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,

Tel: +301-772-1602, 1402

Fax: +301-772-1442

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.

Keywords : Data Warehouse Design, Data Quality, Data Staging Area.

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. Due to the number of the involved stakeholders, the quality of data is of highly *subjective* nature and should ideally be treated differently for each user. On the other hand, in real data warehouse systems, data deficiencies, non-availability or accessibility problems are objective problems that significantly decrease the overall quality of the system. The question that arises, then, is how to tune the design choices in such a way that all the different, and sometimes opposing, user requirements can be simultaneously satisfied. This problem is typically illustrated by the physical design of the data warehouse where the problem is to find a set of materialized views that optimize both the response time for user requests and the global data warehouse maintenance cost.

To address the problem, we will employ a common abstraction of a data warehouse system. In our approach, a data warehouse architecture exhibits various layers of data in

which data from one layer are derived from data of the lower layer. *Data sources*, also called operational databases, form the lowest layer. They may consist of structured data stored in open database systems and legacy systems, or unstructured or semi-structured data stored in files. 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 upper layer of views consists of the local, client warehouses, called *Data Marts*, which contain highly aggregated data, directly derived from the global warehouse. There are various kinds of data marts such as the OLAP databases which may be based on relational database systems or proprietary multidimensional data structures.

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. On the other hand, 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. Each of these sets provides different values for the various quality aspects relevant to the data staging area design. Such a quality factor is for example the *DSA operational cost* which depends on the cost of two basic operations: query answering and the view refreshing. Trying to select a set of views to materialize in order to maximize the overall quality of the data staging area, is the *Quality-Oriented DSA Design Problem*.

Early works [LSTV99,LiTS98,ThLS99,ThSe98,ThSe97] study the DSA design problem and provide methods that generate the view selection from the input queries in order to minimize the operational cost of the data staging area. In these papers the problem is modeled as a state space problem providing algorithms that deliver the optimal solution very rapidly.

In this paper we extend the 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).

The structure of this paper is as follows: in section 2 we specifically define the problem in terms of quality factor definition and measurement and introduce the general framework of the DWQ approach. In section 3 we examine the evaluation of a particular solution to the problem and in section 4 we present the specific cost model for the evaluation of a view. In section 5 we present related work and in section 6 we conclude our results and present future work.

2. General Setting of the Problem

In this section, we are going to survey the methodology for data warehouse quality management, as proposed by the Esprit LTR Project *DWQ* [JV97], as well as its specific specialization for the data warehouse design optimization problem.

2.1 The DWQ Architecture and Quality Metamodel

In the DWQ project we have advocated the need for enriched metadata facilities for the exploitation of the knowledge collected in a data warehouse. In [JJQV99] it is shown that the data warehouse metadata should track both architecture components and quality factors. The general framework for the categorization of data warehouse metadata is presented in Fig. 1.

The framework employs three *layers* of abstraction to describe the data warehouse. On the *metamodel* layer (upper part of Fig. 1), the framework gives a notation for data warehouse architectures by specifying meta-classes for the usual data warehouse objects like "data store", "relation", "view", etc. On the *metadata* layer, the metamodel is instantiated with the concrete architecture of a data warehouse, involving its schema definition, indexes, table spaces, etc. The lowest layer in Fig. 1, which is the *instance layer*, represents the actual processes and data.

Due to the complexity of the data warehouse architecture, we have employed a further classification of data warehouse metadata entities, in a way orthogonal to the three aforementioned layers of abstraction. Specifically, in order to describe a data warehouse we have used a classical database classification of three *perspectives*: a *conceptual*, a *logical* and a *physical* perspective. The *conceptual perspective* captures the world in a vocabulary really close to the one of the final user, employing formalisms like the Entity-Relationship model; the *physical perspective* covers the data warehouse environment in terms of real world, computer-oriented components or events; and finally, the *logical perspective* acts as an intermediate between these two extremes in order to provide a concrete vocabulary of the data warehouse environment, being independent, though, from strict implementation details (employing formalisms like, for example, the relational model).

Finally, for practical reasons, each perspective is partitioned into the three traditional data warehouse *levels*: *source, data warehouse* and *client* level, so that the entities of the data warehouse metadata repository can be attributed to their proper "location" in the data flow of the system.

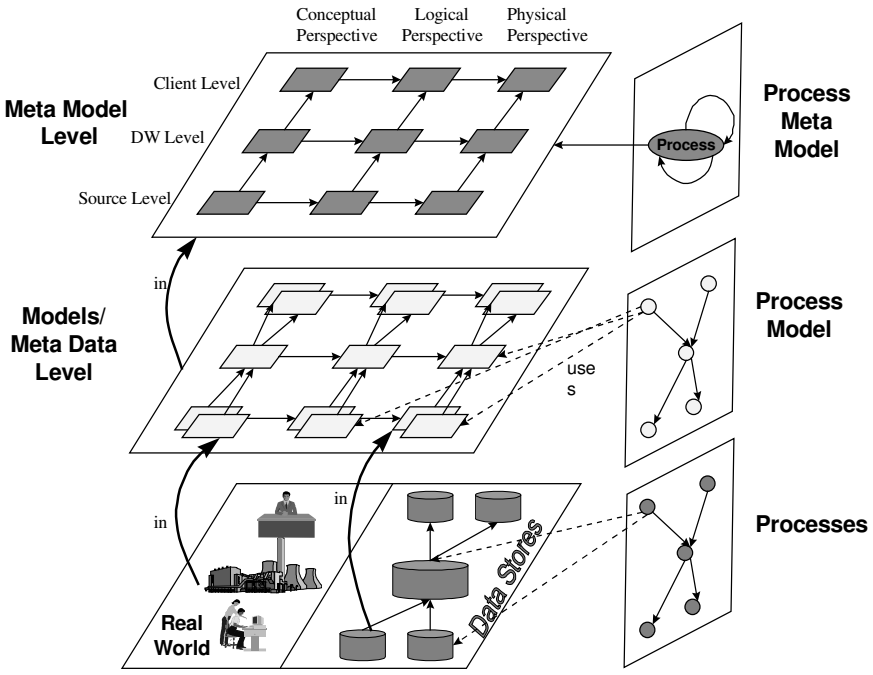

Fig. 1. The Data Warehouse Architecture Framework

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 DWQ quality metamodel (first published in [JQJ98]) is based on the Goal-Question-Metric approach (GQM) of [OB92], originally developed for software quality management. Fig. 2 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, 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 [VBQ00].

2.2 The Problem of DSA Design Optimization

In this section, we specialize the problem of quality-oriented DSA design optimization as a specific instance of the general data warehouse quality management problem. Although there have been many attempts to solve this problem in the past [GuMu99, LaQA97, RoSS96, YaKL97], to our knowledge this is the first time that the formal definition of quality is taken into consideration for the modeling and solution of this particular problem.

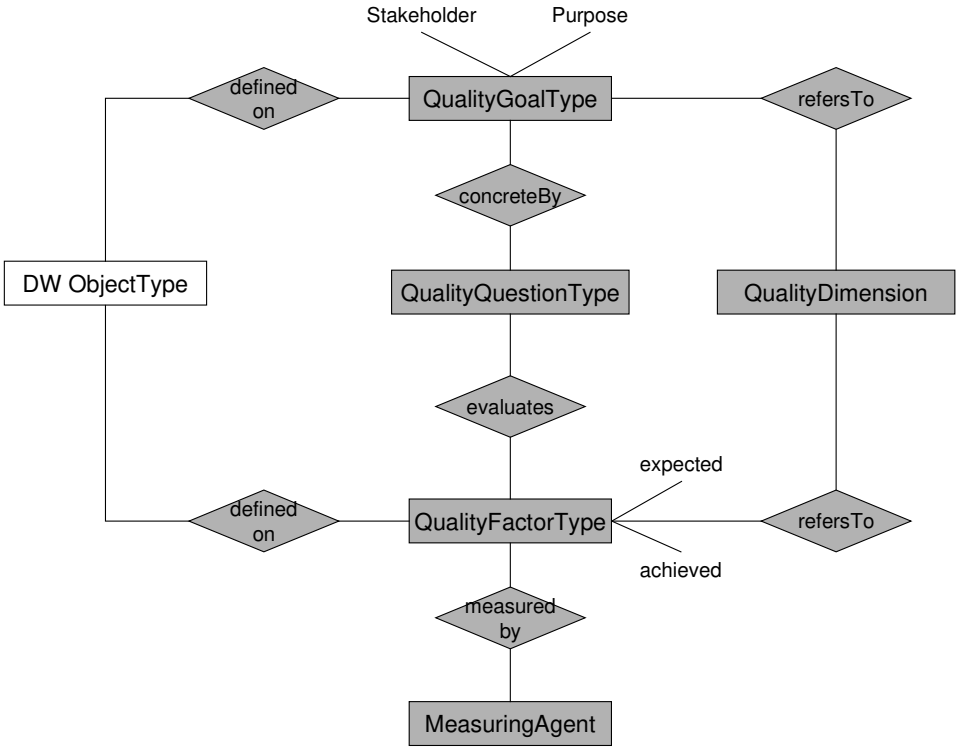

Fig. 2. The DWQ Quality Metamodel (simplified)

Problem Description. 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. All the queries posed to the DSA are evaluated locally using exclusively the data that are stored in these views. The materialized views have also to be refreshed when changes occur to the data of the sources. Note that the term "queries" includes both the possible ad-hoc user queries (seeking for detailed information) and the batch, regularly posed queries used for updating of high level, aggregated views of the data warehouse and its data marts.

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.

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.

Object Types. We can determine three basic object types, for the problem of optimal design of the data staging area: the *sources*, which are considered to be relational (or at least provide a relational interface through a gateway or wrapper), the *queries* posed from the user over these sources and the *intermediate materialized views* that the algorithms produce, which serve as buffers speeding up the answering of the queries. Another interesting object of the problem, although implicit in the problem definition, is the employed algorithm itself.

Quality dimensions. The basic quality dimensions affecting the problem are mainly related to time and space:

1. *Schema Minimality*: the redundancy in the data staging area is kept low, for ease of administration.
2. *Design Efficiency*: the choice of the materialized views must be done as quickly as possible.
3. *Design Consistency*: the constraints over the result of the design process are respected (e.g. the volume of the produced materialized views must be such, that fits in the disk space provided for the data staging area).
4. *Query Timeliness*: the answering of the queries must be done as quickly as possible.
5. *System Availability*: the maintenance of the materialized views must be done as quickly as possible.
6. *Data Quality*: the data delivered to the users must be accurate, complete, up to date and abiding by the internal rules (constraints) of the data warehouse.

Quality factors. For each of the related objects of the problem, we can define relevant quality factors that affect the result of the design of the data staging area. The views of the DSA are related to several quality factors such as the *final number of views in the DSA*, the *available disk space for the DSA*, the *space occupied by each view* (as well as the *total space occupied by al the views*), the *update cost for view* (i.e., the time to perform the update of a view of the data staging area) as well as the *total update time for the DSA*. The queries are characterized, from their *query cost* (i.e., the time needed to perform the query over the data staging area). All objects are also characterized by their data quality. The quality factors

related to data quality are the *accuracy* (i.e., the validity of the data, with respect to real world values), the *completeness* (i.e., the percentage of available information with respect to expected volume of information), the *consistency* (i.e., the percentage of information obeying the database rules), and *the freshness* (i.e., the age of the data with respect to their transaction time). Finally, the design algorithm itself is characterized by the *design time* (i.e., the total time needed for the algorithm to terminate). All the aforementioned quality factors are listed in Table 1, along with their precise definition and measurement method.

Quality Factor	**Definition**	**Measurement Method**	**Related Dimension**
Number of DSA views	The final number of views in the DSA.	Simple enumeration	Schema minimality
Design time	Total time needed for the algorithm to terminate	Through a trivial counter	Design efficiency
Disk space	Available disk space for the DSA	Trivial	Design consistency
Space occupied by each view	Space occupied by each view	Rows * size(row)	
DSA space	Total space occupied from all the DSA views	Sum of the space occupied by all the views	
Query cost for view	Time to perform the query over the DSA	See cost model (Section 3)	Query timeliness
Total query cost	Total time needed for all the queries to be answered from the DSA	Sum of all the query costs	
Update cost for view	Time to perform the update of a DSA view	See cost model (Section 3)	System availability
Total update cost	Total time needed to update all the views of the DSA	Sum of all the update costs	
Accuracy	The validity of the data, wrt. real world values.	Percentage of true over stored facts, or estimation, or sampling	Data quality
Completeness	The percentage of available information wrt. expected volume of information	Percentage of stored over expected facts, or estimation, or sampling	
Consistency	Percentage of information obeying the database rules	Percentage of facts obeying rules over stored facts, or estimation	
Freshness	Age of data wrt. their acceptable age	Maximum timestamp of the data obtained from log scan, or a dedicated column, or temporal DBMS techniques	

Table 1. Quality factors for the Data Warehouse Design Optimization problems.

General overview of the problem. The overall problem of data warehouse design optimization, composed of its objects and quality factors is graphically depicted in Fig. 3. We

present the basic object types (i.e., sources, DSA views and queries) and for each of them we give their relevant quality factors (as already described in this section). Apart from the quality factors and the participating objects, two *quality goals* are depicted in Fig. 3. On the left side, the quality goal "*design an DSA respecting the quality constraints set by the administrator*" is depicted. On the right side, we depict the administrator's quality goal "*evaluate the quality of the produced DSA and the respective queries*".

In the next sections of the paper we will investigate only the design problem (quality goal), since the evaluation problem can be trivially treated as a special case of the former problem. Practically, in the latter case, quality factor act as thresholds for a solution which is produced from an algorithm. For example, one can have constraints like: "the size of the result must not exceed the size of the disk", or "the completeness of all the views is greater than 80%". These problems can be trivially dealt since each time a new state is explored, it can be approved or not by evaluating the predefined constraints over this solution.

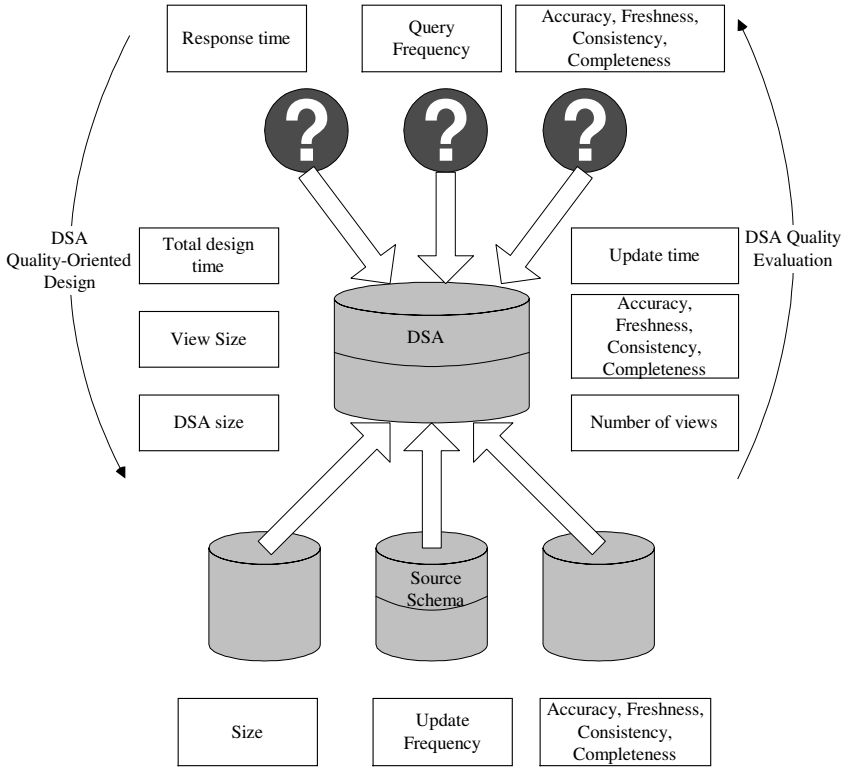

Fig. 3. The Quality-Oriented DSA Design Problem [VaBQ00].

3. Evaluation of the Quality of a DSA Schema

In this section we will deal (a) with the mathematical formulation of the problem, as

already presented in [LiTS98,ThLS99,ThSe98,ThSe97] 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 of the queries over $\mathbf{V}$, $\mathbf{Q}^V$) $\mathbf{C}=<\mathbf{V},\mathbf{Q}^V>$. 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.
- 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. The relation attributes take their values from domains of integer values. Atomic formulas are of the form `x op y + c` or `x op c`, where `x`, `y` are attribute variables, `c` is a constant, and `op` is one of the comparison operators =, <, > , ≥, ≤ but not ≠. A formula `F` implies a formula `F'` if both involve the same attributes and `F` is more restrictive (for example `A=B` implies `A≤B+10`). Atoms involving attributes from only one relation are called *selection atoms*, while those involving attributes from two relations are called *join atoms*.

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 [ThSe97] for a broader discussion):

Edge Removal: A new configuration is produced through the elimination of an edge label by the atom `p` from the query graph of view `V`, and the addition of an associated condition into the queries that are defined over this view. If the query graph of `V` was not kept connected after the edge elimination, `V` is decomposed in two separate views.

Attribute Removal: If we have atoms of the form $A=B$ and A, B are attributes of a view V, we eliminate A from the projected attributes of V.

View break: Let V be a view and N_1, N_2 two sets of nodes labeled by V in G^V such that: (a) $N_1 \not\subset N_2$, $N_2 \not\subset N_1$, (b) $N_1 \cup N_2$ is the set of all the nodes labeled by V in G^V and (c) there is no edge labeled by V between the nodes in $N_1 - N_2$ and $N_2 - N_1$. In this case we replace V by two views V_1 (defined over N_1) and V_2 (defined over N_2). All the queries defined over V are modified to be defined over $V_1 \bowtie V_2$.

View Merging: A merging of two views V_1 and V_2 can take place if every condition of $V_1(V_2)$ implies or is implied by a condition of $V_2(V_1)$. In the new configuration, V_1 and V_2 are replaced by a view V that is defined over the same source relations and comprising all the implied predicates. All the queries defined over V_1 or V_2 are modified appropriately in order to be defined over V. Eliminated selection and join conditions are added to these queries.

Attribute Transfer: Let us have atoms of the form $A=c$ where A is an attribute of a view V of a labeled node N and c a constant. We replace V by V_1 with the same nodes and edges but without the attribute A. We also create the view $V_2 = \pi_A(\sigma_{A=c}N))$. We replace every occurrence of V in $\mathbf{Q^V}$ by $V_1 \bowtie V_2$.

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 the following formulae:

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

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

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

To compute the quality factors of a state, we employ the quality factors of the materialized views that comprise 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 follows:

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

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

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

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

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

One can notice that we choose the popularity (i.e., frequency of reference) of a view as its weight in the formulae for data quality. This is not a clear choice, since other measures

(like for example, size, or number of rows) could be used for that purpose. Still, instead of a purely mathematical formulation of the cost model, we chose to follow a more empirical approach, based on the observation that the quality of some piece of information is heavily dependent on the usage of this information. As [Orr98] mentions "the problem of data quality is fundamentally intertwined in how [...] users actually use the data in the system", since the users are the ultimate judges of the quality of the data produced for them: if nobody actually uses the data, then nobody will ever take care to improve its quality[1].

Notice also, that since data quality is a positive number, the greater its value the lower the respective "cost" value. This, of course, is not affected by the fact that data quality is defined as lower or equal to 1 (as we shall also see in the sequel).

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 [Ullm88], [ThLS99].

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 $\pi_{\mathbf{A}}(R)$, $\sigma_{\varphi}(R)$, $R \bowtie S$. 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. In the third

[1] It is interesting to note here, that out of the several definitions for data quality that we have encountered in the literature, the best seems to be the one found in [Orr98, TaBa98] where data quality is defined as "fitness for use".

case (horizontal partitioning of quality) the table of the DSA is the union of the data coming from different sources, of which the quality is known. In Fig. 4 we present a small motivating example to explain these ideas. Suppose a multinational manufacturing company producing and selling parts. Suppose also, the existence of different databases in the manufacturing and inventory departments, as well as in the European and American headquarters. In the DSA, we wish to collect all the information about parts and orders. In higher levels of granularity, we want to answer user queries on the combination of these relations. Suppose the following schemata:

ORDER, $ORDER_{A,E}$	(ORD_DATE, CUST_ID, PART_ID, QTY, AGREED_UNIT_PRICE, DELIVERY_DATE)
$PART_I$	(PART_ID, INV_DATE, AVAIL_QTY)
$PART_M$	(PART_ID, MAN_DATE, PROD_QTY, UNIT_COST)
PART	(PART_ID, MAN_DATE, PROD_QTY, UNIT_COST, INV_DATE, AVAIL_QTY)

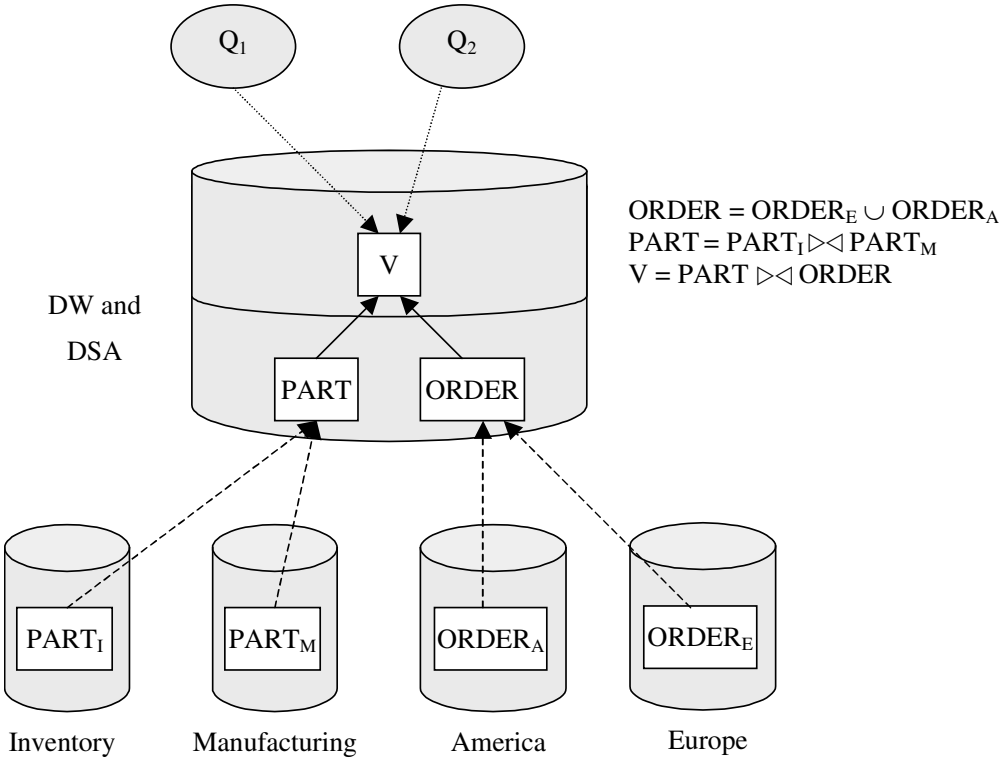

Fig. 4. Motivating Example

Naturally, to compose the information on PARTS in the DSA, we need to join the inventory and manufacturing tables on the primary key PART_ID. In this case, the quality of

columns PROD_QTY and AVAIL_QTY is different (i.e., we have an horizontal fragmentation of quality). In the case of ORDERS, where the DSA table is the union of the source tables, the tuples coming from the European database are of different quality from the tuples coming from the American one; thus we have a case of horizontal fragmentation of quality.

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.

4.1 The Freshness Quality Factor

We define freshness as the age of the information stored in the data warehouse. To make this imprecise definition more concrete, 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.

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

When a relation element is out of date, then its freshness is considered to be zero, whereas it is considered to be 1, exactly after the last update.

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. We assume that **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 $\{=, \leq, \geq, \neq, <, >\}$.

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).

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

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

$$\text{freshness}(R \bowtie S) = \min(\text{freshness}(R), \text{freshness}(S)) \quad (11)$$

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 select, project and join operations, 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 $\Phi=\{\varphi_i\}$, $i=1,..,m$ that divide the relation into a finite set of sub-relations. We assume that each selection condition is 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 $\{=,\leq,\geq,\neq,<,>\}$. 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).

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

In this case, we can no longer assume that the freshness is uniformly distributed among the rows of the relations. The application of the selection, projection and join operations implies the usage of the formulae (13)-(15) for the computation of the freshness of the groups of the relation R. Formally, let r_i be a group of relation R, s_j a group of relation S and r_i', s_j' be the respective groups after the application of the aforementioned relational operators. Then,

$$freshness(r_i') = freshness(\pi_{\mathbf{A}}(r_i)) = freshness(r_i) \qquad (13)$$

$$freshness(r_i') = freshness(\sigma_{\varphi}(r_i)) = freshness(r_i) \qquad (14)$$

$$freshness(r_i')=freshness(r_i \ltimes s_j)=min(freshness(r_i),freshness(s_j)) \qquad (15)$$

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 formulae 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). Suppose for example the following set of formulae over the relation R(A,B,C), each of which defines and a respective group of rows g_i, with quality $freshness(g_i)$:

φ_1: A > 4 and B ≤ 8,

φ_2: A > 4 and B > 8,

φ_3: A ≤ 4 and B ≤ 8,

φ_4: A ≤ 4 and B > 8

After the execution of $(\pi_{A,C}(R))$, the resulting formulae are φ_1':A>4 and φ_3':A≤4. The quality factors of the new groups can be computed 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 [OzVa99], 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. One could assign different measures to compute the overall consistency of a table from the individual values of the consistency of the columns, like for example, the minimum, the average or the product of the individual consistencies (in which case one assumes the independence of the values of the columns of the relations). To avoid making any of these assumptions 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. Naturally, average and minimum are trivially covered by this modeling.

$$\text{consistency}(R) = \sum_1^n w_i * \text{consistency}(A_i) = 1, \ \sum_1^n w_i = 1 \qquad (16)$$

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 (17).

$$\text{consistency}(R) = \frac{\sum_i \text{size}(g_i) * \text{consistency}(g_i)}{\sum_i \text{size}(g_i)}, \ i = 1,..,m \qquad (17)$$

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.

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 in the following equations.

$$consistency(\Pi_{\mathbf{A}}(R)) = consistency(R) \quad (18)$$

$$consistency(\sigma_{\varphi}(R)) = consistency(R) \quad (19)$$

$$consistency(R \bowtie S) = consistency(R) * consistency(S) \quad (20)$$

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. Still, only the join of consistent rows produces a consistent result[2]. If we denote the selectivity of the join as j, the number of consistent rows of the result is $j*(c_R*size(R))*(c_S*size(S))$ and the total number of rows is $j*size(R)*size(S)$. The fraction of these two measures gives us formula (20).

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 $consistency(A_i)$ as the percentage of its consistent values. We denote the result of the application of an operator as $consistency(A_i')$. Then, we can define the following cost formulae:

$$If\ \Pi_{\mathbf{A}}(R),\ consistency(A_i') = consistency(A_i') \quad (21)$$

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

$$If\ R \bowtie_{A=B} S, \quad (23)$$

(a) $consistency(A_i')=consistency(A_i')$ for all $A_i \neq A,B$ and

(b) $consistency(A_i')=consistency(C)$ for $A_i=A',B'$ and $C \in \{A,B\}$ s.t. $distinct(C)=min(distinct(A),distinct(B))$

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 number of distinct values of the result is determined by the column with the fewer distinct values. 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

[2] Here we assume that the result of the join of two consistent tuples is also consistent.

horizontal partitioning for the consistency is based on a finite set of disjoint and complete selection conditions $\Phi=\{\varphi_i\}$ over relation R. Each of these selection conditions defines a group over the relation R. Let r_i be a group of relation R, s_j a group of relation S and r_i', s_j' be the respective groups after the application of the relational operators. Then,

$$consistency(r_i')=consistency(\pi_{\mathbf{A}}(r_i))=consistency(r_i) \quad (24)$$

$$consistency(r_i')=consistency(\sigma_{\varphi}(r_i))=consistency(r_i) \quad (25)$$

$$consistency(r_i \times s_j)=consistency(r_i)*consistency(s_j)) \quad (26)$$

As far as the projection is concerned, we can assume that the inconsistencies are uniformly spread over the removed attributes (or else, we should have some measure of this in a vertical partitioning), resulting thus in the same value for consistency. Checking for primary key duplicates is an example of this case. In the case of selection, the arguments that we gave in the respective case of freshness apply here too. Since the group is the granule for the definition of quality, the quality of the remaining groups remains intact, no matter if they are restricted by the selection condition, or not. In the case of join, we have similar results as with the case of freshness. 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 resulting tuple is consistent only if both the generating tuples were consistent (here we make an implicit assumption that the join does not introduce extra violations of integrity rules). Thus, the possibility for the produced tuple to be consistent is the product of the possibilities of its generating tuples.

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. Measuring accuracy is not a simple issue: sampling techniques [OlRo90] seem to be the best way to perform some kind of measurement.

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. Related Work

H. Gupta and I.S. Mumick in [GuMu99] use an A* heuristic to select the set of views that minimizes the total query-response time and also keeps the total maintenance time less than a certain value. A greedy heuristic is also presented in this work. Both algorithms are based on the theoretical framework developed in [Gupt97] using AND/OR view dags.

In [LaQA97] the same problem is considered for selection-join views with indexes. An A^* heuristic is also provided as well as rules of thumb, under a number of simplifying assumptions.

In [RoSS96], given a materialized SQL view, an exhaustive approach is presented as well as heuristics for selecting additional views that optimize the total view maintenance cost.

In [YaKL97], Yang, Karlapalem and Li propose heuristic approaches that provide a feasible solution based on merging individual optimal query plans.

Theodoratos and Sellis in [ThSe97,ThSe98] formulate the data warehouse design problem as a state space problem for SJ queries without space constrains. In [LiTS98] an experimental evaluations of algorithms based on that model is provided. In [ThLS99] the algorithms are extended for SPJ queries; also new heuristics are presented.

In [WaRK95] the authors propose an extension to the object-relational model, in order to annotate cell values with quality factors. The model is accompanied by the basic relational operators, customized for this particular case. Of course, no estimation of the expected quality is performed, but rather a calculation of all the annotations of all the quality factor values of the new relation.

In [NaLF99] the authors propose a scheme to integrate heterogeneous systems based on the quality factors that annotate each individual system. This model involves on-the-fly computation of query plans based on information for quality. The cost formulae are more generic than the ones that we employ and oriented towards the peculiarities of real time plan generation.

6. 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).

As future work, an interesting line of research is based on the fact 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 modeled

from SPJ queries. The investigation of the optimal DSA design problem by incorporating such operations in the problem formulation is, thus, another interesting challenge.

References

[GuMu99] 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

[Gupt97] H. Gupta: Selection of Views to Materialize in a Data Warehouse. In *Proc. of the 6th International Conference on Database Theory (ICDT)*, pages 98-112, Delphi, Greece, 1997

[JJQV99] 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). A previous version appeared in *Proc. 10th Conf. of Advanced Information Systems Engineering (CAiSE '98)*, Pisa, Italy (1998).

[JQJ98] 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.

[JV97] M. Jarke, Y. Vassiliou: Foundations of data warehouse quality – a review of the DWQ project. In *Proc. 2nd Intl. Conference Information Quality (IQ-97)*, Cambridge, Mass., 1997.

[LaQA97] 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

[LiTS98] 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

[LSTV99] 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).

[NaLF99] 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

[OB92] M. Oivo, V. Basili: Representing software engineering models: the TAME goal-oriented approach. *IEEE Transactions on Software Engineering*, 18, 10, 1992.

[OlRo90] Frank Olken, Doron Rotem: Random Sampling from Database Files: A Survey. 5th International Conference on Statistical and Scientific Database Management (SSDBM), Charlotte, NC, USA, April, 1990, pp. 92-111

[Orr98] K. Orr. Data quality and systems theory. *Communications of the ACM*, **41**(2): 66-71, (1998).

[OzVa99] M. T. Ozsu, P. Valduriez. Principles of Distributed Database Systems. *Prentice Hall*.

1999

[RoSS96] 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

[TaBa98] G.K. Tayi, D. P. Ballou: Examining Data Quality. *Communications of the ACM*, **41**(2): 54-57 (1998).

[ThLS99] 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).

[ThSe97] 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

[ThSe98] D. Theodoratos and T. Sellis: Data Warehouse Schema and Instance Design. *In Proc. of the 17th Intl. Conf. On Conceptual Modeling (ER'98), 1998.*

[Ullm88] J. Ullman. Principles of Database and Knowledge Base Systems, vol. I. *Computer Science Press*. 1988

[VBQ00] P. Vassiliadis, M. Bouzeghoub, C. Quix: Towards Quality-Oriented Data Warehouse Usage and Evolution. *Information Systems*, **25**(2): 89-115 (2000). A previous version appeared in *Proc. 11th Conf. of Advanced Information Systems Engineering (CAiSE '99)*, pp. 164-179, Heidelberg, Germany (1999).

[WaRK95] R.Y. Wang, M.P. Reddy, H.B. Kon. Towards quality data: an attribute-based approach. *Decision Support Systems,* **13** (1995).

[YaKL97] 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