LBL-29900

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

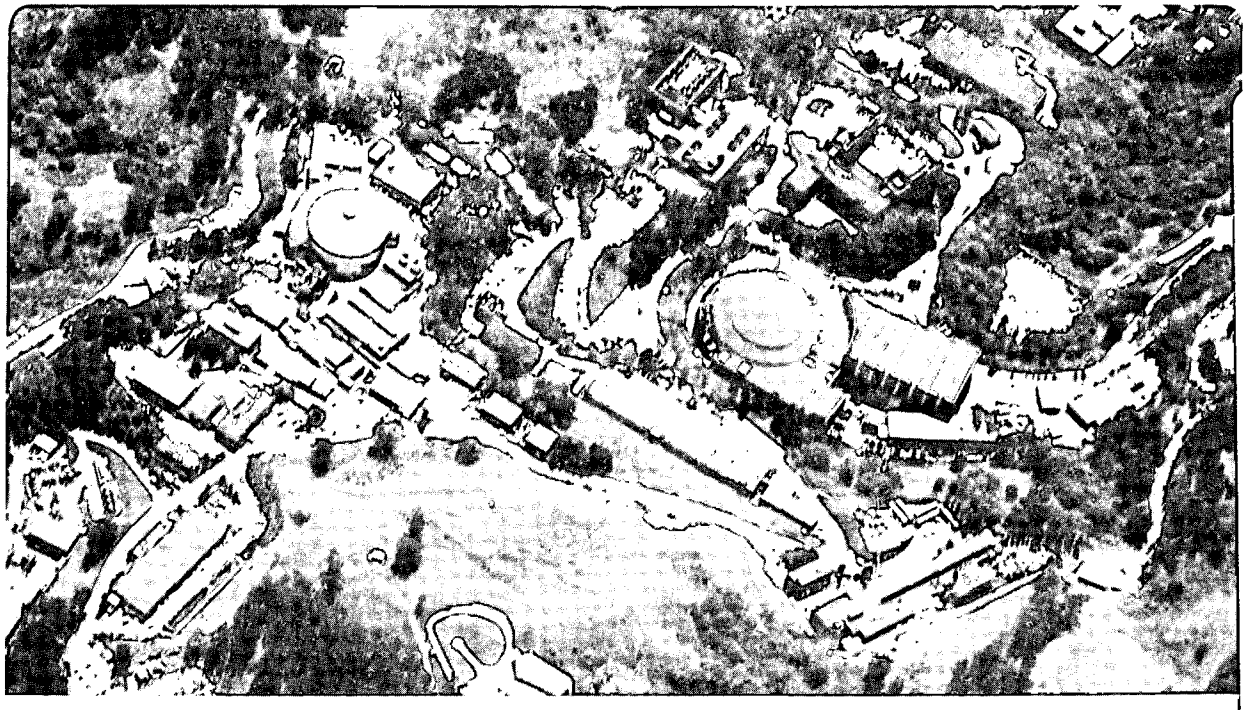
Information and Computing Sciences Division

To be published as a chapter in **Statistical and Scientific Databases**, Z. Michalewicz, Ed., Ellis Horwood Ltd., Publisher, Chichester, England, February 1991

Properties of Statistical and Scientific Databases

A. Shoshani

November 1990



Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098.

DISCLAIMER

LBL-29900

Properties of Statistical and Scientific Databases

Arie Shoshani

Computing Science Research & Development
Information & Computing Sciences Division
Lawrence Berkeley Laboratory
1 Cyclotron Road
Berkeley, California 94720

November 1990

To appear as a chapter in a book entitled
"Statistical and Scientific Databases" by Ellis Horwood Ltd.

This work was supported by the Director, Office of Energy Research, Applied Mathematics Sciences Research Program of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

PROPERTIES OF STATISTICAL AND SCIENTIFIC DATABASES

Arie Shoshani

Information and Computing Sciences Division
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

Abstract

This paper is intended as an introduction to the properties and requirenments of Statistical and Scientific Databases (SSDBs). It discusses the inadequacy of current commercial data management systems for SSDB applications. The paper also contains a discussion of new approaches for the support of complex applications and their implications to SSDB applications. It concludes with observations of future research areas.

1. Introduction

A large number of Statistical and Scientific Databases (SSDBs) are not managed today by data management systems, whether commercial systems or special purpose laboratory systems. Such data include collected statistics by various government agencies (such as health data or business pattern data), and large datasets which result from scientific experiments and simulations (such as climate measurements and climate model simulations). Typically, such databases are stored as files which are then managed and processed by special purpose programs written especially for each of these applications.

The main reason for this state of affairs is that most of the commercial systems available today were designed primarily to support business applications (such as marketing and banking). SSDBs have data characteristics and access requirements that need different tools than currently offered by commercial systems. The different needs are discussed further in the next section. However, the following example can provide some intuition as to the complexity of typical scientific applications.

Consider a particle physics experiment, where particles are collided in order to generate sub-particles. Such collisions can occur millions of times in the course of an experiment, and the results captured by special detectors. The data from the detectors passes many stages of analysis. The first few stages reconstruct the tracks of sub-particles produced. Eventually, collections of tracks for certain sub-particles are analyzed statistically in order to characterize the collisions. In general, scientific data have more complexity than strictly statistical data (such as survey data), because in addition to the measured data, it contains data about the instruments, the environment of the experiment, and the configuration of the experiment. Configuration data can be quite complex, as is the case, for example, in describing a wing configuration for an airplane simulation.

From the example above, one can observe that the data structures of SSDBs can be quite complex: there are temporal sequences of measurements, there are measurement which represent points in space (spatial data), the configuration of the detector represents graphic data (such as contours) and images, and there are various statistical summaries generated in the process of analysis. In terms of operations over the data, searches that have spatial and temporal locality are common, as well as statistical operators in multi-dimensional space.

Research in data management for SSDBs has been going on for about a decade, as exemplified by the series of workshops held approximately every two years since 1981 [1-5]. However, the support for such research has been sporadic. More recently, there is a renewed interest in this topic, because of the realization that large projects such as environmental cleanup, climate modeling, human genome mapping and sequencing, super-computer super collider (SSC), etc., have already, and will continue to generate vast amounts of data at multiple physical locations. The management and coordination of these large datasets can indeed benefit from scientific data management tools. This growing interest has been reflected in a recent workshop sponsored by NSF [6].

The amount of scientific data generated is accelerated due to more sophisticated devices and cheaper computer power. A single scientific experiment can generate hundreds of megabytes of data within days. Many scientific simulations are not carried out to the desired granularity level, only because it is impractical to process and manage the large amounts of data that would be generated. Many practical databases collected for statistical purposes, such as trade data between countries, or the various census data, are too complex to be managed with conventional data management techniques efficiently. In addition, there are large amounts of data that were not collected originally for experimental or statistical purposes, but have tremendous potential when used for statistical purposes. For example, routine patient records in hospitals

can be used for statistical "cause and effect" studies. Business transactions can be statistically analyzed for policy setting and econometric models. For the most part, such sources of routine collections of data are left unused because adequate data management facilities do not exist.

The purpose of this paper is to provide an overview of the properties of SSDBs. In section II we discuss the main differences between the properties of SSDBs and conventional (business) data. Section III contains a discussion of new emerging technologies, and their implications to the management of SSDBs. sections IV and V discuss in more detail properties of statistical and scientific databases, respectively. These sections contain parts of previous articles [7-8]. The last section contains a summary and conclusions on future research in the area.

2. Additional Requirements of SSDBs

As mentioned in the introduction, SSDBs have needs that are not adequately supported by commercial data management systems. To see what are the additional requirements, it is worth mentioning briefly the main advantages offered by data management systems in general. This will be followed by a discussion of the additional requirements.

Data management technology has been used successfully in many applications (mostly business oriented) because they offer the following main advantages:

1) data models which provide an abstraction for representing the structure and semantics of the data,

2) high-level query languages defined over the data models for accessing and manipulating the data,

3) support for concurrent access of the data by multiple users (concurrency control),

4) mechanisms for expressing and validating the integrity of the data,

5) back up and recovery of the database as protection from system failures, and

6) support for efficient physical organization in terms of storage on and access from secondary and tertiary storage.

In general the same advantages that are useful for business applications are also useful for SSDBs. However, for each of the points above there are specific additional or different requirements that SSDBs have. A brief discussion of each of the points above follows.

1) The data model

The data model used in modern commercial data management systems is the relational model. It consists of multiple tables (called "relations") that can be linked explicitly in the query language by associating columns (called a "join" operation). The model is set oriented, that is, the order of instances (rows of a table) is not enforced. For SSDBs this simple table structure is inadequate. For example, ordered structures, such as DNA sequences or temporal sequences, cannot be directly represented. Other data types, such as vectors, matrices, etc. also cannot be expressed by relations in a straightforward and convenient way. SSDBs often include graphs, images, and complex tables (e.g. isotope tables). Such needs are not unique to SSDBs, since they often exist in business data, but they are more pronounced in SSDBs.

Another important requirement is the ability to represent complex objects. Consider the simple example of a contour in 2-dimensional space. It can be thought of as a circular sequence of points, where each point is composed of X and Y coordinates. Each point is a complex object made of two labeled values, and the sequence is a complex object that orders the points and has the property "circular". One can further iterate to define objects that are sets of contours, sequences of contours, etc. This property can be extended to model images, sequences of images (video), data forms, etc. Complex object structures are prevalent in SSDBs. Their representation in terms of relational tables is quite awkward, complex, and unnatural. Thus, data models that can support complex structures directly and naturally are needed for SSDBs. Current approaches to such models are discussed further in the next section.

2) The query language

SQL (Structured Query Language) has become a de-facto standard as the high level query language for the relational model. The basic functionality is provided by constructs for specifying predicates to select rows from relational tables, specifying how multiple tables can be associated ("joined") in a single query, and which columns should be included in the output. Additional features of the language include aggregate functions (such as "count", or "sum") over groups of values, set operators ("union", "difference"), and sorting of the output. SQL has been shown to be a fairly powerful language for expressing data manipulation of table

structures, although there is continuing criticism of its semantic clarity and power.

The obvious corollary from the structural needs of SSDBs discussed in the previous section is that SQL is not powerful enough for SSDBs. For example, if a model for SSDBs supports a sequence construct, then it will imply operators to search such a structure, such as "find a subsequence of length k starting in position n". In general, complex objects are composed of multiple structures, and for each structure there is a set of operators for its search and manipulation.

SSDB applications are often based on different disciplines, such as biology, earth sciences, physics, etc. Accordingly, their requirements can be highly specialized. Suppose, for example, that a "sequence" construct exists in a certain complex data model. This sequence structures could be used to represent seismic events in time, text sequences, DNA sequences, etc. In general, in each of these domains, different operators will make sense. For seismic sequences we may be interested to find if an event occurred at a particular range of time, for text sequences we may be interested in finding words with certain proximity to each other, and for DNA sequences we may want to find overlapping regions between them. Can such complex operations be supported by an underlying high level complex object language? This is indeed the subject of current research. Current approaches will be discussed in the next section.

3) Concurrent access

Support for concurrent access of data for business applications revolves around the concept of a transaction. The paradigm used is that multiple transactions may be applied to the same data item(s) and thus interfere with each other when at least one transaction updates the data. For business applications, such as banking, accounting, or reservation systems, such interference may be disastrous. Thus, most of the work in the past concentrated on coordinating the access of concurrent transactions so that the global behaviour will be equivalent to a serial execution of the transactions.

In SSDBs concurrent access to the data is not an important issue. First, much of the data is not updated, as is the case with data that was generated by scientific devices, such as those used in weather monitoring. Second, if data is updated, it is often done by a single person, as is the case when an analyst removes or corrects "outliers" (data that is outside expected bounds). In such a case, the corrections do not have to be immediately visible, and thus can be made on a locked version of the database. Third, when multiple users need to update the data simultaneously, they often need to access different parts of the database, such as multiple

engineers working on different sections of a joint design. Fourth, the concept of a "long transaction" is often more appropriate for SSDBs, where a scientist may embark on a long analysis (which results in updates). It is not always possible to break such long transactions into a series of short transactions.

For the above reasons, concurrency control, as supported by current commercial systems, is not sufficient. There is a need to support multiple versions of datasets, and to keep track of the correspondence between them. In addition, there is the need to support a master version, where different parts of it are modified by different users. Support long transactions is another important requirement. Finally, for the massive amounts of data that are only read (not modified), there is no need to pay the overhead of concurrency control. As will be discussed below, there is also a benefit to using data structures especially designed for static data.

4) Integrity constraints

Maintaining the integrity of data can be quite complex. Most commercial systems support a simple form of data type integrity, such as checking that values would not exceed given boundaries, or that values would be only in a certain format. However, commercial systems are quite limited, especially for SSDB applications. Even a simple feature, often needed in SSDBs, which checks that data values are only from a given list of values (called "categorical" values), is not always supported by commercial systems. Integrity conditions become more complex to maintain when they involve multiple domains, such as checking that a person's birthdate is smaller by at least 18 years than his/her employment start date (assuming adult employment). In general, integrity conditions may require the power of a programming language for their expression.

In relational systems, integrity conditions can involve data elements from multiple tables. One form of multi-table integrity condition, called "referential integrity" has recently attracted much attention. For example, a "car" table may contain a column for the person who owns it. In order to enforce the condition that every car must have an owner, a referential integrity condition between the relevant columns of the tables must be defined and maintained. This creates problems for SSDBs. It is a tedious and non-trivial task to specify such conditions for complex schemas that contain complex objects. In current systems they need to be expressed with complex constructs, such as "triggers" (e.g. Sybase) or "rules" (e.g. Ingres). If complex objects can be expressed directly in the model, the referential integrity conditions that hold between the components of the complex structures could be inferred, and do not have to be expressed explicitly by the user who specifies the database structure. In general, richer models

for SSDBs should capture more semantics, and therefore the integrity constraints implied by these semantics. This would save the user the chore of expressing them explicitly.

5) Recovery

Mechanisms for back up and recovery are always needed. In conventional applications where transactions are short, it is quite acceptable to back up to the last successful transaction for the purpose of recovery. For SSDBs, where long transactions are more typical, backing up to the last transaction is not acceptable, because too much work may be lost. new mechanisms need to be developed for the support of long transactions.

6) Physical database organization

Physical data structures and access methods are the key to efficient support of queries. Relational database implementations typically organize rows of relations as records in files, and provide additional index mechanisms (e.g. B-trees, hash tables) over the various columns of the relations. SSDB applications need additional types of data organization and access algorithms. For SSDBs the best physical clustering of the data may not be according to records in a file. The main reason is that in SSDB applications the access requirements would benefit from other ways of clustering data. For example, spatial applications typically have local access operators (such as "find neighboring points" for calculations of mesh data). For such applications, physical locality of data according to their spatial locality would reduce the amount of I/O from secondary storage.

Another example, is sparsity of data in multi-dimensional space. The best method for compressing the data has to be weighted against efficient access, and thus depends on the SSDB application's access requirements. Various methods, such as "grid files", "quad trees", etc. were proposed in the literature but are not yet available in commercial system.

Statistical databases often need to access a few columns from a table for performing statistical correlations. The typical "row-wise" organization is quite inefficient for such applications, because entire records have to be read in order to read a few values out of them. The more efficient organization for such applications, is a "column-wise" organization (called "transposed files'). No commercial system currently offers this option of data organization. Finally, it is worth noting that complex objects have a natural clustering of their components. For applications that need to access entire complex objects, "object-wise" clustering would be best.

As can be seen from the above examples, data management systems for SSDBs need to have a rich set of physical organization options to provide efficient performance. A multiplicity of choices makes the problem of query optimization (i.e. the algorithm for the most efficient way to execute a query) so much harder. The management of these options and the way they can interact are a great challenge to designers of such systems.

3. Current Approaches and Their Relationship to SSDBs

There are several new promising approaches that could have an effect on the future management of SSDBs. These approaches were not specifically developed for SSDBs, but for complex database applications in general. However, the capabilities they are designed to provide can go a long way towards supporting SSDB applications. In general, these approaches strive to provide the following capabilities:

1) support for complex objects, and operations over them,

2) support for user defined structures and functions, and

3) the capability to incorporate new physical data structures and access methods.

It follows from the discussion and examples in the previous section, that such capabilities are indeed necessary for the efficient support of SSDBs. The approaches we discuss below are fundamentally different, thus emphasizing different aspects of the capabilities above.

3.1 Object-oriented database systems

The object-oriented methodology was developed in the context of programming languages. This methodology has been found to be very useful for system design [9]. The two main features of this methodology are encapsulation and inheritance. Encapsulation consists of the ability to define data structures and operators over them (also called "methods"), and make them available to users through interfaces. The objective is to hide the details of the implementation of the data structures and operations and make only their interfaces visible. Accordingly, object structures in SSDBs, such as a "seismic event", could be encapsulated, together with specialized operators, such as "find other events that correlate with this event".

The usefulness of this concept to SSDBs is in the power to code new complex structures and operations, and make them callable by name. For example, a DNA sequence could be coded as a linked list structure, and the operation "overlap" could be coded to return the sub-

sequences that overlap between two given DNA sequences. Both the DNA structure and the overlap operation will then be recognized by the system, and could be invoked by name.

The second feature that makes this approach attractive is inheritance. this feature is related to the concept of "generalization", which has been incorporated in many semantic database models. Like generalization, it supports the construct of classes and subclasses (e.g. a "student" is a subclass of the class "person"), and provides the inheritance of its properties (e.g. the "age" of the "person" is inherited by the "student"). However, object-oriented inheritance includes the inheritance of the operators (methods) as well. This feature provides the capability to share code and to make explicit the inheritance association between objects.

Object-Oriented Database Management Systems (OODBMSs) have additional requirements [10]. Support for encapsulation and inheritance is already provided by object-oriented programming languages, such as Object-Pascal or C++. In order to support database applications these languages need to provide persistency to the data, i.e., that the data will exist across multiple executions of the program. In addition, other features need to be added, such as those mentioned in the previous section (concurrency, recovery, physical organization, query language, etc.) The approach to the implementation of such systems is typically to start with some object-oriented programming language and to develop the above features with it.

The obvious difficulties with this approach is that there are no constructs beyond the programming language supported by the OOBDMS for defining complex data structures and operators. This has two drawbacks. First, there are no system supported building blocks to define new structures. For example, it would have been simpler to define a DNA structure if a "sequence structure" and its associated operators were available. This can be remedied by developing software libraries. However, the second drawback is more severe: there are no mechanisms to make the new objects and structures part of some high level query language. They are only callable from a program. Further, even if a query language exists, and even if it can be extended with the new operators, there needs to be an additional component that provides query optimization for the new objects. This requires that the query optimization algorithm will be capable of dynamically adjusting according to the properties of the new data structures and operators. This is a challenging task that will determine the performance of OODBMSs.

3.2 Extensions to the relational model

It is natural to take the approach of extending a popular database model, such as the relational model. If successful, it has the advantage that relational technology has matured into

commercial products, and its concepts are well known. Initially, there were various attempts to extend the relational data structures and relational query languages (such as permitting relations made of relations). In subsequent work, systems were designed to include user defined data structures and operations. Examples of such developments are Postgres [11], and the system described in [12].

The typical approach in such extensions, is to define a complex relational model, and a query language for it. the extensions include, inheritance structures, new data types (e.g., vectors, matrices, images, etc., and even procedures), and a capability to construct complex relations from other relations. Further, the systems are designed to be extendible, in that the user can define new structures and operations in terms of data structures and operators provided by the system. The user defined operations are callable by name from the query language.

The advantage of this approach is that it provides a capability to define complex data structures, and the incorporation of user defined structures and functions. However, it is not clear whether the approach of extending an existing model is natural for SSDB applications. Experience will show whether a conceptual model at a more abstract level will be more useful.

Another difficulty with this approach is similar to the one discussed in the context of OODBMSs; that is, the need for the query optimizer to be dynamically modified when new data structures and operators are introduced. The main difficulty is whether the information for the query optimizer can be specified by the scientific database designer, or whether this will require the expertise of a specialist.

3.3 Extensible database systems

Extensible database systems are based on the premise that a single system cannot be designed to answer the complex needs of various applications. Thus, they are designed to customize specific data management system for each application. The main idea is one of using "building blocks", that is, reusable software that can be selected for the need of a particular application. Additional "blocks" can be provide by users, and the selected blocks are then assembled (compiled) to produce a special purpose system customized for the application. The key to the success of this approach is the design of interfaces (the "glue") that permit the interchangeability of modules and the integration of new modules. Examples of such systems are Exodus [13] and Genesis [14].

Obviously, this approach should be quite attractive for scientific database applications. However, the designer of such a customized system has to have sufficient expertise in the

implementations of DBMSs. Thus, this approach may be considered more appropriate to a software house that can customize a DBMS for the needs of the application.

Extensible database systems have to address explicitly the issue of incorporating new operators, new query languages, and new physical data structures and access methods. While these can be introduced as "plug compatible" modules, there is still the problem of having to modify the query optimizer accordingly. One of the proposed solutions is to use rule-based optimizers to specify the optimization algorithm.

Implications to SSDB research

The above methodologies show great promise for helping in the development of SSDB data management systems. They do not, however, provide any insight as to the properties and needs of SSDB applications. There is a continuing challenge of characterizing the properties of various applications, understanding the modeling requirements, the operators, the access patterns of the application, and the physical structures to support them. For example, applications that involve temporal and spatial data need to be characterized in terms of sequence and multi-dimensional structures, their operators defined, and efficient physical structures for their support developed.

The above tasks are especially difficult to achieve for scientific applications because of the complexity of the disciplines. It requires the collaboration of computer scientists (data management specialists) and scientists from other disciplines. To understand fully the needs of these disciplines in terms of data modeling, operators, and physical structures, requires long and dedicated interactions. The next two sections contain some observations of the properties of statistical and scientific databases.

4. Properties of Statistical Databases

Statistical data bases (SDBs) can be described in terms of the type of data they contain, and their use. SDBs are primarily collected for statistical analysis purposes. They typically contain both parameter data and measured data (or "variables") for these parameters. For example, parameter data consists of the different values for varying conditions in an experiment; the variables are the measurements taken in the experiment under these varying conditions. The data base is usually organized into "flat files" or tables.

The statistical analysis process involves the selection of records (or tuples) using selection conditions on the parameters, taking a random sample, or using a graphics device to point to the items desired. Several variables are then selected for analysis. The analysis may involve applying simple univariate statistical functions to the value sets of the variables (e.g. sum, mean, variance) or using more complex multivariate analysis tools (e.g. multiple regression, log-linear models).

The statistical analysis process may involve several steps. It includes phases of data checking, exploration, and confirmation. The purpose of data checking is to find probable errors and unusual but valid values (called "outliers" by statisticians), by checking histograms or integrity constraints across attributes. The purpose of data exploration is to get an impression of the distribution of variables and the relationships between them. This phase involves taking samples of the data, selecting records, and creating temporary data sets for use in graphical display and preliminary analysis. In the conformation phase, the analyst tests hypothesized distributions (which are based on the observations made in the exploratory phase) against the data base, or relationships between variables (cross tabulations). This process may then iterate several times until satisfactory results are achieved.

At first glance it appears that the necessary data management functions can be supported by existing general purpose data management systems. For example, one can view flat files as relations in a relational data management system, and generate subsets for analysis by using relational operators, such as "join" and "project". However, practice has shown that these data management systems have not been used for SDBs. Instead, one finds that statistical packages are used or special purpose software is developed. Most statistical packages have some data management capabilities, but their primary purpose is to provide statistical analysis tools to the analyst.

There are two main reasons for the fact that commercial data management systems have not been widely used for SDBs. The first reason is the storage and access inefficiency of these systems for SDBs. As will be discussed later, many SDBs have a high degree of data redundancy that can benefit from sophisticated compression techniques. The organization of the data into records (or tuples) makes retrieval inefficient in those cases where only a few attributes are needed for the analysis. Other data organization methods, such as organizing the data by columns instead of rows (called "transposed files") are usually more efficient [15]. Most existing data management systems are designed for high volume interactive transactions with the possibility of concurrent access to the data. The overhead required for the support of concurrent access is not necessary for SDBs, because much of the data is static. Also, analysts

work with their particular subset of the data, and are willing to put up with occasional sequential access to the original data bases.

The second reason stems from the lack of functionality and ease of use. Statistical functions available in commercial data management systems are quite limited, usually to simple aggregate univariate functions such as sum, maximum, or average. Most systems do not have facilities for supporting additional user-defined functions, although some provide an ability to create predefined functions in libraries. In addition, some query languages are quite complex when it comes to specifying aggregate functions. As will be discussed later, the query language can be simplified if the semantic properties of the SDBs are modeled.

Ease of use considerations are much more pragmatic. In order to perform statistical analysis, an analyst must eventually rely on more sophisticated statistical tools such as those found in statistical packages. This means that in order to use a data management system the analyst will need to become familiar with two systems, and the methods used to pass data between them. Often, the analyst will choose to stay with the essential statistical tools provided by the statistical package, and manage with the limited data management tools provided by them.

In the sequel, we identify some characteristics that are common to SDBs. These characteristics are both in terms of the structure and use of the data.

4.1 Category and summary attributes

SDBs can be thought of as having two types of data: measured data on which statistical analysis is performed, and parameter data which describe the measured data. There are several reasons to making this distinction.

To illustrate the reasons, consider for example a simple data base represented in a relation form. The first five attributes are: oil type, state, county, year, month. They represent the parameter data. The last two attribute are: consumption, production. They represent measured data. The attributes for the parameter data are referred to as "category" attributes, since they contain categorical values for the measured data. The attributes for the measured data are referred to as "summary" attributes, since they contain data on which statistical summaries (and analysis) are applied. There are several points to note.

First, note that a combination of the category attribute values is necessary for each of the values of each summary attribute. That is, the category attributes serve as a composite key for the summary attributes. Thus, each summary attribute is functionally dependent on the the

category attributes. This relationship between category and summary attributes is part of the semantics that need to be modelled.

Second, there is a great amount of redundancy in the values of the category attributes, when they are represented in a relation. In many data bases all possible combinations of the category attributes (i.e. the full cross product) exist. In such cases each value of a category attribute repeats as many times as the product of the cardinality of the remaining category attributes. This is the main reason for the organization of SDBs into matrix form, as is the preferred representation for statisticians. A matrix organization replaces the need to store the category values in the data base by representing them as positions of the columns and rows. This suggests the need for the efficient efficient storage and access of category attributes.

Third, the range of category attributes is usually small, from as little as two (e.g. "sex") to a few hundreds (e.g. oil type). In contrast, summary attributes often have large ranges since they always represent numeric measures. Often, category attribute ranges are grouped together so as to have fewer categories, such as using "age groups" rather than "age". Also, category values are more descriptive in nature, and therefore tend to be character data (e.g. industrial classes), while summary values are numeric. Often, coded versions of the text are assigned to long category values. This suggests the need to support the mapping between the codes and the descriptive values.

4.2 Classification hierarchies over category attribute

In SDBs, each of the category attributes can represent a hierarchy of terms. For example, "oil type" can be organized into "crude oil", "heating oil", and "refined products". Each of these categories can be further organized into sub-categories. For example, "refined products" can include "leaded gasoline" and "unleaded gasoline". There are two implications to this property.

First, the data model should have the capability of representing this structure. Note that in the relational model there is no such capability. In order to represent the information in the category hierarchy, each hierarchy will have to be "flattened out" into multiple attributes. Even so, the semantics of the hierarchical relationship will be lost.

Second, summary queries can be requested to any level of the hierarchy. Typically, the summary values are given for categories in the leaves of the category hierarchy. Thus, request for data at higher levels would require summarization from the leaves to the higher levels. For example, the values for oil consumption should be summed up from the leaves of the category

hierarchies, i.e "leaded gasoline", etc.. Thus, the data model will has to include the semantics of summary operators over the category hierarchies.

4.3 Sparsity of the cross product space

In many statistical data bases, there are no summary values for some of the cross product space elements of the category attributes. For example, consider a data base on trade between states by year. Since not every state produces all products and since a state sells its product only to a limited number of states, it follows that many cross product elements of (producing state, consuming state, product, year) are not valid. There are two options of dealing with the sparsity of the cross product space. The first is to leave the null values (or zeros, or any other designated constants) for the summary attribute in the data base and then squeeze them out using compression techniques. The second option is to remove entries that have null values from the data base. The trade off between these options suggests specialized physical organization methods, similar to those that are used for supporting sparse matrices.

4.4 Summary sets

When statistical data bases are very large, it becomes too expensive to work directly with the original data set. Users extract smaller data sets that are of interest to them, apply the usual selection functions to limit the number of entries in the data set (such as only the western states), apply projection functions to limit the summary data they are interested in, and join data from different data sets (although tools for joining are not always available). But in addition, a very common operation is to reduce the number of category attributes by summarizing over them. Thus, in the example discussed above, a user can request total consumption by oil type, by state, by year, so that the consumption values are totaled over the appropriate counties and months.

In an active data base, a large number of summary sets may be generated, suggesting the need for the management of versions of summary datasets, and the maintenance of the relationship between them.

4.5 Stability

A large proportion of statistical data bases are very stable. Initial corrections may be required but very little updating is necessary afterwards. This stems from the primary purpose of SDBs, which is to collect data for future reference and analysis. Once the data is collected, there usually is no reason to change it unless it is for the correction of identified errors. Even

in data bases that are usually associated with a high degree of updating, such as inventories, the transactions are actually recorded over time, if further analysis is desired. Actually, most businesses, such as banks, retail stores, etc., record all transactions as verification that the transaction has taken place, along with the time and person performing the transaction. Thus, these parts of the database are not updated very often.

The stability of SDBs is a benefit since many of the problems that arise in multiple updates to data bases that require concurrency control algorithms can be avoided. There is another benefit to the stability of data bases which takes advantage of the trade-off between retrieval and update operations. If one assumes very little or no updating, it is possible to design more efficient retrieval algorithms on account of slow updating.

4.6 Proliferation of terms

This phenomenon is not unique to statistical data bases, but exists whenever a data base contains a large number of attributes. When a data base has hundreds (or even a few tens) of attributes, it is necessary that some tools be provided for dealing with such complexity.

In order to formulate a query, a user must remember the following things in addition to the details of the query language: the names of data sets (or relations) needed, the names or acronyms of the attributes needed, the possible and legal values for these attributes, and the formats of the values (e.g. the format for age groups, or whether to use capitals in names of cities). In addition, the codes or abbreviations that were assigned to values (e.g. codes for states and counties) must be remembered. It is not surprising that such data bases require specialists to access them.

These difficulties are even more serious in SDBs, for two reasons. First, many data bases have categories that change their definitions over time. An example of this situation is that counties change their boundaries but not their names over time. Also, the same terms are used with slightly different meanings. For example, the term "state" may include Guam and Puerto Rico in one data base, but not in another. The second reason stems from the summary sets. With every new summary set that is created, new names are introduced, or perhaps old names with new meanings. It is necessary to control this proliferation of terms, and to keep track of what exists in the system.

5. Properties of Scientific Data

It is useful to distinguish between different types of scientific data. In this section we describe these types and their main features.

5.1 Experiment and simulation data

Most scientific data result from experiments and simulations. Data from experiments are usually measurements of some physical phenomena, such as the collision of particle beams, or the spectra generated by molecules in a strong magnetic field. Data from simulations typically result from complex computations derived by using values from the previous step of the simulation. Both experiment and simulation data have similar characteristics, and therefore are considered jointly. In order to simplify the terminology used here, we refer to such data as "experiment data", regardless of whether they are experiment or simulation data. Experiment data can be classified according to three characteristics: regularity, density, and time variation.

a) Regularity

Regularity refers to the pattern of the points or coordinates for which values are measured or computed. For example, in physics experiments, detectors are placed in a specific configuration. If the configuration describes a regular grid or some other geometric structure, the experiment is said to have (spatial) regularity. Similarly, many simulations assume some regular grid for which values are computed, and therefore have spatial regularity. In addition, if values are measured or computed at regular time intervals, then time can be considered as another regular coordinate of the data.

In general, regularity implies that a mapping between the coordinates of measured values and the storage locations of these values can be made by means of a computation (such as "array linearization", which is simply a mapping from multi-dimensional space to linear space, similar to FORTRAN array mapping). Therefore, in such cases it is not necessary to store the coordinate values with each measured data value, resulting in storage savings and fast random access. On the other hand, when spatial irregularity exists it is necessary to enumerate the data points, and store their identifiers with the data values.

b) Density

Density indicates whether all the potential data points have actual values associated with them. For example, simulation data of fluid motion computed on a regular grid would have

data values (for velocity, direction, etc.) computed for each point of the grid, and therefore the data is considered dense. On the other hand, in many experiments a large number of measurements that are below a certain threshold are discarded and never recorded. In fact, the level of sparsity can be quite high, i.e. only a small fraction of the potential data points have recorded values. For example, in physics experiments of colliding particle beams, the measured data is only for resulting sub-particles, which occur over a small portion of the detectors that are distributed in space.

Sparsity implies a large number of null values which may be compressed out. The compression technique chosen should depend on the access patterns to the data, such as whether the data are accessed sequentially or randomly.

c) Time variation

Time variation refers to the change of coordinates over time; i.e. the points for which data values are measured or computed change their position from one time unit to another. For example, consider some material that is bent in the course of an experiment. Before the experiment starts a set of points is selected for measuring the material's behavior (such as stress, voltage, temperature). During the experiment the selected points may change their position as a result of the bending action. Time variation is a characteristic found mostly in simulations where a mesh of points are allowed to change their position over time during the simulation process. These simulation methods are generally called adaptive mesh techniques.

Time variation adds an important requirement. In addition to storing the coordinates of points for every time interval, it is necessary to maintain the relationships between the points as they existed in the original mesh. This is needed in order to be able to reconstruct the time sequence of points that correspond to the same original point, and in order to find neighboring points to a given point at any given time.

5.2 Configuration data

Configuration data are data that describe the initial structure of an experiment or simulation. For example, in simulating heat transfer through buildings, the building layout has to be described. Similarly, the configuration of an experiment describes the position of different devices and detectors. The configuration layout actually determines the regularity (or irregularity) of the experiment data mentioned above. Usually, it does not change in the course of the experiment or simulation. However, it can change between experiments or simulations. It is important to keep track of these changes and to associate the correct configuration data with

the corresponding experiment data.

5.3 Instrumentation data

Instrumentation data consists of descriptions of the different instruments and substances used in an experiment, and their changes over time. This data is crucial for the correct analysis of the experiment data. It includes information such as the pressure and temperature of a gas used in an experiment and their changes over time, drift of voltage over time, and the characteristics of detectors and devices as measured before each experiment or a series of experiments. It also includes the log of experiment operations, such as the time that a defective analog-to-digital converter was replaced, and who was in charge of it. Unfortunately, some of this information is collected into unrelated files and log books, thus making their association with the experiment data a tedious task that is prone to errors.

5.4 Analyzed data

The previous two data types are essential in order to support the analysis of experiment data. The analysis process produces many databases that also need to be managed along with their relationships to the experiment data they were derived from and to each other. The analysis process may require several steps. For example, in physics experiments of colliding particle beams, a preliminary histogram over the experiment data can be done in order to estimate parameters that are used to interpret the calibration data of detectors in the next step of the analysis. For each collision, called an event, the tracks of sub-particles produced are reconstructed and kept in a database. From the track data, another database for the event data can be derived, describing the kind of sub-particles produced and their characteristics. Additional steps use databases from this and earlier stages to generate yet more data. It is important to capture the analysis process, the input and output databases of each step, and the relationships between the steps.

5.5 Summary data

Similar to "statistical" databases, which deal with statistical summaries (aggregations) of data sets, scientific databases are often aggregated. For example, in experiments of heat transfer in buildings, the amount of heat lost or gained can be averaged over several points of a wall, summed over entire rooms, or aggregated over days into months. Another example, is the generation of histograms from many experiments to determine the likelihood of a certain phenomenon. As in the case of statistical databases, there is a need to organize, search and

browse collections of summary data, and to preserve their relationship to lower level data from which they were derived.

5.6 Property data

In any scientific field, the summary of information learned over the years is useful to the community at large. There is a substantial amount of work devoted to the organization and classification of properties of materials, substances, and particles. For example, there are several systems devoted to the storage and retrieval of chemical substance properties. Many property databases cannot now be accessed on-line. The data is only available in periodically published books, and may not be up-to-date. Property data is non-uniform: it contains numeric, text, and bibliographic data, as well as images and graphs. This is one of the reasons that for each scientific area special purpose systems have been developed. Data management systems that can deal with such diversity of data types are not generally available. In addition, because of the complex terminology involved with such data, sophisticated search and browsing capabilities are needed.

The following observations can be made relative to scientific databases and the way they are used.

(1) Multi-dimensional data are prevalent in scientific databases. Methods for efficiently managing, accessing, and compressing multi-dimensional data are necessary.

(2) Scientific databases are frequently accessed via proximity searches and successive queries often exhibit locality of reference. Techniques of partitioning the data into cells (or grids) along the coordinates of its dimensions seem to be the most promising for efficiently supporting these needs.

(3) Although scientific databases are usually very large, they can be often partitioned into small independent units during early data reduction. This implies that parallel processing can be applied. For example, in particle physics experiments, each event can be analyzed independently of other events.

(4) Scientific databases include a variety of support data that describe instruments and the configuration of experiments. Often this data is not explicitly organized but rather made part of application programs, a practice that tends to cause many difficulties. The requirements of such support data can be handled for the most part with conventional database techniques, but need to be integrated with the data that result from experiments. Some configuration data need special capabilities found in engineering database

systems.

(5) The analysis of scientific data generates many summary data sets which need to be managed. Special techniques for handling analyzed data and summary data are required in order to manage their metadata, to keep track of numerous data sets.

(6) Temporal aspects of scientific databases are important. They range from time series of the measured data, to logs of instrument variation over time, to the historical sequence of generating different summaries of the data. Thus, support for temporal data structures and operations is needed.

(7) There are many aspects of scientific databases that are similar to statistical databases; in particular, supporting the multi-dimensional aspects of the data and the handling of summary data.

Summary and Conclusions

Statistical and scientific database applications are complex because they involve data and procedures of complex disciplines (biology, physics, etc.). They have requirements that far exceed the capabilities provided by current commercial data management systems. In this paper, we have discussed some of the properties and requirements of SSDBs.

The paper contains a discussion of current approaches to data management for complex applications, and evaluates their adequacy for SSDBs. Three approaches were discussed: object-oriented database systems, extended relational systems, and extensible database systems. It was pointed out that while all of these approaches have the goal of supporting user defined data structures and operations, each of these approaches has different strong points. Object-oriented systems provide an elegant environment for encapsulating new objects and operations, extended relational systems provide a rich complex object model (as well as other features, such as support for rules), and extensible database systems are designed to provide an environment for interchangeable modules, as well as to support dynamic changes to the query optimizer. For SSDBs these features are quite desirable. Thus, it is important that future research will bring about systems that can incorporate the advantages of the three approaches.

The development of flexible and powerful data managements systems does not in itself provide a solution for SSDB applications. The characteristics of each application need to be well understood before an appropriate model of the application can be developed. In addition, new methods for physically supporting the scientific application may need to be developed.

For example, the characterization of statistical database applications, and the development of appropriate models, query languages, user interfaces, and physical structures for these applications, has been going on for over a decade, and is still continuing. The success of such developments requires a detailed understanding of the discipline as well as the needs of scientists in those disciplines.

Acknowledgement

This research was supported by the Applied Mathematics Sciences Research Program of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098.

References

(1) Proceedings of the First LBL Workshop on Statistical Database Management, Menlo Park, California, 1981.

(2) Proceedings of the Second International Workshop on Statistical Database Management, Los Altos, California, 1983.

(3) Proceedings of the Third International Workshop on Statistical and Scientific Database Management, Luxembourg, 1986.

(4) Proceedings of the Fourth International Working Conference on Statistical and Scientific Database Management, Rome, Italy, Springer-Verlag, 1988.

(5) Proceedings of the Fifth International Conference on Statistical and Scientific Database Management, Charlotee, N.C., Springer-Verlag, 1990.

(6) Scientific Database Management, Computer Science Report No. TR-90-21, University of Virginia, August 1990, French, J.C., Jones, A.K., Pfaltz, J.L., eds, to be published in SIGMOD RECORD.

(7) Shoshani, A., Statistical Databases: Characteristics, Problems, and Some Solutions, *Proceedings of the 8th International Conference on Very Large Data Bases (VLDB)*, 1982, pp.208-222.

(8) Shoshani, A., Olken, F., Wong, H.K.T., Characteristics of Scientific Databases, *Proceedings of the 10th International Conference on Very Large Data Bases (VLDB)*, 1984, pp. 147-160.

(9) Special Issue: Object Oriented Design, Communications of the ACM, September 1990.

(10) Dittrich, K.R., "Object Oriented Database Systems: The Notion and the Issues, Proc. 1986 IEEE International Workshop on Object Oriented Database Systems, Pacific Grove, pp. 2-6.

(11) Stonebraker, M., Rowe, L.A., The Design of Postgres, ACM SIGMOD Conference, 1986, pp. 340-355.

(12) Linnemann, V., Kuspert, K., et al, Design and Implementation of an Extensible Database Management System Supporting User Defined Data Types and Functions, Proc of the 14Th VLDB Conference, 1988, pp.294-305.

(13) Carey, M., DeWitt, D., et al, The EXODUS Extensible DBMS Project: An Overview, In Readings in Object-Oriented Databases, S. Zdonic and D, Maier, eds., Morgan-Kaufman Publishing Company, 1989.

(14) Batory, D.S., Concepts for a Database System Synthesizer, ACM PODS, 1988.

(15) Turner, M. J., Hammond, R. and Cotton, F., A DBMS for Large Statistical Databases, Proceedings of the Fifth International Conference on Very Large Databases, 1979, pp. 319-327.

LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
INFORMATION RESOURCES DEPARTMENT
BERKELEY, CALIFORNIA 94720