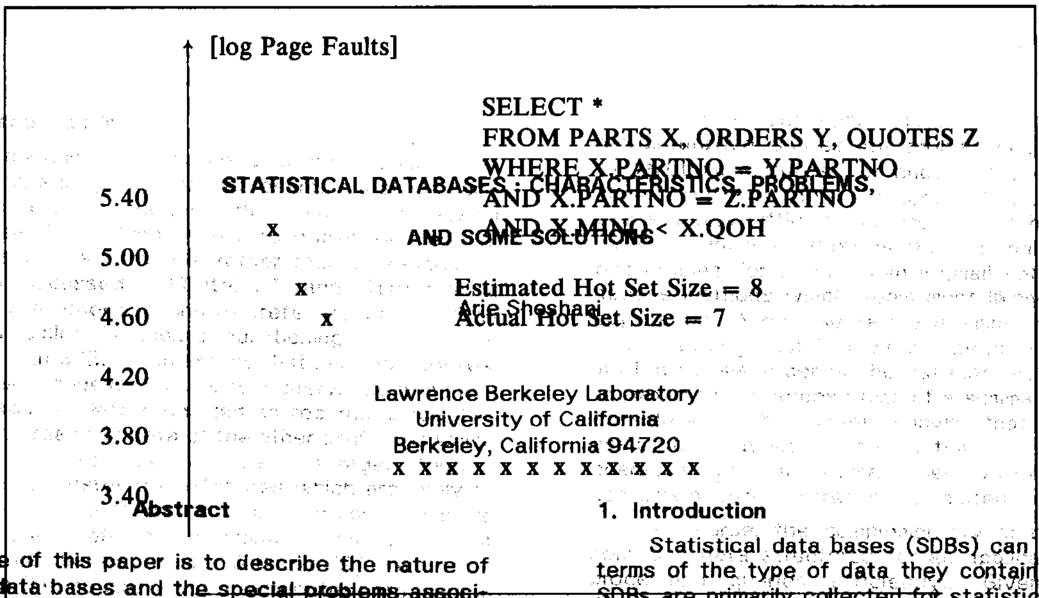
STATISTICAL DATABASES: CHARACTERISTICS, PROBLEMS, AND SOME SOLUTIONS

Arie Shoshani

Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

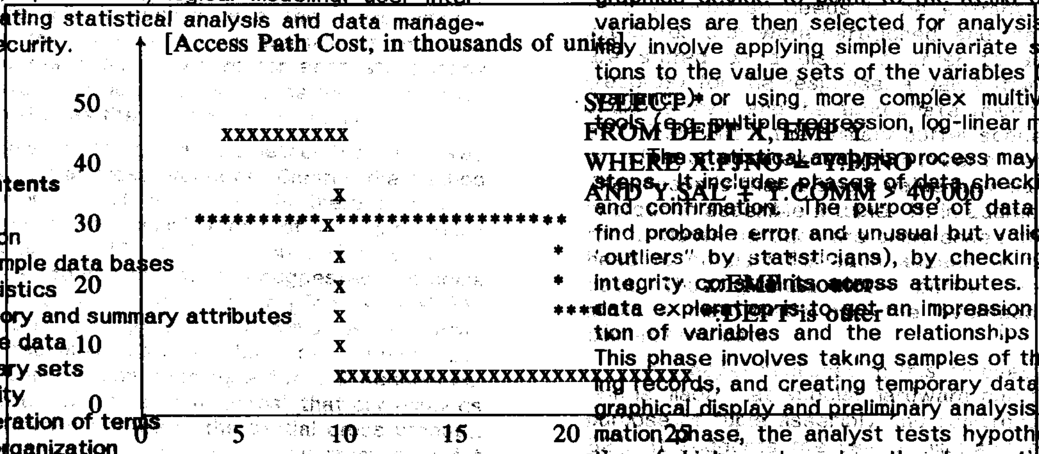


```
SELECT *
FROM PARTS X, ORDERS Y, QUOTES Z
WHERE X.PARTNO = Y.PARTNO
AND X.PARTNO = Z.PARTNO
AND X.MINQ < X.QOH
```

Figure 3. Log (base 10) of measured page faults versus Buffer Size for 3-way join query

Abstract

The purpose of this paper is to describe the nature of statistical data bases and the special problems associated with them. Since statistical data bases are common in a variety of application areas, the paper begins by describing several examples that emphasize the complexity, the size and the difficulties of dealing with such data bases. A description is then given of the characteristics of statistical data bases in terms of data structures and usage. The remainder of the paper describes a large collection of problems, and when appropriate some solutions or work in progress. The problems and solutions are organized into the following areas: physical organization, optimization, logical modeling, user interface, integrating statistical analysis and data management, and security.



```
SELECT *
FROM DEPT X, EMP Y
WHERE X.PJNO = Y.PJNO
AND Y.SAL + Y.COMM > 40,000
```

Figure 4. Computed access path cost for two ways of performing 2-way join. In both cases, an index scan on PJNO is used for both tables.

Table of contents

1. Introduction

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 error 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. A more detailed description of the statistical analysis process can be found in [Boral et al 82]. A compact description of data manipulation capabilities that are important for SDBs can be found in [Bragg 81].

At first glance it appears that the necessary data management functions can be supported by existing generalized data management systems. For example, one can view flat files as relations in a relational data management system, and the generation of 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 for the particular data base in hand, or for a collection of data bases with similar characteristics. A case in point is the Census Data Base, which is collected and processed by special

When a process unfixes a page, the reference count of the page is decreased by one.

When a process terminates, the pages of its stack are allocated to processes whose LRU stack are deficient. This allocation is done by finding a deficient LRU stack and giving it as many free frames as required to complete the number of pages in its hot set size σ. If there are still free pages left, another deficient process is found and frames allocated to it as before. If there are no more deficient processes and there are still free frames left, they go to the free list.

A step by step description of the buffer management schema follows:

1. Initialize. Assign to the free list all the buffer frames.
2. New process arrives. Allocate empty LRU list. Set numref=σ and numall=0. Get as many as σ pages from the free list. Set numall to be the number of pages that were actually obtained (if the free list size is less than σ then numall will be be less than σ and the free list becomes empty).
3. Process requests a page. Search for a page in the buffer pool.
   a. Page found. If page is in local LRU stack, update local stack; else do nothing.
   b. Page not found. A page fault occurs. Have to determine a page to be flushed out of the buffer pool. There are two cases to consider:
      1. If the local stack contains unfixed pages then flush out the least recently used (i.e., use the modified LRU algorithm.)
      2. If the local stack contains only fixed pages then get a frame from the free list, if it is not empty (and flush the corresponding page). If the free list is empty, get a frame from another LRU stack containing an unfixed page. Stacks that are deficient are to be preferred. Increase numall by 1. Note that numall may become larger than numref in this case.
4. Process fixes a page: Increase reference count by one.
5. Process unfixes a page. Decrease reference count by one.
6. Process releases a page: Do nothing.
7. Process terminates: Reallocate the LRU stack among the deficient processes, trying to satisfy completely a process before satisfying another one. If no deficient process exists, return frames to the free list.

4. COMMENTS ON BUFFER MANAGEMENT ALGORITHM

The above schema presents the basic components of a buffer management schema based on the idea of the hot set size. We now discuss the behavior of the algorithm under different situations.

1. The various hot sets scanned during the evaluation of the request need not be all of the same size. The optimizer only computes an average number for the hot set size, based on available statistics such as the sizes of the relations involved, the cardinalities of different columns, the filter factors of predicates, etc. Thus during execution, a process may loop over a set of pages larger than this hot set size. Assuming that numref is larger than the maximum number of pages that the process simultaneously maintains fixed, then step 3b2 will not be executed on its behalf. Thus, its LRU stack will never be increased, and internal thrashing will occur as the process steals pages from itself. Note however, that there will not be any external thrashing. To accommodate for this situation, the optimizer could request a hot set size greater than the average it computed. However, this leads to underutilization of the buffer pool. Another possibility is to dynamically increase the allocation of frames to this process if the free list is not empty. These frames would be released if another process requested frames from the free list and found it empty (i.e., consider the extra frames obtained by the first process to be an extension of the free list). Although this change could be incorporated into the buffer manager, the solution presented in the previous section appears to be a good balance between simplicity and performance.
2. If the hot set size for a process is equal to the average number of fixed pages that it requires, then a situation can occur where more pages than the allocated number have to be simultaneously fixed. In this case, step 3b2 makes sure that the process gets another page. The issue of what happens when all the pages in the buffer are fixed and a request for another one is made is outside the scope of this paper. Solutions that have been adopted in this case include bringing down the system or dynamically extending the buffer.
3. To make sure that the number of deficient processes is kept to a minimum, one can schedule the requests so that the sum total of the hot set sizes of the processes that are being serviced is less than or equal to the size of the buffer pool. The question of scheduling is not considered here. However, we note that this kind of scheduling may lead to deadlock problems that are [illegible] [GRAY 78]. To avoid deadlock problem one can restrict the set of concurrent processes (i.e., reduce the multiprogramming level). Although this does not eliminate the possibility of processes running with stacks that are deficient it does eliminate the issues of deadlock related to buffer requests and considerably reduces the occurrence of internal thrashing. Moreover, knowledge of the hot set sizes of the request issued by processes can be used to decide on an appropriate multiprogramming level. Thus, the restriction on the multiprogramming level appears to be a suitable complement to the algorithms presented here.
4. Note that in the schema presented above, a page in the buffer appears in only one LRU chain. Thus, if a page is shared by two processes, the first one requesting it will put it on its own LRU chain. Although this may seem arbitrary, there is no problem in doing so. If the shared page remains active (e.g., a control page in System R) it will remain high in the LRU chain (i.e., it is unlikely that the page will be flushed out.) On the other hand, if the cross reference to the page is a sporadic event, the page will migrate from one LRU stack to the other. This is not important since these occurrences are infrequent.

purpose software, distributed as flat files whose descriptions are quite complex, and therefore requires special purpose software for subsequent querying. An example of such a system which was designed specifically for [illegible] is the Socio-Economic, Environmental, Demographic Information System (SEEDIS), developed at at Lawrence Berkeley Laboratory [McCarthy et al 82].

Another approach to managing SDBs is by using statistical packages such as SAS [SAS 79] or SPSS [Nie et al 75]. While these packages have some data management capabilities 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 [illegible] when 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 [Teitel 77, Turner et al 79]. Most existing data management systems are designed for high volume interactive transactions with a high probability of concurrent access to data. The large overhead required for the support of concurrent access is not necessary for SDBs. Analysts work with their particular subset of the data, and are willing to put up with occasional sequential access to the original data bases. A short discussion of additional reasons can be found in [illegible].

The second reason stems from the lack of functionality and ease of use. Statistical functionality that is 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. This is in part because of insufficient modeling of the SDBs, as will be discussed later.

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 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. However, [illegible] more sophisticated data structures, such as networks, matrices, or vectors, and the operations to manipulate them are required. In such cases, the choice is between limited capabilities of a single system, or [illegible] problems of interfacing data management systems and statistical packages are discussed [illegible].

The purpose of this paper is to describe the special characteristics and problems of SDBs. When appropriate, solutions that have been proposed in the literature are pointed out. [illegible]

Database Management. In section 2 a few representative examples are described. Section 3 contains a discussion of the special characteristics of SDBs. The remaining sections describe problem areas requiring special attention.

2. Some example data bases

In order to provide an overview of the types of data that comprise statistical data bases, a few representative examples are discussed here. SDBs exist in virtually any enterprise that collects data. Often, the data is collected for the purpose of statistical analysis, such as census data or [illegible] ments. But SDBs are also generated as a result of continuing operations, such as the collection of mortality data in hospitals, or logs of financial transactions in banks. SDBs are prevalent in scientific and socio-economic applications. However, they also exist in any business enterprise that needs to perform data analysis of its performance, market projections, or inventory trends.

Example 1: A scientific experiment

The purpose of this example is to show that the quantity of data collected in SDBs can increase rapidly.

An experiment in atmospheric physics was carried out by the Atmospheric Science Department at Battelle Pacific Northwest Laboratories, Richland, Washington. The purpose of the experiment is to detect the build-up of different elements in the atmosphere such as carbon dioxide, which can lead to the depletion of the ozone layer. As part of the experiment, measurements are taken of the aurora at night to determine the amount of [illegible] automatic scanning photometer, which can take measurements at different azimuths and elevations. [illegible] measurement [illegible] every six degrees (i.e. sixty times for a full circle). The experiment is repeated on the average four times an hour. There are five filters, so five measurements are made at each position. A quick calculation shows that the number of measurements taken in an eight hour night exceeds [illegible] times/hour x 8 hours/night x 5 filters = 57,600). The experiment takes place at ten stations and was designed to run for five years. This brings the total number of measurements for the entire experiment to about one billion.

In addition to the measured data, the parameter data must be stored in some form. For each measurement there are parameter data for the azimuth, elevation, year, month, day, time, filter type, and station number. There may be more information on who ran the experiment, malfunction of devices, etc. The parameter data can increase the size of the data base by an order of magnitude, unless special techniques are applied for efficient storage. Some such techniques are discussed in a later section.

Example 2: County business patterns

The purpose of this example is to show that some "flat files" require a fairly complex description of what the rows and columns represent.

minimum hot point (i.e., the minimum number of pages required to run). The schema presented here guarantees that, if processing of this query is allowed, it will not suffer external thrashing. (Note that for these requests there can be no internal thrashing.) In a high performance system, in order to insure that a certain level of fast queries is active at the same time, the buffer can be divided into two regions; one for fast requests, the other for slow ones. (The initialization step would create two free lists each having the size of the respective region. A reasonable choice is to have both regions of the same size.) The fast requests are thus guaranteed a certain minimum number of frames for them. In order to maintain good utilization of the buffer pool, the free list for the fast queries has to be allowed to take frames from the free list for the slow queries (this will happen when there are no slow queries present in the system.) As before, adequate service can be maintained by restricting the multiprogramming level.

6. The last comment has to do with contention problems in the buffer manager. Clearly, the stack manipulations that are performed on the free list in steps 2 and 3 of the algorithm must be serialized. Since the operations that are done are removal and insertions of elements in LRU chains, the path length through the critical region is no different than that through the code of a standard buffer manager with one LRU stack. Thus this algorithm should not introduce additional serialization problems. In fact, it may even decrease them as the manipulations of private LRU chains need not be serialized.

5. EXAMPLES OF HOT SET SIZE COMPUTATION

As an illustration of how the hot set size can be estimated by the optimizer of a relational database management system, we show some examples using System R. For each two way join between relations R1 and R2, assume R1 is the outer relation and R2 is the inner. Control pages are not included in the expressions below. The following terms are used below

P(R1)	Number of pages for relation R1.
P(R2)	Number of pages for relation R2.
dindex (R2)	Depth of the index on R2.
ls(R2)	Number of pages in the inner loop for R2. It is given by P(R2) divided by the number of different values for the attribute upon which the join is performed. This estimate is based on the uniform distribution assumption.
I(R2)	Number of pages in the index used to access R2.
lsleaf(R2)	Number of index leaf pages scanned on an inner loop of R2.

for a sequential scan on both R1 and R2,

hot point = 1 + P(R2)

for an index scan on R1, sequential scan on R2,

hot point = 2 + P(R2)

for a sequential scan on R1, index scan on R2 (smooth discontinuity), interpolate between

1 + dindex(R2) + ls(R2)

and

1 + I(R2) + P(R2)

Example 1.

Type 2 join:

for a sequential scan on both R1 and R2,

hot point = 1 + ls(R2)

for a sequential scan on R1, index scan on R2,

hot point = 1 + lsleaf(R2) + ls(R2)

where lsleaf(R2) is the number of leaf pages in the inner lo
For an index scan on R1, sequential scan on R2,

hot point = 1 + dindex(R1) + ls(R2)

Example 2.

In Example 1, the first formula is derived by reserving enough frames to contain the entire R2 relation, plus one frame for a data page for R1. If a frame for a data page of R1 is not reserved the access to R1 causes the first page in the R2 loop to be replaced, and consequently the entire set of pages in the R2 loop to be lost. For the second formula, an additional frame is reserved for the leaf pages of the index, which is always accessed before accessing R1 data pages. The third formula presents a smooth discontinuity. The minimum number of faults is achieved when all the access entities for R2 (index and data pages) completely fit in the buffer. The number of faults will increase in a roughly linear way, until only a number of frames sufficient to hold an average loop on the second relation, is available. This assumes substantial rereferences between succesive inner loops. If the number of data pages in the referenced relation is large, and the join filtering is high, data page rereferencing will be very low. In this case, P(R2) is substituted for ls(R2). The formulae in Example 2 are derived using analogous considerations. Values for the estimated number of hot set size obtained using these expressions are shown in Figures 1-3.

The above formulae are easily generalizable to n-way joins, and represent a conservative estimate of the hot points. The number of hot points to be computed varies from n-1 to 3(n-1) (in the case of smooth discontinuities), where n is the number of relations referenced in the query.

A MECHANISM FOR MANAGING THE BUFFER POOL IN A RELATIONAL DATABASE SYSTEM USING THE HOT SET MODEL

Giovanni Maria Sacco and Mario Schkolnick

IBM Research Laboratory
San Jose, California 95193

ABSTRACT

The design of the buffer manager in a Relational Database Management System can significantly affect the overall performance of the system. Thrashing is a common phenomenon that occurs in these systems due to the combination of a regular pattern of accesses made by a process and the competing requests for buffer resources made by concurrently executing processes. In this paper, we present a buffer management algorithm based on a model of database requests. A discussion of problems encountered by traditional methods for buffer management as well as extensions to the algorithm are also presented.

1. TRADITIONAL METHODS FOR BUFFER MANAGEMENT

Database Systems typically use an LRU replacement technique [LANG77] to manage their internal buffer(s). The LRU technique has proved to be more efficient than other methods in reducing the amount of paging that occurs in these systems. Also, the technique is relatively simple to implement, something that is highly desirable in a high performance system. However, as the following examples show, there are many cases where serious problems occur.

1. If one of the processes is running a batch like job, i.e., one that performs a long sequential scan on the data while at the same time requesting pages very rapidly, the pages referenced by it will tend to go to the top of the stack, causing the pages used by other processes to be flushed out of the buffer. This causes the batch like process to take precedence over the other processes. If these are short, fast transactions, the situation becomes intolerable.
2. Another case where LRU behaves very badly is when a process cycles through a set of pages, which is larger than the set of pages that can fit in the buffer. In this case, every new reference to a page causes a fault. This effect is called **internal thrashing**.
3. A more serious problem that occurs in multiuser systems is that one process may use pages more rapidly than another one that has a looping behavior as it references pages. In this case, even if the set of pages that are rereferenced inside the loop may be smaller than the buffer size, the "stealing" of pages by the first process effectively reduces the available frames in the buffer for the second process. This reduction can be such that the second process generates a page fault on every request. In this case we say that there is **external thrashing**.

Mechanisms for dealing with thrashing have been developed in Operating Systems. The most well known of these uses the principle of the Working Set Model [DENN68]. A buffer manager using this model defines a window τ and observes the average number of different pages that are requested by a process in a period equal to τ. This number is called the working set size of the process, σ. A scheduler then ensures that while this process is running, at least σ pages are allocated to it. Alternatively, the buffer manager can give different priorities to page requests from the various processes, depending on their working set size. Processes with a large working set size will get more buffer frames allocated to them. In high performance Database Management Systems, this last mechanism has several drawbacks. In the case of a process, like the batch process described in case 1 above or the one described in case 3, the Working Set Model tries to give to this process many buffer frames, causing external thrashing of other processes. If a process loops over a number of pages larger than the window size τ, as in case 2, the working set mechanism will attempt to reserve τ frames for this process in the buffer, where having just one frame associated to it would have caused the same level of faults to occur. Thus the buffer frames are poorly utilized. Finally, in the case of a process with a looping behavior, as it goes from one loop of pages to the next, the working set size will temporarily increase causing this process to get more frames assigned to it. As before, this causes external thrashing. All of the above cases may also cause scheduling problems if processes are scheduled considering their working set size requirements. Moreover, the working set model is expensive to implement, in terms of instructions executed, a fact that has discouraged its use in high performance database systems.

2. THE HOT SET MODEL

Relational Database Management Systems have high level language interfaces allowing their users to state their processing requirements without specifying how the required data should be accessed. The system has internal mechanisms that decide on the best strategy to access the data. We call these strategies **access plans**. In System R[1], the internal mechanism is called the Optimizer [SELI79]. Since the access plan is generated by the system, it turns out that the pattern of data pages that are referenced can be predicted at the time the access plan is generated. It is not hard to see that for all plans, the pattern of accesses involves looping through sets of pages.[2] This fact has led to the introduction of the Hot Set Model [SACC82] to study this behavior. A set of pages over which there is a looping behavior is called a **hot set**. If the access plan is executed within a buffer whose size exceeds the size of the hot sets, processing will be fast, as the pages that are referenced in a loop will stay in the buffer. On the other hand, if the buffer size is below the value required to contain a hot set, the number of page faults required grows rapidly.

This effect can be observed in Figures 1-3 where the total number of page references made while processing different queries is shown as a function of the buffer size. As can be seen from these figures, there may be more than one looping pattern, causing several discontinuities to occur in this curve. Each one of these discontinuities is called a **hot point**. Also, there is a minimum buffer size under which the query cannot run. This is a particular characteristic of System R. When processing a request, System R forces some pages to remain in the buffer pool, independently of what the LRU algorithm would call for [LORI77]. (A page that is forced to remain in the buffer pool is called **fixed**. Because pages are fixed, System R does not implement a true LRU mechanism for managing its buffer pool.[3]) This last discontinuity is also referred to as a hot point. The largest hot point smaller than the available buffer size, is called the **hot set size** for the query. In particular, the hot set size could correspond to the number of pages a process could keep fixed at the same time. Examples of hot set sizes for various queries are shown in Figures 1-3, using a buffer size of 20 pages. These figures show that running a query without a number of pages in the buffer equal to the hot set size is very expensive. Moreover, when the processing of the request is done under these conditions, the behavior of the Optimizer as presented in [SELI79] is questionable. In fact, if processing is done with a buffer size smaller than the hot set size determined for the access plan, another access plan, with a smaller hot set size might be the best plan to take. Thus, in this situation, the Optimizer incorrectly generates the first access plan.

An example of this situation is shown in Figure 4. For a buffer size larger than 10, the cheapest path is to use EMP as an outer relation, at a cost of 5,000 cost units. However if that many frames are not available in the buffer pool at the time the query is processed, then the cost of using this plan goes up to 44,000 cost units. In this case , accessing DEPT as outer would have been preferable, at a cost of 31,000 units of cost. (If there are more than 20 buffer frames at the time the query is processed, the cost for this plan goes down to 19,000 cost units.)

The above discussion suggests that the hot set size should be determined, and used, when deciding on the best access plan [SACC82]. Examples of hot set size formulas for some access plans that are examined by the System R optimizer are given at the end. Note that the formulas predict the average size of a hot set. This effect is also considered in the schema presented in the next section.

3. A BUFFER MANAGEMENT SCHEMA BASED ON THE HOT SET MODEL

The basic idea for the buffer management schema is to try to ensure that requests are run with an "effective" buffer of size equal to their hot set size. To do this, we maintain separate LRU chains, one for each process. Each chain has a maximum size equal to the hot set size for the *request being serviced by that process*. The buffer frames not associated with any process are collected into a separate LRU list, called the **free list**. Each LRU list has 2 numbers associated to it: **numref**, the number of frames that are requested by the process, and **numall**, the actual number of buffer frames in the LRU list. Numref is equal to the hot set size of the process. Numall can be different from numref if at the time the process requests an LRU stack of size σ, the size of the free list is less than σ. In this case, the entire contents of the free list are assigned to the LRU stack of the requesting process (see step 2b below). When this happens, we say that the list is **deficient**.

When a process requests a page, the content of the entire buffer is examined to determine if the page is there.[4] If the page is found, then if the page is also in the local LRU stack of the process, the local stack is updated. When a page is found, a process may fix the page to ensure it will not be flushed out of the buffer pool while the process is actively working on it. Each page has a reference counter for the purpose of keeping track of the number of times it has been fixed. When a process fixes a page, its reference counter is incremented by one.

If the page is not found, then the least referenced *page in the local stack, whose reference count is zero, is flushed out of the buffer pool and replaced by the requested page*.[5]

[1] In this paper, we use System R [ASTR76,BLAS79] as a model of a relational database management system. The ideas presented here carry over other relational DBMSs.

[2] Note that even for the case of a relation scan, there is a loop between a control page and a data page.

[3] Fixing pages is done for performance reasons. We expect that any high performance relational DBMS will have this feature. The buffer management schema presented below assumes that this is done by introducing a special case in step 3b2. If no pages are fixed, steps 4 and 5 of the schema do not exist and step 3b2 can be simplified.

[4] This can be done efficiently using a hashing technique. This is a well known method for a search a buffer pool.

[5] For simplicity, we do not examine the actions to be taken if a page in the buffer has been modified by a process. Usually, these pages are less preferable as candidates to be flushed out since their replacement involves a write operation to secondary storage.

The reasons for [illegible] files are three fold. First, it is a convenient way of viewing the data on a printed page or another two-dimensional medium, without involving graphics.[2] It is also a natural way of generating a report. Second, statisticians traditionally like to view data in matrix form. Indeed, most statistical packages support only flat files. Third, it is an economical way of representing the parameter data, even though some of it may still repeat (for example, in Figure 1, the different industry categories will repeat for each state). This preference naturally carried over to the storage of the data in this form, but necessitates some way of describing the meanings of rows and columns. Unfortunately, such descriptions can usually be found only outside of the data base.

Example 3: Geographically-based data

This example illustrates the complexity that can exist within a hierarchy of parameters. This is especially significant in geographically-based data bases, since the partitioning of geography can change with the application. In addition, the classification of the parameters can change over time.

Several census data bases use the following geographical hierarchy for gathering information on population, income, etc.:

States
SMSA (Standard Metropolitan Statistical Area)
County (or county equivalent)
Place
Census Tract

The traditional hierarchy is not a strict hierarchy, i.e. descendent elements are not always wholly contained within a *single parent element*. For example SMSAs are not always contained within states, they can cross state boundaries. Counties are always contained in SMSAs, except in New England. Places, which are usually cities, can cross boundaries between counties. Furthermore, places do not cover all areas in the U.S., since very sparsely populated regions are not assigned place names. Finally, in 90% of the cases, census tracts are contained within places; the remainder cross place boundaries. On the other hand all census tracts are contained within county boundaries which are the next level up in the hierarchy.

There is a great deal of irregularity in this hierarchical organization. Much of it came about in an evolutionary fashion over the years. Nevertheless, most statistical analysts have to deal with this situation, usually with no tools other than programming languages and statistical packages. The situation is further complicated since the definitions of these regions change over time due to legislative action or political needs. For example, existing county boundaries may be redefined and new ones introduced. *This is not unique to geography.* Another example in which changes evolve over time and are difficult to follow is in the classification of diseases. Since there are a very large number of diseases, a book of about 650 pages is needed to describe them. Every ten years an international meeting is held to review the classifications, and make changes and additions when necessary. The description of the differences between the two sets of classifications requires a 100 page document [DHEW 75].

It is therefore an extremely tedious task to correlate data over time, or between data sets collected by different agencies. In order to analyze the relationships between such data sets, projects have to include specialists familiar with the data, who may also be required to write special software to correlate them. An example of such a project is the PAREP (Populations at Risk to Environmental Pollution) project [Merrill et al 79], whose goal is to find correlations between cancer mortality and air pollution. In addition to several statisticians the project has a specialist who is an expert on such data.

Example 4: World trade

This example serves to illustrate the fact that many statistical data bases can be quite sparse, i.e. contain a large proportion of null values. Although this is an extreme example of sparsity, it is not uncommon to find data bases that are 40-50% sparse.

Over the years the World Bank has collected a large amount of economic data, mainly as time series [Johansson & Shilling 81]. One such data base contains data on trade activities between countries. There are hundreds of possible trade categories which are organized into hierarchies. The number of countries involved is also large, since the bank is particularly interested in developing countries. This data was collected over a period of several years. Thus, the number of all possible combinations for the different trade categories, by exporting countries, by importing countries, by year, is quite large.

The exact number of possible combinations is not the important issue here. Rather, it should be noted that most of the potential combinations have null values for the corresponding trade data. This is because most countries produce only a small number of trade materials, and trade with a small number of countries. If the data base is stored in matrix form, it will contain 80-90% null values. Even though it is advantageous to store the data in a matrix form to simplify the software for managing them, the storage overhead is unacceptable. In practice, this data base is stored compactly, with null values removed, but requires more complex and specialized software for its access and manipulation.

Example 5: Energy data

This is not an example of a single data base, but rather of a large collection of data bases that are intended to be used together. This example was chosen as a way to show the complexity level that statistical data bases can reach when the data is collected from a diversity of sources.

The mission of the Energy Information Administration (EIA) of the Department of Energy is to collect, validate and analyze the U.S. energy data, and distribute statistical and analytical reports. The data is collected from electric utilities, oil producers, petroleum product retailers, oil refineries, natural gas pipeline companies, and a variety of businesses in the U.S. The data is collected by using fairly complex forms. The information collected with each form (or sometimes a set of forms) is assembled into a data base, called an energy information system. There are 147 such energy systems that the EIA collects and analyzes, although if one considers other government agencies who deal with energy data the total is estimated at about 230.

The data is organized into four major areas of energy sources: coal, petroleum and petroleum products, natural gas, and electricity. Each area consists of about 20-80 energy systems (or data bases), and produces

Random Access by key or equivalent

Columns (i) of Table 2 refers to fixed length tuple, retrieved in a single disc access by TID. Calculations for our model follow those made for Table 1.

Over-flow	tuples	SYSTEM R		Our Model (PRECI)	
		(i)	(ii)	(i)	(ii)
0%	30K	2.0	3.0	1.0	2.1
30%	39K	2.0	3.0	1.3	2.1
45%	45K	3.0	4.0	1.5	2.1
0%	50K	3.0	4.0	1.0	2.1
30%	65K	3.0	4.0	1.3	2.1
45%	75K	3.0	4.0	1.5	2.1

Table 2

Access in our case will be faster if space is reserved initially in surrogate home slots, which will reduce the overflow.

Addition of new attributes

SYSTEM R will require 2 accesses by TID assuming the original pages as full. This will increase the SYSTEM R figures to those under column (ii). Our figures will change to a fixed 2.1 access (column ii), as explained earlier irrespective of overflows and cardinalities. If the original pages are so populated as to allow some later growth without overflowing (in our case if we require less than 10% larger tuple length initially) then figures of Table 2 could remain unchanged for both the models.

Sequential access

Intuitively sequential access by pkey is faster in our case since no tuple is spread over two pages and since tuples are clustered in surrogate home and overflow slots. We also do not have the overhead of SYSTEM R Prefix with each tuple. Any detailed comparison depends on too many assumptions, and hence not attempted.

In the above comparisons, tuples are assumed to be fixed length. As we have not investigated the problem of variable length tuples in our model, we can not compare this case, but intuitively its affect in our model would be like that of the overflows in Table 2 and in SYSTEM R it would increase the access by TID to more than 1 disc access. There are also other forms of accesses which we have not considered; SYSTEM R could have advantages there, although not necessarily so. It is only fair to point out that these figures do not indicate the overall performance of SYSTEM R, which depends on many factors including query optimisation.

In the earlier versions of the Codasyl model [8] records are given their database keys (data-gates) in four location modes: Calc, Direct, Via and System-defined. In some implementations, database keys contain some physical locations such as area number, page number, page offset etc., despite some more recent claims on the independence of database keys from the physical storage. The location modes are used to generate database keys.

The Calc mode employs hashing on a user-defined Calc key to produce database keys. Random access by the Calc key is fast, taking about 1.33 disc accesses for 30 percent overflows (IDMS, IDS-II), as in our case shown in Table 1. Since the usual practice is to employ the remainder hashing, it is difficult to see how these records can be accessed sequentially by the same key with any efficiency. The storage wastage is implementation-dependent, but usually high, averaging around 30 percent.

In the Direct mode, the database key is taken as the value of a user-defined Direct key. In the event of a clash with an existing database key, the system allocates a different database key; if the user forgets this database key later, then it is his problem. These records can be accessed randomly fast by the Direct key except where there are those clashes, efficient sequential access by the same Direct key should be possible, but can not be guaranteed. Note that there is no concept of primary or unique key in the Codasyl model, and if the values of any Calc or Direct key change later, due to updates, then these database keys can not be found.

In the Via mode the database keys are allocated in such a way, that the records concerned are stored close to those of another record type (set type). This mode permits these records to be accessed fast, in some sequence, in association with the set owner. In the last mode, the user has no control over the database keys and hence over the storage locations of these records. Here the notion of fast access by any particular key does not exist in this case except by user-defined indexes.

In the Codasyl model, storage reorganisation will generally be difficult, irrespective of implementors.

5. CONCLUSION

The surrogate implementation technique presented above provides a fast random and sequential access in primary key order in spite of unordered insertions and deletions, with under 10% storage wastage. Access by secondary keys are not adversely affected, and a large measure of storage independence is provided. The technique is implemented in the PRECI [9,10] database system which is based on a canonical data model capable of supporting relational, Codasyl and other user views.

about 10-15 publications (reports) a year, such as "The Statistics of Publicly Owned Electric Utilities" or "World Crude Oil Production". In addition, there are about 40 energy systems on a variety of other topics, such as solar, automobile classifications, and conservation.

It can be easily seen that the amount of different data elements is very large. Indeed, the number of data elements for natural gas and electric power utilities alone is about 3,600. In total, the number of data elements is around 10,000. The number of elements makes it difficult to remember what they are, let alone to remember the precise acronyms used. What makes it even more difficult is that several data elements have similar meanings, and that the difference may be in the methods used to measure them or the units used. There must be some place where such detail information is kept and managed, or the meaning of the data is eventually lost.

There is a great deal of complexity in validating such data. Validation can be as simple as range checking, or as complex as checking whether certain totals on consumption match (or are within bounds of) the corresponding figures on sales. But even simple range checking is not always straight forward, since the ranges change over time and have yearly trends. Validation across data elements from different data bases is often complicated by the need to perform unit conversion and to check that the parameters for the measured values coincide.

Another need is to guarantee that proprietary information will not be compromised. Much of the information reported by companies and businesses, such as prices, quantities sold, etc., could give a competitive edge to a competitor. Usually, information about an individual company is protected by releasing only global figures (i.e. summarized over a set of individual companies). However, as will be discussed later, a data base can be easily compromised if summarization techniques alone are used.

3. Characteristics

A host of problems can be envisioned from the previous examples, some of them quite formidable. The next sections attempt to describe these problems within categories, such a "modeling", "security", etc. But first some characteristics that are common to SDBs are identified in this section. These characteristics are both in terms of the structure and use of the data.

3.1. Category and summary attributes

It should be clear by now that most 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. Why the distinction? In traditional data management, data is organized into record types, relations, or entities whose columns represent attributes. Both parameter data and measured data are described in terms of their attributes, and no distinction is made.

To illustrate the reasons for this distinction, Figure 3 shows a simple data base represented in a table (relation) form. The first five attributes (oil type, state, county, year, month) represent the parameter data, and the last two (consumption, production) represent measured data. The attributes for the parameter data have been referred to in the literature as "category" attributes, since they contain categories 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 in Figure 3.

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. This relationship between category and summary attributes is key to some modeling techniques, as discussed in a later section on logical modeling.

OIL TYPE	STATE	COUNTY	YEAR	MONTH	CONSUMPTION	PRODUCTION
Crude	Alabama	County 1	1977	Jan	500	800
Crude	Alabama	County 1	1977	Feb	700	600
"	"	"	"	[illegible]	[illegible]	700
"	"	"	"			
"	"	"	"			
"	"	"	"	Dec	.	.
"	"	"	1978	Jan	.	.
.	.	.	"	.	.	.
"	"	"	"			
"	"	"	"	Dec	.	.
"	"	"	1979	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	County 2	.	.	.	.
.	.	.	.	.	.	.
.	Alaska	.	.	.	.	.
.	.	.	.	.	.	.
Heating	.	.	.	.	.	.
.	.	.	.	.	.	.

FIGURE 3: An example of category and summary attributes.

A critical factor of this method is the nature of the pkey value distribution. The method works best if the distribution is reasonably uniform, or can be made uniform (by the key compression algorithm). The overflow mechanism cushions against some non-uniformity. We looked at some actual key values: product codes of a manufacturing concern, and the student numbers of an institution; but both turned out to be rather simple. As indicated earlier we have also developed a generalised key compression technique capable of dealing with most non-uniform distribution reasonably well [11]. Note that we have in fact presented a two-part technique, the parts can be used independently of each other:

(i) Surrogate implementation. Its PINDEX can be a B-tree. Indeed in the PRECI implementation, we have an option of specifying a B-tree instead of a hash tree for the PINDEX of a relation if B-tree is expected to be more efficient for that relation.

(ii) Hash-tree. If the depth of a B-tree is greater than two, then hash tree could be a better alternative for unique-key indexes and probably for non-unique-key indexes. In many machines the basic page size (as unit of input-output) is much smaller than 4096 bytes used in SYSTEM R. For instance in our machine (Honeywell 66/80) it is 1280 bytes, requiring a three-level B-tree for more than 4096 tuples (assuming the key and TID sizes to be the same as those used in the SYSTEM R comparison made earlier).

We intend to examine our technique further in the following areas:

(i) Very highly clustered distribution

(ii) Handling of variable length tuples

(iii) Sharing of data pages by tuples of different relations

(iv) Reorganisation of surrogates for improved performance

(v) Use of hash-tree for non-unique secondary key indexes (currently being studied and implemented).

Finally I would like to thank some of my colleagues in the PRECI project for comments and suggestions - they are: David Bell of Ulster Polytechnic, and Talib Abbod, John Edgar, Dirk Nikodem, Malcolm Taylor and Ambrish Vashishta of Aberdeen University. Many thanks also to Professor D. Kerridge of Statistics at Aberdeen University for assisting me with the probability calculation. The work is partially supported by the U.K. Science and Engineering Research Council.

REFERENCES

1. Hall, P. et al. : "Relations and Entities", Modelling in DBMS, edited by Nijssen (North-Holland 1976).

2. Codd, E. F. : "Extending database relational model to capture more meaning", ACM TODS, vol (4:4), p397, December 1979.

3. Comer, D. : "The ubiquitous B-tree", ACM Computing Surveys, Vol 11, no.2, p121, June 1979.

4. Litwin, W. : "Trie Hashing", Proc. of ACM-SIGMOD, 1981.

5. (i) Astrahan, et al. : "SYSTEM R: Relational Approach to Database Management". ACM TODS, Vol.1, 1976.

   (ii) Astrahan et al. : "A history and evaluation of SYSTEM R", RJ2843 (36129) IBM Research Laboratory, San Jose, California, June 1980.

6. Selinger, P.G. et al. : "Access path selection in a relational DBMS", RJ2429, IBM Research Laboratory, San Jose, California, August 1979.

7. Blasgen, M. W. et al. : "SYSTEM R: An architectural update". RJ2581 (33481). IBM Research Laboratory, San Jose, California, July 1979.

8. Codasyl DDLC Journal of Development, 1978.

9. Deen, S.M., Nikodem, D., Vashishta, A. : "The design of a canonical database system (PRECI)", The Computer Journal, vol(24:3), p200, August 1981.

10. Deen, S. M., Edgar, J.A., Nikodem, D. and Vashishta, A. : "Run-time management in a canonical DBMS: (PRECI)", proc. of 2nd British National Confc. on Databases, Bristol, July 1982, edited by Deen, S.M. and Hammersley, P.

11. Bell, D. A. and Deen, S.M. : "Key space compression and hashing in PRECI" Computer Journal (in press), 1982.

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 was mentioned previously where counties change their boundaries but not their names. 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.

The next sections organize the discussion of problems into research areas. Whenever appropriate, some solutions that have appeared in the literature are mentioned. This is not intended to be a comprehensive list of solutions, but rather to pick some representative solutions that we are familiar with as possible approaches to the problems.

4. Physical Organization

Most of the problems discussed in this section stem from the need to compress the data in large SDBs, while permitting fast access. There are a large number of known compression techniques ranging from coding to intricate text compression. The purpose of this section is to highlight some representative techniques that are particularly applicable to SDBs.

4.1. Category attributes

Whenever several category attributes are used jointly to form a composite key, a large storage overhead results. This point was illustrated previously in Figure 3, where there is much repetition of values in the category attributes columns. Because the category attributes form a cross product, the storage requirements are multiplicative in nature. For each additional category attribute in the composite key, the amount of extra storage required is the size of the storage required for the previous category attributes times the number of distinct categories in the additional attribute. It is therefore quite important that some compression techniques be used.

One common technique to reduce this overhead, is to encode the category values, and to store only the codes with the data base. This can result in great savings, since some category values are descriptive text (for example, the industrial categories shown in Figure 1). Furthermore, the amount of storage needed for the category values depends on the number of distinct category values. Thus, only one bit is necessary to encode the two values of sex, and only four bits for the twelve values of months. Two example systems that were specifically designed to manage SDBs, use this technique: the RAPID system [Turner et al 79], and the ALDS system [Burnett & Thomas 81]. As was pointed out in [Gey 81] it is unfortunate that many systems which use encoding, do not provide software for the automatic translation between the original values and the encoded values, but rather leave the burden on the user to determine which codes to use before querying the data base.

The previous technique still requires that the encoded values be stored repeatedly. Another approach is to use the logical extension of the matrix storage form. One can store the list of distinct category values of each attribute once (perhaps in a dictionary). Then, each category attribute can be used to form one dimension of a multi-dimensional matrix. For each combination of values from the category attributes, one can compute the appropriate position in the multi-dimensional matrix. There is a well-known algorithm for such a mapping (called "array linearization"); its use for category attributes is explained in [Eggers & Shoshani 80]. It is worth noting that the mapping is a simple computation, and therefore random access is essentially achieved.

4.2. Sparse data

As was discussed in Example 4 on world trade, SDBs can be quite sparse. The greater the sparseness, the greater the chance that longer sequences of null values can be found in the data. But, in addition, experience suggests that in SDBs null values (or other designated constants) tend to cluster. To see the reason for this, refer back to figure 3. Suppose that a certain state does not consume a certain oil type. Then in the consumption column there would be zero (or null) values in consecutive positions for all the counties in that state, for all years, for all months. Of course, the order of the category attributes will change the length of the null sequences.

This brings up the following interesting problem: given a certain order of the category attributes and given the precise layout of the corresponding measured values, find an efficient algorithm for determining the best reordering of the category attributes such that the length of null sequences is maximized.

The length of sequences is very important since compression techniques can take advantage of them by essentially replacing a sequence with a count and a value. This technique (called run length encoding) can result in substantial reductions in the size of the data, depending on the sparseness of the data base. The main problem with this technique is the need to access the data sequentially once it is compressed. The ability of random access according to relative position is lost. In [Eggers & Shoshani 80] a technique was developed where logarithmic access can be achieved for data whose null sequences have been compressed. The technique, called "header compression", makes use of a header which contains the counts of both compressed and uncompressed sequences in the data stream. The counts are organized in such a way as to permit a logarithmic search over them. A B-tree is built on top of the header to achieve a high radix for the logarithmic access. In a later paper [Eggers et al 81] the technique was extended to sequences of multiple constant values.

This header compression technique is also used to compress sequences of values that vary in size requirements (i.e. one byte, two bytes, etc.). This can be useful in the case where the distribution of summary attribute values is skewed in such a way that the majority of the values are small. As an example, consider seismic activity measurements where most of the measurements consist of low level background noise.

4.3. Transposed files

The tendency for clustering of null values often occurs within a single column (representing a single summary attribute). This suggests that from a compression point of view, it is advantageous to transpose files, i.e. to store values by attribute, rather than as records or tuples. As discussed in [Teitel 77] and [Turner et al 79], there are other reasons to prefer transposed files (sometimes called "attribute partitioning" or "vertical partitioning") in SDBs. It is argued that in SDBs very few attributes are requested in a single query, and it is inefficient to access data organized as records, since it is necessary to read the data of the other attributes which are of no interest from secondary storage. Another approach is to cluster the attributes which are likely to be accessed together, but it is not a simple matter to determine the preferred clustering from a set of representative queries [Hammer & Niamir 79]. Fully transposed files (i.e. no clustering of attributes) are used in the RAPID and ALDS systems mentioned above, and in earlier systems such as IMPRESS [Meyers 69] and PICKLE [Baker 76].

4.4. Partial cross product

The problem of efficiently storing the cross product of category attributes was discussed above. However, there are situations where not every possible combination of the category attributes is valid, i.e. for the combinations that are not valid, the values for all the summary attributes are null. In such a case, the entire entry is missing from the data base. An example of this was shown in Figures 1 and 2 where for each state many industrial categories are missing altogether. This situation is referred to as the "partial cross product".

The problem is to determine whether there is a way to compress partial cross products. Clearly, the method of value encoding still works, but is there a way to further compress the combinations of category attributes which are valid? In [Svensson 79] a technique which involves the use of a tree is suggested, but some redundancy of values is still left. In [Eggers et al 81] another solution is suggested. It combines the array linearization technique used for full cross product, and the header compression technique for null sequences. Imagine a vector of "ones" and "zeros" that corresponds to valid and invalid entries in the partial cross product, respectively. The partial cross product is treated as if it was a full cross product, and array linearization is used to map into this imaginary vector. Then, the header compression mapping is used to map from the imaginary vector into the actual positions of the valid entries. The outcome of this combination is that just a header is necessary to perform the entire mapping and to achieve a logarithmic access time.

5. Optimization

5.1. Selection of physical structures

Because of their special nature, SDBs offer opportunities for storage and access savings that use uncommon physical structures and access methods. In addition to the more common techniques, such as a variety of indexing and hashing methods, the gains that can be achieved by using techniques such as attribute partitioning (either full or clustered partitioning), encoding, array linearization, and different compression techniques must be considered.

Given a statistical data base and a set of representative queries, the problem is to determine how to partition, compress, or encode the data, and which access methods to use to access the data. In general, this is a very difficult problem, even with a limited set of choices. However, there is still the challenge of solving this problem for SDBs, given a small set of storage and access methods which seem most likely to benefit the application. A representative example of such work is described in [Lehot 77], where the problem of determining the optimal order of the category attributes in order to maximize the access time of a summary attribute was considered. The paper shows that under certain assumptions about the distribution of values and query characteristics, an optimal order can be found. It also contains several references to related work.

Of course, the complementary problem to that of selecting data structures for an application, is that of processing queries efficiently.. Given that certain choices were made for the physical organization of a SDB, how can queries be processed optimally. Query optimization is still an active area in conventional data base research. The question is whether the known techniques can be applied to the more uncommon structures that are useful for SDBs.

5.2. Response assembly

Attribute partitioning and compression introduce the problem of assembling the response to a query. Suppose that a response needs to be extracted from several attributes that were partitioned, and that the partitions are stored on a disk. In order to assemble tuples (records) for the response, a value from each partition could be read, and the appropriate values written a tuple at a time. But if the primary buffer space is limited, this would result in an excessive number of reads to the disk. The reason is that a single value from each page is read in order to assemble a tuple. If, in addition, the values in the attribute partition are compressed, there is an overhead to be paid for each access to the compressed data. Thus, given a limited buffer, the problem is to minimize disk reads and the overhead of accessing compressed data.

In a paper published in this proceedings, a suggestion is made that special hardware can be used for the purpose of tuple assembly and compression [Hawthorne 82]. It suggests the use of microprocessors that are organized in a two level hierarchy. The leaf microprocessors are responsible for writing and reading the compressed data and the top level microprocessor(s) responsible for tuple assembly. Thus, the data from the different partitions could be read in parallel, and passed on for tuple assembly. This work is still in the design stage.

6. Logical modeling

Can benefits be gained from modeling the semantics of statistical data bases? Is it worth adding to the complexity of the data model? There is a long standing controversy as to whether logical data models should be semantically simple (such as the relational model), or whether they should contain more semantics about the data structures (such as having generalization hierarchies or distinguishing between entities and relationships). In the case of SDBs, the question is whether to model data types such as "matrix" and "time series", and concepts such as category and summary attributes.

6.1. Representation of category and summary attributes

This section points out some of the work done in modeling of category and summary attributes, and the benefits achieved. It is worth noting that practitioners make a distinction between parameters (which correspond to category attributes) and variables (which correspond to summary attributes) because it provides a better understanding of the content of the data base and how it was established. For example, in a scientific experiment, the parameters that can be set by the experimentor are referred to as the "independent variables", and the measured data as the "dependent variables".

One of the main benefits of modeling the semantics of category and summary attributes is the capability of "automatic aggregation". It is the ability of the system to infer the subsets of values over which an aggregation (or statistical) function should be applied. For example, consider the following query when applied to the data base in Figure 3: "find heating oil consumption in Alabama during 1977". It is obvious that the result should be the total heating oil consumption over all counties in Alabama and over all months in 1977. Yet, without the explicit semantics of category and summary attributes the system would not be able to infer what is obvious to us. The benefit to the user is that it is not necessary to explicitly express which category attributes to summarize over. This can greatly simplify aggregation expressions in query languages.

An example of adding the above mentioned semantics to an existing model is described in [Johnson 81]. Using the framework of the Entity-Relationship model, an additional type of entity is allowed, called a summary set, which captures the semantics of category attributes. In addition, an attribute which is designated as a summary attribute, can have an aggregation function (e.g. sum, average) or any other desired function (defined as a program) associated with it.

6.2. Graph representation

Another possibility is to have these semantic concepts represented internally, so that they are invisible to the user. An example of a system that takes this approach is SUBJECT [Chan & Shoshani 81], in which these semantic concepts are represented as a graph. There are two kinds of nodes: a "cross product" node, and a "cluster node". The nodes can be connected by arcs to form a directed acyclic graph. To illustrate the meaning of these node types, we have represented the data base in Figure 1 as the graph shown in Figure 4. Note that nodes marked "x" are cross product nodes, and those marked "c" represent cluster nodes.

Cluster nodes represent collections of items. Thus, the node "metal mining" at the bottom of the figure, represents the collection of iron ores, lead & zinc ores, etc. The node "metal mining" itself is one of the items under the node "mining". As can be seen, cluster nodes are used to represent a hierarchy of parameters. This is a way of representing the complex category attribute "industrial classes". Cluster nodes are also used to represent the collection of summary attributes under the node labeled "variables".

Cross product nodes are used to represent composite keys of category attributes. Such is the case with the node "state by industry". The semantics of this cross product node is such that each of its instances is made up of a pair of instances, one taken from the node "industry" and one from the node "states".

This graph structure is invisible to the user and is used to support a menu driven interface. The user does not need to know the types of nodes, but the system can make use of them to provide automatic aggregation. The graph can be either browsed by moving up and down the nodes, or can be searched directly with key-words. The sharing of nodes provides the capability to use the same clusters (e.g. state names) across data sets, and to avoid confusion of names. One of the main advantages of this representation is that the user can be shown the content of the data base by gradually revealing more detail when requested. The possibility of viewing hierarchical menus of details alleviates the need to remember names and acronyms.

6.3. Summary sets

Summary sets are simply data base views that are generated by using aggregate functions. The main problem is one of managing a large number of sets. With each summary set new summary attributes are generated. Obviously, the newly computed values of the summary attributes have to be stored if recomputing them every time they are needed is to be avoided. But, is there a way to avoid duplicating the category attribute values? Similarly, new names are likely to be used for the new summary attributes (e.g. "total consumption" when we summarize over consumption), but is there a way of using the same names of category attributes in the summary set?

This is another situation where distinguishing between the type of attributes can be beneficial. If the category attributes are organized as lists of category values (say, in a dictionary), then it is possible for the category attributes of the summary sets to "point" to these lists, and to share the same names. It is easy to visualize this point in terms of the SUBJECT graphs described above. If a category attribute is used in its entirety in the summary set, then a pointer to the corresponding node is all that is necessary. If a selection of a certain category is made, e.g. "Alabama", then the pointer points directly to the node representing "Alabama". If a selection of a subset of the category values was made (e.g. several states), then a new node is created whose members are that subset.

This idea is complementary to the technique described in the section on physical organization above, where the lists of category attribute values are stored only once, and array linearization is used to map between them and the appropriate positions of the summary attributes.

6.4. Aggregation/disaggregation

In Example 3 above on geographical categories, the difficulties of correlating data from different data bases on the basis of a common but not identical category attribute were described. This problem is referred to as "aggregation/disaggregation" because, as shall be seen shortly, it involves both functions. For example, suppose that unemployment rates are to be correlated with median family income. The problem is that unemployment rates may be available by federal regions, but median family income may be available only by census regions. Both federal and census regions represent groups of states, but the groupings are different. In order to perform the correlation task, it is necessary to disaggregate one of the summary attributes (e.g. unemployment rates) to the state level using some proxy variable (e.g., population), and then aggregate back to the desired level.

A recent paper [Merrill 82] describes the technique used to perform aggregation/disaggregation in the SEEDIS system which was mentioned in the introduction. It uses tables to represent the mapping between any two sets of category attribute values of interest. It deals only with mappings for geographical categories, but the amount of detail is still large since there are so many different geographic categories used by federal and state government. In [Johnson 81a] a technique is proposed for modeling these mappings by associating them with relationships of the Entity-Relationship model.

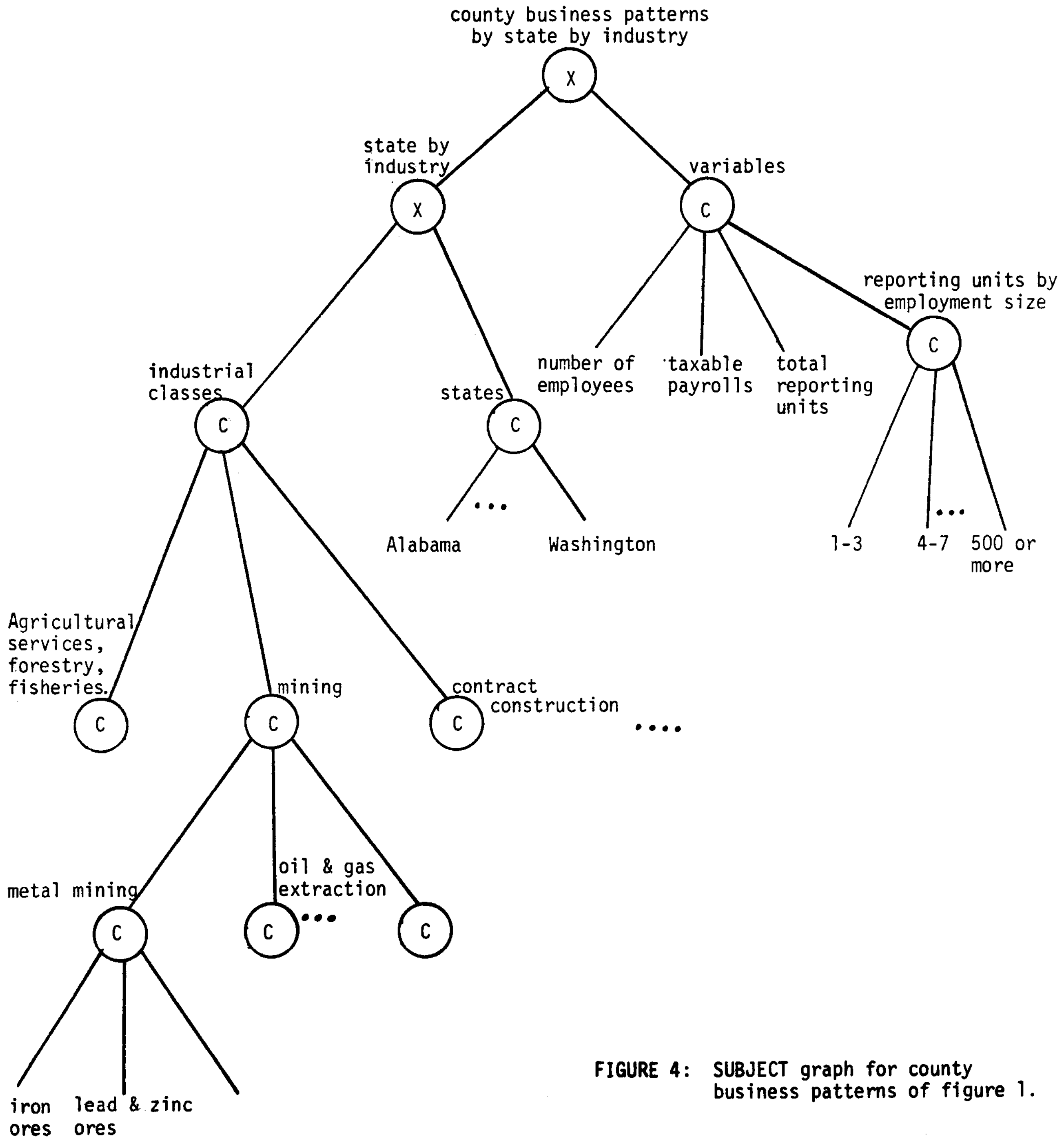


FIGURE 4: SUBJECT graph for county business patterns of figure 1.

7. User Interface

From the Examples section above, it is evident that one of the major problems for a user interfacing to large SDBs is to determine the content of the data base and the terms used for its attributes. Such problems are referred to as meta-data problems, since they deal with information about the data. Meta-data is much more complex than listing the record types (or relations) and the attribute names and types. It includes information such as missing data specification, data quality specification (to indicate how reliable the data could be considered), a history of data base creation and modifications, complex attribute structures (e.g. vectors to represent the boundaries of geographical regions), etc. In a paper published in this proceedings [McCarthy 82], a comprehensive list of requirements for meta-data is given , with special attention given to SDBs.

Whenever it is necessary to deal with the diversity and complexity of data bases such as the energy data of EIA described in Example 5, special techniques of classifying information may be needed. In fact, for the EIA data base, it was helpful to use a technique that is usually used in library systems, called "facet classification". Using this technique, summary attributes are described using facets. For example, some of the facets used for the energy data are: energy source (oil, coal, etc.), function (produced, shipped, etc.), units of measure, dates, etc. Each facet is a hierarchy of terms that can be quite deep, as is the case with energy source. A combination of the terms from the facet hierarchies is then used to describe a summary attribute (for example, heating oil refined in mid-western states during 1977). This technique can avoid conflicts in definition of similar attributes since they have to be defined using terms from predefined facets.

It is interesting to note that the SUBJECT graphs described previously are powerful enough to describe facets, since cluster nodes can represent a facet hierarchy. Indeed, SUBJECT is used to describe meta-data in a hierarchical manner, so that a user can start at a high level (e.g. population data, energy data, etc.), and gradually narrow down to the data set needed.

The distinction between meta-data and data is not always obvious. Information that is stored in the data base can sometimes be thought of as meta-data and vice versa. This is particularly true of the values of category attributes. For example, if a user is inquiring about the content of the data base in Figure 3, it is as natural to ask what are the summary attributes (e.g. consumption) as it is to ask what years are covered or what are the oil types. Again, this argues for associating the list of values of category attributes with a dictionary rather than to store them with the data.

What about query languages? Are there any special problems associated with SDBs? Aggregation is a predominant function that needs to be supported. However, it is perhaps the most awkward function to express in many query languages. In addition to enriching data models to support automatic aggregation, work is being done to simplify the expression of aggregate functions. For example, [Klug 81] proposes an extension to query-by-example in order to support aggregate functions in SDBs. Perhaps a combination of menu driven techniques (such as those used in SUBJECT), graphics techniques (such as described in [Wong & Kuo 82] in this proceedings), and simple command languages can bring about more convenient user interfaces.

8. Integrating statistical analysis and data management

In order to perform statistical analysis, an analyst needs both data management tools and statistical tools. Unfortunately, these tools are not usually integrated into a single system. Data management systems support only a limited number of statistical functions, and statistical packages have limited data management capabilities.

There are three possible approaches to this problem. The first is to enrich statistical packages with more general data base structures and more powerful data management functions. Evidence of this approach can be found in new releases of statistical packages, where many systems now support some kind of hierarchical or network data structure, while past versions supported only "flat files". However, they still lack many functions, such as joining two tables, or supporting summary sets (or views).

The second approach is to enrich existing data management systems with tools useful to an analyst, such as taking random samples, and a library of statistical operators. An example of this approach is described in [Hideto & Kobayashi 81], where statistical facilities are added to a commercial data management system, Model 204.

The third approach involves interfacing statistical packages to data management systems. There are three variations to this approach. the first is to tightly couple each pair of systems. Usually a pair is selected for an application and is expected to last a long time. One such experience is described in [Weeks et al 81]. The second variation involves defining a standard data format that all systems accept. In the long run this is a more effective method to implement since each new system added is only required to communicate with the standard format in order to communicate to all systems. However, this technique may be less efficient in terms of processing time since two translators are involved, unless changes can be made to the software of the systems involved. This approach was taken in the SEEDIS project mentioned above, where a fairly simple standard, called CODATA was quite successful in integrating several components of the system. An essential feature of such a standard is that it is self describing, i.e. that data bases carry their own data definition. The third variation involves a monitor that takes care of interfacing the systems, but presents the user with the impression of a single system. an example of this variation is described in [Hollabaugh & Reinwald 81].

An important point to note is that regardless of the approach taken, it is quite essential that statistical operations should produce self-describing data structures that contain meta-data as well as data. Analysts have been burdened by having to keep hand-written documentation of the meta-data as they perform the analysis. As the analysis process progresses, it becomes increasingly difficult to keep track of these meta-data descriptions.

It is not clear which of the above approaches is the most successful. Perhaps future systems can be designed from the start to accommodate both statistical analysis and data management needs. The system S [Becker & Chambers 80] was designed with this goal in mind. It also uses a certain form of self-describing data structures.

9. Security

Security problems in statistical data bases arise from the wish to provide statistical information without compromising sensitive information about individuals. Providing the necessary security is also referred to as inference control since it is intended to prevent the inference of protected information from any collection of legitimate statistical queries.

Here again, it is convenient to distinguish between category and summary attributes. It is customary to consider the individual values of the summary attributes as the ones that should be protected. Consider, for example, a medical experiment in which individuals are treated with a certain drug and its effect are measured in terms of blood pressure, body temperature, etc. Other information about the individuals may be collected such as medical history, living conditions, income, etc. Clearly, some of the information is sensitive such as income or medical history. Suppose that the correlation between blood pressure and income is to be explored. A legitimate query might be "find the average blood pressure on a certain date for males whose ineome is over 50000". Date, sex and income are the "selecting" or category attributes, while blood pressure is the summary attribute over which a statistical operation is applied. Thus, "blood pressure" is the summary attribute that should be protected.

Unfortunately, it is not sufficient to protect only the summary attributes. Suppose, for example, that the blood pressure of some individual is known to an intruder. By repeated application of the above query at different income levels, he can infer information about the income of the individual, i.e. information about a category attribute can also be inferred. Furthermore, attributes can change their roles between category and summary. For example, in the query "find the average income of males whose blood pressure exceeds a certain level", income is the summary attribute, while blood pressure is the category attribute.

There is an extensive list of papers that discuss problems and propose protection mechanisms for statistical data bases. Several of these mechanisms are briefly discussed here. Interestingly, some of the important results are negative. The different techniques are applicable to different circumstances and needs.

9.1. Limiting the query set

One of the more obvious techniques is to limit the number of cases (individuals) that qualify as a result of a query, called a query set. If the query set is below a pre-specified threshold, then the statistical operation should not be applied over it and the query is refused. However, this technique is quite useless. Intuitively, several queries can be issued whose query sets overlap, until a desired individual value can be inferred. Indeed, this was not only shown to be the case [Schlorer 75, Denning et al 79], but an efficient procedure for finding the snooping queries (called a "tracker") has been devised [Denning & Schlorer 80].

9.2. Limiting intersection of query sets

To remedy this deficiency another approach has been suggested, where the system keeps track of previous queries and verifies that a new query request does not intersect excessively with previous queries [e.g. Dobkin et al 79]. Analyzing audit trails is not a simple matter, and techniques have been proposed for keeping track of the query sets of previous queries instead [Chin & Ozsoyoglu 81]. A new query is allowed as long as the intersection between its query set and those query sets previously answered do not fall below a pre-selected threshold. The main problem with this approach is that whether or not a certain query can be answered depends on the previous queries issued since the time that the data base was created. Also, for large data bases the number of previously accessed data sets may become quite large. This technique is thought to be advantageous for relatively small databases, such as medical experiments where the number of subjects is limited. In contrast, the technique described next is useful only when requested query sets are large.

9.3. Random sample queries

This technique, proposed by [Denning 80], applies the statistical operation on a set of values drawn randomly from the query set. This makes it impossible for users to control precisely the query set from which the responses to their queries are drawn. Clearly, for such an approach to be successful, the query sets requested must be large enough to allow responses based on random samples to be statistically meaningful. Thus, this approach is only useful with large data bases, and for applications whose typical query sets are large.

9.4. Partitioning of the data base

In many applications it is possible to pre-determine that groups of individuals (cases) should be always accessed together for statistical purposes [e.g. Yu and Chin 77]. The individuals within the groups cannot be accessed. In fact, a form of this technique, where the values for the groups are pre-aggregated, is probably the most widely used. Such is the case in census data bases, where information about individuals is prohibited by law. Only pre-aggregated data is available to users. One of the problems associated with such an approach is that sometimes the groups, as defined, may contain only a few individuals, and therefore one can infer information about individuals from the aggregated data. For example, if there are very few American Indian families living in a certain area, then aggregating over racial groups may reveal information about the individual families. This forces values to be suppressed from the data base as long as the number of individuals in a group is below a certain level.

9.5. Perturbing data values

There are two approaches to data perturbation: perturbing output values before presenting them to the user, and perturbing the actual values stored in the data base. Example papers of output perturbation are [Haq 77] and [Achugbue & Chin 78]. [Beck 80] discusses techniques for perturbing the stored values, and also covers previous work in the area.

The main difficulty with this approach is to insure that the error introduced is within acceptable bounds. There is a trade-off between the level of security that can be achieved and the variance of perturbation introduced. With a sufficient number of overlapping queries it is possible to narrow the range of values for an individual. The challenge is in developing techniques that can provide fairly accurate statistical responses, while keeping the inference of the range of individual values sufficiently large for security purposes.

To summarize, providing security in statistical data bases is a difficult problem. There seems to be no single general solution. Therefore, numerous techniques have been suggested that can be used for different applications and needs. Much of the research so far has been applied only with restricted assumptions, such as only SUM queries, or only a single case (record) for each element of the cross product of the category attributes [Kam & Ullman 77]. There is still active research in this area.

10. Concluding remarks

The purpose of this paper was to describe the characteristics and problems that exist in statistical data bases, and to highlight some of the interesting work now emerging in this area. It is inevitable that some work may have been overlooked, especially since this is an inter-disciplinary area.

New research efforts are now emerging in universities, such as the University of Florida and the University of Wisconsin. In addition, there are continuing efforts in laboratories and institutions, such as Lawrence Berkeley Laboratory, Lawrence Livermore Laboratory, Battelle Pacific Northwest Laboratory, Statistics Canada, Bureau of Labor Statistics, and Bell Laboratories.

The area of statistical data bases is quite important in that it encompasses a large variety of application areas that usually deal with large amount of data. Much of these data are now uselessly archived as there are no appropriate tools for their management and analysis. This area poses interesting, challenging, and real problems that should be addressed by data base researchers.

Acknowledgement

I would like to acknowledge the help received from Deanne Merrill and John McCarthy in selecting some of the examples. I am also grateful for the encouragement and support of many of my colleagues at Lawrence Berkeley Laboratory. This work was supported by the Applied Mathematical Sciences Research Program of the Office of Energy Research, U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

References

[Achugbue & Chin 78] Achugbue, J.O., Chin, F.Y., Output Perturbation for Protection of Statistical Data Bases, Dep. Computing Science, University of Alberta, Canada, January, 1978.

[Baker 76] Baker, M., User's Guide to the Berkeley Transposed File Statistical System: PICKLE, Technical Report No.1, 2nd ed., University of California, Berkeley, Survey Research Center, 1976.

[Beck 80] Beck, L.L., A Security Mechanism for Statistical Databases, ACM Trans. Database Syst. 5, 3, (Sept. 1989), pp. 316-338.

[Becker & Chambers 80] S: A Language and System for Data Analysis, Bell Laboratories, July 1980.

[Boral et al 82] Boral, H., DeWitt, D.J., Bates D., A Framework for Research in Database Management for Statistical Analysis, Proceedings of the ACM SIGMOD International Conference on Management of Data, June 1982.

[Bragg 81] Data Manipulation Languages for Statistical Databases -- The Statistical Analysis System (SAS), Proceedings of the First LBL Workshop on Statistical Database Management, Dec. 1981, pp. 147-150.

[Burnett & Thomas 81] Burnett, R. A., and Thomas J. J., Data Management Support for Statistical Data Editing and Subset Selection, Proceedings of the First LBL Workshop on Statistical Database Management, Dec. 1981, pp. 88-102.

[Chan & Shoshani 81] Chan, P., Shoshani, A., Subject: A Directory driven System for Organizing and Accessing Large Statistical Databases, Proceedings of the International Conference on Very Large Data Base (VLDB), 1980, pp. 553-563.

[Chin & Ozsoyoglu 81] Chin, F.Y., Ozsoyoglu G., Auditing and Inference Control in Statistical Databases, March 1981, to appear in IEEE Transactions on Software Engineering.

[Cohen & Hay 81] Why Are Commercial Database Management Systems Rarely Used for Research Data? Proceedings of the First LBL Workshop on Statistical Database Management, Dec. 1981, pp. 132-133.

[Denning et al 79] Denning, D.E., Denning, P.J., Schwartz, M.D., The Tracker: A Threat to Statistical Database Security, ACM Trans. Database Syst. 4, 1, (March 1979), pp.76-96.

[Denning 80] Denning, D.E., Secure Statistical Databases with Random Sample Queries, ACM Trans. Database Syst, 5, 3, (Sep. 1980), pp. 291-315.

[Denning & Schlorer 80] Denning, D.E., Schlorer, J., A Fast Procedure for Finding a Tracker in a Statistical Database, ACM Trans. Database Syst. 5, 1, (March 1980), pp. 88-102.

[DHEW 75] U.S. Department of Health, Education, and Welfare, Comparability of Mortality Statistics for the Seventh and Eighth Revisions of the International Classification of Diseases, DHEW Publication 76-1340, Oct. 1975.

[Dobkin et al 79] Dobkin, D., Jones, A.K., Lipton, R.J., Secure Databases: Protection Against User Influence ACM Trans. Database Syst. 4, 1, (March 1979), pp. 97-106.

[Eggers & Shoshani 80] Eggers, S. J., Shoshani, A. "Efficient Access of Compressed Data," Proceedings of the International Conference on Very Large Databases, 6, 1980, pp. 205-211.

[Eggers et al 81] Eggers, S., Olken, F., Shoshani, A., A Compression Technique for Large Statistical Databases, Proceedings of the International Conference on Very Large Data Base (VLDB), 1980, pp. 205-211.

[Gey 81] Gey, F.G., Data Definition for Statistical Summary Data or Appearances Can Be Deceiving, Proceedings of the First LBL Workshop on Statistical Database Management, Dec. 1981, pp. 3-18.

[Hammer & Niamir 79] Hammer, M., Niamir, B. "A Heuristic Approach to Attribute Partitioning," ACM SIGMOD Proceedings of the International Conference on Management of Data, Boston, 1979, pp. 93-101.

[Haq 77] Haq, M.I., On safeguarding Statistical Disclosure by Giving Approximate Answers to Queries, Int. Computing Symp., 1977, pp. 491-495.

[Hollabaugh & Reinwald 81] Hollabaugh L.A., Reinwald, L.T., GPI: A Statistical Package / Data base Interface, Proceedings of the First LBL Workshop on Statistical Database Management, Dec. 1981, pp. 78-87.

[Hawthorne 82] Hawthorne, P., Microprocessor Assisted tuple access, decompression and assembly for statistical database systems, Proceedings of the International Conference on Very Large Data Base (VLDB), 1982.

[Hideto & Kobayashi 81] Hideto, I., Kobayashi, Y., Additional Facilities of a Conventional DBMS to Support Interactive Statistical Analysis, Proceedings of the First LBL Workshop on Statistical Database Management, Dec. 1981, pp. 25-36.

[Johansson & Shilling 81] Johansson, J.H., Shilling, J.D., Toward the Development of an Integrated Economic data Base at the World Bank, Proceedings of the First LBL Workshop on Statistical Database Management, Dec. 1981, pp. 39-40.

[Johnson 81] Johnson, R.R., Modelling Summary Data, Proceedings of the ACM SIGMOD International Conference on Management of Data, 1981, pp. 93-97.

[Johnson 81a] Johnson, R.R., A Data Model for Integrating Statistical Interpretations, Proceedings of the First LBL Workshop on Statistical Database Management, Dec. 1981, pp. 176-189.

[Kam & Ullman 77] Kam, J.B., Ullman, J.D., A Model of Statistical Databases and Their Security, ACM Trans. Database Syst. 2, 1, (March 1977), pp.1-10.

[Klug 81] Klug, A., Abe -- A Query Language for Constructing Aggregates-by-example, Proceedings of the First LBL Workshop on Statistical Database Management, Dec. 1981, pp. 190-205.

[Lehot 77] Lehot, P., Misuki, M., Rosenthal, A., Szabo, S., On the Optimal Attribute Ordering for an Indexed Sequential File Organization, IEEE Asilomar Conference on Computer Systems, 1977.

[McCarthy 82] McCarthy J., Meta data Management for Large Statistical Databases, Proceedings of the International Conference on Very Large Data Base (VLDB), 1982.

[McCarthy et al 82] McCarthy, J.L., Merrill, D.W., Marcus, A., Benson, W.H., Gey, F.C., Holmes, H., Quong, C., The SEEDIS Project: A Summary Overview of the Social, Economic, Environmental, Demographic Information System, Lawrence Berkeley Laboratory document PUB-424, April 1982.

[Merrill et al 79] Merrill, D., Levine, S., Sacks, S., Selvin, S., PAREP: Populations at Risk to Environmental Pollution, Lawrence Berkeley Laboratory Document LBL-9976, October 1979.

[Merrill 82] Merrill, D., Problems in Spatial Data Analysis, Proceedings of the Seventh Annual SAS Users Group International Conference, San Francisco, Feb. 1982.

[Meyers 69] Meyers, E.D. Jr., Project IMPRESS: Time Sharing in the Social Sciences, AFIPS Conference Proceedings of the Spring Joint Computer Conference, Vol. 34, 1969, pp. 673-680.

[Nie et al 75] Nie, N.H., et al, SPSS: Statistical Package for the Social Sciences, Second Edition, McGraw Hill, New York, 1975.

[SAS 79] SAS Institute, Inc., SAS User's Guide, 1979 Edition, Raleigh, North Carolina, 1979.

[Schlorer 75] Schlorer, J., Identification and Retrieval of Personal Records from a Statistical Data Bank, Methods Inform. in Medicine 14, 1, (Jan. 1975), pp. 7-13.

[Svensson 79] Svensson, P. On Search Performance for Conjunctive Queries in Compressed, Fully Transposed Ordered Files, Proceedings of the International Conference on Very Large Databases, 5, 1979, pp. 155-163.

[Teitel 77] Teitel, R.F., Relational Database Models and Social Science Computing, Proceedings of Computer Science and Statistics: Tenth Annual Symposium on the Interface, Gaithersburg, MD, National Bureau of Standards, April 1977, pp. 165-177.

[Turner et al 79] Turner, M. J., Hammond, R. and Cotton, F. A DBMS for Large Statistical Databases, Proceedings of the International Conference on Very Large Databases, 5, 1979, pp. 319-327.

[Weeks et al 81] Weeks, P., Weiss, S., Stevens, P., Flexible Techniques for Storage and Analysis of Large Continuing Surveys, Proceedings of the First LBL Workshop on Statistical Database Management, Dec. 1981, pp. 310-311.

[Wong & Kuo 82] Wong, H.K.T., Kuo, I., A Graphical User Interface for Database Exploration, Proceedings of the International Conference on Very Large Data Base (VLDB), 1982.

[Yu and Chin 77] Yu, C.T., Chin, F.Y., A Study on the Protection of Statistical Databases, ACM SIGMOD Int. Conf. on Management of Data, 1977, pp. 169-181.