Selected Research Issues in Decision Support Databases

David Maier[1]
Oregon Graduate Institute
maier@cse.ogi.edu

Mary Edie Meredith
Sequent Computer Systems
meredith@sequent.com

Leonard Shapiro[2]
Portland State University
len@cs.pdx.edu

Abstract

A flurry of buzzwords awaits anyone investigating database technology for decision support: data warehouse, Multi-dimensional and dimensional database, on-line analytical processing, star schema, slicing, dicing, drill-down and roll-up. We introduce the area via an example based on a long-ago project to design a repository on energy information for the US Department of Energy. Once we have introduced some terminology, we explore research issues associated with decision support databases, including representation, modeling, metadata, architectures and query processing. The purpose of this paper is to provide researchers with the background they need to contribute to this area.

1 Introduction

Decision-support databases raise issues that are often quite distinct from those encountered in transaction processing systems. In this paper we present what we regard as important areas for research and development for decision-support databases, particularly those aspects that arise from the dimensional view of data.

Some of the new focus on decision-support databases is due to the continued drop in prices of hardware and system software, particularly disk storage and memory. These lower prices in turn have fueled an interest in data warehousing, as companies find it economically feasible to keep large amounts of historical data on line. It is no longer prohibitive to duplicate data from operational databases into a central warehouse. Another reason for adding decision-support capabilities to relational products has

---

[1] This research was supported in part by NSF grants IRI-9118360 and IRI-9509955.

[2] This research was by NSF grant IRI-9119446

been the difficulty of supporting both decision support and transaction processing queries in one environment, especially a distributed environment.

Once companies bring a data warehouse on line, they of course want to derive a competitive advantage from it, which has created a demand for tools to probe and analyze such data resources. Some of the most popular tools are those which support a dimensional view of data, including statistical and OLAP techniques. Statistical and scientific databases have long supported operations that have only recently been discovered by OLAP vendors and researchers [SHO97]. We will provide references to this previous work, although our main focus is on OLAP queries.

In this paper we have tried to encompass both the academic and industrial perspectives, noting where possible the state of commercial progress in these areas. However, it is hard to track every new development, and products are progressing rapidly, so we are bound to be incomplete or somewhat out of date in places.

We begin in the next section by describing a real-life example of a decision-support database, in order to illustrate the new twist that the dimensional view brings to data management. This view of data is not well supported by data models traditionally associated with database management systems, such as hierarchical and relational. While structurally the dimensional view resembles tabular models, many of the data manipulations that arise in this view have not been well supported in conventional data manipulation languages such as SQL. Sometimes they are expressible, but only with convoluted queries or with poor performance [GRA96]. We illustrate common kinds of operations on dimensional data in the context of our example, as they may be unfamiliar to readers used to standard relational databases.

These limitations of conventional DBMS languages with regard to dimensional manipulations have been known for some time, and the market has responded in different ways. Some companies have brought out products with specialized storage managers for dimensional data that store it compactly and make dimensional operations on it efficient. Other companies provide tools that operate on top of existing relational DBMSs. These tools extract data and present it in a dimensional way, often with graphical interfaces that allow direct application of dimensional operators. Of course, extending standard relational systems is another approach, but most vendors have responded slowly. We think there are several factors that account for this slow movement. One is that popular database benchmarks, such as TPC-C and TPC-D, capture only limited aspects of dimensional manipulations, hence have not been a feature or performance driver for dimensional capabilities. Second, new features may be deprecated if they are not part of the SQL standard, and it is often a glacial process to extend that standard. Finally, suitable access methods and performance support may be lacking in the underlying implementations, and such enhancements are expensive.

This is an exciting time for dimensional analysis of warehoused data. The research community has only recently responded to the commercial proliferation of tools, and the commercial community is still in turmoil over which of the approaches outlined in this paper is most appropriate. We hope that this paper provides the research community some guidelines as to which topics in this fruitful field are ripe for picking.

2 Motivating Example

In response to the Arab oil embargo of 1973, and the resulting energy shortage, the DOE started gathering data on energy import, export, production, transport, storage and use in the US, down to the level of individual oil wells and pipelines, in some cases. One of the authors (DM) participated in this effort. States and energy companies were required to report monthly data on production, energy in storage, conversion, and so forth. The purpose of this data gathering effort was that in times of future energy shortage or dislocation (such as a flood blocking a rail line and halting coal deliveries), federal decision makers would be able to assess the severity of a problem, predict its extent (both temporally and geographically) and suggest actions to ameliorate it (generate more electricity at gas-fired power plants and have coal-fired plants back off, for example).

In practice, the data were nearly useless for supporting decision-makers with an urgent problem to solve. The difficulty was that the data, though timely and comprehensive, had been assembled in somewhere between 51 and 53 separate databases. (We never got a definitive count.) These databases did not interoperate. (Remember, it was the mid-1970's.) When an energy emergency arose, the response was to dump the contents of various databases to paper, and stick piles of printouts into a room with a bunch of people, who would try to compare and collate it manually. About two weeks later, the people would emerge from the room with some kind of assessment of the problem, by which time the problem had either resolved itself or become a crisis.

Thus, although the DOE had the information they needed to analyze energy emergencies, they could not bring it effectively to bear in support of decision-makers analyzing those emergencies. Having the data partitioned into 50 or so independent databases was probably necessary, in order to divide and conquer the problem of data modeling and collection for such a broad area.

Enter the Energy Emergency Management Information System (EEMIS) [CRO79a, CRO79b, CRO79c, DAC79]. The goal of EEMIS was to pull data out of the 50-odd individual databases into a single database, where cross-sector analyses could be answered without consuming several boxes of paper. EEMIS was to be what today we would call a *data warehouse*—a central repository that combines and summarizes data from multiple sources on an enterprise-wide basis. In the case of EEMIS, the enterprise was all the energy in the United States.

The job of producing the conceptual model for EEMIS was contracted to the Technology Assessment Group of the Department of Energy and Environment at Brookhaven National Laboratory in New York state. The staff there had one look at it and promptly subcontracted the job to a group of professors and graduate students at the State University of New York at Stony Brook. They struggled with the design problem for a semester, and came up with a design that was a cross between a relational and hierarchical schema, with about 145 field types and 190 segments.

From a high level, the domain didn't seem that complex. There were types of energy whose amounts were measured at certain time periods, for energy facilities and transportation links. For example,

> There were 635,000 barrels of Saudi Arabian Light crude oil in the Exxon storage facility at Bergen, New Jersey at the end of April, 1978, or

> 2.43 million cubic feet of natural gas moved between the Getty Bastion Bay gas processing plant and the Lake Charles LNG terminal in June, 1975.

It seems simple enough. Why the need for hundreds of fields and record types? The problem was all the summary data that was needed. As a simple example, while oil well production figures were reported monthly, in addition, quarterly and yearly totals were of interest. Life gets a bit more complex when considering forms of energy. There are individual *energy types*, such as Aviation Gasoline, Propane and Labuan Light Crude Oil. Energy types belong to *energy groups*, such as Gasoline, Light Products and Foreign Crude. Groups form *categories*, for instance, Wet Gas and Crude Oil. From categories there are *sectors*, such as coal, petroleum and natural gas. At the top of the hierarchy is the "Universal" form of energy. (See Figure 1.)

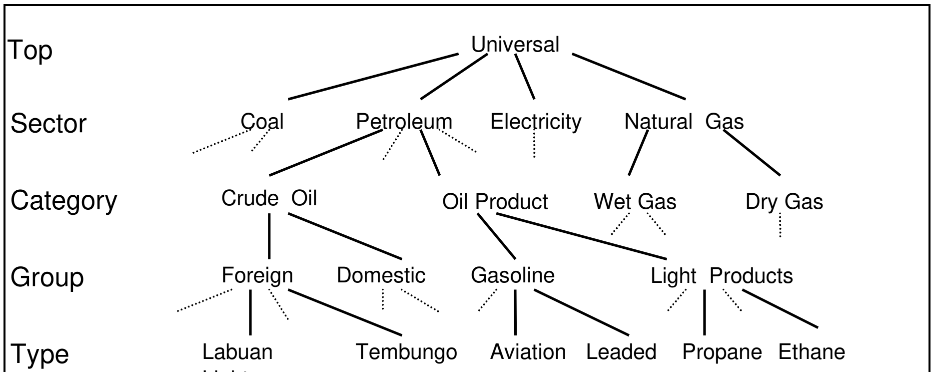


Figure 1. The Hierarchy of Energy Forms

Facilities were more complicated still, as there were two ways to classify them, by function and by geographic location. *Function* classified a facility as to what role it played in the energy chain of import or extraction, conversion, storage, and export or consumption. Examples of facility functions are crude storage, refinery, underground natural gas storage and military consumer base. Functions are further combined into *sectors*, such as petroleum, natural gas or large consumer. (See Figure 2.)

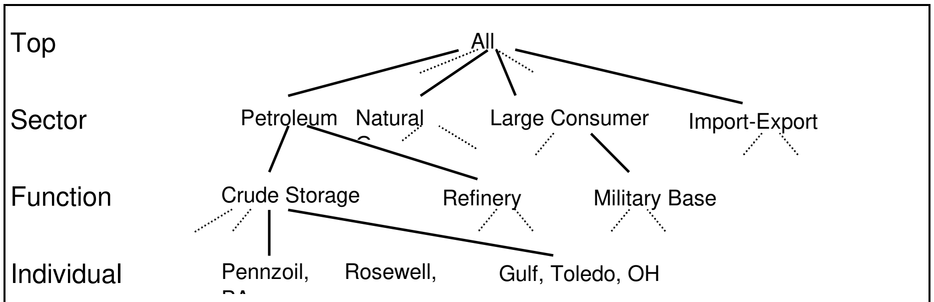


Figure 2. The Hierarchy of Facility Functions

Geographically, the primary grouping was by political boundary: county, state, region (Pacific Northwest, Mid-Atlantic), nation. However, there were also non-political areas, such as oil fields, that cut across state or even regional boundaries.

Transportation links were handled similarly to facilities. A link was categorized by its *mode* (pipeline, rail, barge, truck, oxcart, etc.), the functions of the source and destination facilities it connected, and its geographic location. The last was problematic, as a pipeline, say, could cross many areas.

Finally, there was a grand classification over both facilities and links called "RES Function" (for Reference Energy System, a particular model). It was a broad classification that spanned energy sectors and had classifications such as "extraction", "transmission" and "refining and conversion."

One of the big debates during the database design was whether or not to store aggregated values. On the one hand, computing, say, a national total of all petroleum production over a five-year period from monthly information on individual refineries would take an unacceptable amount of time. However, precomputing and storing all possible aggregations of interest would be prohibitive in space usage. Furthermore, it was not clear that we could predefine all the aggregates needed, since arbitrary groupings in the time dimension (e.g., November 1973-April 1974) might be of interest. In the end, we provided for the schema to store many aggregate values, but did not assume that the data would actually be precomputed for all of them. (Some of the aggregates were already present in the component databases being integrated, and we wanted to be able to capture those values at least.) We had an inkling that there should be a way to manage the precomputation and use of aggregates to make time-space tradeoffs. For example, it probably wouldn't make sense to store both regional and state aggregates, as the former can be computed quickly from the latter. However, perhaps this aspect should be dealt with at the physical design level, rather than being fixed in the logical design.

3 Dimensional Databases and OLAP

Our goal in this Section is to illustrate and define the OLAP style of access to dimensional databases. Our goal is not to be comprehensive, but only to provide enough background to help the reader understand the research issues we raise in Section 4. A more through treatment can be found in [KIM96].

A recent survey [SHO97] demonstrates clearly that many if not all of these concepts have been in use by the statistical database community for years, although the emphases have been somewhat different: the statistical community has emphasized modeling issues while the OLAP community has emphasized performance.

The commercial and application segments of the database community have recognized for a while a frequent pattern of base facts and aggregate dimensions occurring in decision-support databases [DEM94, FIN95, FRA94, KIM95, SQU95, KIM96]. The research community has only recently focused its attention on this area [GRA96, HAR96, SAL95]. (However, there has been a history of work in processing aggregation queries [CHA95].) There are dozens of front-end and back-end products currently on the market to support datasets of this form. Let us be a bit more precise in our characterization.

In this paper, we define a *dimensional database*[3] (DDB) as one that stores one or more kinds of *base facts*, and connects them to *dimensional information* [MEK96]. We construe DDB more broadly than just On-Line Analytical Processing (OLAP) tools. In particular, it does not imply a particular representation for the fact and dimension information. The base facts typically contain numeric fields, which queries aggregate via summary, taking optima (max and min), computing averages and so forth, via the classifications defined in the dimensional information. The dimensional information can also be used to select subsets of the base facts. The EEMIS database had base facts on amounts and costs of energy, and dimensional information on time, energy form, and link and facility function and geography.[4]

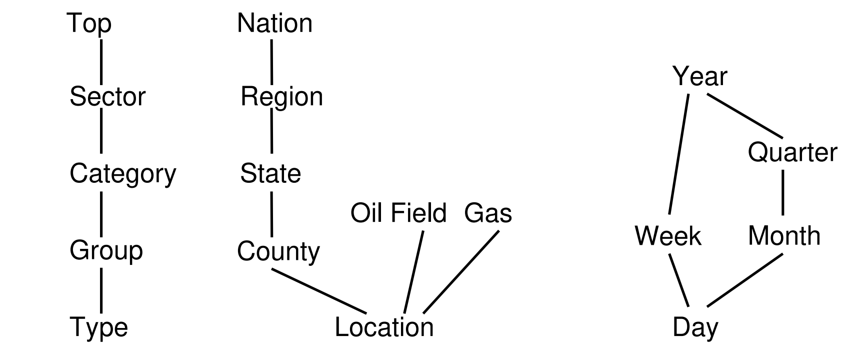
The dimensional information consists of one or more independent *dimensions*, each of which can be structured as multiple *levels* of granularity. A dimension *hierarchy* represents one aspect for classifying the base facts, and the levels represent finer or coarser grouping of base facts along their respective dimensions. In the EEMIS database, there is an energy form dimension, with levels of energy type, group, category, sector and top, as shown in Figure 3(a). The levels along a dimension need not form a strict hierarchy, however. Figures 3(b) and 3(c) show dimensions where

---

[3] Originally the term *multi-dimensional database* was used for what we call dimensional database. However, lately the term multi-dimensional database has come to be identified exclusively with ROLAP architectures, defined in Section 4.3 below.

[4] There is another kind of dataset that could be called dimensional, as found in many scientific and engineering applications. There the dimensions of the dataset represent continuous independent variables, such as time, latitude, longitude and altitude, and values at the interstices are measurements of dependent variables, such as temperature or pressure. While such datasets offer interesting challenges of their own, we do not include them in what we mean by DDBs.

some of the levels contain incomparable groupings over lower levels. For example, in the time dimension, days can be grouped into weeks and months, but week and month boundaries do not line up. (In fact, weeks and years don't line up exactly, but some commercial calendars have conventions for making them line up.)



(a) Energy Form **(b) Geographic Area** **(c) Time**

Figure 3. Levels Within Different Dimensions

Most DDB tools have a component to let a database designer declare and populate the dimensions when setting up a database, but end users might still want to define new groupings dynamically, such as combining months into two-month periods. We will refer to the alternative values within one level of a dimension as *labels*, though we do not mean to imply they are always strings. For instance, "Coal" is a label at the sector level of the energy-form dimension. We point out here that we know of no strictly syntactic characterization of when a particular database is a DDB, such as a certain schema pattern. All examples of DDBs we are aware of include a time dimension, which follows from their being directed at analyzing or characterizing a business *process*, which is by nature an ongoing activity. However, not every database with a temporal aspect is a DDB, nor is every decision support database or data warehouse.

The common access patterns characterize a DDB as much as the structural features. DDBs may be used to generate standard reports, or support a particular canned application, but more interesting is their use in On-Line Analytical Processing (OLAP) [COD94, THE95]. OLAP is ad hoc, exploratory data analysis, where a decision-maker is seeking patterns, trends, anomalies, correlations and so forth in order to better understand the history, current status or future of some aspect of the enterprise. OLAP usage of data is best characterized by the sequences of operations that users perform, rather than just the style of single queries. (The terminology we use is borrowed mainly from Kenan Technologies [KEN94] and Ralph Kimball [KIM96].)

In this paper, we will use "OLAP" to refer to a particular style of access to DDBs. We do not intend it to indicate a particular form of database schema or a particular software architecture, nor do we equate it with all decision-support access to databases.

An OLAP session will often start with *browsing*, or "surfing", the database, looking at different parts of it until some item of interest appears. For example, in the EEMIS database, a user might start by looking through petroleum shipments quarter by quarter and country by country. End-user OLAP tools generally let users view data in a spreadsheet-style display, with labels from one or more dimensions on the rows and columns, and base values or aggregates of those values in the cells. In Table 1 we see a fragment of the display, with labels for nations across the top, and petroleum energy groups and calendar quarter down the left side. The cells contain values for the amount of import of a given energy group in a given quarter from each country.

Imports	Albania	Algeria	Angola	...
foreign crude				
Q1 1973	22.0	106.2	55.3	...
Q2 1973	23.4	88.9	46.7	...
...	...	...	...	...
gasoline				
Q1 1973	9.2	18.0	0.0	...
Q2 1973	5.3	17.2	0.0	...
...	...	...	...	...
light products				
...	...	...	...	...

Table 1. One View of EEMIS Information

Once an area of interest is found, the user may tune his or her view of it. One way to modify the view is via *ranging* (or "data dicing"), where a subset of the fact data is selected based on labels along certain dimensions, attributes connected with those labels (such as the sulfur content of an energy type), or values of the cell data. In Table 2, the view has been limited to three countries, the foreign crude group and calendar quarters from 1976. In this case, we are depicting the entire display, rather than just a fragment of it.

Imports	Algeria	Gabon	Venezuela
foreign crude			
Q1 1976	122.5	43.7	309.5
Q2 1976	101.4	11.2	266.2

Imports	Algeria	Gabon	Venezuela
Q3 1976	96.0	50.3	300.1
Q4 1976	118.1	53.0	288.3

Table 2. View Modified with Range Selections

Another kind of view modification is *rotation* (also called “data slicing” or “pivoting”, though we have seen the term slicing used for both rotation and selection). Rotation allows the user to change which dimensions occur along each side. Table 3 shows the result of rotating the nation labels to the left of the view, with energy group nested underneath, and bring the calendar quarters to the top.

Imports	Q1 76	Q2 76	Q3 76	Q4 76
Algeria				
foreign crude	122.5	101.4	96.0	118.1
Gabon				
foreign crude	43.7	11.2	50.3	53.0
Venezuela				
foreign crude	309.5	266.2	300.1	288.3

Table 3. View Modified via Rotation

The other main kind of OLAP manipulation is to shift the level of aggregation along the dimensions. Going to a lower level of aggregation is called *drilling down*, while going to a more aggregated view is called *rolling up*. For example, in Table 3, the user may wonder about the dip in imports from Gabon in the second quarter, and could decide to drill down to the energy type level under Gabon, and perhaps roll up the quarters in the time dimension to yearly totals, as shown in Table 4. (Note that it is possible to have more than one level of aggregation showing for a dimension at one time.) The modified view shows that exports of one type of crude, Gamba, dropped to zero during the second quarter of 1976, perhaps because of a break in a pipeline or problems at a shipping port.

Drilling down and rolling up need not be limited to levels in the dimension hierarchies. Attributes of dimensions can be used to group values. For example, the crude oil energy category could be split up into low-, medium- and high-sulfur-content oil, or nations could be grouped into OPEC and non-OPEC members.

Imports	Q1 76	Q2 76	Q3 76	Q4 76	1976

Imports	Q1 76	Q2 76	Q3 76	Q4 76	1976
Algeria					
foreign crude	122.5	101.4	96.0	118.1	438.0
Gabon					
foreign crude	43.7	11.2	50.3	53.0	158.2
Anguille	8.1	7.1	9.5	8.0	32.7
Gamba	30.6	0.0	35.5	38.3	104.4
Mandji	5.0	4.1	5.3	6.7	21.1
Venezuela					
foreign crude	309.5	266.2	300.1	288.3	1164.1

Table 4. Drilling Down and Rolling Up

Many other forms of aggregation can be used in manipulating DDB data, besides simple sums: rankings, percentiles, extrema, running totals, and so forth. Other forms of derived data, such as differences and ratios, are also of interest. We note here that base facts will not always be summable values. For example, sums of inventory levels at points in time do not make sense, though other kinds of aggregation do apply, such as averages and minima. Base facts need not even be numbers. A DDB of sales or order information might include a code for mode of payment, if its finest level of information were individual transactions.

We have touched on the fundamentals of DDBs, and will now consider some of the research issues they engender.

4 Research Issues

In this section we discuss various research issues raised by the DDB paradigm. In some cases these issues are relevant only for OLAP queries, and we identify those cases. We begin with external issues, including how the data is to be represented and the importance of metadata. Then we consider internal issues such as query processing, parallelization and materialized views

4.1 Representation

Given the kinds of information and operations common for a DDB, what are the possible logical representations for DDB data? In this section we describe three possible representations for DDB data: a classical schema, like the one used by EEMIS, a so-called star or snowflake schema which is relational but better suited to

OLAP-style queries, and a non-relational, array or cube representation well suited to OLAP-style queries.

4.1.1 The Original Schema

In retrospect, the recommended design for the EEMIS data warehouse was not well matched to OLAP-style queries. The structure of the data tended to overly reflect the organization of the component data sources, which were biased to the reporting structure of the data. That structure tended to overdecompose data relative to the way that users wanted to access it. The EEMIS schema was also biased towards certain dimensions, based on where new labels are likely to be added to a level. For example, new facilities are not added frequently, nor are new energy types, but new time periods are added every month. Thus, EEMIS ended up with hierarchical schemas with facility information at the top, then energy form information, and time periods at the bottom. For example, Figure 4 shows a simplified version of the schema for an oil product bulk terminal (a kind of storage facility).

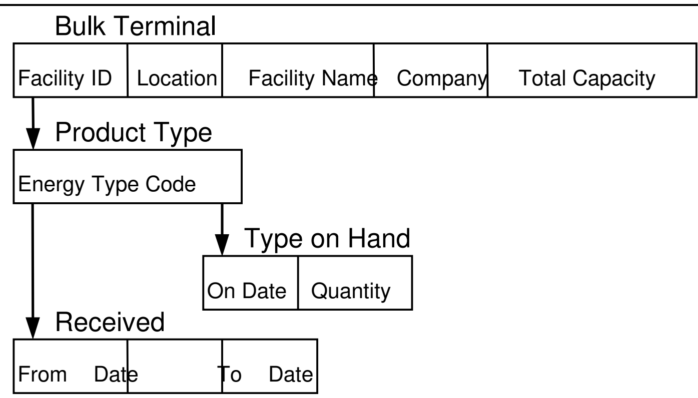


Figure 4. Hierarchical Schema for Oil Product Bulk

Facilities with different functions ended up being represented in different hierarchies, because they had different attributes. This organization supports roll-ups along the temporal dimension adequately, and roll-ups across energy type are not too bad to compute. Roll-ups over facility location or sector are difficult because records from different hierarchies are involved.

4.1.2 Stars and Snowflakes

DDBs implemented in relational DBMSs usually have a schema that does not emphasize one dimension over another. Instead, there is a logical schema build around a fact table that is linked to one or more dimension tables. Although the arrows of Figure 4's hierarchical schema are quite similar to those of Figure 5, those of Figure 4 represent only one-directional access to data. The simplest form is often called a *star* schema—the base fact table is portrayed in the center, and the dimension tables arrayed around it, as shown in Figure 5. The ENERGY FACT records contain foreign keys for the three dimension tables.

Usually, a relational schema needs to be more complicated than this, because of the richness of information in some dimensions. It may be that a dimension has independent subdimensions, such as the location and function aspects of the facility dimension, or that storing labels for all hierarchy levels in a single table creates redundant information. For instance, rows in ENERGY FORM will repeat the relationship between an energy group and its category many times. The redundancy can be worse if there are attributes functionally determined by associated labels, such as the name of the energy commission for a state. Thus, following the tenets of normal forms, the dimension tables can be broken out further, becoming a *snowflake* schema, as in Figure 6. There are other variations on the theme.

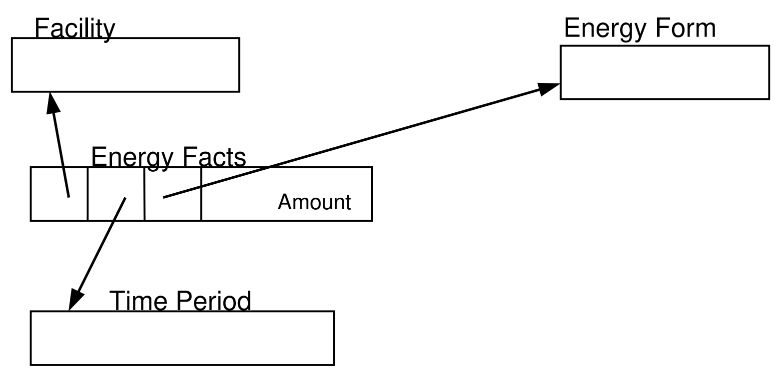


Figure 5. Relational Star Schema for Energy DDB

Many OLAP-style queries can be answered using SQL queries against star and snowflake schemas, via joins of the fact table with one or more dimension tables, grouping and filtering on label fields and aggregating on fact fields. However, such queries can be expensive to compute, and SQL cannot express all varieties of aggregates without extensions or use of multiple statements [GRA96].

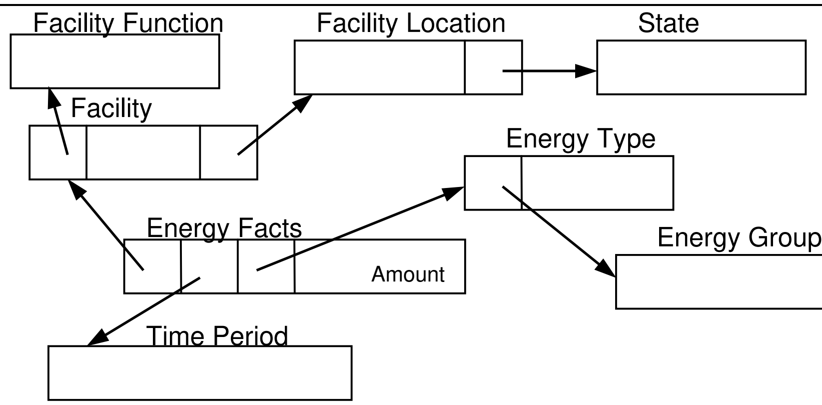


Figure 6. Snowflake Schema for Energy DDB

4.1.3 Array Representation

A third approach is adopted by some DDB products that have their own storage managers. That approach is to store the base fact values in a n_1 by n_2 by ... by n_k array, where n_i is the number of labels at the lowest level in the i^{th} dimension, there being k dimensions in all. The representation has the advantage that label information is stored once, external to the array, and the array stores only base values. Ranging can be done by indexing into the array. The exact set of cells for a particular roll-up can be calculated easily as well. A downside is that base facts might not be dense in all dimensions. For example, crude oil energy types are generally associated with single countries. Gabon exports Gamba crude oil, but no other country does, so the intersection of Gamba and any other country will always have a zero value. Thus, sparse array structuring techniques are needed for space efficiency. This approach works best with base facts of constant size, such as numeric values.

Research Issues: We raise a few issues here about representing DDB fact data with standard relational tables versus a customized representation, such as arrays.

1. Does the representation allow the fact data to be decomposed into a union of small structures? Having smaller pieces can make data more manageable, for archiving and reorganization, for example. Relational tables are easily split into segments. Many relational databases have support for physical partitions of a single table already, usually in support of parallel access or evaluation. These implementations vary greatly in sophistication and functionality [INF96, ORA97, SYB98]. SQL databases are starting to provide union views, which means partitioning could take place at the logical level. Any dimension or group of dimensions could be used for decomposing a fact table. Most array representations would seem amenable to partitioning only along the dimension that varies most slowly in the physical layout.

2. Whether the fact data is sparse or dense relative to the cross product of dimension labels seems to have a great affect on whether tabular or array representation is more space efficient. The tabular representation has the advantage of just not storing a row for a combination of dimensions where no fact value is present, whereas some array representations will devote space to fact values whether they are present or not. On the other hand, the tabular representation explicitly stores dimensional information (at least keys) for each fact, whereas such information is implicitly associated with facts in the array representation, and not stored as part of the fact structure. Neither of the arguments above is the last word—there are schemes to compress tables with frequent repetition of column values between successive rows, and sparse array representations are in use in some DDB products.

Are there any other natural representations for DDBs? Would object-oriented representations help here?

4.2 Modeling Issues

While the dimensional view of data captures much of the modeling complexity of the EEMIS database, there are still some issues not addressed.

4.2.1 Non-Uniform and Non-Tree Hierarchies

The dimension hierarchies we presented showed uniform depth in classification. In fact, that was an oversimplification. For example, in the geographic hierarchy, there was no breakdown for regions, states and counties for foreign countries. Hence, one could drill down on the United States, but not on, say, Saudi Arabia. In the energy form hierarchy, there was no differentiation at the category, group or type levels for electricity—kilowatts are kilowatts. Also, while the EEMIS hierarchies had uniform levels across them, that arrangement may have been artificial. In the geographic dimension, not every facility had an associated oil or gas field. There is no *a priori* reason to expect that the natural levels of classification for production facilities would be the same as for energy consumers. In fact, the consumption sector in EEMIS could have used an additional level of classification based on size.

The relationship between labels at one level in a dimension to a higher (more aggregated) level need not be many-to-one, though our examples have had that structure. The higher level could represent overlapping categorizations of the lower level, giving a many-to-many relationship between labels at the two levels. For example, a store may have groupings of products such as “economy size,” “seasonal” and “promotional,” where a single product could belong to all three groupings.

Research Issue: How can DDB models and tools accommodate non-uniform dimension hierarchies? Can the different representations described previously handle many-to-many categorizations? Can a many-to-many categorization be reduced to several many-to-one categorizations?

4.2.2 Instances, Intervals and Time Shifts

The time dimension has some of the most complex semantics. While the finest granularity in EEMIS for that dimension was monthly reports, "April 1976" did not mean the same thing in all contexts. In some cases, it means a value associated with the entire month of April (or a one-month period ending in April), such as for production from a gas well. In other cases, it means a measurement at a particular time in the month, such as quantity on hand at a storage facility. In the EEMIS data sources, the dates of measurement were not uniform within a month. Thus, it was possible that storage facility A reported figures for 10 April, and facility B reported for 25 April. Figures for A and B might count the same oil twice, if it moved between A and B between the 10^{th} and the 25^{th}. It may be that such problems are more a manifestation of poor design of reporting protocols rather than inherent modeling problems, or that different aspects of time are handled adequately by current temporal data models. However, different data sources that are feeding into a data warehouse may well have different time conventions, and features for interpolation and adjustment would be useful. Note that simply going to a finer granularity for the lowest level in the time dimension could cause a great expansion in the size of the fact data

***Research Issue*:** Are current approaches to time in databases adequate for DDBs, or are new capabilities needed?

4.2.3 Regularizing Data

The data being combined in a DDB may not have uniform domains for the values being aggregated by OLAP queries. For example, a company tracking sales might count individual items at retail outlets, but use case lots in recording institutional sales. Obviously, the units have to be reconciled before aggregating across retail and institutional segments. In the EEMIS domain, different energy sectors had different ways of measuring energy: barrels of petroleum, cubic feet of natural gas, tons of coals, kilowatt-hours of electricity. Combinations across sectors demanded conversion to a common measure, such as BTUs. (The conversion is non-trivial in some cases—the BTUs in a barrel of crude oil depend on the specific gravity of its energy type.) Converting everything to common units as it enters the DDB is probably not acceptable, as a user wants to see the normal units when looking at a single sector.

Not all regularization problems are simple unit conversions. There may be different conventions for descriptive labels in different parts of a business. Converting to a common system upon entry into the DDB would again deprive users of looking at data in its most natural form, and might also lose information if disparate attributes have to be mapped into a "least common denominator" domain. Another possible problem is overlapping of data represented at a lower level. For example, one dimension level might represent the number of physicians in each department of a hospital. If a physician can belong to more than one department, a sum at this level may obtain more than the number of physicians in the hospital. The problem of regularizing data was first investigated by the statistical database community [LES97].

In [KIM96], Kimball offers solutions for accessing facts associated with heterogeneous dimensions(different insurance products in his example) which allows for custom facts and custom dimension attributes. The approach yields a complicated design and requires high maintenance (add a new product line and you have to add tables to the database and possibly modify tools that access it).

***Research Issue*:** Are there simple methods of allowing heterogeneity of data at lower levels of detail and easy addition of new classes, while automatically regularizing it on roll up to higher levels of aggregation?

Specialization

The hierarchies found in DDBs are not the same as the specialization hierarchies supported in object-oriented databases (OODBs). Dimensional hierarchies ultimately are used to relate states of data items—which instances at a finer level combine to form an aggregate instance at a coarser level. OODB hierarchies concern operations and express relationships between types, not instances. That is, such a hierarchy could express that the "Crude Oil Terminal" type specializes the "Energy Storage Facility" type in that any operation defined on the latter type is also defined on the former. But such a subtype relationship does *not* induce connections between instances of those types, for example, that the "amount-in-storage" value of `CrudeOilTerminal C1` contributes to the "amount-in-storage" value of `EnergyStorageFacility E2`. Object hierarchies are good for capturing semantics of data items that aren't wholly identical in structure.

There were places in the EEMIS design where an OO-style hierarchy would have been useful. There are many similarities between an oil refinery and a natural gas processing plant. It would have been helpful to capture that transportation links are in fact a kind of storage facility for "in-transit" energy. The amount of coal on a rail line or oil in a pipeline can be non-trivial. Current approaches to DDBs demand uniformity in the dimensional entities, which may mean omitting information in order to fit a common pattern.

Research Issue: Can the DDB view of data be combined with object-oriented models to allow some heterogeneity in dimensional information, while capturing commonalities for broad aggregates?

4.2.4 Modeling strategies

The classic relational modeling strategies, e.g., E-R diagrams, are inadequate for DDB schemas. One problem is that DDB modeling really starts with determining the business processes to be tracked, what granularity of data is needed to model those processes, and which parts of that data are changeable, which are not specifically addressed by conventional data modeling techniques. Also, classical relational modeling favors normalized data, whereas dimensional data is typically unnormalized (e.g., a relational model of Figure 5 would repeat the association of Oil Product with Petroleum many times). The loss of performance from non-normalized data is minimal since dimensions are typically small. Furthermore, classical relational modeling does not group data by facts and dimensions, which is crucial to DDB performance and to process modeling.

New approaches to Dimensional Modeling have been described for relational databases [KIM96] and OLAP [BUL96]. Discussions regarding aggregate design are available [MAD96]. However, there is a great leap from description and examples to execution. The bulk of the dimensional design happening today is conducted by consultants who simply have a knack for modeling. Systematic, well-defined approaches available to the public for dimensionally modeling processes are lacking. Bitter debate continues regarding even the use of dimensional modeling.

Criteria for designing DDB models need to be developed, analogous to ER diagrams and normal forms for relational models. These criteria could help to answer questions such as which attributes belong in which dimensions (e.g., should the data of Figure 1 be in a single dimension or one dimension per sector?), and whether to store different kinds of values in separate fact tables. A final consideration is that DDB databases are not populated directly from applications, but rather are uploaded from OLTP databases. Thus the design of the DDB schema may be constrained by what data is actually in the OLTP schema. A graphical technique developed for statistical databases [RAS90] may be helpful as a component of new modeling strategies and tools which support them.

Research issues: New modeling strategies and design tools are needed to support OLAP applications.

4.3 Metadata

The underlying data sources that are used to create a DDB are never as uniform as the DDB schema would indicate. First off, there are issues of completeness. The various states and companies reporting data to the DOE varied greatly in how promptly they supplied information. Thus, a particular reporting period might have base data missing for multiple months. Decision-makers need to use the partial data nonetheless. The problems are how to compute functions such as averages when values are missing, and how to indicate to a user that a particular aggregate was computed from incomplete data. The obvious solution is to use some default values for missing data, but this can give misleading results, e.g. if zero is used as a default numeric value. A more challenging solution is to use metadata to indicate how each aggregate should handle missing data, e.g., computing an average only on available data and informing the user somehow about how much data is missing.

Another problem is that some data sources do not record data at the lowest level of granularity in all dimensions. Some of the EEMIS data sources bottomed out at the group level along the energy form dimension, and at the state level in the facility-location subdimension. Thus, it would not be possible to drill down to arbitrary levels in the EEMIS data everywhere in the database. The query execution mechanism must adapt to this situation, for example, by informing the user that certain drilldowns are not possible and why.

A third problem is that certain label values might not be present explicitly in a data source. A database on crude oil imports might not contain "crude oil" as an explicit value in any field. This is an issue in any database, but is especially important in dimensional databases where metadata, such as which relations are fact tables and which are dimensions, is even more critical.

Research Issues: What kinds of metadata are needed to capture the variability in data completeness and granularity that is sure to arise in a DDB? How can end-user tools exploit that metadata to properly annotate results and indicate legality of operations?

4.4 Architectures

Several software architectures have been proposed to support OLAP querying of DDBs [RED95]. We list some of them below.

1. Some products provide special storage structures to hold the array representation described in Section 4.1.3 [KEN94, ESS98, MIC98]. Those structures, typically dimensional arrays with some way to efficiently store sparse data regions, can be augmented with indexes on labels and precomputed aggregates along some dimensions. This architectural approach is sometimes labeled MOLAP, for "multidimensional OLAP". These special storage structures are well suited to OLAP applications, since they require less storage (labels are stored only once for the corresponding slice of the array) and they can respond quickly to OLAP requests (since data is typically clustered and aggregated ideally for OLAP queries). However, there are questions of scaleability with this approach; estimates of the size of database which can be handled with these dimensional array structures range currently from 1 to 50 Gigabytes [INF96, MIC98]. MOLAP architectures can generally populate their specialized structures with extracts from relational databases.

2. Other products, typically based on relational databases, respond to OLAP queries by reading data directly from relational storage structures. These products are highly scaleable, but can suffer from poor performance on OLAP queries. To avoid this, some products use extensive bit-mapping techniques [SYB98], and some apply additional parallel technology [INF98, ORA98]. These are sometimes called ROLAP architectures, for "Relational OLAP." Some relational databases have been tuned to OLAP use, with heavy indexing, optimizations for batch update and bulk access, and extending the query language and processor with new features [RED95].

Conventional systems [INF96, ORA98, SYB98] are being augmented to handle OLAP style queries, but in a limited way. They focus on providing improved algorithms and indexing (cf. Section 4.5.3). Other vendors specialized for OLAP style access [RED97, ESS98] claim to consider the whole environment in a different, more comprehensive way. To some degree this is debated in a DBP&D article "High Noon for the TPCD" [BAL97], as RedBrick believes the DSS benchmark (developed by conventional RDBMS representatives) does not test the things it should for the environment. Each camp satisfies the needs of different users: conventional systems have the advantage of scalability to the terabyte range and the use of legacy systems, whereas vendors specialized for OLAP provide better performance. Which approach is most effective in which cases?

2a. Some OLAP tools [STG97, MIC98, PLA97] read raw data from relational databases but also store some heavily used data in aggregated form. (The database community would call such aggregated data a materialized view.) Performance is much better if the database administrator is wise enough to identify which data will be used heavily, and at what dimensional levels it is aggregated. This architecture is a variant of ROLAP, since the cached data is typically stored in relational tables.

2b. A more sophisticated version of architecture 2a is to interpose an intelligent query manager between the OLAP tool and the DDB, which can perform semantic query optimizations not supported by the query processor of the underlying DBMS and cache results of previous queries. An example of the former is rewriting a query against a union view to access only one of the component tables. An example of the latter is holding on to the rows used to compute an aggregate, in case the user decided to drill down on the same selection of data. This technique of dynamically caching retrieved data will be effective if there is locality to reference patterns, which seems to be the case with OLAP queries (cf. Section 4.5.1).

Note that with any of these architectures, the OLAP interface itself might keep a specialized structure in main memory to organize and cache data.

Research Issues: What are the trade-offs between such architectures? When does it make sense to have a specialized DDB-oriented DBMS, versus conventional systems augmented to handle such applications?

4.5 Query Processing

OLAP queries on DDBs present a rich domain for research on query processing and access methods, as well as query language design.

4.5.1 Query Patterns

Typical OLAP-type sessions result in sequences of related queries, with high likelihood of overlap. Thus there is a kind of "locality" to the queries. This locality is exemplified by the names of common operations such as roll up and drill down, which imply that the data used is related to that for the previous request. Query optimization of these query patterns is not quite the same as multiple query optimization, where the queries are known all at once. Thus, strategies to take advantage of the locality of sequences of requests must speculatively store previous query results or subexpressions. Proposals on how to organize such information are starting to appear [SAL95, HAR96].

OLAP requests often want data aggregated at multiple levels in the same answer. The Data Cube [GRA96] is an approach to improve relational database support for OLAP-type requests by extending SQL to express such patterns and providing a specialized operator to compute them.

As we better understand OLAP-type queries, it will be important to develop benchmarks to capture them. OLAP benchmarks are beginning to be developed [OLA98], though the initial benchmarks are individual queries instead of patterns of several queries. Another issue for benchmarks is adequately capturing the effects of data skew in the test dataset. By *data skew* we mean here the degree to which facts are sparse or dense over the cross product of dimension labels. A good benchmark would reveal sensitivity to data skew.

Research Issues: What other techniques can be applied to sequences of overlapping OLAP queries? Are there new auxiliary access structures that would aid in evaluating these kinds of queries? Does the DataCube extension to SQL adequately cover the space of OLAP requests, or are there common forms of OLAP requests that are still hard or impossible to express in SQL even with that extension. What are actual patterns of customer OLAP queries? What are appropriate benchmarks of query sequences?

4.5.2 Materialized Views

Different users of OLAP interfaces will often issue queries over the same subsets of data and will perform similar aggregations over them, for example a time slice for the most recent year's data or a roll up by states. This commonality makes it likely that storage of materialized views will be effective in reducing the cost of queries, which use all, or part of those views. Some commercial vendors [INF97] have implemented the storage of materialized views, and replication mechanisms implicitly support materialized views by allowing replicated data to be specified by a query [STA94]). Recently Red Brick has introduced a sophisticated version of materialized view storage in its Vista System [RED97]. The research community has contributed to these issues [MUG96].

There are three primary questions here: which views to materialize, how to update the materialized views, and how to incorporate the views into query processing.

Some work has been done on the first problem [HAR96], developing algorithms for choosing the best *n* views to materialize, for a given *n*. This static approach has limited usefulness over the lifetime of a DDB, since subsets and aggregates of interest change over time. There is a lot of advantage to be gained just by looking at materialized views that are aggregates over base tables (as is done in Red Brick's Vista), but materialized views with selections and joins also deserve attention here.

The second question has been studied extensively in the past, primarily for OLTP applications [CKP95, HGM95]. OLAP applications bring new problems to bear. Since many OLAP applications are read only, or data changes only periodically and with batch updates, the update problem is quite different from OLTP environments, where individual updates to base data happen continually. For example, if there are batched updates to a materialized view it may not be worthwhile to update the view, but to invalidate and rebuild it either lazily (when requested) or eagerly.

Concerning the third issue, the query optimizer needs to recognize equivalences among expressions and know when to modify a query to go to the materialized view, or simplify it to use one part of a decomposition. Current relational systems generally require intervention from outside the database to revise queries, either by a programmer rewriting queries to take advantage of database structures, or with an intelligent driver that sits between the OLAP interface and the database, as in architecture 2b in Section 4.3. In some situations it will be necessary to store large numbers of materialized views, and the optimizer will be challenged to search through all these views to determine which are relevant to the presented query, and whether it pays to use them.

Research Issues: Are there effective dynamic algorithms that materialize and dematerialize views according to the needs of the current query workload? How can the determination of which view to materialize and which tables to restructure best be automated? How can the batch mode of DDB updates yield simpler algorithms for updating views? Can algorithms be developed which effectively decide between updating and invalidating a view? How should query optimizers be modified to take advantage of such auxiliary information? What indexes or other structures can be imposed on the set of materialized views, to provide adequate access to materialized views?

4.5.3 Aggregate transforms and bit structures

There is also much work to do on improving conventional relational query processing techniques to perform better on decision-support queries in general, such as are typified by the TPC-D benchmark. Some of our own work [BIL97] has shown the efficiency of query transforms aimed at aggregates, the addition of a bit-join operator, and specialized search heuristics. Certain parts of these queries contribute very little to the overall evaluation time, but consume large amounts of optimization time. Quickly finding a “good enough” plan is better than taking forever to get the optimal plan for such subexpressions.

CCA's Model 204 DBMS originally developed a family of techniques for accelerating the performance of complex queries [ONE87]. These techniques encode critical aspects of data (e.g., which facilities are in the Northwest region) in bit vectors, then use those bit vectors to dramatically improve the performance of queries that formerly had to scan the entire facility table. Current database vendors [SYB98, ORA98, INF98] have implemented some of these techniques and claim that they improve the performance of decision-support queries by factors of 50 or more.

Similar bit encoding techniques have been used by the statistical database community for many years [WLO85, WLO86].

Research Issues: Are there new optimization techniques better suited to optimizing new structures such as bit structures? Are bit-encoding techniques as effective in dimensional schemas as they have been in OLTP schemas? How do they compare with other techniques such as materialized aggregate views?

4.6 Parallel DBMSs

Parallel DBMSs have been very effective at speeding up queries against schemas and query loads that are typical in Decision Support applications, such as OLAP queries. These DBMSs use both interquery and intraquery parallelism. In order to achieve scaleable performance, the data must be spread over multiple discs, so multiple arms can be used as higher speeds are needed. There are many delicate issues to be resolved in implementing a parallel DBMS and in the physical design of data.

Research Issues: How effective will parallel DBMSs be with star and snowflake schemas? Will any of their internals need to be changed to be effective for such schemas? Guidelines for physical data design in parallel DBMSs are well known for OLTP schemas [GHA92]. Will different distribution criteria be more appropriate for star schemas?

4.7 Information System Management

A data warehouse or repository to support a DDB is almost surely part of a larger information system, as the warehouse needs to be populated from somewhere, and that somewhere is generally the operations database or databases of an enterprise. We will refer to these databases that provide information to a warehouse as *source* databases. Many system management issues arise from the movement of data from the source databases to the warehouse. The transfer process itself can take a non-trivial amount of time, as the data need to be extracted from the source databases, run through cleaning and checking routines, probably restructured, and loaded into the warehouse. Even then, the job may not be done, as there are probably indexes and derived datasets (precomputed aggregates and other materialized views) to update.

In large companies, with international operations, it may not be feasible to shut down the warehouse to queries while new data is being added. There may similar restrictions on source databases, in that they can't prohibit updates while data is being extracted. Note that putting the load operation inside a standard transaction is not really a solution. While a transaction would provide isolation of concurrent queries from the new data being loaded, durability of the data being loaded must be provided at a finer granularity. If the load transaction fails, it is not acceptable to lose hours of work. Another problem is global synchronization. The data for the warehouse is likely being drawn from several source databases. While those sources may each be internally consistent, one can not be assume that they are synchronized with each other. For example, in the warehouse is drawing data from an Energy Production source and an Energy Storage source, there is no guarantee that the data from the two sources is current to the same point of time, even if those two sources are accessed simultaneously.

We suspect the solution to these problems lies in breaking large warehouse loads into smaller pieces, while controlling visibility to loaded increments until the full load is completed. A naive example would be to express a warehouse table as a union view. New data being loaded would be placed into a new table in increments. With the advent of partitioning, some vendors [ORA98] allow new data to be stored in separate tables, and then added to the larger table all at once as a partition. Although not intended for this purpose, it might be possible to use the partitioning mechanism to insert updates into a separate table over several transactions (initial loads from different sources plus corrections) then add the new data all at once. Issues are likely to arise if the update cycle does not match the partitioning strategy (days versus months, for example) or updates occur naturally across all partitions not just one. Parallel load utilities (insert/update/delete/bulk loads) from vendors [ORA98, INF98] allow the load to be a whole transaction. Administrators have to work hard to remove errors before the load runs. Otherwise catching errors during the load produces two bad options:(1) allowing the load to continue skipping over input errors which leave users exposed to incomplete data , or (2) aborting the load, which delays the load and wastes resources. What is needed is consistency over a series of transactions, not just one transaction. Then the first option could be taken, several input sources could be used, and the errors corrected before users see the data.

Research Issues: What mechanisms will support simultaneous queries and batch loads to warehouses? How can indexes and views be constructed to better support updates from batch loads? Can views and indexes be segmented the same way as data, so only parts of these auxiliary structures are affected when base data is updated? Are there methods using existing modeling and query facilities that provide the equivalent of segmented loads?

5 Summary and Future Work

While dimensional database applications have been around for decades, there has been a recent increase of interest fueled by the need for data warehouses and the popularity of OLAP. We have set forth research issues in the areas of representation, architecture, query, modeling and metadata, and there are surely dozens more problems of which we are not yet aware.

In addition to attacking the research issues we have presented, it would be useful to prioritize these and related issues. First, it would be useful to have precise characterizations of dimensional databases and of OLAP applications, more specific than those we have provided in Section 3. Next, the claims we have made in Section 4, about the usefulness of many techniques to OLAP applications, should be measured with respect to those characterizations of DDB and OLAP. Finally, based on this foundation, the various data structures proposed in Sections 4.1 and 4.5 should be evaluated.

6 Acknowledgments

The authors would like to thank Lois Delcambre, Kaye Van Valkenburg and the DISC Decision-Support study group for ideas and help that went into this paper. We also thank the referees for many useful suggestions.

7 References

This paper is a revised and expanded version of a previous workshop presentation [MMS96].

[BAL97] C. Ballinger and M. Olson, " High Noon for TPC-D," Database Programming and Design, September 1997

[BIL97] K. Billings, Bottom-up Optimization of TPC-D, Master's Thesis, Portland State University Computer Science Department, 1997, http://www.cs.pdx.edu/~kgb/t/title.html .

[BUL96] Bulos, Dan, OLAP Database Design: a New Dimension, Database Programming and Design, June 1996, Pg 33-37.

[CHA95] S. Chaudhuri and K. Shim. An overview of cost-based optimization of queries with aggregates. *Data Engineering* 18(3), September 1995.

[CKP95] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim, Optimizing queries with materialized views. In Proceedings of the 11th Conference on Data Engineering, Pages 190-200, 1995.

[COD94] E. F. Codd and S. B. Codd. OLAP (on-line analytical processing). White paper, E. F. Codd & Associates, 1994.

[CRO79a] K. Croteau, J. Heller, T. Krawiec, D. Maier and B. Russell. The conceptual model for EEMIS. Dept. of Computer Science, SUNY Stony Brook, April 1979.

[CRO79b] K. Croteau, J. Heller, T. Krawiec, D. Maier, B. Russell and P. Wolk. Final report: The conceptual model for EEMIS. Dept. of Computer Science, SUNY Stony Brook, June 1979.

[CRO79c] K.Croteau, A. S. Kydes and D. Maier. EEMIS data sector correspondence with conceptual database design. Brookhaven National Laboratory Report BNL 51091, July 1979.

[DAC79] J. D'Acierno, A. Hermelee, C. P. Frederickson and K. Van Valkenburg. Methodology for coding the Energy Emergency Management

Information System. Brookhaven National Laboratory Report BNL 51086, November 1979.

[DEM94] M. Demarest. Building the Data Mart. *DBMS*, June 1994.

[ESS98] ESSBASE, by Arbor, Inc, 1344 Crossman Avenue, Sunnyvale, CA 94089 800-858-1666, info@arborsoft.com, http://www.arborsoft.com

[FIN95] R. Finkelstein. MDD: Database reaches the next dimension. *Database Programming & Design* 8(4), April 1995.

[FRA94] M. Frank. A drill-down analysis of dimensional databases. *DBMS*, June 1994.

[GHA92] S. Ghandeharizadeh, D. DeWitt, W. Qureshi, Performance Analysis of Alternative Multi-Attribute Declustering Strategies, Proc. ACM SIGMOD Conf. San Diego, CA, June 1992, Pg 29

[GRA96] J. Gray, A. Bosworth, A. Layman and H. Pirahesh. Data Cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. *Proc. Twelfth Intl. Conf. on Data Engineering*, IEEE, New Orleans, February-March 1996.

[HAR96] V. Harinarayan, A. Rajaraman and J. D. Ullman. Implementing data cubes efficiently. Proceedings of the ACM SIGMOD Intl. Conf. on Management of Data, June 1996.

[HGM95] J. Hammer, H. Garcia-Molina, J. Widom, W. Labio and Y. Zhuge, The Stanford Data Warehouse Project. IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing, 18(2):41-48, June 1995.

[INF96] Informix, http://www.informix.com

[INF97] Informix (formerly Stanford Technology Group's Metacube), http://www.informix.com/informix/products/techbrfs/metacube/2129274.htm

[INF98] Informix Dynamic Server, http://www.informix.com/informix/products/techbrfs/servconn/2073177/idstbrf.htm.

[KEN94] Kenan Technologies. An introduction to dimensional database technology. Kenan Systems Corporation, One Main Street, Cambridge, MA 02142, 1994. http://www.ultranet.com/~kenan/acumate/mddb.htm

[KIM95] R. Kimball and K. Strehlo. Why decision support fails and how to fix it. *ACM SIGMOD Record* 24(3), September 1995.

[KIM96] R. Kimball, The Data Warehouse Toolkit, Wiley, 1996

[LES97] H. Lenz, A. Shoshani, Summarizability in OLAP and Statistical Databases, *Statistical and Scientific Database Management*, Olympia, Washington, USA, 1997.

[MAD96] M. Madsen, Warehouse Design in the Aggregate, *Database Programming and Design*, July 1996, Vol9, No.7.

[MMS96] D. Maier, M. E. Meredith and L. Shapiro, Bringing Knowledge to Bear: Challenges for Decision Support Makers", in Multimedia, Knowledge-Based & Object Oriented Databases, Springer-Verlag, 1996.

[MEK96] M. Meredith and A. Khader, Divide and Aggregate: Designing Large Warehouses, Database Programming and Design, June 1996. http://www.dbpd.com/khader.htm

[MIC98] Microstrategy's DSS Agent, http://www.strategy.com

[MUG96] I.S. Mumick, A. Gupta, Workshop on Materialized Views: Techniques and Applications, following SIGMOD '96.

[OLA98] OLAP Council, OLAP Benchmark, http://olapcouncil.org/bmark.html

[ORA97] Oracle8(TM) Partitioning Option, An Oracle Business White Paper, June 1997 http://www.oracle.com/st/o8collateral/html/xo8pbwp4.html

[ORA98] Oracle8 Boosts Data Warehousing, Data Mart Support For Network Computing Era http://www.oracle.com/corporate/pressroom/html/PR62497.1ora8.html, June 27, 1997.

[ONE87] O'Neil, Partick E., Model 204 Architecture and Performance, Second Annual Workshop on High Performance Transaction Systems, September 1987, in Springer-Verlag LNCS Volume 359, pages 40-57.

[PLA98] Platinum Technology, Inc, INfoBeacon, http://www.wcmh.com/lantimes/96jul/607a041b.html

[RAS90] M. Rafanelli, A. Shoshani, STORM: A Statistical Object Representation Model, 5th International Conference SSDBM, Charlotte, N.C., USA, April 3-5, 1990, Proccedings. *Lecture Notes in Computer Science,* Vol. 420, Springer 1990.

[RED95] Red Brick Systems. The data warehouse: The competitive advantage for the 1990's. White paper, 485 Alberto Way, Los Gatos, CA 95032, March 1995.

[RED97] Red Brick Demonstrates Industry's First and Only Aggregate-Aware RDBMS for Data Warehouse Applications at 1997 Builder Forum User Meeting, http://www.redbrick.com/cgi-bin/nph-press/rbs-g/html/rbspress.htmf?/rbs-g/press/97-06-11-97aggr_aware

[SAL95] B. Salzberg and A. Reuter. Indexing for aggregation. Unpublished memorandum, College of Computer Science, Northwestern University, 1995.

[SHO97] A. Shoshani, OLAP and Statistical Databases: Similarities and Differences, *ACM SIGMOD Intl. Conf. on Management of Data*, San Jose, May 1995

[SQU95] C. Squire. Data extraction and transformation for the data warehouse. *Proc. ACM PODS*, Tucson, May 1997, Pages 185-196.

[STA94] D. Stacey, Replication: DB2, Oracle or Sybase, in Database Programming and Design, December 1994, Vol 7, no 12. Reprinted in SIGMOD Record, December 1995, Vol24 no 4.

[SYB98] SYBASE, 6475 Christie Ave, Emeryville, CA 94608, 800-8-SYBASE. http://www.sybase.com/products/dataware/iq.html

[THE95] L. Thé. OLAP answers tough business questions. *Datamation*, May 1, 1995.

[WLO86] H. Wong, J. Li, F. Olken, D. Rotem and L. Wong, *Bit Transposition for Very Large Scientific and Statistical Databases*, Algorithmica 1(3): Pg 289-309, 1986.

[WLO85] H. Wong, H. Liu, F. Olken, D. Rotem and L. Wong, Bit Transposed Files, *Proc. Conf. on Very Large Databases*, 1985, Pg. 448-457.