Chapter 2: Multidimensionality in Statistical, OLAP, and Scientific Databases

2.1 Introduction and Background

There is a lot of data that can be viewed as multidimensional data. The term multidimensional databases typically refers to a collection of objects, each represented as a point in a multidimensional space. Even data that is represented in a tabular form, such as relations can be thought of as multidimensional data, if each row (tuple) is thought of as an object, and the columns (attributes) are thought of as the dimensions. For example, consider the following table: Employee (personID, age, sex, salary) shown in Figure 1a. If each person is represented as a point in the multidimensional space of (age, sex, salary), then that table can be represented as in Figure 1b.

The utility of representing data in the multidimensional space is that it is more natural to view certain features of the data in this way. For example, it is natural to view clusters in the multidimensional space. In figure 1b, one can easily see that there is a small cluster of highly paid people (perhaps representing managers who are generally older) and a larger cluster of lower paid people. We can also see "outliers" as is the case with the younger person with a high salary. Of course, these concepts extends to data in more than 3 dimensions, but cannot be viewed as easily. The problem of viewing high-dimensional data to identify clusters, outliers, and various patterns, has been the subject of several research projects. An extensive review of such methods is provided in (Keim & Kriegel, 1996), and will not be discussed further here.

personID	age	sex	salary
1234	28	F	110,000
2345	56	F	150,000
3456	27	M	60,000
4567	30	M	70,000
5678	61	M	130,000
6789	28	F	80,000
7890	25	M	50,000
8901	22	M	65,000
9012	34	F	70,000
•	•	•	•
•	•	•	•
•	•	•	•

Figure 1a: An "Employee" table

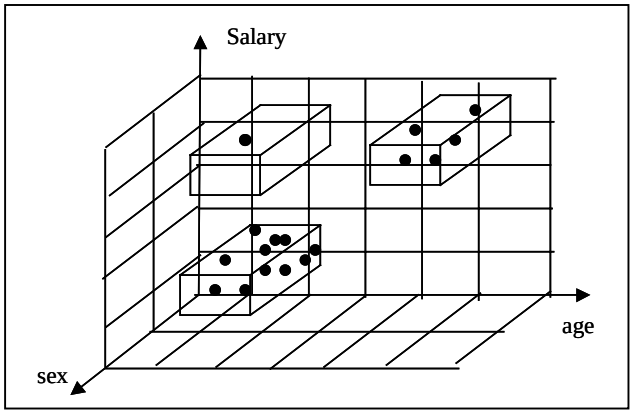

Figure 1b: A 3-D view of the table

Some data is naturally multidimensional such as 2-dimensional or 3-dimensional spatial data. For example, Climate Modelers prefer to view their observed or simulated data in a multidimensional structure representing space (2 or 3 dimensions), time, and variables being measured (temperature, wind velocity, etc.) In this case, certain operations, such a

selecting spatial regions, or performing the operation of "monthly means" on the data are very common, and need to be supported.

Another reason for viewing data in the multidimensional space is summarization. This need is most obvious in databases that represent statistical data or in databases used for decision support. These are referred to as "Statistical Databases" and "On-Line Analytical Processing" (OLAP), respectively. In the OLAP literature the multidimensional space is referred to as a "cube", and by selecting sub-ranges or summarizing over sub-ranges of the multidimensional space, one generates "sub-cubes".

In general, one can summarize over an entire dimension or over a region of the dimension. To illustrate a summarization over an entire dimension consider again the database of Figure 1. One can summarize (using the operation COUNT) over the dimension "sex" to produce the lower dimensional database: "number of employees by age by salary". This is shown in Figure 2a. Note that this summarization produced a new "summary measure". Each point in this 2D space represents now the measure: "number_of_employees". This is typical of statistical databases where the base data, called "micro-data", is summarized to form "macro-data". When only part of a range is selected or summarized over, the dimensionality of the product does not change. For example, selecting only lower paid younger people in the example of Figure 1 still produces a 3-dimensional sub-cube.

Another aspect of Statistical and OLAP databases is that each dimension can have a category hierarchy associated with it. For example, "age" can be organized as "age groups" of 1-10, 11-20, etc., and "salary" can be organized as "salary-level" of "low", "medium" and "high". In this case, summarization can take place over any one of the dimensions. This action does not reduce the dimensionality of the "cube". Figure 2b shows the result of summarization on the dimensions of Figure 2a. Dimension hierarchies can get fairly complex depending on the type of dimension. For example if one of the dimensions is "products" sold in a department store, then it can have a large number of levels in the category hierarchy.

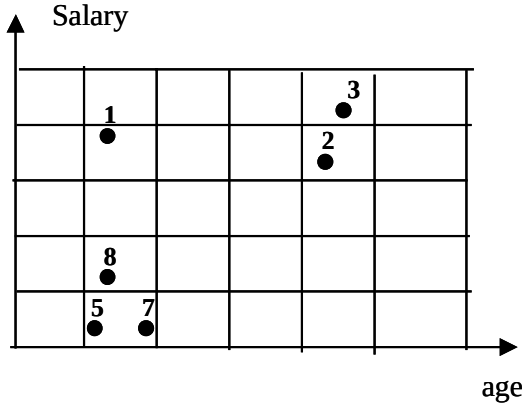

Figure 2a: A summary database in 2D space

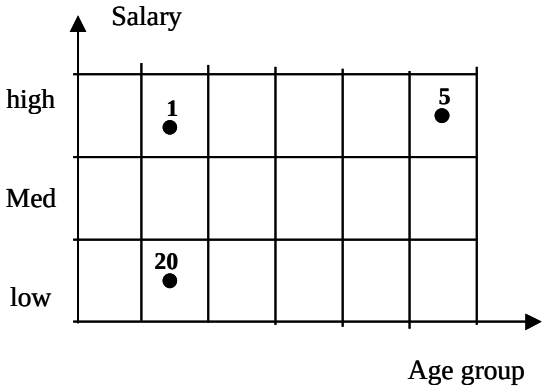

Figure 2b: The database further summarized on each dimension

In a previous paper, we identified multidimensionality as a common aspect both scientific and statistical databases (Shoshani & Wong, 1985). In this document, we elaborate on the concepts multidimensionality as well as category hierarchies, and discuss how they are used in summary and scientific databases in the next two sub-sections.

2.2 Summary Databases

Because both Statistical Databases and OLAP databases are mainly designed for summarization operations, we refer to both with the generic term "Summary Databases". The concepts of "Statistical Databases" were introduced in the 1980's (Chan & Shoshani, 1981; Shoshani, 1982) and this was followed by much activity in this area. The requirements of OLAP were introduced in a white paper by (Codd & Associates, 1993). This was followed by a paper on the Data Cube (Grey et al, 1996), which is an extension to the relational model to support OLAP databases. It was not apparent initially that both the statistical database and OLAP areas are addressing a similar problem. However, in the article (Shoshani, 1997), we compared Statistical Databases and OLAP databases in terms of their data models, data structures and operators. We found the fundamental concepts similar, but the work performed in these two domains tended to emphasize different aspects. While much of the work in the Statistical Database area emphasized conceptual modeling and formalization of operators, the OLAP area emphasized data structures and operational efficiency. However, more recently, there is more attention paid to the formal definition of the multidimensional aspects of OLAP databases and their query language; see, for example (Tsois, Karayannidis & Sellis, 2001; Cabibbo & Torlone, 1997). In this section, we discuss some of the main concepts of Summary Databases, showing parallels of the Statistical Database terminology to those of OLAP.

2.2.1 Conceptual modeling

As mentioned in the introduction, there are two structural features to summary databases: 1) the multidimensional space, and 2) the category hierarchy associated with each dimension. Each dimension (such as "sex" or "age") has a set of categorical values (or categories) associated with it (such as the "sex" categories: "male", "female", and the "age" categories: 1, 2, …, 100). The multidimensional space is simply a cross product of dimension categories. The categories of a dimension can be further grouped into a category hierarchy (such as grouping "age" into "age groups" of "1-10, …, 91-100).

Initially, each point in the multidimensional space is associated with some object identifier (such as "person_ID"), as was shown in Figure 1a. As mentioned above, this is referred to this database as the "base data" or the "micro data". Summary databases are constructed/derived from the micro data by applying a summary operator (such as "count", "sum", "average", etc.) to produce a "summary measure". This operation transforms a "micro database" into a "summary database".

Consider again the example of Figure 1, but we add "projectID" to the employee table. In the customary "relational" notation, this will be represented as: [Employee

(person_ID, age, sex, projectID, salary)]. Note that this notation alone does not represent the fact that there are functional dependencies between Person_ID and each of age, sex, projectID, and salary. If we use ":" to denote "functional dependency" this may be represented as [person_ID: age, sex, projectID, salary]. Constructing/deriving a summary database from this micro database amounts to creating a new "summary measure" (or "variable") on the multidimensional space. For example, we can choose to derive a summary database with a summary measure of "average-salary" associated with the multidimensional space (age, sex, projectID). We may use the notation for that as [(age, sex, projectID): average-salary]. That is, there is a functional dependency from the multidimensional space defined by (age, sex, projectID) to average-salary. Thus, every point in this multidimensional space has an average-salary associated with it.

Of course, the summary database can be represented as a table (a relation), such as [salary-info (age, sex, projectID, average-salary)], but the semantics are then lost. The main concept of summary databases is to capture the multidimensional space and the dependency of the summary measure on that space. Further, the concept of a category hierarchy needs to be explicitly captured. This can be done by a "->" notation, such as [age -> age-group], or [city -> state -> region]. For the example above we might group "projectID" into "project-type". Our summary database would then be represented as:
[(age -> age-group , sex, projectID -> project-type): average-salary].

The above notation captures the two main structural concepts as well as the summary measure. In the area of statistical databases a graphical notation was used to show these concepts explicitly (Shoshani, 1982). Figure 3a represents the example above in that graphical notation, where the "S" node represents a "summary measure", the "X" node represents a "cross product" of the multiple dimensions underneath it, the "C" nodes represent category classes. The categories of a "C" node may be grouped into categories of the "C" node below it. Another graphical notation preferred by some is the Universal Markup Notation (UML) where the summary measure is at the bottom of the graph and the categories are at the top (see, for example, (Pedersen, Jensen & Dyreson, 1999). This is shown for the same example in figure 3b.

Our purpose in showing the above graphical notations is to emphasize that these are various forms to represent the same concepts. In the OLAP area, relational table structures are used, but additional structure was necessary to capture the concept of a multidimensional "cube", and the category hierarchy structure. In order to represent the semantics of a "cube", a "star schema" is used, where the multidimensional cube and the summary measure are represented in a "fact table" that is placed in the center of the "star", and "dimension tables" represent dimension categories as well as the hierarchical structure of the categories. We show this tabular notation for the example above in Figure 3c. Note that without external labels to the tables there is no way of telling which is the "fact table" and which are the "dimension tables". Also note, that in the fact table there is no way of telling which is the summary measure, except from the meaning of the names of the columns. Similarly, there is no explicit way of telling in the dimension tables that there is a classification hierarchy, except by guessing from the labels of the columns. The reason for this situation is that the tables are devoid of any semantics.

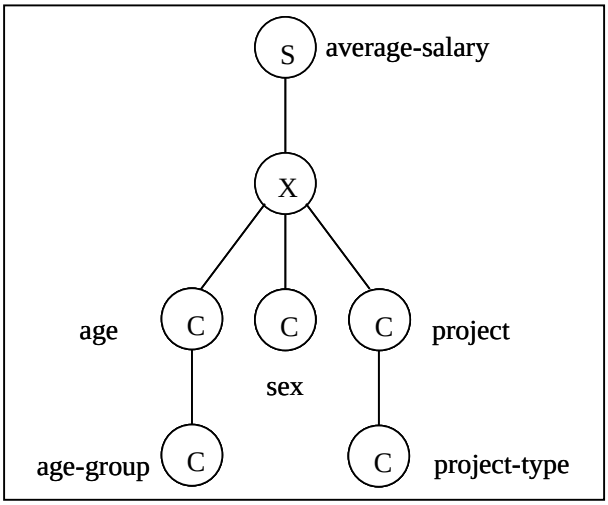

Figure 3a: A graphical representation of a summary database

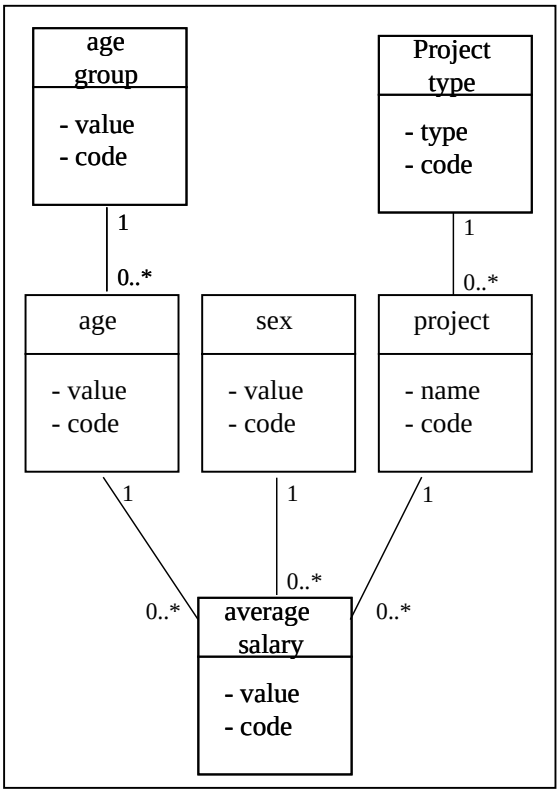

Figure 3b: An inverted graphical → representation using UML notation

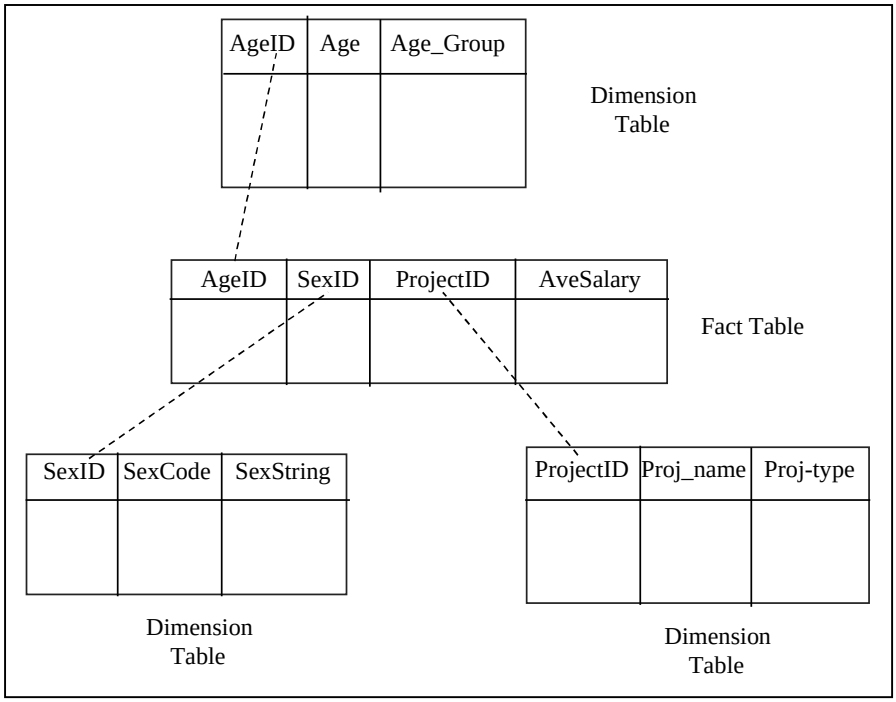

Figure 3c: A Star Schema representation of the summary database

Other graph representations used in the literature follow the Entity-Relationship notation (e.g. Tsois, Karayannidis & Sellis, 2001; Cabibbo & Torlone, 1998) or other data cube notations (e.g. Lehner, Ruf & Teschke, 1996).

2.2.2 Supporting related information

In practice, the categories involved in the summary databases may have secondary data associated with them. In the above example, a "project-ID" may have "project-description" or "project-deadline" associated with it. Furthermore, projects may belong to city departments, and the cities may have attributes such as "size" and "total-budget", and "mayor".

Where in the summary model does such information fit? One solution taken by many authors is to extend the summary model to include additional structures that support associations and even generalization/classification (e.g. Lehner, Ruf & Teschke, 1996; Sellis, 2000). Another approach is to provide for "links" between an "object" data schema and a "summary" data schema. This approach can also be used in cases where the "object" databases and the "summary" databases can reside on different systems. In this case the links cab be supported as a federation of databases using the well-known concepts of "wrappers" and a "mediator". We have investigated this approach in (Pedersen, Shoshani, Gu & Jensen, 2000), and built a prototype system on top of the ORACLE relational system and Microsoft's "OLE DB for OLAP" to illustrate its usefulness. This approach can be illustrated in the example shown in Figure 4.

In this example we show a single link between the category class "project" in the summary database and the "project" object class in the object database. This is a one-to-one link. In general, one can have multiple links set between the database schemas. Also, links can represent other association cardinalities (one-to-many, etc.), or even have attributes defined for the link instances. Once this federation is defined and the links instantiated, it is now possible to ask queries that span both databases. For the example in Figure 4, suppose that the summary database is for all the employees in a particular state. This database is maintained by the state's financial office. Suppose that another "public works" object database shown on the right is maintained by the state's public projects office. One can formulate the query "find Average-Salary by age for all projects in the in cities that have a budget greater than $10 million. This query implies that only the projects in departments that belong to cities with a budget greater than $10 million will be selected, and the results summarized over them, as well as over sex.

The linking of databases allows the separation of summary data from other related information. This can be done in a single database system or across a federation. In the example of figure 4, the separation allows each office to manage its own database, and the federation allows joint queries. For systems that implement OLAP databases as relational tables, (such as Microsoft's "OLE DB for OLAP"), "links" are implied when additional tables for the related information are added, but this requires that all the databases are managed by a single system. Also, the semantics the summary part of the tables and the other information is not explicit. The treatment of summary database

semantics explicitly as part of an "object" models is still missing from commercial products. The use of links can facilitate this capability, even if they are applied in a single database system, let alone as a way to define federations between summary and object database systems.

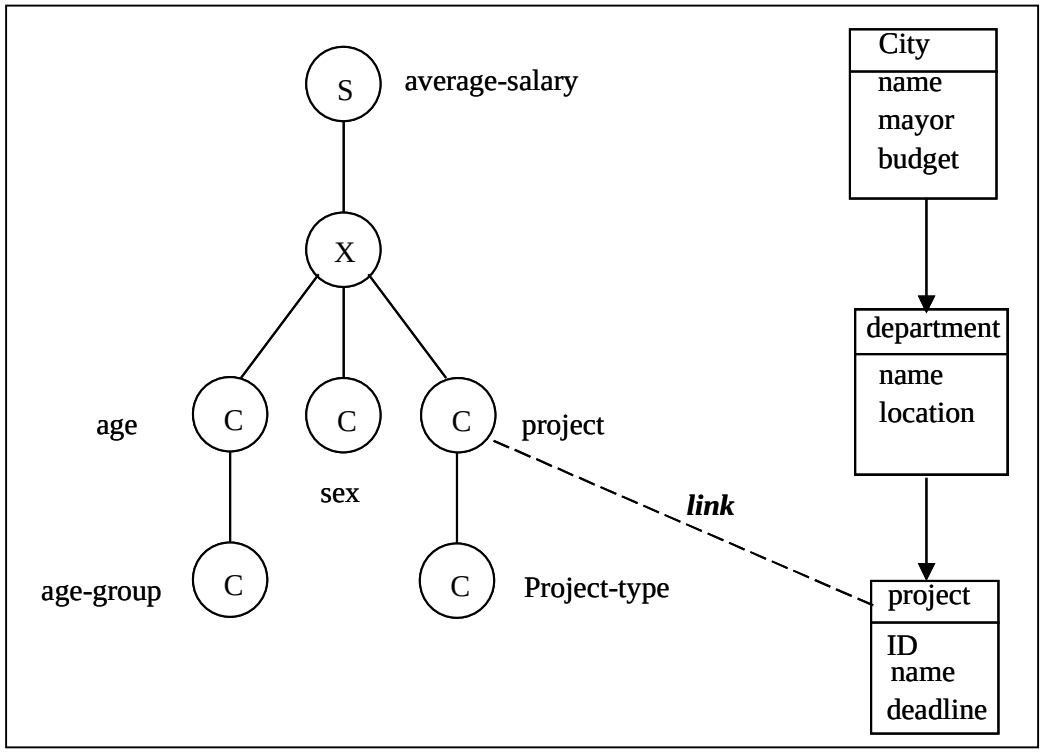

Figure 4: using links to federate "object" and "summary" databases

2.2.3 Implied aggregation

One can take advantage of the structural semantics of summary databases to imply the aggregate operations that need to be performed. This can greatly simplify the query expressions for specifying operations on summary databases. The implied aggregation is based on the multidimensional property, the category hierarchy property, and the "aggregate operator" associated with the summary measure. We discuss first the semantics of the "aggregate measure".

Once a summary database is created, the "summary measure" has an "aggregate operator" (also referred to as "summary operator") associated with it. Typical aggregate operators are "count" "sum" "maximum", "minimum", and "average". For example, the summary measure "population" has the aggregate operator "count" associated with it, and the summary measure "average-income" naturally has the aggregate operator "average" associated with it. We note that in order to support further aggregation with the "average" operator, the system have to carry (remember) the "sum" and "count" of the

base items involved. More complex operators, such as “mean” and “standard-deviation” can also be calculated by carrying the appropriate base values.

We can illustrate the concept of implied aggregation with a simple example. Consider again the summary database of figure 3. Let’s assume the database represents “average-salary-in-California”. Recall that it has the following schema:
[avg-sal-Cal (age -> age-group , sex, projectID -> project-type): average-salary],
where “avg-sal-Cal” is the name of the database.

Consider the following query applied to this database: “get average-salary for woman by project-type”. While this seems perfectly understandable to us, there are 3 implied instructions for performing this query:

1) In calculating the result, aggregate over all ages for each (sex = female, project-type) combination. This is implied because the “age” dimension was NOT mentioned at all in the query.
2) In calculating the result, aggregate over all projects that are in the same project-type. This was implied because of the category hierarchy between “projectID” and “project-type”
3) In calculating the result, use the aggregate operator “average”. This is implied because of the default operator associated with the summary measure “average-salary”.

As can be seen from this example, the implied aggregation is based on the three structural properties of summary databases: the multidimensionality, the category hierarchy and the default aggregation operator. As mentioned above this can simplify the query language. We used such a language in (Pedersen, Shoshani, Gu & Jensen, 2000). The language is called SumQL, has an SQL format, but takes advantage of the implied aggregation to simplify the query. As an example, the above query is given below as a SumQL query.

SELECT average-salary
BY_CATEGORY sex, project-type
FROM avg-sal-Cal
WHERE (sex = female)

Note that joins are not necessary, and the implied aggregation operations (“sum over ages” and “sum over projects”) are omitted.

Summarizing over category hierarchies requires that these hierarchies are well-formed. For example, aggregating “population” from cities to “states” is not well-formed, since states contain regions other that cities, such as farm-lands, or small towns. However, if we added one fictitious city called “other” to the list of cities, and associated with it all the non-cities population, then we can summarize correctly to the state level. We call the property of being able to correctly aggregate to the next higher category level “summarizability”. We studied the summarizability conditions of summary database in (Lenz & Shoshani, 1997).

2.2.4 Efficiency considerations

The main efficiency issue in summary databases is how to efficiently compute aggregate operations over the cells of the multidimensional space, and how to extract efficiently subsets of the multidimensional space. There are basically two types of aggregation operations, corresponding to the multidimensional property and the category hierarchy property. These are:

A.1) Aggregating over an entire dimension. This is referred to in the OLAP literature as a "*consolidate*" operation. For example, in the summary database [population_in_US (state, sex, age): population], aggregating over age to generate [population_in_US (state, sex): population] is a "consolidate" operation over the dimension "age".

A.2) Aggregating over a dimension to a higher level of the category hierarchy. This is referred to in the OLAP literature as a "*roll-up*" operation. For example, suppose that we have the mapping of "states" to "regions" (such as "west", "mid-west", "south", and "east"), then the summary database [population_in_US (state, sex, age): population], can be rolled-up to produce [population_in_US (region, sex, age): population].

As for selecting efficiently subsets of the multidimensional space, most of the attention in the literature was given to two types of "subset selection" operators:

S.1) Selecting a single value out of a dimension. This is referred to in the OLAP literature as a "*slice*" operation. For example, selecting only a single state (say "Alabama") for the summary database [population_in_US (state, sex, age): population], will produce [population_in_Alabama (sex, age): population]. Notice that this reduced the database from a 3-dimensional database to a 2-dimensional database, which is the intuition for the term "slice".

S.2) Selecting a range of values out of a dimension. This is referred to in the OLAP literature as a "*dice*" operation. For example, selecting only the ages 1-18 to get only children's population) from the summary database [population_in_US (state, sex, age): population], will produce the multidimensional subset (also referred to as "sub-cube") [population_of_children_in_US (state, sex, age): population], where age is limited to 1-18. In this case the dimensionality of the sub-cube is the same as the original summary database.

Note that although "slice" can be considered a special case of "dice", they may require support of different data structures and indexes, since they are analogous to performing a single value selection (where hash indexes are most effective for a single attribute) vs. a "range" selection (where tree indexes are most effective for a single attribute). However, in summary databases, multi-attribute indexes are needed for these operations.

There are other operations of interest, such as taking the union of two cubes, or disaggregating to lower levels of the category hierarchies, but most of the published works published about processing queries efficiently are primarily concerned with the

above 4 operations. As an example for such a cube-query-language (CQL) that supports the above operations and extension to them, see (Bauer & Lehner, 1997).

Summary Query Languages combine these 4 basic operations in various ways. The problem is how to efficiency compute and generate the results for ad-hoc queries involving these operators. The main idea for dealing with this problem is to materialize (pre-compute) sub-cubes. However, just considering the "consolidated" sub-cubes (the operation referred to as operation A.1), is problematic since there are 2^n-1 possible combinations of sub-cubes to consider. For example, a summary database with 3 dimensions (x,y,z) has sub-cubes with the following dimensions: (x,y,z), (x,y), (x,z), (y,z), (x), (y), (z). A seminal paper addressing this problem was (Harinarayan, Rajaraman & Ullman, 1996). Essentially, the problem they addressed can be formulated as follows: given a limited space (say 20% of the original database size), and the cardinality of each dimension, which sub-cubes should be materialized, given equal probability of all possible ad-hoc queries. They developed an analytic algorithm to determine the optimal selection of sub-cubes to materialize.

Following this work, there was a large number of works that dealt with algorithms that are sensitive to the query patterns or that are better suited for performing update operations. For example, see paper on Dynamic Data Cubes (Geffner, Agrawal & El Abbadi, 2000), which also summarizes previous work in this domain.

2.3 Scientific Multidimensional Databases

In the area of scientific databases, the aspects of multidimensionality that are of interest depend on the application areas and the purpose of organizing or viewing the data as a multidimensional dataset. Scientific databases are growing in numbers and in size at a fast pace as a result of superior instrument automation and faster parallel computers. Instruments are now capable of measuring physical phenomena with higher precision and in shorter time intervals. For example, satellites are now sending large quantities of measurements in the upper atmosphere. Similarly, runs on parallel supercomputers with thousands of processors generate large simulated datasets. These advances require better data management techniques, especially for multidimensional data. In a previous paper (Shoshani, Olken & Wong, 1984) we identified multiple problem areas common to various scientific databases. In this sub-section, we discuss in some detail four main aspects of dealing with large multidimensional data: space-time representation, clustering in multidimensional space, indexing in multidimensional space, and supporting classification structures.

2.3.1 Space-time representation

Since many scientific databases represent physical phenomena, the need to represent space-time is common in many applications. A few examples are Climate, Aerodynamics, Combustion, and Fluid Dynamics. In such applications, the typical method of conducting research is to develop theoretical models, to simulate these models,

and to compare the simulation results to real observed data. Simulation models typically partition space into some cell structure, the most common of which is a rectangular cell structure. For example, in climate simulations, a point in the cell structure represents a certain longitude, latitude, and height position (X,Y,Z). The simulations are typically run in time steps, where each time step contain a collection of X,Y,Z points, and each point has a set of tens of "variables" (such as "temperature", "wind velocity", "pressure", etc.) associated with it. These types of datasets are so common, that specialized file formats have been developed for them, such as HDF5 [NCSA/HDF5] and NetCDF (UCAR/Unidata/NetCDF).

The file organization of these datasets typically follows the way they are generated. It is a linearized representation of the multidimensional dataset, in the order of time-space-variables (i.e. the multidimensional space is linearized in the order of T,X,Y,Z, and for each point in this space the values for (V1,V2,…,Vn) are stored). Each file format representation has an access library associated with it to extract desirable subsets of the data for subsequent analysis. The main problem associated with this format organization, is that for large simulations that span multiple files, the access requires reading small pieces from many files, making the reading of a subset of data quite inefficient. For example, if a Climate analyst wishes to visualize the temperature pattern around the equator for the last 10 years, this implies that all the files containing the time steps for the last 10 years have to be accessed, and for each file the equator points have to be accessed, and for each of these points, only the "temperature" has to be extracted. Consequently, organizing the data according to the intended access pattern is desirable.

The problem of matching the file organization to the predicted access pattern has been studied (see, for example, (Chen, et al., 1995)), but in practice is not easily applied. There are two reasons for that: 1) the access patterns may suggest physical organizations that conflict with each other, or can change over time, and this requires automated reorganization of the data, and 2) for large datasets (some simulation datasets can be in the order of a few terabytes) it is very expensive to re-organize the data. Another approach is to cluster the data that is being accessed (referred to as "hot clustering"), but this approach is not in wide use, since it requires managing duplication of data. The most useful technique used currently, is to partition the simulation dataset into one variable at a time (sometimes referred to as "vertical partitioning"), so that access to a particular variable does not require accessing the other variables. This simple solution is effective for tasks that access only a few variables at a time, such as visualization.

Many physical phenomena have regions of high activity and region of little activity. For example, an ignition of fuel in a combustion cavity has regions of high turbulence close to where the fuel is injected. Similarly, a stress model of an airplane wing has more stress activity in the joints, and climate models have more activity in certain storm regions. For this reason, there are models that use "Adaptive Mesh Refinement" (AMR) models. The main idea is to use meshes that are more and more refined in the active regions, thus saving a lot of computation and space in the non-active regions. Such representation requires specialized data structures. This is a very important and active field of research in the mathematics area, but is not addressed at all by the database

research community. See (Plewa, 2001) for a comprehensive set of pointers to such work by mathematicians around the world.

Accessing space-time datasets requires the support for specialized operators. Spatial operators include the specification of time spans (e.g. summer months), partial overlap of time periods, etc. Similarly, spatial operators may include distance functions (e.g. find all the points within a given radius) and spatial containment or overlap. The support for such operations requires specialized data structures that depend on the intended application. These have been covered extensively in the literature. There are several survey papers on the subject. See (Güting, 1994; Abraham & Roddick, 1999; and Boehm, Berchtold & Keim, 2001).

We note that space-time databases also have a natural dimensionality hierarchy. The time dimension is usually used to summarize the data in order to reduce them to a degree that can be handled by an analysis or visualization tool. For example, in climate simulation datasets, it is common to summarize the variables to the level of "monthly means". Such summarization is also useful for a high level visualization of the data, which can lead to "zooming into" regions of interest. Similarly, the summarization over the spatial regions (which requires support of methods for spatial averaging) is also used in order to reduce the amount of data that the user has to deal with.

2.3.2 Clustering in multidimensional space

Given that objects are represented as points in a multidimensional space, it is natural to think about the object clusters as an indication of some common phenomenon. Similarly, an outlier (a point by itself) in the multidimensional space indicates an unusual phenomenon. Various data mining techniques are based on these observations, and numerous methods for data clustering have been proposed (see surveys by (Keim & Hinneburg, 1999) and (Jain, Murty & Flynn, 1999)). In this sub-section, we will describe an example that illustrates the importance of high dimensional clustering for very large scientific databases, and the reasons that most of the proposed clustering methods are inadequate.

Our example draws from our experience with large databases in High Energy Physics (see (Shoshani, et al., 1999), for more details). High Energy Physics experiments consist of accelerating sub-atomic particles to nearly the speed of light and forcing their collision. A small part of the particles collide and produce a large number of additional particles. Each such collision (called an "event") generates in the order of 1-10 MBs of raw data collected by a detector. A typical rate of data collection is about 10^8-10^9 events/year. Thus, the total amount of data collected is very large, in the order of 300 terabytes to 1 Peatbyte per year. This is why it is important to have efficient indexing and clustering analysis techniques.

In order to be able to find interesting clusters, summary data is extracted for each event. Each event is analyzed to determine the particles it produced and summary properties for each event are generated (such as the total energy of the event, its momentum, and

number of particles of each type). The number of summary elements extracted per event is typically quite large (100-200). Thus, the problem is one of finding clusters over a billion elements each having 100 descriptors or more. This problem can be thought of finding clusters in the 100-dimensional-space over a billion points. The total amount of space required assuming 4 bytes per value is 400 GBs.

There are two aspects to this difficult problem: size and high-dimensionality. The large size implies that the clustering analysis method must be linear to be practical. Since clusters cannot be detected in very high-dimensional space, the high-dimensionality aspect requires dimensionality reduction methods; that is, methods that select fewer dimensions that represent the clustering properties the best. However, the known techniques are either non-linear and/or they are effective for small in memory datasets. Similarly, the known linear clustering methods on large datasets are inadequate because they based on cell-partitioning methods where the dimensions to be analyzed and the binning of each dimension are pre-selected. Therefore, hybrid techniques have been proposed: running dimension reduction techniques on a sample of the data, and apply the results of this step to the linear clustering methods. We describe such a methodology in a recent paper Otoo, Shoshani & Hwang, 2001). Another effective approach reported recently, is based on using different cutting planes for each dimension, and finding the optimal grid partitioning (Hinneburg & Keim, 1999).

2.3.3. Indexing in multidimensional space

Indexing multidimensional data has been recognized as a specialized area for a long time now. If each dimension is indexed separately, then searching for objects in the multidimensional space will require the intersection of the result of all the index searches, which is an inefficient operation. Therefore, specialized multidimensional indexing methods have been developed. A recent survey covers such indexing methods (Boehm, Berchtold & Keim 2001). The specialized methods include such well known indexing structures as the R-tree (Guttman, 1984), and various improvements to it, such as the R+ tree (Sellis, Roussopoulos & Faloutsos, 1987), Grid Files (Nievergelt, Hinterberger & Sevcik, 1984), Quad-trees (Samet, 1984) and Pyramid Indexes (Berchtold, Böhm & Kriegel, 1998), just to name a few. Such indexing methods work especially well for low-dimensional applications (below 6-7 dimensions) and for queries that involve all the dimensions. However, if we need to access part of the dimensions form a high-dimensional dataset, such methods become inefficient because too many branches of the index structures are involved in the search.

As a case in point, consider the high-dimensional space described in the High Energy Physics application in the previous sub-section. Recall that we described the problem as having a very large number of objects (many millions to a billion), and the number of dimensions (or attributes) in the hundreds. The typical search query is a range query over a few of the dimensions. This is referred to as a “partial range multidimensional query”. For example, a query might be “find all objects (events) that have energy between 5 and 7 GEV and the total number of particle produced was more that 10,000”. There are two problems with using the conventional multidimensional indexing methods: the high-

dimensionality (100 or more dimensions), and the fact that the query is a "partial range query".

The solution to this problem requires another approach. We can take advantage of the fact that scientific databases are typically "append only" and to organize the index into "vertical partitions" where each partition represents a single dimension over all the objects. Further, each vertical partition can be organized as a set of compressed bitmaps. The operation of a partial range query can then be transformed into logical operations over the compressed bitmaps. We describe this technique in (Shoshani, et al., 1999) and further developed it in (Wu, Otoo, Shoshani, 2001). This discussion is illustrative of the special indexing needs of scientific high-dimensional databases; especially as such databases are growing in size.

2.3.4 Supporting Classification Structures

It is a common practice to use specialized terms to describe concepts or objects in scientific domains. If the number of such terms is large, then they are organized as "classification structures", usually hierarchies of terms. For example, in biology organisms are classified by Kingdom, Phylum, Class, Order, Family, Genus, and Species. In medicine, there is a long history of international classification of diseases, contained in a large book, and updated every 10-20 years (the latest is ICD-10). Similar large classifications exist in pharmacology, chemistry, material science, etc. These are often referred to as "ontologies". In fact, there are general products that are designed to allow the user to develop his/her classification. An example, in the area of clinical information, is an open source system, called openGALEN (http://www.opengalen.org). Hierarchical organization of concepts is not unique to scientific data. It is a natural way of classifying and categorizing concepts in any domain. For example, in the business world, products are naturally organized into hierarchical categories. This requirement is the same as the category hierarchies we discussed in statistical and OLAP databases, but can be more complex in scientific databases, since the classification structures can be many levels deep.

Scientists who need to use classification structures in their work find it extremely difficult to use commercial relational databases. This is because the tabular organization of information is not convenient for representing these classification hierarchies. To illustrate this difficulty, consider a simple example of organizing products as shown in figure 5a. In this figure, each row represents a single product as well as the higher-level categories it belongs to. The problem with this representation is that all the higher-level categories have to be repeated for all the products. This causes several difficulties. 1) For deep hierarchies, the number of columns is as large as the hierarchy depth. Even for the simple example used in Figure 5, one can imagine refinement of products to additional levels, such as "milk-products" organized into "cheese", "milk", etc. sub-categories, and each of these into "non-fat milk", "low-fat milk", etc. Thus, a 10 deep hierarchy will require 10 columns, where the values in 9 of the columns repeat for each instance of the "leaf" column. 2) This repetition is problematic not only because of space considerations, but also for data entry, and potential for errors. 3) Categories often have a

"description" item or a "code" associated with them. In the representation of Figure 5a, these have to be repeated as well.

product	product-group	product-class	product-category
banana	fruit	food	agricultural
orange	fruit	food	agricultural
apple	fruit	food	agricultural
…	…	…	…
cheese	Milk-product	food	agricultural
yogurt	Milk-product	food	agricultural
…	…	…	…
tomato	vegetable	food	agricultural
broccoli	vegetable	food	agricultural
…	…	…	…
chair	furniture	wood	manufactured
table	furniture	wood	manufactured
desk	furniture	wood	manufactured
…	…	…	…

Figure 5a: representing a category Hierarchy in a tabular form

Product-ID	product-name	product-type	product-parent
1	agricultural	category	--
2	food	class	1
10	fruit	group	2
11	vegetable	class	2
…	…	…	…
100	banana	product	10
101	orange	product	10
…	…	…	…
200	tamato	product	11
201	broccoli	product	11
…	…	…	…

Figure 5b: an alternative more compact representation

As a result of the above difficulties, an alternative, more "normalized" representation is often used, as shown in Figure 5b. In this representation each category instance at any level is represented only once, eliminating the repetition. To support the category hierarchy, each row points to its parent category, i.e. pointing to the appropriate row ID. The problem with this representation is, that while the table is more compact, operations over it are not supported by relational systems and languages, such as SQL. This is referred to as the "transitive closure" operation. For example, suppose that we wish to summarize sales for all food products in some organization. Using the table in Figure 5b, one would have to start with row 2, then for it find all the "children" by performing a "join" with the same table. This will produce rows 10, 11, …, etc. For each of these, the next level of "children" has to be found. This will then produce rows 100, 101, …, 200, 201, …, etc. Writing SQL to generate that is too difficult for the user, so the work is delegated to programmers to incorporate into special-purpose user interfaces.

We note that the "transitive closure" operation is straightforward in the case of Figure 1a. For the above example, one has only to specify "select products where product-class=food". This is the reason that many implementations end up using this representation in spite of its repetitiveness.

Finally, we should also consider the "fully normalized" alternative shown in Figure 5c. In this representation each category level is represented in its own table. An entry in one table has a pointer to an entry in its parent category table. This representation is similar to that of Figure 5b, but by splitting the category types into separate tables, it eliminates the duplication of the "product-type" column. We added in Figure 5c a "description" column for each of the category levels to show that this "normalized" representation eliminates their repetition as well. The advantage of this representation is that a "transitive closure" is expressible by specifying "joins" between the tables. The disadvantage is the need to deal with multiple tables, and the cost of performing a large number of "joins".

product-Category ID	**product-category**	**product-Category description**
PCAT-1	agricultural	farm products
PCAT-2	manufactured	Factory products
…	…	…

product-Class ID	**product-Class**	**product-Class description**	**product-Category ID**
PCLA-1	food	retail	PCAT-1
PCLA-2	animal feed	wholesale	PCAT-1
PCLA-3	wood	mahogany	PCAT-2
…	…	…	…

product-group ID	**product-group**	**product-group description**	**product-Class ID**
PGRP-1	fruit	organic	PCLA-1
PGRP-2	vegetable	organic	PCLA-1
…	…	…	…
PGRP-100	furniture	Hand-made	PCLA-3
…	…	…	…

product-ID	**product-**	**product-description**	**product-group ID**
PRD-1	banana	large	PGRP-1
PRD-2	orange	large	PGRP-1
…	…	…	…
PRD-200	chair	kitchen	PGRP-100
…	…	…	…

Figure 5c: A normalized version of the category hierarchy

Our purpose in discussing these alternatives of tabular representations of category hierarchies is to show why relational databases are rarely used for such structures, or used in a limited way. It points to a need to support such structures as specialized objects. Actually, this problem is more complex in real systems, since the hierarchy structures are not always as well organized as our example implies. In reality, there are cases where a category of one level may point to a parent category two or more levels above it. A recent paper that addresses this problem is (Pedersen, Jensen & Dyreson, 1999).

Another aspect of classification structures that makes them even more complex to support in a tabular form, is that a collection of objects can be classified in more than one category hierarchies. For example, the product category hierarchy of Figure 5 could be classified with a category hierarchy "manufacturer" where manufacturer could be further organized by location (city, state). Thus, the same base objects (in our case, "products") can be classified according several properties they may have. These properties that can be used to classify a collection of objects are referred to as "facets" (Batty, 1998). A database system that supports category hierarchies should permit the search the objects with category specifications in multiple facets. This is equivalent to having a multidimensional space of facets, where each facet can be represented as a category hierarchy.

2.4 Future trends

It should be evident form the above discussion that the two concepts of multidimensionality and category hierarchies are fundamental to summary and scientific data. In fact, it is our belief that they are fundamental to any domain that requires classification of objects or concepts into manageable collections. Further, since any collection of objects can have multiple "facets" (or attributes), these facets define a multidimensional space. There are two implications to these observations: the need for explicit conceptual modeling of these concepts, and efficient data structure to support these two fundamental concepts.

As far as conceptual modeling is concerned, it is necessary to support multidimensional classes and a category-hierarchy classes as first class concepts. By "first class concepts" we mean that they are visible to the user. In the area of object modeling the two dominant concepts are "object classes" (which represent a collection of objects, such as "products"), and "associations" (which represent the associations between object classes, such as the association between "products" and "manufacturers"). Object classes have attributes associated with them (such as "product-cost" or "product-weight"). We argue here that these have to be complemented with the concepts of multidimensional classes, and category-hierarchy classes. For example, it should be possible to define a "product-type" as a category-hierarchy class and associate the object class "products" with it. This is shown schematically in Figure 6a. Furthermore, it should be possible to define a multidimensional class with two dimensions, one being "product-type" and one being "manufacturer" and associate this multidimensional class with the "product" object class. This is shown schematically in Figure 6b.

The technology to support efficient data structures for the multidimensional class and the category-hierarchy class may vary depending on the application. For high-dimensional classes a special index may be necessary, for summary databases an efficient summarization technique may be necessary, and for very rich and deep category hierarchies a specialized software package may be necessary. The main issue here is what is the infrastructure that can support such diversity. Extending the relational paradigm, the "object-relational" approach permits plugging "data blades" or "data

containers" to support specialized data types. This approach was adopted by commercial vendors, especially, Informix and ORACLE. However, this approach provides a table-driven view of the conceptual model. Furthermore, this approach does not permit one data type to be built from other data types. This will require a "data blade" to use other "data blades" as building blocks.

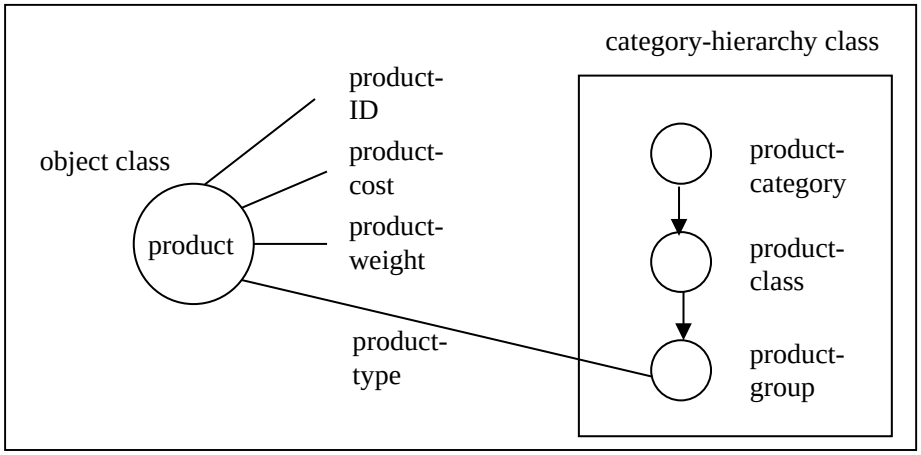

Figure 6a: A schematic of an object model for supporting category-hierarchy class

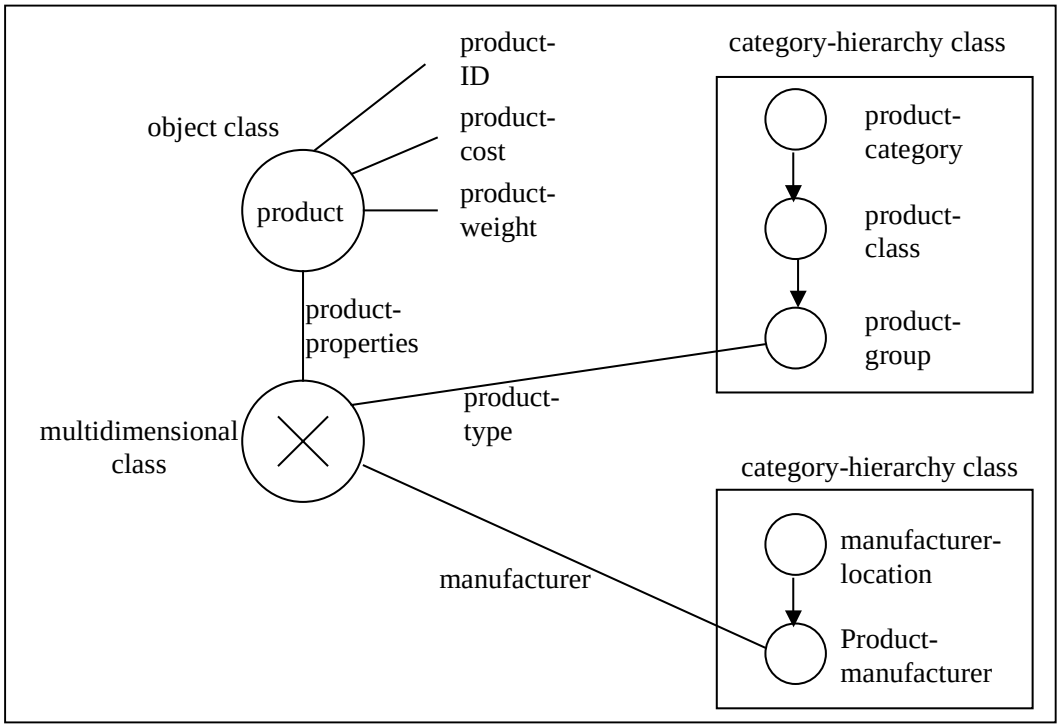

Figure 6b: a schematic of an object model to support a multidimensional class whose dimensions are two category-hierarchy classes.

We believe that the concepts of federation technology and the ability to associate object classes using "links" is a more flexible approach. Federated databases should permit putting together sub-systems that support the object-based data model with various other sub-systems as needed, so that the external conceptual model can support richer classes of objects and operators over them. In particular, we advocate a federated architecture to support sub-systems for object, multidimensional, and category-hierarchy classes. This architecture could exist under a single database system, or could be applied to federate multiple database sub-systems.

References

Abraham, A., Roddick, J. F. (1999), Survey of Spatio-Temporal Databases, International Journal on Advances of Computer Science for Geographic Information Systems (GeoInformatica) 3(1), 61-99.

Batty, D. (1998), WWW -- Wealth, Weariness or Waste: Controlled vocabulary and thesauri in support of online information access, D-lib magazine, The magazine of Digital Library Research.

Bauer, A., Lehner, W. (1997) The Cube-Query-Languages (CQL) for Multidimensional Statistical and Scientific Database Systems, International Conference on Database Systems for Advanced Applications (DASFAA), 263-272.

Berchtold, S., Böhm, C., Kriegel, H-P. (1998), The Pyramid-Technique: Towards Breaking the Curse of Dimensionality, International Conference on Management of Data (SIGMOD), 142-153.

Boehm, C., Berchtold, S., and Keim, D. A. (2001), Searching in High-dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases, ACM Computing Surveys, Volume 33.

Cabibbo, L., Torlone, R. (1997),Querying Multidimensional Databases, International Workshop on Database Programming Languages (DBPL), Lecture Notes in Computer Science, Springer-Verlag, 319-335.

Cabibbo, L., Torlone, R. (1998), A Logical Approach to Multidimensional Databases, International Conference on Extending Database Technology (EDBT), 183-197.

Chan, P., Shoshani, A.(1981), Subject: A Directory driven System for Organizing and Accessing Large Statistical Databases, International Conference on Very Large Data Bases (VLDB), 553-563.

Chen, L. T., Drach, R., Keating, M., Louis, S., Rotem, D., Shoshani, A., (1995), Efficient organization and access of multi-dimensional datasets on tertiary storage systems, Information Systems Journal, 20(2), 155-183.

Codd, E. F., & Associates (1993), Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate, white paper, commissioned by Arbor Software (now Hyperion Solutions).

Gray, J., Bosworth, A., Layman, A., Pirahesh H. (1996), Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total, International Conference on Data Engineering (ICDE), 152-159.

Geffner, S., Agrawal, A., Abbadi A. (2000), The Dynamic Data Cube, International Conference on Extending Database Technology (EDBT), 237-253.

Güting, R. H. (1994), An Introduction to Spatial Database Systems, VLDB Journal 3(4), 357-399.

Guttman, A. (1984), R-Trees: A Dynamic Index Structure for Spatial Searching. International Conference on Management of Data (SIGMOD), 47-57.

Harinarayan, V., Rajaraman, A., Ullman J. D., (1996), Implementing Data Cubes Efficiently, International Conference on Management of Data (SIGMOD), 205-216.

Hinneburg, A., Keim, D. A. (1999), Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering, International Conference on Very Large Data Bases (VLDB), 506-517.

Jain, A. K., Murty, M. N., Flynn, P. J. (1999), Data Clustering: A Review. ACM Computing Surveys, 31(3), 264-323.

Keim, D. A., Hinneburg, A. (1999), Clustering Techniques for Large Data Sets - from the Past to the Future, International Conference on Knowledge Discovery and Data Mining (KDD), Tutorial Notes, 141-181.

Keim, D. A., Kriegel, H-P. (1996), Visualization Techniques for Mining Large Databases: A Comparison, IEEE Transactions on Knowledge and Data Engineering (TKDE), 8(6), 923-938.

Lehner, W., Ruf, T., Teschke, M. (1996), CROSS-DB: A Feature-Extended Multidimensional Data Model for Statistical and Scientific Databases, International Conference on Information and Knowledge Management (CIKM), 253-260.

Lenz, H-J., Shoshani, A., (1997) Summarizability in OLAP and Statistical Data Bases, International Conference on Scientific and Statistical Database Management (SSDBM'00), 132-143 (http://www.lbl.gov/~arie/papers/summarizability.SSDBM97.ps)

Nievergelt, J., Hinterberger, H., Sevcik, H. C. (1984), The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems (TODS) 9(1), 38-71.

National Center for Supercomputing Applications (NCSA)'s HDF5 home page: http://hdf.ncsa.uiuc.edu/HDF5.

Otoo, E. J., Shoshani, A., Hwang, S-W. (2001), Clustering High Dimensional Massive Scientific Datasets, International Conference on Scientific and Statistical Database Management (SSDBM), 147-157.

Pedersen, T. B., Jensen, C. S., Dyreson, C. E. (1999), Extending Practical Pre-Aggregation in On-Line Analytical Processing, International Conference on Very Large Databases (VLDB99), 663-674.

Pedersen, T. B., Shoshani, A., Gu, J., Jensen, C. J. (2000), Extending OLAP Querying to External Object Databases, International Conference on Information and Knowledge Management (CIKM), 405-413.

Plewa, T. (2001), Compilation of Sources for Adaptive Mesh Refinement for Structured Grids, http://www.camk.edu.pl/~tomek/AMRA/amr.html.

Samet, H. (1984), The Quadtree and Related Hierarchical Data Structures. ACM Computing Surveys 16(2), 187-260.

Sellis, T., Roussopoulos, N., Faloutsos, C. (1987), The R+-Tree: A Dynamic Index for Multidimensional Objects, International Conference on Very Large Data Bases (VLDB), 507-518.

Shoshani, A. (1982), Statistical Databases: Characteristics, Problems, and some Solutions, International Conference on Very Large Data Bases (VLDB), 208-222.

Shoshani, A.,Wong, K. T. (1985), Statistical and Scientific Database Issues. IEEE Transactions on Software Engineering (TSE) 11(10), 1040-1047.

Shoshani, A., Olken, F., Wong, K. T. (1984), Characteristics of Scientific Databases, International Conference on Very Large Data Bases (VLDB), 147-160.

Shoshani, A. (1997), OLAP and Statistical Databases: Similarities and Differences, Symposium on Principles of Database Systems (PODS), 185-196.

Shoshani, A. Bernardo, L. M., Nordberg, H., Rotem, D., Sim, A. (1999) Multidimensional Indexing and Query Coordination for Tertiary Storage Management, International Conference on Scientific and Statistical Database Management (SSDBM), 214-225.

Tsois, A., Karayannidis, N., Sellis, T. K. (2001), MAC: Conceptual data modeling for OLAP, International workshop on Design and Management of Data Warehouses (DMDW).

University Corporation for Atmospheric Research (UCAR), Unidata's NetCDF home page: http://www.unidata.ucar.edu/packages/netcdf.

Wu, K., Otoo, E. J., Shoshani, A. (2001), A Performance Comparison of bitmap indexes, International Conference on Information and Knowledge Management (CIKM), 559-561.