16] by: (a) providing a formal treatment with respect to specific, objective quality factors, and (b) proposing extended cost formulae for the evaluation of the benefit of a solution, so that quality factors are taken into consideration for the computation of the final views of the data warehouse. For the latter, different options are considered for the characterization of the quality of relations (global annotation of the relation with quality factors, annotation of subsets of rows and annotation of specific columns). The reader is encouraged to refer to the long version of this paper [15] for a broader discussion of these topics.

The structure of this paper is as follows. In section 2 we define the problem in terms of quality factor definition and measurement. In section 3 we examine the evaluation of a particular state of the problem and in section 4 we present the specific cost model for the evaluation of a view. In section 5 we conclude our results and present future work.

2. General Setting of the Problem

Each object in the data warehouse can be subject of quality measurement. Since quality management plays an important role in data warehouses, it can be part of the metadata schema of the data warehouse and quality information is explicitly linked with architectural objects. The Esprit LTR Project *DWQ* [2] first proposed a quality metamodel (initially presented in [3]) which is based on the Goal-Question-Metric (GQM) approach. This approach was originally developed for software quality management. Fig. 1 shows a simplified conceptual view of the DWQ Quality Meta-Model, where each architecture object is linked to a set of quality goals and a set of quality factors.

A *quality goal* is a high level, user-oriented, abstract requirement, defined on any kind of data warehouse metadata object. It is used to express natural-language goals like 'improve the data loading process with respect to its timeliness for source s1'. A *quality factor* represents an actual measurement of a quality value, i.e., it relates quality values to measurable objects. Quality factors like 'average time for the loading of s1', 'downtime for source s2' are used for the objective evaluation and improvement of the state of the examined objects. The measurements for the quality factors are produced from customized *metric agents*. *Quality dimensions*, like 'timeliness' or ‘availability’ are used both as a vocabulary to define quality goals and as conceptual groups of quality factors (enriching their semantics with high level information). The bridge between the abstract, subjective quality goals and the specific, objective quality factors is determined through a set of *quality questions*, provided by the *DWQ-methodology*, described in [14].

The Data Staging Area (DSA) of the data warehouse can be abstractly considered as a set of materialized views defined over distributed heterogeneous databases, involving little or no aggregation. For a given set of different source databases and a given set of queries that the data staging area has to service, there is a number of alternative sets of materialized views that the administrator can choose to maintain. Each of these sets has different refreshment and query-answering costs while some of them may require more disk space than the space available for the data warehouse. Also, each of these views can be differently evaluated with respect to the quality of its data (in terms of accuracy, freshness, consistency etc.). We will show that these quality factors can be computed from the respective quality factors of the

sources.

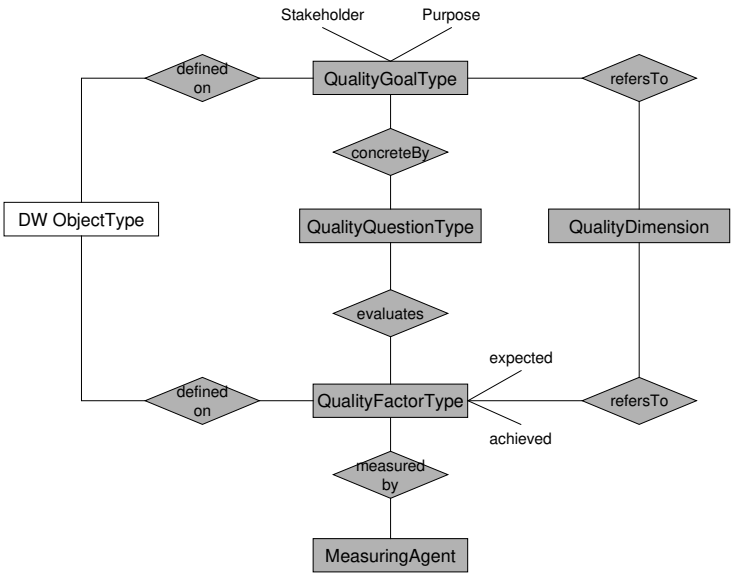

Fig. 1. The DWQ Quality Metamodel (simplified)

The DSA design problem is the selection of the set of materialized views with the maximum overall "benefit" (minimum overall "cost") that fulfill all the predefined requirements, set by the user (e.g., the solution fits into the available space). These views have to answer all the queries posed to the data warehouse without accessing the sources. Obviously, the benefit (or cost) can be quantified with respect to the aforementioned parameters as well as with respect to the total execution time of the design algorithm.

The overall problem of data warehouse design optimization, is focused on two *quality goals*: On the one hand, the quality goal "*design a DSA respecting the quality constraints set by the administrator*" is of interest. On the other hand, the administrator's quality goal "*evaluate the quality of the produced DSA and the respective queries*" is also of great importance.

3. Evaluation of the Quality of a Specific DSA Schema: Cost Model

In this section we will deal: (a) with the mathematical formulation of the problem, as already presented in [5,6,7,10,11,12] as a state space problem, and (b) the introduction of a novel cost model for the evaluation of the benefit (or actually cost) of each state, which takes into consideration the quality of the sources.

Mathematical formulation of the problem. We consider that a non empty set of queries $\mathbf{Q}$ is given, defined over a set of source relations $\mathbf{R}$. The data warehouse contains a set of materialized views $\mathbf{V}$ over $\mathbf{R}$ such that every query in $\mathbf{Q}$ can be rewritten completely over $\mathbf{V}$.

Thus, all the queries in $\mathbf{Q}$ can be answered locally at the data warehouse, without accessing the source relations in $\mathbf{R}$. By Q^V, we denote a complete rewriting of the query $Q \in \mathbf{Q}$ over $\mathbf{V}$. Consider a *Data Warehouse Configuration* (i.e., a set of views $\mathbf{V}$ and a set of rewritings $\mathbf{Q}^V$ of the queries over $\mathbf{V}$) $\mathbf{C}=<\mathbf{V}, \mathbf{Q}^V>$. The data warehouse design problem can then be stated as follows:

– *Input*: a set of source relations $\mathbf{R}$, a set of queries $\mathbf{Q}$ over $\mathbf{R}$, a set of constraints involving specific thresholds for several quality factors and a cost model involving different quality factors, each with different importance (weight).

– *Output:* a data warehouse configuration $\mathbf{C}=< \mathbf{V}, \mathbf{Q}^V >$ such (a) that all the constraints are satisfied and (b) the cost of the solution is minimal.

The problem is investigated for the case of Selection-Projection-Join (SPJ) conjunctive queries without self-joins. A set of SPJ views $\mathbf{V}$ can be represented by a multiquery graph. A multiquery graph allows the compact representation of multiple views by merging the query graphs of each view. For a set of views $\mathbf{V}$, the corresponding multiquery graph, $\mathbf{G}^V$, is a node and edge labeled multigraph resulting by merging the identical nodes of the query graphs for the views in $\mathbf{V}$.

We define the following five transformation rules that can be applied to a data warehouse configuration (see [11] for a broader discussion): (a) *edge removal* (when a join is removed from the views and added to the queries), (b) *attribute removal* (elimination of redundant attributes from projection lists), (c) *view break* (when the graph of the view is bipartite), (d) *view merging* (when a view V_1 implies another view V_2) and (e) *attribute transfer* (where an attribute is removed from a view which has this attribute pinned to a particular value).

Quality Evaluation. We define the cost of a specific *state* as a function of three major quality factors: query cost, update cost (which combined form the composite quality factor operational cost) and data quality (which is the combination of quality factors such as accuracy, completeness, freshness and consistency). Formally, if $\mathbf{V}$ is a state, and w, w_q, w_u, w_1, w_2, w_3, w_4, are constant positive weights, smaller or equal to 1, such that $w_q+w_u=1$ and $w_1+w_2+w_3+w_4=1$ we have formulae (1)–(3).

$$\text{Operational cost}(\mathbf{V}) = w_q * \text{query cost}(\mathbf{V}) + w_u * \text{update cost}(\mathbf{V}) \quad (1)$$

$$\text{Data quality}(\mathbf{V}) = w_1 * \text{accuracy}(\mathbf{V}) + w_2 * \text{freshness}(\mathbf{V}) + w_3 * \text{completeness}(\mathbf{V}) + w_4 * \text{consistency}(\mathbf{V}) \quad (2)$$

$$\text{Cost}(\mathbf{V}) = w * \frac{\text{operational cost}(\mathbf{V})}{\text{data quality}(\mathbf{V})} \quad (3)$$

To compute the quality factors of a state, we employ the quality factors of the materialized views that form this state. More specifically, if we denote a specific state $\mathbf{V}$ as a set of views $\{V\}$, we can compute its aforementioned quality factors as in formulae (4)–(7).

$$\text{Query cost}(\mathbf{V}) = \sum_{V} \text{query cost}(V) \quad (4)$$

$$\text{Update cost}(\mathbf{V}) = \sum_{V} \text{update cost}(V) \quad (5)$$

$$\text{Accuracy}(\mathbf{V}) = \frac{\sum_{v} \text{popularity}(V) * \text{accuracy}(V)}{\sum_{v} \text{popularity}(V)} \quad (6)$$

$$\text{popularity}(V) = \text{number of queries using V} \quad (7)$$

The formula for accuracy applies also for completeness, consistency and freshness.

4. Evaluation of the Quality of a View

After defining the quality factors for a state as functions over the quality factors of its views, all that remains is to define the formulae for the quality factors of a view, by using the quality factors of the source relations. The evaluation of the query cost of selections, projections and joins is done in terms of estimation of the time needed to compute the result. For the computation of the update cost, for each source table R, the model assumes the existence of two auxiliary differential tables ΔR^{+}, ΔR^{-}, which are used to capture the insertions and deletions over the previous version of R. The specific formulae are found in [10,13].

As far as the computation of the data quality of a view is concerned, in this section we provide specific cost formulae for different quality factors, namely *freshness*, *accuracy*, *completeness* and *consistency*. Given a source relation R, we define metrics for the expected quality of $\pi_{\mathbf{A}}(R)$, $\sigma_{\varphi}(R)$, $R \rhd\lhd S$. Orthogonally to these definitions, we explore the possibility of annotating each source with quality factors in more than one ways. Specifically, we explore the possibility of assigning a single value for the quality of (a) the whole relation, (b) several attributes of the relation (i.e., assuming a *vertical partitioning* of the quality of the relation), and (c) a subset of the rows of the relation (i.e., assuming a *horizontal partitioning* of the quality of the relation). In the sequel, when we refer generically to any of the aforementioned options (i.e., relations, attributes, or subsets of relations) we will call them *relation elements*.

There are several reasons that make these assumptions realistic. Remember that the DSA is a centralized pool of data within the data warehouse, acting as a gatherer and cleaner of data from the sources. Normally, in the lower materialization level of the DSA one can find reconciled "mirrors" of the source tables/files. In the first case, where a single quality factor is assigned to a whole relation, we do not take into consideration how the data of a higher-level table are collected. In the second case (vertical partitioning of quality), the involved table is collecting different measures for the same entity from different sources. Suppose, for example, the case of a DSA table joining facts from the manufacturing and the inventory database of a company: different quality measures can be applied to the measures coming from the different sources. In the third case (horizontal partitioning of quality) the table of the DSA is the union of the data coming from different sources, whose quality may be different.

In the next subsections, we provide definitions and metrics for the aforementioned quality factors. For the cases (b) and (c) we will also provide formulae that compute the value of the quality factors for the whole relation, based on its elements. Moreover, for all three cases, we explore the behavior of the quality of a relation, when a relational operator is applied to it.

Notation. We assume that $\mathbf{A}$ is a set of attributes and φ is a selection condition defined as a conjunction of terms of the form $A\theta v$, where A is an attribute, v is a value and θ belongs to the set $\{=, \leq, \geq, \neq, <, >\}$. Also, let r_i, s_j be relation elements (i.e., relations, attributes or groups of rows) and r_i', s_j' be the respective relation elements after the application of selection, projection and join operations.

4.1 The Freshness Quality Factor

We define freshness as the age of the information stored in the data warehouse. To make this intuitive definition more formal, we suppose that for each relation element r we can assume a maximum time delay for its refreshment, which we call *max delay(r)*. In other words, *max delay(r)* is the longer period that the relation element r is left without being updated and we still accept it as valid for querying purposes. Then, freshness can be defined as formula (8) indicates.

$$\text{freshness}(r) = \max\left(0, \frac{\text{max delay} - \text{time elapsed from last update}}{\text{max delay}}\right) \quad (8)$$

In the sequel, we will explore the behavior of freshness, under the application of the select, project and join operations over a set of underlying relation elements.

$$\text{freshness}(r_i') = \text{freshness}(\pi_{\mathbf{A}}(r_i)) = \text{freshness}(r_i) \quad (9)$$

$$\text{freshness}(r_i') = \text{freshness}(\sigma_{\varphi}(r_i)) = \text{freshness}(r_i) \quad (10)$$

$$\text{freshness}(r_i') = \text{freshness}(r_i \triangleright\triangleleft s_j) = \min(\text{freshness}(r_i), \text{freshness}(s_j)) \quad (11)$$

Case 1: Freshness is defined over the whole relation. We assume that the quality of the source is uniformly distributed over all the rows of the relation. The quality of the result of the application of the aforementioned relational operators is defined from equations (9)-(11). Note that for the case of join, we follow the general guideline that a row is as old as the oldest of its attributes.

Case 2: Freshness is defined over the attributes of a relation. Suppose a vertical partitioning for the quality of a relation $R(A_1, A_2, \ldots, A_n)$, i.e., some attributes of R are annotated with a value for their freshness. We define the freshness of R as the minimum of all the freshness values of the annotated attributes, i.e., each table is as old as the oldest of its attributes. In this definition, we ignore the columns whose freshness is not set. In this case, if we assume that the freshness is uniformly distributed among the rows of the relations, the quality of each column after the application of selection, projection and join, can be computed from the formulae (9)-(11).

Case 3: Freshness is defined over groups of a relation. Suppose now, a horizontal partitioning for the quality of a relation R. Formally, we assume a finite set of disjoint and complete selection conditions $\mathbf{\Phi}=\{\varphi_i\}$, $i=1,..,m$ that divide the relation into a finite set of sub-relations. We will call each of the produced set of rows as a *group*. Then if the relation R is the union of the groups g_i, $i=1..m$, we define the freshness of R, as in formula (12).

$$\text{Freshness}(R) = \frac{\sum_i \text{size}(g_i) * \text{freshness}(g_i)}{\sum_i \text{size}(g_i)}, \; i = 1,..,m \qquad (12)$$

Once more, the application of the selection, projection and join operations implies the usage of the formulae (9)-(11) for the computation of the freshness of the groups of the relation R.

As far as the projection is concerned, things are simple. Even if the definition of the groups is changed after the projection (i.e., certain attributes participating in the conditions of Φ are projected out), the freshness of the remaining rows is not affected. What could be affected, though, is the freshness of a new group, which can be recomputed from formula (12).

In the case of selection, there is no effect on the freshness of the different groups, although the overall quality of the relation can be affected. If the selection condition does not involve attributes of Φ, we can assume that the freshness is uniformly spread over the rows of the result (i.e., the attributes of the selection condition are independent of the attributes of Φ). If, on the other hand, the selection condition involves attributes of Φ, freshness is not affected again, since the granule for the definition of freshness is the group. In the case that only a subset of a group remains after the selection condition, we have no extra information on its quality, thus we can assume that it remains unaffected. If, on the other hand, a whole group is left out of the result, this affects only the whole relation, but not the freshness of the remaining groups.

In the case of join, things are straightforward again, since a join is the combination of a Cartesian product and a selection operation. Each row of R, belonging to a certain group r_i, is joined to row of a certain group s_j, of table S. The freshness of the resulting tuple is again the minimum freshness of the involved tuples. If table R has n groups and table S has m groups respectively, we can anticipate m*n groups for the result. Naturally, existing results on automatic reasoning for possible incompatibilities between groups can be exploited in this case too [8], in order to eliminate meaningless groups in the result.

4.2 The Consistency Quality Factor

We define consistency as the percentage of the data of a relation that abide by the integrity rules of the database. In other words, we can define consistency as the fraction of tuples respecting these rules over the total number of tuples of a relation. Remember that one of the reasons for the existence of the DSA is data cleaning; thus integrity rules are not always enforced by declaration, but rather checked by scanning the contents of the imported data.

Again, we shall consider the three options that we explored in the case of freshness. Suppose a vertical partitioning for the quality of a relation $R(A_1, A_2, \ldots, A_n)$, i.e., some attributes of R are annotated with a value for their consistency. This means that we count the distinct number of values of each column and derive the percentage of acceptable ones as the consistency of the column. We define the consistency of the relation as the weighted sum of the individual values for the consistency of the columns of the relation. The weights are arbitrarily set by the designer.

In the case where we assign different values for consistency to subsets of the rows of a relation, we can estimate the overall consistency of the table as a weighted sum of the individual values, with the size of each group playing the role of weight. Supposing that we have m different groups of rows, the relation R is the union of the groups g_i, $i=1..m$, and the overall consistency of the relations R, is given by formula (14). Next, we proceed to investigate the effects of the application of the selection, projection and join operations over a relation with respect to its consistency.

$$\text{consistency}(R) = \sum_1^n w_i * \text{consistency}(A_i),\ \sum_1^n w_i = 1 \tag{13}$$

$$\text{consistency}(R) = \frac{\sum_i \text{size}(g_i) * \text{consistency}(g_i)}{\sum_i \text{size}(g_i)},\ i = 1,..,m \tag{14}$$

$$\text{consistency}(r_i') = \text{consistency}(\pi_{\mathbf{A}}(r_i)) = \text{consistency}(r_i) \tag{15}$$

$$\text{consistency}(r_i') = \text{consistency}(\sigma_\varphi(r_i)) = \text{consistency}(r_i) \tag{16}$$

$$\text{consistency}(r_i \rhd\lhd s_j) = \text{consistency}(r_i) * \text{consistency}(s_j)) \tag{17}$$

Case 1: Consistency is defined over the whole relation. Then, we can assume that the quality of the source is uniformly distributed over all the rows of the relation. The quality of the result of the application of the aforementioned relational operators is defined by the equations (15)-(17).

For the case of join, simple mathematics can prove the aforementioned formula. Suppose that c_S is the percentage of consistent rows of S and c_R the respective percentage for R. Then, we have two groups for each table, the consistent and the non-consistent group. A row from relation R, is joined to both consistent and non-consistent rows of relation S. If we assume that the result of the join of two consistent tuples is also consistent and denote the selectivity of the join as j, the number of consistent rows of the result is $j*(c_R*\text{size}(R))*(c_S*\text{size}(S))$ and the total number of rows is $j*\text{size}(R)*\text{size}(S)$. The fraction of these two measures gives us formula (17).

Case 2: Consistency is defined over the attributes of a relation. In this case, we assume that for each column A_i we have a measurement $\text{consistency}(A_i)$ as the percentage of its consistent values. We denote the result of the application of an operator as $\text{consistency}(A_i')$. Then, we can define the cost formulae (21)-(23).

$$\text{If } \pi_{\mathbf{A}}(R),\ \text{consistency}(A_i') = \text{consistency}(A_i') \tag{21}$$

$$\text{If } \sigma_\varphi(R),\ \text{consistency}(A_i') = \text{consistency}(A_i') \tag{22}$$

$$\begin{aligned} &\text{If } R \rhd\lhd_{A=B} S, \\ &\text{(a)} \quad \text{consistency}(A_i') = \text{consistency}(A_i') \text{ for all } A_i \neq A,B \text{ and} \\ &\text{(b)} \quad \text{consistency}(A_i') = \text{consistency}(C) \text{ for } A_i = A', B' \text{ and} \\ &\qquad C \in \{A,B\} \text{ s.t. } \text{distinct}(C) = \min(\text{distinct}(A), \text{distinct}(B)) \end{aligned} \tag{23}$$

The projection operation clearly does not affect the value of the quality factor. For the selection operation, we assume a uniform distribution of inconsistencies over the values of a

column; thus the selection leaves unaffected the consistency of the involved columns. For the case of the join operation, we assume that there is no correlation for the columns which are not involved in the join; thus we their consistency is not affected. As far as the columns involved in the join are concerned, the column with the fewest distinct values determines the number of distinct values of the result. Suppose `B` is this column (i.e., `C=B` for formula `23`). If we denote the selectivity of the join as `j`, the number of distinct values of the result is `j*distinct(B)` and the number of consistent ones is `j*distinct(B)*consistency(B)`.

Case 3: Consistency is defined over groups of a relation. Again, we assume that the horizontal partitioning for the consistency is based on a finite set of disjoint and complete selection conditions over relation `R`. For all three cases we have similar results as in the case of freshness; thus we do not elaborate more on this issue.

4.3 The Accuracy and Completeness Quality Factors

We define accuracy as the percentage of the data of a relation that capture the actual, real-world values of the entities they represent. In other words, we can define accuracy as the fraction of tuples respecting this property over the total number of tuples of a relation. Due to its definition, all the formulae regarding accuracy are identical to the ones regarding consistency; thus we do not repeat them here.

We also define completeness as the fraction of stored values over the number of expected values. In real systems, administrators normally know the size of data they are expecting to receive/produce from a database. Moreover, one can also automate the calculation of this quality factor by counting NULL values in the proper columns. The formulae for the calculation of completeness under the fragmentation assumptions that we have made for the rest of the quality factors are also identical to the ones of accuracy and consistency. Practically, the only difference lies in the fact that the number of stored values is moved from the denominator to the numerator for all the proofs on the formulae of completeness. Strangely enough, this does not affect the final result, which is identical to the formulae for consistency and accuracy.

5. Conclusions and Future Work

In this paper we have investigated the quality-oriented DSA configuration problem. We have extended previous approaches to the problem, (a) by formally treating the problem with respect to specific, objective quality factors, and (b) proposing extended cost formulae for the evaluation of the benefit of a solution, so that quality factors are taken into consideration for the computation of the final views of the data warehouse. For the latter, different options have been considered for the characterization of the quality of relations (global annotation with quality factors, annotation of subsets of rows and annotation of specific columns).

An interesting direction of further research is based on the remark that modeling the DSA as a set of materialized views is a clever, but incomplete abstraction of the real world. In practice, one of the main roles of the DSA is the reconciliation and cleaning of the heterogeneous and possibly inconsistent data that come from the data sources. This involves complex computations, schema transformations and integrity checks that are not yet modeled from our approach.

References

1. H. Gupta and I. S. Mumick: Selection of Views to Materialize Under a Maintenance Cost Constant. In *Proc. of the 7th International Conference on Database Theory (ICDT)*, pages 453-470, Jerusalem, Israel, 1999
2. M. Jarke, M.A.Jeusfeld, C. Quix, P. Vassiliadis: Architecture and quality in data warehouses: An extended repository approach. *Information Systems*, **24**(3): 229-253 (1999).
3. M.A. Jeusfeld, C. Quix, M. Jarke: Design and Analysis of Quality Information for Data Warehouses. In *Proc. of the 17th International Conference on the Entity Relationship Approach (ER'98)*, Singapore, 1998.
4. W. Labio, D. Quass and B. Adelberg: Physical Database Design for Data Warehousing. In *Proc. of the 13th International Conference on Data Engineering (ICDE)*, pages 277-288, Birmingham U.K., 1997
5. S. Ligoudistianos, D. Theodoratos and T. Sellis: Experimental Evaluation of Data Warehouse Configuration Algorithms. In *Proc. of the 9th International Workshop on Database and Expert Systems Applications (DEXA), Data Warehouse Design and OLAP Technology (DWDOT'98)*, pages 218-223, Vienna, Austria, 1998
6. S. Ligoudistianos, T. Sellis, D. Theodoratos, Y. Vassiliou. Heuristic Algorithms for Designing the Global Data Warehouse with SPJ Views. In *Proc. International Conference on Data Warehousing and Knowledge Discovery, (DaWaK'99)*, pp. 66-105, Springer LNCS 1676, (1999).
7. F. Naumann, U. Leser, J.C. Freytag: Quality-Driver Integration of Heterogeneous Information Systems. *In Proc. of the 25th International Conference of Very Large Databases (VLDB)*, pages 447-458, Edinburgh, Scotland, 1999
8. M. T. Ozsu, P. Valduriez. Principles of Distributed Database Systems. *Prentice Hall*. 1999
9. K.A. Ross, D. Srivastava and S. Sudarshan: Materialized View Maintenance and Integrity Constraint Checking. Trading Space for Time. *In Proc. of the ACM SIGMOD Conf. On the Management of Data,* pages 447-458, 1996
10. D. Theodoratos, S. Ligoudistianos, T. Sellis. Designing the Global Data Warehouse with SPJ queries. In *Proc. of the 11th International Conference on Advanced Information Systems Engineering (CAiSE'99)*, pp. 180-194, Springer LNCS 1626, (1999).
11. D. Theodoratos and T. Sellis: Data Warehouse Configuration. *In Proc. of the 23rd International Conference of Very Large Databases (VLDB)*, pages 126-135, Athens, Greece, 1997
12. D. Theodoratos and T. Sellis: Data Warehouse Schema and Instance Design. *In Proc. of the 17th Intl. Conf. On Conceptual Modeling (ER'98), 1998.*
13. J. Ullman. Principles of Database and Knowledge Base Systems, vol. I. *Computer Science Press*. 1988
14. P. Vassiliadis, M. Bouzeghoub, C. Quix: Towards Quality-Oriented Data Warehouse Usage and Evolution. *Information Systems*, **25**(2): 89-115 (2000).
15. P.Vassiliadis, S.Ligoudistianos, T. Sellis, D. Theodoratos. Quality-Oriented Optimization of the Design of the Global Data Warehouse (long version). Available at http://www.dblab.ece.ntua.gr/~pvassil/publications/epy01_long.ps.gz
16. J. Yang, K. Karlapalem, Q. Li: Algorithms for Materialized View Design in Data Warehousing Environment. *In Proc. of the 23rd International Conference of Very Large Databases (VLDB)*, pages 136-145, Athens, Greece, 1997