Research Issues in Data Warehousing

Ming-Chuan Wu and Alejandro P. Buchmann

DVS1, Fachbereich Informatik
Technische Hochschule Darmstadt
lastname@dvs1.informatik.th-darmstadt.de

Abstract. Data warehousing is a booming industry with many interesting research problems. The database research community has concentrated on only a few aspects. In this paper, We summarize the state of the art, suggest architectural extensions and identify research problems in the areas of warehouse modeling and design, data cleansing and loading, data refreshing and purging, metadata management, extensions to relational operators, alternative implementations of traditional relational operators, special index structures and query optimization with aggregates.

1 Introduction

Data warehouses (DWs) can be viewed as an evolution of management information systems [32]. According to a 1995 *META Group* survey, the DW market (including hardware, database systems and tools) is expected to expand from $2 billion in 1995 to $8 billion in 1998. This growth is reflected by the announcement of DW products from most major software companies in the past few years.
A DW is a repository of integrated information from operational (OLTP), and legacy systems that provides the data for analytical processing and decision making. The data in a DW is cleansed, temporal (historic), summarized, and non-volatile. Table 1 summarizes the main functions of operational databases (DBs) and DWs, and justifies the need for separate DWs.
In the DB research community, data warehousing and its problems have sparked recent discussion. However, industry and the research community are approaching the problems from different points of view. Industry appears mostly concerned with multidimensional modeling [44, 40, 49], multidimensional DBs [49, 2], On-Line Analytical Processing (OLAP) [13], Relational OLAP [40, 1, 19, 3, 50, 54], SQL extensions [20, 46], indexing [18, 42, 8]. The DB research community has concentrated heavily on view-maintenance problems [26, 21, 23, 24, 43, 48, 61, 62, 14, 31, 41, 47, 7, 34, 55, 10, 22, 27, 58, 57, 60]. Update-detection [35], view selection [25], cache management [51], query processing [15], query languages [45, 37], and a relatively small number of papers concerning the multidimensional model and OLAP [5, 37, 20, 53, 17] are distant seconds. Multidimensionality is gaining attention in the DB research community, especially, the question of extending the relational model with the CUBE operator [20, 5]. However, several questions arise: a) Where is the intersection of the work proposed by the research community and the requirements of industry? b) Are they

Aspects	**Operational Databases**	**Data Warehouses**
User	System Designer, System Administrator, Data Entry Clerk	Decision Maker, Knowledge Worker, Executives
Function	Daily Operations, (On-Line) Transaction Processing	Decision Support, (On-Line) Analytical Processing
DB Design	Application Oriented	Subject Oriented
Data	Current, Up-to-date Atomic, Relational (Normalized), Isolated	Historical, Summarized, Multidimensional, Integrated
Usage	Repetitive, Routine	Ad hoc
Access	Read/Write Simple Transaction (*usually*, involving 1-3 Tables)	Read mostly Complex Query (involving more Tables)
System Requirements	Transaction Throughput, Data Consistency	Query Throughput, Data Accuracy

Table 1. Differences between operational databases and data warehouses

solving the problem together? c) What technologies do we still need for data warehousing? To answer these questions we will examine *modeling issues*, *architectural issues*, *maintenance issues*, *operational issues* and *optimization issues.* Based on that analysis we can see where the proposed work can be placed in the problem space of data warehousing, and identify needed work.

2 Modeling Issues

The *relational model* and the *multidimensional model* are the main data models for data warehousing discussed in the literature. In Section 2.1, we focus on the essence of analytical processing, and try to clarify some misunderstandings about the relative merits of both models. In Section 2.2 we discuss modeling the DW using Select-Project-Join Views, an extreme case of using the relational model in the DW, and show some problems. In Section 2.3 we propose a three-level model for the DW.

2.1 Choosing a data model for the data warehouse

The essence of analytical processing

The *static aspects of analytical processing* include an *object for analysis* and a collection of *variables.* The object for analysis can be defined as a function (or mapping) of its corresponding variables, and each variable represents a dimension of the domain space. For example, if w is defined as a function f of x, y, z, denoted by $w = f(x, y, z)$, then the domain of the function f is the multidimensional space constructed by the three dimensions x, y and z. Suppose that w denotes sales, and let x be the products, y the regions, and z the time. Then, for a certain instance of (x_0, y_0, z_0), *i.e.*, for Product x_0, in Region y_0, at Time z_0, we have the sales w_0, denoted by $w_0 = f(x_0, y_0, z_0)$.

Along each dimension, *hierarchies* can be defined. Figure 1 shows two hierarchies defined on the dimension x. Suppose that the domain of the variable x is $\{1, \ldots, 12\}$, representing the months of a year. For the hierarchy in Figure 1(a), we define $x' \in \{\alpha, \beta, \gamma, \delta\}$ as quarters, $x'' \in \{\theta, \phi\}$ as half-years and $x''' \in \{\zeta\}$ as year, where $\alpha = [1, 2, 3]$ (the first quarter), $\beta = [4, 5, 6]$ (Q2), $\gamma = [7, 8, 9]$ (Q3), $\delta = [10, 11, 12]$ (Q4), $\theta = [1, \ldots, 6]$ (the first half-year), $\phi = [7, \ldots, 12]$ (the second half-year), and $\zeta = [1, \ldots, 12]$. For the hierarchy in Figure 1(b), we define $\hat{x} \in \{\mu, \nu\}$ as seasons and $\tilde{x} \in \{\xi\}$ as year, where $\mu = [1, 2, 3] \cup [10, 11, 12]$ (the winter season), $\nu = [4, \ldots, 9]$ (the summer season), and $\xi = [1, \ldots, 12]$.
Month, *quarter*, *half-year*, *year* and *month*, *season*, *year* form two hierarchies of the time dimension. *Month* is the hierarchy element representing the atomic data, while *year* is the most aggregated (summarized) hierarchy element.

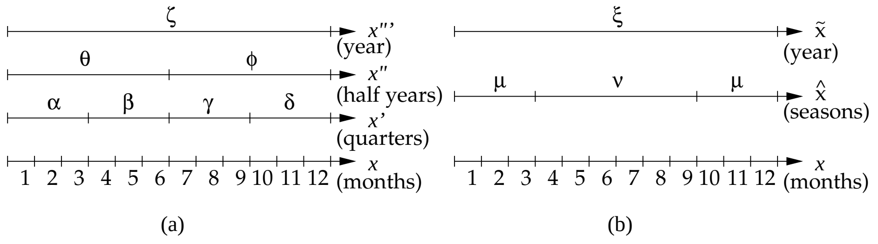

Fig. 1. Dimension hierarchies of x

The *dynamic aspects of analytical processing* consist of the analytical activities, such as forecasting, comparing, ranking, rolling up, drilling down, growth analysis, *etc.* The primitive operations for those activities include *fetching*, *comparing*, *sorting* and *data consolidation.*
Data consolidation is the basic operation for most of the analytical processing, including rolling up, ranking, forecasting and growth analysis. Data consolidation is the process of aggregating detailed data into single blocks of summarized information. If we want to aggregate w along the dimension x to the level x', it can be represented as

$$w' = F(x', y, z) = \sum_{x \in \{x'\}} f(x, y, z)$$

The above equation aggregates the monthly sales data to a quarterly sales summary. It is also possible to consolidate data along multiple dimensions simultaneously. This is referred to as *multidimensional analysis.* In Figure 2, we depict the aggregation discussed above for part of the time dimension. In Figure 2(a), $w = f(x)$, while in Figure 2(b), $w' = F(x') = \sum_{x \in \{x'\}} f(x)$, where $F(\alpha) = f(1) + f(2) + f(3)$, $F(\beta) = f(4) + f(5) + f(6)$, $F(\gamma) = f(7) + f(8) + f(9)$ and $F(\delta) = f(10) + f(11) + f(12)$.

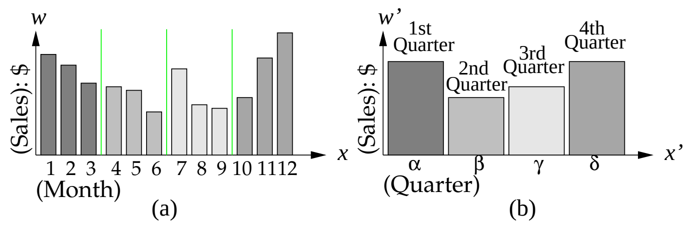

Fig. 2. (a) $w = f(x)$, (b) $w' = F(x') = \sum_{x \in \{x'\}} f(x)$

Misunderstandings about multidimensional and relational models
Many authors argue about the advantages and disadvantages of multidimensional and relational OLAP. A multidimensional OLAP system consists of a multidimensional DW with the OLAP tools directly built on the multidimensional DW, while the relational OLAP system consists of a relational DW and an OLAP engine with multidimensionality [50, 1, 3, 13, 40, 2, 33, 49, 19, 54]. Misunderstandings are found in some of the articles. For example, [44] states: "*Entity-Relationship modeling is the heart of the relational model. The explicit relationships between customers and sales orders, or between hamburgers and buns are burned into the design of the database.*"

Many misunderstandings are caused by not separating carefully the conceptual and the logical design. The data model used in the conceptual design phase need not necessarily be the data model chosen in the logical design phase. If we choose the multidimensional model for conceptual modeling, it does not imply that the DW will be built on a multidimensional DB system. In fact, we propose to use the multidimensional model for the conceptual design of a DW, and choose a relational DB server as the data store.

The DW prepares the data for analytical processing which requires multidimensionality. Therefore, the model for DW modeling should be *capable* and *suitable* of expressing the multidimensionality. The two basic data constructs provided by the multidimensional model are *facts* and *dimensions*. These two constructs let the analysts view the data in the way they perform the analysis work. In Figure 3(a) shows a simple star schema that resulted from multidimensional modeling. Facts correspond to sales data, and "products", "geographical regions" and "time" are dimensions of sales. Analysis on sales is performed along any (or all) of the three dimensions, *e.g.*, 'analyze the sales growth of *Product A* and *B* in *Region West* in the past two years'. As shown in Figure 3(b), multidimen-

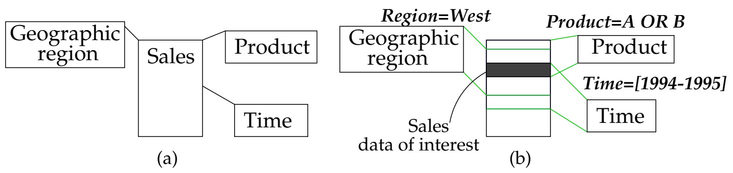

Fig. 3. A simple star schema of the multidimensional model

sional data modeling provides users with a clear and natural view of data. Data can easily be accessed by manipulating the dimensions, something that is usually done in analytical processing. The multidimensional model meets the conceptual modeling needs of a DW.
However, this does not imply the exclusion of other data models, such as the ER model and the relational model. In Table 2, we show the mapping among the constructs provided by the ER model, the multidimensional model and the relational model. For the sake of simplicity attributes are omitted.

ER Model	**Multidimensional Model**	**Relational Model**
Relationship	Fact	Relation
Entity	Dimension	

Table 2. Mapping among Data Models

Though the mapping among the models is possible, not all models are equally suited. Taking the ER model as an example, the ER model is not good for modeling analytical processing, because the model emphasizes *first* identifying entities, and *then* defining their interrelationships, which map to dimensions and facts in the multidimensional model, respectively. Reversely, in analytical processing we usually define our objects for analysis first, then their dimensions. In the relational model the only data construct provided is the relation, which is defined as a subset of the Cartesian product of the domains of the attributes that define the relation, denoted as $r(R) \subseteq (\text{dom}(A_1) \times \ldots \times \text{dom}(A_n))$. From the formal definition of a relation, we can see that the relational model is originally quite good in nature to model the multidimensionality. For instance, for each A_i, it can represent a dimension of the relation R. Even so, it will not be a good candidate in the conceptual design phase for modeling a DW, since the relational model does not model the data in the way that human analysts usually view their data, *i.e.*, an analysis object and its corresponding dimensions. Nevertheless, the relational model will be a good candidate for logical design of a DW, because of its inherent multidimensionality and its solid mathematical foundation for query processing.

2.2 Modeling data warehouses using SPJ-views

Modeling the DW using SPJ-Views (Select-Project-Join) is probably the quickest way to construct a DW[1]. The views are defined using SQL commands and materialized in the DW for direct access. They are usually summary data from the operational DB systems and are provided for periodical reports.
The primary advantages of using materialized views for modeling the DW are:
1. *Ease of definition and flexibility to change.* Based upon the aggregation requirements of the DW users, SPJ-Views can be easily defined and redefined

[1] Here we assume that an integrated relational schema is available, and the SPJ-views are defined on it, since we discuss in this section the suitability of a data model for modeling multidimensionality, not for schema integration.

according to changes in users' requirements.
2. *Improved availability.* Materialized views provide access to the data regardless of the availability of the data sources.
3. *Provision of Cumulative and Historical Data.* Views with aggregates provide summarized and historical data from sources which are no longer available.
The disadvantages, however, are
1. *Lack of Multidimensionality.* The SPJ-View, as its name says, is a projection of some selected tuples which are possibly the result of joining tables from different sources. A materialized view can be referred to as a table. A single table is not ideal to model the multidimensional essence of analytical processing. Even if we define a universal relation by joining all tables, the view presented to the DW users is a mass of data containing duplicates, which consume not only storage space but also time for maintenance.
2. *Space and Time Required for Maintaining the Common Subset of Views.* Because of the lack of multidimensionality, if we still want to use views to provide different *slices* of the data cube, *i.e.*, bounding all the dimensions and letting only two of them free, the number of views will grow dramatically. For example, for a three-dimensional data cube, there will be $C_2^3 = 3$ different kinds of slicing, and for each kind of slicing, the number of slices depends on the cardinality of the bounded dimension, as Figure 4 shows, there are all together (n+m+p) slices.

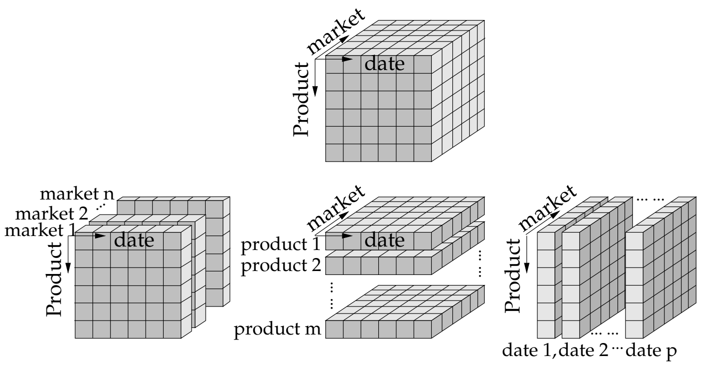

Fig. 4. Different slices of a 3-dimensional data cube

Furthermore, if we want to store the views including some, if not all, of the *rotations* or the *roll-ups/drill-downs* of the data cube, the DW will grow so rapidly that it will be soon overloaded with maintenance work. Also, the operational systems will be affected, since the view maintenance components keep sending queries to evaluate the incremental changes to the views with respect to any change in the source sites. In addition, most views will have large common subsets of data. The storage space required to store the views might be far more than simply replicating all the operational DBs.
From the above discussion, the disadvantages of using SPJ-Views to model the DW seem to outweigh the advantages.

2.3 Three-level modeling

The three-level modeling comprises *conceptual modeling*, *logical modeling* and *physical modeling*, and corresponds to the three accepted phases of database design. We propose to use the multidimensional data model for the conceptual modeling, and the relational model for the logical and physical modeling (due to its solid mathematical foundation for query processing).
In Section 3, we present a logical warehouse architecture corresponding to the three levels of schemas, and in Section 5 we will discuss the extensions to relational technology needed to make relational DB systems suitable for the data warehousing environment.

3 Data Warehouse Architectures

In this section we discuss the architectural issues of data warehousing. In Section 3.1, we propose a logical architecture of a DW system based on [40]. In Section 3.2, we map some proposed physical architectures onto this logical architecture, followed by a discussion of work to be done.

3.1 The logical architecture

The architecture of any warehouse depends on the following four factors: the volume of goods, the characteristics of the goods, the arrangement of the goods and the way of retailing. The architecture of a DW similarly depends on the volume, the characteristics, the arrangement, and the usage of the data.
Volume of data A survey by the META Group, Inc. in 1996 shows that more than 70% of the organizations plan to implement DWs larger than 50 GB, and roughly 50% are planning DWs of over 100 GB. Only about 13% of the organizations surveyed will have DWs smaller than 10 GB. Consequently, we need a *Data Management Layer* which supplies services for efficient storage, retrieval and management of large amounts of data.
Characteristics of data To support properly a DW application a variety of specialized storage structures are required. Therefore, we extract the functionality of storing data from the Data Management Layer, and identify it as a separate layer — the *Data Store Layer*. Providing the Data Store Layer, the implementation details of the physical storage structures will be hidden from the users of the Data Management Layer and will allow the use of special purpose storage solutions, whereas in [40] a Database Layer was defined covering the functionalities of both the Data Management Layer and the Data Store Layer.
Arrangement of data The arrangement of data, like that of any other product, depends on how it will be retailed. For sugar it might mean whether 1 Kg sacks should be packed 12 to a box or 24 to a box. Similarly, how data will be aggregated cannot be answered till we know how they will be retailed. Therefore, in the data warehousing architecture, an *Application Interface Layer* is defined, which provides services to conceptually arrange data in the way needed by the

applications. The services for the arrangement of the data are application dependent. If requirements of the applications change, only the Application Interface Layer needs to be changed, leaving the underlying lower layers unchanged.

Usage of data The usage of the data by the decision makers will occur either through special-purpose applications or generic OLAP front-end tools. They form another layer, the *Presentation Layer*.

The system diagram of the logical architecture is shown in Figure 5.

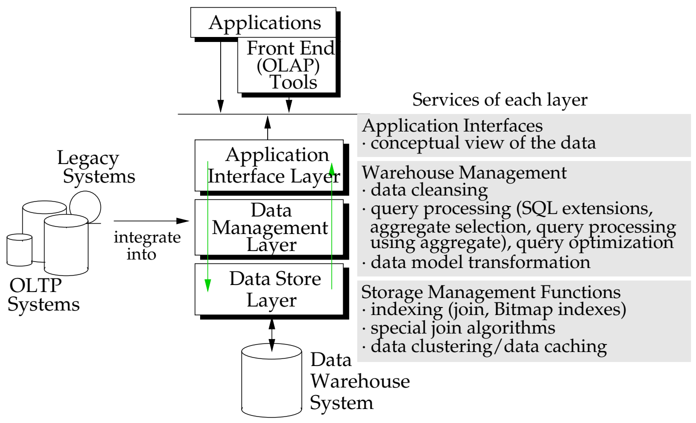
Applications
Front End (OLAP) Tools
Services of each layer
Application Interfaces
· conceptual view of the data
Legacy Systems
Application Interface Layer
Warehouse Management
· data cleansing
· query processing (SQL extensions, aggregate selection, query processing using aggregate), query optimization
· data model transformation
Data Management Layer
integrate into
OLTP Systems
Data Store Layer
Storage Management Functions
· indexing (join, Bitmap indexes)
· special join algorithms
· data clustering/data caching
Data Warehouse System

Fig. 5. The logical architecture

Each layer provides services for the next higher layer, or for the intra-layer housekeeping process. The Data Store Layer provides the Data Management Layer the services for storing the data, building indexes (Bitmap indexes, or special join indexes), data clustering, *etc.* The Data Management Layer, in turn, provides services for higher level management of the warehouse data, *e.g.*, load utilities, data model transformation between external sources and the logical schema, data cleansing, query processing, query optimization, *etc.* Next, the Application Interface Layer provides data access facilities suitable for specific applications, including data model transformation between the conceptual multidimensional schema and the logical schema. It also contains utilities to generate extracts that are frequently offloaded to desktop resident OLAP tools. The Presentation Layer includes graphical presentation and reporting tools. It typically runs on a desktop environment whereas the other three layers typically exist on the server side. The presentation layer therefore also includes the desktop-resident processes needed for extract generation.

This architecture is an extension of that proposed by [40]. It breaks out the Data Store Layer and redefines the functionality of the layers. Though we define the DW as a data repository for analytical processing, we would like to keep the logical architecture independent from applications and front-end tools. Therefore, the Presentation Layer is not further defined.

3.2 The physical architectures

The logical architecture must be mapped to a physical architecture. We introduce some proposed physical DW architectures, and discuss how they map onto the logical architecture. In doing so we will concentrate because of space limitations on the architecture of the DW, *i.e.*, the back-end side.

Data warehouse architecture for multidimensional OLAP systems The multidimensional OLAP systems utilize a multidimensional DB to store the warehouse data and analytical applications are built directly on top of it, as shown in Figure 6. In such an architecture, the multidimensional DB system serves as both the Data Store Layer and the Data Management Layer. Data from sources are conformed to the multidimensional model, and summaries along all dimensions are precomputed for performance. Since the data is stored multidimensionally, data accesses are directly submitted to the multidimensional DB systems. The main problem of this architecture is the lack of scalability [40].

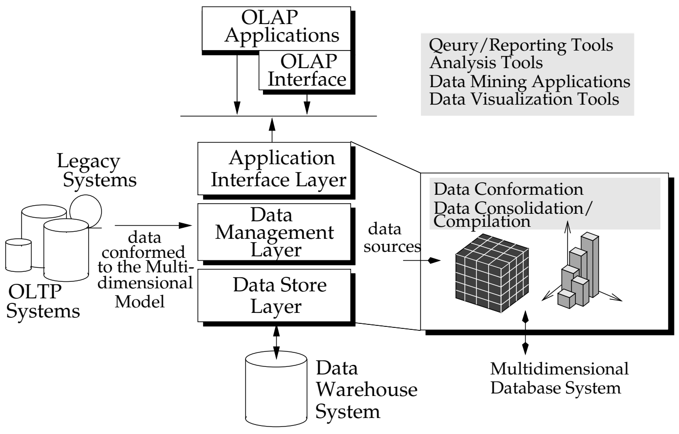

Fig. 6. The warehouse architecture for the multidimensional OLAP system

Data warehouse architecture for relational OLAP systems The Relational OLAP systems use a relational DB as the DW. Both the Data Store Layer and the Data Management Layer are relational, however, extensions to the relational model are required to support the multidimensional analysis requests from the Application Interface Layer. The relational OLAP engine, at the Application Interface Layer, views the underlying data multidimensionally, and provides multidimensional operations for its users. The architecture maps onto the logical architecture as shown in Figure 7.

In a DW environment, the access patterns, operations, data organization, performance criteria, metadata management, transaction management and query processing are very different from operational DB systems. Therefore, relational DBMSs tuned for the operational environment, cannot be directly transplanted into a DW system. For example, since most queries are read-only, the complex concurrency control mechanism used in the operational DBs is an overkill.

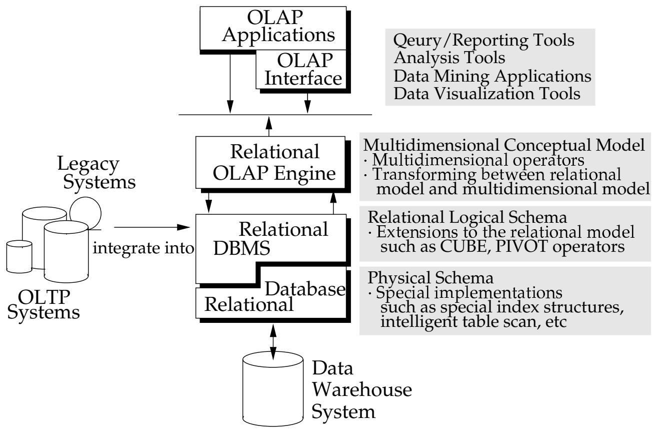
OLAP Applications
OLAP Interface
Qeury/Reporting Tools
Analysis Tools
Data Mining Applications
Data Visualization Tools
Legacy Systems
Relational OLAP Engine
Multidimensional Conceptual Model
· Multidimensional operators
· Transforming between relational model and multidimensional model
Relational DBMS
integrate into
Relational Logical Schema
· Extensions to the relational model such as CUBE, PIVOT operators
OLTP Systems
Database Relational
Physical Schema
· Special implementations such as special index structures, intelligent table scan, etc
Data Warehouse System

Fig. 7. The warehouse architecture for the relational OLAP system

Transaction management in the DW will relax some of the ACID properties, *e.g.*, large table scans. In analytical applications which involve gigabytes of data, some extent of inconsistency might be tolerated. Because of the large data volume, we need special implementations of the traditional operators (*e.g.*, join, parallel scan) or special data structures (*e.g.*, Bitmap indexes) to support the complex operations required by OLAP.

Data marts Data marts are proposed for small data volumes (< 10 GB), or data with less than 10 dimensions using multidimensional DBs. Data marts usually serve as departmental OLAP systems, focusing on selected subjects and dimensions, while DWs serve as enterprise OLAP systems spanning the entire organization.

Virtual data warehouse A virtual DW is a collection of views, which are usually summaries of the atomic data, and serves for efficient query processing. It is easy to build, but does not support much multidimensionality.

4 Data Warehouse Maintenance Issues

DW maintenance issues include data cleansing, initial loading, subsequent loading (refreshing), incremental loading, data purging and metadata management.

4.1 Data extraction, cleansing and initial loading

Data extraction is both a political and a technical problem, because it requires opening up existing systems with *structural* and *semantical* discrepancies. Without first resolving these discrepancies, the derived information will be misleading. In essence, methodologies proposed for schema integration and multidatabase systems can be applied to cleanse the data (for references see *e.g.*,[4]). The problem is complicated by the fact that multiple, often incompatible, aggregation hierarchies are used both in the target and in the source systems.

Once the data is cleansed, it can be loaded into the DW. Data loading includes scanning, filtering, sorting, partitioning, aggregating, indexing, integrity checking. Data volume plays a critical role in this problem. Loading 1 TB may take more than three months of time. In [6], a parallel loading approach using dataflow parallelism was proposed to speed up the loading process. Besides, partitioning and aggregating will affect query performance later. The criteria and evaluation model for partitioning and aggregating data still need to be defined.

4.2 Data refreshing and data purging

Data refreshing After the initial loading, the updates at source DBs should be propagated to the DW. This propagation process is called data refreshing. When a refreshing mechanism is designed, the following issues should be considered: *consistency requirements*, *refreshing timing*, *the modes of refreshing*, and *refreshing techniques*.
Four levels of DW consistency were defined in [62]: *convergence*, *weak consistency*, *strong consistency* and *completeness*. According to the requirements of the application, corresponding consistency criteria can be chosen for the DW.
As to the timing of refreshing, it may occur after every single update at any source, or it can be done periodically, *e.g.*, daily, or monthly. Refreshing can also be performed according to the needs of the end users, or after the occurrence of some significant event. Obviously, "when to refresh" is a trade-off between consistency and time, the size of the window of opportunity, the slack capacity of the OLTP systems that may be impaired by the propagation of data, *etc.*
The modes of refreshing are on-line or off-line refreshing. On-line refreshing allows the applications of the DW to continue while refreshing is performed. However, integrity of the data should be guaranteed. If an update is propagated from one source DB to a table at the DW, and aggregates have been built on this table, the refreshing process should include updating both the table and its aggregates in *one* ACID transaction, otherwise any user query that interleaves with the refreshing process and performs drill-downs/roll-ups on the table and its aggregates, might discover the inconsistency. As mentioned earlier, application-dependent correctness criteria (*e.g.*, the significance of inconsistency) might be needed to take into account As for off-line refreshing, all applications should be stopped, when the refreshing is performed.
Refreshing techniques fall into two categories: *recomputation* and *incremental techniques*. Examples of recomputations are full extraction from base tables, or re-evaluation of the view definition against the base relations. Recomputation is not very cost-effective, especially, if the source data is less volatile in nature. However, for legacy systems or flat files, it might be the only choice. Incremental techniques include *snapshot refresh*, *transaction shipping*, *incremental view maintenance*, *detecting irrelevant updates*, *self-maintainability*. Snapshot refreshing uses triggers to update the *snapshot log*, which will be used later to update the *snapshot table*. In the transaction shipping approach, a *log transfer manager* runs at the source site, and uses the replication definition and the DB log of the source site to detect the relevant updates. If any relevant update is detected, the

log transfer manager transfers the log records to the replication server, which generates SQL updates according to the log records, and executes the updates against the replicated data. As for materialized views, incremental view maintenance algorithms use the view definition to generate view update queries, which will be sent to the sources, and answers of the queries will be applied to the views. Self-maintainability describes the property that a materialized view can be maintained without referring to source data other than the updated tuple, the view definition and the view itself.

Table 3 shows the relationship between the grade of consistency, refreshing timing and refreshing modes. For example, for *off-line*, and *continuous refreshing* (refresh after every update), the highest level of consistency, which can be achieved, is *Completeness*; and for *off-line*, and *periodical refreshing*, the highest reachable level of consistency is *Strong Consistency.* On the other hand, for *on-line* refreshing, the level of consistency can not be determined, since it still depends on other factors, such as the *activeness* of the sources, self-maintainability of the warehouse data, *etc.* Table 3 is not complete. It can be expanded to include more factors, and can then serve as a guideline for determining what level of consistency can be achieved by using a certain refreshing mode, strategy or technique.

Continuous (after every update)	On-Line	undetermined
	Off-Line	Completeness
Periodically	On-Line	undetermined
	Off-Line	Strong Consistency

Table 3. Levels of consistency achieved by different refreshing strategies

Data purging Another maintenance issue of the DW is the purging of the old data, which is rarely mentioned in the literature. The definition of the *old data* depends on regulations of the organization and the requirements of the applications. We propose the following purging policies.

– Cleaning-Up. Old data is 100% removed from the DW, including atomic and aggregated data, but the metadata about the old data remains in the DW. If we want to refer to the purged data, the system will be able to tell where the operational data resides. The re-load of the data could be manual or automatic.

– Selective Purging. Old data is selectively purged as a function of *space*, *access frequency* and *performance requirements.* Figure 8 illustrates this. The data cube

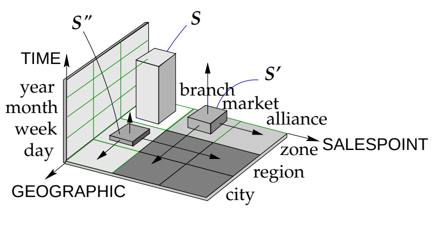

Fig. 8. Aggregates in the multidimensional space

S stores the most detailed data and can be used to derive all other aggregates along all dimension hierarchies. Therefore, we can purge all the aggregates. However, the data cube S is very space-consuming, while the aggregates S' and S'' are relatively small in size. If the applications will not do any sales analysis on the old data along the "zone" hierarchy element of the geographic dimension, we can remove S and S' from the DW, and leave S'' in the DW. Moreover, even if we *sometime* need to access the old sales data at the "zone" level, it might be much cheaper to leave both the S' and S'' in the DW and purge S than to save S and purge S' and S''. A mathematical model is needed to evaluate the selection.

– Archiving. This policy stores old data on other storage devices, *e.g.*, tapes, or other disks, before purging them. Basically, only the most detailed data is archived, since aggregates can be derived from them. This policy is used when the operational DB systems or the legacy systems are not able to provide historical data any more and the historical data will be accessed sporadically.

– Selective Purging and Archiving. This policy is the combination of Purging and Archiving. In the cases that the operational systems do not provide historical data and the access frequency to the old data is relatively high, we could archive the detailed data onto a tape before purging it and selectively leave some aggregates in the DW.

Combinations of these policies could be used. What is needed are clear criteria for data removal from the DW backed up by the corresponding cost models.

4.3 Metadata management

Metadata management is another important issue in the warehouse maintenance. Since data is integrated from different sources, both the schematic mapping among different sources and the semantic mapping among different data should be maintained. In addition, since historical data might also be kept in the DW, version control information for the metadata is required. Based on the classification of the metadata in [9], we divide the metadata into three categories: *administrative metadata*, *application specific metadata* and *auditing metadata*.

The administrative metadata includes:

– *Schematic metadata* that consists of the warehouse schema, the schema of the source DBs and their mappings to the warehouse schema. The warehouse schema, in turn, consists of the logical schema, the physical schema (table partitioning information, indexing structures, *etc*), data extracting, cleansing and transformation rules, data refresh conditions and the mapping between the logical and physical schema.

– *Semantical metadata* that consists of the semantic mapping among similar data from different sources. The similar data items with mismatched semantics are comparable, only if the mapping functions are defined. For example, suppose that sales data from two different operational DBs is integrated into the DW. However, one operational DB stores sales data including the value-added tax, while the other DB stores sales amounts without it.

– *User profiles* that contain all the users' information, authorization, access control, and user groups.
– *Conceptual metadata* that contains the conceptual schema, which is the high level schema for the DW users instead of DW administrators, dimensions, dimension hierarchies, aggregates, and the mapping to/from the logical schema.
The application specific metadata is composed of a set of common terminologies specific to the application domain, application constraints and other policies.
The auditing metadata includes data lineage, warehouse statistics (such as value distribution of a dimension), audit trails and error reports. The handling of metadata is far from perfect in today's DW products and requires a comprehensive approach. The Metadata Coalition is chartered to address these issues [39].

5 Operational and Optimization Issues

In Section 3, we discussed using a RDBMS as the data store and the data management components of a DW system. However, the operations provided by relational systems do not efficiently fulfill the needs of the applications. There are two complementary approaches to extend a relational system. One is to add new operators, *e.g.*, CUBE, ROLLUP, extensions to GROUP BY. The other is to reimplement some traditional relational operators. We discuss these approaches in Sections 5.1 and 5.2, respectively. Optimization is discussed in Section 5.3.

5.1 Extending relational operators

ROLLUP, CUBE and Cross-Tab
In analytical processing, it is essential that detailed data and summaries are presented in cross-tabulated form (cross-tab, pivot table), as shown in Table 4. However, to generate a common two dimensional cross-tab, requires three aggregation functions, three GROUP BY operators and two UNION operators in standard SQL.

BIKE	**1994**	**1995**	**Total**
black	600	720	1320
white	550	610	1160
Total	1150	1330	2480

(a)

BIKE		**1994**	**1995**	**Total**
Mountain Bike	black	450	550	**1000**
	white	250	360	**610**
City Bike	black	150	170	**320**
	white	300	250	**550**
SubTotal	black	600	720	**1320**
	white	550	610	**1160**
Total		**1150**	**1330**	**2480**

(b)

Table 4. Sales Summary (a) two dimensional cross-tab, (b) three dimensional cross-tab

The structure of the result in Table 4 does not conform to commonly accepted relations.[2] Using standard SQL to express a multidimensional cross-tab would be a nightmare for DW users. To let a user easily express a roll-up or a cross-tab query, the following extension to the SQL GROUP BY was proposed in [20].

```
GROUP BY { ( <column name> | <expression> ) [ AS <correlation name> ]
           [ <collate clause> ], ... } [ WITH ( CUBE | ROLLUP ) ]
```

Using this SQL-extension to generate Table 4(a) results in

```
SELECT Product_type, Year, Color, SUM(Sales)
FROM Sales
WHERE Product_type = "BIKE" AND Year BETWEEN 1994 AND 1995
GROUP BY Product_type, Year, Color WITH CUBE;
```

This leads to further questions. First, how to transform the relational results by CUBE or ROLLUP operators to the multidimensional forms? In the architecture we proposed, this task should be performed by the Application Interface Layer. Second, how to efficiently implement the CUBE and the ROLLUP operators? As the number of dimensions increases, the work of computation explodes exponentially. Some work has been proposed in [5, 20], but many open questions remains. Third, if some levels of summaries are available, how does the CUBE operator take advantage of them?

Histograms and other extensions

A common operation in analytical processing is to aggregate along an aggregation hierarchy. Usually, we model the dimension and its hierarchies into the DW. However, some dimension hierarchies would be better expressed as functions, *e.g.*, the time dimension. Suppose that in a sales table, time of each trade is recorded. Time is modeled as a long integer and stored as offset in seconds from 00:00:00 January 1900. Assume that we perform sales analysis on a daily, weekly, monthly and quarterly basis, and we also analyze the sales distribution between the rush-hours and the lunch-hours. Then, we have to model two hierarchies on the time dimension in our DW. One is $time \rightarrow hour \rightarrow day \rightarrow week$, and the other is $time \rightarrow hour \rightarrow day \rightarrow month \rightarrow quarter$. Grouping data into a higher hierarchy element involves a join operator, followed by a GROUP BY operator.
On the other hand, if we model the time dimension as functions, the grouping could be done by a single table scan and a GROUP BY operator. However, standard SQL does not support functions (the construction of histograms) in the GROUP BY clause. If functions were allowed in the GROUP BY clause, we could simply say:

```
SELECT day, SUM(Sales)
FROM Sales
GROUP BY Day(Time) AS day;
```

[2] However, it is the common format for decision makers. The data in such a cross-tab fit well into the multidimensional model. The figures in the kernel of the table are the *facts* in the multidimensional model, and the upper-most row and the left-most column are the *dimensions*, *i.e.*, the product name, the color and the years. The right-most column and the last few rows are the summaries generated by the operators CUBE or ROLLUP.

For details on proposed extensions see [20]. Other useful extensions in Red Brick Intelligent SQL (RISQL) are: *Rank*, *Tertile*, *RatioToReport*, *Cume*, $MovingSum(n)$ and $MovingAvg(n)$, where n is an integer. For a detailed description of these operators see [46].
Similar to CUBE and ROLLUP, we could define many new aggregation functions to extend SQL. This, however, is only syntactic sugar. Much work needs to be done in devising efficient implementations for the new operators.

5.2 Reimplementation of the relational technology

Traditionally, indexes map attribute values onto tuples of a table. Two separate indexes built on two attributes of the same table do not work together to identify the tuple set which corresponds to the combination of the two indexes. Instead, the result sets must be combined or a compound index must be built. Besides, the order of the attributes in a compound index matters. For example, if we want to build compound indexes on the three attributes, {A, B, C}, it results in $3! = 6$ different permutations. Indexes improve the query processing performance, but, building and maintaining them is time consuming.
For data warehousing, some special indexing techniques, such as Bitmap index, join index and Bitmap join index, have been rediscovered or introduced. We will briefly discuss them to point out areas of further research.
Bitmap index The Bitmap index uses a bitmap vector to represent the membership of the tuples in a table, *i.e.*, whether or not the tuples have an attribute with the same value. The table R in Figure 9 has an attribute Gender, with domain {F, M}. We can use the bitmap vector B^F to represent those tuples, which belong to the group "female". If $B^F[i] = 1$, then the i-th tuple has Gender=F, otherwise it is 0. To be complete we have to build another bitmap vector B^M. If $B^F[i] = B^M[i] = 0$, the tuple does not belong to either group.

Table: R

...	Gender	...
	M	
	F	
	M	
	F	
	F	
	M	

B^F	B^M
0	1
1	0
0	1
1	0
1	0
0	1

Fig. 9. Bitmap indexing

Bitmap indexes are suitable for narrow domains, especially for membership functions with 0, or 1 as function values.[3] Bitwise operations, such as AND and OR, can be applied on several bitmap vectors to retrieve the tuples corresponding to multiple indexes at one time. This is especially good for StarJoins, which will be discussed later.

[3] For attributes with higher cardinality, bitmap indexes can be selectively build on the most frequently used attribute values. Alternatively, domain transformation or subdomain indexing can be applied.

A research problem, which to the best of our knowledge has not yet been properly addressed, is using the bitmap matrix from different Bitmap indexes to cluster data, or partition the fact table. The fact tables are usually very large in size. If we could partition them according to access pattern into several smaller tables, we could avoid a large table scan. However, this will also result in reorganizing all the Bitmap indexes of the table. The problem of sparse bitmap matrixes, is discussed in [42].

Join index and Bitmap join index A join index [56] is the result of joining two tables on a join attribute and projecting the keys (or tuple id's) of the two tables. To join the two tables later, we can use the join index to fetch the tuples from the tables followed by a join. In relational DW systems, it is of interest to perform a multiple join (a StarJoin) on the fact tables and its surrounding dimension tables. Therefore, it will be helpful to build join indexes between the keys of the dimension tables and the corresponding foreign keys of the fact table. If the join indexes are represented in bitmap matrices, a multiple join could be replaced by a sequence of bitwise operations, followed by a relatively small number of *fetch* and *join* operations.

The problem associated with join indexes is that two separate join indexes cannot efficiently work together to fetch the desired tuple, like the Bitmap join index does. On the other hand, Bitmap join index has its own problems, such as sparsity, which the join index does not have. Algorithms for efficiently finding the common subset of two join indexes will be desirable for compensating the flaw of join indexes against the Bitmap join index.

StarJoin using Bitmap join index Join is an expensive operator, especially, for large tables. In the DW environment, multiple joins are performed on the fact table and its dimension tables to produce a report. This multiple join is called StarJoin. In [42], an efficient method of processing StarJoin using Bitmap join index is proposed.

For each tuple in the dimension table, a bitmap vector is constructed. This bitmap vector records the positions of the tuples in the fact table, that will join to the tuple in the dimension table, by setting the corresponding position in the bitmap vector to 1. The StarJoin is carried out in three phases. In the first phase, for each dimension, select the bitmap vectors of the tuples, that satisfy the selection conditions. Then, a bitwise OR is performed on the bitmap vectors selected from the same dimension table, resulting in one bitmap vector for each dimension. In the second phase, a bitwise AND will be applied on the bitmap vectors from all dimension tables to produce the final bitmap vector. Finally, the resulting bitmap vector will be used to select tuples from the fact table and join those tuples to the dimension tables.

Bitmap join indexes could greatly improve the StarJoins in the DW. However, some improvements are still possible. First, for larger fact table, the bitmap indexes get sparser. To solve this problem, bitmap compression using run-length encoding, or maintaining a list of tuple id's instead of a bitmap vector could be used [42]. Second, combining other techniques, such as data partitioning, with the bitmap join index, to eliminate the sparsity in the bitmap vectors and at the same time avoid large table scan could yield good improvements. This approach affects the structure of the bitmaps, and is still an open question.

Parallel join Much work about parallel join has been proposed [36, 16, 38, 11, 12, 30]. In the DW environment, joins occur in the StarJoin forms, *i.e.*, many tables join with one table. Beside the parallelism inside a single join, the parallelism among the multiple joins of a StarJoin could be explored, *e.g.*, the first phase of executing StarJoins using Bitmap join indexes discussed above could be performed in parallel. Further references to other special join methods can be found in [52].

Data partitioning Partitioning a large table horizontally or vertically into small tables can improve the query performance by avoiding scans of a large table, or by performing the table scan in parallel. Partitioning algorithms developed as fragmentation algorithms for distributed DBs need to be reevaluated for data warehousing and partitioning criteria need to be identified. A fact table can be horizontally partitioned according one or more dimensions, say by product or by (product, time). How will this affect the computation of the CUBE operator? What performance improvement can we gain from it for the processing of StarJoin using bitmap join indexes? How about the negative effects, *e.g.*, inter-table searches caused by table partitioning?

A fact table can also be vertically partitioned according to its dimensions, *i.e.*, all the foreign keys to the dimension tables are partitioned as separate tables. Obviously, in the distributed environment where the fact tables and the dimension tables are stored at distributed sites, a parallel semi-join can be easily applied by sending the vertical partition of the foreign keys to the dimension table. However, reconstructing vertical partitioned tables involves join operations, which is what we want to avoid. Intensive work on this issue is still needed.

Intelligent Large Table Scan In the DW environment, since most queries are read-only scanning is a basic operation that can be parallelized. A piggybacking scan for large tables was adopted by Redbrick. The basic idea is to avoid redundant table scans by consecutive queries, which request access to the same table. In other words, consecutive queries share the same results.

The idea of sharing the table scan result should be further spread to the sharing of all common subsets of query results, not just for table scans. Moreover, this sharing could be done by inter-process, or inter-thread communication. However, in a client-server environment, coordination and the communication protocols among different clients and the server are needed to be defined in order to let queries from different clients, but in the same local sub-network, share the table scan results.

5.3 Optimization issues

Aggregate/view selection To improve the performance of query processing, it is desirable to precompute aggregates/views along some dimensions and store them in the DW. However, due to space and time limitations not all possible aggregates can be precomputed and stored. To decide what should be precomputed in [53] a mathematical model was proposed to estimate the storage needed for storing an aggregate. In [28] greedy algorithms were proposed to select a set of views to materialize with respect to the storage space and the average response time of queries. In [25] a framework is proposed for the selection of views to optimize the query response time within polynomial-time. Much more work is needed in this area, particularly in combination with other issues that can benefit from these results or provide cross-fertilization, such as purging strategies, and modeling/design issues.

Query processing using aggregates or views The query processor of the DW must be capable of exploiting existing aggregates. In [15], algorithms were proposed to identify the relationship between the queries posed by users and the view definitions to see whether the queries could be answered by using the views or not. In addition, there are still many research problems concerning query processing using aggregates, *especially*, using the information contained in dimension hierarchies.

Other proposed query processing techniques Many query processing and optimization techniques can also be applied to the relational DW system. For example, executing selections before joins, interleaving group-bys and joins [59], parallel query processing techniques [29], *etc.*

6 Concluding Remarks

We tried to clarify some widespread misunderstandings about the relative merits of the relational and the multidimensional models for data warehousing. Based on expressiveness, scalability and mappability of the models we have adopted a multi-tier, multi-model architecture in which the multidimensional model is used as a conceptual model for design and user interaction in the Application Interface Layer and the relational model is used at the lower Data Management Layer and for the interaction with the operational source systems and the Data Store Layer.

We identified open research problems in the areas of DW modeling, maintenance, operation and optimization. Some of the topics have an impact in several areas. Work needs to be done in the near future in:

1. *Data warehouse modeling and design.* Design methodologies and design tools for DWs are needed with the appropriate support for aggregation hierarchies, mapping between the multidimensional and the relational models, and cost models for partitioning and aggregation that can be used from the early design stages. Adaptation of existing view integration work is required.
2. *Data warehouse architectures.* Research into future integration of legacy DWs is required, particularly as many companies are developing data marts of limited scope. Other interesting architectural issues concern the unbundling/streamlining of relational DBMSs, since data warehousing does not require full DBMS capabilities, such as, concurrency control and ACID properties.
3. *Data cleansing and loading.* Support mechanisms and metadata management for the resolution of conflicts during data cleansing are required, as well as parallel loading utilities.

4. *Data refreshing and purging.* Strategies for efficient data refreshing are required, and clear criteria and cost models for data removal from the DW.
5. *Extension of relational operators.* The set of relational operators must be extended to satisfy the user/application requirements. These extensions must include efficient mapping between the multidimensional and the relational worlds.
6. *Efficient implementation of operators.* The new operators must be efficiently implemented and the traditional operators, such as joins, must be reimplemented in an optimized manner for the warehouse requirements, particularly to support StarJoins and to take advantage of special indexing. Other operators that need reimplementation to take advantage of DW characteristics are large table scans, which must be parallelized, and partial results must be usable by multiple queries.
7. *Special indexing.* Data warehousing has special indexing requirements that are partially satisfied by bitmap indexes. These require extensions to deal better with sparsity, and to perform selective indexing, domain transformations and subdomain indexing.
8. *Query optimization.* Research in query optimization that exploits the information available in aggregation hierarchies and join indexes, as well as issues of space/time trade-offs for computed views is needed.

We identified a research agenda and pointed out areas in which the DB research community can contribute and participate in the data warehousing boom.

References

1. Aberdeen Group, *Data Warehouse Query Tools: Evolving to Relational OLAP*, Market Viewpoint, 8(8), July 1995.
2. Arbor Software, *The Role of the Multidimensional Database in a Data Warehousing Solution*, white paper available from *http://www.arborsoft.com/papers/*, 1995.
3. Arbor Software, *Relational OLAP: Expectations & Reality*, white paper available from *http://www.arborsoft.com/papers/*, 1995.
4. S. Agarwal, A.M. Keller, G. Wiederhold, K. Saraswat, *Flexible Relation: An Approach for Integrating Data from Multiple, Possibly Inconsistent Databases*, ICDE Conf., Taipei, 1995.
5. S. Agarwal, R. Agrawal, P. Deshpande, A. Gupta, J.F. Naughton, R. Ramakrishnan, S. Sarawagi, *On the Computation of Multidimensional Aggregates*, VLDB Conf., Bombay, 1996.
6. T. Barclay, R. Barnes, J. Gray, P. Sundaresan, *Loading Databases Using Dataflow Parallelism*, SIGMOD Record, 23(4), December 1994.
7. J.A. Blakeley, N. Coburn, and P. Larson, *Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates*, ACM TODS, 14(3), 1989.
8. C.J. Bontempo, C.M. Saracco, *Accelerating Indexed Searching*, Database Programming and Design (also available from *http://www.dbpd.com/*, July 1996
9. S. Chaudhuri, U. Dayal, *Data Warehousing and OLAP for Decision Support*, Tutorials of 22th VLDB Conf., Bombay, 1996.
10. S.S. Chawathe, A. Rajaraman, H. Garcia-Molina, J. Widom, *Change Detection in Hierarchically Structured Information*, SIGMOD Conf., Montreal, 1996.
11. M.S. Chen, P.S. Yu, K.L. Wu, *Scheduling and Processor Allocation for Parallel Execution of Multi-Join Queries*, ICDE Conf., Tempe, 1992.

12. M.S. Chen, P. S. Yu, K.L. Wu, *Optimization of Parallel Execution for Multi-Join Queries*, IEEE Trans. on Knowledge and Data Eng., 8(3), 1996.
13. E.F. Codd, S.B. Codd, C.T. Salley, *Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate*, white paper available from *http://www.arborsoft.com/papers/*, 1993.
14. L.S. Colby, T. Griffin, L. Libkin, I. S. Mumick, H. Trickey, *Algorithms for Deferred View Maintenance*, SIGMOD Conf., Montreal, 1996.
15. S. Dar, H.V. Jagadish, A.Y. Levy, D. Srivastava, *Answering Queries with Aggregation Using Views*, VLDB Conf., Bombay, 1996.
16. V. Deshpande, P. Larson, *The Design and Implementation of a Parallel Join Algorithm for Nested Relations on Shared-Memory Multiprocessors*, ICDE Conf., Tempe, 1992.
17. C.E. Dyreson, *Lazy Information Retrieval From an Incomplete Data Cube*, VLDB Conf., Bombay, 1996.
18. H. Edelstein, *Faster Data Warehouses: New tools provide high-performance querying through advanced indexing*, Information Week, Dec. 4, 1995.
19. R. Findelstein, *Understanding the Need for On-Line Analytical Servers*, white paper available from *http://www.arborsoft.com/papers/*.
20. J. Gray, A. Bosworth, A. Layman, H. Pirahesh, *Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total*, ICDE Conf., New Orleans, 1996.
21. T. Griffin, L. Libkin, *Incremental Maintenance of Views with Duplicates*, SIGMOD Conf., San Jose, 1995.
22. A. Gupta, I.S. Mumick, V.S. Subrahmanian, *Maintaining Views Incrementally*, SIGMOD Conf., Washingtion, D.C., 1993.
23. A. Gupta, I.S. Mumick, *Maintenance of Materialized Views: Problems, Techniques, and Applications*, Data Eng. Bulletin, 18(2), June 1995.
24. A. Gupta, I.S. Mumick, K.A. Ross, *Adapting Materialized Views after Redefinitions*, SIGMOD Conf., San Jose, 1995.
25. H. Gupta, *Selection of views to materialize in a data warehouse*, To be appeared in Int'l Conf. on Database Theory (ICDT), Greece, January, 1997.
26. J. Hammer, H. Garcia-Molina, J. Widom, W. Laobio, Y. Zhuge, *The Stanford Data Warehousing Project*, Data Eng. Bulletin, 18(2), June 1995.
27. J.V. Harrison, S. Dietrich, *Maintenance of Materialized Views in a Deductive Database: An Update Propagation Approach*, Workshop on Deductive Databases, JICSLP, 1992.
28. V. Harinarayan, A. Rajaraman, J. Ullman, *Implementing Data Cubes Efficiently*, SIGMOD Conf., Montreal, 1996.
29. W. Hong, M. Stonebraker, *Optimization of Parallel Query Execution Plans in XPRS*, Distributed and Parallel Databases, 1(1), January 1993.
30. H.I. Hsiao, M.S. Chen, P.S. Yu, *On Parallel Execution of Multiple Pipelined Hash Joins*, SIGMOD Conf., Minneapolis, 1994.
31. R. Hull, G. Zhou, *A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches*, SIGMOD Conf., Montreal, 1996.
32. W.H. Inmon, *Building the Data Warehouse*, John Wiley & Sons, March, 1996.
33. Kenan Technologies, *An Introduction to Multidimensional Database Technology*, Available from *http://www.kenan.com/*, 1995.
34. H.A. Kuno, E.A. Rundensteiner, *Using Object-oriented Principles to Optimize Update Propagation to Materialized Views*, ICDE Conf., New Orleans, 1996.
35. W. J. Labio and H. Garcia-Molina, *Efficient Snapshot Differential Algorithms for Data Warehousing*, VLDB Conf., Bombay, 1996.
36. C. Lee, Z.A. Chang, *Utilizing Page-Level Join Index for Optimization in Parallel Join Execution*, IEEE Trans. on Knowledge and Data Eng., 7(1), February 1995.

37. L. Libkin, R. Machlin, L. Wong, *A Query Language for Multidimensional Arrays: Design, Implementation, and Optimization Techniques*, SIGMOD Conf., Montreal, 1996.
38. T. Patrick Martin, P. Larson, V. Deshpande, *Parallel Hash-Based Join Algorithms for a Shared-Everything*, IEEE Trans. on Knowledge and Data Eng., 6(5), 1994.
39. Metadata Coalition, *http://www.metadata.org/*.
40. MicroStrategy, Inc., *The Case for Relational OLAP*, white paper available from *http://www.strategy.com/*, 1995.
41. C. Morpain, M. Cart, J. Ferrié, J. Pons, *Maintaining Database Consistency in Presence of Value Dependencies in Multidatabase Systems*, SIGMOD Conf., Montreal, 1996.
42. P. O'Neil, G. Graefe, *Multi-Table Joins Through Bitmapped Join Indices*, SIGMOD Record, 24(3), September 1995.
43. D. Quass, A. Gupta, I.S. Mumick, and J. Widom, *Making Views Self-Maintainable for Data Warehousing*, PDIS, Miami Beach, 1996.
44. N. Raden, *Modeling the Data Warehouse*, white paper available from *http://www.netcom.com/*.
45. S. Rao, A. Badia, D. Van Gucht, *Providing Better Support for a Class of Decision Support Queries*, SIGMOD Conf., Montreal, 1996.
46. Red Brick Systems, *Decision-Makers, Business Data, and RISQL*, white paper available from *http://www.redbrick.com/rbs/whpapers.html*, 1995.
47. K. A. Ross, D. Srivastava, S. Sudarshan, *Materialized View Maintenance and Integrity Constraint Checking Trading Space for Time*, SIGMOD Conf., Montreal, 1996.
48. N. Roussopoulos, C.M. Chen, S. Kelley, A. Delis, Y. Papakonstantinou, *The Maryland ADMS Project: Views R Us*, Data Eng. Bulletin, 18(2), June 1995.
49. K. Sahin, *Multidimensional Database Technology and Data Warehousing*, Database Journal (also available from *http://www.kenan.com/acumate/*, December 1995.
50. M.J. Saylor, M.G. Acharya, R.G. Trenkamp, *True Relational OLAP: The Future of Decision Support*, Database Journal, November-December 1995.
51. P. Scheuermann, J. Shim, R. Vingralek, *WATCHMAN : A Data Warehouse Intelligent Cache Manager*, VLDB Conf., Bombay, 1996.
52. D.K. Shin, A.C. Meltzer, *A New Join Algorithm*, SIGMOD Record, 23(4), 1994.
53. A. Shukla, P. Deshpande, J.F. Naughton, K. Ramaswamy, *Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies*, VLDB, Bombay, 1996.
54. Stanford Technology Group, Inc., *Designing the Data Warehouse on Relational Databases*, white paper available from *http://pwp.starnetinc.com/larryg/whitepap.html*, 1996.
55. M. Staudt, M. Jarke, *Incremental Maintenance of Externally Materialized Views*, VLDB Conf., Bombay, 1996.
56. P. Valduriez, *Join Indices*, ACM TODS, 12(2), June 1987.
57. J. Widom, *Research Problems in Data Warehousing*, Int'l Conf. on Info. and Knowledge Management (CIKM), November 1995.
58. J.L. Wiener, H. Gupta, W.J. Labio, Y. Zhuge, H. Garcia-Molina, J. Widom, *A System Prototype for Warehouse View Maintenance*, Workshop on Materialized Views: Techniques and Applications, June 1996.
59. W.P. Yan, P. Larson, *Performing Group-By Before Join*, ICDE, Houston, 1994.
60. G. Zhou, R. Hull, R. King, J.C. Franchitti, *Supporting Data Integration and Warehousing Using H_2O*, Data Eng. Bulletin, 18(2), 1995.
61. Y. Zhuge, H. Garcia-Molina, J. Hammer, J. Widom, *View Maintenance in a Warehousing Environment*, SIGMOD Conf., San Jose, 1995.
62. Y. Zhuge, H. Garcia-Molina, and J. L. Wiener, *The Strobe Algorithms for Multi-Source Warehouse Consistency*, PDIS, Miami Beach, 1996.

This article was processed using the LaTeX macro package with LLNCS style