D2I

Programma di ricerca (cofinanziato dal MURST, esercizio 2000)
Integrazione, Warehousing e Mining di sorgenti eterogenee

A survey of Logical and Physical Design Issues

MATTEO GOLFARELLI, STEFANO RIZZI

D2.R2 **31 marzo 2001**

Sommario

The bulk of the research on Data Warehousing is concentrated on logical and physical issues, since they mainly determine the system performance. As to relational Data Warehouse, the logical issues concern the definition of the relational structure of the DW and those optimizations that can be carried out by tuning such structures, while physical issues mainly deal with the index selection problem. The research carried out in the last ten years produced a lot of interesting results that are essential to reduce the query response time. The scope of this paper is to survey the main results in the literature as a base for the research in the ongoing project.

Tema	Tema 2: Progettazione e integrazione di Data Warehouse
Codice	D2.R2
Data	31 marzo 2001
Tipo di prodotto	Rapporto tecnico
Unità responsabile	BO
Unità coinvolte	BO
Autori	Matteo Golfarelli, Stefano Rizzi
Autore da contattare	Stefano Rizzi DEIS - Università di Bologna Viale Risorgimento, 2 40136 Bologna, Italia srizzi@deis.unibo.it

A Survey of Logical and Physical Design Issues in Data Warehousing

Matteo Golfarelli, Stefano Rizzi

DEIS - Università di Bologna
Viale Risorgimento, 2 - 40136 Bologna

Abstract. The bulk of the research on Data Warehousing is concentrated on logical and physical issues, since they mainly determine the system performance. As to relational Data Warehouse, the logical issues concern the definition of the relational structure of the DW and those optimizations that can be carried out by tuning such structures, while physical issues mainly deal with the index selection problem. The research carried out in the last ten years produced a lot of interesting results that are essential to reduce the query response time. The scope of this paper is to survey the main results in the literature as a base for the research in the ongoing project.

1 Introduction

In the last ten years the database community devoted increasing attention to the research themes concerning the development of decision-support systems, that usually lean on a central repository called *Data Warehouse* (DW). The enterprises, after having invested a lot of time and resources to build huge and complex information systems, ask for support in quickly obtaining summary information which may help managers in planning and decision-making. Data warehousing systems address this issue by enabling managers to acquire and integrate information from different sources and to query very large databases efficiently.

The topic of data warehousing encompasses application tools, architectures, information service and communication infrastructures to synthesize information useful for decision-making from distributed heterogeneous operational data sources. This information is brought together into a single repository suitable for direct querying and analysis and as a source for building logical *data marts* oriented to specific areas of the enterprise.

It is universally recognized that a DW leans on a multidimensional model where data are organized in *cubes*; each cell contains information useful for the decision process, while each axis represents a possible dimension for analysis.

From a functional point of view, the data warehouse process consists of three phases: extracting data from distributed operational sources, organizing and integrating data consistently into the DW, accessing the integrated data in an efficient and flexible fashion.

Building a DW for an enterprise is a huge and complex task, which requires an accurate planning aimed at devising satisfactory answers to organizational and architectural questions. There is substantial agreement on the fact that, when planning a DW, a bottom-up approach should be followed: the *data mart* playing the most strategic role for the enterprise should be identified and prototyped first in order to convince the final users of the potential benefits; other data marts are built progressively, to be finally integrated bottom-up into the global warehouse.

Designing the DW requires techniques completely different from those adopted for operational information systems since different system requirements must be fulfilled. A methodological framework for design is an essential requirement to ensure the success of complex projects. In [6], to this aim we developed a six steps methodology that starting from the operational data sources produce the final global DW schema.

Even if practical experiences have shown that the lack of a conceptual design step is one of the main reasons of DW projects failures [30], the bulk of the research on Data Warehousing has been concentrated on logical and physical issues since they mainly determine the system performance. The researches carried out in the last ten years produced a lot of interesting results that are essential to reduce the query response time. The scope of this paper is to survey the main results in the literature as a base for the research in the ongoing project.

Section 2 presents the framework we will refer to and introduces the choices and the assumptions we made. Section 3 surveys the literature concerning the logical issues: the main logical models and optimizations will be presented. Section 4 surveys physical issues focusing on the class of indexes used in relational DBMSs to address the DW specific characteristic. Finally, Section 6 discusses the most interesting open issues.

2 The global framework

It is universally recognized that a DW leans on a multidimensional model and thus it is often called multidimensional database. A *multidimensional database* can be modeled as a set of *multidimensional schemes* (MDS). The component of MDSs are facts, measures, dimensions and hierarchies. A *fact* is a focus of interest for the decision-making process; typically, it models an event occurring in the enterprise world (e.g., sales and shipments). *Measures* are continuously valued (typically numerical) attributes which describe the fact from different points of view; for instance, each sale is measured by its revenue. *Dimensions* are discrete attributes which determine the minimum granularity adopted to represent facts; typical dimensions for the sale fact are product, store and date. *Hierarchies* are made up of discrete *dimension attributes* linked by -to-one relationships, and determine how facts may be aggregated and selected significantly for the decision-making process. The dimension in which a hierarchy is rooted defines its finest aggregation granularity; the other dimension attributes define progressively coarser granularities. A hierarchy on the product dimension will probably include the dimension attributes product type, category, department, department manager.

Given a MDS we call *aggregation pattern*, or simply *pattern*, any subset of dimension attributes such that no attribute in the pattern functionally determines, even transitively, any other attribute in the pattern.

Each aggregation pattern determines a *view* storing the set of measures in the MDS aggregated according to the attributes in the pattern.

Logical and physical issues strictly depend on the way in which multidimensionality is implemented at the logical level. MOLAP (Multidimensional On Line Analytical Processing) solutions implement multi-dimensionality using multi-dimensional DBMSs; this is the most natural and direct representation of the data, it determines high-performance solutions but it is not widespread since it presents problems in terms of scalability and sparsity. In fact, within a cube of data usually less than 20% of the cells actually contain information [3], while the others correspond to events that did not take place. In multi-dimensional DBMSs all the cells are represented and consequently extra-space is required[1]. A further obstacle to the spreading of MOLAP solutions is the prejudice of designers and analysts in adopting unknown technologies. On the other hand, ROLAP (Relational On Line Analytical Processing) solutions are based on the well-known relational model that is a standard de facto in databases and does not present the sparsity problem since only the cells that actually contain information are stored. In the following we will refer to the relational solution that is the one adopted in the ongoing project. As to the reference architecture (see Figure 1), we will assume the presence of a reconciled data level that disjoins the problems concerning data extraction from the operational data sources from those concerning DW data organization and access. The reconciled level may be materialized or not depending on the size of the DW and on the complexity of the data stage process. Building the DW data level from the reconciled one requires further transformations; different approaches have been devised to accomplish them in a semi automatic fashion [5].

3 The Logical level

The logical design phase encompasses the definition of the relational structure of the DW and those optimizations that can be carried out tuning such structures. A key role in driving logical optimization, we argue, should be the workload of the system. Given that DW queries are extemporary by their nature, we can represent them conceptually as partial range queries where one or more dimensions of the cube are associated with a union of range of values [24]. The extreme case is where the size of the range is one for all dimensions yielding a *point query*. The range could be continuous (i.g "*time between Jan 94 and July 95*") or discontinuous (i.g. "*first month of each year*"). In the following we will refer to the class of queries sketched as On Line Analytical Processing Queries (OLAP queries). It should be noted that OLAP queries can retrieve data from two or more cubes that share the same dimensions, in this case we will refer to them as *drill-across queries*.

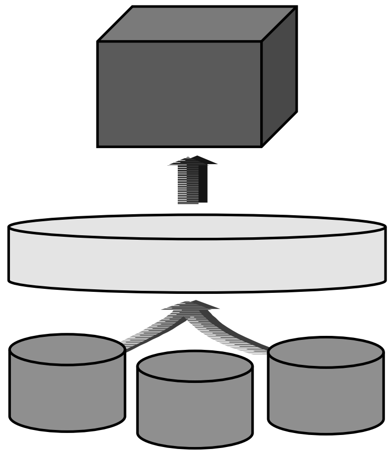

Figure 1: The reference data architecture

3.1 Logical models

ROLAP solutions implement multidimensionality adopting the so called *star schema* or its derived schemas. Basically a star schema is composed of:

- A set $DT_1,\dots,DT_n$ of relations, called *Dimension Tables*, one for each dimension, characterized by an identifier d_i that uniquely identifies each tuple and a set of attributes representing the dimension of analysis at different aggregation levels. The relations are completely denormalized and thus functional dependencies between the attributes.
- A relation *FT*, called *Fact Table*, whose identifier is given by the foreign keys $d_1,\dots,d_n$ from all the dimension tables; the relation *FT* usually contains a set of measures *V*.

Example 1 The star schema for the *Sales* data cube will be composed by the following relations:

```
SALES(StoreID,DateID,ProdID,Quantity Sold, Price)

STORE(StoreID,Name,City,State)

DATE(DateID,Date,Week,Month,Year)

PRODUCT(ProdID,Product,Type,Category,Brand)
```

Star schema rationale can be better understood considering the characteristics of OLAP queries that retrieve the measures stored in the fact table filtering data on the basis of dimensional attributes. For this class of queries the computational cost is mainly determined by the joins necessary to bind the fact table measures to the dimensional attributes. Star schema denormalization reduces the number of joins necessary to filter the data. Obviously denormalization requires more disk space for storing dimension tables but this does not heavily affect the database size since the dimension tables are usually much smaller than the fact

[1] Several techniques to avoid dummy cells storing have been developed but they negatively affect the system performance.

table. Analyzing the data at the finest aggregation level (*primary views*) may be useless and thus most of the queries work with aggregated data (*secondary views*). A technique commonly used to reduce the response time for frequent queries is to pre-compute and consolidate such data at one or more aggregation pattern. When aggregated data are consolidated the star schema may be inadequate: both aggregated and non-aggregated data should be stored in the same fact table whose size may become intractable[2]. In that case star schema derived models can be adopted. In the *constellation schema* aggregated data are stored in different fact tables called *materialized views*, while a unique dimension table is used for each dimension of analysis. Finally, in the *snowflake schema* also the dimension tables are split: the attributes whose aggregation level is coarser than the materialized view one are stored separately and a reference to these attributes is inserted in the original dimension table.

Example 2 The snowflake schema for the *Sales* data cube considering a materialized view on the aggregation pattern {`Month,Category`,`Store`} will contain the following tables:

```
SALES(StoreID,DateID,ProdID,Quantity Sold, Price)
SALES1(StoreID,MonthID,CategoryID,Quantity Sold, Price)
STORE(StoreID,Name,City,State)
DATE(DateID,Date,Week,MonthID)
MONTH(MonthID,Month,Year)
PRODUCT(ProdID,Type,CategoryID)
PRODUCT(CategoryD,Category,Brand)
```

It should be noted that by snowflaking we partially normalize the dimension tables; as a consequence the space required for storing the tables will decrease to the detriment of the queries that work on dimension attributes that have been spread through multiple tables.

3.2 View materialization

The view materialization problem is the search for the secondary views that minimize the workload response time under a disk space constraints[9][27]. View materialization has attracted the attention of several researchers due to its impact on performance and to its complexity, which derives from the size of the search space that grows exponentially in the number of dimensional attributes. In fact, given a multidimensional cube, any combination of dimensional attributes potentially defines an useful aggregation pattern. For example, in the absence of hierarchies, a multidimensional cube with n dimensions leads to 2^n possible views. In presence of hierarchies the number of valid aggregation levels is reduced by the presence of functional dependencies. A concise representation of the valid aggregation patterns is encoded in the

[2] Performance decay is more evident for aggregated data. In fact, even if aggregation strongly reduces the number of tuples, their retrieval is slowed down by the high number of non-aggregated data stored in the same table.

Multidimensional lattice (*MD-lattice*) [1] that has been widely used in view materialization related researches. An MD-lattice for a data cube is a lattice where the ordering relation is defined on the coarseness of aggregation pattern that can be determined considering functional dependencies (see Figure 2).

The MD-lattice size is still intractable for real applications, thus usually a set of user-relevant queries is used to yield a significant reduction of the search space. The patterns that are actually useful to reduce the workload response time can be determined iteratively. Starting from the set of patterns that optimally solve each single query the patterns that solve two or more queries are added until a fix point is reached [1]; the views determined are usually called *candidate views*.

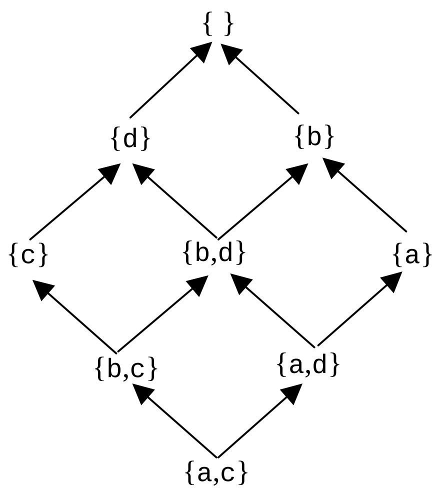

Figure 2: The MD-lattice for a simple MDS with two dimensions{a,c} whose hierarchies are { a$\rightarrow$b, c$\rightarrow$d }

Several heuristic algorithms have been proposed to determine the best subset of the candidate views that fits a disk space constraints [11][13][15]. The problem is NP-complete since it can be formalized as a knapsack problem with constraints. To the best of our knowledge, all these approaches consider each single lattice separately. On the other hand, we claim that the whole data mart should be involved in order to optimize also drill-across queries since they may weigh significantly upon the workload and cannot be optimized considering each single lattice separately. This result can be obtained substituting each drill-across query with two or more single-fact queries and optimizing all the involved lattices at the same time. Of course the problem will be now formalized as a multi-knapsack problem with constraints [23].

Besides the approach presented so far, other solutions that do not fit our framework have been presented in the literature [26]. From a wider point of view, views could be selected in order to satisfy different design goals like quality factors of the DW [28] or maintenance constraints [10].

Despite the large amount of researches that have been carried out, several problems are still open. In our opinion two main issues take the priority:

- View cardinality estimation: very few papers focus on estimating the cardinality of the views that, during the design phase, cannot be directly derived from the data [14]. A wrong estimate could invalidate the solution determined.
- Query expressiveness: OLAP queries expressiveness is limited if compared with the queries formulated by the users whose execution would require, for example, sequences of aggregate operators to be applied to a measure and multiple selections to be applied to aggregated data [7].

3.3 View fragmentation

Vertical fragmentation is aimed at optimizing both the queries requiring a subset of measures and the drill-across queries and can further enhance the effectiveness of the materialized views.

Each view usually includes several measures which describe the same fact but, within the workload, are seldom requested together. Thus, the system overall performance may be increased by *partitioning* the views to be materialized into smaller tables, each including only the measures which typically appear together within the queries. On the other hand, drill-across queries are formulated on multiple data cubes; as such, they involve measures taken from two or more views. The access costs for these queries may be decreased by *unifying* these views into larger tables where all the measures required are stored together.

By vertical fragmentation we mean the creation of fragments of views, each including measures taken from one or more views with the same pattern as well as the key associated to that pattern. Fragmentation may achieve two goals together: partitioning the measures of a view into two or more tables, and unifying two or more views into a single table. While partitioning may be useful whenever only a subset of the attributes is typically required by each query, unification may be convenient when the workload is significantly affected by drill-across queries.

It is remarkable that partitioning entails no significant storage overhead. In fact, on the one hand, surrogate keys require a few bytes to be stored. On the other, though on primary views the number of dimensions may exceed the number of measures, this is less likely on secondary views for two reasons. Firstly, it may be necessary to include in them also derived measures and support measures for non distributive aggregation operators [8]. Secondly, within a coarse aggregation pattern, one or more dimensions may be completely aggregated (in this case, the corresponding foreign key is dropped from the key of the view).

The view fragmentation problem is much more complex than view materialization since, for each view, several fragments could be potentially created. In [19] the authors define vertical fragmentation as a linear programming problem whose solution identify the set of fragments (defined on the base of a given workload) to be materialized.

3.4 Logical design

Logical design requires the logical structure of the DW to be determined. To the best of our knowledge, no works address the problem of selecting the best logical model from an algorithmic point of view, while several authors present tips and thumb rules that, together with the designer's experience, should drive the choice. For example, in [16] the authors discourage snowflaking since, in general, it interferes with the overall performance; thus should be adopted only for low cardinality attributes.

In [13] the authors show that a greedy algorithm that always materializes the most promising view next finds a solution that yields at least 63% of the optimal speed up. Their results are extended for the case where different probabilities for view usage exist. In [12], the problem of simultaneously choosing views and indexes, on both primary and secondary views, is investigated. The modeling is not much different from the previous one, since a view is defined with respect to its aggregation pattern; yet its selection condition is also taken into account. With respect to the previous one, here the cost model is much more complex.

4 The Physical level

Index selection and definition are among the main issues DW researchers deal with since indexes, together with materialized views, represent the most promising way to improve DW performance. Using indexes different from the well known B^+-Tree used in most of the operational DBMSs is worthwhile due to the DW process features. In particular, read-only accesses when the DW is on-line allow the use of indexes with an high update cost, furthermore the periodic refresh of the DW allows optimal indexes reorganization. While some indexes have been specifically developed for DW purposes, others were already well-known but though seldom used in operational applications.

4.1 Bitmap Index

Bitmap index, first developed for database use in the Model 204 project [20], is probably the most important result obtained in the DW physical optimization field. A bitmap index for a column c with values $c_1,c_2,\ldots c_k$ is a B^+-Tree whose entries are $c_1,c_2,\ldots c_k$ each associated to a data portion that contains a bit array for it. Thus, bitmaps are a new way to specify lists of RIDs for specific column values (see Figure 3).

The bitmap index is more suitable for low cardinality attributes since its size strictly depends on the number of distinct values of the column on which it is built. We say that bitmpas are dense if the portion of bits set is large. A bitmap index for a column with 32 values will have bitmaps with average density of 1/32. In this case the disk space needed for a RID-list index (which requires about 32 bits for each RID present) is equivalent to that used in the bitmap index. Besides disk saving, the main advantage of bitmap index is due to the fact that Boolean operations, such as AND, OR and NOT are extremely fast for bitmaps. This operations are very common in DW queries were multiple conditions need to be tested for filtering the interesting tuples.

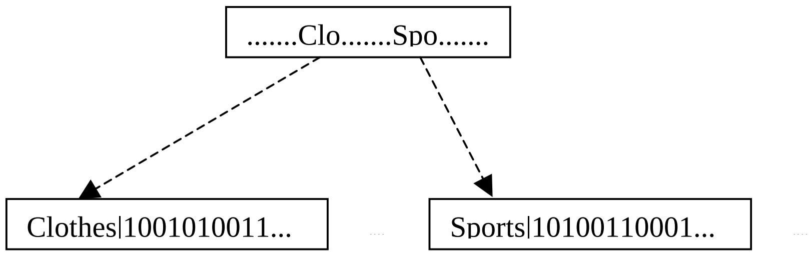

Figure 3: A bitmap index for attribute `Department`: the B+-Tree leafs contain the different values for the attribute and the associated bit vector whose bits are set if the corresponding tuple contains that value.

Bitmap related research mainly focused on:

- Compress bitmap [2] in order to allow sparse attributes to be efficiently indexed.
- Specialize bitmap index according to different query types, including range queries, aggregation queries and OLAP queries [21][31].
- Define algorithms for the index selection problem [17].

4.2 Join index

It is well known that joins strongly affect the computational cost of OLAP queries and consequently many efforts have been devoted in reducing such cost. Research goes in two directions: on the one hand, more effective implementations of join algorithms like hybrid hash join are devised [4][18]; on the other, devoted data structure are adopted. A join index [29] between two relations maintains pairs of identifiers of tuples that would match in case of join. A B+-Tree is sometimes built over the join index in order to avoid the index complete scan in presence of selections. The join index may be maintained by the database system, and updated when tuples are inserted and deleted in the underlying relations. While in systems exposed to continuous update the maintenance cost can be overcome by the savings achieved in performing the join, in DW applications such cost does not affect the query response time.

Typically, join indexes link the primary key of a dimension table with the corresponding foreign key in the fact table, speeding up all the queries that require to get data from both the fact table and that dimension table.

Starting from the idea of the join index, other multi-column indexes have been devised. In particular, *Star Join Index* extends join index to multi-table joins by concatenating column values from different dimensional tables and lists RIDs in the fact table for each concatenated values. Star join index allows multidimensionality to be explicitly represented. Although it will be very efficient when all of the indexed columns are accessed or when the leading columns of the index are required, the performance will be sub-optimal when other columns are accessed since the index structure is dependent on the ordering of the columns. Thus, the number of star join indexes needed to combine arbitrary column restrictions is a product of the number of column in each dimension determining a “combinatorial explosion” in terms of the number of independent columns.

4.3 Bitmapped Join Index

The low flexibility of the star join index can be overcome by the *Bitmapped Join Index* that, in its simplest form, can be defined as a bitmap index on a table *T* based on a single column of a table *S*, where *S* commonly joins with *T* in a specific way. For example, considering the *Sales* fact, the `Category` attribute belongs to the Product dimension table that must be joined with the Sales fact table to restrict the Sales rows to a range of categories. This can better be accomplished by creating an index for the values of `Category` on the Sales table (see Figure 4).

Sales-Category Bitmap Join Index

Sales RID	Cat-I	Cat-II	Cat-III	Cat-IV	Cat-V
1	1	0	0	0	0
2	0	1	0	0	0
3	1	0	0	0	0
4	0	0	1	0	0
5	0	0	0	1	0

Sales

	StoreID	DateID	ProdID	QTY Sold	Price
1	123	124	900	1500	700
2	246	450	666	2000	190
3	345	781	897	3900	1500
…	……	……	……	……	……

Figure 4: The join bitmap index on the left shows which tuples in the fact table correspond to products of category 'Cat-I'. No join is necessary to retrieve the Sales data.

Such solution does not change the design of the Sales table, since the index on `Category` is for virtual column through a join. The number of indexes of this type increases linearly with the number of useful columns in all dimension tables, since the power set of the column restrictions can be obtained “ANDing“ different arrays of bits.

Bitmapped join index, first proposed by O’Neil and Graefe [21], is now implemented in the principal commercial DBMSs and probably represents the most effective and flexible structure for reducing multi-tables join.

4.4 Physical design

Physical design includes the set of optimizations that, starting from the logical schema, determine the physical schema of the DW. Physical design requires the specific features of the selected DBMS to be known. The questions that should guide the designer are: “*Which indexes are implemented ?*”, “*How many indexes can be used at the same time ?*” and finally “*Which join algorithms are implemented ?*”.

The main issue related to physical design is the index selection problem. On the one hand, this problem is harder than in operational database due to the different class of indexes that can be adopted; on the other, the well-known characteristics of the logical models adopted simplify the problem.

Despite the high number of indexing techniques devised and the relevant number of query execution plans proposed [22], only a few works focus on the selection of indexes. In [17] the authors propose both an optimal algorithm and a set of thumb rules that should be adopted when the problem size is intractable. Rules, that are justified by considerations based on the adoption of appropriate cost functions, state that indexes should be created on keys, on attributes involved in joins and when their size fits the main memory.

5 Conclusions and open issues

The paper surveyed the main issues related to logical and physical design in relational DW. Even if the huge amount of research carried out leads to a strong performance improvement, many questions require further attention. In particular, we believe that different optimizations should be integrated in a true design methodology that also comprehends the conceptual design phase. An integrated methodology will allow coherent design solutions to be adopted during the different phases. Furthermore, relevant issues like the view cardinality estimation and the definition of physical cost functions have not received enough attention yet.

References

[1] E. Baralis, S. Paraboschi and E. Teniente. Materialized view selection in multidimensional database. In *Proc. 23rd Int. Conf. on Very Large Data Bases*, Athens, Greece, 1997.

[2] C. Chan and Y. Ioannidis. Bitmap Index Design and Evaluation. In *Proc. of ACM SIGMOD Conf*., 1998.

[3] G. Colliant. Olap, relational and multidimensional database systems. *SIGMOD Record*, 25(3), 1996.

[4] G. Graefe, A. Linville and L. Shapiro. Sort versus hash revisited. *IEEE Transactions on Knowledge and Data Enginnering*, 6(6), 1994.

[5] M. Golfarelli, D. Maio and S. Rizzi. The Dimensional Fact Model: a Conceptual Model for Data Warehouses. Invited Paper, *International Journal of Cooperative Information Systems*, vol. 7, n. 2&3, 1998.

[6] M. Golfarelli and S. Rizzi. Designing the data warehouse: key steps and crucial issues. *Journal of Computer Science and Information Management*, 2(3), 1999.

[7] M. Golfarelli and S. Rizzi. View Materialization for Nested GPSJ Queries. In *Proc. 2nd International Workshop on Design and Management of Data Warehouses (DMDW'2000)*, Stockholm, Sweden2000.

[8] J. Gray, A. Bosworth, A. Lyman and H. Pirahesh. Data-Cube: a relational aggregation operator generalizing group-by, cross-tab and sub-totals. *Technical Report* MSR-TR-95-22, Microsoft Research, 1995.

[9] A. Gupta, V. Harinarayan, and D. Quass, Aggregate query processing in data warehousing environments. In *Proc 21st Very Large Database Conf.* (*VLDB95*), Zurich, Switzerland, 1995.

[10] A. Gupta, H. Jagadish and I.S. Mumick. Data Integration using Self-Maintainable Views. In *Proc 5th Int. Conf Extending Database Tecnology*, 1996.

[11] A. Gupta and I.S. Mumick. Materialized Views. *MIT Press*. Cambridge Massachusetts, 1998.

[12] H. Gupta, V. Harinarayan and A. Rajaraman. Index selection for OLAP. In *Proc. Int. Conf. Data Engineering*. Binghamton, UK, 1997.

[13] H. Gupta and I.S. Mumick. Selection of Views to Materialize Under a Maintenance Cost Constraint. In *Proc Int. Conf.Database Theory* (*ICDT99*), Jerusalem, Israel, 1999.

[14] P.J. Haas, J.F. Naughton, S. Seshadri and A.N. Swami. Sampling-based estimation of the number of distinct values of an attribute. In *Proc 21st Very Large Database Conf.* (*VLDB95*), Zurich, Switzerland, 1995.

[15] V. Harinarayan, A. Rajaraman and J. Ullman. Implementing data cubes efficiently. In *Proc ACM SIGMOD*,Montral,Canada, 1996.

[16] R. Kimball. The data warehouse toolkit. *John Wiley & Sons*, 1996.

[17] W.J. Labio, D. Quass and B. Adelberg. Physical Database Design for Data Warehouses. In *Proc IEEE Proc. of 6th Int. Conf. on Data Engineering(ICDE97)*, Birmingham, U.K, 1997.

[18] Z. Li and K.A. Ross. Fast Joins Using Join Indices. *VLDB Journal*, 8(1), 1999.

[19] V. Maniezzo, A. Carbonaro, M. Golfarelli and S. Rizzi. An ANTS Algorithm for Optimizing the Materialization of Fragmented Views in Data Warehouses: Preliminary Results. To appear in *Proceedings First European Workshop On Evolutionary Computation In Combinatorial Optimization*, Como, 2001.

[20] P.O. O'Neil. Model 204 Architecture and Performance. In *Proc. 2nd Int. Workshop on High Performance Transactions System*. Asilomar CA, 1987.

[21] P.O. O'Neil and G. Graefe. Multi-Table Joins Through Bitmapped Join Indices. *SIGMOD Record*, 24(3), September 1995.

[22] P.O. O'Neil and D. Quass. Improved query performance with variant indexes. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*. Tucson, AZ, May 13-15, 1997.

[23] C.H. Papadimitriou and K. Steiglitz. Combinatorial optimization. *Prentice Hall*, Englewood Cliffs, 1982.

[24] S. Sarawagi. Indexing OLAP Data. *Data Engineering Bulletin*, 20(1), 1997.

[25] D. Theodoratos and T. Sellis. Data warehouse configuration. In *Proc. 23rd Int. Conf. on Very Large Data Bases*. Athens, Greece, 1997.

[26] D. Theodoratos and M. Bouzeghoub . A General Framework for the View Selection Problem for Data Warehouse Design and Evolution. In *Proc. ACM Third International Workshop on Data Warehousing and OLAP (DOLAP'00)*, Washington, 2000.

[27] D. Theodoratos and T. Sellis. Data Warehouse Configuration. In *Proc. 23rd Int. Conf. on Very Large Data Bases*. Athens, Greece, 1997.

[28] D. Theodoratos and M. Bouzeghoub . Data Currency Quality Satisfaction in the Design of a Data Warehouse. To appear on Int. Journal of Cooperative Information Systems, Special Issue on Design and Management of Data Warehouses, *World Scientific*.

[29] P. Valduriez. Join indices. *ACM Transactions on Database Systems*, 12(2):218-246, 1987.

[30] P. Vassiliadis. Gulliver in the land of data warehousing: practical experiences and observations of a researcher. In *Proc. of DMDW'2000*, June 2000.

[31] H.K.T. Wong H-F. Liu, F. Olken, D. Rotem and L. Wong. Bit Tranposed Files. In *Proc. Int. Conf. on Very Large Data Bases*. Stockholm, Sweden, 1985.