Title

PROCEEDINGS OF THE SECOND INTERNATIONAL WORKSHOP ON STATISTICAL DATABASE MANAGEMENT, SEPT. 27-29, 1983, LOS ALTOS, CALIF.

Authors

Hammond, Roy
McCarthy, John L.

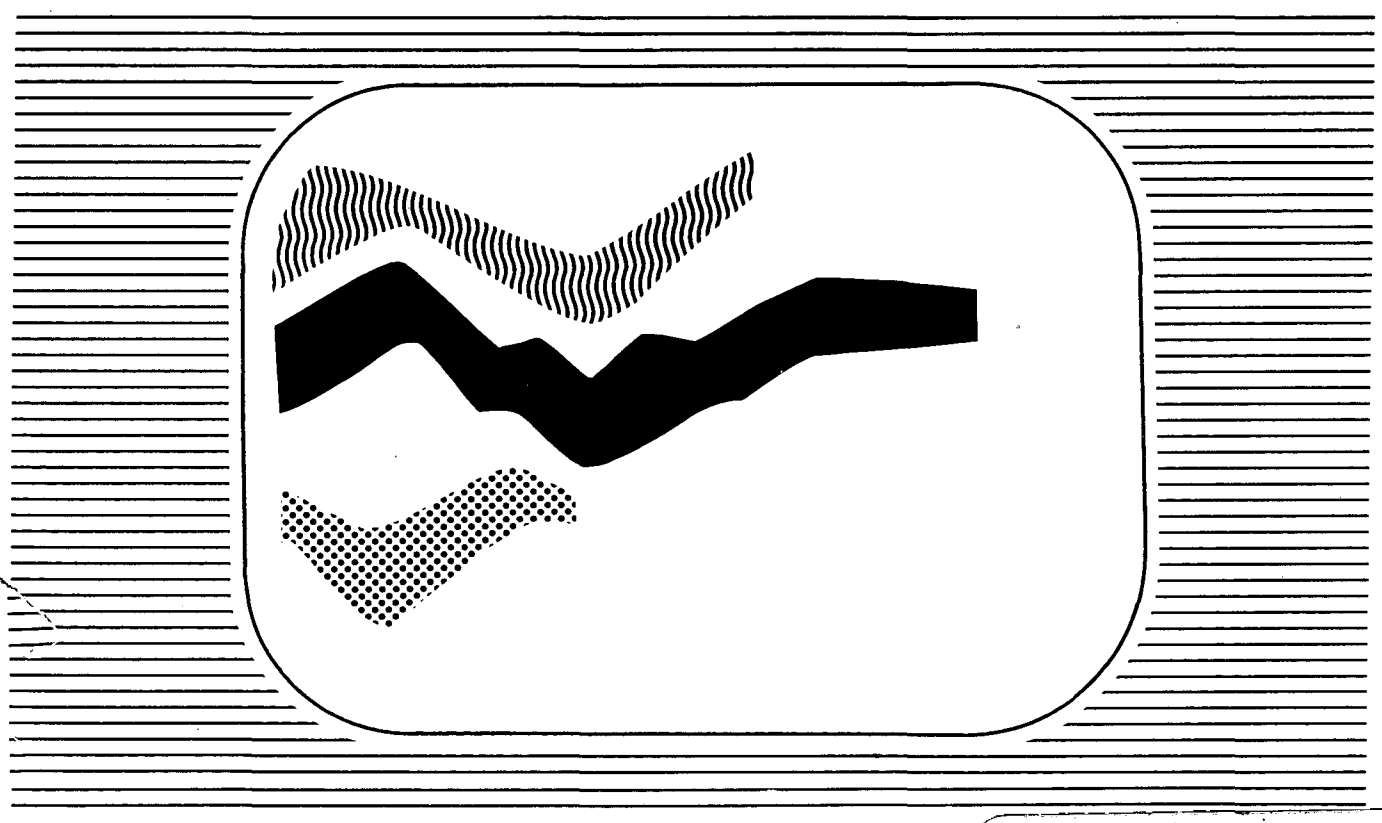
Publication Date

1984-01-17

LBL-16321 c. 1
UC-13
CONF-830950

Proceedings of the Second International Workshop on Statistical Database Management

September 27–29, 1983 Los Altos, California



Sponsors:

Lawrence Berkeley Laboratory, University of California

U.S. Department of Energy, under Contract No. DE-AC03-76SF00098

In Cooperation With:

Association for Computing Machinery, Special Interest Group on Management of Data

American Statistical Association, Statistical Computing Section

IEEE Computer Society, Technical Committee on Database Engineering

Statistics Canada,

DISCLAIMER

Proceedings of the Second International Workshop on Statistical Database Management

September 27–29, 1983 Los Altos, California

Roy Hammond and John L. McCarthy, Editors

Sponsors:

Lawrence Berkeley Laboratory, University of California

U.S. Department of Energy, under Contract No. DE-AC03-76SF00098

In Cooperation With:

Association for Computing Machinery, Special Interest Group on Management of Data

American Statistical Association, Statistical Computing Section

IEEE Computer Society, Technical Committee on Database Engineering

Statistics Canada,

PREFACE

The Second International Workshop on Statistical Database Management will be held September 27-29, 1983 in Los Altos, California. The purpose of this workshop is to bring together statisticians and computer scientists, statistical database system users and system builders, to exchange ideas on statistical database management. "Statistical databases" contain statistical information or are used for statistical analysis, and they present recognized problems that current data management and statistical software do not fully address.

The goals of this workshop are the same as its predecessor held in December 1981, but its organization differs in a number of ways. The proceedings are being published prior to the Workshop so that they can: 1) report research results and work in progress, 2) provide an intellectual introduction to most of the workshop participants, 3) provide a point of departure for working group discussions. Only a few of the papers will be presented orally at plenary sessions of the three-day workshop. Working groups of five to ten participants each will meet the first two days to discuss and draft reports on selected topics which have been used to organize these proceedings into sections. These reports will be presented on the final day of the workshop and published in the winter, 1984 IEEE *Database Engineering Newsletter.*

In response to our call for papers, 18 foreign and 42 U.S. papers were submitted. Each paper was evaluated by at least three program committee members, and 46 papers were accepted. In addition, 10 unrefereed papers were solicited for the section on *Time Series and Econometric Database Management* as part of a special working group.

Participation in the Workshop is by invitation, and includes all authors of accepted papers, members of the special Time Series and Econometric working group, and the program committee members. About 80 statisticians, database researchers, system developers, and others are expected to attend.

As organizers of the 1983 workshop, we would like to extend our thanks to all authors who submitted papers for consideration; to program committee members who read papers and made valuable suggestions for improvement; to Pam Weeks and Gwen Harlee who organized the special econometric working group; to Carl Quong, LBL's Computing Science and Mathematics Department Chairman, and Jean-Paul Trudel, Director-General at Statistics Canada, who supported the workshop since its inception; to Julia Snyder, Carole Agazzi, Lesta Nadel, Meri Jones and Virginia Sventek at Lawrence Berkeley Laboratory; and finally to Mike Jeays, Dave Emery and Shirley Jones at Statistics Canada without whom the workshop could not have taken place.

John L. McCarthy,
General Chairman

Roy G. Hammond,
Program Committee Chairman

ORGANIZING COMMITTEE

General Chairperson

John L. McCarthy, *Lawrence Berkeley Laboratory, USA*

Program Chairperson

Roy G. Hammond, *Statistics Canada, Canada*

Program Committee

Rick Becker, *Bell Telephone Laboratories, USA*

Francis Chin, *University of Alberta, Canada*

Ivor Francis, *University of Otago, New Zealand*

Jerome Friedman, *Stanford Linear Accelerator Center, USA*

James Gentle, *IMSL, Inc., U.S.A*

Ron Helms, *University of North Carolina, USA*

David Hoaglin, *Harvard University, USA*

Gregory A. Marks, *Institute for Social Research, USA*

Wes Nicholson, *Pacific Northwest Laboratory, USA*

Gordon Sande, *Statistics Canada, Canada*

Diane C.P. Smith, *Computer Corp. of America, USA*

Peter Stevens, *Bureau of Labour Statistics, USA*

Stanley Su, *University of Florida, USA*

Harry Wong, *Lawrence Berkeley Laboratory, USA*

CONTENTS

1. How do Analysts Work? User Interface Issues

See Also. . . .

DATA-BASES FOR CLINICAL HISTORIES.

Anthony D. ELLIMAN.
Brunel University, Uxbridge, ENGLAND.

Medical researchers frequently investigate the progress of a disease or treatment by collecting a number of case histories. Data only becomes available when suitable patients present themselves and each patient may need to be followed for some years. A study of this type is a long term exercise. Such studies are only made possible by the availability of computers for data collection and analysis.

Several data-base systems are available for clinical research, but many of these do not give enough attention to the dynamic aspects of this type of work. To identify the problem areas models for both the investigator's and patient's interaction with the data-base are presented.

1. INTRODUCTION

In clinical research the data-base will hold medical histories from known cases. These case histories accumulate from clinical contacts with patients, and as such represent the investigator's view of the world.

In medical research the data are subjected to analysis to find evidence to support hypotheses. Care is required to ensure that the process of selecting and recording data does not distort the record and thereby invalidate the conclusions drawn from it. A significant amount of the information in a data-base comes from the relationships between the items, and this can easily be distorted or lost.

It is important to remember that, like a photograph, the data-base is only a view of reality. Once the picture has been taken, the imperfections and omissions are frozen into the record. If the colours of the trees might be important, colour film must be used; however unless there were an interest in thermal profiles, infra-red film would not be required. In the same way a data-base is only a partial record of reality, whose value depends on the preservation of those facts which may prove to be relevant.

Since a data-base is only a partial reflection of the real world it is important that all its users understand the rules by which data were selected for inclusion. In small groups all users tend to participate in the data collection process and build up a common understanding of the selection rules. As the user group grows or changes, the importance of keeping a record of these rules becomes paramount. When other research teams share the data they must be able to see the rules by which it was collected. One solution to this

problem is to store the rules as part of the data-base, i.e. the system should include a data dictionary.

2. INVESTIGATOR BEHAVIOUR

A significant amount of medical research is concerned with long term problems. In such circumstances there will be a continuing growth in the information and this will present some difficulties. In the early 1970s several American research centers were surveyed by Palley to determine the data processing needs of clinical investigators[7]. This showed that in over twenty percent of the studies, a single patient would be followed for at least 500 days. Since patients enter a sample at different times it seems reasonable to assume that the data collection period will generaly cover several years. This survey also suggests that a patient is likely to be seen on several occasions during a study.

The naive solution is to collect paper records during this period and transcribe them onto the computer when they are complete. This denies any advantages that might come from using a computer system as part of the data collection process. Constructing a model of the investigator at work will help clarify some of the possibilities.

Sibley's CLINFO system[8] uses a simple three stage model.

1. Design study - decide what data are to be collected.
2. Collect data taking care to check its validity.
3. Review and summarise data using graphical and statistical techniques.

Although the second and third stages may overlap this model does not allow for a reassessment of the design. The initial selection of data items is often intuitive and investigators should conduct a pilot study to test the design.

This is not enough. It would be a poor researcher who spent several years collecting data without learning anything from it. Figure 2.1 shows a more complex model which recognises the learning process that occurs once data collection has begun. This model contains a cyclic process of collection and examination of data followed by revision of the study.

The first iteration models a pilot study but these data are now retained within the full sample and subsequent iterations take place whenever the investigator wishes to review the situation. This progressive construction and analysis of the data-base allows the investigator to improve his observational techniques. In addition changes may arise either from the publication of other research or from the desire to start formal collection of data previously noted informally.

These changes can take several forms. They may simply involve including additional observations, or they may require changes or refinement of items already held in the data-base. Finally they may take the form of discontinuing the collection of items that become clearly irrelevant to the study.

For data management software to be of value in these circumstances, it must accommodate changes in an easy and efficient manner. In 1977 Chen[5] described how an evolving view of data can be handled by the entry-relationship model. There can be major problems practical problems in applying such conceptual changes to live data, but fortunately the most likely changes in research data are the easiest to implement. Here again a data dictionary can perform a valuable role. The system can take a new description of the modified data items, compare this with the existing description and reorganise the data accordingly. The addition or deletion of data items can be handled automatically, but refinement of existing data often requires action by the researcher. Since it can identify the cases to which a change applies and draw them to the researcher's attention the computer can be a considerable help in the updating process. If the original data-base includes annotations or references to a paper file this process is further simplified.

3. PATIENT BEHAVIOUR

The investigator is not the only active participant in a study. Unless it is a short in-patient study patients must return to the clinic or hospital for repeat examinations. To study the implications of this the following model (figure 3.1) is proposed.

Clinicians usually see more patients than they are following closely for research purposes. Before entering the sample a patient faces a selection process to determine whether his case is "of interest". Those who are chosen must first be given a surrogate identity within the data-base. This is followed by recording a clinical history and the results of the first examination. The investigator may start treatment and then wait for the situation to develop. Subsequently the patient is re-examined and the new results recorded. This cycle of waiting and re-examination is repeated until enough information has been obtained and the patient leaves the sample.

The waiting period causes some difficulties. At each re-examination the correct surrogate must be found within the data-base. Ideally the intervals between the examinations are consistent for all patients but in practice patients do not make regular visits to clinics. Further problems occur when patients move away or refuse to attend. The data-base must allow for both broken and truncated series of examinations.

3.1 Dynamic Parameters

Many of the observations made during a clinical study are spot checks on continuously varying parameters such as pulse, blood pressure or weight. These measurements are usually repeated after each waiting period. Frequently the interests of the researcher are not focussed on the individual observations but on the trends or changes in the measured parameter. To test hypotheses about trends it will be necessary to compare the profiles of these "dynamic" parameters.

By way of example, consider the following study of a treatment program for hypertension. To examine the response to various drugs a series of blood pressure readings is collected for each patient. The researcher is interested in the way in which blood pressure

falls following the start of treatment and each patient is seen regularly at a Monday morning clinic. But it is only possible to compare readings taken at the same clinic when the patients start treatment at the same time. If Mr Jones starts treatment two weeks after Mr Smith, then Mr Jones' readings must be compared with those of Mr Smith taken a fortnight earlier. It is possible that Mr Jones and Mr Smith will start treatment on different days of the week. Here no two Monday readings are comparable and some form of interpolation will be necessary.

It is common for ad-hoc systems to store the data on a relative time scale, such as blood pressure at first clinic, blood pressure at second clinic, and so on. The discrepancies resulting from varying starting dates are usually ignored. This solution, already unsatisfactory, becomes even more so in the presence of the irregularities and missed examinations discussed above.

In general the reference points for comparing profiles of dynamic parameters are best established when a particular hypothesis is to be tested. Clearly a data-base that permits this must not entangle the events with the observations. In a paper record this is done by recording a date (and time) at the head of the sheet or in the margin. The way in which we speak of time has been studied by Bruce[2] and formal analyses of information by Brunjes[3] and Bubenko[4] recognise the special role played by time.

The most common implementations use a simple time-stamping technique[8,10]. This has the additional advantage that it is not necessary to determine the criteria for an event before any data are recorded. If the researcher wishes to redefine an event he may do so. The event time is re-evaluated for each patient and the results re-analysed in the light of the new definition. Thus enabling different sets of criteria to be tested which may lead to a deeper understanding of the disease processes involved.

An important contribution here is TOD, the Time Oriented Database[6,9] This data-base marks each measurement with a contact number and records a list of contact dates for each patient. In this way it is possible to identify events, critical periods and profiles for each case. An interesting development of TOD is its use by Blum as a test vehicle for a knowledge based expert system in medical research[1].

Dekeyser and Bolour[11] have proposed a method for modelling time within clinical applications. This scheme permits more complex questions relating to the time, duration or sequence of events to be assessed. A good representation of time also permits longitudinal models to be used in data validation[12].

4. CONCLUSION

A data-base system is an essential part of clinical software. Since the relationships between items are important this cannot be based on simple matrix models of data; a more advanced approach, possibly the network or relational model, is required. The inclusion of a data dictionary not only provides documentation but may also support automatic verification of the data.

Collection and validation of data should be viewed as a continuous process interleaved

with data analysis. As the study continues the investigator will wish to modify the content and structure of the data-base. The data-base management software must be sufficiently flexible to allow this to occur.

Cases are generally followed over a significant time period. The time-stamping of observations, determination of event times, and analysis of trends are fundamental to medical research. The data-base must support all these activities. It should be possible to define subsets of the data and event or reference times at the analysis rather than data collection stage.

The TOD database illustrates the structure required for medical research projects. It contains both schema or data dictionary and information about the time at which measurements were taken. Within certain limitations it permits data collection to run in parallel with data analysis.

References.

1. Blum, R.L. Automating the Study of Clinical Hypothesis on a Time-Oriented Database: The RX Project. Proceedings: MEDINFO '80. Lindberg, and Kaihara, editors. I.F.I.P. North Holland, pp. 456-460, 1980.

2. Bruce, B. C. A Model for Temporal References and Its Application in a Question Answering Program. Artificial Intelligence, Vol. 3, pp. 1-25, 1972.

3. Brunjes, S. An Anamnestic Matrix Toward a Medical Language. Computers and Biomedical Research, Vol. 4, pp.571-584, 1971.

4. Bubenko, J. A. The Temporal Dimension in Information Modeling. In Architecture and Models in Data-base Management Systems. Nijssen, G., editor, Amsterdam, North-Holland, 1977.

5. Chen, P. P. S. The Entity-Relationship Model - A Basis for the Enterprise View of Data. Proceedings: AFIPS National Computer Conference, Vol. 46, pp. 77-84, 1977.

6. McShane, D.J. et al., TOD: A Software System for the ARAMIS Data Bank. Computer (I.E.E.E.), Vol. 12, pp. 34-40, November 1979.

7. Palley, N. A., and Groner, G. F. Information Processing Needs and Practices of Clinical Investigators - Survey Results. Proceedings: AFIPS National Computer Conference, Vol. 44, pp. 717-723, 1975.

8. Sibley, W. L. et al., Data Management for Clinical Research. Proceedings: AFIPS National Computer Conference, Vol. 46, pp. 63-68, 1977.

9. Weyl, S. et al., A Modular Self-Describing Clinical Databank System. Computers and Bio-Medical Research, Vol. 8, pp. 279-293, 1975.

10. Woodyard, M., and Hamel, B. A Natural Language Interface to a Clinical Data Base Management System. Computers and Bio-Medical Research, Vol. 14, pp. 41-62, 1981.

11. Dekeyser, L. J., and Bolour, A. On the Modelling of Time in a Clinical Data Base Applications. Proceedings: Fifth Annual Symposium on Computer Applications in Medical Care, I.E.E.E., New York, pp. 311-316, 1981.

12. Veling, S. H. J., and Van't Hof, M. A. Data Quality Control Methods in Longitudinal Studies. In: Kinathropometry II, International Series on Sport Sciences, Ostyn, M., Beunen, G. and Simons, J. editors, University Park Press, 1980/1.

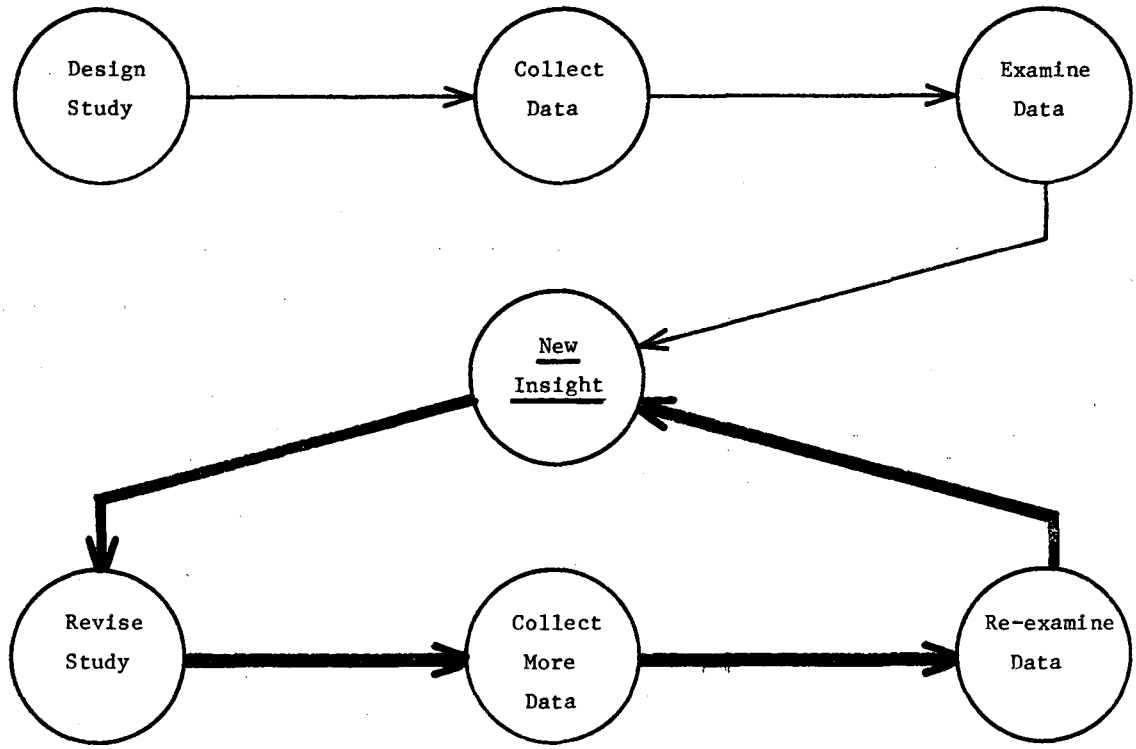


Figure 2.1 Model of Researcher Behaviour.

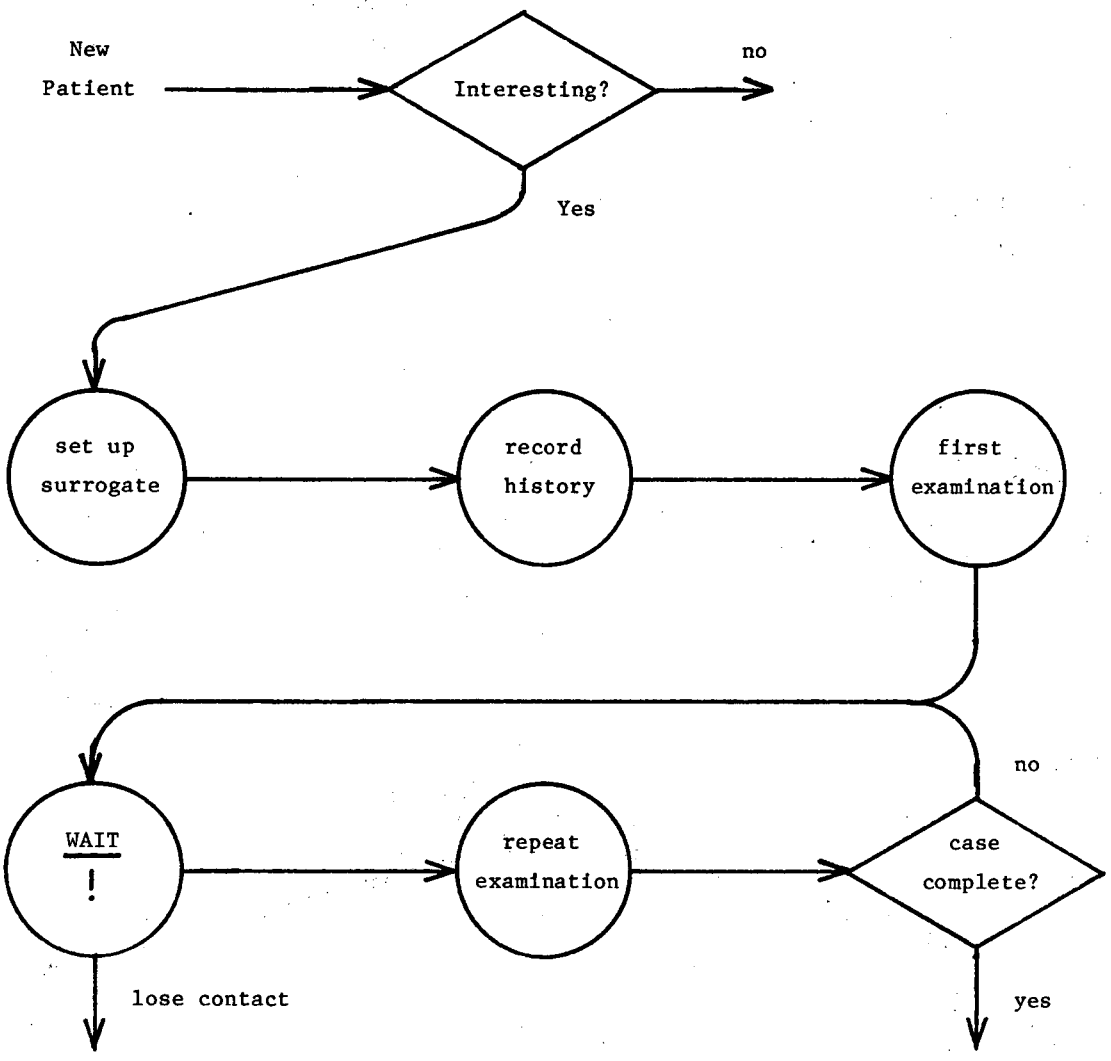


Figure 3.1 Model of Patient Study Interaction.

FEATURES OF A SYSTEM FOR STATISTICAL DATABASES*

Gultekin Ozsoyoglu and Z. Meral Ozsoyoglu

Department of Computer Engineering and Science
Case Western Reserve University
Cleveland, OH 44106

Abstract.

We list desirable features of statistical databases (SDB) and summarize the architecture and major features of a statistical database management system, called the System for Statistical Databases (SSDB). SSDB uses a semantic data model designed to facilitate the data manipulation tasks of statistical database users. It incorporates tools to model operational populations such as experimental, cleaned, interpreted and representative populations, and has the capacity to model summary tables, histograms, matrices, crosstabulations, scatter diagrams and two-dimensional plots. The data model uses the compartmentalization concept to minimize the effects of SDB security enforcement. The SSDB software is partitioned into certified and uncertified modules, and the flow of control within SSDB is designed to allow SDB security checking at execution time. Finally SSDB has a screen-oriented query language to manipulate summary and raw data. The paper concludes with a brief overview of the existing and proposed systems.

I. INTRODUCTION

Database researchers working on the statistical database (SDB) security problem usually define an SDB management system (SDBMS) as "a database management system that provides statistical information to users" where statistical information is defined as simple summary statistics such as SUM, COUNT, MEDIAN, etc. of some data in the SDB. The SDB security problem is then defined as controlling the use of the database in such a way that "statistical" queries are allowed but protected information in the database cannot be inferred from responses to queries.

While the above definition of SDBMS is sufficiently general to investigate the SDB security problem, a more useful definition of SDBMS may be given by observing the statistical usage of data by statisticians and users from other disciplines. These users employ a stored collection of data to (a) execute statistical data analysis procedures that range from simple summary statistics to advanced procedures like discriminant or factor analysis, (b) rearrange and manipulate data either for efficient storage and maintenance of data or for input to procedures in (a), and (c) to edit and tabulate summary data. Thus, in this proposal we define SDBMS as a database management system that provides capabilities (i) to model, store and manipulate data and (ii) to apply statistical data analysis techniques to data in the SDB. Clearly data modeling, storage and manipulation capabilities should be developed in a manner suitable for the operational usage of data by SDB users. We also believe that an SDBMS should provide to users SDB security enforcement capabilities.

With few exceptions, there are presently two approaches used to meet the needs of SDB users. One approach is to use special-purpose SDBMS's for specific applications (such as RAPID [TuHC 79] for census-like applications with very large, nearly static data). The other approach (hereafter called the statistical package approach) is to use a statistical package (such as SAS [SAS 79]) for function (ii), and a file management system plus customized application programs or utilities for function (i). The statistical package approach has few data modeling tools. Data independence and data sharing are usually difficult to achieve. There are incompatibility problems among files (each file usually has different label, format and value conventions [ChaS 81]). There are no powerful data manipulation languages, and SDB security mechanisms are nonexistent. One advantage of the statistical package approach is that

*This research is supported by the National Science Foundation under Grant MCS-8306616.

users are free to choose any statistical package they like. The current SDBMS approach removes most of the above-listed disadvantages of the statistical package approach, with the exception of SDB security mechanisms. However, the majority of current SDBMS systems are limited in their capacity since they are tailored to specific applications. We think that the current SDBMS approach should be enhanced by utilizing advanced data models for SDB's, powerful query languages that manipulate objects of these models, and SDB security mechanisms.

In Section II we discuss desirable features of general-purpose SDBMSs. Section III summarizes design features of the System for Statistical Databases (SSDB). Section IV contains a brief overview of the existing and proposed SDBMSs with respect to the features discussed in Section II.

II. DESIRABLE FEATURES OF AN SDBMS

This section argues that a general-purpose SDBMS should have a semantic data model with the capability to manipulate various data objects commonly used by SDB users, and that various SDB security mechanisms should be enforceable by the SDBMS.

A. The need for a semantic data model

Users of statistical packages who perform statistical data analysis always use conceptualizations of the real world to model the data they deal with. In social science research, for example, a researcher (i.e. the SDB user) may want to "describe social reality" or "to construct a social theory". The researcher makes decisions about variables (i.e. attributes of individuals) at the conceptual level and forms some general ideas concerning the interrelationships and causal effects of variables upon each other. In current practice, the above conceptualizations and data are separated. Certainly a data model will be useful to represent and maintain these conceptualizations, not only for documentation purposes, but also for implementation purposes. Several advantages follow from the data model approach. First, providing the total information contents of the SDB will help users in their exploratory data analysis by making them aware of the richness of data. Second, it will enable users to use simple aggregate query expressions to request information from the SDB because of the explicit semantics of the data [Shos 82]. Third, the data model will help in enforcing integrity, validity and security of the SDB. For example (due to the existence of a unified data model) labeling, formatting, value, and naming incompatibilities will disappear. Finally, the data model, if designed properly, may also serve as a data model of a conventional, ordinary (i.e. "corporate" [Date 81]) database (CDB), which may be desirable for those applications where there are SDB and CDB users.

Usually, during the exploratory data analysis phase, original conceptions of SDB users are modified through an iterative process of alterations and creation of new conceptions (inspired perhaps due to alterations). This iterative process may cause a data model representation of the real world to change or to be extended frequently. Therefore, there may be a need for a time period in which the SDB user experiments with a portion of the representation while having exclusive control of that portion. Once a part of the representation becomes more or less stable, and the user reaches conclusions about his understanding of the real world represented by that part of the data, he may move to "higher levels of analysis" requiring more complex representations. Individuals in the "lower level" form members of units in a "higher level" which is also represented in the data model, and the iterative process is repeated. From this brief discussion of processes in statistical research we conclude that an SDB semantic data model is indeed useful for SDB's.

Due to the differences in data manipulation characteristics, semantic data models for SDB's and CDB's should differ. The special utilization characteristics of SDB's necessitate incorporation of new operational tools into the SDB data models. An example may be the tools to model representative, experimental, interpreted or cleaned subsets of the data. Other objects that SDB users routinely use include summary tables, histograms, crosstabulations, scatter diagrams, two dimensional plots [OzsO 82b], sets, vectors, matrices [Shos 82, McCa 82b] and time series [McCa 82b]. These needs arise not so much from the semantic modeling aspect of data, but from the operational needs of SDB users. Since these abstract objects are manipulated using different operations, it is beneficial to define customized operations for different object types. This argument is in line with recent advanced semantic data model

proposals that contain a rich set of abstract data types and customized operations for each type.

B. The need for new query languages

Once the conceptual models of SDBMS's are implemented using semantic data models with new objects having abstract data types, powerful data manipulation languages operating on these objects are needed. For example, summary tables are used by SDB users to tabulate and compare redundant summaries of raw data. Therefore, SDB data models should naturally represent summary tables, and there should be a database language to define and manipulate summary tables.

C. The need for SDB security mechanisms

The SDB is said to be compromised when, using responses to queries, users deduce protected information in the SDB. Compromise usually occurs when the protected attribute value of some record in the SDB is uniquely identified as the attribute value of a certain individual.

Protecting SDB's from compromise is a difficult task since there are various inference mechanisms that may be employed by users. The security problem of the SDB involves inference and is therefore inherently different (and more difficult to solve) than the security problem of CDB's related to access control.

Some SDB's used in application areas such as political planning, medical research and strategic defense planning contain security-sensitive information. Researchers and practitioners in database and statistical computing areas generally agree that SDB security is an important area. But there seems to be little enthusiasm among statistical software vendors and SDB users [LBLW 81] for introducing costly, complex and restrictive protection mechanisms into SDB's. The main objection of vendors of statistical software packages is that presently there is no demand from their customers for SDB security protection mechanisms However, there have been publicly reported incidents of SDB users illegally deducing protected information [Park 76]. As far as we know, SDB security mechanisms are applied only in census databases (in the form of security checks in 2,3-dimensional summary tables) and in some medical databases [Schl 82]. Moreover, the majority of recently proposed SDB security techniques [Denn 82] are as yet untested in real world SDB applications. We feel that any SDB architecture must be designed to support a variety of security mechanisms available if and when they are needed to enforce security. However, the architecture should also try to minimize any ill effect and cost due to applying security restrictions by encapsulating security-sensitive data in the SDB.

D. Other advantages and potential problems

What other advantages follow from an "enhanced" SDBMS approach that contains a conceptual (semantic data) model, a query language manipulating new objects and a capability to enforce SDB security mechanisms and to execute statistical analysis procedures by maintaining them within the SDBMS?

(a) Compared with users of the statistical package approach, the task of SDBMS users are simpler since they need to become familiar with only one software system. Moreover, using innovative graphical query languages (such as QBE [Zloo 77], Abe [Klug 81] or STBE [OzsO 82a]) or user interfaces (such as GUIDE [WonK 82]) the SDBMS approach may be more user-friendly.

(b) The enhanced SDBMS approach removes potential incompatibilities and the often problematic data transfer between statistical package software and file management or SDBMS software.

(c) Uniformity among operations (syntactically and semantically) can be achieved since one software system stores and manipulates data, and executes statistical procedures.

(d) Users of the enhanced SDBMS approach will use simpler and fewer queries in their statistical analysis since all the tasks of retrieving data from files, preparing data for input and storing data after analysis, can be expressed by queries of only one language.

(e) Having a centralized control of data and a statistical procedures library within the SDBMS allows the utilization of a security technique known as intentional resolution [Mins 76]. If a user would like a multiple regression analysis, the SDB uses the protected raw data without making it available to users, and returns only the results of the analysis to users. This way users are provided with the results of the analysis they request but

prevented from seeing the protected raw data input to the statistical procedure. Another case might be to process the protected raw data and provide users with an intermediate data analysis. An example is to give users correlation coefficient matrices (instead of the protected raw data) for possible input to procedures like factor analysis and canonical correlation.

(f) Centralized control of data by the enhanced SDBMS approach allows execution-time SDB security enforcement.

What are the possible pitfalls of the enhanced SDBMS approach? We envision the following potential problems.

(1) Implementing SDB security enforcement (such as a security kernel or data-tagging [OzsC 82]) may make the SDBMS software less efficient, and the storage requirements may be higher. We think that a careful testbed implementation will answer this question.

(2) Feasibility and efficiency of the implementation of a conceptual (semantic data) model needs to be demonstrated on real life applications.

(3) A more important issue, perhaps, is the acceptability of a conceptual data model to SDB users. A concern has been raised by practitioners in the statistical computing area that the eagerness of SDB users to get on with their research usually results in bypassing the maintainance of simple data definitions such as variable identification [Mark 81]). This then raises the user acceptability of the SDBMS approach that requires a full conceptual data model to be in place from the start*. We think that all the advantages listed above outweigh this inconvenience to users. In the long run, users benefit from having a well-specified data model by gaining a better understanding of the data. Nevertheless, this problem has to be investigated by experimenting with a testbed implementation of the enhanced SDBMS approach.

SYSTEM FOR STATISTICAL DATABASES (SSDB)

This section briefly discusses the features of a general-purpose SDBMS, the System for Statistical Databases (SSDB) [OzsO 82b] designed as a response to the problems listed in Section II. It should be pointed out that SSDB is an ongoing project, and several of its components are currently being researched. However, this section reports only those aspects of the design that are currently concluded.

A. Architecture

SSDB architecture consists of three levels, external level, conceptual level and internal level, in the conventional sense. The external level is concerned with individual SDB user views; the conceptual level defines the community view (i.e. the conceptual view). The internal level is concerned with secure, efficient and effective access and utilization of data. Figure 1 illustrates the three levels. Each user has a data definition and manipulation language to manipulate data objects at the external level. SSDB is designed by modifying a three level CDB architecture level by level thereby making it potentially possible to serve as a CDB to some users.

Conceptual Level:

At the conceptual level, there is a Compartmentalized SDB Conceptual Model and a Security Database. The conceptual model uses the Heterogeneous Operational Data Model (HODM) as the data model and contains compartments of security-sensitive information. Figure 2 represents the conceptual level.

Conceptual Model consists of three layers. Innermost layer, Data Model, is concerned with the representation of the real world in a natural and semantically rich manner. The middle layer provides tools to model representative, cleaned and interpreted subsets of data, and the outermost layer is concerned with easy and convenient representation of aggregate data (i.e. summary tables) and other new objects of the SSDB such as matrices, histograms, two-dimensional plots, etc..

The Security Database contains security-related information for each compartment such as security constraints [ChiO 81], user groups and user knowledge constructs [ChiO 81], query and update history of abstract objects [OzsC 82, ChiO 82, OzsO 81].

External level:

At the external level, there are

* In order to help solve this problem, "stand-alone" views are proposed in [OzsO 82b]. However, users are still required to specify the schema of the stand-alone view.

virtual and stand-alone user views. Virtual views are either derived or exact replicas of a part of the conceptual model. Processing a query about a virtual view involves the usual processes of mapping the query to conceptual model and to internal model, and optimization. Stand-alone views are not completely mappable to the conceptual model. Stand-alone views are introduced to manage the iterative stage (i.e. unstable data representation) of users' exploratory data analysis where data or its parts are experimental and are not shared, and the corresponding model is not stable. Processing a query about a stand-alone view involves mapping the query to the internal level and optimization.

Internal level:

With the exception of matrices and vectors, all objects of HODM are represented as nested relations [OzsO 83] at the internal level of HODM. A nested relation is a relation with set-valued or simple-valued columns.

File organization techniques for nested relations are currently being investigated. Since SSDB is general-purpose and allows updates, existing SDB file organization techniques such as transposed files and cross product files [EggS 80] have to be modified. Another research problem is the maintenance of aggregate values in the presence of updates to the database.

For compartments in the internal model, all data has security-related tagging [OzsC 82]. The tagging is used in verifying correctness of the physically retrieved data during execution time.

B. Data Model

The HODM is designed by modifying the Data Abstraction (DA-) Model [SmiS 77a, SmiS 77b]. The main reason for choosing the DA-model is that operational characteristics of SDB users can be incorporated into the data model as specialized generalization hierarchies of the DA-model, thus making this model a natural choice. It should be pointed out that the query language of SSDB does not utilize the structure (i.e. various hierarchies of HODM). The structure of HODM is used for browsing through the model integrity checking, SDB security enforcement and documentation. Query language of SSDB refers directly to abstract objects for data manipulation, and avoids navigation through the structure of HODM.

DA-Model:

The DA-model uses aggregate and generic objects to name relationships and classes. The real world is modeled as a set of aggregation hierarchies intersecting with a set of generalization hierarchies. In the HODM, each abstract object forms a population about which statistical analysis can be be performed. Within compartments there are additional constraints placed upon generalization hierarchies [ChiO 81]. Innermost layer of HODM uses only aggregation and generalization hierarchies, and is a semantic data model for corporate database (CDB) users as well as for SDB users. Figure 3 contains a university database.

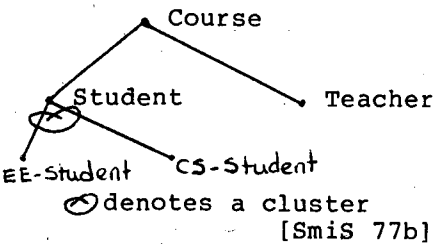


Figure 3. Representation of university database using the DA-Model

Operational Data Model (ODM):

ODM is used for the middle layer of the conceptual model of SSDB, and incorporates hierarchies of various types of populations, called operational populations, that are needed because of the operational characteristics of SDBs. There are four types of operational populations, representative, interpreted, cleaned and experimental populations, each created (perhaps iteratively) from a population of the DA-Model called main population. Each type of population may form a hierarchy. Each type of operational population has a prefix (e.g. RP, IP, CP, EP) to indicate its type.

(a) Representative populations (RP) are user-specified subsets of data. They are created to facilitate the efficiency of the operation, or due to either the inability to process large amounts of data or the sufficiently general nature of the conclusions drawn from representative samples using statistical procedures.

(b) Interpreted populations (IP) are created by classifying individuals

in a given population using different interpretations of individuals' attribute values. For example, "U.S.-made-cars" may be interpreted as "cars with at least 85% U.S.-made parts" or "cars produced by U.S.-companies".

(c) Cleaned populations (CP) are created by eliminating some of the individuals in them by selection tests that involve user requests or processing missing, suppressed and perturbed values and correctness ranges.

(d) Experimental populations (EP) are user-created temporary subsets of other populations. They are used mainly to draw conclusions about phenomena not yet modeled by the database.

Figure 4 contains a representation of a university database using ODM. CP-SECRETARY-PERMITTED contains all those secretaries who are not assigned to security-related jobs (i.e. a cleaned population). IP-EXPERIENCED-ENGINEER-1 and IP-EXPERIENCED-ENGINEER-2 are two different subsets of engineers selected according to different "experience" criteria. Within a generalization hierarchy (or more correctly, subset hierarchy), notations  and  denote upward and downward existence dependency respectively (i.e. existence of a population is defined through the existence of other populations). Similarly, notations  and denote upward and downward existence dependency within an aggregation hierarchy respectively. For example, IP-EXPERIENCED-ENGINEER-1 is existentially dependent to ENGINEER, and RP-ASSIGNMENT is existentially dependent to ENGINEER and RP-PROJECT.

Heterogeneous Operational Data model (HODM):

HODM is used for the outer layer of the conceptual model of SSDB, and incorporates new objects to help users model data according to their data manipulation needs. These objects are summary tables, crosstabulations, matrices, histograms, scatter diagrams and two-dimensional plots. Reference [OzsO 82b] briefly describes operations involving these objects. It should be pointed out that histograms, scatter diagrams and two-dimensional plots are mostly viewed as visual comparison aids utilized by SDB users and thus operations involving them are restrictive.

Among the new objects in HODM, summary tables are very common. In fact, the Table Processing Language System [Uslb 80] produces only summary tables. As an example a summary table scheme is shown graphically below:

CANCER-PATIENTS		OCP	OCP
			SEX
	AGE	COUNT	COUNT

In this example, the summary table name is CANCER-PATIENTS. It has a column category attribute forest consisting of two trees. The first tree has a single node, OCP. The second tree has two nodes OCP and SEX where the root of the tree is OCP. The row category attribute forest consists of a single attribute AGE. In a summary table, each cell is defined by a set of row category attributes (which appear in a root-to-leaf path in a row category attribute tree) and a set of column category attributes.

C. Query Language

All populations at the conceptual model of SSDB are represented as relations. SSDB uses a screen-oriented, two-dimensional query language, called Summary-Table-by-Example (STBE) to manipulate summary table and relations [OzsO 82a]. STBE is a relational-calculus based language. It is similar Zloof's Query-by-Example [Zloo 77] and may be considered as an extension of Aggregates-by-Example [Klug 81].

D. Security Considerations

In [OzsC 82] we have argued that for secure processing, SDBMS should be certified to guarantee that it works correctly. However, since certification is not an easy task, the software for certification should be minimized. The SDBMS of SSDB is grouped into modules and only a very small part of the SDBMS is certified. Execution flow within SDBMS is designed as follows. Population and attribute types (i.e. logical object types) involved in the query are derived by certified modules. A certified security kernel accesses the physical database and retrieves physical objects. After a secure mapping of physical and logical objects (each physical object in a compartment is tagged by its logical name), SDB security enforcement rules are applied. Procedures in statistical procedures library are not certified, but they run isolated from users and other SDBMS modules.

One may think that certification is too big a task for SSDB. However, there is already a successful implementation effort [DowP 77, DowP 79] that applies

almost the same scenario above with the exception of SDB security enforcement. Moreover, a majority of the SDB protection techniques can be implemented with very small code. Therefore we think that the proposed SDBMS execution flow is practical. We should also note that we propose tagging for only those objects that are in compartments.

Since we design SSDB as a general-purpose SDBMS suitable for various applications we allow updates in the SDB. Consequently there should be some secure update handling mechanisms for objects in compartments. In [OzsO 81] we discuss a variety of update handling techniques that enhance security in a single population.

IV. DATA MODELS AND QUERY LANGUAGES OF EXISTING SDBMS'S

To contrast the design characteristics of SSDB with other systems, this section briefly surveys data models and query languages of existing SDBMS'S. Other characteristics of current SDBMS's can be found in [Shos 82].

The discussion below should not be interpreted as a claim that the systems below are not successful. Indeed there are reports [Hamm 81 and others] that they are very successful for use in those applications for which they are targeted. However, we think that for a general-purpose SDBMS, requirements listed in Section II should also be satisfied.

A. Data Models

Traditional data models are used in existing SDBMS implementations and proposals. For example, RAPID [TuHC 79], and the Table Producing Language System [Uslb 80] use the traditional relational network, and hierarchical models respectively. Recently Ikeda and Kobayashi [IkeK 81] reported an SDBMS implementation on top of a relational DBMS.

There are only three semantic data models specifically proposed for SDB's: SAM* (Semantic Association Model) [Su 82], the cluster and cross product hierarchies of SUBJECT [ChaS 81] and the infological framework of RAM [RapS 75, Sund 78]. The first two models are structured, redundant, and similar to semantic networks. SAM* has seven different association (relationship) types and supports sets, matrices, vectors, and time series as complex data types. A proposed implementation of SAM* uses G-(generalized) relations. Cluster and cross product hierarchies of SUBJECT use the concept of summary sets, distinguish between category and summary attributes for semantic modeling and simpler query expressions of summary statistics, and utilize cluster and cross-product abstractions to model both raw data and matrix representation of summary data. From the examples in reference [ChaS 81], cluster abstraction is a combination of generalization abstraction of [SmiS 77b] and attribute association of SAM*. Cross product abstraction is, on the other hand, a new data modeling tool proposed to represent summary data using a n-dimensional matrix data type. The infological framework of RAM uses elementary messages of groups of objects. To represent statistical surveys, a box structure which is always a n-dimensional cube (not an arbitrary structure) is used. Classification of objects according to their data types is not mentioned.

B. Query Languages

Existing query languages of CDB's can be used to produce summary statistics. However, the syntax and semantics for statistical queries are usually not well-defined. Moreover, operations on objects with complex data types (such as matrix, summary table, etc.) do not exist.

Aggregates-by-example (Abe) [Klug 81] is a language proposed to ease the task of formulating statistical queries. Abe is a relational query language similar to Query-by-Example (QBE) [Zloo 77]. It uses the subquery concept to simplify complex statistical query expressions.

Among user interfaces, SUBJECT uses a set of seven menu driven commands (not a query language) that includes browsing capabilities in the cluster and cross product hierarchies to help users locate and retrieve aggregate information. GUIDE [WonK 82] is a graphics-based user interface that contains subject directories, help messages, zooming facilities and partial query formulation features.

C. SDBMS Implementations and Proposals

Presently there are various SDBMS implementations and proposals. These systems include RAM[Sund 78], SUBJECT [ChaS 80], RAPID [TuHC 79], SEEDIS [McCa 82a], Ikeda and Kobayashi's system [IkeK 81], Table Processing Language System [Uslb 80], System S [BecC 78], GENISYS [ManD 81] and others. The common

features of all these systems are:

1) None has any SDB security mechanisms.

2) Except RAM and System S, all of the above systems interface to a statistical package for statistical analysis procedures. RAM has a "macro data processing subsystem". System S maintains and utilizes statistical procedures as functions (about 250) in an interactive environment.

3) Except System S, SUBJECT, and Ikeda and Kobayashi's system, none of the above systems model and support abstract data types. System S uses vectors, matrices, and time-series as concrete objects without a data model. Cross product abstraction of SUBJECT can be considered as an object with a matrix data type, but SUBJECT does not have explicit operations for the cross product abstraction. The system of Ikeda and Kobayashi uses the summary table object, but has very limited operations involving summary tables.

4) Since abstract object types are not supported, none of the above have query languages manipulating abstract objects.

REFERENCES:

[BecC 78] Becker, R.A., Chambers, J.M., "Design and Implementation of the S System for Interactive Data Analysis", Proc., IEEE COMPSAC Conf., Nov 1978.

[ChaS 81] Chan, P., Shoshani, A., "SUBJECT : A Directory Driven System for Organizing and Accessing Large Statistical Databases", Proc., VLDB Conf., 6, 1980.

[ChiO 81] Chin, F.Y. and Ozsoyoglu, G., "Statistical Database Design," ACM TODS, 6, 1, March 1981.

[ChiO 82] Chin, F.Y., Ozsoyoglu, G., "Auditing and Inference Control in Statistical Databases," IEEE TSE, 8, 6, Nov. 1982.

[Date 81] Date, C.J., An Introduction to Database Systems, Vol. I, Third Ed., Addison-Wesley, Reading, Mass., 1981.

[Denn 82] Denning, D.E., Cryptography and Data Security, Addison-Wesley, Reading, Mass., 1982.

[DowP 79] Downs, D., Popek, G.J., "Database Management Systems Security and INGRES," Proc., VLDB Conf., 1979.

[EggS 80] Eggers, S., Shoshani, A., "Efficient Access of Compressed Data," Proc., VLDB Conf., 1980.

[Hamm 81] Hammond, R., "Metadata in the RAPID DBMS," Proc., LBL Workshop on SDBMS, Dec. 1981.

[IkeK 81] Ikeda, H., Kobayashi, Y., "Additional Facilities of a Conventional DBMS to support Interactive Statistical Analysis", Proc., LBL Workshop on SDBMS, Dec. 1981.

[Klug 81] Klug, A., "Abe--A Query Language for Constructing Aggregates-by-Example", Proc., LBL Workshop on SDBMS, Dec. 1981.

[LBLW 81] Panel Session: Data Manipulation Issues of SDBs, Proc., LBL Workshop on SDBMS, Dec. 1981.

[ManD 81] Maness, A.T., Dintelman, S.M., "Design of the Genealogical Information System," Proc., LBL Workshop on SDBMS, Dec. 1981.

[Mark 81] Marks, G.A., "Characteristics and Evolution of the OSIRIS Data Definition Languages," Proc. LBL Workshop on SDBMS, Dec. 1981.

[McCa 82a] McCarthy, J.L., et al., "The SEEDIS Project: A Summary Overview", Lawrence Berkeley Lab., LBL-14083, 1982.

[McCa 82b] McCarthy, J.L., "Metadata Management for Large Statistical Databases", Proc., VLDB Conf., 1982.

[Mins 76] Minsky, N., "Intentional Resolution of Privacy Protection in Database Systems," CACM, 19, 3, March 1976.

[OzsC 82] Ozsoyoglu, G., Chin, F.Y., "Enhancing the Security of Statistical Databases with a Question-Answering System and a Kernel Design", IEEE TSE-8, 3, May 1982.

[OzsO 81] Ozsoyoglu, G., Ozsoyoglu, Z.M., "Update Handling Techniques in Statistical Databases", Proc., LBL Workshop on SDBMS, Dec. 1981.

[OzsO 82a] Ozsoyoglu, Z.M., Ozsoyoglu, G., "STBE--A Database Query Language for Manipulating Summary Data", Report CES-82-2, CWRU, July 1982.

[OzsO 82b] Ozsoyoglu, G., Ozsoyoglu, Z.M., "SSDB-An Architecture for Statistical Databases", Report CES-82-11, CWRU, Oct. 82.

[OzsO 83] Ozsoyoglu, Z.M., Ozsoyoglu, G., "An Extension of Relational Algebra for Summary Tables", Report, CES-83-3, CWRU, March 1983.

[Park 76] Parker, D.B., Crime by Computer, Scribner's, New York, 1976

[SAS 79] SAS Institute, Inc., SAS User's Guide, Raleigh, N.C., 1979.

[Schl 82] Schlorer, J., "Query Based Output Perturbations to Statistical Databases", Technical Report, Univ. of Ulm, Oct. 1982.

[Shos 82] Shoshani, A., "Statistical Databases: Characteristics, Problems and Some Solutions", Proc., VLDB Conf., 1982.

[SmiS 77a] Smith, J.M., Smith, D.C.P., "Database Abstractions: Aggregation", CACM, 20, 6, June 1977.

[SmiS 77b] Smith, J.M., Smith D.C.P., "Database Abstractions:Aggregation and Generalization", ACM TODS, 2, 2, June 1977.

[Su 82] Su, S.Y.W., "SAM*: A Semantic Association Model for Corporate and Scientific/Statistical Databases," Tech. Rep. #8182-6, Database Systems Research and Development Center, Univ. of Florida, Gainesville, Florida, June 1982.

[Sund 78] Sundgren, B., "RAM--A Framework for a Statistical Production System," Tech. Rep., National Central Bureau of Statistics, FACK, Stockholm, Sweden, July 1978.

[TuHC 79] Turner,M., Hammond, R., Cotten,P., "A DBMS for Large Statistical Databases", Proc., VLDB 1979.

[Uslb 80] Bureau of Labor Statistics, "Table Producing Language System", Version 5, Washington, D.C., July 1980.

[WonK 82] Wong, H.K.T., Kuo, I., "GUIDE: Graphical User Interface for Database Exploration," Proc., VLDB Conf., 1982.

[Zloo 77] Zloof, M.M., "Query-by-Example: A Database Language," IBM Systems Journal, 1977.

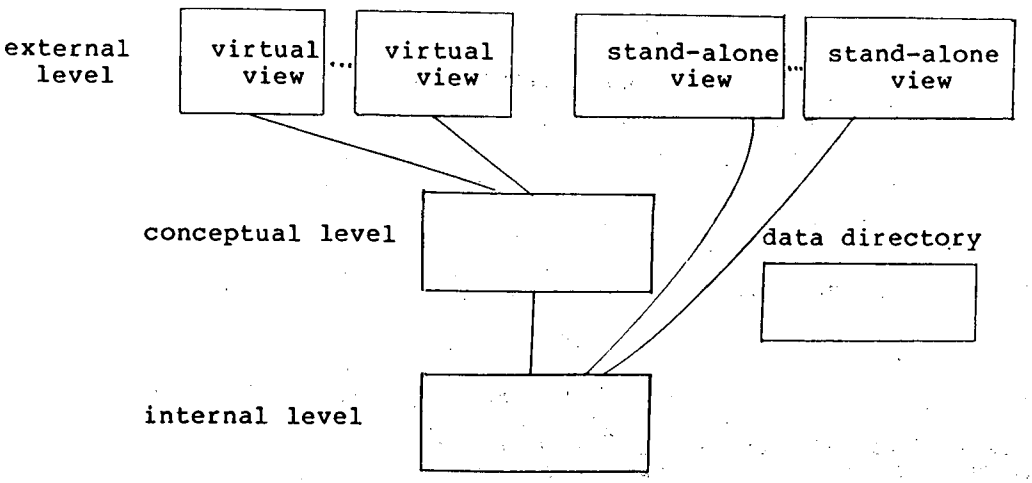


Figure 1. Levels of the SSDB architecture

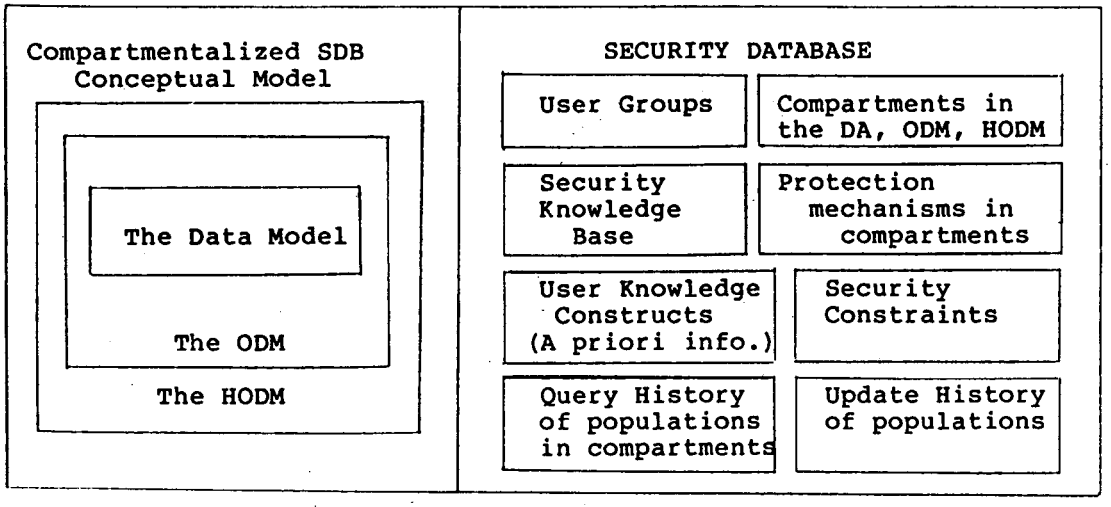


Figure 2. Conceptual Level of SSDB.

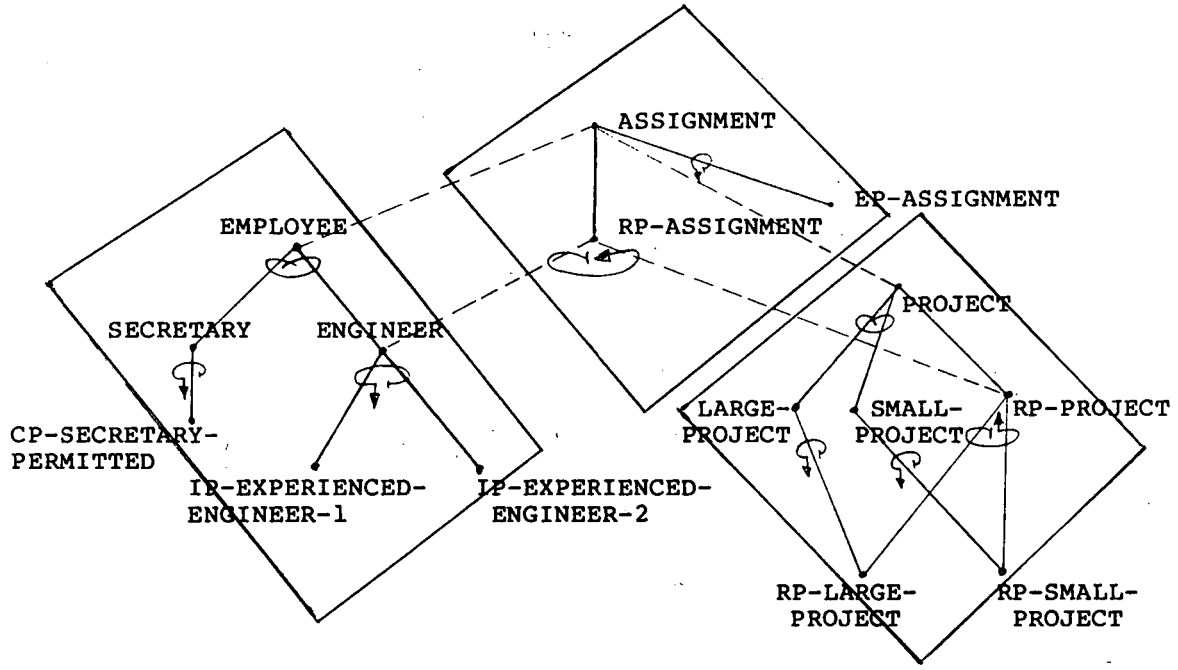


Figure 4. Representation of the database of employees, assignments and projects using ODM

ISSUES RELATED TO MEDICAL STATISTICAL DATABASES*

John M. Long, Joseph R. Brashear
University of Minnesota, 2829 University Avenue S.E., #408
Minneapolis, Minnesota 55414

Abstract

The underlying system for most if not all medical statistical databases is the patient medical record whose automation is still in a development stage. The automation of this underlying system is not standardized and its standardization is the subject of debate. Ethical and legal considerations add to the difficulty in generating statistical databases containing medical data regardless of its use in demographic, epidemiologic, or other research and planning applications. None of these problems appear to have short range solutions. To minimize the impact of these problems on statistical databases in support of planning and research needs, the structure and content of statistical databases using medical data can and should be kept as simple as possible.

1. BACKGROUND

The medical tradition in the United States places a high value on the one-to-one doctor patient relationship. This aspect of medical tradition has had a decided impact on medical record computerization in that the provider is entrusted with the responsibility for a patient's medical data including its dispersion to other users. As there are different types of medical care providers, there are different types of medical record databases. The style and format of each varies with the anticipated use of the data. Content and organization varies with perception of standards and the style of the medical practice.

To date, the approach to formation of medical record databases remains highly individualized. There is no widely adopted standard even though several systems have been proposed or developed. The degree of automation varies widely. The possibility of automation in the forseeable future is often in doubt.

Amidst this apparent disarray, there is evidence that the need for systemization and standardization has been recognized. Medical record database systems have been developed for individual hospitals and medical practices. Systematic approaches to the organization of the medical record have been propounded. Standards for collection, maintenance, and dispersion of medical data are being developed.

2. UNIQUENESS OF THE MEDICAL RECORD

Are medical records so unique that they require specially designed database management systems? At one time the new user was so overawed by computing that he failed to grasp the realistic potential and limitations of computers. This position seems to be reversed in medical applications. The computing professionals can be so overwhelmed by the field of medicine that they fail to realize that they are faced with an essentially but not totally standard problem. Even among those who do understand both the computer's capabilities and the medical record, there are substantive differences of opinion regarding the unique position held by the medical record.

An overwhelming characteristic of the medical record is the enormous amount of data about each individual that is collected over time. A large amount of data is being collected and recorded in a variety of ways by one group of individuals to describe or monitor another group of individuals. Progress in medical science brings about a shift in priorities, needs and emphases. The health care providers cannot anticipate the impact of future developments on their practice. However, they have a clear responsibility to see that the patient is protected and benefits from medical progress. Therefore, there is a tendency to collect every scrap of data possible.

3. SPECIAL DESIGN PROBLEMS

The attempts to computerize medical records coupled with recognition of a need for medical data by users outside of the traditional

*The authors' concerns for these problems are related to the maintenance of and abstracting from the medical records of patients enrolled in the Program on the Surgical Control of the Hyperlipidemias (POSCH) a national multiclinic clinical trial supported by the NHLBI grant #HL15265-10.

provider-patient relationship has led to increasing concern about privacy. One cannot design flat files of data which allow individual items to be picked off for use without consideration of this issue. Either external or internal restrictions, probably both, are necessary.

Tradition has placed the responsibility for protection of medical data with its custodian, the provider. This gives the provider of medical care direct control over the dissemination of a large body of medical data. Even when a patient has given blanket consent for release of data, the provider will still exercise control over what data are actually released. This circumstance is based partly on ethical and legal considerations. However, release of data is also affected by its availability and cost to access.

Users of medical data are forced to go to a source which is fragmented, unsystematic and non-standard. The data source, the provider, is faced with the choice of refusing a request, undertaking a costly abstracting chore, or simply providing the entire record and hoping for the best.

4. IMPACT ON DATABASE DESIGN

There are recognized and legitimate needs for data to be extracted from individual medical records and aggregated for analysis. There are legitimate reasons for the many different approaches to medical record computerization by providers. All of these become points of conflict which tend to cloud or overwhelm basic computing design issues.

For the moment, let us consider the extremes of two dimensions of the question. The many concerns of the provider lead to a comprehensive accumulation of data about each individual over long periods of time. The dynamic nature of medical science results in highly variable or as yet undefined retrieval characteristics. Security measures beyond simple secrecy are required and access should be severely limited.

These design considerations are very different from those for a statistical database containing medical data. The content of the database is defined before data accumulation begins and does not change over the life of an analysis effort. The lifetime of the database can be controlled to be as brief as possible. The amount of data collected on each individual can be minimized to alleviate the need for extreme security measures.

These two different sets of design characteristics probably cannot be reconciled in any expedient manner. The question should become one of whether they ought to be reconciled. Integrated database systems which satisfy diverse needs for access to large databases do exist. However, there is control over the structure and content of the underlying data.

We contend that the degree of control and standardization required cannot be attained in a reasonable time frame for the medical record. The debate over medical record databases will continue. The highly individualistic and entrepreneureal nature of health care delivery will be an impediment to rapid progress. Rather than continue to debate the global issues of medical record database design in every arena, let us solve some computing problems.

If we regard the comprehensive medical record database as a black box source of medical data, we can segregate and simplify many of the issues. The content of the black box, control over access to it, safeguards against inappropriate use, and patient consent to use individual data become the concern of the provider. The structure of the comprehensive database and its degree of automation become a choice for the provider. Willingness to provide data becomes a legal or ethical decision for the provider.

The user of a statistical database containing medical data has fewer design decisions to make. To assure privacy and security, the minimum number of data items which will serve a specific purpose will be requested and the life span of the statistical database should be limited. The concept of informed consent by the patient for a specific use will do away with concern about combining or linking data. The variance in degree of automation within the black box will force a simplified or minimal data structure.

A part of good design practice is recognition of the fact that we cannot impose radical changes on data sources to meet computing requirements. We can make the result of a computer application so desirable that the

data source will want to cooperate and will take steps to reduce the cost of cooperation.

Selected References

1. Jelovsek, et al, Guidelines for User Access to Computerized Medical Records, "Journal of Medical Systems" 2:241, 1978.

2. Long, JM, Brashear, JR, The POSCH Information Management System: Experience with Alternative Approaches. "Journal of Medical Systems" 4:355-366, 1981.

3. Oberst, et al, Computer Applications to Private Office Practice, "Springer-Verling", Fall 1983.

MANAGEMENT AND DISPLAY OF DATA ANALYSIS ENVIRONMENTS FOR LARGE DATA SETS *

Robert A. Burnett, Paula J. Cowley, and James J. Thomas
Pacific Northwest Laboratory
Richland, Washington 99352

ABSTRACT

Data analysis is typically an iterative process in which the choice of the next analysis operation is largely determined by the results of previous operations on the data set. With large data sets, many analysis paths may be explored before meaningful results are obtained. Along each path, the analyst creates a sequence of "data analysis environments," each environment being a frame or "snapshot" of the data set and associated descriptions, conditions, models, and analysis results. The data analysis environment may be changed incrementally through temporary data modifications, subsets, samples, or statistical operations; or, the analyst may wish to restore the conditions of a previous environment as a starting point from which a new analysis path can be generated. Existing analysis systems, however, lack facilities to maintain, save, or restore all of the components required to completely describe or reconstruct a data analysis environment.

This paper describes ongoing research at Pacific Northwest Laboratory (PNL) in data management and display techniques for multiple data analysis environments. Specifically, research is being conducted in four major areas: (1) the development of a model of the data analysis process incorporating the concepts of data analysis environments; (2) the design and use of data modification definitions (differential files) to represent multiple versions of a large data base; (3) the use of data dictionaries/directories to manage, describe, and control multiple data analysis environments; and (4) the application of graphical display and interaction techniques to the examination and selection of data analysis environments. The results of these research efforts will be integrated to provide a new dimension in interactive data analysis.

* Work supported by the U. S. Department of Energy, Contract DE-AC-06-76RLO 1830.

1.0 PROBLEMS AND OPPORTUNITIES IN ANALYZING LARGE DATA SETS

Data analysis is typically an iterative process made up of many operations that collectively refine the data. The analyst uses various data manipulation functions and statistical algorithms to extract useful information. The course of an analysis is often charted "on the fly" via a sequence in which the next operation is dependent upon the results of previous operations. Often the analyst will pursue a particular path and then decide to return to a previous point in the analysis to try a different approach [Denning, et al 1983]. However, it may be impossible or, at best, difficult and time-consuming to restore the analysis to a previous state.

While these observations apply regardless of the size of the data set, the difficulties associated with data analysis are compounded as the size of the data set grows. As the number of data variables increases, a more lengthy analysis may be required to derive a set of meaningful statistics, since there are more opportunities for unforeseen relationships. Operations on data sets with a large number of observations tend to require more time and more computer resources. The end result is a considerable amount of time and effort required to organize and manage the analysis of a large data set.

Data processing activities on a computer are performed within the context of a "computing environment." This environment is defined by the hardware, the operating system, and the application software being used. Software is usually developed and tested with the aid of the operating system and associated program development tools which constitute a "programming environment." Similarly, during an interactive data analysis session, the analyst is working in an ever-changing "analysis environment" of available or currently active data elements, stored results from prior analysis steps, sampling or selection criteria, available operations, and other system or user-supplied specifications. These factors describe the relevant surroundings or "data analysis environment" which has been established by the operations performed on the original data set and by the software environment. We can define a data analysis environment as a combination of the following components:

(1) the state of the currently active data set or subset, including any temporary modifications to the original data set, new records or variables which may have been added, and analysis results which have been stored for later review and possible additional analysis

(2) the status of operational and user interface parameters (data selection or sampling conditions, convergence criteria, default command options, plotting parameters, etc.) which affect the mode of interaction, the interpretation of commands, and the disposition of analysis results

(3) a description or listing of the sequence of analysis operations which produced this environment from a prior known environment

(4) information describing the statistical model being used in the analysis

(5) comments entered by the analyst to describe the environment and document the analysis process.

Although each statistical or data manipulation operation potentially creates a new (though perhaps only slightly different) data analysis environment, several steps may be required before a useful set of results are obtained. Multiple operations are often required to move from one well-identified stage of the analysis to the next stage. Only the analyst knows for sure when a significant new data analysis environment has been reached. The analyst should therefore have the means to name, describe, and save a distinct data analysis environment for later identification and use.

During the exploratory stages of an analysis, many of these environments may be created temporarily for hypothesis testing. Thus many data analysis environments may be created, examined, and then either discarded or set aside for possible later use. The analyst is confronted with the problem of managing and referencing these environments.

Current statistical packages do not provide facilities to maintain complete data analysis environments as defined above. Some statistical analysis packages allow the user to save elements of the current environment; however, there are limitations in the types of information which can be saved and the flexibility and efficiency of saving and restoring the environment. For example, the Minitab statistical package [Ryan, et al 1981] allows the user to save a "worksheet," which consists of a snapshot of the working data set at the time the worksheet was saved. However, there is a limitation in the size of the worksheet that Minitab can handle, and there is no provision for saving only a subset of the

worksheet. In addition, there is no provision for storing descriptive information about how the user arrived at that particular worksheet. Log files of Minitab command sequences can be saved, but the user must keep track of which log files are associated with each worksheet.

The 'S' language and system for interactive data analysis, developed at Bell Laboratories [Becker and Chambers 1981], goes much farther than most statistical analysis packages in allowing the user to define components of an environment. The user can define new data structures as the analysis progresses. The results are returned as data structures that become part of the data base. 'S' also allows the analyst to maintain a journal of the steps taken during the analysis session. The analyst can edit this journal to remove superfluous commands and then apply the edited journal file to the same data set or to a different data set.

Several data and file management systems have some features similar to those described above. DATATRIEVE [Digital Equipment Corporation 1980] allows the user to define "collections" which are usually created by subsetting the data set in some way. BASIS [Battelle Development Corporation 1981] allows the user to save operations as procedures that can be re-executed as desired.

These techniques can be viewed as limited approaches to the definition and storage of data analysis environments. However, these systems lack facilities to automatically maintain a record of the saved environments and their relationships to each other and to the overall analysis. In addition, they require the entire data base snapshot to be physically saved, together with additional information which the software may require to enable a complete restoration of the environment. This is usually not practical for large data sets.

It is even more difficult to return to a previous data analysis environment if that environment has not been explicitly saved. The analyst must somehow try to undo the operations which have resulted in changes to the data set since the prior environment was established. However, it may be difficult or impossible to back out previous analysis steps because the data base updates may be irreversible. The user's only option may be to restore an earlier version of the data set from a backup copy and repeat the previous sequence of operations to reconstruct a prior analysis environment.

Two problems may occur when the user tries to restart a previous analysis sequence: 1) The data set may be so large that it is very time-consuming to re-establish the desired environment; and 2) the analyst may not remember or have access to the exact sequence of steps which were used to construct the former environment.

In practice, the user normally maintains an abstract or high-level view of the manner in which a particular environment was created and the significance of that environment with respect to the overall analysis. Often information of this type is noted briefly on a sheet of paper and subsequently misplaced. If comments about the analysis process are included as part of the environment, they can be extremely useful to the analyst, particularly when returning to a data set after a period of time [Denning, et al 1983]. These analysis descriptions can provide documentation of the rationale for applying a particular operation to the data set.

Another useful item of information is a description of the statistical model being used in the analysis. Some statistical packages allow the analyst to define the model directly to the system. For example, the GLIM (Generalised Linear Interactive Modelling) system [Baker and Nelder 1978] allows the analyst to define a model formula and then determines and performs the appropriate analysis steps without user intervention.

In summary, there is a need for tools to maintain high-level descriptions of data analysis environments for the user's convenience and to provide rapid restoration of previous environments. The discussion above has described many areas where facilities to manage, display, and control data analysis environments can be of significant help in performing an analysis. Many of these problems are not serious for small data sets but become critical in terms of time and difficulty when the data set is large.

2.0 CONCEPTS AND SOFTWARE NEEDS FOR DATA ANALYSIS ENVIRONMENTS

This section describes a new approach to data analysis based upon the management and display of networks of data analysis environments. These concepts have evolved from work in data management and analysis systems in the Analysis of Large Data Sets (ALDS) project at Pacific Northwest Laboratory (PNL) over the past four years.

Data manipulation operations form a large part of interactive data analysis, especially during the early stages of preparing a data set for analysis, and also during the exploratory

phases of the analysis itself. The ALDS Data Editor (ADE) [Thomas, et al 1981], an interactive data editor and subset generator, was developed to provide some of these data manipulation capabilities. Experiences with ADE during its use to manipulate and to subset a number of large data sets at PNL led to the identification of some important characteristics of the data manipulation process for large data sets. One of these characteristics was the need to be able to fully control and verify the current status of the processing environment. This is especially critical for large data sets because the consequences of an incorrectly specified operation or a processing error tend to grow exponentially with increasing size of the data set [Muller 1970]. Full control means the analyst can interrupt any process at any time, determine the status, and decide whether to continue the process or abort it, saving partial results if appropriate.

A second important characteristic of data manipulation is the significant number of conditions or environmental parameters that are often attached to a data manipulation operation or sequence of operations. For example, subsets of a data set are frequently defined by specification of a logical (Boolean) condition for case selection, by a specification for random sampling, or both types of specifications. Cases and variables for inclusion in the subset may also be explicitly specified. It was found to be important for the analyst to be able to specify and verify each of these conditions individually via clause-structured commands. It is also important for the analyst to be able to quickly and freely move from one subset to a previously defined subset or to move back to the full data set and have the associated conditions or environmental parameters automatically carried along. From these and other characteristics of the data manipulation process, the concept of temporary data manipulation environments was conceived. The broader concept of data analysis environments has been a natural outgrowth of these ideas.

Another direct result of the development of and experimentation with ADE was the concept of "virtual subsets." It is usually impractical and often prohibitive to physically store subsets of a large data set if the subset is a significant fraction of the entire data set or if many subsets must be simultaneously maintained. A virtual subset is a definition of a subset; this definition may be in terms of pointers to the included cases and variables or in terms of a description of the selection conditions that define the subset. In ADE, the virtual subset is stored in lieu of physically replicating the actual data values included in the subset. A physical subset is generated only upon explicit request by the user. The concept of virtual subsets led to the idea of storing data modification descriptions and virtual subset definitions in differential files to represent temporary versions of a large data set.

The concepts of dynamic data manipulation environments, virtual subsets, process control and verification, and user specification of environmental conditions via clause-structured commands were embodied in an interaction model for manipulation of large data sets [Thomas 1982]. This model was developed to formally decompose and describe the interaction sequences between the analyst and the system during interactive exploration and data manipulation, and to provide a framework for the evaluation of the impact of large data sets on the design of interactive data analysis software.

Current research at PNL is addressing several problems in the management and display of multiple data analysis environments for large data sets. We are developing and evaluating techniques to provide the user with control and flexibility to examine existing environments, define new environments, establish simultaneous parallel analysis activities in multiple environments, and easily move the "interaction window" from one environment to another. The data manipulation/interaction model is being extended and refined to more fully model the data analysis process as a network of interrelated data analysis environments. The model will serve as a frame of reference for evaluating the effectiveness of the methodology and for comparing alternative techniques.

To accomplish the above objectives, we are conducting and integrating research efforts in three major areas:

(1) the design and use of multiple differential files to store variations of a master data set in the form of data modification definitions and subset definitions, each differential file representing a different "virtual data set"

(2) the development of a data dictionary/directory system to maintain definitions of data analysis environments. These definition would include logical (user-level) comments, descriptions of associated statistical models, and access pointers to a) differential files containing definitions of the physical state of the virtual data

set associated with each environment, and b) journal files containing the sequence of analysis operations which produced each environment

(3) the development and evaluation of graphic-based display and interaction techniques that would provide the analyst with a) a graphical representation of the currently active analysis environments and their interrelationships, and b) a convenient means of graphically selecting, monitoring, and controlling multiple data analysis environments.

The following sections describe each research area in greater detail.

2.1 DATA MODIFICATION DESCRIPTIONS

Frequently during a data analysis session, the analyst wishes to temporarily modify one or a few data values and repeat a series of analysis steps on the modified data set. For example, it may be necessary to remove one or more outliers or supply estimated values for missing data items. In fact, the analyst may want to generate several independent versions of the data set, each containing a different set of data modifications based on different criteria. Each version of the data set would represent part of a unique data analysis environment. Statistical procedures could then be applied to each of the resultant data sets, with the option of returning to one or more of the modified data sets and performing additional analyses, redefining the procedures, or augmenting the modifications.

It would not be desirable to directly store the temporarily modified values in the master data base, even if the original values were saved and could be restored later. If this were done, other concurrent users would have to be temporarily denied access to the updated portion of the data set. An alternate approach would be to generate a local physical copy of the data base (or a subset thereof) for each set of temporary modifications. With large data sets, however, it is generally impractical or impossible to do this, especially if many subsets or modified versions of the data set must be maintained.

For situations in which the modifications apply to a small fraction of the entire data set, a better approach would be to store updated records and/or concise descriptions of the individual modifications separately from the data base in a modification file or "differential file" [Severance and Lohman 1976]. During a data base access, the appropriate updated values from the differential file would be substituted for the corresponding original values stored in the main data base. This would in effect allow multiple users to maintain local modifiable copies of a data set without requiring redundant data storage. Interference among temporary updates by different users is also eliminated.

The concept of using differential files to store updates to a data base is not new [Severance and Lohman 1976]. Historically, there have been two major uses of such files. One use has been to provide an effective way to implement backup and recovery in a data base environment [Aghili and Severance 1982; Verhofstad 1978; Batory and Gotlieb 1982]. The second application has been to save updates to a data base until all the updates can be applied at once in a batch mode (e.g., [Battelle Development Corporation 1981]).

We are extending the differential file concept to the implementation of multiple versions of a large data set for multiple concurrent data analysis environments.

The design of a differential file for data analysis applications involves selection from among several different logical representations. The most straightforward technique for storing a data modification is to simply store each updated record in the differential file. This approach is often combined with a multiple hashing scheme, called a Bloom filter [Gremillion 1982], to help determine whether the most recent version of a record is in the differential file or in the main data base, thus attempting to avoid an exhaustive search of the differential file for each data base request.

Another method of storing a data modification is to store a description of the modified record or variable rather than the complete set of values contained in the record or variable. For example, if only one field in a record were updated, one could simply store the record identifier, the field identifier, and the updated value. The most concise form of data modification description is a symbolic vector, in which an entire variable is defined by a data transformation in the form of a stored set of computational rules (e.g., an equation) involving other variables in the data base. The concept of storing descriptions of data modifications is similar to the concept of virtual subsets (data base subset descriptions) as implemented in the ALDS Data Editor [Thomas, et al 1981]. A modified data set or subset, as represented by data modification descriptions stored in a differential file, could be defined as a "virtual data set."

In the course of research at PNL in the above issues, several specific questions are being addressed: What types of data and file structures are most effective for storing data modification descriptions for data analysis environments? How should a differential file be organized to facilitate efficient access to the virtual data set? How should permanent updates to the main data base be handled to avoid invalidating existing virtual data sets as stored in differential files?

2.2 DATA DICTIONARIES/DIRECTORIES FOR DATA ANALYSIS ENVIRONMENTS

Traditionally, data dictionaries have been used to store meta-data describing the various data bases implemented on a data base management system (DBMS), as an aid to the data base administrator [Curtice 1981; Martin 1977; Date 1982], and more specifically, as an information resource for corporate data bases [Plagman and Altshuler 1972]. Data dictionaries may be combined with a data directory to form an integrated Data Dictionary/Directory (DD/D) which additionally provides information on the location and structure of data base components. This information is needed by the DBMS to access the required data. Thus a DD/D can be used to control access, to insure integrity, and to enforce security in a data base system.

The data dictionary/directory concept has been used frequently in general scientific and business data base management systems, but has seldom been used in statistical data analysis systems. However, such a capability is required for the development of an interactive data analysis system which allows the analyst to save multiple data analysis environments, examine the set of environments which are relevant to the current analysis, and arbitrarily select and restore a previous environment. Such a system needs a mechanism to organize and manage information describing each of the data analysis environments and their relationships to one another.

We are developing a two-level Data Dictionary/Directory System (DD/DS) to manage multiple data analysis environments. The system includes a top-level DD/D to describe the original data base and the entire collection of data analysis environments (modified data sets and subsets, various parameters and options, models, analysis results, summary statistics, and comments) that have been derived from the original data base. The top-level or master DD/D contains pointers to the locations of differential files containing definitions of subsets and modifications; these files comprise the second level of the DD/DS. Each modification description file in turn contains a description of a modified data base state in terms of differences from the original or main data base. The master DD/D also contains high-level user-supplied descriptions of each environment and pointers to journal files containing the command sequences (operations) which produced a given environment from a previously referenced environment. In summary, the DD/DS serves as an information resource and data base access control mechanism for an entire analysis of a large data set.

The DD/D must be capable of storing textual comments supplied by the analyst to further describe the environment. Other factors such as information required to generate displays of data analysis environments (See Section 2.3) needs to be included in the Data Directory/Dictionary System. Research is being conducted to determine the best configuration, content, and organization of data dictionaries/directories to facilitate the management of data analysis environments.

Development of a data dictionary/directory system to organize and manage multiple data analysis environments is an outgrowth of ongoing work at PNL in the development and evaluation of data dictionaries/directories for large self-describing transposed data bases [Burnett and Thomas 1982]. A preliminary data dictionary/directory system has been implemented and tested. The system is interfaced to the ALDS data management software and provides access control to data bases stored in multiple Self-Describing Binary (SDB) data files [Burnett 1981; Burnett and Thomas 1982].

2.3 GRAPHICAL INTERFACES TO DATA ANALYSIS ENVIRONMENTS

During the interactive, exploratory phases of data analysis, many analysis environments may be created. To be useful, each of these environments must be easily distinguishable and identifiable. Recently the need has been expressed for "automated cartography of exploration" in data analysis [Tukey 1982] (e.g., a roadmap showing where the analyst is and where he has been during the analysis process). We are developing and evaluating alternative methods of graphically representing information that describes a collection of related data analysis environments. The representation techniques must allow the analyst to easily visualize the relationships among multiple environments, to identify and

select a specific environment, to obtain more detailed information about an environment, and to verify the manner in which a particular environment was created.

Some research has been done in the area of graphical interfaces to data base systems. For example, a system called GUIDE [Wong and Kuo 1982], developed at Lawrence Berkeley Laboratory, uses a graphical network representation of data elements and their relationships as a means of "guiding" the user in the formulation of a data base query. The higher-level relationships among a set of data analysis environments could also be represented using similar graphical techniques. A network graph can be used to depict a starting environment (often the original data base) and the environments that have subsequently been defined by the analyst. Each node of the graph represents an environment. The directed paths between nodes indicate the ancestor-descendant relationships among the various environments. Figure 1 illustrates one way in which a network of data analysis environments could be presented.

Network structures, as presented to the analyst on a display device, cannot show a complete description of an individual analysis environment. The user must therefore be able to select an environment and request more detailed information on that environment. The detailed information could be represented graphically, as text, or as a combination of text and graphics.

Color and geometry can be used to convey information about data analysis environments. Environments that arise from such operations as subsetting and transformations could be represented using different colors and shapes to discriminate among different classes of environments and operations.

Several techniques have been developed in the areas of word processing and editing [Meyrowitz and Van Dam 1982; Lerner 1982] that can be applied to graphical representations of data analysis environments. Both the Xerox Star workstation [Xerox Corporation 1982] and Smalltalk [Ingalls 1978] use a screen that is capable of displaying a full page of a document plus a large menu area. The Star presents "graphical icons" that resemble the entity to which the user is referring. The user performs a task by using a "mouse" to move the cursor to the appropriate icon on the screen. For example, to save a file, the user moves the cursor to the file folder icon. To dispose of a file, the user moves the cursor to the trash can icon.

The Apollo Domain system [Apollo Computer Inc. 1982] is an example of a system which gives the user the ability to create "windows" on the display screen, move these windows around, and change their sizes. The display screen can be compared to a desk; the windows then become documents on the desk and the windows can overlap like pieces of paper on the desk. Multiple windows can be active at one time. The user can make the window of interest more prominent and still be able to see the other windows.

There are disadvantages to using a conventional single-screen alphanumeric terminal to display data analysis environments. When the analyst brings up the graphical display of environments, the screen showing the latest data analysis activity is lost. Rather than using a single conventional terminal, some of the multiple windowing techniques described above can be applied, or more than one screen can be used. If two or more screens are used, one screen could depict the latest data analysis activity while another is used to display information on the environments. A third device could be used for data plots. A sample three-screen configuration, shown in Figure 2, consists of an alphanumeric control terminal used by the analyst to interact with the system, a graphics device to display the various data analysis environments, and a high-resolution display device on which graphs of data sets are displayed. A split screen, such as the one that Star uses, could be used instead of two screens to display the latest activity and the environments at the same time. Moving the windows around, as Apollo does, allows the analyst to concentrate his effort on the area of primary concern - either the analysis activity or the environment - and to move easily from one to the other.

The discussion above describes some ways in which data analysis environments can be presented to the user. Research is being performed to determine the best display techniques for graphically representing data analysis environments and allowing the analyst to interactively manage and control the data analysis process. Topics being investigated include: What information is required to convey the essential characteristics of a data analysis environment to the analyst? How should this information be presented? How can window-based systems and conventional workstation configurations be used most effectively?

3.0 SUMMARY

This paper has described a simple model of the data analysis process as a treelike structure of data analysis environments. The leaves of the tree represent different data analysis environments defined during the course of the analysis and the branches represent different analysis paths. The need for techniques to control, manage, and display these environments has led to research in the areas of (1) the storage of data modification descriptions in differential files, (2) the use of data dictionaries/directories to manage, describe, and control multiple data analysis environments, and (3) the application of graphical display and interaction techniques to the examination and selection of data analysis environments.

4.0 REFERENCES

Aghili, H., and D. G. Severance. 1982. "A Practical Guide to the Design of Differential Files for Recovery of On-Line Databases." ACM Transactions on Database Systems, Vol.7 No.4, pp. 540-565.

Apollo Computer, Inc. 1982. Apollo System User's Guide. Release 4.0. Chelmsford, MA.

Baker, R. J., and J. A. Nelder. 1978. The GLIM System Manual. Rothamsted Experimental Station, Harpenden, Herts, England.

Batory, D. S., and C. C. Gotlieb. 1982. "A Unifying Model of Physical Databases." ACM Transactions on Database Systems, Vol. 7 No. 4, pp. 509-539.

Battelle Development Corporation. 1981. BASIS User's Guide, Columbus, Ohio.

Becker, R. A., and J. M. Chambers. 1981. 'S' - A Language and System for Data Analysis. Bell Laboratories, Murray Hill, NJ.

Burnett, R. A. 1981. "A Self-Describing Data File Structure for Large Data Sets." In Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface, pp. 359-362. Springer-Verlag, New York, NY.

Burnett, R. A., and J. J. Thomas. 1982. "Data Management Support for Statistical Data Editing and Subset Selection." In Proceedings of the First LBL Workshop on Statistical Database Management, pp. 88-102. Lawrence Berkeley Laboratory, Berkeley, CA.

Curtice, R. M. 1981. "Data Dictionaries: An Assessment of Current Practice and Problems." In Proceedings of the Seventh International Conference on Very Large Data Bases, Cannes, France.

Date, C. J. 1982. An Introduction to Database Systems. 3rd ed., Addison-Wesley, Reading, MA.

Denning, D., W. Nicholson, G. Sande, and A. Shoshani. 1983. National Research Council Panel Report on Statistical Database Management, Washington, D.C.

Digital Equipment Corporation. 1980. Datatrieve-11 V2.0 User's Guide. Maynard, MA.

Gremillion, L. L. 1982. "Designing a Bloom Filter for Differential File Access." Communications of the ACM, Vol. 25 No. 9, pp. 600-604.

Ingalls, D. H. H. 1978. "The Smalltalk-76 Programming System: Design and Implementation." In Proceedings of the Principles of Programming Languages Symposium.

Lerner, E. J. (ed.) 1982. "Programming for Nonprogrammers." IEEE Spectrum, Vol. 19 No. 8, pp. 34-38.

Martin, J. 1977. Computer Data-base Organization. Prentice-Hall, 1977.

Meyrowitz, N., and A. Van Dam. 1982. "Interactive Editing Systems: Part I and II." ACM Computing Surveys, Vol. 14 No. 3, pp. 321-415.

Muller, M. E. 1970. "Computers as an Instrument for Data Analysis." Technometrics, Vol. 12 No. 2, pp. 259-293.

Plagman, B. K., and G. P. Altshuler. 1972. "A Data Dictionary/Directory System within the Context of an Integrated Corporate Data Base." In AFIPS Conference Proceedings: Fall Joint Computer Conference, Vol. 41, Part II, pp. 1133-1140. AFIPS Press, Montvale, NJ.

Ryan, T. A., B. L. Joiner, and B. F. Ryan. 1981. MINITAB Reference Manual. Duxbury Press, Boston, MA.

Severance, D. G., and G. M. Lohman. 1976. "Differential Files: Their Application to the Maintenance of Large Databases." ACM Transactions on Database Systems, Vol. 1 No. 3, pp. 256-267.

Thomas, J. J., R. A. Burnett, and J. R. Lewis. 1981. "Data Editing On Large Data Sets." In Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface, pp. 252-258. Springer-Verlag, New York, NY.

Thomas, J. J. 1982. "A User Interaction Model for Manipulation of Large Data Sets," In Computer Science and Statistics: Proceedings of the 14th Symposium on the Interface, Troy, NY.

Tukey, J. W. 1982. "Another Look at the Future." In Computer Science and Statistics: Proceedings of the 14th Symposium on the Interface, Troy, NY.

Verhofstad, J. S. M. 1978. "Recovery Techniques for Database Systems." ACM Computing Surveys, Vol. 10 No. 2, pp. 167-195.

Wong, H. K. T., and I. Kuo. 1982. "GUIDE: Graphical User Interface for Database Exploration." In Proceedings of the Eighth International Conference on Very Large Data Bases, Mexico City, Mexico, pp. 22-32.

Xerox Corporation. 1982. 8010 Star Information System Reference Guide. Dallas, TX.

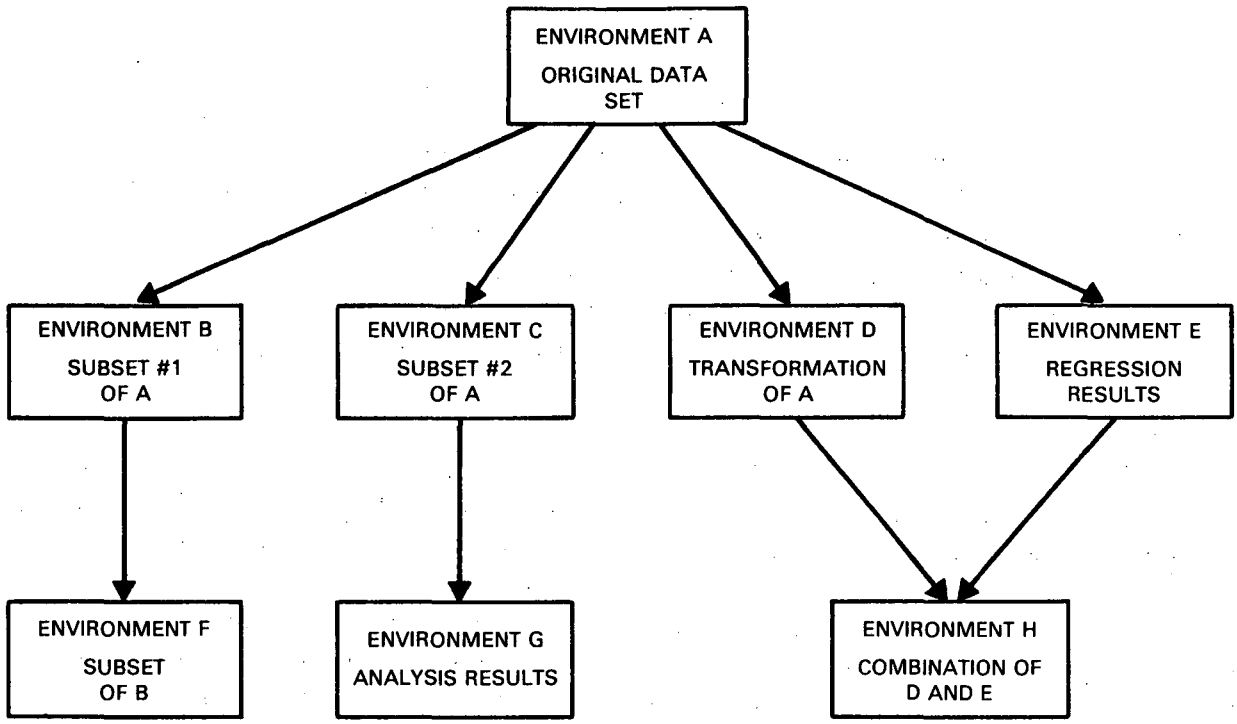


Figure 1. A Graphical Representation of Data Analysis Environments

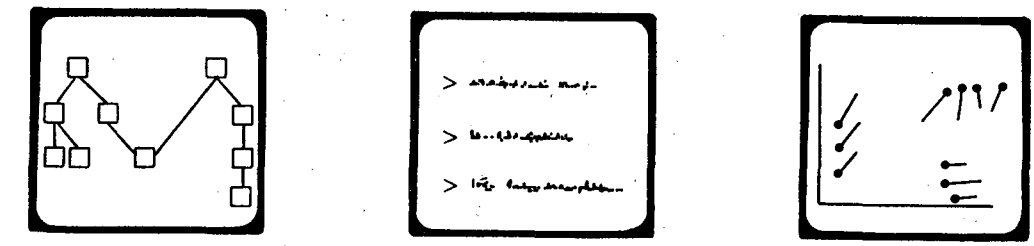


Figure 2. A Three-screen Configuration for Data Analysis

MODEL FOR A CLINICAL RESEARCH DATABASE

Anne Ipsen Goldman, Ph.D.
Associate Professor, Biometry Division, University of Minnesota, Minneapolis, MN 55455

Abstract

A model for the organization of a clinical research database is described. The model is based mainly on the Minnesota Bone Marrow Transplant and the Leukemia-Lymphoma Databases which are characterized by patient data with non-rectangular format, prospectively collected as patients are followed. The SIR data management system is used because of its flexibility and the ease with which statistical analysis files can be created. A main focus of the file organization is an emphasis on events rather than lab values as endpoints. Records for patients, event labels, and protocol are separate case types and linked together during retrieval.

1. INTRODUCTION

Researchers in clinical medicine are finding that a professionally designed and managed database can be a crucial component in the effective conduct of their research, particularly of chronic disease. These databases have a complex structure of data collected from many sources as patients enter the project over time and are followed during the course of their disease. The principal purpose is to provide data for statistical analysis of research concerning patient care. Patient care may also be aided indirectly by providing information for clinical management, for scheduling of clinic visits, and for accounting purposes; but those activities are ancillary and not the raison d'être of the database.

The organization of the database and the administrative structure that supports it depends on the type of clinical research being conducted. The main types of research organization are: cooperative clinical trial, cooperative study group, comprehensive cancer center, research group, registry, single instution-single trial. The associated databases differ mainly in complexity because of the number of clinical centers involved, the types of patients enrolled, and the amount and sources of the data to be collected. It is worthwhile to give some examples and characterize some of the differences.

Examples of the largest Cooperative Clinical Trials are the recently concluded cardiovascular trials (MRFIT [1] and HDFP [2]). These had 10 to 20,000 participants, followed at over 20 clinics for a period of 5 to 7 years, but all participants were enrolled in one protocol. The data were centrally managed and analyzed by a coordinating center. Although the database for such a trial may contain a vast amount of complex data, the primary purpose may be the estimation and testing of treatment group differences for a single parameter. In addition, however, ancillary studies may be added to the main protocol and many analyses of subgroups carried out.

Cooperative Study Group. The Veterans Administration has a long history and a well deserved reputation for conducting excellent cooperative studies [3]. They are also common in cancer research (e.g., the groups known as ECOG, CCSG, CALGB). These have many cooperating clinical centers, each submitting data for a few patients (typically less than 50), enrolled in one of a changing variety of randomized trials for different cancer diagnoses. Each trial may have only 100-200 patients followed for 1 to 3 years. Again, a central statistical center manages and analyzes the data.

This work was partially supported by the National Cancer Institute Grant P01-CA21737 and by the Coleman Leukemia Research Fund.

The National Cancer Institute has designated and supports Comprehensive Cancer Centers. They are similar to the previous but are located at a single institution. There are still many protocols, but not all are randomized trials; the patients have different types of cancer, but all are treated at one large referral hospital. The statistical activities are carried out by an "Epi-Stat" unit which may create one or more statistical databases.

Research Groups exist at single institutions, and specialize in a single disease (e.g., Leukemia, Diabetes) or treatment mode (radiation therapy, bone marrow transplantation). The investigators of such a team may represent a broad spectrum of disciplines and research interests such as chromosome abnormalities, graft rejection, infections, immune function, chemotherapy, etc. If their data management and statistical needs are extensive, they may have a statistical support unit which creates and uses one or more databases for analysis.

Data from patients with a particular disease may be placed on a Registry. Examples are the International Bone Marrow Registry and the SEER Cancer Reigstry [4, 5, 6]. These databases contain all the patients with a particular disease (e.g., breast cancer) or group of diseases (e.g., any cancer), from one or more institutions or geographical area. The data collected includes demographics and details of diagnosis but information on treatment and follow-up may be lacking. The usual purpose of the database may be to estimate incidence of disease and factors associated with differences. Only well managed registries can provide adequate data for such purposes. In keeping with this purpose the number of variables may be more limited and the database organization simpler than for the previous types.

When a single trial is conducted at a single institution, a complex database may not be needed. The quality of the research is, however, improved if the data are prospectively collected, especially if a number of patients are to be followed over a period of time or the patient records and the protocol are complex.

Although the different types of clinical research described above vary in the types and complexity of their databases, they have many features in common. The data deal with individual patients rather than aggregates. The records within a case are arranged in a hierarchy or tree-structure which may be cross-linked in a network. The data are ideally collected prospectively, hence the database continuously changes over time as new patients are enrolled and current patients followed. Because statistical analysis requires a static database, subsets of the data are periodically retrieved from the parent database to create statistical databases for analysis of individual research projects. This paper describes a model for the administration and organization of clinical research databases. The model has evolved over time and is particularly suited for working with a research group as characterized above. It is primarily based on two University of Minnesota databases: the Bone Marrow Transplant Database and the Leukemia-Lymphoma Database. There has also been experience with databases from several other projects starting with the VA-NHLBI Mild Hypertension Study [7] (a cooperative trial 1974-76), and two still-ongoing projects in Testis Cancer [7] and Breast Cancer.

2. THE PATIENT DATA

The Bone Marrow Transplant database currently has 350 patients registered with 90-120 to be added each year. The Leukemia-Lymphoma database has 1100 patients with 120 new cases expected yearly. Patients are registered on the database on admission. Demographic data, details of the diagnosis, history and past treatment are part of the baseline information collected. Results of special laboratory studies, such as hematology, genetic match for transplantation, cell markers, pathology reports, and chromosome studies, are also collected

according to the needs of special studies. During hospitalization and subsequent followup at clinic visits, details on chemo-therapy and other treatment, occurrences and dates of response, side-effects, infections, recurrence of disease, and death are recorded as they happen. The patient may be entered on one or more research protocols, some of which involve randomization to one of several study arms. The details are recorded on a special protocol record which can be protected with special passwords to mask double-blind randomization codes. Throughout the followup period, any special or repeated laboratory or other data specified by protocols are also recorded.

Whenever possible, the data are collected and coded soon after the event happens. Although the hospital chart is used as a guide, it is seldom explicit nor complete enough for research purposes. Additional data and error corrections are extracted from conferences with attending physicians and nurses while the events are still fresh in their minds.

3. DATABASE STRUCTURE AND MANAGEMENT

There are two features of the data which necessitate the use of a database structure rather than an ordinary computer file for storing the data: the non-rectangular format of the file schema and the prospective nature of the data collection process. The schema is case oriented, where the primary case type consists of the data for a patient, organized as an hierarchy of records collected in a tree structure. There are many different record types corresponding to registration, lab., pathology, chromosome, treatment, event records, etc. Since many records of the same type can best be described with a non-rectangular schema. The data are prospectively collected and continuously updated and upgraded as the patient is followed. It is therefore necessary to be able to add data and new patients in a routine manner. Neither of these activities are within the scope of a simple data file. Some of the record types are very long, especially the complex laboratory reports, containing several sort keys to order the reports by date, sample number, subsample, procedure, etc. Some record types are very brief, containing only a date of onset, an event code, and possibly a date of resolution.

Creation and maintenance of the database is carried out using the SIR (Scientific Information Retrieval) system. This system has several features which are extremely important to our type of databases: (1) ease of scheme modification, (2) analysis software interface, (3) archiving of files, and (4) networking of case types. The creative use of these capabilities is the hallmark of our databases and has made possible cost and efficiencies as well as extensive quality control of the data.

a. Ease of schema modification. In a long-term project involving many researchers and many protocols, it is inevitable that frequent changes have to be made in the database structure. The Bone Marrow Transplant Database started out with very limited goals of recording a little information about the 50 or so patients a year being transplanted. By starting small, we quickly had a working, usable system. It has seen major growth and several extensive overhauls over the last three years as the usefulness of the database became realized. Record types have been added, others removed; labels for new codes are added as needed, and the whole file has been restructured several times.

b. Analysis software interface. Many reports are produced directly by SIR, especially for quality assurance, data management, patient summaries, and general administrative purposes. SIR is, however, not designed for statistical analysis but does have the capability of easily retrieving cases and records according to specifications of the user. These retrieval files can be stored as system files directly readable by SPSS, BMDP

or SAS statistical packages or even IMSL and other FORTRAN language programs. Variable names and labels from the SIR file automatically become part of these retrieval files. As pointed out at the beginning of the paper the main purpose of the database is statistical analysis of research projects. For preliminary examination of parts of the database, a temporary retrieval file is created and analyzed. Final analysis of a protocol is usually a process which extends over a period of time, while the database is continuously changing. Analysis must be performed on an unchanging, stable file. Consequently, at the close of a protcol or other research project, a separate file is created containing only those patients and variables specified in the protocol for analysis. After the report is finished, that file and some of the special procedures which were used for its analysis are stored on a magnetic tape. This file then provides permanent documentation for the protocol report and is available if a manusript needs to be revised, sometimes many months after the file was created. The complex main database is thus the parent of a series of static statistical databases used for analysis.

c. Archiving of files. Another feature of SIR is that the whole database, including its associated procedure file can be readily archived, stored on tape, and brought back to disk. For smaller or less active databases, such as the Testis Cancer Database, there is no point in paying for disk storage of the database during periods when it will not be accessed. An automatic archiving procedure has been designed for use by an oncology nurse with only on-the-job computer training. She brings up the file when needed for analysis or data input, and the project assistant periodically archives any changed files and clears the disk. Similarly, the cost of keeping a large database constantly on-line can be considerable. If only part of the file is active during a work week, it is possible to keep such a subfile on the disk, while the rest is archived until needed. The cost effectiveness or archiving depends not only on the structure and usage of the database, but also on the charging algorithm of the computer center. A check on this algorithm can sometimes reveal unsuspected ways of saving on storage costs.

d. Networking. The Bone Marrow Database has 3 different kinds of cases. The central type is obviously that containing the records of a patient. There are, however, two other types which can be linked during retrieval: the protocol case and the event label case. We have 37 protocols of which 21 are active at the moment. Some of these are simple treatment protocols, describing treatment for a single rare disease with only one or two patients enrolled. Most protocols are for research to study transplant preparatory regimen, prevention of infection, graft-versus-host disease, or maintenance chemotherapy. A single patient may be enrolled on several protocols and every patient is enrolled on at least one. Just keeping track of the information concerning the protocols, which patients are enrolled and to which arm they have been randomized is a non-trivial data management problem. In addition, it is necessary to monitor that eligible patients are enrolled, that the rate of accrual is as projected by the sample size estimation, and that special data needed for the study are being collected. By having a case type which describes protocols, there is a two-fold gain: the protocols become easier to administer and when the time comes to analyze, a link between the protocol case and the enrolled patient cases is automatic.

The third type of case is really a file of labels. When the database was first designed, most laboratory reports in the patient's chart and many clinical signs, such as daily maximum temperature, were coded and stored. Two problems with this

approach were quickly discovered. The amount of storage required was prohibitive, and most of the information was useful only for daily patient management, but was not in usable form for research purposes. Statistical analysis is usually of outcome variables, such as time to death, contributing causes of death, time to recurrence, occurrence of infection, graft-versus-host disease, and other complications. These outcome variables need to be carefully defined in the research protocol and uniformly coded on the database as they occur. It is often not possible to reconstruct an event after the fact based on recorded signs and symptoms. We therefore created an event record for these outcomes. The temperature is not important, but the onset of significant fever may signal the start of an infection; the event fever, date of onset and of resolution is coded. The white blood count is not interesting per se; reaching a certain level post transplant implies successful engraftment of the marrow; engraftment is coded with the date.

Each event is coded using the SNOMED (Systematized Nomenclature of Medicine) standard coding system. The label for each event code was initially stored on the SIR file containing variable lables, but each time a new type of event was encountered and coded, the whole database had to be restructured. It is much more efficient to store these event labels and codes as separate records of a label-case. Those retrievals requiring events to have English labels can be run with a link between the patient's event record and the correct record from the label-case.

4. ADMINISTRATION OF THE DATABASE ACTIVITY

By "administration" is meant the coordination of the people and computer activities, not just the management of the data. The clinical management of bone marrow transplant patients requires a large varied medical staff all contributing to the patient's chart. The number of researchers is also large. Consequently, coordination of effort, provided by the database committee, is crucial. Some of the procedures instituted by this committee have had important consequences and are worth mentioning.

Whereas the ability to link event codes and their labels is a convenience, the reorientation of the database to an emphasis on events rather than symptoms has had important scientific implications. The concept is quite simple and it is easy to underestimate the improvement in the power of the database. Much close interaction with the clinical staff has been required to specify which events should be coded, how they are defined, when a patient had a problem, and when it was resolved. A standardized code for graft-versus-host disease and its severity has been developed with the result that all such events are uniformly coded throughout the database. Such standardization imposes more control over the work of individual investigators, but the overall effect is a marked improvement in quality and consistenty and little restriction on innovation. It has also become necessary to insist that the description of each protocol must specify, at the time it is initiated, which endpoints are to be analyzed and what events are of interest, and if any non-routine variables are to be coded onto the database. These are, of course, basic sound scientific principles which are not, however, always followed.

Because events are coded by a medical records technician from incomplete charts, the quality control process involves the attending physicians. Biweekly complication conferences are held to review all records of all patients in hospital or recently seen in clinic. These conferences, which were instituted for the sake of the database, have proven so valuable in improving patient care that the original motivation has been forgotten by the clinical staff and they are viewed simply as good clinical practice. Thus, the accumulating records on a case undergo period review scheduled by the

computer. The primary responsibility for assuring quality of the data rests on the shoulders of the physician most familiar with the case.

The database committee also reviews each protocol before it is approved for patient enrollment. Details of data to be recorded, variables, times and methods of statistical analysis must be specified. These must be reviewed to see what impact the new protocol will have on the workload of the small data center. If endpoints are specified which are not part of the routine, the principal investigator of the protocol may be asked to participate in their collection and quality assurance. A final review of the scientific and ethical merits of the research and a comparison with other competing protocols is also carried out. Only a limited number of patients can be transplanted and priorities must be set to optimize their contribution to research. Again, fortunately, the investigators view this process as necessary for improving the overall quality of the group's research effort, rather than interference with individual creativity.

The database is periodically monitored for signs of serious trouble which might necessitate the early closing of a protocol. There are three main reasons why a study might be stopped early: 1. low accrual. 2. high incidence of severe side-effects. 3. the early occurrence of a statistically significant difference between treatment groups. These stopping rules are carefully formulated and monitored by the database committee to protect both against panic and undue optimism by the research group.

5. CONCLUSION

Modern medical research often requires team effort, especially in the investigation of complex diseases. The integration of a computerized database into the research process can be of great help in increasing efficiency and scientific quality. An example has been given of the design, management, and analysis of such a database serving a research group within a single institution. It has many similarities with other clinical research databases being intermediate in complexity in the spectrum of the types described in the introduction to this paper. Each situation is unique, but I have summarized the approaches and procedures which have been found useful at the University of Minnesota in order to model the design of such databases.

REFERENCES

1. Sherwin R: Kaelber CT, Kezdi P, et al.: The Multiple Risk Factor Intervention Trial II. The development of the protocol. Prev. Med. 10: 402-425, 1981.
2. HDFP Cooperative Group: The Hypertension Detection and Follow-up Program. Prev. Med. 5: 207-215, 1975.
3. Kathe BA, Chan Y, Buehler BA, Evans JH, Fehm P: Protection of patient rights and welfare in the VA Cooperative Studies Program. Controlled Clinical Trials 2: 267-274, 1981.
4. Cutler SJ: Cancer registries; opportunities and responsibilities. J. Nat. Cancer Ins. 57: 741-742, 1976.
5. Laszlo J, Cox E, Angle C: Special article on tumor registries. Present and future prospects. Cancer 38: 395-462, 1976.
6. Surveillance Epidemiology End Results, Incidence and Mortality Data: 1973-1977, NCI Monograph 57, 1981.
7. Perry HM Jr, Schnaper HW, Goldman A, et al: Evaluation of drug treatment in mild hypertension, Ann. NY Acad. Sci. 304: 267-293, 1978.
8. Goldman A, Bosl G, Johnson K, Fraley EE, Kennedy BJ: Prognostic factors in cancer of the testis: An approach to identification.

RESEARCH NEEDS AND DATABASE DEVELOPMENT: PUSH AND PULL

Barbara Stone Meierhoefer

Research Associate
The Federal Judicial Center

*The author is a research associate with the Federal Judicial Center in Washington, D.C.

Abstract

This paper traces the inception of a database reorganization project that got its stimulus from a research question. When asked to look into what constitutes an overburdened court in terms of per judge caseload, researchers at the Federal Judicial Center developed a research design calling for relatively sophisticated time series and survival analyses involving large amounts of data over a ten year period.
Though the necessary data were available from the administative agency which prepares annual statistical reports for the federal court system, they were not organized in a way to permit the case tracking analyses called for by the research design.
As a result, the Center is now in the process of developing a research oriented database which will integrate pertinent federal court case data from 1970 through 1982 and be updated as new information becomes available.

What follows describes how the statistical methodology devised for a particular research project can stimulate the reorganization of an administrative database. The story also contains a warning to other researchers concerning the data requirements of currently popular statistical techniques such as time series and survival analysis. We present this from the point of view of 'users' who must work with large amounts of data that were collected for purposes largely unrelated to research needs.

The Research Question

Tucked away in the judicial branch of the federal government is the Federal Judicial Center, the small agency for which I work. The Center's statutory responsibilities include research, training, and systems development for the federal courts, with an emphasis on court management.

One type of research question that the Center addresses concerns the number of cases that a judge can effectively handle. Judicial caseloads that exceed the judges' capacity can lead to unreasonable delay and other deteriorations in the quality of justice. To avoid these deleterious consequences, there needs to be a good estimate of what this capacity is in order to provide the system with an adequate number of judges.

The magic number of 400 cases filed per judge per year is currently used as a standard capacity threshold in the federal district courts. There has been some dissatisfaction, however, with using this figure to recommend and allocate new district judgeships. Additionally, there is no accepted capacity limit for the appellate courts. Therefore, the Center was asked to explore further the relationship between workload and court burden.

The Research Design

There are a number of problems in assessing "how many cases a judge can judge". First, there will obviously be variation among judges. Second, there will be variation depending on the type of case involved because some are inherently more demanding than others. Additionally, it is unlikely that a single number can adequately identify the point at which a district is overburdened with cases. The impact of 400 filings per judge in one year will surely depend on the history of case filings in the particular district. There also seems to be a barn door problem with using the current state of affairs to recommend new judgeships to be filled in the future.

We have therefore (in typical researcher fashion) rephrased the task to be that of investigating how many cases of a particular type, and under what circumstances, an average judge can handle without experiencing overburden (defined initially for our purposes as delay). The general approach was to examine the distribution of case-processing times to see if we could identify historical patterns of filing, termination and pending caseloads that lead to delay.

Determining the time from the filing to the disposition of a case is a straightforward task. The real question of interest, however, is how much of that time constitutes 'delay'. The time required by judges and attorneys for case preparation and deliberation is not 'delay'. Only that portion of processing time that exceeds the necessary lifespan of a particular case should be considered.

Though these lifespans for individual cases are unmeasurable, "typical" lifespans for particular types of cases can be estimated by their average disposition time over a range of courts and years. Using this as a base, courts which are

experiencing delay can be identified and their caseload history examined for clues to the causes of the current problem. This information should assist in developing warning signals indicating that a court could be headed for "backlog" trouble; cue when and where new judgeships or other court personnel slots should be created; and allow investigation of the management techniques of those courts that seem best able to avoid backlog difficulty.

The Statistical Requirements

The following steps were planned to accomplish the major objectives of the task outlined above.

1. Our research question includes the phrase "particular types of cases". Unfortunately, the existing data system identifies over 200 types of civil cases and approximately 250 different criminal cases. Our first task is to develop useful case typologies that reduce this to a manageable number for which lifespans are to be calculated. Using a construction sample, we intend to use both (1) comparisons of survival distributions to tell us whether the survival curves of particular casetypes are similar or dissimilar and (2) cluster analysis to group casetypes that are similar as to particular case processing attributes. The reliability of cluster assignment will be assessed by discriminant analysis using a number of validation samples.

2. Calculate the 'typical' lifespans for the identified casetypes, using ten years' worth of national case filing and termination information.

3. Using comparisons of survival distributions, compare the individual courts against the national norms to identify courts that have experienced delay or evidence unusual case processing patterns.

4. Examine the caseload history for these courts using time series analysis, and display trends with three-dimensional plotting programs to assist in the explanation of the results.

The Data Problem

You will notice that these are so far only plans, not accomplishments. There is a good reason for this. The project has experienced delay of its own because the data we need, though available, are not structured in a way that fits the statistical analyses we plan to undertake.

For both the survival and time series analyses, we need followup information on all (or a sample of) cases filed during particular time frames. The data we need is spread over more than 30 separate data tapes. We are faced with a classic example of rearranging data collected for administrative purposes to fit research needs. The size of the problem is considerable, involving over one million records.

The Administrative Office of the U.S. Courts is, as its name implies, the administrative arm of the federal court system. Its Statistical Analysis and Reports Division maintains all of the data needed to address our basic project goals including information on case type, district, and date of filing and termination for each case filed in the federal courts. The data are collected for the primary purpose of preparing annual statistical reports.

The data collection process is based on forms completed by court personnel. When a case is filed, an 'opening' form is filled out by a clerk in one of the 95 district courts and 12 courts of appeals in the federal judiciary. A 'closing' form is submitted and matched with the 'filing' form upon case termination. The

Administrative Office has a tape for each "statistical year" (running from July 1 to June 30) that includes opening information for all cases filed during the year. They also have separate tapes which contain both filing and termination information for all cases terminated during a particular year. Up to 1980, still another set of tapes were compiled to indicate which cases were still pending (filed but not yet resolved) at the end of each statistical year.

This database organization is satisfactory for yearly reports. It does not, however, allow for tracking of filing cohorts (all cases filed during a particular period) over time. Suppose we want to compare survival times for different types of cases filed in, say, 1973. We need information from all of the termination tapes from 1973 onward, and the pending tape for the last year of the followup period. Only then can we have complete information, because every case filed in 1973 has either been since terminated or is still pending as of last count.

To select all of the pertinent data from as many as 10 data tapes for each of the various analyses we have planned would be a major undertaking. There are over 100,000 records on each tape. The available variables, acceptable codes, and tape layouts have changed over the time period in which we are interested. We need a database which integrates the pertinent information.

Preparing the Data

The allure of an integrated database containing the detailed time information from the Administrative Office is irresistible given the data disaggregation flexibility then available for various time series and survival analyses. We are taking the following steps under contract to this end:

1. Document the changes in tape layout, variable definition and acceptable codes for the years from '70 through '82.
2. Verify the information in the tapes themselves, flagging consistency check failures (e.g., a case can not be terminated before it was filed) and out-of-range values.
3. Select the variables to be kept and decide on a master format for the data.
4. Design a system for entry and storage of the data.
5. Update the system annually as new data become available.

Researchers are fairly good at telling database system designers their needs. We want a system that integrates existing data and incorporates future data in such a way that it can be easily accessed and fits the requirements of the software packages we know and love.

On the other hand, we 'users' are not really sure how much is possible at what investment of time and resources. For example, we learned through Step 1 of our data reorganization project that shaping the data takes time. Documentation of the changes made to an administrative data system over a ten year period is in itself an exacting and time-consuming task. Any ideas from those of you in the area of database design that can help us as we back our way into a more sophisticated database would be appreciated.

Research Topics in Statistical Database Management

Dorothy Denning
SRI International

Wesley Nicholson
Battelle-Pacific Northwest Labs

Gordon Sande
Statistics Canada

Arie Shoshani
Lawrence Berkeley Labs

Abstract

This report identifies research topics in statistical database management. These topics are grouped into four major areas: characteristics of statistical databases, functionality/usage, metadata, and logical models.

1. Statistical Databases Characteristics

Computer scientists, especially designers of database systems, commonly ask statisticians and data analysts to identify the characteristics or features of a database that identify it as a statistical database. Searching for a profound answer to this question has perplexed data analysts. Many conclude that there are no characteristics which uniquely identify a statistical database. In principle, any collection of quantitative information residing in a computer is a candidate statistical database. As soon as the body of information is interrogated and statistically analyzed, either in total or by sampling or subsetting, it becomes a statistical database.

There are, however, important characteristics that should be built into a database if it is going to be useful for statistical analysis. These characteristics involve adequate description of the quantitative information in the database (i.e., the inclusion of appropriate metadata as defined in Section 3 below.). Such description is essential to understanding inferences evolving from data analysis. Certain kinds of description or definition are almost always included in the database because it is well known that the particular description is critical to understanding the data. On the other hand, certain other information is almost never included even though a detailed analysis will uncover subtleties that are correlated with such description and often cannot be modeled without it. A simple example will serve to illustrate the point. In a database of hospital records, the subject is always described as male or female. This description is important for prognosis and treatment. Periodic readings of blood pressure are also included in the database. On the other hand, the conditions under which the blood pressure was taken -- patient lying down, standing up, sitting; recording made on the left or right arm -- are almost never included. If the protocol dictates taking the blood pressure on the left arm with the patient lying down, then that information should be included in the database. If there is a variety of conditions, then each blood-pressure reading should be accompanied with a descriptor. When does such detailed information become important? When blood pressure is correlated with treatment protocol, we wish to minimize the random error in the measurements. Clearly if systematic changes in readings can be associated with the position of the patient or the arm on which the reading was made, then that random variability is reduced and a more precise statement can be made about the effect of a specified treatment.

There are distinct types of quantitative data that may be recorded in the database. For each type, there are general conditions which should be met if the information is to be described adequately for detailed statistical analysis.

1.1. Missing Data

Almost every statistical database has incomplete records. Proper statistical treatment of missing data usually depends on the reason for the missing data. For example, in a seismology file listing individual station seismometer magnitudes associated with particular earthquakes, values missing because a station was not operational should be ignored in an estimate of earthquake magnitude. On the other hand, values missing because the signal was either below the seismometer threshold or beyond the seismometer range and off scale, bound the magnitude of the earthquake and should be utilized in an estimate of earthquake magnitude.

As in the seismometer example, there are several possible reasons for a missing value. A set of tags to identify the particular type of missing value should be included in the file. In the seismology example, the tags would at least include "non-operational," "below threshold," and "offscale."

In some situations, such as with questionnaires, the logical structure may influence the interpretation of a missing value; e.g., whereas for males it is not important

whether a question on the number of pregnancies is answered, for females, it is critical to distinguish between a nonresponse and zero.

Most database management systems identify missing values but lack proper tagging capability. Research is needed to improve missing value treatment, and, in particular, to include sufficient information in retrievals so that missing values (either included or excluded) can be properly handled during data analysis.

1.2. Data Quality

Knowing the quality of data is important for statistical analysis. For example, if data are keyed into a file from a remote terminal, how frequently are typographical errors made? Are the data cross checked before being accepted? If data come from a measurement instrument, what is the resolution of that instrument? What is the reproducibility of independent measurements on that instrument? Has that instrument undergone modification during the time that the total set of data was collected? Or further, is that instrument recalibrated every day prior to data collection? These are all important questions; their answers may well influence the way the data are handled in any statistical evaluation. The file should include such data quality information. If the quality is uniform over the entire file, this information can be included in the file descriptor; if it varies in a haphazard fashion, it may be necessary to attach it to each datum.

Further considerations with respect to data quality involve the frequency of spurious measurements through either a breakdown in the data-generating system or the introduction of a rare physical phenomenon which grossly changes the measurement process. For example, in a chemical analysis for trace constituents a contaminant in the apparatus could cause major variation in the measurement. Here explanatory flags should accompany the data corroborating the presence of a contaminant or suggesting the possibility of a contaminant.

Finally, when data are collected over a period of time, there may be changes in the data-collection process; e.g., in the method of reporting, measuring, validating, or summarizing. To sort out such effects, a time stamp should be associated with each datum giving the time when the data were generated, and the time of the particular file update when the data were included.

In many situations it is useful to have a "degree of believability" associated with data. For example, economic data on developing countries may be obtained by estimates. Using such data for economic forecasts or evaluation should take into account the believability of the data. Another source of imprecise data is introduced by imputation. Imputed data values should be marked as such and not interpreted as reliable data.

Current database management systems do not have facilities for keeping track of data quality. Research is needed to find economical ways of storing information about data quality, and to find ways of passing this information to the data analyst.

1.3. Data Sparseness

In many data sets, there are structured patterns of missing data. This is particularly the case for designed experiments where the "design" is an optimum sparse coverage of the independent variable levels. Here the structure allows encoding which could materially reduce database storage requirements.

To reduce storage requirements, designers of databases often change the logical structure of the data. For example, a file may be partitioned into multiple segments, or data values (e.g., year) included with a data element name. This practice can obscure the meaning of the data and complicate retrieval.

Research is needed on the handling of sparse data to find ways to economize storage, to describe metadata, and to optimize retrieval while keeping the logical description independent of storage considerations.

1.4. File Freezing

Many databases are dynamic in the sense that they are continually being updated. If a statistical analysis is to be performed, there will be a natural time cutoff. All data resident in the file as of the cutoff point must be identifiable. Thus there must be a capability to segment on time so that information that comes in after the cutoff will not erroneously get into the statistical analysis and possibly bias the results. As a consequence of file freezing, there may be several versions of the same file in existence.

Research is needed to find techniques that impose proper time constraints on retrievals. Research is also needed to find techniques for efficiently storing multiple versions of large files.

1.5. Imprecise Keys

In statistical analysis, information may be needed from various parts of a single file or from several files. Often, this must be done by making a cross reference linkage using imprecise keys. For example, in a hospital database system, all the information on a patient might be retrieved using the patient's name as an imprecise key to search portions of the same file or several files (name is usually an imprecise key because there may be several people in a database with the same name). A file structure that allows cross referencing with such imprecise keys is very useful for statistical analysis. In statistical databases, subsetting and retrieval using imprecise keys is a difficult question that needs research.

1.6. Security

When a statistical evaluation is to be done on a file that contains sensitive information, the question of privacy protection arises. The confidentiality dilemma is to provide useful summary information while protecting the privacy of the individuals. Suitable mechanisms for protecting information may depend on the logical data model. Research is needed to determine what is obtainable within the constraint of summary information

criteria, and how to provide security mechanisms in a multiuser environment.

2. Functionality/Usage

Several issues were raised regarding the desired functionality or usage of statistical databases.

2.1. Subsetting

The key to successful data analysis lies in finding interesting subsets of the data. This requires the capability for multiple key retrievals or, more generally, for retrieval of any identifiable subset of data (e.g., all PhD's in the age bracket 25-40 living in California and earning more than $50,000 annually). Once a subset of data has been formed and analyzed, it is often desirable to retain the subset for further analysis, for aggregation, or for decomposition into smaller subsets. For example, the salaries for the preceding subset of PhD's may be aggregated by profession or by sex, or the subset of PhD's in the computer industry may be extracted for a more detailed analysis. Because subsets are obtained or retained for the purpose of aggregating or summarizing over certain attributes, they are often called summary sets.

Many commercial database systems have facilities for specifying and retrieving arbitrary subsets. The storage and retrieval mechanisms of these systems are not always efficient, however, for statistical database structures, e.g., sparse data. Research is needed to find efficient techniques for statistical databases; transposed files are a good beginning.

Some commercial database systems support view definitions, which permit subset definitions to be saved and managed by the database system. The data in a view is derived from the current state of the database when the view is retrieved, rather than being stored as a separate data set. With large statistical databases, views may not allow efficient enough access to certain subsets; hence, it may be preferable to store these subsets separately. Additional metadata is then needed for describing the subsets and their relationship to the main database. Research is needed to develop techniques for managing these retained subsets.

2.2. Sampling

In addition to forming identifiable subsets of data, it is often desirable to extract samples of the data. This is particularly true for large databases, where it may be infeasible or impractical to analyze the entire database. Sampling can also provide a means of protecting the confidentiality of sensitive data.

Most existing database systems do not support data sampling. Research is needed to develop efficient techniques for defining, retrieving, and retaining samples, and for combining sampling with other subsetting operators.

2.3. Data Analysis

Many existing database systems have operators for computing counts, sums, maxima, minima, and means. Although full data analysis capability should not be the goal of statistical database management systems (see Section 2.6), research is needed to determine which data analysis operators can and should be included in such systems. For example, it is quite efficient to perform the sampling operations in the data management system. In addition, new methods are needed for accessing complex data structures, e.g., hierarchies, by data analysis programs.

The results of data analysis should be self-documenting; that is, they should contain metadata describing the resulting structure. Existing systems do not provide this capability, and research is needed to develop analysis tools that produce self-documenting structures.

2.4. Adaptive Data Analysis

Data analysis is an adaptive process, where intermediate results determine subsequent steps in the analysis. It is often desirable to go back to an earlier step and try a different path. With appropriate computer graphics, much of the analysis could be done on-line without recourse to hard copy.

Existing database systems do not support this form of adaptive analysis. Research is needed to develop techniques for recording analysis paths, and to develop graphical aids for moving along these paths.

2.5. Historical Data

Traditionally, historical data has been difficult to assemble for analysis. If it is saved at all, it is usually archived on tapes. With on-line database systems, historical data can be retained and retrieved by the database system. Research is needed to determine how historical data is best managed.

2.6. Data Management and Statistical Analysis Interface

The data management software and statistical analysis software should not form a single monolithic system that attempts to provide all capabilities for all users. Even if we could predict what capabilities would be required, it would be difficult to develop and maintain such a monolith. On the other hand, the user interface should provide the image of a single system. The data management and statistical analysis capabilities should be constructed from building blocks that allow their easy interface. Research is needed to determine what building blocks are needed, and to develop a methodology for constructing and interfacing them. Several interfacing styles are possible; for example, the database system may drive the statistical analysis system or vice-versa, or both systems may operate as coroutines.

2.7. Distributed Systems

Local and nonlocal computer networks can provide access to distributed databases and to computing resources not available at the user's personal work station. Several scenarios are possible; for example, data from one or more sites may be assembled at a user's personal work station for analysis; data collected at different sites may be analyzed at the sites (e.g., to reduce the volume), and then transmitted to a central database system for further

analysis; data managed at a personal work station may be sent to a more powerful machine for analysis, and the results returned to the work station, possibly for additional analysis. Before any of these scenarios can be fully realized, research is needed to develop mechanisms for managing distributed statistical data and distributed analysis.

3. Metadata

Metadata is information about data. The panel has repeatedly emphasized the importance of metadata for statistical data. Often data becomes obsolete because the information about its content and meaning is nonexistent or lost. The following is a collection of metadata issues that could benefit from further research.

3.1. Meaning of Data

Most data management systems, as well as statistical packages, have a data definition capability for the specification of a data field descriptors such as type, size and acronym. This type of information is necessary for computer manipulation of the data. However, this information is not sufficient to characterize the meaning of the data to people. A description of the origin of the data, how it was collected, when it was generated and modified, and who is the responsible person for its collection is also needed. The description should include the full names of data entities and an explanation of what they represent. Data types of statistical databases are often complex, such as time series, vectors, or categorical variables. In addition, special types of data values may be required, such as codes for missing, unavailable, or suppressed values.

The lack of metadata is even more acute when data is collected through automatic data systems. Here it is necessary to be able to collect some of the metadata automatically as well.

3.2. Metadata of Subsets

As was mentioned in section 2, a large number of subsets can be generated in the data analysis process. In addition, new data values can be generated by computations over previous data values. The metadata for these newly created data sets include the origin from which the data sets were obtained, the operations (selection, sampling, computations) involved, descriptions of the data elements, who created the data sets, and time of generation.

Most of this information can (and should) be automatically obtained by the system at the time of subset creation. Some additional semantic information must be obtained from the user if he wants to keep these data sets for future use. The open research issues are how to capture and store this information efficiently. In particular, if data sets are generated from each other, they would have much descriptive information in common that should not be stored repeatedly.

3.3. Metadata Management

It is necessary to organize and manage metadata, just as it is the case with data. However, metadata typically contains much text, and its structure can be more complex than just text strings. It is therefore necessary to manage metadata with tools that can handle text. Most data management systems and statistical packages have very limited capabilities in this area.

One should be able to retrieve and search metadata, just as one does with data. For example, it should be possible to ask the system for the data sets generated by John Smith after February of this year, or to search for all data sets that have information about a certain topic in a hierarchical fashion. Research is needed to determine how to organize the (mostly) textual information so that it can be searched, retrieved, updated, and automatically maintained.

3.4. Consistency

Unfortunately, the meaning of terms change over time, and they may be inconsistent across data sets. This occurs often when similar data is collected over long periods of time. For example, the boundaries of a county may be redefined in a certain election year, but the change is not reflected in the name of the county. Clearly, it is invalid to compare data collected for that county over several years which include the change, yet it is commonly done because the corresponding metadata does not reflect the change.

Another reason for confusion is the use of the same terms for different data elements. This occurs often when new data sets are generated from existing ones. For example, one data set may contain information about income generated by an average over the entire set, while another may be generated by averaging over a sample. If both data elements are labeled the same (e.g. income), it is easy to make mistakes in comparing them. These changes should be captured in the metadata, and be readily available when the data sets are used. At the same time there should be a way to indicate that the data elements are related.

The reverse problem is one of using different terms for the same data element. It is particularly important if the same data element, such as "state", is used by more than a single file, since this information is necessary to determine if the files are comparable (joinable) over this data element. Using different terms in the same file requires the support of a synonym capability.

Another related need is the use of metadata for comparing or merging data from data sets whose parameters are similar but not identical. For example, suppose that the partitioning of ages into age groups in two data sets is not the same. In order to compare or merge these data sets on the basis of age groups, one needs the metadata describing the age groups.

3.5. Reformatting

It is not realistic to assume that at some point there will be a standard for data formats over all systems. Therefore, the need for reformatting data is inevitable. Metadata should be used to facilitate the automatic reformatting of databases. Research is needed to determine how to organize the metadata and how to use it for the purpose of reformatting. Perhaps a standard for metadata

specifications can be developed.

3.6. Distributed Data

There is additional metadata that is necessary when databases are distributed over several nodes of a computer network. For example, suppose that data is collected and analyzed at several hospital nodes on patients response to a certain drug. If one was to combine such information, it is necessary to synchronize the state of these databases as well as the correspondence between the items involved. Research is necessary to determine what status information should be kept, and how to coordinate such information for queries that involve several nodes.

There is very little development of distributed systems that can handle statistical data, mainly because the difficulties in implementing such systems seem too great. But, as was discussed by many members of the panel, the trend is indeed towards distributed systems of work stations. As powerful personal work stations come down in price, so it is more likely that future data analysis will be performed on a work station that is connected to other work stations and central machines through a computer network. The central machines are likely to contain data that are of interest and are shared by many users, while the work stations will contain temporary or private data sets that analysts currently work on. Thus, we believe that it is not too early to conduct research in the area of metadata in distributed systems.

4. Logical Models

Logical modeling is that part of database management concerned with the meaning of data collected about the real world. The typical logical model encountered in a statistical textbook is the rectangular array or observation on a case by attribute basis. The current status is that the real world is more complex than the logical models of database systems, but that logical database models are more complex and diverse than the logical models handled by standard statistical algorithms.

4.1. Complexity of Data

The data organizations encountered in statistical textbooks are data matrices or contingency tables. The mathematical machinery used is the matrix and vector algebras or calculus. The traditional interface with computer science has been the numerical analysis of the computational processes needed to implement the arithmetical processes.

When the data becomes more complex, of which the hierarchical relationship of individuals to a family is an example, differing information is relevant in different subsets of the data, and the classical notations quickly loose their elegance and power. In complex situations, the identification of an appropriate unit of analysis, and the collection of data for that unit, may become substantive problems. All of this may have the additional complication of missing and erroneous values. The notation needed to deal with other types of relationships, such as networks, is often weak and has weak associated theory. With complex data structures, the interface with computer science grows to include algorithms and data structures, computational complexity, and database management.

4.2. Missing Data

A common characterization of complex situations is the need to use and identify insightful subsets. In the presence of missing and erroneous data, this may be difficult. The missing data may arise for many reasons - not observed and not defined or relevant are the standard cases. The ability of database systems to approximately deal with the various types of missing data is weak in current practice. The initial machinery typified by the not-a-number symbols (NaNs) of the IEEE floating point standard have not been expanded or integrated into control mechanisms (query languages) of database systems.

4.3. Data Aggregation

The various attributes of data may be more complex than is realized. Hierarchical relationships may be multifaceted in practice. For example, in geographic aggregations, the notion of county and metropolitan area are intermediate between municipality and state and of equal standing; either may be embedded in a strict hierarchy. The form of the aggregation may change over time so that both analysis and representation are further complicated. Simple responses may be either multiple or repeated in practice. The representation of complex data which has been fully and correctly observed is now possible, but the methods to deal with partially or incorrectly observed data have not been developed.

4.4. Documentation

The logical data model is part of the description of the data and should be included in the documentation of the data. The metadata has the role of communicating both the internal technical facts about the data, including the data models used in its representation, and the external information available about the data. The meaning of the data may be derived both from the data models and the external knowledge about the data.

Logical data models should be associated with good analysis methods. The models that are available await analysis techniques, some of which may arise in the interaction of statistics and algorithm design. Some of the known problems with existing models are the identification of appropriate analysis units, and the bringing of data to those units. The current algorithms often are weak in the presence of the various forms of missingness and errors present in data.

Acknowledgements

Mervin Muller joined some of our discussions, and we are grateful to him for sharing with us his experience and insight.

References

There is an extensive literature covering the different aspects of statistical databases and statistical software. Instead of giving a long list of references, we mention a few surveys and collections of papers, all of which con-

tain many references.

Reference 1 below is an introductory paper to the area of statistical databases. It discusses several problem areas and surveys some existing solutions and work in progress. Reference 2 discusses extensively metadata structures and needs. Reference 3 discusses the security aspects of statistical databases, and surveys existing and proposed controls. Reference 4 contains numerous papers and abstracts presented at a specialized workshop on statistical database management. Reference 5 is a large volume that describes and compares statistical packages and other noncommercial statistical software. Reference 6 is the proceedings of an annual conference that has been held over the last 15 years, and that contains (especially in the more recent issues) several papers on statistical databases.

1. Shoshani, A., Statistical Databases: Characteristics, Problems, and Some Solutions, *Proc. Eighth International Conference on Very Large Data Bases*, Sept. 1982, pp. 208-222. (Copies available from: VLDB Endowment, P.O.Box 2245, Saratoga, CA 95070.)
2. McCarthy, J.L., Metadata Management for Large Statistical Databases, *Proc. Eighth International Conference on Very Large Data Bases*, Sept. 1982, pp. 234-243. (Copies available from: VLDB Endowment, P.O.Box 2245, Saratoga, Ca. 95070.)
3. Denning, D.E. and Schlorer, J., "Inference Controls for Statistical Databases," *IEEE Computer*, (to appear July 1983).
4. *Proceedings of the First LBL Workshop on Statistical Database Management*, Dec. 1981. (Copies available from: Computer Science and Mathematics Dept., Lawrence Berkeley Laboratory, Berkeley, Cal. 94720.)
5. Francis, Ivor (Editor), *A Comparative Review of Statistical Software*, 1977. (Copies available from: The International Association for Statistical Computing, 428 Princes Beatrixlaan, 2270 AZ Voorburg, Netherlands.)
6. *Proceedings of the Computer Science and Statistics: Annual Symposium on the Interface.* (Copies available from different places, depending on the year of the symposium.)

2. Workstations and Special Purpose Hardware

See Also. . . .

A MULTI-TREE AUTOMATON FOR EFFICIENT DATA TRANSMISSION

K.A. HAZBOUN

Pennsylvania State University
University Park, PA 16802

J.L. RAYMOND[1]

Ohio Bell Telephone
Breckville, Ohio 44141

ABSTRACT: A two-stage finite state implementation of an efficient compression algorithm is outlined. This design is proposed for the economic transmission of large volume of data within a distributed network of statistical databases. The encoder/decoder stage of the design is based on a reversible semantic-independent variable-length character technique that makes use of the group locality of character reference behavior and the variable frequency of character occurrence within a well-defined subgroupings of the character set. The two-stage automata, while specifically designed for a modified version of the Hazboun-Bassiouni Multi-Group algorithm, is extended to support any single character look-ahead procedure.

Key Words and Phrases: Automaton, finite state machine, data compression, statistical databases, microcode memory, regular language, failure node, failure bit string, accepting state, local tree re-entry, address, look-ahead logic.

1. **Authors' current addresses:** K.A. Hazboun, IASC, 302 Rackley Bldg., Penn State University, University Park, PA 16802; J.L. Raymond, Ohio Bell Telephone, Corporate Data Center, Room 445, 6889 Snowville Road, Breckville, Ohio 44141.

1. INTRODUCTION

The increasing use of computer-based systems, to support our modern, technological, administrative and office environment needs, mandates the availability of highly integrated distribution systems. The driving force of converting raw data into information is a highly efficient communication network among the various databases and the user. We believe that the increasing demand for information at the various nodes of distributed network does not have to be matched with an increasing economic burden of acquiring large storage devices and high-speed transmission media; rather, the problem may be alleviated through the software and hardware implementation of efficient data compression techniques.

The functional scheme presented in this paper, is a two-stage finite state machine implementation of a highly efficient compression algorithm. The algorithm is a reversible semantic-independent variable-length character encoding method that makes use of two observed characteristics of the distribution of characters within most statistical databases. First, the group locality of character reference behavior. Second, the variable frequency of character occurrence.

A horizontal finite state machine in its purest form will assign only one function to each bit, thus eliminating any vertical logic for decoding instructions. A typical microcomputer uses the maximum binary representation for an operand field, e.g., 2 bits to produce 4 functions or 3 bits to produce 8 functions; either would require external logic. On the other hand, a horizontal finite state machine uses as many bits in the micro-word as the number of functions desired, thus no combinatorial logic is needed. A microcomputer requires many clock cycles in order to fetch, decode, and execute an instruction. Whereas, a finite state machine executes all these functions in one clock cycle, typically, running at the serial data rate.

Another difficult area to deal with is the synchronous to asynchronous interface. A subsystem must deal with two timing systems. The state machine portion runs synchronously with the bitstream clock but the control logic needed for host control must operate with system timing. On the other hand, a typical microcomputer requires

extra timing interface for the network medium and the host interface. The choice of a state machine versus a microcomputer chip used as a controller is governed by two factors, speed and complexity. A complex logic tree would certainly require a microcomputer chip. A binary tree is ideal for a state machine which runs at memory speed (or can be operated at one bit per second to observe all processes if desired). A microchip would be adequate for most common data rates but multiplies the system interface complexity at both ends for a simple procedure such as outlined in this paper.

In the following sections, we first define the data compression algorithm, the subgroupings of the character set, the corresponding tables and local binary trees. The various stages of the finite state machine are then outlined. The model contains the elements required to describe the mode behavior in terms of its inputs, outputs and timing. The paper also provides a separate section on possible extensions and limitations of the proposed design.

2. ALGORITHM

Two observed properties of data structures within a statistical database are the group locality of character reference behavior, and the variable frequency of occurrence of different characters within a well-defined subgrouping of a character set. The first property refers to the tendency for a string of characters (e.g., a data field within a record) to consist of a specific subset of the character set, such as, alphabets, digits, successive blanks or zeroes. This locality of character reference behavior may extend over two or more adjoining fields. The second property implies that the skewness in the frequency of characters [1] may be extended to a well-defined subgrouping of the same character set [2]. The division of the character set into different subgroups and the variable frequency of character occurrence within each subgroup are the determining factors in lowering the length of bit representation per character.

Using a Multi-Group (MG) encoding scheme, Hazboun and Bassiouni [2] have demonstrated an average character compression as low as 2.8 bits per character, and reported an overall compression efficiency of 17%-40% over the Huffman algorithm. The Multi-Group algorithm is constructed as a two-level hierarchy of Huffman-type binary trees. The first level represents the local trees for the subgroups of the character set and the second level represents a set of binary trees which act as the switching mechanism between any two subgroups in the event that the next character of the string being encoded belongs to a different subgroup.

In this section we present a modified version of the Multi-Group algorithm, denoted as MMG, which is more adaptable to a hardware implementation. In this modified version, the authors have eliminated the two-level hierarchy of the local trees and the associated failure trees with no loss of meaningful expression. Additionally, the Multi-Group string attribute (the consecutive occurrence of a single character within a character stream, e.g., zeroes, blanks, etc.) which had been limited to the 'blank' character in the original algorithm was extended to any character in use. A subset of the ASCII character set and its subgroupings is presented in table 1. While the newly constructed local trees are described in figure 1 as a state transition diagram using the subset of characters given in this example, they may be extended to the full complement of any character set.

Sub-Group	Character	Relative Frequency	Multi-Group Local Code
I	A	6	001
	I	5	111
	N	5	100
	O	6	000
	S	4	110
	T	6	101
	Y	1	0111
	⊬	4	0110
II	0	16	11
	1	6	010
	2	7	100
	3	6	011
	4	5	001
	5	9	101
III	⊬	14	1
IV	-	1	11
	&	1	10
	$	1	01

TABLE 1: The restricted character set, their relative frequencies, and the Multi-Group local binary codes.

Let, N, represents the number of characters within a specified character set; s, the number of subgroups; n_i, the number of characters within subgroup i. By definition,

$$N = \sum_{i=1}^{s} n_i.$$

For each subgroup, S_i, a local binary tree is constructed. Appended to each local tree are the extension states which implement the switch pointer mechanism to each remaining subgroup. The leg of

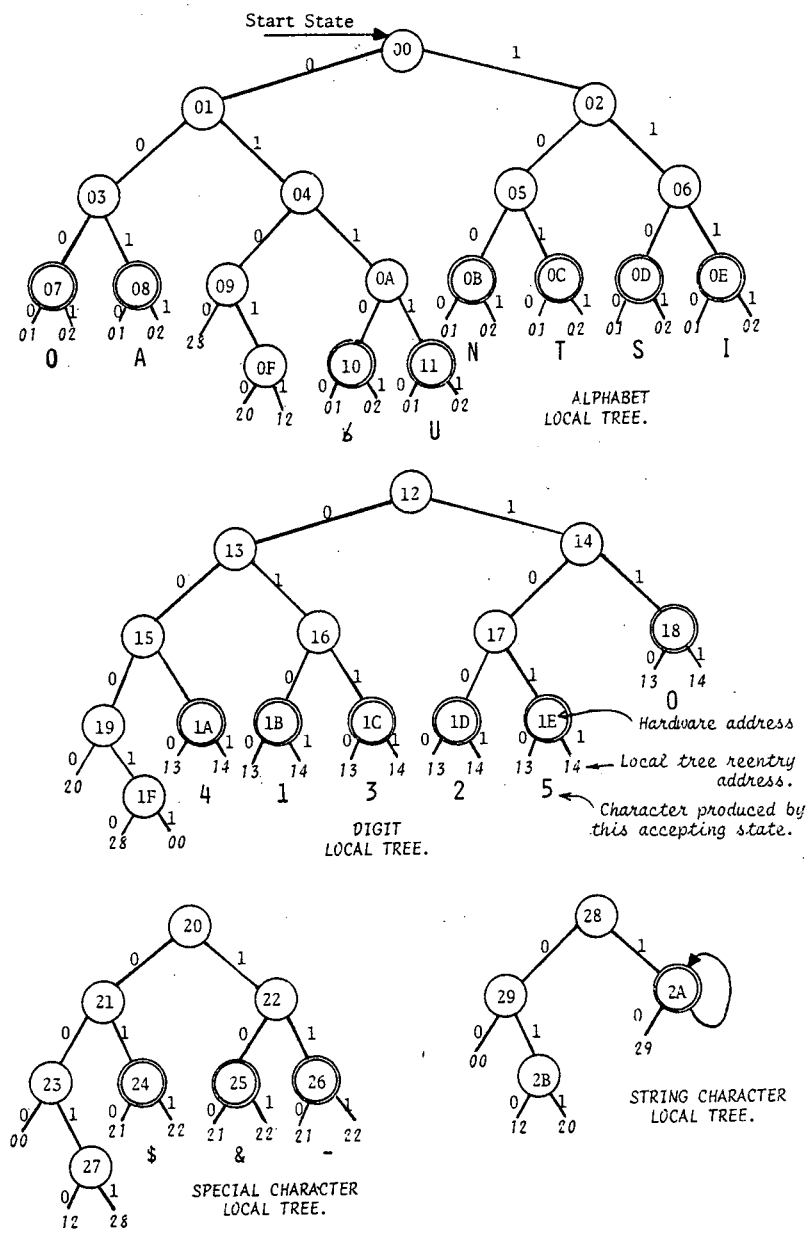


FIGURE 1: TRANSITION STATE DIAGRAM: The above diagram represents the construction of the transition state diagram corresponding to the four subgroups as defined in the restricted character set of Table 1. The finite automaton accepts a bit string if the sequence of transitions corresponding to the symbols of that particular bit string leads from the start state to any accepting state and then back to the reentry address of the local tree. Each accepting state (double circled) represents a single and unique character within the restricted character set. If a failure occurs, the entry address points to the vertex of one of the remaining local trees.

the tree to which the state switch indicators are appended is the failure indication leg of that local tree. If subgroup; S_i, has, n_i characters then the local tree for that subgroup has, $n_i + s - 1$, external states. The additional, s -1, external states represents the entry states of the remaining local trees. Note that these local trees are constructed by applying the Huffman's algorithm on the appropriate frequencies of the, $n_i + s - 1$, external states, where the, s - 1, nodes are assigned a joint relative weight equal to the failure frequency within S_i multiplied by the sum of the weights of the n_i characters of S_i.

The Encode/Decode Function

Either of two approaches may be adopted for building the coding tables and control decode local trees for a particular system environment or an individual database. First, the generalized approach where statistical information is collected from a representative sample of databases within a system environment. Second, individualized statistical information is collected from each database to reflect any variation among the different databases. The latter approach undoubtedly produces a more efficient compression. In that case, each file will have its individualized coding tables and, consequently, its own level of security. If greater security is desired, it is possible to implement a dynamic start code to be used by the decoder to start processing the bit stream. This code could be implemented as a run time option known only to the user of the file and not available in any database file.

When this implementation is used for transmission, the switching of translate tables must be coordinated by the sending host; i.e., if data has been sent using one local tree structure and is desirable to change to another tree structure better suited for the next file to be sent, the sending host will transmit a file containing the new table using the current tables. The receiving host acknowledges receipt of the new tables and instructs the sending host to load its new tables and wait for the next transmission which will be returned by the receiving host using the new table. If the load has been successful, the sending host will acknowledge receipt of the transmission using the new tables. Both ends are now in sync for block count, check sum, etc. Any failure of this process should cause both ends to adopt the last successful pattern.

In the compression process, if the current character belongs to the same subgroup as that of the last encoded character, the bit representation of the current character, as obtained from the coding table corresponding to the local tree of its subgroup, is used directly. However, if the current character belongs to a different subgroup, a 'failure bit string' (FBS) is transmitted, see table 2. The FBS string, which is used to indicate a subgroup failure and to specify the next local tree, consists of two parts: 1) a failure indicator (FI), which is the bit representation of the failure path through the local tree, and 2) a switch pointer (SP) indicating the next local tree to be used. Table 2 represents the failure bit string matrix among the various subgroupings of the character set.

	Alpha	Digits	String Group	Special Characters
Alpha	-	01011	0100	01010
Digits	00011	-	00010	0000
String Group	00	010	-	011
Special Characters	000	0010	0011	-

TABLE 2: FAILURE BIT STRING MATRIX: The above matrix represents the overhead of a failure from any subgroup to all the other subgroups.

3. PROCESSOR IMPLEMENTATION

The Multi-Group encoder/decoder is conceived as an I/0 board in a host machine. Parallel data is exchanged over the bus and the host downloads the table memories and controls all functions via status flags and I/0 commands. The host is also responsible for data check sum, data blocking and any sequential block verification hidden in the first part of the data. This model is presented in an unbundled design which is amenable to hardware or firmware application in bit-oriented data transmission disciplines or mass storage systems.

Transforming the MMG algorithm into a finite state model which is amenable to a hardware implementation requires attention to the following concerns:

1. Characters will produce bit strings of variable length.

2. Any one string may represent different character depending on the local tree where that character reside.

3. The one character look-ahead capability by definition requires at least one extra register to store a character while the following character is being analyzed.

4. The model must function independently of any external assistance (i.e. the burden of look-ahead must reside in the model and not in the host.)

The design was divided into two parts, a decode state machine and an encode state machine, see

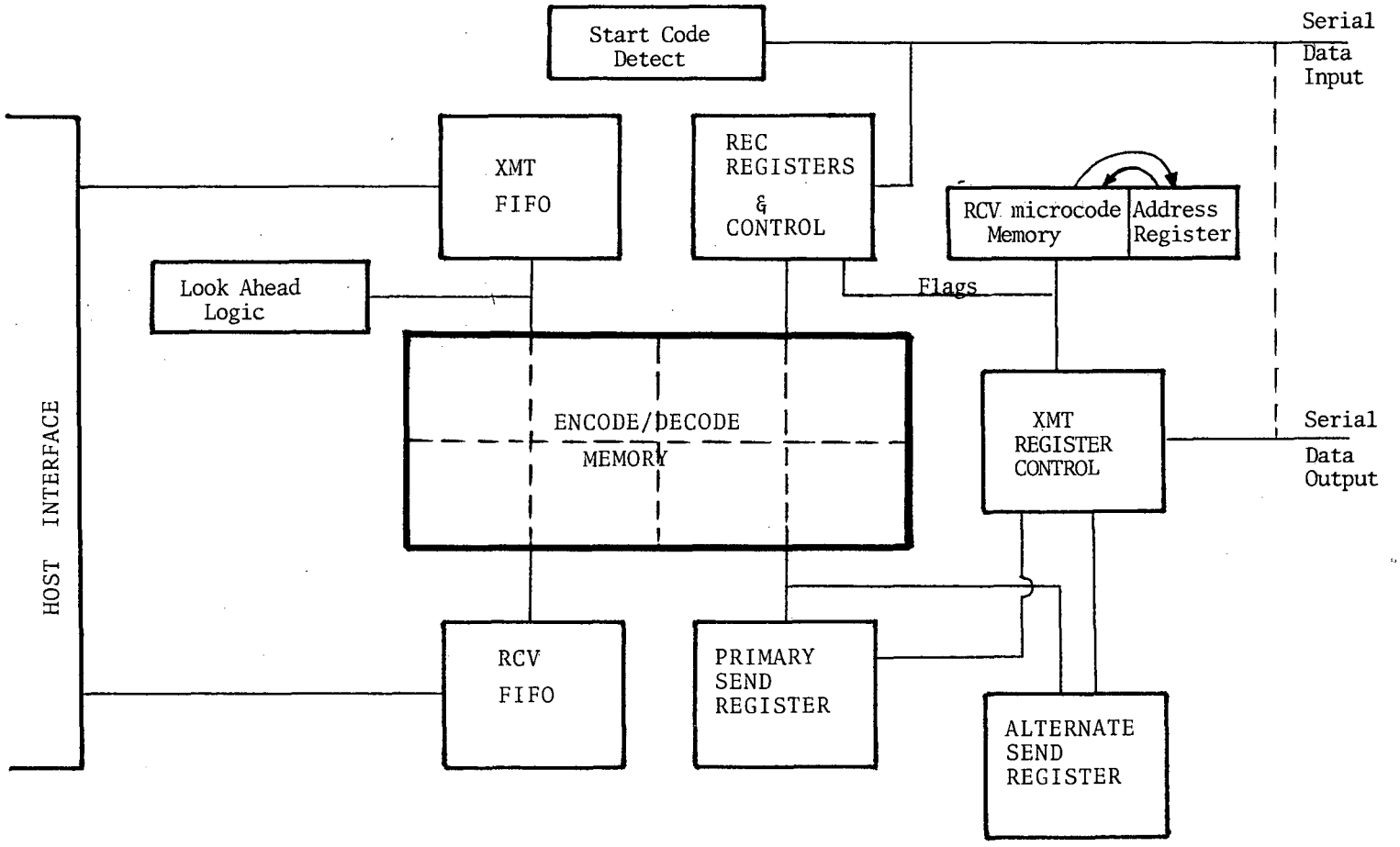


Figure 2: Functional Block Diagram of the Finite State Machine

figure 2. The decode state machine consists of an input register and control logic, a receive FIFO to hold decoded characters, the microcode memory and address register, and one half of the encode/decode memory. A serial bit stream and clock stream are sent into the decoder. Upon recognition of a start pattern the receive transition diagram will advance according to the 'zeroes' or 'ones' received. As each accepting state is reached, the pattern contained in the receive register at that moment in time is used as an address within the map of the receive-half of the encode/decode memory. The resulting data word, at the output of the memory, is loaded in the receive FIFO for transfer to the host.

The encode state machine consists of the transmit FIFO, the transmit-half of the encode/decode memory, the primary and alternate send registers, transmit control logic, look-ahead logic, and three flags from the receive state machine which are used to monitor the transmitted bit stream. As parallel data is processed, the look-ahead logic operates on characters with string attributes. The characters are applied to the transmit-half of the encode/decode memory as addresses. The output data of the encode/decode memory is loaded into the primary and alternate send registers. Serial data is transmitted from one of these two registers at every state of the transmit machine based on flags obtained from the transmit data word and three flags coming from the receive microcode memory which identify significant patterns and strings.

In addition to its own logic the encode machine uses the decode pattern detection firmware. The decode machine will always execute a jump instruction based on its current state and the next bit being received. It simply selects one of two jump addresses provided by a microcode data word. The choice is made by the next "zero" or "one" being received as shown in the Receive Code tables outlined in Appendix A. In addition to carrying two jump addresses, the data word carries map select bits; these bits select the block of memory required by the current tree to properly decode a pattern. The three bit flags identify the current state as either an accepting state or a failure bit string state.

The encode machine actually consists of two distinct sections, the register control section and the look-ahead section. The register control section uses the three single bit flags from the decode machine (which monitors the output bit stream) to identify the last bit of the three types of strings being sent -- valid strings, failure indicator (FI) strings, and switch pointer (SP) strings. The look-ahead logic section examines each character and determines the bit string to be sent by using a pair of flags obtained from the transmit-half of the encode/decode memory. Each memory word in the transmit-half carries two string fields, map select bits and two single flags, see appendix A.

Parallel data is exchanged via FIFO registers used as the synchronous/asynchronous interface. Use of a FIFO simplifies the host interface to the model.

The heart of the state machine is the single instruction processor which has all the power needed to climb through a logic tree of binary decisions. The microcode memory carries two jump addresses: 1) decode map select bits, and 2) register control bits. The microcode address register simply selects one of two jump addresses based on the current data bit being received within a subgroup. In this case, the map select bits are the same at every location and the valid pattern bit indicates that an accepting state has been reached, the current pattern must be decoded, and the receive register zeroed out except for the next arriving data bit.

In transmit mode, all receive activity occurs for one purpose -- to identify an accepting state or failure within a subgroup. Additionally, in this mode, the encode/decode memory functions as an extension of the microcode memory but running at character instead of bit rate. The requirement for one character look-ahead imposed on the transmit logic requires two extra hardware registers over what would be required for a one-to-one character mapping such as the Huffman code [3]. Any further enhancements such as parsing for string content (e.g., 2-character look-ahead) would take the model out of the class of the machine being used here.

Internal Logic

The receive state machine is a series of binary decisions based solely on the current bit being processed. The transmit mode uses all the receive logic with additional look-ahead feature. The transmit state machine requires a hardware register both prior to and after the encoding process since a valid character which has 'string attribute' cannot be processed until the following character is examined. Consequently, the character must be stored in a register until the following character's attributes are determined.

The encode/decode memory is subdivided two ways. First, into two parts by function of send or receive. Second, by the number of local trees within each half. In the decode state, the encode/decode memory map select bits are obtained from the receive microcode memory. In the encode state, the map select bits are obtained from the encode/decode memory acting as an extension of the receive microcode memory. In the transmit map word shown in appendix A, five fields are defined -- a primary data field, and alternate data field, map select bits, and two single-bit flags. The primary data field carries either the string representation of a valid character or the switch pointer for an invalid character. The alternate data field carries a failure indicator or failure bit string. The map select bits point to the valid map for a character regardless of the tree in which it is

located. The two bit flags indicate 'string attribute' and 'valid/invalid' characteristics within the current subgroup -- all characters have flags and data entries in all subgroups.

Decoder

Let M = (K, L, d, q_o, F) represent the decode automaton where K, represents the set of states; L, the input alphabet; F, the set of final states; q_o, the start state; and 'd', the transition function such that

K = (00, 01, 02, ... 34)
L = (0, 1)
F = (07, 08, 0B, 0C, 0D, 0E, 10, 11, 18, 1A, 1B, 1C, 1D, 1E, 24, 25, 26, 2A)
q_o = (00)

and, d (q_x,0) and d(q_x,1) which represent the two transition states or jump addresses to which the current state may advance based on the nect input data bit, are given in appendix B. The closure property may simply be interpreted from the tables since the decode automaton must jump to any one of two states for every bit received.

Encoder

Let M' = (K', L', d', q'_o, F') represent the encode automaton where,

K' = (1024 address locations of the encode map)
L' = (Full character set, Invalid flag, String flag, and Binary map select bits)
F' = (Set of valid characters within their own local trees)
q'_o = (111111111111) = Address of start code pattern

and, where the valid transition functions for the alpha, digits special characters and string characters are represented in Table 3. Naturally, d'(q'_x, 0, 1, yy) = $\emptyset$ for yy = 01, 10, 00 since a character cannot be both 'valid' and 'string' in any subgroup except the alpha local tree, and d' (q'_x, 1, 1, yy) = $\emptyset$ for yy = 11, 00 since a character can be invalid and 'string' only in yy = 01 (digit local tree) and yy = 10 (special character local tree). The representation '$\emptyset$' denotes an empty set.

Every character in the valid character set does produce data in every tree and thus constitutes the full set of states. The subset of accepting states are the address locations represented by each character within a valid tree. Additionally, in the primary data field, every address location in every tree carries either a valid string to represent a character or a switch pointer to the tree where that character is valid. The alternate data field in every address location in every tree carries either a failure bit string, a failure indicator or is left unused in the case of invalid non-string characters. Any bit combination (if applied to any address location of any tree) will produce a defined result.

The encoder stage has a more difficult task since look-ahead logic requires extra registers to accomodate a character stream with string attributes. If the current character is valid, the primary data field carries the encoded string and the alternate field carries the failure indicator (FI) for that local tree. If the current character is simply invalid (non-string), the primary data field carries only the switch pointer portion of the failure bit string and the alternate data field is meaningless. A simple failure (see fig. 3) is handled by sending the failure indicator left in the alternate data register (by the previous valid character) and appending the switch pointer (SP) from the primary data field of the current invalid character.

Any occurrence of a 'string attribute' character is handled as follows: if it occurs in a tree where it is valid, it must be saved in a holding register and

Binary Flag Conditions	Alpha	Digits	Special	String
$\overline{INV} \oplus \overline{STR}$	d'(q'_x, 0, 0, 11)	d'(q'_x, 0, 0, 01)	d'(q'_x, 0, 0, 10)	d'(q'_x, 0, 0, 00)
INV $\oplus \overline{STR}$	d'(q'_x, 1, 0, 11)	d'(q'_x, 1, 0, 01)	d'(q'_x, 1, 0, 10)	d'(q'_x, 1, 0, 00)
$\overline{INV} \oplus$ STR	d'(q'_x, 0, 1, 11)	-	-	-
INV $\oplus$ STR	-	d'(q'_x, 1, 1, 01)	d'(q'_x, 1, 1, 10)	-

TABLE 3: VALID TRANSITION FUNCTION MATRIX: The above table represents the valid transition functions for alpha, digits, special characters and string characters respectively. INV, STR denote the invalid and string flag conditions.

the FIFO dumped to gain access to it. If is is also a 'string' type character, the FI and SP are both sent from the alternate data field. If it is non-string then the saved character is regated to the encode memory and sent normally. These processes are outlined on the right half of fig. 3.

A complex failure occurs when a 'string attribute' character appears in a tree where it is invalid. Since it is multiply-defined, the next character must be analyzed as before and correct FI @ SP sent. In figure 3, the upper left quadrant outlines the hardware processes which take place when a space is encountered after a digit or special character. (Note that a space 'failure' in the string or alpha tree still goes only to one other tree).

The encode automaton changes major states at character or pattern rate instead of bit rate with the exception of the look-ahead logic. The serial bitstream produced by the encode automaton is fed immediately to the decode automaton for the sole purpose of identifying accepting states which mark the end of a variable length pattern transmission.

The encode transition state diagram is presented in figure 3. The encode memory maps for the alpha and digits are defined in appendix A. The memory maps for $d'(q'_X, M, N, 10)$ for M, N = 0,0 1,0 1,1 and $d'(q'_X, M, N, 00)$ for M, N = 0,0 1,0 which represent the special characters local tree and string local tree respectively, can be similarly constructed from the data in table 1.

Extensions & Limitations

Several possible variations may be implemented. First, the chosen groups do not have to be mutually exclusive, i.e., some characters can belong to more than one group. It is sometimes more efficient, for example, to put the blank character separating two alpha words in the local tree of the alphabets and a decimal point '.' in the local trees of both the digits and the alphabets. Furthermore, the 'string subgroup' can be extended to more than the blank character (e.g., zeroes) if desired. However the hardware logic of the outlined model assumes that there is only one special subgroup of characters with 'string' attributes; i.e., if a character is in the 'string' subgroup, it can only be a member of one other major subgroup or class.

A more sophisticated scheme is to let the Statistics-Gathering phase define the subgrouping of characters based on the distributional and the corrolational characteristics of the file being processed. If the statistics gathering program discovered that, in a particular database or transmission function , many numeric fields contained a '$' and a '.', those two characters could be placed in both the digit and special character groups. The objective is to eliminate numerous microcode subroutine calls of the special character group while encoding the numeric fields of a statistical file.

The design presented in this paper is half-duplex. To implement a full duplex system require duplication of the receive portion of the design. Additionally, two practical assumptions were made. First, a start pattern must be agreed upon. Second, a start group must be agreed upon. The start code must be loaded in the primary data field at address "all ones", and the alternate pattern field will carry the failure indicator for the alpha group, assumed to be first. Any preamble needed for system synchronization can be sent and it will be ignored until the specific start pattern is detected. If this start code were obtained from an I/O register, a password type security is possible. The design assumes a hard wired start code.

5. CONCLUSION

A simple two-stage finite state machine for the efficient transmission of data within a distributed network was presented. The design is based on a modified version of the Multi-Group algorithm for data compression. The discussion throughout this paper is intended to stimulate interest and research in this field of data management by persons possessing modest resources without access to a laboratory with advanced instrumentation.

Acknowledgement

We would like to thank John Dorband (HRB-Singer, Inc, State College) who was kind enough to review the processor implementation of this design and provide us with his constructive comments.

References

1. Hazboun, K.A. and Bassiouni, M.A. "A Multi-Group technique for data compression". Proceedings International Conference on Management of Data (SIGMOD), Assoc. of Computing Machinery, Orlando, FL, pp. 284-292, Jun 1982. K.A.Hazboun is one of the current authors.

2. Huffman, D.A. "A method for the construction of minimum redundancy codes". Proc. IRE, Vol. 40, No. 9, Sept 1952, pp. 1098-1101. This is the original paper describing the fact that characters do not occur with equal frequency along with an explanation of how the Huffman coding scheme leads to a minimum variable length codes.

3. Wells, M. "File compression using variable length encodings". The comptr. J. 15, 4 (1972), 308-313. The use of both hardware and software techniques for coding and decoding data using variable coding lengths are presented. A graffic representation of a method of assigning Huffman variable length codes is also offered. In the current MMG implementation, if the map select bits are loaded as all zeroes in both the encode and decode state machines, then a one-to-one mapping of characters to Huffman variable length strings is possible. However, the MMG algorithm is more efficient that the Huffman scheme.

* * * * * * * * *

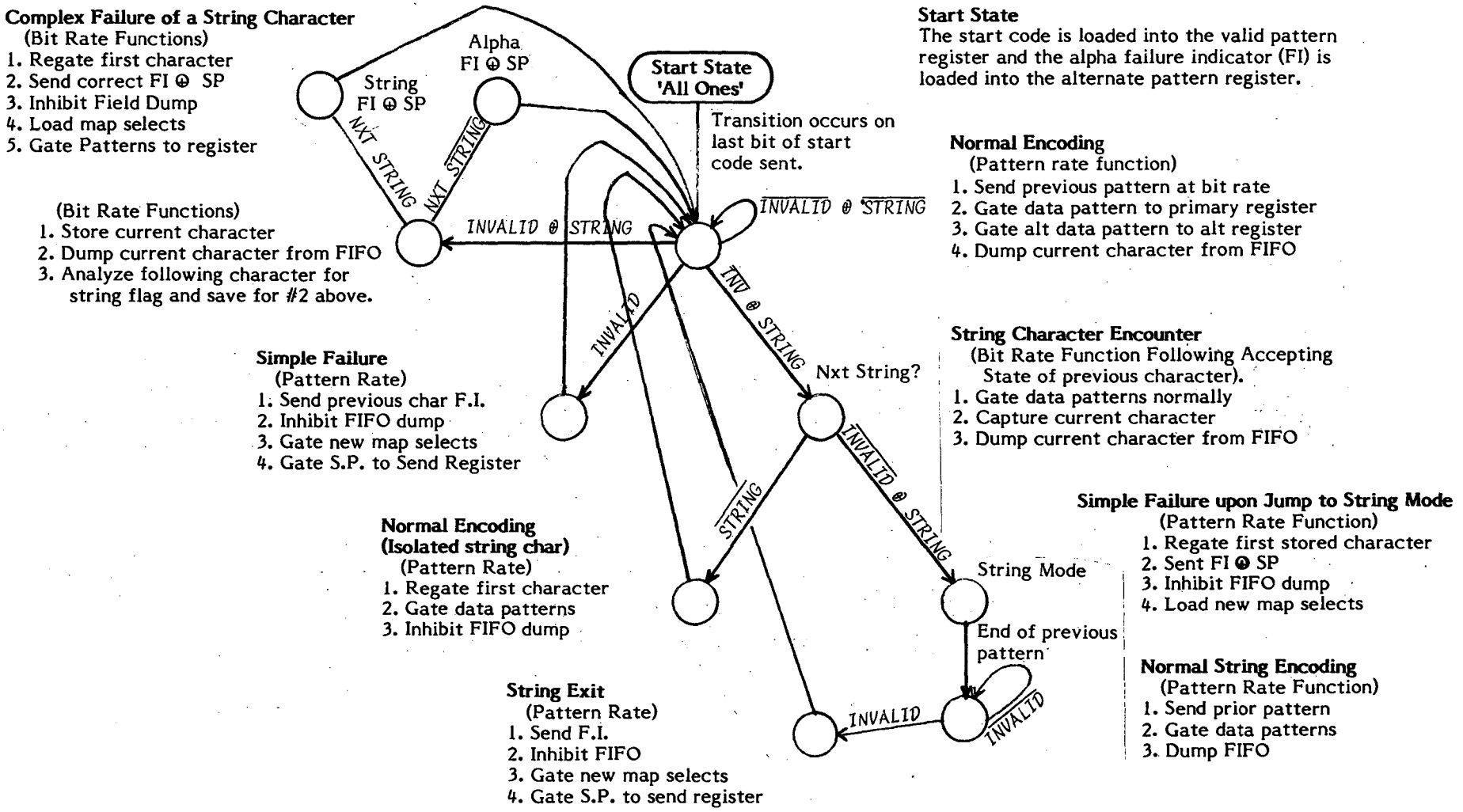


FIGURE 3: ENCODE TRANSITION STATE DIAGRAM: The encode transition state diagram defines the behaviour of the transmit state machine within any local treee for any path defined for that local tree. The defined and undefined paths for each local tree are given in table 3.

APPENDIX A

ENCODE MEMORY MAPS
$d'(q'_x, M, N, 11)$ for M, N = 0,0 1,0

q'_x	Valid/ Invalid Flag (M)	String Attribute Flag (N)	Primary Pattern Output	Alternate Pattern Output	Valid Map Pointer
Encode Memory Map for Alpha					
A = 41 HEX	0	0	001	010	11
I = 49	0	0	111	010	11
N = 4E	0	0	100	010	11
O = 4F	0	0	000	010	11
S = 53	0	0	110	010	11
T = 54	0	0	101	010	11
U = 55	0	0	0111	010	11
ß = 20	0	1	0110	0100	11
0 = 30	1	0	11	-	01
1 = 30	1	0	11	-	01
2 = 32	1	0	11	-	01
3 = 33	1	0	11	-	01
4 = 34	1	0	11	-	01
5 = 35	1	0	10	-	10
- = 2D	1	0	10	-	10
& = 26	1	0	10	-	10
$ = 24	1	0	10	-	10
UNDF = 20	0	0	0110	010	11
Encode Memory Map for Digits					
A = 41 HEX	1	0	11	-	11
1 = 49	1	0	11	-	11
N = 4E	1	0	11	-	11
Q = 4F	1	0	11	-	11
S = 53	1	0	11	-	11
T = 54	1	0	11	-	11
U = 55	1	0	11	-	11
ß = 20	1	1	11	10	11
0 = 30	0	0	11	000	01
1 = 31	0	0	010	000	01
2 = 32	0	0	100	000	01
3 = 33	0	0	011	000	01
4 = 34	0	0	001	000	01
5 = 35	0	0	101	000	01
- = 2D	1	0	0	-	10
& = 26	1	0	0	-	10
$ = 24	1	0	0	-	10
UNDF	1	0	11	-	11

ENCODE MEMORY MAPS for the alpha and digit local trees. In all cases q'_x defaults to some character. In the alpha local tree, the alternate pattern for the blank carries the switch pointer as well as the failure indicator. For the digit tree, an undefined character will be directed to the alpha local tree where, the character can then be encoded. When a blank is encountered in the digits or special subgroup, it is necessary for two switch pointers to be present and selected based on the next character. If another 'blank' is encountered then the failure indicator and switch pointer will send the automaton to the string local tree. If not, it will send it to the alpha tree. The encode memory maps for the special character and string subgroups can be similarly constructed.

APPENDIX B

q_x	$d(q_x, 0)$	$d(q_x, 1)$	Accepting State Flag	Map Select Bits	SP	FI
Receive Microcode for Alpha						
01	03	04	0	00	0	0
02	05	06	0	00	0	0
03	07	08	0	00	0	0
04	09	0A	0	00	0	0
05	0D	0E	0	00	0	0
07	01	02	1	00	0	0
08	01	02	1	00	0	0
09	28	0F	0	00	0	1
0A	0F	10	0	00	0	0
0B	01	02	1	00	0	0
0C	01	02	1	00	0	0
0D	01	02	1	00	0	0
0E	01	02	1	00	0	0
0F	20	12	0	00	0	0
10	01	02	1	00	0	0
11	01	02	1	00	0	0
Receive Microcode for Digits						
12	13	14	0	01	1	0
13	15	16	0	01	0	0
14	17	18	0	01	0	0
15	19	1A	0	01	0	0
16	1B	1C	0	01	0	0
17	1D	1E	0	01	0	0
18	13	14	1	01	0	0
19	20	1F	0	01	0	1
1A	13	14	1	01	0	0
1B	13	14	1	01	0	0
1C	13	14	1	01	0	0
1D	13	14	1	01	0	0
1E	13	14	1	01	0	0
1F	28	01	1	01	0	0
Receive Microcode for Special Characters						
20	21	22	0	10	1	0
21	23	24	0	10	0	0
22	25	26	0	10	0	0
23	00	27	0	10	0	1
24	21	22	1	10	0	0
25	21	22	1	10	0	0
26	21	22	1	10	0	0
27	12	28	0	10	0	0
Receive Microcode for String						
28	29	2A	0	10	1	0
29	00	2B	0	10	0	1
2A	29	2A	1	10	0	0
2B	12	20	0	10	0	0

DECODE TABLES. The above tables represent the complete state function for the receive state machine of the four subgroups of the example given in table 1. $d(q_x, 0)$ and $d(q_x, 1)$ represent the two jump addresses to which the current state may advance based on the next input data bit. The actual data output defined by the accepting states can be found from the transition state diagram (fig. 2) corresponding to each address (q_x) and these tables, e.g., locate the first '1-Flag' in the Accepting State column at address 07 which represents an 'O' in the Transition State Diagram of figure 1.

A Relational Database Machine for Efficient Processing of Statistical Queries

Hamid Farsi and John Tartar

Department of Computing Science, University of Alberta
Edmonton, Alberta, Canada

Abstract

This paper presents the design of a data base machine for supporting statistical data bases. The primary objective of this research is to introduce an architecture which performs efficiently in executing relational operations on fully transposed files. This objective is met by utilization of two functionally specialized subsystems: category and summary subsystems. The category subsystem exploits the parallel, content addressed search capabilities of associative memories (AM) while the summary subsystem employs a set of functionally equivalent processors suitable for evaluating statistical functions. In order to support processing of category attributes, the design of an extended associative memory has been considered and its features are presented. The most important feature of this AM stems from its ability to sort selected tuples of a relation, with respect to several attributes simultaneously and at the same time to produce an inverted list for each attribute. Sorting is performed by enumeration. The sorted inverted lists are used for efficient execution of relational projection and join operations.

1. Introduction

Problems associated with the management of large data bases have received the attention of many computer designers in seeking an efficient solution. As a result, a number of design proposals have been appearing, offering solutions from software optimization techniques to harddware implementation of data base management system functions. The variety of solution alternatives have appeared to mark the innovation of a computer architecture for data base applications, known as a Data Base Machine (DBM). A DBM is defined as a dedicated backend computer which manages the data base and performs the requested operations on data with the objective of releasing the mainframe from difficult and time consuming data processing tasks.

Statistical database management systems have different applications from traditional DBMS, although they suffer from the same problems to some extent. There have been few specific DBM design proposals for handling statistical data queries. An example is the design of MAS by Hawthorn [1] which has a limited capability and cannot be considered a true DBM, but rather a multiprocessor complex that handles decompression and assembly of tuples in statistical queries.

The problems and characteristics of statistical data base queries,are introduced in section 2, upon which the design objectives will be based. Section 3 discusses the solutions provided by existing DBM designs and section 4 presents architectural features of the proposed DBM in supporting statistical data operations. A short description of the system architecture follows in section 5. Section 6 presents an implementation of relational operations and an algorithm for a sorting scheme. Finally, section 7 presents some concluding remarks.

2. Query Processing - Characteristics and problems

Execution of a query on a data base mainly consists of three processing phases: data transfer, data qualification and data manipulation. During the data transfer phase, the required raw data are brought into the processors' memory from secondary storage devices. The processor then performs a search operation on the contents of its memory, selecting those tuples that satisfy the selection conditions, thus the qualification phase. In the data manipulation phase, the specified operations (such as updating, aggregate operations, statistical function evaluation, etc.) are executed on the qualified tuples.

Statistical database systems generally are concerned with evaluation of statistical functions on a large set of selected tuples. Thus, each phase of statistical query processing makes a nonnegligible contribution to the overall execution time. Detailed descriptions of statistical data base characteristics can be found in [2] and [3].

Two important characteristics of statistical queries should be mentioned here. In statistical queries normally a small number of attributes are required to be processed. Moreover, attributes of relations, based on their usage pattern, can be classified as category and summary attributes. However, the role of category and summary attributes may change for different queries. This implies that every attribute of a relation can be in either class.

It has become evident that the data transfer phase causes the most serious problem in processing queries on large data bases. This is mainly due to the movement of a considerable amount of redundant data between the processor memory and secondary storage devices. This problem becomes more severe for statistical data bases in which size of relations (tables) are very large in terms of both cardinalities and record size.

In a single processor architecture, the system performance in evaluating complex statistical functions with a large set of numeric data can degrade by becoming CPU-bound, even if I/O overhead is disregarded. Similarly, the execution of a complex operation such as a relational join in the qualification phase requires a considerable amount of CPU resources. These imply that in a sequential processing architecture the data manipulation and qualification phases may indeed degrade the system performance.

3. Some existing solutions to the problems of statistical queries

The major contribution of DBMs in processing data base queries is that of enhancing the system performance through facilities that reduces the time required for the data transfer phase. This has been accomplished by the use of special processors, referred to as filters. Examples of DBMs employing these devices are, DBC[4], SPIRIT-III[5], CAFS[6] and DIALOG[7]. Since the processing speed of filters in these machines match the speed of data transfer, the data transfer and qualification phases are processed concurrently. However, performance of these machines for statistical data bases degrades for two reasons: First, the size of records in relations of statistical data are larger than the size of records in a normalized relation. As a result, there will be fewer records on a track of a disk, causing a fewer number of records to be processed in each disk revolution. Moreover, regardless of the number of attributes required by a query, the entire record must be transferred to the filters. Second, because of the high selectivity factor in statistical data, many records will pass the filters to the processor memory, causing a decline in the filters' potential use and movement of unwanted attributes. In the design of SPIRIT-III, a group of attribute filters is utilized to remove the unnecessary attributes from the selected tuples. The same capability can be provided for in other designs.

A better solution to the problem of data staging is offered by a special physical storage structure, referred to in the literature as fully transposed files, in which attributes of a relation are stored on separate files. A fully transposed file structure facilitates the transfer of only those attributes required by the query, not the entire relation. As a result, the data transfer time as well as processing time will be reduced considerably. An example of a statistical data base system utilizing transposed files is RAPID [2] which is operational at Statistics Canada.

Taking the nature of statistical functions into account and considering the theory of statistics and mathematical set theory, the applicability of multiprocessor parallel architecture for the efficient evaluation of statistical functions is clear. For example, in order to evaluate a statistical function such as variance, covariance, or a regression model in terms of a large set of numerical data, it is possible to partition the given set into several subsets and assign the processing of each subset to an individual processor. The final value of the function can be computed by combining the results produced by all the processors. This property also holds for relational operations. Therefore, the processing time of the data manipulation and qualification phases can be reduced by use of an architecture which utilizes parallel processing elements. Although few data base machine designs have explicitly discussed their application to statistical data bases, multiprocessor architecture has been included in most of the proposed designs.

4. Architectural features

The storage structure used in this design is the fully (completely) transposed file. It is concluded in [2] and [8] that the performance of transposed files declines as the number of attributes required by a query increases. In order to remove this dependency, a group of devices are employed which are capable of searching on different files in parallel while preserving the relationships between the files. In the realization of parallel processing of files (attributes) a storage strategy is used to facilitate the parallel transfer of data from storage devices to these processors. This requires the partitioning of attributes into several equal size files, each stored on a separate device. e.g. a track of a cylinder in magnetic disk units. The detail description of the storage layout is given in section 5.

The processing time of the qualification phase depends on the type of relational operations involved. It ranges from a simple selection operation on a single relation to a m-attribute join operation on (m+1) relations. Join is the most important and difficult operation that should be efficiently handled in a design. It is obvious that the best possible method of implementing a join between two or more relations is by means of a sort-merge which works essentially on presorted lists of records.

The effectiveness of inverted files for the efficient access of data in a large data base is well understood. But because of the large number of tuples in relations, the versatility of data and the uncertainty in usage pattern of attributes, the process of producing, sorting and maintaining inverted files is very time consuming. These facts detract from their general usefullness and can contribute to an overall degradation in system performance.

This architectural design introduces a parallel algorithm which generates run-time inverted lists for one or more attributes of selected tuples. With this feature, it is possible to join two or more relations in O(C/N) time complexity, where C is cardinality of the relations and N is the number of processors. The benefit of inverted lists then can be realized in the implementation of complex relational operations and in the evaluation of statistical functions in data qualification and manipulation phases respectively. A set of special-purpose parallel processors are utilized in order to process these two phases of the query processing efficiently. A description of this architecture is provided in section 5. For convenience, new names have been adopted for the components of the system architecture which are self explanatory in the context of statistical data base characteristics.

5. System architecture

The overall architecture of this system is composed of three functionally specialized subsystems: the storage subsystem, the category subsystem and the summary subsystem. Figure 1 shows the conceptual system architecture of this design. As the names imply, each subsystem is devoted to process a class of attributes with different characteristics.

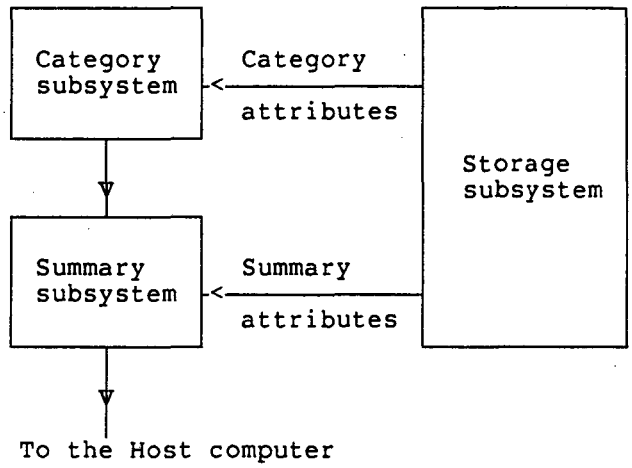


Figure 1 – Block diagram of the system architecture.

5.1. Storage subsystem

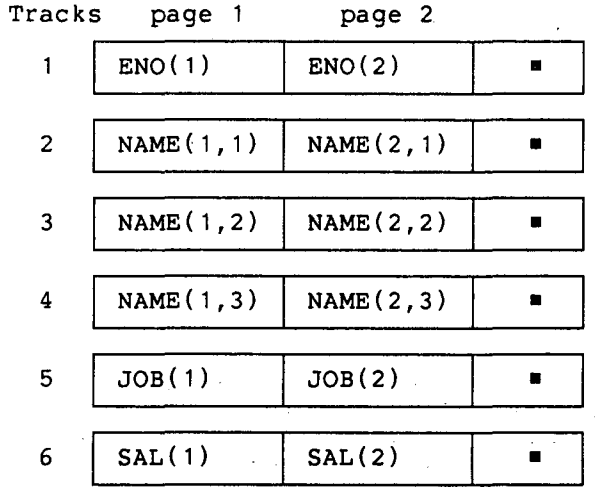
Moving head disks with parallel read write capabilities are employed as secondary storage devices. Relations of the data base are stored on tracks of the disks as fully transposed files. The tracks are partitioned into equal size pages, each containing an attribute or part of an attribute. Figure 2 shows the partitioning of relation EMPLOYEE into equal size pages and storage layout of the attributes of this relation on tracks of a cylinder. With this storage layout

the data staging time is minimized since only those attributes required by the query are staged into the memory subsystems. Maximum parallelism can be achieved in response to a data staging request by the device controller's parallel transmission of as many pages as can be handled by the other subsystems.

ENO (1)	NAME (1,1)	(1,2)	(1,3)	JOB (1)	SAL (1)
100	Smit	h,Jo	seph	SEC¤	10000
101	Jone	s,Su	san¤	MGR¤	30000
102	Brow	n,Ka	thy¤	ENG¤	22000
103	Smit	h,Jo	hn¤¤	MGR¤	30000
104	Smit	h,Pe	ter¤	TECH	20000
105	Whit	e,Ha	rold	MGR¤	25000

ENO (2)	NAME (2,1)	(2,2)	(2,3)	JOB (2)	SAL (2)
106	Gray	,Tom	¤¤¤	SEC¤	10000
107	Hall	,Ted	¤¤¤¤	TECH	15000
108	Gree	n,Mi	ke¤¤	SEC¤	10000
109	Brow	n,St	eve¤	MGR¤	35000
110	Thom	as,L	isa¤	MGR¤	32000
111	Grov	e,Pa	ul¤¤	SEC¤	10000

(a)



(b)

Figure 2 – Representation of a partitioned transposed file on tracks of a magnetic disk unit.(a) Partitioning of relation EMPLOYEE into equal size pages.(b) Storage layout of pages on tracks of a cylinder.

5.2. Category subsystem

This is the most central subsystem of the proposed design. The primary objective of the design of this system is to extend the parallel search capabilities of content-addressable (associative) memories in the efficient processing of relational operations on transposed files. The basic function of this subsystem is to flag those tuples that have satisfied the query conditions and to generate addresses associated with the flagged tuples. Based on its functional characteristics, this system can be veiwed as a three-level hierarchy of components, shown in Figure 3.

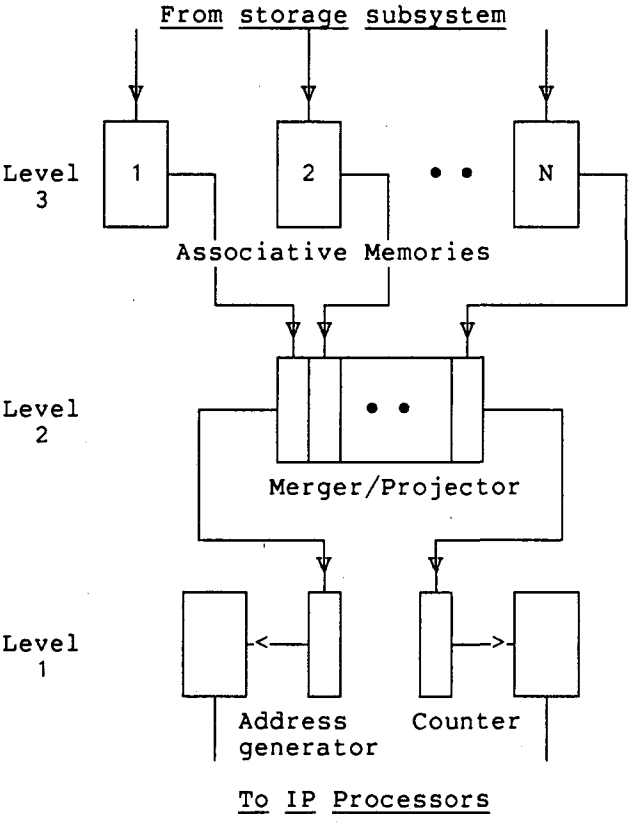


Figure 3 – Logical view of category subsystem.

On level 3 of the hierarchy there are N distributed-logic associative memories, each with a comparand, a mask and two response registers x and y. The structure of the associative memories is similar to the memories designed in [9] with the exception that the capabilities of the memories employed here are limited to equality and threshold searches only. The hardware implementation of equality-threshold (=,<,>>) functions in an associative memory involves no complex logic circuitry. Moreover, the resulting signals need to propagate in one direction only, along the bits in a word, making the words functionally independent from one another. Consequently, the length of memories can be expanded easily to accommodate larger sets of data.

On Level 2 of the hierarchy there is the PROJECTOR/MERGER that processes the responses generated by the associative memories of the level below. This component is an associative memory with equality search capability. Each bit column of this memory corresponds to an associative memory on Level 3. Its function is to project the results of search on associative memories and to maintain these results for future decisions so that previously selected tuples do not participate in subsequent searches. Its major function however, is to merge (combine) the responses generated by AMs to produce the final results which satisfy the query expression in its entirety.

On Level 1 of the hierarchy there are two components: a parallel counter and an address generator. The parallel counter is constructed from a set of full adders, and is capable of counting the number of responses in an array of size n in $O(\log_2 n)$ gate delay times. Detailed discussion for this device can be found in [10]. The task of the address generator is to generate quickly addresses of the tuples satisfying the query conditions and to transfer these addresses to the summary subsystem. This component resembles the hardware implementation of a heap tree which is capable of generating addresses of K responses in a memory of size n in O(log n+K) gate delays.

5.3. Summary subsystem

The primary function of this system is to perform the required statistical functions on summary attributes of the qualified tuples. A set of functionally equivalent parallel processors called Post-Processors (PP) are used for this purpose. Before PPs apply the specified functions on selected tuples, the data produced are collected and partitioned into equal size, disjoint subsets. Data can be addresses, attribute values or both. Figure 4 shows the block diagram of this system.

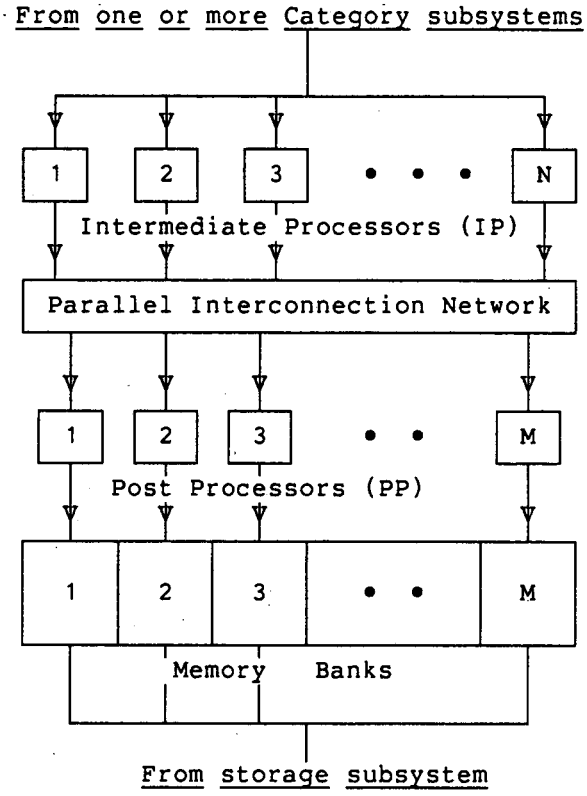


Figure 4 – Block diagram of summary subsystem.

The process of collecting and grouping the data is carried out by a set of processors called Intermediate Processors (IP). The purpose of partitioning addresses into disjoint subsets is to minimize the access conflicts to pages of summary attributes. The purpose of attribute partitioning is to distribute the work equally among postprocessors. The PPs have access to a common memory bank which contains the summary attributes. A memory controller is in charge of scheduling access requests and resolving any conflicts, using a predefined priority scheme.

6. Relational Algebra Processing

In most statistical data base queries apart from statistical functions and analysis, the process of selecting qualified tuples involves the use of the same set of operations as in relational data bases. In this section we describe the implementation of relational algebra operations (i.e. selection, projection and join). Because parallel sorting has been used as a basic building block in the design of algorithms, we describe the sorting algorithm first. The following notations are used in illustration of the algorithms.

Notation

N : Number of Associative Memories and Intermediate Processors.

M : Number of Post Processors.

K : Length of associative memories in number of words, as well as the length of a page of an attribute.

w : Length of a word in array memory of each associative memory.

C_s : Cardinality of relation S.

|S.s| : Average length of a data item in attribute s of relation S.

nsa, nca : Number of summary and category attributes specified in a query.

p_s : Maximum number of pages of category attributes that can be staged into the associative memories and processed in parallel, i.e. p_s=N/nca. This set of attributes will be referred to as *segments*. A segment therefore occupies (|S.s.|/w)*nca associative memories.

d_s : Number of unique tuples in a projected relation.

L_s : Number of loads required to process the entire set of files containing the category attributes, i.e. $L_s=C_s/(K*p_s)$.

A relation is divided into equal size segments, each segment contains K tuples, except possibly the last one. A segment is spread over as many files as there are attributes in the relation. A page of a transposed file may be stored on several tracks of a cylinder as shown in Figure 2. Without loss of generality, it is assumed that |S.s|<=w. That is, a page on a track contains a page of an attribute in its entirety and moreover, a page of an attribute occupies exactly one AM.

Parallel sorting algorithm

The sort algorithm is divided into three stages which can be processed in a pipeline manner. In the first stage, the category subsystem with the help of IP processors sorts contents of the associative memories by enumeration. This process is repeated until all the records have been staged into the associative memories and sorted. At the end of this stage, each IP processor will have L_s lists whose elements are unique and sorted within each list.

In the second stage, each IP processor performs a binary sort-merge on its sublists. The controller, based on information provided by IP processors, determines a group of disjoint subintervals and assigns a postprocessor to each subinterval. Each IP in turn divides its sorted list into M groups, belonging to a different interval and transfers each group to the corresponding post processors.

In the third stage, a post processor receives N sorted lists, one from each IP and performs a binary sort-merge on these lists. As a result, each PP will have in its possession a sorted inverted list of length d_s/M whose elements are greater than the elements in its left hand side processors and less than those in its right hand side processors. Second and third stages are clear and need no explanation. The algorithm for the first stage is provided below:

To illustrate the algorithm consider the relation S with n attributes $s_1,s_2,\ldots,s_n$ to be sorted with respect to each of the attributes $s_1,s_2,\ldots,s_m$ (m<=n) while their association is to be represented by inverted lists. The data items $s_{i,1},s_{i,2},\ldots,s_{i,m}$ are referred to as the values of attributes $s_1,s_2,\ldots,s_m$ in the ith tuple of the segment in the AMs, as well as the content of the ith word in the associative memories holding the attributes $s_1,s_2,\ldots,s_m$.

For simplicity, assune that the attribute s_j is stored on the jth AM. $c_{i,j}$ is also referred to as the rank of $s_{i,j}$ among all the words in the jth AM. Each segment of relation $S'(S.s_i,$ i=1,2,...,m) in the AMs is associated with an IP processor which receives the data and acts accordingly. Since each segment in an AM is processed indepen-

dently, for simplicity throughout the algorithm we consider one segment and its associated IP processor.

Algorithm - Sorting a relation on m attributes:

Step 1 - Load associative memories with one page of the attributes $s_1, s_2, \ldots$ and s_m.
Let i = 1

Step 2 - Move the contents of word i into the comparand registers, i.e. $s_{i,1}, s_{i,2}, \ldots, s_{i,m}$ forms the content of comparand registers associated with $AM_1, AM_2, \ldots, AM_m$, respectively.

Step 3 - Perform a search on these AMs. As a result, response bits $x_j y_j$ (j=1,2,...,k) will be in one of 00, 01 or 10 states, indicating that the content of word j is greater than, equal to, or less than the argument in the comparand register.

Step 4 - Employ the parallel counter to enumerate number of "10" responses (less than comparand value) in each AM. For this case, m numbers $c_{i,1}, c_{i,2}, \ldots, c_{i,m}$ are generated.

Step 5 - Transfer these data to the corresponding IP. The record which is transmitted has the following format: $(s_{i,1}, \ldots, s_{i,m}: c_{i,1}, \ldots, c_{i,m})$.

Step 6 - Let i = i + 1, if i < K then proceed to Step 2, else proceed to Step 7.

Step 7 - If there are more pages to be processed, proceed to Step 1, else STOP.

While the category system executes this algorithm, the corresponding IP processor rearranges the attributes and produces the inverted lists as follows: For each segment of attributes in AMs, it uses m data arrays $A_1, A_2, \ldots, A_m$ with the same length K as the AMs, and m mark-bit arrays $MA_1, MA_2, \ldots, MA_m$ also the same length K. Each element i of these data arrays has a field pointing to a bucket. This bucket is used for storing the addresses of tuples associated with the key i. The mark-bit arrays are used to mark and later locate the elements of the data array which contain a key value.

Step 1 - Allocate a new set of arrays $A_1, A_2, \ldots, A_m$.

Step 2 - Receive from category system $(s_{i,1}, \ldots, s_{i,m}: c_{i,1}, \ldots, c_{i,m})$.

Step 3 - Store the key values in their corresponding data arrays at proper locations and the tuple IDs in the associated bucket.

```
For j = 1 to m do
  begin
  MAj(ci,j) = 1
  Aj(ci,j) = si,j
  bucket(ci,j) = bucket(ci,j) U i
  end
```

Step 4 - If there are more tuples to be processed, proceed to Step 2, else proceed to Step 5.

Step 5 - Arrange the inverted lists - To do this, the mark-bit arrays MA are used to locate the keys in the data arrays and to eliminate any existing empty locations.

Step 6 - If there are more pages to be processed, proceed to Step 1, else STOP.

Figure 5 shows the implementation of this algorithm on a single column of data.

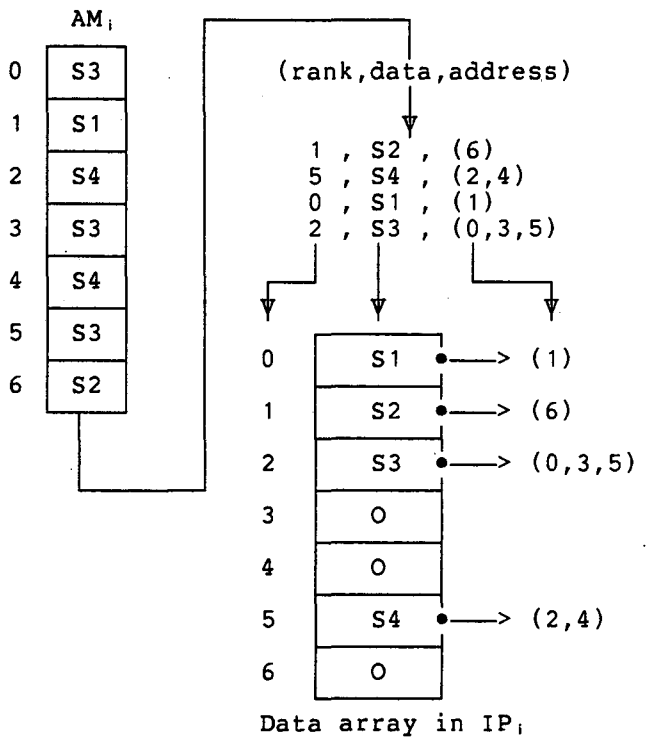


Figure 5 — Implementation of the sorting algorithm.

In order to derive the order of time complexity of the sort algorithm relation S is considered to be sorted on its prime key attribute, i.e. m=1 and all the values in the attribute are distinct. Let S_i be the time complexity of the ith stage. The time of the first stage is equal to L_s times the time of data staging plus the time of sorting N pages L_s times. Thus, $S_1=O(C_s/N+C_s/N)$. Stage 2 is a binary merge on L_s lists, each K words long. Therefore, $S_2=O((C_s/N)*\log_2 L_s)$. Similarly, stage 3 is a binary merge on N lists, each $C_s/(N*M)$ words long. Thus, $S_3=O((C_s/M)*\log_2 N)$. Therefore, the overall time complexity of the sorting algorithm is :
$O(C_s/N + (C_s/N)*\log_2 L_s + (C_s/M)*\log_2 N)$.

Selection operation

Selection is the simplest but very important relational algebra operation. Selection conditions are normally presented in a disjunction or conjunction normal form (DNF,CNF). An expression in DNF can be evaluated for all the $K*p_s$ tuples residing in AMs simultaneously. The processing cost of the selection operation in terms of the number of comparisons, is $O(L_s)$. For the best case where nca=1, the required number of comparisons is $C_s/(N*K)$ and for the worst case where nca=N, the number of comparisons would be C_s/K. In either case, data can be searched faster than the storage system can provide them. However, due to the distinct storage organization, the effective data transfer rate of the storage system is higher than those DBMs employing tuple oriented schemes. Thus, this system provides a considerably better response time which is an advantage over other data base machine designs.

Projection operation

Since fully transposed files are the basic storage units in this design, projection of a relation on specified attributes already exists. However, in order to eliminate duplicate tuples among these attributes, the sorting algorithm is employed. Therefore, the time complexity of this operation is the same as the sorting algorithm except that, the term C_s is to be replaced by d_s, number of unique tuples in the projected relation. The cost of data staging will remain the same.

Join operation

As in projection, the sorting algorithm is used to perform a join between two or more relations. To join relations R(a,b) and S(b,c) on attribute b, the sorting algorithm generates an inverted list for each attributes R.b and S.b. These lists are then distributed over the PPs. Postprocessors merge the lists and with the occurrence of a hit (e.g. equality), addresses in associated buckets give the logical addresses of the tuples in R and S relations. These addresses then are translated to the physical location addresses in the memory bank where the summary attributes are residing.

If any statistical function is to be evaluated on summary data, the PPs will carry it out and transfer the result to the user. The time complexity of executing a join operation in this system is equal to the time of sorting both relations, one at a time, plus the time of parallel merging of two inverted lists. This time is proportional to the cardinalities of the relations involved, divided by M.

7. Summary and conclusions

The processing steps involved in executing statistical queries have been specified. Based on these specifications and the characteristics of SDBs, the design of a DBM architecture is proposed which provides solutions to some of the problems of statistical query processing. Since the data staging phase of a query execution is the main factor in degradation of system performance, the objective of this design is to provide for efficient performance in executing relational operations on transposed files. This objective is met by distributing the work over two functionally specialized subsystems, each of which is responsible for executing the required operations on a different class of attributes, i.e. category and summary attributes. A partitioned AM is used as the main component of the category system. Conceptually, the associative memory can be viewed as a hierarchically structured associative memory with equality-threshold search capabilities. Motives behind the design of this AM can be seen in the following statements of merits :

1. In processing fully transposed files, the relationship between attributes of tuples must be kept properly. Because of the tabular structure of AMs, they are suitable devices for serving this purpose.

2. Parallel content-addressed search capabilities of AMs will be enhanced, By partitioning AMs into equal, smaller, and

independent AMs, it is possible to carry out search operations on different category attributes specified in one or more queries. Thus, a faster search can be achieved.

3. In executing a selection operation on a single relation, regardless of the number and types of clauses in the query expression, responses can be produced with one comparison.

4. The most important feature of this component is its ability to sort the selected tuples with respect to several attributes simultaneously and at the same time produce an inverted list for each attribute. Sorting is performed by enumer- ation, i.e. the number of keys less than a particular key in an array specifies the location of that key in the array, had it been sorted. The sorted inverted lists are used for implementation of com- plex relational operations, projection and join.

A considerable improvement in processing statistical queries can be achieved by means of functionally specialized subsytems. Distribution of statistical query phases over these subsystems enhances the system performance both in terms of response time and throughput. This improvement is due to the following:
First, because of functionally specialized subsystems, parallel pipeline processing is achieved.
Second, the problem of data movement between system components is alleviated to the extent that only required category attributes will be staged into the category subsystem and only qualified summary attributes are brought into the summary subsystem.
Third, because of implicit representation of qualified tuples by means of inverted lists and addresses, data exchange between processors causes no communication difficulty.

References:

[1] Hawthorn, P. "Microprocessor Assisted Tuple Access, Decompression and Assembly for Statistical Database Systems," Proceedings, Eighth Int. Conf. on VLDB, Sept. 1982, pp. 223-233.

[2] Turner, M. J., R. Hammond and P. Cotton, "A DBMS for Large Statistical Databases," Fifth Int. Conf. on VLDB, Oct. 1979, pp. 319-327.

[3] Shoshani, A. "Statistical Databases: characteristics, problems and some solutions," Proceedings, Eighth Int. Conf. on VLDB, Sept. 1982, pp. 208-222.

[4] Banerjee, J., D.K. Hsiao, and K. Kannan. "DBC - A Data Base Computer for Very Large Databases," IEEE Trans. on Computers, vol. C-28, NO. 6, June 1979, pp. 414-429.

[5] Kamibayashi, N. and K. Seo, "SPIRIT-III: an advanced relational data base machine introducing a novel data-staging architecture with Tuple Stream Filters to preprocess relational algebra," AFIPS Conf. Proc., 1982, pp. 605-616.

[6] Babb, E. "Implementing a Relational Data Base by Means of a Specialized Hardware," ACM Trans. on Data Base Systems 4, NO. 1, March 1979, pp. 1-29.

[7] Wah, B. W., and S. Bing Yao. "DIALOG-a Distributed Processor Organization for Database Machine," AFIPS Conf. Proc., vol. 49, 1980, pp. 243-253.

[8] Batory, D. S. "On Searching Transposed Files," ACM Trans. on Database Systems, Vol. 4, No. 4, Dec. 1979, pp. 531-544.

[9] Ramamoorthy, C. V., J. L. Turner and B. W. Wah, "A Design of a Fast Cellular Associative Memory for Ordered Retrieval," IEEE Trans. on Computers, Vol. c-27, pp. 800-815, Sept. 1978.

[10] Foster C. C. and F. D. Stockton, "Counting Responders in an Associative Memory," IEEE Trans. on Computers, Vol. c-20, pp. 1580-1583, Dec. 1971.

SIBYL: An Economist's Workbench

Sandra I. Heiler(+) and Rita F. Bergman(-)

(+) The World Bank, 1818 "H" Street, NW, Washington, DC 20433.
(-) Computer Corporation of America, 1600 Wilson Boulevard, Arlington, VA 22209

ABSTRACT

This paper presents a report on work in progress on the development of the SIBYL system, an economist's workbench at the World Bank. The principle objectives are to provide capabilities for sharing data among a large and diverse user community, handling a growing number of databases, and providing access to a collection of analytical and modeling packages.

The implementation is oriented towards using off-the-shelf software, a large mainframe with Model 204 DBMS to handle the databases, and a group of microcomputers running UNIX. The key components integrating this architecture are a database template, a set of generalized procedures for browsing, updating and extracting data, and translators between the databases and the analytical and reportwriting packages.

The project has been in progress since May 1982 and a prototype system is installed for three user groups. It will continue to evolve over the next year with additional development focused on providing a batch update facility, writing validation procedures, refining the user interface, and establishing a local network.

INTRODUCTION

SIBYL is a specialized computing system dedicated to satisfying the needs of economists for managing economic and social data - specifically timeseries - and for analyzing, projecting and presenting them. The system is currently under develoment by the World Bank, supported by consultants from Computer Corporation of America, System House International, and several other companies.

The concept of a workbench derives from the idea of providing a suitable environment for access to materials, and carefully selected tools for economists similar to those available to other professionals, e.g., the carpenter's workbench or the chemist's laboratory. The successful implementation of the programmer's workbench (UNIX PWB)[1] demonstrated the feasibility of providing the needed facilities to support programming professionals through an integrated computing system. The purpose of the SIBYL development is to provide a similarly friendly and complete set of facilities for economists.

The World Bank environment places several specific requirements on any system to support economic work. First, it must be capable of supporting several hundred economists and researchers who have varying levels of motivation and expertise in using computing systems, and whose problems present varying levels of complexity. Second, the Bank's databases are large, numerous, and diverse; their structure will change over time and their number will increase[2].

The functions of the SIBYL system are to support maintenance and analysis of timeseries data, primarily social and economic indicators, and trade, commodities, and international debt statistics. The system provides a central facility for sharing official Bank data and distributed facilities for analysis and presentation of the data. It supports creation and management of timeseries databases, statistical analysis, econometric modeling, reportwriting and graphic presentation.

This report describes work in progress on the system. We believe that it embodies an uncommon but effective approach to storing and processing

timeseries data and an innovative use of microcomputer technology.

SYSTEM FEATURES

The implementation of the system in providing for management and analysis of timeseries data differs from most other approaches in three important ways: First, it uses a general-purpose commercial DBMS, Model 204[3], for management of its databases. Second, it provides a loosely integrated collection of individual packages, a "tool-box" of facilities for statistical analysis, modeling, reportwriting and graphic presentation, rather than a single monolithic system to support all of these functions. Not every user necessarily will have or use the same tools. Third, it uses microcomputer workstations running UNIX to perform statistical analysis, modeling and reportwriting functions. The microcomputers are linked to a mainframe where centralized databases reside.

This approach has both good and bad points. The main benefit of using a general purpose DBMS is that it allows us to provide a full set of data management functions (including joins) with no time investment in data management software development. Also, because the function of data administration was new in the Bank when the project started, there was little knowledge of the available data and the relationships among them and no broad, long-range view of the functions that needed to be performed. Using a DBMS allows us to put up databases quickly, to modify their structure where necessary, to add new databases easily, and to combine data from disparate databases in an ad hoc fashion.

There is an obvious disadvantage to using a DBMS for timeseries. While its storage and access mechanisms are reasonably efficient for large and numerous databases, they cannot be expected to be as efficient as those storage and retrieval mechanisms that are specifically designed for timeseries[4]. However, the inherent flexibility of the DBMS in responding to changes in the number and structure of the databases as well as the variety of the functions to be performed on them, offsets the higher cost of processing and storage.

The primary benefits of the tool-box approach using off-the-shelf software are that it minimizes software development time by using existing software, except for the interfaces between packages, and it encourages incremental system releases instead of a single release at a much later date. It also provides flexibility for continuously adding new tools as new functions are required or as new products become available. In addition, the tools are often more powerful, more task specific, or more appropriate to a particular user's expertise than could be provided for by a comprehensive system. In this way, users can be selective in the portions of the system that they learn and use.

The disadvantages of such an approach are that many of the packages use different languages and some complex functions may require the use of multiple tools, requiring the user to pass from one language environment to another. To minimize this we have tried to select tools with overlapping functions so that simple tools are available for simple jobs with more sophisticated tools for complex jobs, but we have not completely eliminated the problem.

COMPONENTS OF THE SYSTEM

Each of the major components of the system is described briefly below. Some are included in further detail in the hardware and software architecture descriptions given later.

The Databases

The databases include various types of timeseries data, along with definitions, supporting information (such as code conversion tables and source and units information), and explanatory footnotes that may apply to any level of data in the database. The system supports official databases (which are only

updated on a fixed schedule after careful validation of the updates), working versions of the official databases, (which are continually updated), personal databases, and extracts from the official databases (whose contents are frozen for analysis or for the generation of Bank publications). The security mechanisms of Model 204 are used to control the status of each database. Many of the databases are relatively large, e.g. 60 million characters. The final number of databases to be supported is expected to be several hundred.

A Pro Forma Template

The pro forma template is a guide for mapping the logical structures of a timeseries database into suitable Model 204 structures. It is needed for two reasons: 1) to establish efficient structures within Model 204 to be used for handling each of the peculiarities of the data, specifically timeseries and footnotes; and 2) to provide some measure of consistency among disparate databases so that a standard set of access procedures can be prepared.

The template accommodates data that are peculiar in several respects. Each database contains not only timeseries data but definitions, supporting information, and footnotes that may apply to any data in the database. The data may contain timeseries of any periodicity, including usual ones (such as fiscal year, annual) and uneven ones. Multiple periodicites may occur within a single database or file. (For example, weekly, monthly, and annual versions of currency conversion rates are maintained). If the data are thought of as multi-dimensional arrays, they are sparse with respect to all dimensions except time. (For example, many economic indicators that are available for the developed countries are not available for developing countries).

Access Procedures

A set of screen-based procedures for browsing, updating, and extracting is provided for each database. Users fill in the screens with their parameters for the particular functions they require and the procedures generate Model 204 requests to perform the functions.

While the overall functions and basic screen layouts are common to all databases, the screens and procedures must be database-specific because each has its own unique fields and structural details. These procedures are generated for each database, because the number of databases makes it infeasible to hard-code procedures for each one. The data needed for generating the procedures are established by the template and stored in the database it describes.

Analytical/Reportwriting Tools

The system includes packages for statistical exploration, econometric modeling, spreadsheet analysis, reportwriting and graphics. Most of the packages are commerically available, off-the-shelf products but some are Bank-produced systems that solve problems specific to the Bank. Many of these packages have overlapping functions, to allow users to choose packages based on their specificity to a particular problem or the user's familiarity with them. This also means that a user can apply very simple tools to simple problems, reserving the need to upgrade to more sophisticated tools for complex problems. The system also includes standard utility software.

Translators

Because SIBYL uses off-the-shelf

packages that process data in formats that are different for each package and different from the databases, translators must be present for each package to convert data from the database format to the package format. Packages that produce data that need to be stored in the database (e.g. modeling packages, which produce projections, but not reportwriters) also require reverse translators. To support translation for packages on the microcomputers, a standard database format is provided on the microcomputers under UNIX. The extract procedures place data in this format when the intent is to process the extracted data with a package that runs on the micros.

HARDWARE ARCHITECTURE

SIBYL is implemented on the central IBM computing facility at the World Bank which was upgraded from an IBM 3031 to a 3083 in April, 1983. The operating system is VM, under which CMS and OS/VS1 run. Because the IBM center is run on a cost-recovery basis, users are obligated to pay for their processing and storage costs.

In addition to the IBM mainframe, SIBYL uses Codata M68000-based workstations that reside within the environment of each of the user groups. Each Codata includes 1 million bytes of memory, a 5.25 inch floppy disk, an 84 million byte hard disk, and access to a cartridge or 9-track tape drive. Experience to date indicates that each configuration can support about five or six simultaneous users, depending on which packages are running. Printers may be shared or dedicated to a terminal. The workstations are purchased outright by each user group, and continuing costs are for maintenance only.

The workstation/mainframe approach was chosen because each type of hardware provides benefits that cannot be attained from the other.

- The workstations provide low-cost means of processing some types of requests, without incurring the operating and storage costs of using the mainframe. The recurring costs associated with running the workstation are limited almost entirely to maintenance. They can provide the needed speed, power, memory and disk space for many analytical, report generation and graphics tasks. In addition, the workstations provide users with a level of control over their computing environments that is not available to users of the mainframe; with a workstation, users control the available hardware configuration, the load level on the machine, and the prioritization of jobs.

- The mainframe, on the other hand, is capable of providing the high speed, power, amounts of memory and disk space that are required for very long or complex jobs or for storing and managing large databases. In addition, many software packages were designed solely to be run on mainframe computers and cannot cost-effectively be ported to workstation hardware. Finally, the mainframe is the most logical place to store shared databases because it is accessible by all users.

The workstations are linked to the mainframe through dedicated phone lines (9600 baud) and dial-up lines (1200 or 4800 baud). Direct hardwiring is difficult because the Bank is situated in several buildings in several different city blocks. Since the workstation does not resemble a standard peripheral device for the IBM software (3270 terminal), the workstations are connected through a 3270 emulator.

SOFTWARE ARCHITECTURE

The main components of the software are Model 204, the DBMS that runs on the mainframe; UNIX, the operating system on the microcomputers; the analytical packages; the translators; and

the template.

Model 204 was selected to manage the centralized databases following a benchmark of several DBMS's for mainframe computers. The main reasons for its selection were that it provides the needed flexibility for continuously adding new databases or changing the structure of existing ones; it supports ad hoc combining of data from different files; and it is efficient for large and numerous databases and large numbers of concurrent users. Model 204 is also used for other major database applications in the Bank.

UNIX was selected as the operating system for the microcomputers because of its increasing use on a wide-range of machines, the availability of a number of analytical packages which already run under it, and the breadth and ease-of-use of its utilities. It is flexible enough to support many different kinds of end-user functions, ranging from editing and basic wordprocessing to sophisticated statistical applications. In addition, it is a powerful system that supports system development well. It provides a large number of commands, and the command language itself is expandable. Simple commands may be readily combined to perform more complex functions. Finally, because UNIX is a somewhat standardized system, software that runs under UNIX is usually portable from one hardware system to another. As other suitable microcomputers enter the market, the software can be easily transferred to a new environment. This frees SIBYL from being tied to a particular vendor, or even a particular processor or architecture.

Several analytical and modeling packages comprise the SIBYL tool box. These packages run on either the IBM or the Codata depending on which is the more appropriate environment. The packages include:

- HANDE - A Bank system that assists in repetitive calculations and is particularly useful for production of Bank-standard tables.
- SAS - A statistical package developed by the SAS Institute.
- SIM2 - A Bank system for simple models.
- TROLL - An econometric modeling package developed by the MIT Center for Computation Research in Economics and Management Science.
- MULTIPLAN - An electronic spreadsheet from Microsoft.
- AMP - A Bank-developed system that includes a modeling package and a reportwriter that is especially suited to timeseries.
- S - A system for exploratory statistical analysis developed by Bell Labs.

Translators have been built to convert from the database structure to each of the packages formats, and vice versa as appropriate.

Initially, no translators were to be provided between packages. This meant that to pass data from one package to another, it had to be re-stored in the database format and then translated to the second package. We did this to limit the number of translators that had to be written. However, since most packages do not handle supporting information or footnotes, the process of translation to the packages involves removing information as well as reformatting. We have therefore established a second standard format for translating to the packages which does not include supporting information or footnotes. The most obvious choice for the second standard format was the spreadsheet, and translators between the spreadsheet and several other packages are under development.

The pro forma template provides a general format for all timeseries databases within Model 204. The template consists of a common set of characteristics for data content and structure with an associated set of metadata. The metadata facilitates the automatic generation of database-specific procedures so that new procedures can be produced whenever a new database is designed and loaded.

All data for a particular database - timeseries, footnotes, conversion tables, other support information and the database description (field names, datatypes, etc.) - are stored in a single Model 204 file, using different record types. Within this file, a common set of index fields (or keys) is used. Each record type is associated with a particular combination of index fields. For example, in a database where timeseries are identified by country, indicator, and source, record-types are provided for country data, indicator data, country/indicator data, country/source data, etc. The template specifically identifies index fields, informational fields, footnotes, periodicity, user-defined period names, and period indices.

Index and data fields are named and described by the database designer. Other fields have standard names and descriptions. The system distinguishes footnotes from user-defined data fields (which may contain textual information) because footnotes must be recognized and treated specially by computational programs (which inform the user of their presence) and report generators (which automatically print them).

STATUS/PLANS

The project has been under development since May 1982, and the prototype system was released to an initial group of users in early October. There are currently three distinct user groups in different divisions in the Bank. Since the release of the prototype, new information has been learned about the databases and user behavior that was not evident at the outset of the project. Our experience thus far indicates that:

- The template is flexible enough to handle specifications of virtually all timeseries databases of the types found in the World Bank. In addition, storage and processing of databases designed under the template are reasonably efficient.

- The programs needed to generate database-specific procedures are sophisticated and complex. Although it is feasible and effective to generate these procedures, it requires more effort than was previously assumed.

- Interactive data entry and updating on the mainframe are considered to be expensive by the users for large amounts of data. The generated procedures must also provide for delayed batch updating and must be prefaced by an inexpensive facility for data entry. Transaction validation and modification prior to initiating the batch job must also be present to protect the integrity of the database.

- Extracts from the Model 204 databases and working files on the Codata for batch updates present a data management problem that is separate from the management of the databases themselves. The operating environments on the IBM and UNIX on the Codata do not provide sufficient tools to handle this easily.

The next phase of the CSEW project will focus on three areas of development. The first is providing a reliable facility for manipulating extracts on the workstation, and sending batch updates to the mainframe. Our initial evaluation has resulted in the selection of Ingres[5] to run on the Codatas. The facilities will minimize connect time on the IBM, provide a localized capability for data manipulation, including arithmetic and Boolean techniques, and provide an audit trail which can be used to generate transactions for batch updates. Several concerns are raised with using a DBMS on the workstation, such as performance, particularly with respect to the multiple users, file sizes, etc; handling duplicate data on the IBM and Codata; and the impact on package translators. These issues, as well as others, are currently under investigation.

A second area of development will be on the implementation of a local network to improve the speed and reliability of communications. In addition to improving data transfer and back-up, the network would offer the option for load balancing among user sites.

Lastly, we plan to devote some effort to improving the user interface. The current menu-based system is adequate for new or infrequent users, but it becomes inconvenient and tedious for experienced users. It is important to provide a mechanism for moving quickly through the functions, either through some global command language, or a more natural query formulation process. Access to a data dictionary should provide a facility for identifying the database contents. The level of detail to be included in the data dictionary is still under discussion because the relevant user requirements are not completely defined.

1. Dolotta, T. A. and Mashey, J. R., An Introduction to the Programmer's Workbench, Bell Laboratories Technical Report. UNIX is a trademark of Bell Labs.

2. Johansson, J. H. and Shilling, J. D., "Toward the Development of an Integrated Economic Database at the World Bank." Proceedings of th First LBL Workshop on Statistical Database Management, December 2-4, 1981.

3. Model 204 is a database management system for IBM mainframe computers. It is a product of Computer Corporation of America.

4. Eggers, S. J. Olken, F., and Shoshani, A., A Compression Technique for Large Statistical Databases; and Eggers, S. J. and Shoshani, A., Efficient Access of Compressed Data, Proceedings of the Sixth Conference on Very Large Databases, 1981.

5. Ingres is a relational DBMS which is available for M68000-based microcomputers through Relational Technology, Inc.

3. Connecting Hetrogeneous Systems and Data Sources

See Also. . . .

ALDS PROJECT: MOTIVATION, STATISTICAL DATABASE MANAGEMENT ISSUES, PERSPECTIVES, AND DIRECTIONS*

James J. Thomas and David L. Hall
Pacific Northwest Laboratories
Richland, Washington 99352

ABSTRACT

The Analysis of Large Data Sets (ALDS) project was initiated at the Pacific Northwest Laboratories in 1978 through funding from DOE/BES Applied Mathematical Sciences. At that time, it was evident that the technical community's ability to collect scientific data was far outstripping existing capabilites to manipulate, display, and analyze such large data sets. Therefore, a new research direction in the analysis of large data sets was established. The ALDS project is composed of a team of statisticians and computer scientists. Their close interaction has been a key factor in contributing to our discoveries and future directions for analyzing large data sets. To help guide this program, an interdisciplinary team of consultants and reviewers were gathered together on an periodic basis to review progress and suggest research directions.

This paper will discuss the motivation and initial goals of the ALDS project, the impact of large data sets, the data management issues addressed by ALDS, current research tasks and their impact on statistical data base management, and perspectives.

1.0 MOTIVATION & INITIAL GOALS

The Analysis of Large Data Sets (ALDS) project was conceived jointly by Pacific Northwest Laboratory (PNL) and the DOE/BES Applied Mathematical Sciences Research Program in 1978. At that time it was evident that the technical community's ability to collect scientific data was far outstripping existing capabilities to manipulate, display, and analyze such large data sets. It was also recognized that a major component in solving tommorrow's energy problems was the development of this analysis capability. Therefore, a new research direction in the analysis of large data sets was established within the DOE/BES Applied Mathematical Sciences Research Program with PNL as the lead contributor. This section discusses the overall problem of analyzing and managing large quantities of data and discusses some of the ALDS research.

Large data sets have existed as long as researchers have been collecting data. At times data has been successfully analyzed with dramatic results. A familiar case is the nationwide retrospective study using experimental data and vital statistics records on the effects of smoking. The study concluded that those who smoke excessively have an increased chance of developing lung cancer [Brown 1972]. An example from the private sector is the analysis by Bell Telephone Laboratory statisticians [Gabbe 1967] of proton data from the Telstar I satellite. This analysis significantly increased understanding of the earth's radiation belts. Although these and other examples of successful analyses exist, historically there was no general methodology, applicable across many fields, for analyzing large data sets. Furthermore, there had been no research efforts directed specifically at the problem. Most researchers who had large data sets were necessarily interested in the analysis of only their particular data. Thus, almost all existing large data set analysis techniques were special purpose with little consideration given to the problems and opportunities common to all large data sets.

This lack of a general methodology and the importance of such was recognized by the Institute of Mathematical Statistics (IMS), the theoretical statistics professional society. In February of 1978 IMS held its first special topics conference in 44 years. The topic of the conference was the analysis of large data sets. Examples of large data sets were given, some specific analyses were presented, and general analysis philosophy was discussed. At the end of the conference, it was evident that an important problem, spanning many disciplines, had been addressed, but that there was no general solution.

*Work supported by the U.S. Department of Energy, Contract DE-AC-06-RLO 1830.

Since 1978 the recognition of the analysis of large data sets as an important research area has grown steadily. In addition to the ALDS project at PNL, several other institutions have begun research programs that address certain aspects of large, complex data sets. Researchers at Oak Ridge National Laboratory (ORNL) are studying classification and pattern recognition strategies as a way of reducing dimensionality in large, complex data sets. Friedman and Breiman have recently begun a project in the analysis of large data sets researching 1) methods of subsampling that weight sparse areas of the population, 2) software for selecting several subsamples and performing contrasts on each sample and 3) techniques for identifying outliers and imputing missing values. The American Statistical Association (ASA) held an invited paper session on the analysis of large data sets at its 1982 annual meetings. Within the computer science community there is an annual international conference on Very Large Data Bases that addresses the data management problems of large data bases. Large data sets have been a major reason for the recent increase in interest by the computer science community in the data management problems of statistical data bases. Lawrence Berkeley Laboratory (LBL) organizes an annual Workshop on Statistical Database Management. Researchers at LBL, PNL, the University of Florida and the University of Wisconsin are supported by AMS to study data management issues specific to the data analysis applications of statistical and scientific data bases. Thus from 1978 to the present the analysis of large data sets, with impetus from AMS, has grown from a recognized but unaddressed problem to an active research area with several groups of researchers studying various aspects of the field.

When the ALDS project was begun in 1978, the overall objectives were to develop a large data set analysis capability and to apply this capability in the interests of DOE goals. To realize these objectives, an interdisciplinary team of statisticians and computer scientists was formed. The goals for the first three years were:

1) to survey current large data set analysis activities and software

2) to research new large data set analysis methodologies

3) to develop large data set analysis tools

4) to integrate appropriate existing and new methodologies into a viable large data set analysis software system

5) to establish a statistical laboratory dedicated to large data set analysis, based on a widely available minicomputer

6) to analyze certain large data sets

7) to disseminate information on the analysis of large data sets to the research community and to encourage research in the area.

To guide and advise the ALDS project in reaching these goals, the ALDS Review Panel, composed of leading statisticians, computer scientists, and data analysts, was organized to meet periodically to review the project. These goals have been attained for the most part. A later section of this paper will trace ALDS accomplishments in pursit of these goals in the area of data base management.

To gain some perspective on the methodologies for large data set analysis and management, it is instructive to consider how the amount of data can impact the analysis process. Operations that are only minor annoyances with small data sets can become major roadblocks with large data sets. It is time consuming to enter the data or move it around. Multiple, nearly identical copies of a data set can require much more storage space than is available, and keeping track of what has been created is very tedious. Data handling and management in general are large problems.

The analyses that are performed are also limited by the size of the data set. With a great many variables, the possible combinations of variables that must be considered can be overwhelming. The effectiveness of some analysis tools can be seriously reduced. For example, plots may be so saturated that no interesting features can be discerned, and searching or ordering the data set may require so much time that these operations are avoided. Some analysis procedures may be too time consuming to perform unless a useful outcome is a certainty; consequently, the analysis may be constrained and important features left undiscovered. Though it may not be an adverse effect, the whole course of analysis may proceed differently with a large data set. Rather than analyzing the complete data set at once, the analysis may have to proceed iteratively through the steps of subsetting, analyzing and verifying. Thus, part of large data set research must focus on expanding current computer science and statistical methodolgy to minimize the adverse effects of size.

2.0 DATA MANAGEMENT ISSUES FOR THE ANALYSIS OF LARGE DATA SETS

During the initial conceptual stages of the ALDS project, both the computer scientists and statisticians worked closely together to identify the most critical problems. Questionaires were constructed and data analysts were interviewed. Statisticians were observed during analysis and a users manual for a desired system was written. It became clear that a critical limiting factor in the analysis of large data sets was data manipulation during the exploratory data analysis processes. Because of their size, large data sets require much data massaging before analysis can be started. Furthermore, an analyst needs to be able to quickly understand and access the several related components of a data structure. But since great size usually entails great complexity, this can be a formidable task.

To address the data management problems, the ALDS project decided to build a prototype system. Initially, adapting existing data management tools to satisfy some of the needs was considered, but many limitations were found in existing data management tools. Some of these limitations were:

1) It was difficult to transfer data from the data management tools to other tools such as statistical or graphical routines for analysis.

2) The typical data management system was overburdened with overhead. This overhead is typically required to allow for complex data structures and operations such as concurrent updating.

3) Most data management systems require a preconceived data structure. However, none usually exists for the large data sets being analyzed.

4) The data analysis process was dominated by column-oriented access rather than row-oriented access. This characteristic had a major performance impact on the analysis in large data sets. Therefore, systems that contained "transposed file" format were seriously considered.

5) Existing data management software would be difficult to modify at PNL.

Because of these limitations, the ALDS Review Panel recommended that the ALDS team develop their own prototype analysis and management system particularily suited for large data sets.

Our first major development was to define a set of "kernel" data management functions required for data manipulation on large data sets [Burnett, 1982]. These data management functions were designed around a Self Describing Binary (SDB) tranposed file format [Burnett, 1981a]. The three dominant characteristics of that proposed file format were 1) that it was relational in structure, 2) that the file contain descriptive information about the data for user access, and 3) that the data be stored in transposed file format. The transposed file format provided fast access to the data and offered new avenues for data compression. This self describing data file format has now been utilized in several other systems.

The low-level SDB and kernel data management functions provided the base for the development of a tool that allowed the data analyst to have the same flexibility with data that computer scientists have in editing source programs and that data management experts have through the use of query languages in data management packages. This tool was called the ALDS Data Editor [Thomas, 1981]. The primary function of this tool was to enhance the data-handling process prior to the exploratory data analysis. Included in this tool were capabilities to create subsets based upon relational expressions, to transform data, to select data based on missing values, and to restructure data into new files. A special facility called a "virtual subset" allowed the analyst to select a subset based upon a relational expression and then perform subsequent operations on that subset. The subset was formed internally by storing a definition of the subset rather than by replicating all the values in the subset. This reduced the required storage and improved the operational characteristics of manipulating large data sets, which typically involves analyses on numerous subsets. The virtual subset provided a technique for analysis (temporarily) with limited storage. If needed, actual subsets containing all selected data values could then be generated. The ALDS Data Editor also contains a limited number of statistical tools including random selection, ordering, and histograms.

It was imperative that the analytical and graphical tools be tightly coupled to the data manipulation tools. This was accomplished to a limited degree by providing access through the same data format to all tools within the ALDS system. For example, MINITAB was used as the primary analysis tool. An interface was provided between MINITAB and the SDB file

format. This provided the functional capability of going from one tool to the next. Also this allowed the ALDS project to bring up a prototype system quickly. Ideally, the data management tools, the graphics tools, and the statistical tools would be under one single conceptual system.

It was equally important to the data analyst to have a workstation environment. To accomplish this a DEC VAX 11/780 with a high resolution color Ramtek was acquired. Other facilities included black and white printer/plotter, color film recorder providing 35mm, 16mm, and 8X10 hardcopy, desktop color plotter, a letter quality printer, and user-mountable 300MB disks. It should also be noted that an important part of the analysis of large data sets research project was the preparation of material for presentations as well as publication. Therefore, tools in the form of a viewgraph generator and text processing tools are also part of the data analysis workstation.

In conclusion, the prototype system did enhance the exploratory data analysis process. With the system, data analysts are now interactively manipulating millions of data points and exploring new techniques for visualizing high dimensions on large data sets.

Many enhancements could be provided to the prototype system that would achieve improvement in efficiency and functional capabilities. These enhancements, however, would not provide "the next generation data analysis system." For purposes of this paper, we will define the next generation system as one that provides an order of magnitude more capabilities as measured by the analyst's time to analyze large data sets and new capabilities not previously available. For the next generation system, the ALDS team first attempted to define the impact of large data sets on the interaction style for interactive data manipulation on large data sets [Thomas, 1982a]. This experiment lead us to believe that the next generation analysis systems will be designed around the conceptual model of the data analysis process. An initial attempt at such a conceptual model of only the data manipulation processes was called an Interaction Model [Thomas, 1982b]. The feedback from presentations of this model confirmed that others believed in this approach and a generalization to the data analysis process was conceived. This generalization is formulated around a concept called "data analysis environments." A brief description is contained in the next section and a more thorough discussion is presented elsewhere in these proceedings [Burnett, 1983].

3.0 ALDS RESEARCH AND DATA BASE MANAGEMENT

Current ALDS research has resulted from PNL experiences in attempting to overcome analysis or data handling difficulties and trying to overcome the problems inherent in large data sets. Each also represents an area where current statistical or computer science methodology is not sufficient to permit the effective analysis of large data sets. All ALDS research areas are either directly concerned with or are significantly impacted by data management issues. The following subsections explain how the current research areas relate to data base management for large data sets.

3.1 EXPLORATORY DATA ANALYSIS GRAPHICS

Most interesting large data sets are characterized not only by a large number of cases but also by a great many variables reported for each case. Both the number of cases and the dimensionality of the data present a challenge to the effective use of statistical graphics. Because of the high dimensionality, the number and complexity of multivariate relationships that can be explored is very large. However, there is usually a tradeoff between the number of points and the number of dimensions that can be displayed. With standard display techniques, this tradeoff is not really a problem since most provide only a two- or three-dimensional view, although with enough points even simple displays can become saturated. Thus, there is a need not only to expand graphical techniques to allow the display of four, five, or more dimensions, but also to make the displays useful in the context of large data sets. In order to effectively display high dimensional data, the variables will most likely have to be scaled, binned, or otherwise transformed to control the high dimensional viewing. This process requires extensive data manipulation before graphical viewing.

Because the visualization of graphics displays is a natural environment for interactive data analysis, computer science tools must be developed for direct analyst interaction with the data displayed in its full dimensionality. With many dimensions, the number of lower-dimensional views of the data that can be created is astronomical. It is physically impossible for an analyst to study each one. Thus automatic techniques are needed to generate low-dimensional representations, to manage the large amounts of data generated by such a process, and to evaluate each representation as to its usefulness to analysis.

3.2 MANAGEMENT AND DISPLAY OF DATA ANALYSIS ENVIRONMENTS

When the analysis of a large data set is attempted, the first problems to be encountered are usually data management problems resulting from the amount or complexity of the data. The iterative nature of data analysis tends to proliferate data sets and results. Thus, if multiple subsets of the data or analysis results are allowed to accumulate, the storage capacity of any system can be saturated quickly. Such proliferation also severely taxes the organizational ability of most analysts. At any stage of an analysis, there are usually many different analysis paths that should be followed, depending upon which models are assumed or which hypotheses are entertained. Some of these analysis paths will most likely result in dead ends, while others will suggest new aspects of the data to examine. A simple model of the analysis process is a tree-like structure with the nodes representing different versions or subsets of the data or results and with the branches representing different analysis paths. The analyst moves between nodes using analysis or data management software. As long as the analyst is moving down a branch, most existing analysis systems will suffice, although keeping track of the many subsets and results generated along the way can be a problem. The serious deficiencies in existing systems become apparent when it is desired to return to a previous node and start a new branch or to combine the results of several branches. With a large data set it is not possible to save everything because of time and storage limitations. Restoring saved nodes can also be very time consuming. If the desired nodes have not been saved, it can be a difficult task to repeat the steps that produced the nodes. These difficulties are a hinderance to analysis that may result in leaving useful paths unexplored.

A more complete view of the analysis process replaces the nodes in the tree with "data analysis environments." Each environment represents not only the version of the data set at that time but also the sequence of operations that produced it, the statistical model under which it was produced, the status of system parameters, and descriptive text. The realization in a data analysis system of this "data analysis environment" model of the analysis process would permit more efficient and thus more complete analyses of large data sets. Research is required to evaluate the application of several computer science concepts to this "data analysis environment" model. These concepts include data modification descriptions (differential files), data dictionaries/directories, and graphical network representation and interaction techniques.

3.3 UNIFIED LINEAR SPACE INFERENCE

Much of classical small sample statistics consists of confirmatory analysis procedures; that is, methods for examining the degree to which suspected patterns in or characteristics of an experiment are confirmed by the observed data. Methods for estimating the parameters in a model and testing hypotheses about the parameters are commonly used. Within the context of confirmatory analysis, large data sets conceptually require a similar analysis - parameters will be estimated, hypotheses tested and patterns examined. However, within the context of large data sets, confirmatory analysis can profit from some new approaches. With large data sets, very general models and hypotheses may be formulated with complex constraints. Linear space methodology needs to be developed in the framework of large data sets to provide the flexibility to accommodate the very general models and hypotheses that assume only that the data are vectors in some linear space. To take full advantage of large data sets, aspects of inference in infinite-dimensional spaces must be considered. The efficient use of these techniques will require especially effective means of handling the very large sparse matrices generated by the linear space methods.

Examples of large data sets where this methodology would have immediate application tend to have time series as their data "points." Though difficulties can occur in the areas of editing or processing time, a single time series of any length requires no new methodology for an effective analysis. However, when multiple time series, structured as some form of analysis of variance (ANOVA) design, are collected, current methodology lacks a coherent approach to the analysis. Moreover, the intelligent storage and management of this type of data is vital for a coherent analysis. An area where this is especially obvious is multisource data. Often data relating to some question are available from a wide variety of sources spanning many different collection eras. Moreover, the data are usually collected for different purposes; thus, the ANOVA structure of this data is after-the-fact. As a result, the "design" is incomplete in that the intervals between points in the series may vary between series or in that only partial information on the levels of the factors for some series may be known. Classical analyses can proceed only by the imposition of restrictive assumptions or by discarding some of the information in the data. The

unification of linear space methodology and infinite space inference will permit this type of problem to be addressed directly and more effectively.

3.4 STATISTICAL SUMMARIZATION STRATEGIES FOR DATA COMPRESSION

Since large data sets pose considerable problems in the areas of data storage and retrieval, an immediate solution is to use data "summaries" to reduce the volume of data. This solution, however, raises the question of what are adequate summaries. The answer depends both on what will be required of the data set and on the structure of the data. In special cases, all of the information in the data can be reduced to a few statistics. For instance, if the data are independent and from the same Gaussian distribution, the sample mean, sample variance and sample size are all that is required to answer any distributional question. It should be noted that even in this simplest of all cases, there are still many questions that cannot be answered by the sample mean and variance; for example, what is the maximum of the data? Another problem with a data reduction of the type illustrated above is its lack of "robustness;" that is, once the data have been reduced there is no way to check if the Gaussian assumption is acceptable or to see if the order of the data furnishes information concerning the independence assumption. There is thus a tradeoff between preserving the data to answer unanticipated questions and reducing its volume.

A summarization strategy that retains much of the information for a wide variety of distributions is to use an empirically determined density estimate as the summary. For univariate data, several such estimates are available, although research is needed to determine which would be useful in the contexts of data compression, retrieval, and computation.

For multivariate data, considerably fewer density estimates are computationally attractive and many questions remain unanswered. For example there exist no strategies for determining which of the many possible low-dimensional summaries should be saved. Even for bivariate data, research is needed on how to construct useful bivariate bins and how to approximate densities from nested regular grids.

4.0 PERSPECTIVES

Reflecting on what has been achieved and learned by the ALDS project, several points are worth noting. It is hoped that these will be of benefit to others considering research in statistical data base management on large data sets.

1) A characteristic of data management tools from the analyst's standpoint is that they must be hidden tools in the analysis of the large data sets process. One cannot afford the luxury of going between separate tools and requiring the user to understand different command interfaces.

2) The next generation of systems will be driven by conceptual levels of the data analysis process. There is an obvious lack of understanding and tools to help characterize this process. The ALDS team as well as others in the technical community have attempted a questionaire and interviewing approach. This offers some insight, but ususally results in a list of desired functions out of existing data analysis systems.

3) The graphics component is an intergral part to effective data manipulation. It is necessary to understanding initial data structures and cannot be considered a secondary tool.

4) An interdisciplinary approach of statisticians and computer scientists continually working together is required to address the above research issues.

5) Flexibility is of the utmost importance. Under certain circumstances, any data base can be a "statistical" data base. Characteristics of statistical data management systems that cause flexibility to be limited are of marginal use.

5.0 REFERENCES

Brown, B. W. Jr. 1972. "Statistics, Scientific Method and Smoking." In Statistics: A Guide to the Unknown, pp. 40-51. Holden-Day, San Francisco, CA.

Burnett, R. A. 1981a. "A Self-Describing Data File Structure for Large Data Sets." In Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface, pp. 359-362. Springer-Verlag, New York, NY.

Burnett, R. A., and J. J. Thomas. 1982. "Data Management Support for Statistical Data Editing and Subset Selection." In Proceedings of the First LBL Workshop on Statistical Database Management, pp. 88-102. Lawrence Berkeley Laboratory, Berkeley, CA.

Burnett, R. A. 1983. "Management and Display of Data Analysis Environment for Large Data Sets." In Proceedings of the Second International Workshop on Statistical Database Management. Lawrence Berkeley Laboratory, Berkeley, CA.

Gabbe, J. D., M. B. Wilk, and W. L. Brown. 1967. "Statistical Analysis and Modeling of the High-Energy Proton Data from the Telestar I Satellite." Bell System Technical Journal, Vol. 46 No. 7, pp. 1301-1450.

Thomas, J. J. 1982a. "The Impact of Large Data Sets on Interaction Style for Data Manipulation Languages." In Proceedings of the First LBL Workshop on Statistical Database Management, pp. 157-159. Lawrence Berkeley Laboratory, Berkeley, CA.

Thomas, J. J. 1982b. "A User Interaction Model for Manipulation of Large Data Sets." In Computer Science and Statistics: Proceedings of the 14th Symposium on the Interface, Troy, NY.

Thomas, J. J., R. A. Burnett, and J. R. Lewis. 1981. "Data Editing on Large Data Sets." In Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface, pp. 252-258. Springer-Verlag, New York, NY.

DATA MANAGEMENT WITHOUT A DATABASE MANAGER

Michael A. Fox

UCLA Hospital Computing Facility Los Angeles Ca. 90024

Abstract

A file-handling full screen data entry, verification, updating, and display system, with overtones of a data base management system without the overhead is described. Using a high level specification language, created automatically by describing a form on the screen, a PL/1 program is generated which will display empty forms allowing data entry, verification and retrieval (by example). Files or transactions to data base systems may be constructed, and existing files formatted for display, post hoc field validation and restructuring. The system is complete but may be interfaced with other systems and, because it is a program generator, used to provide building blocks (reusable code) for other full screen applications.

Key words and phrases: full screen, data entry, edit, display, reusable code, design specification, code generator, form design, IBM 3270, reliable code, data management, file management.

Introduction

The design of reliable programs is expensive and time consuming. In recent years we have seen an emphasis on methodologies aimed at providing robust, comprehensible, and maintainable systems. This thrust manifests itself in such areas as design specifications, reusable code, and program generators. A practical working example of these software practices is presented.

Much of the statistical worker's job is concerned with data management. This system was designed to do the screen handling tasks of data management and perform the storing and retrieval when the expense, overhead, and commitment to a DBMS is unjustified. To perform analysis one collects data, which must be validity-checked and arranged in suitable order. Occasions arise when complex editing rules have to be applied to data and there is always the desire to view, in a legible format, individual cases for possible update.

These tasks are essentially the same for all data sets. Passing from one study to the next should not therefore require repetition of mundane programming tasks. Finesse at the specification level is needed. The current system-- Design A Form addresses the need for creating a system with many of the features of a data base management system but which preserves the physical form of the data and does not require the overhead of a DBMS. For a very large quantity of highly structured data a DBMS is mandatry. By using transactions to such a system Design A Form can be used as a powerful front end.

The system is a program generator- a specification directing a compiler to produce PL/1 source code. This source is complete and will on compilation and linking to the nucleus of the system enable data to be entered retrieved and updated. A code generator removes any constraint implied by the specification language and further permits the generated code to be used as building blocks in systems that require screen management.

System Requirements

The initial requirements for the system are a full screen data entry method which to the user looks like a form. This "form" should allow single-field and across-field validation to be performed, to any degree of complexity. Facsimiles of the filled-in forms should be available. Files created by the system should be simple data files. Other systems could immediately use these files and conversely files created by other systems should be readable by this system. A mechanism should be available to pass the data entered or derived in other ways, such as transactions. Thus more complex data structures may be accommodated. Retrieval, updating, restructuring, and post hoc data validation should be supported.

Presentation of data either for entry

or retrieval is via a form displayed on the video terminal. There are two aspects to forms: the geometric or graphical design which is captured by drawing directly on the screen; the attributes of the fields which specify the color, highlighting, and the validity checks, edit rules and other aspects of the input data. The specification language captures both these aspects. The "drawn" form is read and the first stage of specification is automatically obtained. Each field is then displayed, together with a set of attributes that may be chosen, and the ability to enter and alter editing rules is provided. The second stage in specifying the form is thereby accomplished.

In any system, as the number of options increases so does the richness of the specification language. To create a system which will not suffer from artificial constraints imposed by the specification language it would appear that the latter would have to evolve into a programming language. Extreme simplicity and a high degree of flexibility are not, however, mutually exclusive. The reconciliation of these requirements has been realized in a program generator. The target language chosen was PL/1 and the display device the IBM 3270 series of both monochrome and color terminals. Communication between this full screen device and a program is via a data or message stream and a controller. As only fields that change need be transmitted, the minimum amount of data need be sent. The onus is, however, on the program to determine which areas of the screen have changed. There is no higer language level support for this device in full-screen mode. This need is addressed by providing support through generated PL/1 code.

It must be emphasised that although the system generates PL/1 source code, in the vast majority of cases, the specification language captures all that is required and going from form design to data entry, updating, and retrieval requires no programming knowledge. It is a task easily accomplished by non-technical people. Complex situations involving multiple forms with context-dependent decisions, will require some intervention. Experience has shown that this open-ended feature is welcomed by those who use packaged programs but have needs that transcend them.

System Design Considerations

In program writing the most serious errors are those involving control structures (dynamic conditions for following a particular path are incorrectly computed) and data structures (bounds of arrays are erroneous). Syntax errors are, for the most part, found by the compiler. Generating source code, via a specification language, offers a number of advantages. The most important is that the production of error-free programs is an achievable goal. Simple applications can be "brought up" in a few minutes.

Customizing starts off with complete, correct code and is therefore at minimal risk with respect to the introduction of errors. This is because all the "hard" error-prone program facets such as management of data structures and, in this system, screen-handling code is produced automatically. Adapting the code to perform such operations as complicated table-look ups and embedded computations can of course introduce errors. These, however, will be confined to small domains and are easily detectable.

This system is an example of "reusable code", each new application can be thought of as a progeny varying from its parent in its specific function, yet sharing a common, reusable, core. Others my view this system as a tool for generating screen management code which can be incorporated into other systems.

The concept of abstract data types and encapsuled modules is used in code that is "hidden" from the user thereby retaining its viability. The generated code is, however, open to the user. Hidden code ensures that it can only be approached and used in a controlled way. Open code provides for easy and swift customization. The capsule "screen" which consists of the abstract data type used to hold and manage data going to and from the display device is available as hidden code. This means that the user can control what is displayed, and receives back the analyzed replies to the screen. The mechanism for doing this is hidden from the user and therefore cannot be compromised. Each screen is represented by an open code PL/1 procedure which contains three distinct parts: a PL/1 structure whose elements contain all the replies; the actual stream of characters sent to the terminal controller; and a "case"

block (called Select in PL/1). In this case block each field which has a reply is represented and is available to the user. In-line code can easily be incorporated - a marked advantage over having to provide a subroutine. A developer who wishes to use the system beyond that which is generated, via the specification language, has only to know the specifics of his data and not how screen management is accomplished. In the same vein a user may want to use only part of the generated code in conjunction with another program. For this situation the PL/1 structure is a compact way of holding the variable information presented on the screen. Automatic production of the "message" to be sent to the controller relieves the user of the task of constructing the correct sequence of control information. The "case" block facilitates operations on individual replies.

Since logic between screens is independent of the hidden code, versatile applications can be designed with ease. For example, in a financial planning example three screens were used for each case: the first to collect data and the others to display computed results. A procedure was written to do the required computations. The only system housekeeping required was declaring the PL/1 structures defining the three screens to this procedure and steering, via calls, the display of each individual screen. Indeed skeletal steering procedures are part of the system and are generated when multiple screen applications are defined. A user then had available a system that dynamically displayed the consequences of, for example, a fluctuating dollar value on the cash flow situation of a company that deals with purchasing and leasing goods.

Form or Screen Design

A program "BLANK" presents a screen to the user, on to which the form or tableau is typed. This will consist of legends or narrative material, field names, and reply fields (denoted by asterisks). Lateral movement of objects on the screen is accomplished via the terminal's positioning keys(insert and delete). Vertical movement and copying of a line to another line is done by typing the target row number on the source row and using a function key. When the desired result is achieved a function key is pressed resulting in the translation of the screen into an instance of the specification language. For example if Income ****** were typed on line 10 column 5 on the screen it would be translated into

Income (10,5,6) This is of the form

```
Legend or field name
      (row,column,length)
```

After form design, attributes and edit rules may be applied to each field interactively.
The specification Income (10,5,6,NB), contains the attributes "NB", this means accept up to six numeric characters (N) insisting that this field be entered i.e. non blank (B). Figure 1 shows this field with the editing criteria chosen together with the rules generated by placing bounds on the entered value. For the field "Sex" to take on only the values "M" or "F" with a user-defined message the customised rule is written as

```
:¬='M' &¬ ='F' Sex can only be M or F :
```

Other field attributes include date, right justification, full replies, default values compute fields, and value carry over (from one form to the next) are shown in figure 1. The most general specification for a single field is

```
Legend (row,column,length,attributes)
  : failure condition message ;
    failure condition message ; ... :
```

On data entry and update, erroneous entries will be signaled by writing a message, sounding a bell, positioning the cursor and, highlighting the offending field. The operator can either correct the field or, by pressing a function key, force the reply to be accepted.

Retrieval and Updating

The screen displayed when retrieving and updating is shown in figure 2. Some of the options need explanation: "Y" implies a pattern match is to be performed over non contiguous fields or field fragments; "=" is restricted to pattern matching over contiguous fields or field fragments and is therefore cheaper than "Y"; "@" is a range over the supplied value, the default value is 10%; "(" is a substring taken over a contiguous set of fields the last of which may be partial; "S" evokes the display of an information pannel which gives, among other things, the current

default values, which can be reset. A mask character is used as a 'wild' value and to define the scope of the search as in option "(".
Retrieval is by example. An empty form is presented on to which search criteria is typed. This in conjunction with a condition (equality, greater than, less than, substring, or range, mentioned above) is used to find cases. In addition, all cases can be sequentially displayed or, for post hoc editing, a search for cases that fail an edit rule selected. In the update mode, cases can be altered with dynamic recalculation of compute fields. Pressing a function key will redisplay the changed case. Editing criteria and reformatting data can be respecified.

It is unfortunate that designers of data-gathering forms still continue the archaic practice of encoding items at source rather than leaving this to the computer (males=1 etc required by some packages for grouping variables). Design A Form takes the position that source data should be readable and therefore provides a translation mechanism to go from a source of one size to a target of another. The default value in the printed facsimile of the form is the source while the target value is written to the file. On retrieval both the stored value and the inverse translation can be displayed.

Interface Considerations
In the normal situation the files created by the system are standard systems files, with the data appearing as characters without any embedded control information. Such files can immediately be read by any other program as raw data. Since "ghost" fields (usually filled with blanks) can be specified variable values can be separated from each other to facilitate "free form" entry in other packages.

This detachment of the files from the system provides for complete data independence. This makes feasible the use of the system as a "front end" to a data base system for situations where the data are too voluminous for simple files. In this situation transactions would be sent between the two systems. One variant of this system exists that acesses save files created by the BMDP statistical package. Queries written in "Sequel" like languages can readily be serviced.

System Limitations
Functional separation and modularity are part of the design philosophy of this product. Features that rightly belong in other systems are excluded. As the system has overtones of a data base management systems (DBMS), the important differences must be stressed.

Here data is looked at in a case-by-case manner, a DBMS is more global in approach. In a good DBMS, searches do not usually require passing the whole file. In inverted systems much logic can be performed on the reference files or directories before looking at the actual data. Retrieval using logical expressions is limited in this system. Although hierarchical cases are supported, no connectivity is internally maintained between the various record types (relations). Cross-case computation (means, minima, etc.) is a simple customization task: however, anything more complex is the province of a statistical package. A DBMS excels in handling relationships between volatile data. This system is geared towards changes to data items rather than relations. File, rather than data base functions, are performed, although new cases may be inserted anywhere and deletions performed.

For a very large file the overhead in computer cost, storage, and more complex operator training required by a DBMS is warranted. There are, however, many situations when data management should be restricted to files. In such situations a DBMS would be too complex. Many studies fall into this category. Instances exist where data remains static and essentially lost because no suitable means is available to read and comprehend it. A large series of files containing encoded information concerning land utilization lay unused because of the difficulty in using it. A translator to these files was incorporated into this system which was used to display the data in English in a pleasant format that encouraged use. A DBMS would have been too costly and too difficult for the casual user, and since the nature of the searches necessitated either passing the whole file or searching until the first record satisfying the conditions was found, no advantage would have been gained in using the more powerful search strategies of a DBMS.

This system, because of its ease of use, its file handling features, its

simple interface to other packages, its flexibility, and its parsimony can play an important part in data management.

Figure 1.

Selection of attributes and specification of rules

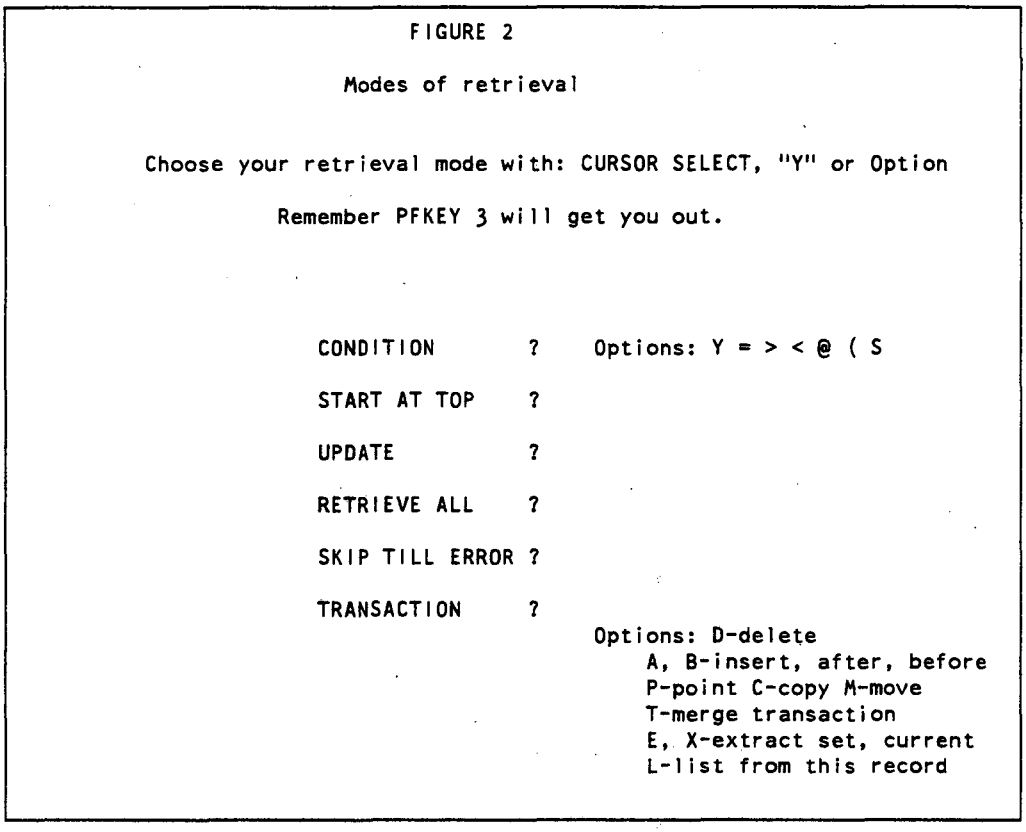
Current field is : Income
Attributes NB
Edit rules : ≤ 15000 Too small ; ≥ 60000 Too large :

Select the attributes to be applied with CURSOR SELECT or "Y"
After you press ENTER the new attributes will be shown.

Integer	(N)	Y
Floating point	(F)	?
Right justify	(R)	?
Non blank	(B)	Y
Compute or defined fields	(C)	?
Date field	(D)	?
Full width reply	(W)	?
Set carry forward	(S)	?
Ghost field (set length)	(G)	
Before/After		B
Default value (set value)		
Translate field (set target size)		
Align (set position from right)		
Minimum value		15000
Maximum value		60000

FIGURE 2

Modes of retrieval



Development Implications of an Interactive, Portable, User-Friendly, Statistical Database Management System or The Separation of Church and State

Gordon L. Schiff
Hoffmann-La Roche Inc.
Nutley, New Jersey 07110

This is a definition of terms and discussion of the design and development of an interactive, portable, user-friendly statistical database management system from the perspective of a computer scientist. The definitions are fairly clear and can be found in most textbooks but their interactions, particularly when a system is being designed and developed which attempts to include all of them, have been given relatively little attention. This paper will focus on how the major structural elements of an "ideal" statistical database management system impinge on each other during the design and development phases of that system. There will also be a discussion of how these elements and their interactions affect the people designing, writing and using the system.

This is a definition of terms and discussion of the design and development of an interactive, portable, user-friendly statistical database management system. The definitions are fairly clear and can be found in most textbooks but their interactions, particularly when a system is being designed and developed which attempts to embody all of them, have been given relatively little attention. This paper will focus on how the major structural elements of an "ideal" statistical database management system impinge on each other during the design and development phases of that system.

The paper consists of two parts. The first part is a brief discussion of the definitions of the terms portable, user-friendly, and statistical database management system and their interactions while designing a system which attempts to embody all of them.

What will follow is a discussion of the implementation of these three primary attributes in a system with a focus on their impact on the computer scientists - programmers and statisticians developing the system, the same professionals bringing the system up on a different computer, and the end users; primarily statisticians, but a few computer scientists as well.

It is important to emphasize that this paper is not a description of a system which has already been built, and is not an explicit design document. It is hopefully a complete review, from the perspective of a computer scientist who works with statisticians, of the factors which must be given careful consideration when designing and building such a system.

DEFINITIONS

A portable system is one which can be brought up and run on any machine, no matter what the word size and manner of representation of numbers, with an absolute minimum of effort on the part of the person/people installing it on the new machine. Numerical precision differences between machines should be easily correctable by the installer(s) or, if not correctable, clearly and completely documented by the developers in the installation guide. In addition, the user's manual should be written so that it does not depend on the machine that the system is running on. This latter point is one which seems to have been neglected by the developers of a number of the more popular, "portable", statistical packages in existence today.

A user-friendly system should be just that: friendly to the user. The designers and developers of the system need not be psychologists or group therapy leaders, but they should make the users comfortable when running the system.

The system should guide the user clearly through every phase of its use. Questions presented to the user during an interactive session should be clear and unequivocal with a detailed explanation of each question, examples of answers to the questions and consequences of each of those answers readily available.

An extensive, thorough, multi-level help facility or subsystem should also be available at every point in the operation of the system if the user wants or needs to use it. This help facility should be written to supplement, not replace, the users' manual.

A user-friendly system should recover gracefully from and offer possible solutions to I/O errors and computational errors resulting from processes such as those which may attempt to divide a number by zero, determine the square root of a real negative number, or require a datum that is missing.

The system should also be unaffected by hard system crashes insofar as the database being worked with when a crash occurs will not be affected. In addition, the system should allow the users to easily get back to where they were in an interactive session at the time that the system crashed.

A good deal of time has already been spent and more will be spent, especially at this workshop, discussing what a statistical database management system is. A basic definition of one might be as follows: a database management system which allows for the rapid storage and extraction of data in a format suitable for statistical analysis. A method for satisfying this definition might be that the statistical analysis procedures should themselves be part of the system and it should be easy to add, remove or replace these procedures.

RELATIONSHIP BETWEEN PORTABLE, USER-FRIENDLY AND STATISTICAL DATABASE MANAGEMENT SYSTEM

Most of the relationships between these three attributes have, as one of their nodes, portability. This is not surprising, since it is probably the most difficult one to implement in a large system.

Making the users comfortable while using the system but allowing it to run on any machine means that in most instances, good programming will have to be substituted for innovative hardware. Along the same line, it will have to be good writing with a statistician, not a computer scientist or ichthyologist in mind.

The methods of trapping computational and I/O errors vary from machine to machine, and memory and storage management differences between machines will affect the preservation of data upon occurrence of a system crash. Differences in the loading and/or binding of programs from one machine to another will affect the easy addition, removal and replacement of statistical procedures.

Finally, the size of the system is an important consideration when writing a portable software package.

DESIGN AND IMPLEMENTATION

For quite some time now computer scientists and statisticians have been meeting in an attempt to recognize and solve common problems. The Interface Symposium and this meeting are typical examples of the effort. This paper explores an area in which those two professions are going to be obliged to work in close harmony. No attempt is being made to determine which is church and which is state or the benefits of being affiliated with one or the other. It is simply important to note that when this effort is undertaken the two professions will have to be extremely cooperative with, and understanding of, each other.

The first major decision which must be made during the development stage of a system such as this is in which language to write the system. With portability in mind, it will first be necessary to study the compilers/interpreters available on all possible, existing target machines paying particular attention to word size, precision, and I/O and memory management facilities available. At the present the language of choice, primarily because of its popularity, would probably be FORTRAN, although by the time this effort is undertaken, PL/1 or PASCAL (or ADA?) might be universal enough to be considered. It will be important that everyone involved with this initial phase of the design effort have facility and be comfortable with a number of major computer languages. It is also important not to pick FORTRAN just because "everybody knows it" and "every machine has a FORTRAN compiler".

A second part of this initial design phase will be the development of standards for everyone who will be writing the code. The standards should include the types of statements allowed and a thorough list of the types of statements not allowed. They should also include internal documentation standards for the programs. The standards will have to be well written, easily used, and on the desks of everyone who is writing code for the system.

One standard which can be stated immediately is that absolutely no machine or assembly code at all is allowed. With the definition of portability in mind, it is obvious that machine or assembly code cannot be used in the system if it is to be brought up easily on many different machines.

The next major decision in the design phase of this project would be what the overall structure of the system should look like. The terms "top-down" and "modular design" have become popular buzzwords, but they are very powerful design tools when dealing with large systems. A top-down, modular, or inverted tree approach to the design of this system would seem to be perfect for a number of reasons.

Any machine-dependent routines performing such tasks as I/O and memory management and those routines with machine-dependent constants, which would affect the precision of results, could be placed at the very highest levels of the overall structure. This would allow for easy access/creation/alteration of logical unit numbers for I/O and modification of constants by those installing the system on a different machine. As an aid to implementation on different machines, standard sets of constants for machines with different word lengths could be made a part of the appropriate high-level routines.

The actual statistical and data base manipulation routines would exist at the very bottom of the tree. These leaves would have their constants already set at a considerably higher level of the tree and would not have to be altered in order to be brought up on another machine. The "proven" algorithms would remain intact. It would be the responsibility of the statisticians participating in the development phase to insure that these algorithms were indeed proven. Incidentally, internal documentation is extremely and equally important at all levels of the tree. These routines would produce results or manipulate data items and return them to a higher level routine for appropriate disposition. This disposition would include output to the user, input to another algorithm and input to the data base. It is important to emphasize that there would be no I/O operations from within these routines. This absence of machine-specific code would allow for relatively easy addition, removal and replacement of leaves by the developers and the end users.

The impact of computer systems hardware on portability would be of considerable importance during

the design phase. One of the first items which would have to be resolved would be the structure of the database. Anyone familiar with the theory of database management systems has heard or read the terms network, hierarchical and relational when reference is made to the structure of such systems. There are advantages and disadvantages to each of them, and there are very few commercially available database management systems which are pure examples of any one of these structures. No attempt will be made in this paper to describe these structures and their differences. It is simply important to note that they have different requirements in the areas of data storage and retrieval. Differences in machine word size would affect pointers, sorting and searching algorithms and actual data storage, and this is where careful thought would have to be given to the portability of the system.

A relational database management system seems to be the best from the point of view of speed of data retrieval and ease of use, but the memory requirements of such a system might be too large to allow it to be portable. If the number of overlays or amount of paging required for a simple data retrieval operation were too large, the delay would be intolerable for the end user. At the same time, the installer(s) should not have to redesign the database management system to make it fit on their machine.

The modular design considerations would have to be taken into account when designing the overlay structure, if such a structure were necessary. For true virtual memory machines this would not be a problem, but there are few of them around. If an overlay structure is necessary, the design should allow for different ones for machines of different word lengths and memory sizes. The installation documentation and/or files should allow the installers to select the appropriately sized set of overlays. As an addendum to this point, it is to be noted that the system should function at approximately the same speed no matter what the overlay structure, but it would probably be slower on the smaller machines.

One aspect of portability which must be taken into account when writing the system is that obviously, the same statistical procedures processing the same data but running on two different machines should produce the same, with allowances for precision differences, results. This will require careful programming to eliminate problems resulting from round-off error, truncation and internal accumulation of round-off error.

I/O routines would have to use standard methods, with nothing tricky or unique to the machine the system was developed on. Since everyone does not have light pens, mice or touch sensitive displays, the I/O routines should not be predicated on these hardware innovations.

Building a user-friendly system means that overall design considerations will be strongly influenced by the requirements for a help facility, graceful recovery from I/O and computational errors and easy recovery from hard system crashes.

Help in using the system should be available to the user from any level of the system. If all I/O, including interaction with the user, is occurring at a high level of the system, the help facility could be built in at, or immediately below this level. This would require that the system maintain, possibly through the use of a set of flags or global variables, a knowledge of what the user is doing during an interactive session. When the user requests help, the system needs to be able to provide the appropriate explanations. This self-informing portion of the system would have to be readily and easily expandable as new leaves are added to the overall system.

A selection of the method for interacting with the user would affect the design of the help facility. Two possible options are the use of menus with help available for each item on each menu if the user wants it, or a simple question-and-answer approach with clear explanations of each question readily available.

Setting up appropriate, universal methods of error recovery may be the most difficult aspect of this entire design effort but again, a top-down, modular approach may facilitate the process. One might devise a system of flags or error returns which could be set at the lower levels and tested and acted upon appropriately at the higher levels. Depending on the language used, there might also be a machine-specific, high-level process which would disable system trapping of I/O and computation errors, allowing for a "normal error return".

Insulation of the system from hard, computer crashes will also be a difficult part of the design effort. Requiring that a recent backup copy of each database is maintained at every site will help. In addition, no database should be left open when it is not directly involved with some data storage/retrieval operation. In order to minimize the affects of a crash on the users, a log or audit of each user session could be kept. This would be fairly easy to do within the framework of a top-down system. All input from the user could be written to a log file by one of the high-level I/O routines. This would be done before the command or request was processed. When a system crash occurred, the user could, when the system came back up, either obtain a printed copy of the audit file and re-enter the commands during a new interactive session, or use the audit file as a command file which would be executed up to the point of the crash. This would leave the user at, or just before, the point she or he was at when the system crashed.

The other aspects of creating a user-friendly system depend on a good working knowledge of the

native language where the system is being created (usually English), and a good working knowledge of statistics. Whatever mode of interaction with the user is selected, the questions, explanations, examples, help facilities and manner of presentation of results will have to be clear and unambiguous. While we are not about to create a good version of ELIZA, the user should feel comfortable interacting with the system. It would not be a place for short, cryptic questions or explanations. Highly technical questions and comments, with appropriate explanations available, would be acceptable, but not the three-word gems of statisticianese or computerese.

The impact of user-friendliness on the person or people installing the system on a new machine would revolve primarily around the methods used for error recovery and insulation from computer system crashes. If a high-level process to disable system trapping was selected, the installers would have to create a new process for each machine. With appropriate complete documentation of that process, including all that it does, where and when it does it and expected input and output, the task would not be terribly difficult. Again, as with the machine-dependent constants, a standard set of processes might be made available, and the appropriate one could be used for each machine.

If frequent backups of the databases were required, the installer(s) at a new site would have to be most strongly reminded of this, and the standard operating procedures for system backup at that site might have to be modified. The use of log or audit files as command files would present a problem at some sites. The installer(s) might have to provide the users at that site with a preprocessor program which would make the log file acceptable to the machine's command processor.

The impact of a user-friendly system such as the one being described in this paper on the user community would probably be an increase in the use of computing facilities by statisticians and an increase in cooperative efforts to solve statistical problems. A truly user-friendly system would "bring into the fold" a number of statisticians who presently see the computer as a large, expensive calculator. It would be so easy for them to perform complex calculations in a matter of minutes, study the results and perform them in a different way, that their productivity would increase immensely.

Implementation of the two primary characteristics of a statistical database management system, rapid extraction of data in a format suitable for statistical analysis and ease of addition, removal and replacement of statistical procedures, have already been discussed but their impact on the design of the system will now be considered. Rapid extraction of the desired data will require that the structure of the database management system be such that storage/retrieval algorithms are fast and efficient. It is important to note that the system should only retrieve the data that is needed to satisfy a request. The format of the retrieved data might simply be vectors, but the system should be able to reformat the data where necessary for input to a statistical procedure or other output process. A relational database management system might do the job very nicely, but on a machine with limited memory this could prove costly. A standard for the format of data extracted from a database would have to be imposed on the extraction process. The statistical procedures or the twigs above them could alter this format where necessary, but with these standards it would be relatively easy to add, remove and replace statistical procedures if they existed at the very outermost points of the system.

With a top-down structure, alteration of the array of statistical processes should be nothing more than a straightforward recreation of the executable system. This could be done at any site with a command file which had been modified to reflect the alterations. The most important point, from the developers' perspective, is that new processes should be documented, tested, and proven or appropriate caveat emptor warnings about the processes given.

CONCLUSIONS

The most important conclusion that can be reached from this paper is that the design, building and implementation of an interactive, portable, user-friendly, statistical database management system will require the joint efforts of computer scientists and statisticians. The members of these two disciplines participating in the effort will have to work in a close, cooperative fashion in order to create such a system. The resulting system will prove to be immensely rewarding, although for different reasons, to both groups. The computer scientists will have solved some interesting problems unique to their profession and the statisticians will have a system which will enable close collaboration in the solution of statistical problems. If groups of statisticians in different cities, states or countries are able to use the same computer system as an aid in solving problems, they will be able to communicate with and help their colleagues much more easily than is presently the case.

DISTRIBUTED DATA MANAGEMENT IN A MINICOMPUTER NETWORK: THE SEEDIS EXPERIENCE

Deane Merrill, John McCarthy, Fred Gey, Harvard Holmes

Computer Science and Mathematics Department
Lawrence Berkeley Laboratory
Berkeley, California 94720

Abstract

This paper describes distributed data management aspects of SEEDIS (Socio-Economic Environmental Demographic Information System). SEEDIS is an experimental system for the retrieval, analysis, and display of geographically linked data. SEEDIS operates on nine computers in a nationwide network. Users at any location select and retrieve all data in the same way, regardless of whether they are stored locally or at a remote location.

The network implementation has been substantially modified during 1983. New enhancements include: local caching of data files to improve efficiency; linking to an automatic tape library (ATL) to make larger volumes of data accessible; node independence to facilitate automatic sharing of data among autonomous SEEDIS installations without the need for central control; improvements providing robust operation despite unreliable network connections; and automatic recording of all cache transactions for subsequent statistical analysis.

1. History and Background

SEEDIS (Socio-Economic Environmental Demographic Information System) is an experimental integrated computer system for the retrieval, analysis and display of geographically linked data [1]. SEEDIS embodies 60 person-years of cumulative integrated development, supported since the early 1970's by the Department of Energy, Department of Labor, Environmental Protection Agency, and other government agencies. SEEDIS is used both as a development testbed for computer science research, and in selected applications.

A major task of SEEDIS is the integration and organization of data from diverse sources. Used primarily by universities and government agencies, SEEDIS fills a need not met by two other kinds of systems available in the private sector: time series financial systems used for modeling and predicting economic trends, and small-area demographic systems used to access census data for market site analysis [2].

On the average, SEEDIS is used about 500 times per month. Usage is equally divided between development and applications. The Populations at Risk to Environmental Pollution (PAREP) project, which is concerned with relationships between human health and environmental pollution, provides and uses data on mortality, cancer incidence, socio-economic characteristics, and air quality [3]. 1980 Census reports being produced for the Department of Labor will require incorporation of most of the 1980 Census of Population and Housing, bringing the size of the SEEDIS database to about 50 gigabytes (500 tapes at 6250 bpi) [4].

SEEDIS data currently available to the interactive user include 350 million individual data values on disk and over 5 billion data values on a tape-based mass storage system. Data are available for about a million distinct geographic areas. These include eighty different types of geographic entities (e.g., states, counties, census tracts, enumeration district/block groups, etc.).

The size of SEEDIS databases, financial constraints, and the need for local control over data stored at dispersed geographic locations prompted development of techniques for data retrieval and display in a distributed computing environment. SEEDIS meets the needs and resources of small groups in the research community who can afford a small computer but not the resources required for on-line storage of large databases, nor the costs of timesharing on a large mainframe computer. SEEDIS software is in the public domain; it runs in the standard DEC (Digital Equipment Corporation) VMS operating system on a VAX 11/780 computer. To access SEEDIS databases at LBL (Lawrence Berkeley Laboratory), DECNET hardware and software are required.

2. Initial Network Implementation

SEEDIS operates in a homogeneous network of DEC VAX computers and uses standard DECNET facilities. The network presently comprises some 50 minicomputers. There are currently nine VAX-11/780's running SEEDIS. These are located in the San Francisco Bay area, the state of Washington, Washington DC, and North Carolina. Program modules, area and data definition files, and geographic base map files (about 75 megabytes) are stored at each SEEDIS site, or "node." Selecting (i.e., specifying for retrieval) or displaying data (e.g., mapping) does not involve network access, so response time depends only on the local system load and the speed of the user's terminal connection.

This work was supported by the Office of Health and Environmental Research and the Office of Basic Energy Sciences of the U.S. Department of Energy under Contract DE-AC03-763F00098; and the Department of Labor, Employment and Training Administration under Interagency Agreement No. 06-2063-36.

2.1. Distributed Data Operations

After the user has specified data selections, SEEDIS automatically extracts the requested data values from local and remote files, copying them into a self-describing file in the user's working space. Standard DECNET facilities automatically provide shared access to archived data files (about 1 gigabyte) on disk packs mounted on two of the nodes in Berkeley. Except for response time, the difference between retrieval of locally-stored and remotely-stored data is not apparent to users.

DECNET naming conventions automatically permit transparent access to remote files without additional programming effort. For example, lblg::dba0:[mydir]xyz.dat is a file in directory "mydir" on disk drive dba0 on node LBLG. Since data are stored on a particular disk pack and not a particular drive, SEEDIS maintains tables specifying the name of the disk pack on which database is installed, (e.g., SEEDIS005). The VMS operating system automatically assigns logical names to locally mounted disk packs, so data can be directly accessed by disk pack location, (e.g., disk$seedis005:[mydir]xyz.dat).

Special software was written to extend the standard DEC capabilities to remotely mounted SEEDIS packs. For example, whenever SEEDIS is invoked at any node, a background process searches the network for disk pack SEEDIS005; if it is found on drive dba0 at remote node lblg, a local system logical name assignment is established to translate disk$seedis005 to lblg::dba0:. If the pack is not found, any previous assignment for disk$seedis005 is canceled.

Disk packs SEEDIS001 through SEEDIS005, containing on-line SEEDIS databases, are located at LBL and can be mounted by an operator on either of the nodes LBLG or LBLH. A program DSCHED, which can be invoked from any node, allows remote users to easily determine when a disk pack will be mounted, or to request future mounting.

2.2. Initial Implementation Limitations

The initial 1979 SEEDIS network implementation had several limitations. First, even for small requests, data extraction took 20 to 30 minutes for remotely stored data, as compared to 2 or 3 minutes when data were stored locally. The difference was due to overhead in underlying DECNET remote file access protocols, which were not well understood at the time the SEEDIS data extraction module was written. Second, only a small fraction (about seven percent) of all SEEDIS databases could be stored on disk packs and an even smaller fraction could be on line at any given time. Data which had originally been stored on an IBM photodigital mass storage device now reside only on tape. In the absence of another low cost mass storage device, new mechanisms were necessary to access the large amount of archival data. Finally, the original implementation did not provide for automatic updating of SEEDIS system tables on data locations across the network, so changes required intervention by a central database administrator. While this was tolerable initially, it was clearly preferable to give each node independent responsibility to alter physical storage locations of its own individual data sets in a way that could be automatically communicated to other SEEDIS nodes.

3. Distributed DBMS Enhancements

During the past year, a number of improvements have been made to overcome limitations of the initial network implementation. They include mechanisms for: local caching of data files to improve efficiency; linking to an automatic tape library (ATL) to make larger volumes of data accessible; node independence to facilitate automatic sharing of data among autonomous SEEDIS installations without the need for central control; improvements providing robust operation despite unreliable network connections; and automatic recording of all cache transactions for subsequent statistical analysis.

3.1. Caching

In order to speed up access to frequently-used data and to provide an automatic mechanism for allocating scarce on-line storage space to the most frequently-used data, a simple system of caching was introduced. Archived files containing data required by the user are temporarily copied in their entirety to a disk cache at the user's local node. Archived data are partitioned so that no single file occupies more than a small fraction of the total cache. Files remain in the local cache for shared use until the space is needed for a more recent request. Every file is marked with the date and time of last access; least recently used files are removed first. Each file's lifetime depends on its utilization and the size of the cache, which is set by the local system manager.

Precautions are taken to prevent deadlock or thrashing: (1) a user request is immediately rejected with a message if the data request will exceed the total available space in the cache; (2) user requests are completely processed one at a time; (3) recently requested or used data have a guaranteed minimum lifetime of several hours in the cache, regardless of the number of pending cache requests; (4) a safety margin of about 2000 blocks (one megabyte) is maintained for necessary housekeeping functions.

All cached files copied from archive locations reside in a "temporary" cache subdirectory. All cache updates are accomplished in batch mode by a pseudo-user CACHE. The date of last access of each file (plus a constant increment) is automatically maintained by the VMS operating system. Another portion of the cache consists of small "permanent" files which are periodically updated but never deleted. These files contain pointers to information at other nodes.

This caching scheme is largely transparent to SEEDIS users, but it has involved an important enhancement to the user interface. Following standard SEEDIS procedures for data selection, the user defines a geographic scope and level (for example California by county) and then selects desired data elements from one or more on-line data dictionaries. After data selection is complete, the user types "extract" to append the data values to

his/her working data set.

In the new caching implementation, the "extract" command first automatically copies entire archived files to the cache if they are not there already, and then extracts selected data from the cached files to the user's working directory. If the required data are not already in the cache, the user is warned to expect a delay. The user may choose either to wait or to put the process into the background by typing an interrupt character (control-Y). The user then can type "show" to check the status of the cache request, "cancel" to cancel the request and begin another unrelated SEEDIS task, "continue" to complete the requested extraction as soon as caching is completed, or "quit" to leave SEEDIS. In all cases the background caching process proceeds to completion. Re-entering SEEDIS an hour or two later, the user can extract the requested data without delay.

If the requested data reside on the automatic tape library (ATL) at LBL (see below), the cache request requires access to BKY, the Lawrence Berkeley Laboratory computer center operating system. The user is prompted for a BKY account number and password, if not already specified in the user's login command procedure. Interactive help is available for the new user who needs to open a new BKY computer account.

3.2. Automatic Tape Library Mass Storage

The initial implementation of SEEDIS on CDC computers in the mid-1970's made use of an IBM photodigital storage device, the "chipstore." When IBM discontinued support for that product in 1979, SEEDIS databases were moved to a tape-based mass storage "gettape-stotape" system (GSS). This system, developed at LBL, implemented a self-describing UNIX-like directory structure for tapes and optionally makes use of an Automatic Tape Library (ATL) connected to the CDC machines.

When SEEDIS was initially reimplemented on the Distributed Computer Network VAX's, there was no link to the ATL. Selected databases were installed on disk for the initial implementation. At present, installed data occupy 1 gigabyte on five disk packs. The 1979 network implementation accessed only files on disk packs mounted at nodes in the network. In order to access data on tape, the tape had to be manually mounted, copied to disk, and installed in SEEDIS, a time-consuming and labor-intensive process. Although SEEDIS tapes contained much useful data (including most of the 1970 U.S. Census), they were virtually inaccessible. SEEDIS use did not justify the number of disk packs, let alone disk drives, required to keep the data on line.

With anticipated arrival of 1980 census data, there was a need for low-cost, moderately quick access to mass storage. Although optical disks had seemed a likely answer in the late 1970's, that technology was still too costly and unreliable. In order to fill this need, the SEEDIS project proposed a network link from the VAX machines to the Computer Center's CDC computers, in order to access and make use of the Automatic Tape Library and its GSS mass storage tape file system.

Two-way communication with the ATL is accomplished by programs BKYSUBMIT and BKYCLAIM, which are installed on every SEEDIS node. BKYSUBMIT and BKYCLAIM use DECNET to talk to a special network node DGATE, which in turn communicates with the ATL over a high-speed hyperchannel link. The 1983 SEEDIS network implementation includes an interface to BKYSUBMIT and BKYCLAIM, including proper handling of the various errors that can occur. As a result, low-priority SEEDIS data are now gradually being moved to tapes on the ATL, freeing valuable disk space for more important files and caching.

3.3. Node Independence

One of the most serious drawbacks of the 1979 implementation was the difficulty of modifying archived data files. Every node had an identical copy of program modules, database lists and data dictionary files. Files at every node had to be modified if any changes were made to publicly installed data files. Obsolete data files could not be removed until new software and data dictionaries were installed on every SEEDIS node, a time-consuming process even with only nine nodes. With additional SEEDIS nodes planned for 1983 and beyond, a better solution was required.

One of the guiding principles of the 1983 SEEDIS network implementation has been node independence. Every node should have the ability to install its own data locally, which it may optionally share with other nodes on the network. When a data file is installed, modified, or removed at any node, new information must automatically propagate to every node that has access to that file. The procedures for installing data must be simple and robust enough that only minimal consultation will be required from LBL staff.

The 1983 implementation allows data to be installed at any node, whether or not that node is connected to other SEEDIS nodes on the network. Optionally, the installed data may be flagged for public access, in which case the data become available to remote users as soon a network connection is established. The existence of data is made known to other users through a summary data base directory, which may be printed off line or browsed on line. A copy of the on-line directory is maintained at every node as described in the following example.

Suppose a user at the ETADC node (in Washington, DC) installs or modifies a public-access data file. With the permission of the local system manager, s/he uses documented installation procedures to automatically modify certain files in the local subdirectory seedis/etadc. This portion of the file system contains all ETADC node-specific SEEDIS information. In particular, it contains pointers to permanent archive locations of data and documentation installed by ETADC users. (The data may actually reside elsewhere, for example on the ATL in

Berkeley, California).

The installation procedure invokes a batch process at ETADC, which in turn causes batch processes to be initiated at every other presently connected SEEDIS node. The subordinate processes modify files in the "permanent" portion of their local cache. For example, node RX in Seattle has a subdirectory cache/perm/etadc where it maintains current copies of small files describing ETADC-installed data (i.e., a copy of seedis/etadc from node ETADC). Conversely, node ETADC has a subdirectory cache/perm/rx where it maintains current copies of small files describing RX-installed data (i.e., seedis/rx at node RX).

When the network is down, there is no guarantee that the directory cache/perm/etadc at RX is a correct copy of seedis/etadc at ETADC. If the RX network connection is down at the time ETADC data are installed, that information is kept in a small file at ETADC, and SEEDIS periodically resubmits the same batch update request (once a day until successful). In addition to the broadcast of updates, each node regularly (once a day) checks all other connected SEEDIS nodes to bring information from other nodes in its own "permanent" cache up to date.

Periodically (once a day) at each node, the information in all the subdirectories cache/perm/(anything) is merged and reformatted, to form a global database directory (also in cache/perm). This global database directory is the primary source of information at each node for SEEDIS users and data retrieval software.

The list of known SEEDIS nodes is itself a file which is automatically maintained at every node. For example, a file in seedis/etadc at ETADC identifies ETADC as being a public-access SEEDIS node. When SEEDIS is installed at ETADC, it attempts to broadcast that fact to every node on the network; those which have installed SEEDIS automatically receive and record the information in their directories cache/perm/etadc. Even if ETADC is temporarily disconnected, it is remembered as a SEEDIS node in future broadcast attempts from other nodes. If SEEDIS is deinstalled at ETADC, the information is properly recorded at each node the next time it achieves a network connection with ETADC.

3.4. Robustness Considerations

The caching software needs to be unusually robust to cope with a still unreliable hyperchannel link and DECNET phone connections that may operate only a few hours a month. On several occasions when the hyperchannel was inoperative for an extended period, the requested data were automatically and correctly put in the cache when the link was restored three weeks later. Even such a delayed response is valuable to certain classes of remote users, provided the data are certain to arrive sooner or later without further attention. Users do not have to remain on line waiting for the data to arrive. Once in the local cache, data used with some regularity are likely to remain available for months or longer.

Another aspect of robustness concerns the ability of the system to correctly recover from power failures, system crashes, scheduled and unscheduled shutdowns, VMS system updates, and well-intended but incorrect actions by system managers. In general, the contents of disk files are the only reliable records left by an interrupted job -- batch queues may not survive system updates or system crashes. Each SEEDIS node maintains a record of a pending job it expects to find in the batch queue of every other SEEDIS node, together with the password required to resubmit that job if necessary. This job (which runs once a day at each SEEDIS node) performs routine maintenance operations and keeps alive its "clones" at all other connected nodes. As the job runs only a few minutes a day, the likelihood of its being removed from the batch queue (due to a crash while it is running) is small. The likelihood of such a disaster affecting every node simultaneously is negligible. In other words, the system becomes more robust as more nodes are added (like the brooms of the "Sorcerer's Apprentice!"). Once started at a node, it can be permanently turned off only by a deliberate action of the system manager, for example by deleting critical files or removing the login privilege of the pseudo-user CACHE.

4. Recording of Cache Transactions

Since January, 1983, transaction records of every cache request have been continuously recorded in a compact machine-readable form. Usage patterns are being statistically analyzed to isolate bugs and improve efficiency.

Between January and June, 1983, the caching mechanism was continuously tested via daily automatic submission of randomly generated requests. Two nodes connected via DECNET shared a common cache on a single disk. Files were routinely and correctly cached from the ATL; delays varied from 20 minutes to 20 days depending on the state of the hyperchannel link. Fewer than 1 percent of the requests failed, in all cases due to hardware error. On only three occasions did the system fail irrecoverably and require intervention -- twice when hyperchannel hardware malfunctions caused the cache to overflow, and once when disk hardware errors caused the batch queues of both nodes to be simultaneously destroyed.

Under normal day time load conditions, a typical small request involving the ATL takes about an hour -- 5 minutes to formulate and submit the request, 20 minutes in batch queues, 10 minutes to read the tape, 20 minutes to put the data in the cache, and 5 minutes to copy the requested data from the cache. Subsequent requests for the same data would require only 10 minutes -- 5 minutes to formulate the request and 5 minutes to extract the data. Some of these times will be reduced in the future by improving the efficiency of the software.

5. Conclusions

Major enhancements have recently been implemented to permit efficient and robust access to distributed data in SEEDIS. Specifically (1) an automatic caching mechanism provides local shared access to user-selected subsets

of SEEDIS databases; (2) automatic access to 50 gigabytes of archived data is achieved through a hyperchannel link to an automatic tape library; (3) data can be independently installed, modified, or removed at any node, with all changes automatically recorded in copies of a global database dictionary at every other node; (4) every node is responsible for initiating periodic housekeeping functions at every other node, so that the whole network is much more robust than any individual node; (5) a continuous log of every cache transaction is being recorded for statistical analysis. So far, caching has been implemented for only one SEEDIS database -- a portion of the 1980 Census that was too large to reside on disk. During late 1983 and early 1984, the mechanism will be implemented for most other major SEEDIS databases including most of the 1980 Census. Distribution of a new version of SEEDIS in 1984 will give remote users automatic access to a vastly increased database, with no increase in local disk storage requirements. At the same time, remote users will be able to install their own SEEDIS databases and make them mutually accessible to other SEEDIS nodes.

References

1. McCarthy, J. L., Merrill, D.W., Marcus, A., Benson, W. H., Gey, F.C., Holmes, H., and Quong, C., "The SEEDIS Project: A Summary Overview of the Socio-Economic, Environmental, Demographic Information System," Lawrence Berkeley Laboratory Report, PUB-424, Rev. May, 1982.

2. Merrill, D. "Overview of Integrated Data Systems: Context, Capabilities and Status," Lawrence Berkeley Laboratory Report, LBL-15074, October, 1982. In Proceedings of the 1982 Integrated Data Users Workshop, Reston, VA, October, 1982.

3. Merrill, D. and Selvin S. "Populations at Risk to Environmental Pollution (PAREP): Project Overview," 1976-1982, Lawrence Berkeley Laboratory Report, LBL-15321, December, 1982. Included in "An LBL Perspective on Statistical Database Management," H. Wong, editor, Lawrence Berkeley Laboratory Report, December, 1982.

4. Department of Labor, Employment and Training Administration and Lawrence Berkeley Laboratory. Report 1: "Population Characteristics: 1980 Census of Population," Lawrence Berkeley Laboratory Report, LBL-14636, April, 1982. Report 2: "Employment and Training Indicators: 1980 Census of Population," Lawrence Berkeley Laboratory Report, LBL-14637, April, 1982. Report 3: "Social Indicators for Planning and Evaluation." Report 5: "Equal Employment Indicators." Three additional reports are in preparation.

An Integrated Research Support System for Inter-Package Communication and Handling Large Volume Output from Statistical Database Analysis Operations

by

G.D. Anderson and Tim Snider
McMaster University
Barry Robinson, SIR INC
Jerry Toporek, BMDP Statistical Software

Abstract:

This paper describes work underway to develop an integrated research support system designed to link together into a unified system a generalized DBMS, a relational database query system, statistical packages, a graphics system, text editors and a generalized screen oriented output handler.

The work is being carried out on the Hewlett Packard HP9000 32 bit micro computer system under HP-UX; HP's implementation of the UNIX™ operating system. The intent of the project is to provide a unified environment designed specifically for the researcher.

1. Introduction:

Most longitudinal studies require that complex hierarchically related data be collected, maintained and updated as the study progresses over time. Statistical analysis, reporting and graphical display on the other hand, require only simple rectangular (or flat) files. A package that is suitable for managing a complex database will not likely provide all of the statistical and graphical tools needed for analysis. To manage databases, perform statistical and graphical analysis and to conduct necessary updating and reporting tasks, the researcher needs convenient access to more than one single program package. This paper deals with the following aspects of the use of multiple systems in statistical database analysis:

- A description of the set of computer facilities needed by a researcher working with Statistical databases.
- A discussion on how most of these needs can be met using a combination of existing packaged systems available through software vendors.
- A description of additional facilities needed and a proposed approach to providing those facilities.
- A description of a menu driven user interface system designed to integrate access to and use of these various systems.

2. User Requirements for Statistical Database Analysis

In this section, we will attempt to justify why we feel the following set of facilities are required for effectively working with statistical databases:

- Access to a comprehensive DBMS for database creation, maintenance and administrative functions.
- An easy to use relational query language retrieval system with which to produce simple flat file units of analysis from the potentially complex structured data being managed by the database management system.
- Access to at least one comprehensive batch statistical package for large scale model building and hypothesis testing.
- Access to an interactive statistical analysis and display system for immediate investigation of smaller data sets.
- Linkage to a flexible interactive graphics package system through which data can be displayed graphically.
- An output management system by which the large volumes of listings generated in the process of statistical database analysis can be viewed, annotated and modified in preparation for listing or archiving.

2.1 DBMS Characteristics:

Three types of database model are discussed when referring to database management systems; the relational model, the hierarchical model and the network model.

The relational model is the simplest conceptually. Each type of record collected in a study is viewed, in the relational model, as a simple table (relation). The record variables make up the columns of a relation and the obsevations form the rows. As many relations are formed as are necessary for the different types of records collected in the study. For example, a clinical study might have demographic records, initial assessment records, follow-up records and final assessment records. Any interrelationships which may exist between individual relations are retained in the values of variables included in the relations. For example, each relation in the clinic example would have a variable containing the patients unique study id. Variables in one relation possesing a relationship with one or more variables in other relations are called candidate keys. Any desired relationship is 'realized' only at the time that a particular retrieval is requested. The relational model, through a well defined relational

algebra, provides very powerful rules by which relations may be joined on candidate keys during a retrieval to create new relations or desired units of analysis.

The hierarchical and network models differ from the relational model in the sense that they generally impose a very rigid structure on the data records. Whereas the relational model allows the user to choose the relationship desired at the time a retrieval is requested, the hierarchical and network models require the user to choose a specific set of relationships to be operationalized as part of the original design. Systems based on these models then physically implement the chosen design in the form of index keys, inverted lists or even chains of pointers actually embedded in the individual records. Databases implemented using a hierechical or network system have the advantage of highly efficient access to records and sets of records in the order defined in the design. These databases generally suffer the disadvantage, however, that they do not retain the flexibility of the relational model for easily accessing records in other ways not anticipated in the original design.

Database management needs for statistical databases require aspects of all three models as we will discuss in the next two sections.

2.2 DBMS Requirements for Building and Maintaining a Statistical Database:

The fixed structure of a specific hierarchical or network model is extremely important for maintaining an ongoing database. The database administrator needs access to a dictionary driven DBMS which supports a defined structure, which then enforces database integrity, provides for efficient interactive input and updating of records and maintains the database as an integrated whole.

Building and maintaining a statistical database requires a DBMS which provides:

- The ability to implement potentially complex hierarchical and network data structures which are capable of modeling the real nature of the data being collected.
- A centralized schema (data dictionary) facility within which data characteristics (meta data) as well as integrity constraints can be defined.
- Both interactive data entry access and volume batch data entry and modification facilities.
- Security provisions, database integrity checking and database recovery facilities.
- Report generation, descriptive statistics and tabulation procedures.
- A comprehensive set of maintenance utilities for unloading, reloading, subseting, merging, recovering, restructuring and transporting the database.

2.3 DBMS Requirements for Statistical Analysis and Graphical Display:

With the availability of a comprehensive database management system to handle the capture, editing, updating, reporting and general maintenance of a complex set of data, the researcher's primary concern is the production of flat files for analysis. Although some file management capabilities are being added to statistical analysis systems, they do not provide a full range of capabilities provided by a DBMS.

If the DBMS has a 'simple to use' query retrieval system linked into the database, the researcher can accomplish analysis most easily by creating flat files containing necessary units of analysis directly from the database as required. The resulting flat files can then be passed to the statistical or graphical analysis system where the desired analysis can be done. This approach has the added advantage that the centralized schema maintained within the database contains extensive meta data which can be passed to the analysis system with the retrieved units of analysis. This eliminates, for example, the data description and manipulation step required by statistical systems and allows the user to go right into statistical analysis.

The primary difficulty with retrieving analysis files directly from a database has been the inability to view a database relationally which was initially structured hierarchically or as a network. Clearly, the database administrator needs the structured model while the researcher needs the flexibility of the relational model.

In sectioon 3, we will review the SIR database management software package. The SIR system allows the user to logically impose multiple hierarchical structures with any necessary interconnecting network access paths on a set of relations. This structured model is maintained through independent index files which provide direct keyed access to any record in the database. This logical structure provides the rigid model needed by the database administrator for building, updating and maintaining the ongoing database. To meet the needs of the researcher, the SIR software provides a relational query system which allows the user to completely ignore the structured model and to treat the database purely as a set of relations.

A second difficulty with retrieving analysis sets from a database has been the fact that the user must interact independently with several separate program packages. In addition to moving manually between various systems, the user must define the various command files

separately for each system. This process often means moving repeatedly in and out of a system text editor and much interaction with the computer operating system. A major emphasis in this project will be the provision of a smooth, automated, menu driven user interface between separate systems.

2.4 Output Handling:

The use of a statistical analysis package often results in large volumes of output. Such large output can not easily be viewed and comprehended as it is being produced. The output from batch oriented packages is also often wider than the standard 80 column width of most computer terminals. Yet it is not always cost (or time) effective to print the output to hard copy. Very often the analysis may only be preliminary. A quick look at several results is sufficient to allow discarding an initial output and proceeding on to the next analysis.

The researcher needs a flexible system for terminal screen handling of large files of output. This is particularly important when computing at a distributed microcomputer workstation which may not include a local high speed printer. A comprehensive output handling system should be considered as a necessary part of the integrated research system. The following is a list of the characteristics we feel are required in an output handler program:

- The ability to browse sequentially forward and backward through a file and to window sideways when the length of lines exceed the screen width.
- A facility for marking locations in the file so that they can be recalled at will.
- A comprehensive pattern searching capability.
- The ability to 'window' together side by side on the screen parts of the file from different locations for comparison purposes.
- A facility for annotating the output and adding notes which become part of the resulting hard copy.
- A flexible capability for cutting and pasting a listing in order to eliminate unwanted output.

3. Available Packaged Systems which Can be Used as Components of an Integrated System:

This section describes existing packages available to include as components in an integrated statistical database analysis system. These include the SIR research database management software(1), the Minitab interactive statistical system(2), the BMDP(3) and SPSSX(4) large volume batch statistical systems, Hewlett Packard DSG graphics package(5) and the vi full screen editor(6).

3.1 Database Management:

The SIR software system is probably the most comprehensive set of database management facilities currently available that is especially designed for the research environment. The system presently runs on 12 seperate computer systems and is highly portable to other systems with 32 bit processors and virtual memory.

The following is a description of the main features of the four seperate packages which currently make up the SIR software: SIR/DBMS, the SIR database management system; SIR/FORMS, a full screen data input, modification and viewing system; SIR/SQL+, a relational query language retrieval system and SIR/HOST, a host language interface system:

3.1.1 SIR/DBMS - A Research Database Management System:

The following is a list of the main features of the SIR/DBMS:

- A relational data structure on which one can superimpose any hiererchical, network and relational data model with multiple record types.
- An integrated data dictionary for establishing the database structure and for naming, labeling and documenting the database variables and record types.
- Extensive facilities for data quality control including checks for invalid and out-of-range data, tests for consistency between data items and special handling of missing and undefined data.
- Security levels for reading and writing variables or entire records.
- Interactive access to the data dictionary.
- A structured procedural retrieval language, consistent with the other SIR/DBMS facilities (data dictionary, data input and built in procedures), that enables the user to navigate the database with a minimum of programming. The retrieval language also provides facilities for direct database modification, report generation

and interactive terminal input to an executing retrieval program.
- A set of simple statistical procedures that operate directly on the summary records created by a retrieval. These include frequency distributions and histograms, descriptive statistics, scattergrams, line printer plots and simple linear regressions.
- A sophisticated tabulation procedure patterned after the TPL program produced by the Bureau of Labor Statistics. This procedure produces device-independent, camera-ready tables. Individual table cells can contain frequency counts, means, minimums, maximums, standard deviations, medians, quartiles and percentages. The user also has complete control over all output format options.
- A report generator which allows the user to produce simple reports with Column headings, breakpoints, sorting, totals, subtotals and formatting automatically produced. With more extensive user specifications, complex hierarchical and branched reports can also be produced.
- Direct system file creation for BMDP, SPSS and SAS. The system files created by SIR/DBMS can be used directly by these statistical packages.
- A complete set of database maintenance utilities including subseting, merging, restructuring, unloading, reloading and transporting facilities.

3.1.2. SIR/FORMS - Interactive Data Entry and Retrievals:

The SIR/FORMS subsystem is an integrated system for interactive, screen-oriented data entry, modification and retrieval. It permits the user to enter, retrieve, delete or modify data in a SIR/DBMS database.

The SIR/FORMS subsystem provides the screen designer with a comprehensive set of design capabilities to handle a variety of application requirements. Among these are:

- Security and activity control for individual or groups of users.
- Conditional execution of screens.
- Linkage of screens along hierarchical or network paths.
- Validation of input based on existing data in the database.
- Customized help and error message texts.
- A wide range of data verification capabilities.
- Automatic creation of Log and Journal files.

SIR/FORMS makes direct use of the SIR/DBMS centralized data dictionary to:

- Create default screens.
- Provide automatic data quality control.
- Provide automatic help screens for each data item.

In addition to data entry and modifiction, SIR/FORMS allows simple data retrieval and query functions without any programing by the user. These retrievals can be done by record key values, a range of key values, inverted list lookup or by a data value in a field. In the latter case, the operator simply types the search criteria values in the appropriate fields and SIR/FORMS retrieves the matching records and displays them one at a time. This can be considered as a type of query by example capability.

3.1.3 SIR/SQL+ - A Relational Query System:

SIR/SQL+ is an interactive relational query system that allows users to interrogate a SIR database using an English like relational language. SIR/SQL+ is an extended implementation of SQL (IBM's Structured Query Language). In addition to providing the full SQL, SIR/SQL+ can take advantage of the existing structure of the database to perform retrievals with maximum efficiently. Frequently used queries can also be saved as part of the database.

The following examples illustrate the use of SIR/SQL+ to perform retrievals. The structured SIR/DBMS retrieval language code necesary to generate the same output is also shown for comparison:

- List the employee ID, name and salary of all managerial level employees (position level greater than 10):

```
SQL+:
   SELECT EMPLID NAME SALARY
   FROM DEMOGRAF
   WHERE PLEVEL GT 10

SIR/DBMS:
   RETRIEVAL
   PROCESS CASES
   .  PROCESS REC DEMOGRAF
   .    IFTHEN (PLEVEL GT 10)
   .      WRITE NAME SALARY
   .    END IF
   .  END PROCESS REC
```

```
END PROCESS CASES
END RETRIEVAL
```

- List the results of a 10% raise given to all the employees in departments 3,8 and 10. Sort the output by salary in descending order. If there are two people with the same salary, sort them by employee ID.

SQL+:

```
SELECT EMPLID NAME 1.1*SALARY
FROM DEMOGRAF
WHERE DEPT EQ ANY (3,8,10)
ORDER BY SALARY DESC EMPLID
```

SIR/DBMS:

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC DEMOGRAF
.    IFTHEN (DEPT EQ 3 OR 8 OR ID)
.      MOVE VARS EMPLID NAME
.        COMPUTE NEWSAL=1.1*SALARY
.        PERFORM PROCS
.        AUTOSET
.      END IF
.    END PROCESS REC
END PROCESS CASES
REPORT FILENAME=SALRPT/
       SORT=NEWSAL(D),EMPLID/
       PRINT=EMPLID,NAME,NEWSAL/
       NOTOTALS/
END REPORT
END RETRIEVAL
```

- For the whole company, find the average salary for each educational category (eg. 1=High School, 2=Some College, 3=BA or BS, etc.).

SQL+:

```
SELECT VALLAB(EDUC) MEAN(SALARY)
FROM DEMOGRAF
GROUP BY EDUC
```

SIR/DBMS:

```
RETRIEVAL
PROCESS CASES
.   PROCESS REC DEMOGRAF
.     MOVE VARS EDUC,SALARY
.     PERFORM PROCS
.     AUTOSET
.   END PROCESS REC
END PROCESS CASES
REPORT FILENAME=EDUCRPT/SORT=EDUC/
       NOTOTALS/PRINT=VALLAB(EDUC)
       MEAN (SALARY)/BREAK=EDUC/
END REPORT
END RETRIEVAL
```

The VALLAB function displays the meaning (value label) of the education categories rather than their codes.

In addition to these simple examples, SIR/SQL+ will handle full relational JOIN queries in which data from two or more relations (SIR/DBMS record types) is joined.

3.1.4 SIR/HOST - A General Host Language Interface:

SIR/HOST provides access directly into a SIR database from any standard high level language able to call a FORTRAN subroutine library. Through SIR/HOST, the user can take advantage of the storage, maintenance and retrieval capabilities of the SIR/DBMS database directly from within his own programs.

3.2 Statistical Systems:

The most popular and widely used statistical packages in North America today are BMDP, Minitab, P-STAT, SAS and SPSS. All of these except SAS have been converted to a large number of computer systems and will readily transport to any 32 bit computer with virtual memory. The recent work at SAS Institute on portable SAS promises to make that system much more broadly available in the future as well.

BMDP, PSTAT, SAS and SPSS are all similar with respect to their provision of a reasonably broad and comprehensive array of statistical procedures and the batch nature of their execution. All work on flat files of records on disc and can, therfore, handle indefinitely large data sets.

Minitab is decidedly different than all of the others in that it works with an in memory worksheet array and is completely interactive. This approach gives it great flexibility and makes the system very simple and easy to use. Minitab is, however, limited in its ability to handle really large research problems. Minitab is thus complementary to the other four systems mentioned above and should probably be included with whichever one of the batch systems is chosen in order to provide a complete spectrum of statistical capability to the researcher.

The P-STAT, SAS and SPSSX systems provide file management facilities in addition to statistical procedures. SAS and SPSSX also provide report generators and tabulation procedures. If the researcher is able to extract analysis files easily from the statistical database and is able to accomplish reporting and tabulation needs within the DBMS, these facilities in the statistical system become less important.

3.3 Graphical Systems:

As a result of great interest in the area of graphics in recent years, much work has been done to provide graphics software and the hardware to support it. Most computer vendors now provide comprehensive graphical software packages which support the graphics devices that they produce.

General packages (eg. the ISSCO and Precision Visuals Inc. products) run on a large number of computers and support multiple vendors graphics devices. In addition, both SAS and SPSS have graphics options available for their systems on some computers.

Whichever approach is adopted to provide graphical support to the research, a mechanism is needed for getting flat files from the database to the graphical system in much the same way this mechanisim is required for statistical systems.

3.4 Editors:

As mentioned earlier, access to flexible text-editing facilities is important to the researcher. Most computer systems have several text editors available and different users prefer different editors. In addition, many systems also provide some type of editor capability as an integral part of the system. SIR and P-STAT are examples. The SIR system provides a comprehensive and quite general line oriented editor. Often users wish to use the editor provided with a system like SIR because they become very comfortable with it and because it is identical on every computer on which they use the system. In this way, moving between computers doesn't require them to learn a new set of editor commands.

System specific editors, on the other hand, are often tailored to the computer and offer advantages such as full screen capability. Thus the user needs flexibility in using the editor of choice without paying a high overhead in terms of moving between systems and re-assigning files.

4. Development Considerations for the Output Handler

The output handler will allow a researcher to examine large output files in as natural and flexible a manner as he examines printouts. This is currently done in a somewhat ad hoc manner. For example, we manipulate large files using a number of tools supported by the UNIX operating system. These include the vi full screen editor as well as several programs that compare files, merge files, separate files, etc. The most obvious problem with handling output files in this manner is that these are all separate programs, each of which requires a different set of parameters and cryptic commands to perform its function. A less obvious problem is that all of these programs treat files in a very general manner, as a sequence of characters, or at most as a sequence of lines. Output from statistical packages always has some higher level structure such as tables, graphs, text, etc. and there is a great loss in efficiency if this is ignored.

To solve these problems, the output handler will be an integrated package which supports most of the functions of a full screen editor plus additional capabilities to allow multiple windows into multiple files, automatic comparison of windows, etc. In addition it will recognize the high level structure of an output file and use that to facilitate searching, comparing, and cutting and pasting of files.

In operation it will accept output as it is produced by a statistical package. It will build an index into the file based on the syntax, or structure of the output. The researcher can then direct his searches, comparisons etc. in terms of the structure of the output. For example, let us consider the steps necessary to compare two tables. First of all the tables are located by repeatedly executing the "next table" command. This is very rapid because the output handler has built an index of the tables in the file. Once the desired tables have been located, the display screen may be divided in two and one table put into each window for visual comparison. If the tables are too big to be completely displayed they may be scrolled side to side and up and down. Another situation might require the assembly of several columns from different tables. Again, the tables may be found and the columns selected quickly and easily because the output handler recognizes the structure of a table. The columns may then be assembled in a separate window and then saved as a file for later printing or review.

5. The Proposed Comprehensive Research Support System:

The Statistical Software Group (SSG) at McMaster University has, for five years, been converting and distributing statistical software (BMDP, Minitab, SCSS and SPSS) on the HP3000 16 bit minicomputer. The SSG has presently entered into an agreement with Hewlett Packard to convert and distribute statistical packages, database management software and research support tools on the

new HP9000 32 bit microcomputer running under the HP-UX operating system (HP's implementation of Bell Laboratories UNIX(7) operating system).

In addition to making the individual packages available, it has been decided that an overall user interface system is needed to help the user make effective use of these individual packages in concert.

The proposed system will provide the user with a menu driven user interface subsystem which will integrate the following component systems:

1. SIR/DBMS research database management system for statistical database definition, administration and maintenance.
2. SIR/FORMS for full screen forms data entry, modification and viewing.
3. SIR/SQL+ for relational database retrievals of units of analysis.
4. A choice of BMDP or SPSSX batch statistical systems for large scale statistical analysis.
5. The Minitab interactive statistical analysis and display system.
6. Hewlett Packard's DSG Decision Support Graphics package.
7. The vi full screen editor and the SIR/DBMS line oriented editor.
8. An output handling subsystem as described in section 4.

The user interface will allow the researcher to move smoothly between these component systems as well as provide the mechanism for handling the communication of data between systems via temporary files, pipelines and buffers. This removes one of the last machine specific burdens that the statistical researcher has traditionally had to face.

All of this will be made available on the HP9000 32 bit microcomputer system. The HP9000 computer ranges in size and power from a desk top workstation version with a single 32 bit CPU, an I/O processor and one megabyte of memory to a multi-user system with three 32 bit CPU's, an I/O processor and eight megabytes of memory.

Plans call for incorporating support for distributed 16 bit processor workstations into this research support system as well. In the distribution workstation environment, the SIR/DBMS database would likely reside on the central HP9000 where the SIR/SQL+ retrievals would be done. The resulting flat file would then move to the workstation where the researcher would do required statistical processing, graphical displays, output processing, etc. on the local personal workstation. The resulting output might then be sent back to the HP9000 for printing if a large output was required. Alternatively, one or more of the workstations on the network might have a printer or graphics plotting device for public use where required output could be produced.

References

1. SIR software is a product of SIR Inc., 820 Davis St., Evanston, IL 60204.

2. Minitab is a product of the Minitab Project, 215 Pond Laboratory, University Park, PA 16802.

3. BMDP is a product of BMDP Statistical Software, Inc., P.O. Box 24A26, Los Angeles, CA 90024.

4. SPSS X is a product of SPSS, Inc., Suite 3300, 444 N. Michigan Ave., Chicago, IL 60611

5. Dicision Support Graphics is a product of the Hewlett Packard Company, 3495 Deer Creek Road, Palo Alto, CA 94304.

6. The vi full screen editor is part of the UNIX™ system.

7. UNIX is a trademark of Bell Laboratories. For literature contact:
North America:
Western Electric Company Inc., Software Sales and Marketing, Guilford Center, P.O. Box 25000, Greensboro, NC 27420.
International:
AT&T International, Mount Kemble Ave., Route 202, P.O. Box 7000, Basking Ridge, NJ 07420.

INTEGRATING DATA AND DOCUMENTATION IN A MULTI-NATIONAL RESEARCH PROJECT: THE IEA SECOND INTERNATIONAL MATHEMATICS STUDY

Richard G. Wolfe
The Ontario Institute for Studies in Education, Toronto, Canada

With recent advances in the design of statistical software systems, data files with a hierarchy of observations can be stored with appropriate linkage of data and documentation and analytically aggregated and disaggregated. However, additional layers of organizational complexity need to be incorporated into the design of statistical databases when it is desired to integrate research studies that are parallel in sampling and instrumentation, but not identical. This occurs when a research survey is is repeated over several years or in several countries. Integration may be facilitated with a central documentation file of tables and text that capture inter-study variation and can drive analysis programs. This is illustrated in an application to a 20-country set of school mathematics surveys.

0. Introduction

Over the last several years, the designers of statistical software systems have begun to recognize that many data files, especially in the social sciences, have a structural complexity that exceeds a simple observation by variable matrix. An important example concerns a hierarchy of observations (e.g., students, teachers, schools) where each level may introduce new variables. In such a case, the organization of the statistical database must include linkage of data and data documentation between levels and should facilitate the analytic processes of aggregation and disaggregation.

It is suggested in this paper that sometimes additional layers of organizational complexity will need to be incorporated into the design of statistical databases. The complexities arise from the integration of research studies that are parallel in sampling and instrumentation, but not identical. For example, this would occur when a research survey is repeated over several years. The main example in this paper is of a set of very large-scale surveys in school mathematics that are being carried out in a coordinated but not centrally controlled fashion across 20 countries.

First, the nature of the data organization and documentation problem in the mathematics project is described. Some of the complications include hierarchical sampling and instrumentation, intended and accidental national variation, simultaneous item and respondent sampling, and multi-language translation. Second, the design objectives and constraints for database integration are discussed. It is especially important to capture the national variation in the surveys and to facilitate repeated but differentiated analysis. Compression and portability are also important. Third, the approach being taken to integration is described. This involves a central documentation file containing tables and structured text that define the internationally standard instrumentation and data coding as well as national deviations from the standards. Fourth, it is explained how the central documentation file is used as basic reference documentation and also as a source data file for programs that produce codebooks and statistical package setups, adjusting automatically to the specifics of each country's data. Fifth, consideration is given to what aspects of the system might be considered for incorporation in general-purpose software.

1. The IEA Second International Mathematics Study

The IEA (International Association for the Evaluation of Educational Achievement) is a consortium of governmental and private research institutes around the world that carries out a program of coordinated educational research studies in a variety of content areas. The IEA Second International Mathematics Study involves 20 of the IEA countries (some are jurisdictions within a country) in an analysis of mathematics teaching and learning at two levels in secondary education: one is called population A and corresponds to students in the school grade where most students are age 13, and the other is called population B and corresponds to students who are studying mathematics in the final year of secondary education. Some of the

countries are carrying out the study only for population A, so across the 20 countries there are a total of 35 surveys.

It is important to note that while the general design of the study is set in international meetings and defined in international instruments and operating manuals to be followed by the national centres, each centre is responsible for the detailed design and implementation, including translation, sampling, administration, and initial data coding and processing. Control mechanisms for assuring consistency and quality of the surveys include having international sampling referees, obtaining back-translations of the national instruments, and carrying out an elaborate audit of the data and reports submitted by each country.

These surveys are of large scale and national scope. For a given country and study population, there might be 150 schools, 300 teachers, and 7000 students participating. In most cases, stratified random cluster sampling is used, although the particular nature of the sampling stratification and hierarchy depends on the school system of the country. For example, some countries have distinct subsystems of schools (e.g., academic and vocational training); others have tracking within a single system (e.g., algebra and basic mathematics classes in comprehensive schools). Essential first aspects of the data documentation and storage concern the identification of the sampling structure, linkage of records from student to classroom to teacher to school to stratum, and recording of appropriate replication categories and sampling weights at each stage. Of course, these aspects vary from country to country.

Response information is collected at each level of the school hierarchy. Students take knowledge tests in mathematics and answer background and attitude inventories. Teachers fill out questionnaires covering their educational backgrounds, their attitudes toward mathematics, and their topic coverage and teaching practices. Teachers also provide reaction to the student test, indicating for each item whether the content needed to answer the item was included in classroom instruction; this is called the opportunity to learn (OTL), and becomes a variable in analyzing student achievement. School principals fill out a questionnaire concerning facilities, length of the school year, organization of the mathematics instruction, etc.

The core of the surveys is the mathematics test, and it is administered through a process of item sampling. The pool of test items is too large (180 at population A, 136 at population B) to administer to each student, without using an impractical amount of classroom time. So the pool is divided into stratified random forms, of which each student takes 2. (The details of constructing the forms differed between the two populations and, to some degree, between countries.) The assignment of forms to students is randomized in each sampled classroom. So for purposes of analysis, each test item is responded to by an inner-penetrating subsample of the total student sample, and each student takes a stratified random subsample of the items. Special statistical procedures are needed, of course, in descriptive analysis of item and subtest response and in explanatory analysis of student achievement.

The intricacies of the multi-stage sampling design, of multi-level data instrumentation and collection, and of the item sampling process mean that the organization and documentation of each national survey is complex and extensive. From the perspective of international data processing, archiving, and analysis, the complexity is compounded by national variation, both accidental and intentional.

Accidental variation is bound to occur in a study of this magnitude. Mistranslations, misprintings, and misunderstandings of the operating manuals will contribute to variation. The international standards have improved and shifted over the several years of the study, and national studies have begun and ended at different times.

The more fundamental, intentional variation derives from the fact that each national survey is intended to be a good and practical research study for the purposes of national analysis and interpretation. One obvious national interest and point of variation is in the nature of the sampling stratifications, since a country will usually want to differentiate teaching practices and student performance between educational groupings of local definition and importance. For practical reasons, a country may need to reduce the sizes of the item samples--e.g., students might take 1 rather than 2 forms--or ensure that some kinds of items are given to larger samples of students. For political or pedagogical reasons, a country may need to delete some of the content of the

instruments. Or a country may need to add questions and test items to the instruments in order to capture essential local context. A major and internationally coordinated variation occured in population A in that for 8 of the participating countries, the students took the mathematics tests at the beginning as well as the end of the school year, and during the year, the teachers answered extensive questionnaires on specific content areas.

2. Design Objectives for Database Integration

There are two international data processing centres for the study: the Department of Education in Wellington, New Zealand, is responsible for the population B surveys and for the cross-sectional population A surveys, and the University of Illinois, U.S.A., is responsible for the longitudinal population A surveys--that is, the 8 with the extra topic-specific questionnaires and beginning-of-year testing. The data processing load is extraordinary. There are 35 surveys, each with at least 13 and as many as 25 different data collection instruments, with extensive information collected from 3 levels (student, teacher, and school) plus linkage and sampling information from 2 other levels (classroom and stratum). Each country submitted copies of the original instruments, with back-translations where possible, and detailed audits of national options, modifications, deletions, and omissions. In the course of the data processing steps of checking, cleaning, and merging, hundreds of data files and thousands of working versions are involved.

The management of this data processing operation is obviously difficult and worthy of careful attention as a problem in database management. But that is another story. We are concerned here with defining and implementing the output of the data processing. The final goal of the data centres is to produce a data bank containing all the response data and the corresponding documentation for the 35 surveys. This is immediately used to produce analyses for the international reports on the study, for which the international processing centres have major responsibility. The data bank also needs to be immediately distributed to author-analysts for other international analyses and back to the national centres for national reporting and regional comparisons. Finally, the data bank needs to be archived and made available for future secondary analysts. Since a study like this is only done once every decade or two, and since the data array is so extensive and rich, it is likely that the most significant findings will arise in reanalysis over the coming years.

The database design problem is to store the data and data documentation from the study in a manner that captures the details of each national survey and, at the same time, integrates the data and documentation in terms of the international plans and standards. The complexities of any one of the surveys tax or exceed the capabilities of current statistical database software. The integration across surveys requires a new kind of organization. Several design objectives should be met.

Integration. There must be a true integration of the documentation, in which the common, international information--variable names, codes, texts, etc.--are stated once, and national variations are presented as exceptions. This is necessary simply because of the size of the problem: the international coding documentation for each population takes about 100 pages. If this were repeated and modified for each country, there would be 3500 pages of documentation, and that would be impractical to generate and maintain.

Utility. The integrated documentation should be usable both manually and through computerized interpretation. Some analysts will want to read about the characteristics of the data and then devise their own procedures for accessing and processing them. Other analysts will want to obtain access to data files through their favourite statistical systems, and so the documentation and data must be made accessible to those systems. As will be explained later, an important capability is automatic generation and execution of analyses adjusted to the characteristics of each national survey.

Compression. The data files themselves should be kept reasonably small, without too much redundant linkage and filling. An earlier IEA study has a databank that requires 10 computer tapes, and this leads to unfortunate costs and administrative hassles (e.g., in customs) for copying and transporting. All the data from the current mathematics project will fit, in theory, on one standard computer tape.

Portability. The entire data and data documentation system should be reasonably portable across computing environments.

The international data processing centres must be responsible to 20 different national centres, with a wide variety of hardware and software facilities. Current and secondary analysts will also have particular computer constraints. The goal must be to deliver the data and documentation in a universally readable form, together with as portable as possible an arrangement for extracting information and feeding it into local systems.

3. Central Data Documentation File

Since no existing statistical database package can meet those design goals adequately, it has been necessary to improvise a system. This system is based on a central, integrated data documentation file, as illustrated in Figure 1. (Actually, one central file was established for each of the two populations, with equivalent codings in areas of overlap. The following discussion applies to either.) The central file was initially set up to contain the internationally standard information, and as the data development work has proceeded, it has had notations concerning national variations added to it.

In the interests of portability, the physical format of the documentation file has been kept very simple. It is an ordinary text file, with 80-character lines. There are about 6500 lines of international information, and now about 4000 additional lines of national notations. The content is a mixture of unformatted explanatory material, including the essential auto-documentation of the file itself, and formatted tables and structured text, which contain the documentation of the survey data and the national variations. These different kinds of information are set off with lines of asterisks, slashes, and minuses, partly to facilitate a human reader's scanning of the file, but also so that a computer program can easily locate the tables.

The central file begins with an introduction, containing a table of contents and an explanation of the general formatting conventions for the rest of the file. The main body of the file is divided into 6 sections, each beginning with a unformatted text explaining the purpose of the section and defining the specific format of its table or structured text.

Basic parameters and sizes. The table in this section is organized by country. For each country, there is one line for each data collection instrument (test, questionnaire, etc.). For example, one can determine here which parts of the mathematics test a country administered at the beginning and the end of the school year.

Definition of the response data layout. The actual response data for each country are stored separately, and the details of the storage scheme are not necessarily linked to the specification of this documentation file. For example, there may be independent distribution of the data files in the form of some popular package, such as SPSS. However, for purposes of the ultimate compression and long-term storage of the data, a storage scheme is described in the documentation file. The main emphasis of the present paper is on the documentation file, but some features of the data storage scheme are worthy of note. The most portable record format is used: 80 character records. All logical record organization and linkage is coded within the physical, 80-character records. Logical records have a special starting symbol (asterisk), and a record length indication. The data part of the record begins with a code indicating the level of the data (student, teacher, classroom, school, stratum, population) and special linkage keys that vary depending on the level. The linkage keys show which instruments are included and which units at one level are associated with which at another.

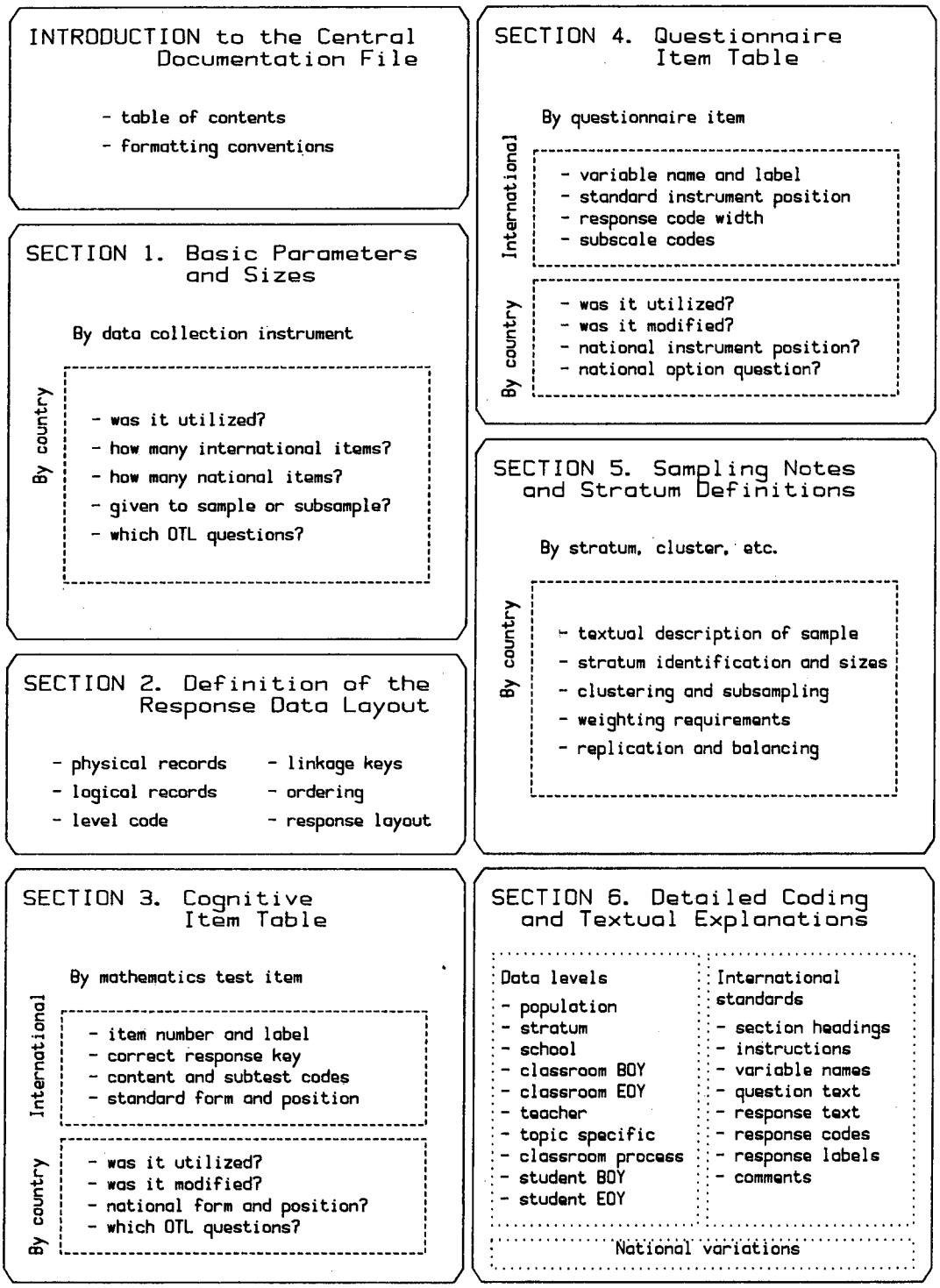
Cognitive item table. For each of the mathematics test items, there are two lines with international information followed by one line of national information for each country. For example, one can determine whether the use of an item in a given country followed the international standard and where the item appeared in the national test instruments. This information is generated from audits of the instruments and back-translations at the national and the international centres.

Questionnaire item table. For each of the items on the student, teacher, and principal questionnaires, there is one line of international information. This is followed by a line with coded and textual comments whenever a country deviates from the international standard, as determined through the national and international audits.

Sampling notes and stratum definitions. In this section information is included for each country on the sample design, stratification and cluster codes, stratum weights, etc. What information

Figure 1

Outline of the Central Documentation File for the IEA Second International Mathematics Study



needs to be stored for a country is determined by the international sampling referees.

Detailed coding and textual explanations. The final section of the documentation file contains the questionnaire texts from the international instrumentation together with substitute or supplementary texts to describe national variations. An extract (partly artificial) from the population A documentation file is given in Figure 2.

As can be seen in the extract, the international and national information are collated into a single structured text. The important structural conventions are exemplified in the extract:

- The text is keyed to the questionnaire item table through the variable names that appear here at the left margin. The extract contains material for variables SAREA, SENROLB, SENROLG, SAPOPA, SAPOPG, SMEET, SPOLFF, and SPOLPP. For each variable, the name and the exact text are given. For multiple-choice response variables, the response codes and texts are also given, together with short labels where appropriate.

- A hierarchy of text to surround variables is given by lines beginning with numbers. This includes the instrument names, section names, and other material that applies to sets of items. A numbered text applies until superseded by a text with an equal or higher number.

- Comments about national variation are enclosed in special brackets formed with the country's code number. In the extract, notations are made concerning a country number 97, called "Zembla". All material in the (+97...+) brackets applies only to Zembla. New or replacement variables and surrounding text for Zembla are defined in lines begining with (+97+). Lines that apply except in Zembla are marked with (-97-). The last four lines in the extract indicate that in Zembla a different categorization of calculators was made than in the other countries.

- The last part of the extract also illustrates a system for simplifying the description of partly parallel questions. A special name beginning with "*" is associated with a text string. The string will be inserted whenever that special name is encountered.

4. Use of the Central Documentation File

Even if no further computerized use is made of the central documentation file, it is providing a systematic framework for recording and preserving the information about the international standards and the national variations that is vital for on-going and future analysis. It replaces what in other studies have been haphazard, unintegrated collections of separate documentation files or integrated files with inadequate notation of true inter-survey variation. Several computerized applications make the central documentation file even more useful.

A program has been prepared to read the central documentation file together with a selection of one country and the specification of a set of questionnaire variables. The program produces a annotated codebook by carrying out these steps:

- The basic parameter and size table is examined to find out which instruments are included for the country.

- The questionnaire item table is examined to determine the labels, widths, and standard locations of the variables, together with any special notations entered for the specified country.

- From the last section, the detailed texts and codings are extracted, together with all indicated surrounding information. Specially bracketed material is included only when appropriate.

Another program goes a step further, producing not just the codebook, but also a data file containing the extracted variables together with a setup, including the labels, to read the data into a statistical software package. So far, the setup is restricted to SPSS, which is the package available in most of the national centres.

Figure 2

Extract from the Documentation File's Section of Detailed Coding and Textual Explanations

```
9 School questionnaire
7 Section A - to be completed by the school principal
SAREA Which of the following best describes the community served
   by your school? (+97 In Zembla, the national capital is
   counted in the 5th category+)
   /1 rural /2 suburban /3 urban /4 urban-suburban
   /5 inner-city metropolis (i.e., for cities with a total population
   greater than half a million) = inner city metro
(+97+)3 In Zembla, the enrolments are not differentiated between
   boys and girls.  The totals are given under the international
   variables for boys.
2 What is the total enrolment of full-time (or full-time equivalent)
   secondary students in your school?
SENROLB boys (+97 and girls+)
SENROLG girls
2 What is the number of population A students in your school?
SAPOPB boys (+97 and girls+)
SAPOPG girls

. . .

7 Section B - to be completed by the head of the mathematics department
   (+97 In Zembla, the information came from central records.+)
SMEET How frequently are meetings of the mathematics teachers held
   in the school? /1 never /2 less frequently than once a semester or
   term = infrequent /3 once a term or semester = once a term
   /4 once every month /5 once every two weeks = fortnightly /6 once a
   week or more frequently = weekly or more

. . .

*DEPTPOL Which of the following best describes your department's
   policy on the use by population A students of
*CALPOL calculators in the mathematics classroom?
   /1 no policy formulated.  Teachers allow use as they see fit =
   no policy /2 students are forbidden to use calculators in the classroom
   = forbidden /3 students may use calculators, but they are not provided
   by the school = permitted /4 calculators are provided by the school,
   but used only rarely in the classroom = provided low use /5 calculators
   are provided by the school and are used frequently in the classroom
   =provided used /6 question does not arise (e.g., calculators are not
   available to students) = no calculators
(-97-)SPOLFF *DEPTPOL 'four-function' *CALPOL
(+97+)SPOLFF *DEPTPOL non-programmable *CALPOL
(-97-)SPOLPP *DEPTPOL preprogrammed multifunction and programmable *CALPOL
(+97+)SPOLPP *DEPTPOL programmable *CALPOL
```

A potentially more significant kind of application involves running programs against the central documentation file to produce a series of processings, one for each national data file, adjusted to the characteristics of the file. For example, there is interest in the content area of achievement in algebra. In the cognitive item table for population "A", about 30 items are flagged as being part of the algebra subtest. The assignments of these items to test form and the positions in test form are given for each country. In addition, country-specific ratings are given of the adherence of the item to international standards. A program can:

- read that information from the central documentation file;
- determine according to rules which items to include for each country;
- determine the location of the items in the response data files; and
- go get the data, country by country, or prepare setups for some other program to do that.

The final output might be an item by country table containing the mean levels of performance, with indications of gaps where data were missing.

When it is noted that there are about 40 subtest areas to be examined for each of the 35 surveys, the need for this kind of automation is apparent.

5. Conclusions

The details of the method used to integrate the statistical database for the IEA Second International Mathematics Study are, admittedly, ad hoc and particularized to this project. But for multi-national studies of this sort, the approach represents an effort to reach a new standard for integration of surveys and for recognizing and recording the variations that exist between surveys. Such variation has always been present in earlier studies, but conventional data banks hide rather than reveal.

It would be convenient if this kind of integration and annotation could take place within a general-purpose environment. The experience in designing and using the current system suggests the importance of several features. First, since integrated studies, at least multi-national ones, will have to operate in a variety of computing arrangements, it is important that the basic central files be accessible in as neutral as possible a form. Some users of an integration simply want to read the documentation. Second, it has proved very useful to make a formal coding of the variable by study matrix of information concerning quality and application. This is effectively a database on variables. Initially, it forces a complete audit and evaluation of the data files; later, it makes automatically adjusted analysis possible. Third, until this kind of system is incorporated into high-level processing and analysis packages, attention must be given to the interface, and the current approach of producing setups has proved useful to analysts.

Paste -- A Tool to Put Application Systems Together Easily

Stephen E. Weiss, Pamela L. Weeks

U.S. Department of Labor, Bureau of Labor Statistics
Washington, D.C., USA

ABSTRACT

The collections of statistical packages, tabulation systems, and database systems that have grown up in the past 10 years constitute a significant advance in the tools available to the statistician. Unfortunately, each of these tools is limited. The developer of an application system using these tools is often faced with the difficult problem of stringing these systems together to get a complete application system. Two approaches have been applied to alleviating this problem. Arguments are advanced against each of these approaches. A new solution to the problem is proposed. The solution is a system called PASTE whose purpose is to Put Application Systems Together Easily. A preliminary design of this highly flexible and extensible data description and file transformation system is described.

1. THE PROBLEM

In the ideal world, an application designer at the Bureau of Labor Statistics (BLS) or some other statistical organization should spend nearly all of his time determining what he wants his computer system to do. He should then drop his specification, along with the data and a description of the data, into a super statistical system which will produce the specified outputs.

In the real world things aren't quite so simple. In the past 10 years, there have grown up a number of excellent statistical systems which are steps toward producing our ideal world. Systems such as PSTAT, SAS, TPL, SPSS, RAPID, SIR, ORACLE and INGRES are each steps toward the ideal super system. The systems vary in what they do, how easy they are to use and how portable they are. They do have two important things in common. They are all tools which are designed to make the life of the statistician easier. They are all limited in what they can do.

The statistician wishing to do a regression analysis or produce a nicely formatted table on an IBM mainframe would in general be ill advised to write his own PL/1, FORTRAN, or COBOL program. The statistical packages that exist will certainly produce the desired results faster and often at less cost. Even more importantly, the output produced by the statistical system is more likely to be correct that the output of the application program.

The problems come because of the limited capabilities of the existing systems. The Table Producing Language (TPL) will produce tables of almost any format if the input data is in an appropriate format, but it will not do a regression analysis. SAS will do the regression analysis, but if your data is hierarchical, you may have a lot of programming to do either to prepare your data for loading into SAS or within SAS itself. A system designer may find that one general system does nearly everything he wants for a particular job. He then must find a way to trick the system into doing those last few things. There are, for example, many very ugly and cryptic TPL jobs around BLS which were written to get results from TPL that the system was not designed to produce. In some cases the application designer cannot get a single general system to produce the results he wants regardless of his machinations. Then he must get his data out of one general system and into another. This often involves a fair amount of COBOL, FORTRAN, or PL/1 programming.

2. CURRENT SOLUTIONS

There have been two approaches which have been advanced to solve this problem of limited systems. One has been to create system interfaces. At the last meeting of the Workshop of Statistical Database Systems, we described the interfaces which allow TPL to directly access data stored using the RAPID or TOTAL database systems. We have also built an interface to move data from TPL into SAS. Others have created interfaces to move data from SAS into TPL and from ADABAS into TPL. There are probably several other TPL interfaces that we are unaware of.

The biggest problem with the individual interface approach is that it is an unending task. In general, interfaces, especially good ones, require a lot of work. There are just too many potential interfaces to devote the needed time to each of them. Further, there are certain important interfaces which will probably never be built. For example, there is a seasonal adjustment program which is used by some TPL users in BLS and Statistics Canada. It is very important to the work of these people. An interface between it and TPL is unlikely because the use of the seasonal adjustment system is not extensive

enough to justify the development of such an interface.

The second solution to the problem of not being able to do everything in one system is for one system to copy another system. For example, the developers of the SIR database system wanted an interface from SIR into TPL. It was technically very difficult for the SIR people to develop the interface themselves. SIR is not currently used in BLS, so we did not have a reason to build the interface ourselves. The SIR solution was to use the TPL manuals as a system specification and build their own tabling system to duplicate TPL. We haven't had the opportunity to try the SIR tabulation system but their manuals certainly do look familiar. The SAS Institute's PROC TABULATE was also strongly influenced by TPL.

This copy approach makes good sense, especially to a commercial software vendor. He has a more powerful and hence more marketable system. He also has total control over the entire system and so does not have the problems of maintaining an interface with a foreign system. Finally, by copying another system, the developer is relieved of the difficult tasks of requirements analysis and language design.

The user of statistical systems views this copy approach ambivalently. On the plus side, the copy approach may result in a more efficient system than the interface approach. Also, the copy approach can sometimes result in an easier to use system. Finally, the copier can modify the language being copied so that it looks more like the copier's language. The resulting similarity in user language should make request writing easier for the user.

The copy approach however does have its drawbacks. One is that it is very difficult to reproduce all of the functionality of another system. This is especially true if the copier copies only user language and not internal design. Invariably important features will be difficult to implement in the new system. So the user will end up with a subset. And, as luck will have it, he will eventually need to use features that the subset does not contain. Some BLS users of the SAS PROC TABULATE have found this to be the case. Thus the user may be forced back into the original system. Further, if the original system adds new features, the copier will have to duplicate the work in his copy rather than get the new features for free.

Of course the biggest problem with copying another system is that it requires even more work than building an interface. Thus again, important but not widely used systems like the seasonal adjustment system will not be copied. The community of developers of generalized statistical software systems is really quite small. If we spend too much of our time copying the capabilities of each other's systems, we will not be able to provide the statistical processing community with the new functionality and ease of use they need.

3. THE PASTE SOLUTION

We at BLS are currently pursuing a new approach to the problem of incomplete systems. The system which we have tentatively called PASTE will be a generalized interface system. The purpose of PASTE is to enable the application developer to Put Application Systems Together Easily. The application developer writes the various parts of his system in the languages of the higher level statistical packages and databases. He then uses PASTE rather than writing programs to paste together these parts into a complete application system.

To show what the goal of PASTE is, we will examine an application system. Suppose an application programmer wishes to take a sequential hierarchical file, load it into a RAPID database, extract some of the data, use SPSS to calculate some additional information from the extracted data, feed his data into TPL to table it and photocompose it and finally feed the result into a page makeup system such as ATL to insert it within the text of a book.

The application programmer first writes a program to split the hierarchical file into flat files for loading into RAPID. To do this he must write a programming language description of his data and programming logic to split the file into flat sequential files. Also the program must add key fields to the files to facilitate the eventual reconstruction of the hierarchical file. The programmer then must write a RAPID data description of the input flat sequential files plus a description of the data as it will exist in RAPID. Now he can use the higher level RAPID utilities to load his data into RAPID. There happens to

be a RAPID to SPSS interface but for the purposes of this example we will pretend it does not exist. Thus the user must instruct RAPID to dump out the desired subsets of his files. He must then write an SPSS description of the dumped data and use the high level SPSS language to calculate the additional information he needs. Notice that RAPID output and SPSS input facilities are flexible enough so that there is probably no need for a program between the two systems in our application. The system developer probably will need to write a program with its data description to recombine the flat files back into a hierarchy for his TPL request. Then he must write one more data description for TPL. Now the system developer is finally finished with data descriptions and programming languages since the TPL, photocomposition, and page makeup systems are nicely interfaced.

To do this straightforward application system, the application developer has had to describe his data six different times and has written two programming language programs. The multiple data descriptions are not only a waste of time, but also a significant source of errors. This is especially the case when the translation of data description from one system to another is complicated. For example, a field described to RAPID as FIXED DEC(p) where p = decimal digits, must be described to TPL as PACKED n where n is the largest integer <= p/2 + 1.

Using PASTE the application builder's job will be somewhat easier. He will begin by describing his sequential hierarchical file using the PASTE data description facility. This facility will allow him, if he chooses, to specify nearly all the data description he will need for his entire system. For example, he can specify the data type to be used when the data is in RAPID and the print labels to be used when the data is in a TPL table. The developer may add to or change his descriptions at a later time if he chooses.

In the high level PASTE language, the application developer will request that his data be split into a collection of flat files and that RAPID input and storage data descriptions be generated. He will then use the high level RAPID utilities to load his data into RAPID. At this point, if we again ignore the existence of the RAPID to SPSS interface, the application developer has a choice. He can either use PASTE to extract the subset of data from RAPID and prepare an SPSS data description or he can use the RAPID utilities to dump out his data. If he uses the RAPID utilities, he must provide PASTE with information about changes in the format of the data coming out of RAPID. Notice that the programmer does not need to provide an entirely new data description since most of the descriptive information in PASTE is still correct. The application developer now uses SPSS instructions to specify the additional processing he needs for his application. Since the resulting data is not created by PASTE, the application developer must supplement his PASTE data description with information about the new data. He then uses PASTE to specify the reconstruction of his sequential hierarchy and requests that a TPL data description be created to describe it.

What we have described is an ideal application of the PASTE system. In this application, a complete data description is provided only once. The only additional data description required is information about data transformations done outside of PASTE. Further, in this example, all of the required data transformations could be provided by the available statistical systems or by PASTE itself. In some real applications none of the statistical systems nor PASTE will be able to provide certain file transformation functions. In such cases programs in PL/1 or other programming languages will still have to be written.

4. OTHER USES OF PASTE

In the above example we saw how PASTE could aid in creating an application system by stringing together several different systems. Will PASTE help us when all of our processing will be done by a single system? The answer is often "Yes." PASTE will allow its users to put data into the ideal format for the high level system rather than make the system strain to process awkwardly organized data. Consider a TPL request for percent changes. Such requests are very awkward to write if the data is arranged as a collection of time series in which each record contains a year's worth of data as a 12 month repeating group. The problem is most acute when we wish to find the change between January of one year and December of the previous year.

We can use PASTE to transform each 12 month record into 12 separate records.

Each record will contain 2 months of data, the "current" month and the "previous" month. Note that each month's data appears twice in the new file, once as a current month, and once as a previous month. This redundancy is not dangerous since the transformed file is a temporary file to be used by TPL and then discarded. The resulting file with a format of 2 month repeating groups is ideal for a TPL percent change request. The required TPL request is now straightforward, as is the PASTE request which restructured the data. The system resulting from using PASTE and TPL should be much easier to understand and maintain that the system which uses TPL directly.

Actually PASTE can be of value even to the builder of an application who does not use any higher level systems except PASTE. Data files tend to be created in a form which is designed to be convenient for the data collection process. Often this structuring of the data is not ideal for the application which uses the data. The application developer can restructure his data by writing a special program. This is often unacceptable because another application may need the data in a different configuration. The usual solution is that the application developer just lives with the data in its awkward structure and writes a convoluted program to process it. This is frequently done without the application programmer even realizing that restructuring will make his program simpler. With PASTE the application programmer will have an easy-to-use tool for configuring his data in a convenient form for his application. We hope that the existence of this tool will not only make restructuring of data easier, but also will make the application programmer more conscious of the need to structure his data to facilitate his application.

5. PASTE MAKEUP

The PASTE system will consist of two main parts. The first is a very flexible data description module or DESCRIBER. The second is a data transformation module or TRANSFORMER. The two modules interact in several ways. When a transformation of the data is specified in the high level PASTE language, the TRANSFORMER must extract information about field locations and data types from the DESCRIBER, since this information is not explicitly present in the user's transformation request. This interaction between data transformation and data description is common to many high level language systems. The PASTE system also has a somewhat less common interaction between data transformation and data description. When a data transformation is specified, the PASTE system not only generates code to transform the data but also generates code to transform the data description so that it will match the transformed data. In general, the PASTE language will blur the distinction between data and its description. The exceptions to this will be the initial data descriptions plus the PASTE statements which explicitly request that data description be generated for use by an external system, e.g., a request to generate a PL/1 description of a file.

The user interface to the DESCRIBER will be primarily via an interactive menu and fill-in-the-blanks system. The user begins by specifying statistical systems and databases which will be used in his application system. PASTE returns a list of the different types of data description that may be used with these systems. The application developer chooses those that he plans to use. This information is used to construct a set of fill-in-the-blank forms. If the user already has a description of his data, he may be able to request that the DESCRIBER use this description to fill in some of the blanks for his new description. If he later decides to add more description, he may do so.

The data description will have a hierarchical structure. At the lowest level will be the data element or field. The description of the data element will include such things as field size, data type, values the element can take, and the print labels that will be associated with them. The next level up is variously called record, case, or observation. At this level, the user will specify the fields that occur in a record and their order. He will also specify structures such as repeating groups. Above this is information about files. This includes clustering and other ordering information which can potentially be used in estimating the cost of various transformations. At the highest level is information about the interconnections between files. At this level is the information about the pairs of fields which may be used for joins when the data is in a relational system. It is also the level for the information that TPL uses to produce heirarchical paths through a database. (See our

paper, "Must We Navigate Through Databases?" in the Proceedings of the First LBL Workshop on Statistical Database Management.)

The most important characteristic of the data description facility will be its flexibility. The system should make it unnecessary to keep information about data on scraps of paper or in our heads. To assure the flexibility the users will need, we will have to make the system easily extensible. The developers of PASTE and perhaps even sophisticated users of the system should be able to add new types of data description. To do this they assign a name to the new data description element. If the new element is a complex item such as a TPL print label, they will also specify a grammar for the element. Finally they may specify semantic checks for the individual productions of their grammar. This will enable the system to check for such things as the size and alphabet conventions for names used in a particular system. PASTE should also be able to detect and flag such errors as duplicate names. The goal of these checks is to attempt to assure that when a PASTE request is made to produce a data description for a system, the description will be error free.

The second major component of PASTE is the TRANSFORMER. Unlike the DESCRIBER which stores descriptive information, the TRANSFORMER does not store anything. The TRANSFORMER operates in one of two modes. In one mode the system operates as a record processor. It reads a record or group of records, does some calculations, and writes out zero or more records. As an optimization the system may read and write a block of records at a time, but this is not essential. This mode of operation can be used for simple tasks such as changing the data format of a field or splitting a hierarchical file into a set of sequential files.

In some cases, the desired transformation requires more complex processing. The system recognizes these situations and automatically switches to the information extraction mode. In this mode of operation, which is used by TPL, information is extracted from the incoming records and is temporarily stored. The information is stored in a condensed form which does not preserve the identity of the record from which it came. When all of the input records have been processed, the system constructs a new set of records from the stored information. The information extraction mode is more expensive than the record mode, but is also much more powerful.

The user interface to the TRANSFORMER will probably be a command oriented language. This might be implemented by selecting the appropriate words from a menu. Typical commands are: CHANGE COST TO PACKED DECIMAL; WRITE PL/1 DESCRIPTION OF PERSONS; SORT PERSONS BY FAMILY# THEN AGE.

6. PROBLEMS AND CONCLUSIONS

The biggest problem with the PASTE system from the user's point of view will be the description of data after it has been manipulated by systems other than PASTE itself. If the external system is what we earlier called a record processor, the specification of the changes in the record format will be relatively easy. Records may be split or joined, fields may be added, deleted, or rearranged, extra summary records may be created or data types changed, but the basic identity of most of the fields will be unaltered.

If the external system which changes the data operates as an information extraction system, the user may find it difficult to describe the changed data file by a specification of changes of the old data. In such cases the user should probably redescribe his data to PASTE as a new file. Using PASTE he will still find his job easier than without it since he should be able to use parts of his old data description in describing his new data. In particular, many of the large sets of codes and associated print labels will be reusable.

As we have seen, the PASTE system has the potential to greatly simplify the work of the application system builder. For this potential to be realized we must design and implement a system which satisfies two conditions. It must be significantly easier to transform files using PASTE than using a standard programming language. Further, PASTE must be usable with a large share of the database and statistical processing systems used by statisticians.

PIGAS

AN INTERACTIVE STATISTICAL DATABASE MANAGEMENT SYSTEM

WARTELLE M., KRAMAR A., JAN P., KRUGER D.

Département de Statistique Médicale

Institut GUSTAVE-ROUSSY
Villejuif (France)

ABSTRACT

PIGAS is an interactive statistical database management system whose main assets include logical data checking, data entry for meaningful variables only, three different types of missing value codes, non-rectangular data structure and user interfaces with BMDP, GLIM or specifically written FORTRAN programs.

The structure of the PIGAS language is the same as one passes from one phase of a research study to another, whether it be data updating, checking or analysis, thus requiring only a minimum amount of initiation in its use. It is particularly appealing since data verification is completely specified through variable names as they appear on the questionnaire. Instructions given in how to fill in a questionnaire can thus be directly incorporated in the data checking process, thus ensuring a good quality data base.

1. INTRODUCTION

PIGAS is an interactive statistical database management system designed especially for persons with little or no formal computer training. This general purpose package can be used by all personnel engaged in the study of a particular research project. For example by data managers responsable for data collection, data management, data editing and physicians or statisticians for data analysis.

PIGAS was developped for computer analysis of medically oriented research studies at the INSTITUT GUSTAVE-ROUSSY but may be used for other types of research projects.

Different functions of the system will be presented :

1 coding of missing values
2 describing variables whether unique or occuring several times in any data record
3 controlling the data by logical commands
4 updating the data
5 creating new variables as part of the file or temporarily during the analysis phase
6 analyzing the data by :
- statistical functions incorporated into PIGAS
- interfaces with BMDP, GLIM or a specifically written FORTRAN program.

The PIGAS user is guided by a menu which is displayed on the terminal screen.

2. PIGAS AND MISSING VALUES

The results of a particular research project depend on the quality of the questionnaire used for data collection.

It is essential to formulate clearly the questions and to provide a code for each answer to ensure that data will be collected in the same manner for every case of the study.

One should always use the same code for answers yes or no (for example, the answer yes is coded 1 and the answer no is coded 0) and a standard code for missing values whatever their type (date, value of a dose,...).

During the phase of data collection, certain variables may be considered as missing for three reasons :

- UNKNOWN

The information is unknown : the person interrogated refused to answer, a certain measurement was not made, ...

The information is lost and cannot be obtained. It is coded "UNKNOWN".

- HOLD

The information is not yet available but shall be communicated later. This information is coded "HOLD". This particular feature enables the data manager to introduce in the file all the variables of the questionnaire without having to wait until all the information is collected.

- NOT APPLICABLE

Some variables are meaningless for certain cases of the study. For instance, the questions :

- "Age at which you started smoking"
- "Last type of tobacco smoked"

are meaningless for non-smokers.

If these two questions are preceeded on the form by another question with answers yes or no, for instance :

"Have you ever smoked (yes, no) ?"

they will be considered as meaningless for non-

smokers.

PIGAS automatically codes these two variables "NOT APPLICABLE". This notion is an option offered when describing variables and shall be described in the next section.

3. DEFINITION OF TERMS

In PIGAS language, each questionnaire constitutes one CASE which is distinguished from all other cases by the value of a CASE-IDENTIFIER.

A questionnaire consists of several records which regroup all the information relative to the same theme.

Each case may have a variable number of such records which are called CHAPTERS.

For example : an identification chapter, a preliminary examination chapter, a treatment chapter, and a follow-up chapter in a medical research study.

Each question is called a VARIABLE. Questions relative to a part of the population are gathered in a PARAGRAPH preceeded by a LEADING VARIABLE, a question with a "yes" or "no" answer only. In the example of the preceeding section, the question :

"Have you ever smoked (yes, no) ?"

is the LEADING VARIABLE of the paragraph called "TOBACCO". The questions :

- "Age at which you started smoking"
- "Last type of tobacco smoked", meaningless for non-smokers, are the variables of the paragraph "TOBACCO".

4. DESCRIBING THE QUESTIONNAIRE TO PIGAS SYSTEM

PIGAS displays questions on the terminal screen to help persons who wants to describe a questionnaire.

To describe a chapter, the following information needs to be provided (an example is given in appendix 1).

- NAME - the name of the chapter used to identify it in the file (ex : "CHILD")
- "LIBELLE" - label used for the purposes of edition.
- OBLIGATORY (Y,N) - indicates if the chapter is necessary for every subject or not at the moment the case is created.
- REPETITIVE (Y,N) - whether the chapter may occur several times for the same case.

For example, in a study on Mothers and their Children, the chapter "CHILD" is filled in for each birth. The number of children may vary from one mother to another.

For the PIGAS system, the chapter "CHILD" is considered as REPETITIVE.

- NAME OF THE REPETITIVE CHAPTER IDENTIFIER - used for repetitive chapters only. These variables (up to 3) permit to distinguish one chapter occurence from another.

In order to distinguish each child from the next, the REPETITIVE CHAPTER IDENTIFIERS are the date of birth and the rank of birth (in case of gemellary births). Each child constitutes a unique occurence of the chapter "CHILD".

Double occurences are not allowed.

Occurences are stored according to the increasing values of the repetitive chapter identifiers.

- MINIMAL AND MAXIMAL NUMBER OF OCCURENCES (limits of which are 0 to 40).

Once the chapter is declared, the following information needs to be provided for each variable concerning this particular chapter :

- its NAME (up to 8 characters)
- its "LIBELLE" (up to 30 characters)
- the TYPE : E = "numeric" (maximum of 8 digits)
  F = "floating point" (up to F15.6)
  A = "alphanumeric"
  D = "date" (expressed in day, month, year or month, year or year only ; one can also indicate the century)
  L = "logical" (coded 0 or 1).
- LG : the number of spaces to be filled in for its coding
- MIN, MAX : its range of variation (minimale and maximale value)
- CRX : whether or not the missing value code "UNKNOWN" is allowed or not.
- NATURE : whether or not the variable belongs to a paragraph.

An example is given in appendix 2.

Variables in a repetitive chapter are declared only once no matter how many occurences for each case and are referred to by an index number. The description of variables or the structure of chapters can always be modified if necessary (especially useful when certain incompatibilities became evident during a trial run on the first few cases).

The information describing the questionnaire is useful for data checking. A minimum amount of errors, such as the value of a variable outside the range of variation, a missing chapter declared obligatory,... can thus be detected and reported. More complete checks taking into account logical relations between variables will be discussed in the next section.

5. ENSURING LOGICAL CHECKING OF THE DATA

Logical relations may exist between several variables in the questionnaire and can be very easily verified through the use of logical controls written in PIGAS language.

The logical relations can apply to variables from the same chapter or from different chapters.

There are two types of logical relations :

- UNCONDITIONAL : true for every case. For example, the relation "birth date is earlier than all other dates" must be true for every case.
- CONDITIONAL : concern only cases for which the first condition is true. They are expressed as follows :

  IF CONDITION
  THEN EXPRESSION

When a logical relation is false, an error message appears on the terminal screen at the end a record update for a case and errors are reported on paper at the end of the interactive session.

Logical controls may be modified and new ones may be added at any time. PIGAS checks all the cases of the file each time logical controls are modified.

6. CREATING NEW VARIABLES

New variables may be created using the declared variables of the questionnaire.

In PIGAS language, these new variables are called "GENERATED VARIABLES".

For example : on a questionnaire are noted the date of birth, the date of first treatment and the date of last follow-up of a case. Using these dates, the following variables may be created :

- age at first treatment
- age at last follow-up
- delay between the first treatment and the last follow-up date.

The GENERATED VARIABLES must be described in the same way as the original variables of the questionnaire indicating :

- its NAME
- its "LIBELLE"
- the TYPE (numeric or date)

The value of a GENERATED VARIABLE is computed with logical or arithmetical equations.

There are two types of GENERATED VARIABLES :

- UNCONDITIONAL GENERATED VARIABLES : GV = ARITHMETICAL OR LOGICAL EXPRESSION
- CONDITIONAL GENERATED VARIABLES :

```
IF  CONDITION 1
    THEN  GV = EXPRESSION 1
    ELSE IF  CONDITION 2
       THEN  GV = EXPRESSION 2
.....................................
          ELSE IF  CONDITION N
             THEN  GV = EXPRESSION N
             ELSE  GV = EXPRESSION N+1
```

When the condition or the expression cannot be evaluated (i.e. because of a missing value code in a variable), the generated variable is coded "UNKNOWN".

Generated variables may be interactively modified, deleted, or new ones may be added at any time. When created or modified, they are recalculated for each case in the file. During an updating session, generated variables are recalculated only for those cases modified.

Temporary variables may also be created during the analysis phase using the same syntax but are lost after each new call to the generating function.

7. PIGAS LANGUAGE

A logical relation consists of several logical expressions linked together by logical operators. A logical relation is expressed as follows :

EXPRESSION 1 "LINK" EXPRESSION 2 "LINK" ..... EXPRESSION N Logical controls are written using the variable names, relational operators ("EQ", "NE", "LE", "LT", "GE", "GT"), linking operators ("ET", "OU" for "AND", "OR"), mathematical functions and particular functions.

Up to three levels of nested parenthesis are authorized.

For a unique variable, its name completely identifies the variable.

For repetitive variables, the index number of the occurence corresponding to their respective repetitive chapter identifiers need or need not to be indicated.

When indicated, the variable is considered as a unique variable, otherwise each occurence of the variable is considered.

Some operators are specific to the treatment of occurences :

- The operator I

The operator I is used to compare two neighbouring occurences of variables.

For example, the date of entry (name DATEN) are noted for each hospitalisation. To control that dates of entry are increasing, the following expression is used :

DATEN(I) LT DATEN(I+1)

I is also useful to compare a repetitive variable to a unique variable.

For example, the date (name DATDOS) and the result (name DOSE) of doses of a certain substance are regularly noted in a repetitive chapter. When the dose reaches the value p, the subject presents a certain condition. The date of the condition (name DATHEP) is a unique variable noted in a summary chapter describing the condition. To control that the date of the condition in question is the date for which the result of the dose is p, the following expression is used.

```
IF  DOSE(I) GE p
THEN  DATHEP EQ DATDOS(I)
```

- The index number of an occurence

It can be 1, 2, ..., N : N being the index number of the last occurence. In the hospitalisation example :

DATEN(1) is the date of the first entry

DATEN(N) is the date of the last entry

Constants may be used in the comparison of variables. They can be numeric constants (up to 8 characters), date constants expressed in day, month, year or alphanumeric constants (up to 8 characters) expressed between quotes (" ").

Wild characters may be used in alphanumeric constants. Suppose an alphanumeric variable (name DRUG) is coded with 6 characters. Every drug which name begins by "ASP" can be selected using :

DRUG EQ "ASP***"

Mathematical operators (add, substract, multiply, divide, power) and Mathematical functions (exponential, natural logarithm, decimal logarithm, square root, absolute value, minimale and maximale

value of several variables) are provided.

Other functions are also particular to PIGAS language :

- 2 functions (DEL and ANI) used for the calculation of a delay between two dates. The unit used for the calculation may be day, month or year. The result of function "DEL" is rounded to the nearest unit whereas the result of function "ANI" is the number of completed units.
- Function CLAS which transforms a continuous variable into a discrete variable.

Particular functions for treatment of repetitive variables are provided. These functions are expressed under the form :

FUNCTION(VAR)

where VAR is a repetitive variable name. The result is a unique variable.

- The functions MIN and MAX give the minimale/maximale value taken by a repetitive variable which can be either numeric or date. For example, the value of doses are regularly noted and one is interested in the minimale (maximale) value observed. The syntax is :

MINDOS=MIN(DOSE)
MAXDOS=MAX(DOSE)

The results of functions MIN and MAX are coded "unknown" only when all the occurences of the variable are coded missing.

The computation of the functions MIN and MAX only involves non-missing values.

- Function SOM is used to compute the sum of the values of a repetitive numeric variable.

For example, at each session of radiotherapy, a chapter is filled in with the date and the dose delivered. The total dose delivered may be obtained as follows :

DOSTOT=SOM(DOSE)

The result of function DOSE is coded "unknown" if at least one occurence the value of the variable DOSE is coded as missing.

- Function SOMX ignores those variables with a missing value code. For example, the total dose delivered is obtained as follows :

DOSTOT=SOMX(DOSE)

- The functions IMIN and IMAX give the index number of the first occurence for which the value of the repetitive variable is minimale/maximale.

For example, date and results of doses are regularly noted. The following may be used to find the date at which the value of dose was minimale(maximale) :

A=IMIN(DOSE) or A=IMAX(DOSE)
DATMIN=DATDOSE(A) DATMAX=DATDOSE(A)

The treatment of missing values in the variables used is the same as for functions MIN and MAX.

- Function IND gives the number of occurences of a variable in a repetitive chapter.

For instance, the number of occurences of the follow-up chapter for each case is obtained as follows :

NBFU=IND(FU)

FU being the name of the follow-up chapter.

- Function INDX gives the number of occurences for which a repetitive variable is not coded missing.

For instance, to compute the mean dose, one can use :

MEANDOS=SOMX(DOSE)/INDX(DOSE)

MEANDOS can thus be computed even if one dose is missing at any one occurence.

8. UPDATING THE DATA

Data may be recorded either in batch processing mode or in interactive mode.

If a great amount of cases need to be introduced in the file at one particular moment, a data entry operator may assure this part of the updating process. Data are then introduced in the file in batch processing mode.

To record or to update a case in the file, the following information needs to be provided :

- the value of the case identifier
- the type of update to be performed on the case
- the name of the chapter or the variable name to be modified.

Several types of updates are possible :

- introducing a new case in the file
- deleting a case
- adding a new chapter to a case already recorded
- deleting a chapter for a case
- updating the value of a variable
- updating several variables of the same chapter.

To update a chapter, the up, down, left, right arrow keys may be used to move the pointer on the terminal screen until it reaches the character to be modified.

Each time a variable is updated, PIGAS checks and reports any coding errors. The code of a variable must match its description (cf. Section 4.0).

Erroneous values may or need not be immediately corrected. If not immediately corrected, the variable is recorded in the file with the "hold" code.

Once a case is updated, logical checks are performed on the case and errors are reported before any new updates can be made. When logical controls report errors and the case is not immediately corrected, the case is recorded in the file but temporarily excluded from statistical analysis. Only edition of variables of the case may be obtained.

9. ANALYZING THE DATA

The analysis stage is interactive and may concern the whole population or just a part of it.

9.1 SELECTION OF CASES

PIGAS offers four functions in order to select cases for treatment. The mnemonic of the function and the condition must be indicated.

The selected cases constitute a working subfile.

. Function SD

For example, the condition is :

AGE LT 50

If AGE is a unique variable, all cases with AGE less than 50 are retained for analysis.

If AGE is a repetitive variable, in matched case-control studies for example, the case and his controls are retained only if the condition is true for each occurence.

. Function SIF

Only the first occurence of the repetitive chapter for which the condition is true is retained in the working subfile.

For example, the condition :

DOSE GE p

will select the first occurence of the repetitive chapter for which DOSE is greater than or equal to p.

. Function SIG

Same as SIF except that all occurences are retained.

. Function ECLAT

Function ECLAT breaks down the notion of a case and considers all occurences as unique and independant variables.

The number of cases indicated equals the total number of occurences for which the condition is true.

For example, the condition :

DOSE GT 0

selects all records with DOSE greater than 0.

. Function POP

Function POP is used to obtain calculations for each value (except missing values) of the variable name cited.

For example, by typing in the variable name SEXE after the function POP, all the following calculations will be done separately for each sexe.

. HIERARCHY OF SELECTIONS

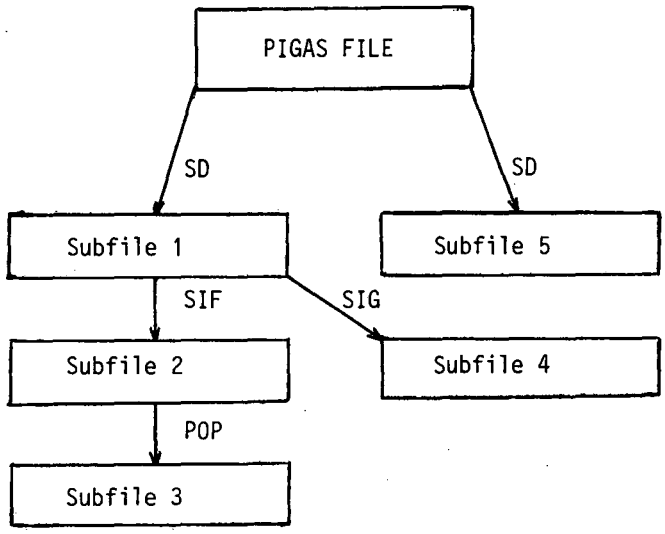
There are three levels of selections :

level 1 - SD

level 2 - SIF, SIG, ECLAT

level 3 - POP

For example, consider the following successive selections : SD, SIF, POP, SIG, SD. Working subfiles are created as described on the following figure ;



9.2 EDITION FUNCTIONS

Three functions exist in order to obtain sorted edition of variables. By default, the list is sorted according to the ascending value of the case-identifier.

The list may be sorted in ascending order according to other sort keys (up to 10 by numbering from 1 to 10 the variables to be sorted.

* function ED lists one or more unique or repetitive variables with one line per case. The number of variables is limited by the size of the terminal screen (80 characters).

* function EDPAP same as ED but only printed on the output file (up to 132 characters).

* function EDCOL used for repetitive variables with numerous occurences. The repetitive variables are printed with one line per occurence.

9.3 CROSS-TABULATIONS

These functions give tables, one line of which corresponds to a unique combination of values of all variables mentionned as well as their frequency of occurence.

* function HIST1 - used for unique variables or repetitive variables for which the index number of occurence is indicated. Tables cannot exceed the size of the terminal screen.

* function HSTLRG same as HSTCOL but tables are only printed on the output file.

9.4 STATISTICAL FUNCTIONS

Cases for which logical controls failed do not take part in the analysis as well as those variables with missing value codes.

Variables may be :

- variables of the questionnaire
- generated variables
- temporary variables

Variables may be unique or repetitive with the index number of the occurence indicated either :

- as the result of a selection of the occurence (SIF or SIG)
- by the index number of the occurence.

The syntax is very simple ; the mnemonic of the chosen function then the variable names must be indicated.

* function HIST2 and function HISTV - block histograms (cf. appendix 3)
* function C - histogram of a discrete variable.
* function MOY - mean and variance of a continuous variable
* function M - histogram of a continuous variable
* function ANDAR - Anderson-Darling Statistic for testing the normality assumption.
* function MED - median value of a variable
* function MGEO - geometric mean of a variable
* function TABC - RxC contingency table without any test
* funtion CC - RxC contingency table analysis, Chi-square and G (maximum likelihood) tests and Chi-square with continuity correction in the case of a 2x2 table.
* function CM - Analysis of variance
* function MM - Simple linear regression (1 independant variable)
* function CCC - RxCxK contingency table analysis
* function CMM - Analysis of covariance
* function MMM - Multiple linear regression (2 independant variables)
* function REG - Stepwise multiple linear regression (forward then backward)
* function KPL - estimated survival rates according to the KAPLAN-MEIER method.
* function SUAC - estimated survival rates according to the actuarial method
* function LGR - comparison of 2 or more survival curves using the LOG-RANK test (cf. appendix 4)
* function LGRA - comparison of 2 or more survival curves using the stratified LOG-RANK test.
* specific functions for the analysis of matched case-control studies with a variable number of cases and controls in each strata.
* function call to BMDP - which allows direct access to BMDP (cf. appendix 5)
* function call to GLIM - which allows direct access to GLIM.
* function USER - used to call FORTRAN program written to analyse data stored in PIGAS file (cf. appendix 6)
* function VAR - used to extract variables of a PIGAS file in order to create a subfile readable by FORTRAN programs. The subfile is created in the user's directory. PIGAS indicates the FORTRAN format to be used for reading extracted variables. Cases with missing values are transcribed in the subfile and it is up to the user to treat them accordingly.

9.5 EDITOR FUNCTION (EDT)

All the PIGAS commands (described in sections 9.1 to 9.4) are recorded in a "command file". The results are recorded in another file named "results file".

At any time during the interactive session, PIGAS allows the use of the computer text editor in order to :

- display or modify the command file
- display the results file.

When the command file is modified, all the commands included are recomputed.

PIGAS offers the possibility of saving the command file for future use. For example, routine calculations done every six months can be saved in this way without having to retype all the functions.

10. FILE PROTECTION

Each study is recorded in a specific PIGAS file identified by a name and a password. This name and this password are known by a person in charge of the study and by the computer manager. The names of persons authorised to access the PIGAS file as well as their code are checked at every login.

11. DATA RECOVERY

PIGAS files are saved daily.

The last two versions of all PIGAS files are kept so that in case of a user error or any other related incident, one can always go back to the N-1st version.

In case of power failure during an interactive session, only the results of the last interactive active is lost.

In case of program failure, all access to the PIGAS file of the study by the user are not possible until the computer manager corrects the problem.

12. TECHNICAL CONSIDERATIONS

PIGAS runs on a DIGITAL VAX 11/780 computer at the INSTITUT GUSTAVE-ROUSSY (Medical Statistics Department), in Villejuif. PIGAS may run on other computers similar to VAX. PIGAS is written in FOR-

TRAN (according to the standard 77, X: 39-1978. For the terminals the standard is ANSI X3.64 - 1977).

13. CONCLUSIONS

The most essential point to any statistical exploitation still depends on the way the questionnaires are filled in.

The ideal questionnaire should be set up in such a way as to take advantage of all the possibilities PIGAS has to offer : notably incorporating leading variables so as to diminuish data entry time, and logical checks through variable names so as to correct data during the updating process.

From our experience, most questionnaires with errors are detected during the updating process and corrected almost immediately, which is much more desirable than having to correct data during the analysis phase because of inconsistencies encountered during a cross-tabulated examination of the data.

The statistician thus spends less time on data checking and more time in the data analysis phase.

PIGAS has been operational since january 1982 and so far handles approximately 150 research projects. The number of cases vary from 50 to 4 000, the number of variables from 50 to 500, most of them being repetitive.

PIGAS in its actual form is very well adapted to the internal structure of our department and is also used by other departments who need to analyse studies with a non-rectangular data structure. The flexibility of the system has been demonstrated and especially appreciated by physicians working in direct collaboration with our department.

14. BIBLIOGRAPHY

Dixon, W.J. "BMDP" Statistical Software Manual
1982 Edition.
Department of Biomathematics
University of California Press.
Los Angeles, California

Poynard T., Wartelle M., Poitrine A., Kruger D., Naveau S., Jan P., Trebuchet L., Flamant R., Chaput J.C.
"Archivage informatique d'un service d'hépato-gastroentérologie. Comparaison avec l'archivage manuel."
To appear in "Nouvelle Presse Médicale".

Baker R.J., Nelder J.A. "GLIM 3 Generalised Linear Interactive Modeling". (1978) Rothamsted Experimental Station, Harpenden, Herts, England.

APPENDIX 1 : DESCRIPTION OF THE TABLE OF THE CHAPTERS

*** TABLE DES CHAPITRES ***

NOM	LIBELLE	OBLIG	REPET	MIN	MAX	CLES DE TRI DU CHAPITRE
IDENT	IDENTIFICATION CHAPTER	OUI	NON			
FOLLOW	FOLLOW-UP	NON	NON			
META	METASTASES	NON	OUI	0	5	DATMETA

APPENDIX 2 : DESCRIPTION OF THE FOLLOW-UP CHAPTER

****
* CHAPITRE FOLLOW *
****

NOM	LIBELLE	TYPE	LG	MINIMUM	MAXIMUM	CRX	NATURE
NAME	NAME OF THE CASE	A	3			NON	ISOLEE
DATFU	DATE OF FOLLOW-UP	D	6			NON	ISOLEE
AORD	ALIVE OR DEAD	L	1	0	1	NON	DIRECT
CAUSE	CAUSE OF DEATH	E	1	1	3	OUI	PARAG
STATE	STATE OF HEALTH	E	1	0	2	OUI	PARAG

DESCRIPTION DE LA FICHE FOLLOW

```
ID                          COL   1 -   2 : 22
NBCENTER                          3 -   4 : --
NBCASE                            5 -   8 : ----
NAME                              9 -  11 : ---
DATFU                            15 -  20 : ------
PARAGRAPHE AORD
AORD                             21 -  21 : -
 SI OUI :
     CAUSE                       22 -  22 : -
 SI NON :
     STATE                       23 -  23 : -
```

APPENDIX 3 : HISTOGRAM EXAMPLE

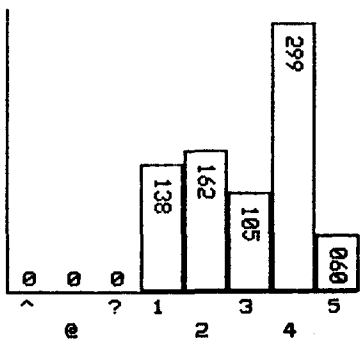
Number of cases as a function of the stage of a particular disease.

Function syntaxe :

```
FONCTION SOUHAITEE HIST
VARIABLE : STAGE
```

Result:

STAGE



('^' is the symbol used
for the "UNKNOWN" code
'@' is the symbol used
for the "NOT APPLICABLE" code
'?' is the symbol used
for the "HOLD" code)

APPENDIX 4 : LOG-RANK TEST EXAMPLE

```
TRAIT is the treatment variable coded 1 or 2
DATTAS is the date of randomisation
DADERNO is the date of last follow-up
ETAT is the patient status at last follow-up

 FONCTION  : LOGR
LA VARIABLE : TRAIT CONTIENT LES CODES ^,@,?     0 FOIS
LA VARIABLE : DATTAS CONTIENT LES CODES ^,@,?    0 FOIS
LA VARIABLE : ETAT CONTIENT LES CODES ^,@,?      0 FOIS
LA FONCTION DELAI N'A PAS ETE CALCULE            0 FOIS
                                  PAR MANQUE DE PRECISION
LA VARIABLE : DATDERNO CONTIENT LES CODES ^,@,? 0 FOIS

 ********************
 * TEST DU LOG-RANK *
 ********************
```

CLASSE	NBSUJ	DC OBS	DC EST	DC OBS/EST
1	12	6.	8.3823	0.7158
2	13	11.	8.6177	1.2764

CHI2	DDL	PROBABILITE
1.3357	1	0.24780

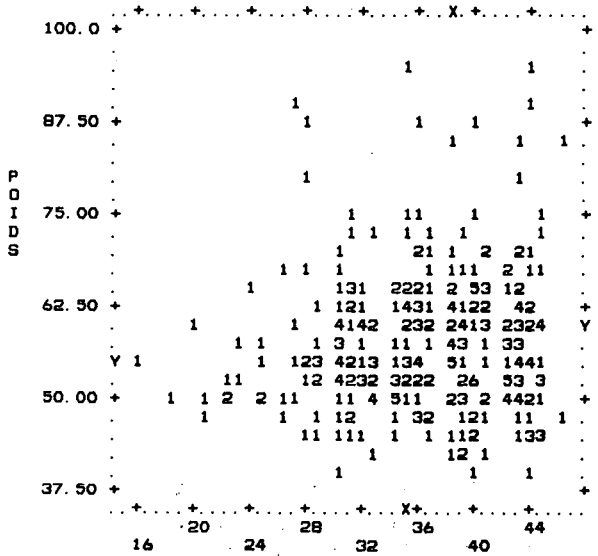
APPENDIX 5 : BMDP

```
FONCTION SOUHAITEE BMDP
NOM DES VARIABLES
WEIGHT
AGE

NOM DU PROGRAMME : 6D

BMDP6D - BIVARIATE (SCATTER) PLOTS
DEPARTMENT OF BIOMATHEMATICS
UNIVERSITY OF CALIFORNIA, LOS ANGELES, CA 90024
(213) 825-5940                    TWX  UCLA LSA
PROGRAM REVISED JUNE 1981       MANUAL REVISED -- 1981
COPYRIGHT (C) 1981 REGENTS OF UNIVERSITY OF CALIFORNIA
    17-DEC-82        AT 16:55:18
 .............................

NUMBER OF CASES READ. . . . . .                    335
```




```
N=  335
COR= .0768

     MEAN     ST. DEV.     REGRESSION LINE     RES. MS.
X   36.481     5.9377   X= .04879*Y+ 33.667     35.154
Y   57.669     9.3425   Y= .12079*X+ 53.262     87.028

VARIABLE    2  AGE       VERSUS VARIABLE  1  WEIGHT

CPU TIME USED      6.990 SECONDS
```

APPENDIX 6 : USER

```
FONCTION SOUHAITEE USER
VARIABLE  1 : CENTER
VARIABLE  2 : CASE
VARIABLE  3 : DATRANDM
VARIABLE  4 : GROUP
VARIABLE  5 : CHEMO
VARIABLE  6 : SURGERY
VARIABLE  7 : HISTOLOG
VARIABLE  8 : RADIO
VARIABLE  9 : FOLLOW

NOM DU PROGRAMME : PROG
```

****
* CENTER : VILLEJUIF *
****

Case number	Date of randomisation	Missing records
5	3/ 5/1981	: RADIOTHERAPY
6	22/ 5/1981	: FOLLOW-UP
8	16/ 6/1981	: RADIOTHERAPY
11	22/ 8/1981	: SURGERY
12	18/ 8/1981	: RADIOTHERAPY
16	25/10/1981	: SURGERY : HISTOLOGY : RADIOTHERAPY
24	8/10/1980	: SURGERY : HISTOLOGY : RADIOTHERAPY
28	5/11/1981	: SURGERY
32	4/ 1/1980	: RADIOTHERAPY
45	6/ 5/1981	: FOLLOW-UP
46	22/ 5/1981	: SURGERY : HISTOLOGY : RADIOTHERAPY

SIMULATORS, STATISTICAL ANALYSIS, AND DATABASES

D.H. Scuse
Associate Professor

A.N. Arnason
Professor

Department of Computer Science
University of Manitoba
Winnipeg, Manitoba

Abstract

This paper describes the advantages of integrating simulators, statistical analysis programs, and databases in a common statistical information system. We have found that the integration of the components in a statistical information system greatly enhances the value of the individual components, makes the resulting system easier to use by end-users, and reduces the amount of work required to implement new applications and modify existing applications.

1. INTRODUCTION

For a statistical information system to be useful in the current research environment, it must contain routines that manipulate data organised in complex data structures (databases, as opposed to simple flat files) and routines, statistical analysis programs or SAPs, that analyze these data. The integration of the statistical analysis programs and databases makes the system more powerful than the equivalent separate systems, one to manipulate databases and the other to perform statistical analyses. As we shall show, the value of the statistical information system is also enhanced by the addition of simulators that permit end-users to develop and refine management sampling or data collection strategies.

There are many systems that provide excellent facilities for the manipulation of complex statistical databases (SIR, RAPID), for the analysis for statistical information (SAS), or for simulation (SIMSCRIPT). However, few of these systems provide acceptable facilities for even two of these functions (manipulation, analysis, and simulation) and no general-purpose system that we are aware of provides all three functions. As applications become increasingly complex, it will not be acceptable to have to copy information manually from one system to another as different functions are required when processing the information; instead, integrated systems that provide all three of these functions and make them available directly to end-users in a friendly manner must be developed.

Our interest in simulation, databases, and the analysis of information has led to the development of several information systems that provide some but not all of the facilities that we feel are necessary in a statistical information system. These systems include the POPAN system [1] for the maintenance and analysis of mark-recapture databases for wildlife sampling experiments, the MANHIS system [2] for the maintenance and analysis of both social services and medical data in community health centres, and the Fisheries Information System [3] for the simulation and storage and reporting of fish-hatchery management data. Of these systems, the Fisheries system is the most recent and most advanced, integrating state-of-the-art database technology with a powerful second-order Markovian simulator system.

In the remainder of this paper, we indicate the advantages of integrating simulators, statistical analysis programs, and databases into a common statistical information system. The problems involved in implementing such a system and the techniques that we have found to be useful in eliminating some of the problems are also examined.

---

* This research was supported by grants from the Natural Sciences and Engineering Research Council of Canada, the Canada Department of Fisheries and Oceans, and Algas Resources Ltd., Canada.

2. INTEGRATION

In this section we examine the advantages obtained when the various components are integrated into a common statistical information system. The database component is assumed to include an interactive query/update facility that permits the user to insert information into the database and to examine selected portions of the database, preferably without having to know the structure of the information in the database.

2.1 SIMULATORS AND DATABASES

The importance of databases as support for developing, fitting, testing, and running simulations has recently been discussed by Markowitz [4]. As co-developer of both the SIMSCRIPT II language [5] and of the EAS-E database system [6], he has given some thought to the facilities that a database system must have to provide this support. EAS-E is a Query/Update database system that permits the structuring and manipulation of data using the same time-ordered, entity-attribute-set view of a system as is taken by SIMSCRIPT in modelling that system. Such a view is important for simulation involving scheduling, queuing, and contention by transactions (temporary entities) for limited physical resources (permanent entities); for example, in a job-shop scheduling model, machines (permanent entities) may own a set of tasks (temporary entities) which in turn may own other sets (e.g. job stages) and attributes (type classifications, service time durations, etc.). While EAS-E is not (yet) fully integrated with SIMSCRIPT, the advantages of adopting a parallel data structuring in the database and the simulation are already apparent: browsing through the database or extracting reports from it orients the end-user to the structural and set membership/ownership relationships that he will have to use in simulating the system; the same code for generating reports from the database can be used to generate equivalent reports on the simulated model; the database is structured in a way to accept data from the simulated model, preserving its structural relationships; real data in the database can be extracted (usually as time-ordered sets) for driving various mechanisms of the simulations (arrival times of temporary entities, with their attributes, and other demand or service-time mechanisms). Markowitz has pointed out [4], however, that this is not sufficient support for many simulation applications. Often real data is of limited value in production runs of the simulator (it is of most value in validation runs). Instead, one wishes to extract data on such exogenous mechanisms as inter-arrival times or service times, fit theoretical or empirical models to the data, and then use these mechanisms, or hypothesized modifications of them (e.g. the same form of distribution but with an increased mean and/or variance) in the simulation. This gives the user the freedom to explore hypothesized strategies or non-existent but plausible environments to judge how the system responds. Such experiments with simulation models are often the main purpose for their construction. Clearly this facility requires the further integration of the system with statistical analysis features for exploratory data analysis, model fitting, and testing. Once such models are fitted, most simulation systems already incorporate methods of parametric or empirical random variable generation, given the fitted parameters.

In the analysis of real data, statistical analysis methods are useful for summarizing and revealing relationships in the data. Since simulations can frequently produce voluminous data on system performance, access to standard statistical analysis procedures can greatly assist in the process of summarization. As we explain later, we feel that this facility is best provided by having the simulator write out its results, in as detailed and unprocessed a form as possible, to the database. It is then the role of the database query system to provide flexible, on-line end-user control over the data summaries and reports.

The entity-attribute-set representations of SIMSCRIPT/EAS-E are not always the most appropriate or natural system representations. Our Fisheries system involves a more restrictive system representation, but as a result, we have been able to make much greater progress in integrating the simulator with the database, and have a much clearer idea of how statistical processing features can be implemented. As with the SIMSCRIPT/EAS-E system however, the crucial correspondence between the structuring of the database and the structural relationships in the simulation is preserved. In our case the basic structural relationship is one more suitable to Markovian systems: data on the actual hatchery consists of two types of variables: exogenous events (which we call management interventions, consisting of setting stocking levels in tanks, feeding rates, water temperature and flow rates) and endogenous results of these interventions (called observations, consisting of censuses of fish numbers and weight distribution, water quality variables, etc.). In addition there are static system descriptors which may be real or hypothetical (called physical conditions, such

as numbers and size of tanks, water availability, filter capacities and characteristics, fish species and food type characteristics as they affect growth rate). The database must, and does, permit the user to follow groups of fish forward and back in time, within tanks or relocating to other tanks as groups of fish are split among tanks or transferred to other tanks. The user can specify the variables to be displayed as he navigates through the data. The important relationships that must be preserved are the time orderings of events and observations within tanks (e.g. it is vital to know if a census, at the same time as a transfer, was before or after the transfer). The simulator allows the end-user to specify initial conditions (physical conditions) and a sequence of interventions interactively. A multilevel growth model then generates the observed results of the management strategy to a degree of detail depending on the level chosen (e.g. the basic model only simulates growth and imposes no constraints on density or water quality; higher level models also simulate effects on water quality). The results of such a simulation can be saved to the database and are, in every respect, comparable to real data (except for an additional variable to indicate that they arise by simulation and hence may be deleted from the database, unlike real data).

The preservation of an exact correspondence between real and simulated data enables a number of desireable capabilities. First, real physical conditions and interventions can be extracted from the database and used to drive the simulator. The simulated results are saved in the database and are associated with the interventions that gave rise to them. Thus the same code (in the database query system) used to compare two real growth strategies can be used for comparing real and simulated outcomes for validation purposes. Secondly, the results of a successful management strategy, obtained by simulation against a real or hypothetical physical configuration can be saved and extracted later as the strategy is actually implemented.

The statistical analysis requirements of this system fall into two main categories: statistical summaries and calibration. The first has already been mentioned as it involves the ability to compare two growth runs (real/real; real/simulated; or simulated/simulated). The problem is that aquaculturists want analyses based on derived statistics (e.g. instantaneous growth rate, food conversion ratios, etc.) which require data from the same or different database segments at two different times. This involves extensive checks to ensure that the derived statistics are meaningful (e.g. that no deaths or transfers intervene between two censuses) and is best provided through the data dictionary and database interface facilities described later in this paper.

Calibration involves the extraction of data from the database in forms (case by variable data) suitable for non-linear growth model fitting and the fitting of various ancillary models (variance in growth, oxygen consumption, fish metabolite production rates, etc.). This involves the same problems as for generating derived variables. Full integration of calibration with the system would require some means of designating groups of segments in the database as belonging to calibration sets suitable for a particular model fitting routine and/or programming extensive checks that sets of data chosen by the end-user are complete and consistent for use by a particular fitting routine.

2.2 STATISTICAL ANALYSIS AND SIMULATORS

Statistical analysis procedures may be roughly categorized into survey analyses and model fitting analyses. The former often involve massive amounts of data, sometimes incorporating complex relationships among records, where the main objective of the analysis is to summarize properties of the variables (means, variances, histograms) or of relationships among variables (crosstabulations, breakdowns, scattergrams). Model fitting analyses attempt to elucidate structure and relationships by positing, fitting, and testing for goodness of fit, some theoretical model for the expected values of the observations (data) and the variation of the data about expectation (residual error distribution). The model parameters, or rather their estimates formed in the fitting stage, may be of interest in providing insight into the mechanisms that gave rise to the data, or the fitted model itself may be of primary interest for use in prediction, interpolation, optimisation, or simply as a data summary.

Most general-purpose statistical packages (SPSS, GLIM, BMDP, SAS, etc.) have capabilities spanning both of these categories. None, however, incorporates simulation capabilities, and so are rather poor in supporting the major planning and interpretation concerns of statistical users in both these categories.

We do not propose that these concerns can be fully met by simulators, particularly for survey analyses. However, methods for random

sampling from sequential files are well known [7], and it would be quite straightforward and useful to have a general simulation capability for generating random samples according to various standard designs (stratified, ratio, multi-stage, etc.). Experimentation with such a system would be of great help to the end-user in planning efficient surveys. What limits the usefulness of this method is the necessity to provide a file to be sampled. This might be an actual census or sample survey from some comparable experience, but more likely the file itself would have to be simulated, using the distributional and relational characteristics from some previous survey analysis, plus some hypothesized model assumptions (e.g. multivariate normal or multinomial distributions for variables) to generate simulated populations.

For the end-user concerned with model fitting, the much greater usefulness of simulation for planning and insight is illustrated by the POPAN system. This large batch system, while specific to marking and banding data from animal sampling experiments, edits and structures the end-user's data, gives him access to a comprehensive set of models for parameter estimation (of survival, birth rates, abundance), and includes a very general but easy-to-use simulation capability. All these functions are directed by a BMDP-like paragraph-structured command language. The simulator permits the user to simulate a theoretical population's stochastic dynamics (recruitment, death, emigration) and impose stochastic sampling at user chosen intensities and frequencies. The simulator is general enough to permit him to specify models that incorporate mechanisms that violate assumptions of the analysis methods (e.g. distributions for heterogeneous survival or capture rate among animals). The data are generated in a form that is (almost) indistinguishable from real data, and so can then be analysed using the model fitting procedures, presenting results exactly as for real data. The only difference is that the system reports the "true" population properties, against which the estimates can be compared. Simulations without assumption violations are useful in planning allocation of sampling effort so that experiments will yield adequate precision in the estimates. Simulations with assumption violations are useful in determining the bias or robustness of an estimate to various types and degrees of assumption violations and the power of tests to detect such violations. Conclusions from such runs lack the generality of an analytic investigation by a mathematical statistician, but they reveal the same sort of insights, can be carried out easily by a non-statistician end-user (e.g. a biologist or wildlife manager), and can be tailored to the specific population parameters and dynamics which the end-user has to hand.

Such capabilities would also be enormously useful, for the same sorts of purposes, in more general statistical analysis systems. Capabilities to generate stochastic data from wide classes of models (including various independent or correlated error or residual distributions) would be useful for planning, robustness, and power studies in many classes of analysis: linear, log-linear, general linear, failure time, even multi-variate models.

Note however that it would be a mistake to "tack-on" such simulation features to a statistical analysis program. In fact, the statistical analysis program must be incorporated within a general statistical model simulator. This way, the simulator can invoke the analysis procedure on the (stochastically) generated data file automatically. This is, to our mind, the only realistic approach if replicated simulations are to be allowed. Replicated realisations of a given model and/or the sampling imposed on it are essential for investigating bias and power, though are less essential for experiment planning. Thus the simulator, and not the statistical analysis program, is responsible for generating the data file, carrying out the analysis, extracting and accumulating statistics to be reported over replicated simulations, conversion (e.g. to means, standard deviations, proportions over replications), and report generation. The simulator must also report on the simulation mechanisms and the "true" properties of the model. Analysis of real data is then a special case of a single replication where file generation consists merely of extraction of cases and variables from a file or database of real data. This system structure would confer advantages in analyzing real data which was (exactly) replicated. The sorts of summaries across replications suggested here could then be applied to real data as well; this is a rather rare requirement, but one which is difficult to meet with most statistical analysis programs at present.

We defer until later a discussion of the role played by a database incorporated into such a system, though it is clear that some database support has been implied in the discussion in this section.

2.3 STATISTICAL ANALYSIS AND DATABASES

Since the use of databases in statistical information systems is becoming the accepted practice (one has only to examine the number of papers involving databases at conferences such as the First LBL Workshop on Statistical Database Management and the Symposia of Computer Science and Statistics: the Interface), in this section we discuss only briefly the advantages to the integration of databases and statistical analysis programs, as they apply to our experience.

The use of a database instead of the more common flat file makes the conceptualization and manipulation of data by end-users during the analysis of the data significantly easier. For example, as shown in the papers [8,9,10,11] on volume testing presented at the 13th Symposium of Computer Science and Statistics and the First LBL Workshop on Statistical Database Management, the analysis programs were easier to design and understand when the underlying system supported databases than the equivalent analysis programs when the underlying system supported only flat files. When a database is used, it is normally not necessary to perform complex file manipulations (merges, sorts, etc.) in order to place the data in the correct format for the analysis. The use of the database facilities also makes the management of information being prepared for analysis easier than in the flat file analysis system since most database systems support some type of interactive method for entering data into the database and automatically perform integrity checking of the information as it is entered to ensure that it is valid.

The integration of the statistical analysis program with a database permits the end-user to examine portions of the database interactively using the query component of the database system before defining the analysis program. Such interactive access to the data often gives the end-user a better "feel" for the data, enabling him to develop appropriate analyses more quickly.

The integration of statistical analysis programs and databases does not have to be a one-way transfer of information (from the database to the statistical analysis program); it is frequently desireable to be able to store the results of an analysis back in the database to avoid having to recalculate them the next time that the information is required. For example, summary information that contains means and variances may be a useful addition to a database. Such summary information is then available to the user through the database query component. Of course, such information may be invalidated should the underlying data be modified but statistical databases tend to be modified infrequently compared with entity-oriented databases (personnel, inventory, etc.).

2.4 SIMULATORS, ANALYSIS, AND DATABASES

The advantage of integrating all three components has already become evident in the Fisheries system which contains a simulator and database system but does not contain statistical analysis programs. Both the data generated by the simulator and the data generated by the hatchery are stored in the fisheries database. In order to compare the simulated results of an experiment with the real results for the same experiment, it was necessary to export the real and simulated data outside of the Fisheries system in order to perform the analysis. If the Fisheries system contained the appropriate analysis routines, it would have been much easier to perform the comparison. The same process must also be followed for model calibration, with the additional re-importing of the fitted model parameters into the species characteristics file of the database before the end-user can make use of the system's powerful model validation capabilities.

An interesting use of the integration of all three components is to use the simulator to generate test data for a new application before data are actually collected. These simulated data can be stored in the database and then queried and analyzed by the user to ensure that the data are complete. Such a technique would be useful in experiments involving surveying or sampling in order to reduce the possibility that part of an experiment is performed before it is realized that insufficient or incorrect data are being collected.

3. IMPLEMENTATION TECHNIQUES

The major problems involved in integrating the components of a statistical information system can be reduced to more manageable problems through the use of current database and software engineering technology. The techniques that are described in this section have already been implemented (in whole or in part) in the Fisheries, POPAN, and/or MANHIS systems and have been found to work well. The organisation of components used in the Fisheries system (extended to suit the current discussion by the addition of statistical analysis) is shown in Figure 1. Access to the system is through a common user interface. The major components then interact with a data dictionary/database system (but not directly

with each other); this makes the components more modular, more independent of each other (so that changes in one component do not affect the other components), conferring great advantages on ease of development while preserving the maximum power and flexibility of the system. As was illustrated in Section 2, the pairwise or vertical interaction of the components with the DD/DB system reduces application coding.

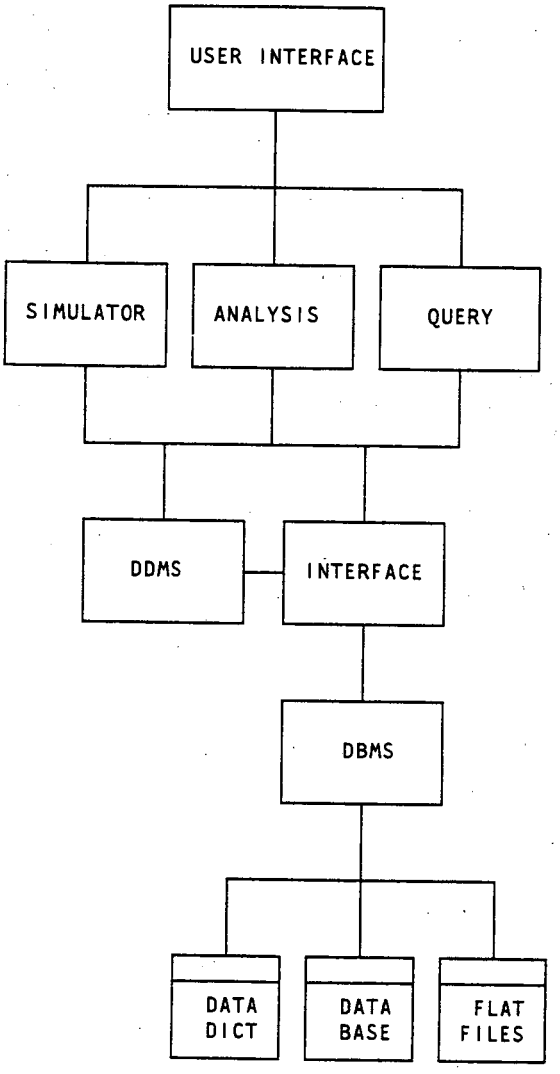


FIGURE 1 STATISTICAL INFORMATION SYSTEM

One of the major problems encountered in the design of an interactive system is the technique used to parse the users' commands and then to pass the necessary information to action routines to process the commands. In order to provide a common method of access to all programs in the Fisheries system, a generalized user interface, called EASYPARSE, was developed. With EASYPARSE, users need learn only one set of conventions in order to use the Fisheries system. EASYPARSE contains a powerful Backus-Naur Form (BNF) parser that was designed for users who are technically competent in their own areas but are not familiar with the intricasies of interactive systems. The parser is keyword oriented although it can also parse commands with positional parameters. The syntax of the commands that the parser is to process is defined in an external table which is read by the parser during system initialisation. After the parser has recognised a command and its parameters, the parser isolates the command parameters by creating a list of pointers that point to the individual parameters and subparameters. This list of pointers creates a level of indirection between the parser system and the action routines that process the commands, making it possible to make minor syntax changes to a command without having to modify the action routine. The parser system automatically abbreviates all commands, keywords, and aliases as much as possible (as long as no ambiguities are created); however, for particularly critical commands, keywords, or aliases, a minimum number of characters that must be entered for the command to be recognized can be specified when the command syntax is defined. To simplify the entering of commands and parameters, the parser permits the definition of substitution variables, variables that are assigned character strings by the user. Whenever a substitution variable is entered by the user, the parser system replaces it with the character string associated with that variable. Thus, frequently-used phrases can be replaced by a substitution variable. The parser system also supports a sophisticated command-editing facility which permits the user to modify portions of the current command or a previous command without having to retype the entire command. EASYPARSE has recently been prepared as a stand-alone system for use with any interactive program and is described in [12].

In order to make the statistical information system as general as possible, the description of the processing to be performed on each database should be stored in a data dictionary instead of being "hard coded" into the system components. The Fisheries system is dictionary-driven and contains little logic

that is specific to the manipulation of hatchery information; as a result, it has been possible to make major revisions to the system without having to modify the application programs. This technique has also been used very successfully in the RAPID system [13]. The meta data stored in the data dictionary describe the organisation of the database, the input/output formats of data, security information, integrity information (horizontal and longitudinal edit and consistency checks), data transformations (for example, the value of the variable SEX might be input and output as M or F but stored internally as 0 or 1), command syntax information for the user interface, and descriptive information. The data dictionary should be stored in a database that is manipulated by the database system so it is not necessary to write low-level database manipulation routines as part of the data dictionary management system (DDMS), and, even more importantly, so that it is possible to display the contents of the data dictionary using the query system. This last feature is important for end-users since it provides a convenient means of obtaining the descriptive information about the databases maintained by the system, such as the use of a particular field and its input and output formats.

The Fisheries system does not permit programs that require data from a database to issue requests directly to the database management system (DBMS); instead, an interface was placed between the DBMS and the processing programs, and all programs issued requests for data to the interface instead of to the DBMS. This interface was originally developed because the only DBMS available (IBM's IMS) at the university at the time was not suitable for the type of manipulations being performed (and we did not want to develop a complete DBMS by ourselves), but the interface was gradually improved until it became a high-level database management system in its own right. The interface supports the relational model of a database and a sophisticated database manipulation language by translating the requests passed to it into the equivalent requests to the low-level DBMS being used. The interface also provides many additional facilities such as the ability to dynamically display program debugging information. The interface has proven to be valuable because it can be extended quite easily and because it has permitted the other components of the statistical information system to be made independent of the actual DBMS being used to store the database. Further details of the interface can be found in [14].

In statistical information systems it is important that the user be permitted to define fields whose values are derived from the values of other fields. The use of the data dictionary system and the database interface in the Fisheries system has made the manipulation of such derived fields reasonably easy. Derived fields are first defined to the data dictionary; then, when the derived field is accessed, the database interface invokes a special routine to obtain the value of the derived field. The value may already be stored in the database (for example, as the result of adding summary data to the database, as was mentioned earlier), in which case, the value is read from the appropriate database segment; otherwise, if the value has not already been calculated, its value is calculated dynamically by processing the values of the fields from which the value is derived. We feel that the manipulation of derived fields is an important consideration in the design of a statistical information system.

To facilitate access to a database by the end-user, the query component of the system must require as little knowledge of the structure of the database as possible. As was mentioned earlier, the database interface in the Fisheries system supports the relational model of databases so that users need not be aware of the links between segments/tuples in the stored version of the database. However, even the relational model is more complex than was desired for end-users since they would have to be aware of the grouping of fields into tuples. Consequently, an abstract model of the database was defined for the users. This abstract model permits the user to access the database by field name without having to know the name of the tuple in which the field is defined. The query component performs the mapping from the abstract model to the relational model before it issues a request to the database interface for information. The support for multiple models of the database has been facilitated by the use of the data dictionary: the data dictionary contains a description of each of the three models of data, including the name of each field, the tuple in which it is defined, and how the tuple must be accessed using a particular model. As a result, the mapping processes in the query component and the database interface have been made completely general, using the data dictionary description of how a particular field or tuple is to be processed during the mapping from one model to another.

While much of the discussion in this section has been specific to the Fisheries system, the techniques used in that system are generally applicable to the problems encountered when creating a generalized statistical information system. In fact, the Fisheries system is such a generalized system, capable of manipulating any database; the only component that it lacks is the analysis component.

4. CONCLUSIONS

The statistical information system described in this paper has two main functions: to provide the user with sophisticated facilities for manipulating statistical information, and to reduce the amount of programming required to implement a new application. The interactive nature of the system encourages users to experiment with the system and the user interface eliminates much of the frustration that end-users typically feel when dealing with a complex system.

Since the system is dictionary-driven, few application programs should be required in order to implement a new application. Instead, the system administrator defines the data requirements of each application to the data dictionary. This approach to the development of applications has recently been emphasized by Martin [15]. The use of software interfaces to control access to critcal portions of the system permits debugging and trace information to be generated during program testing by toggling various switches in the interfaces instead of having to recompile programs during testing in order to insert and then remove debugging statements.

We are currently in the process of extending modules written for the Fisheries system in order to construct a prototype of this statistical information system on the University of Manitoba's Amdahl 470/V8 computer system. Unfortunately, the scope of the information system precludes our being able to implement it on a small computer system.

REFERENCES

[1] Arnason, A.N. and Baniuk, L., (1978), "POPAN2: A Data Maintenance and Analysis System For Mark-Recapture Data", Charles Babbage Research Centre, St. Pierre, Manitoba.

[2] Scuse, D.H. and Trute, B., (1981), "The Manitoba Health-Centres Information System", Proceedings of the Fourteenth Hawaii International Conference on System Sciences, Medical Information Processing Track, 60-70, January, 1981.

[3] Arnason, A.N., Schwarz, C.J., and Scuse, D.H., (1981), "An On-Line Simulator and Database System for the Management of a Commercial Fish Farm", Proceedings of the Winter Simulation Conference, Atlanta, Georgia, Volume I, 141-152, IEEE, New York.

[4] Markowitz, H.M., (1981), "Barriers to the Practical Use of Simulation Analysis", Proceedings of the 1981 Winter Simulation Conference, Atlanta, Georgia, Volume 1, 3-9, IEEE, New York.

[5] Kiviat, P.J., Villanueva, R., and Markowitz, H.M., (1973), "SIMSCRIPT II.5 Programming Language", C.A.C.I. Inc., Los Angeles, California.

[6] Malhotra, A., Markowitz, H.M., and Pazel, D.P., (1980), "EAS-E: An Integrated Approach to Application Development", RC8457, IBM, T.J. Watson Research Center, Yorktown Heights, New York.

[7] Fan, C.T., Muller, M.E., and Rezucha, I., (1962), "Development of Sampling Plans by Using Sequential (Item by Item) Selection Techniques and Digital Computers", Journal of the American Statistical Association, Volume 57, 387-402.

[8] Teitel, R., (1981), "Volume Testing of Statistical/Database Software", Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface, Springer-Verlag, New York.

[9] Robinson, B., (1981), "Scientific Information Retrieval (SIR/DBMS)", Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface, Springer-Verlag, New York.

[10] Bragg, A., (1981), "Volume Testing of Statistical Software, The Statistical Analysis System (SAS)", Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface, Springer-Verlag, New York.

[11] Schmitz, P., (1982), "Using the INGRES Relational Database System in Statistical Applications", First LBL Workshop on Statistical Database Management, 363-367.

[12] Arnason, A.N., Cameron, H., Karasick, M. Paulley, G., and Scuse, D.H., (1983), "EASYPARSE: A Generalized Easy-To-Use Parser Interface for User-Oriented Systems", Congressus Numerantium 37, January, 1983.

[13] Hammond, R., (1981), "Metadata in the RAPID DBMS", Proceedings of the First LBL Workshop on Statistical Database Management, 123-131.

[14] Scuse, D.H., (1982), "Database Interfaces", Australian Computer Journal, Special Issue on Software Engineering, Volume 14, Number 2, 71-74, May, 1982.

[15] Martin, J., (1982), "Application Development Without Programmers", Prentice-Hall, New Jersey.

4. Time Series and Econometric Database Management

See Also. . . .

CANSIM, THE CANADIAN SOCIO-ECONOMIC MANAGEMENT INFORMATION SYSTEM

Martin Podehl, Statistics Canada

Abstract

CANSIM, the Canadian Socio-Economic Information Management system is Statistics Canada's computerized data bank and information retrieval service. This paper gives an overview of the current status and development activities of electronic information services provided by Statistics Canada.

INTRODUCTION

Econometric and statistical analysis requires two ingredients: data and analytical software tools (in addition to a computing environment of course). Statistical data are either available in the public domaine or are the result of a researcher's own statistical collection program. They are either micro data, for example the result of a survey, or they are macro data, of aggregated nature, for example economic statistics. Analytical software tools are available in a wide variety of packages which have been developed over the years and are being refined constantly to take advantage of newly developed algorithms as well as new hardware/software environments.

In the beginning of statistical computing the main concern was with the development of efficient and sound algorithms. Statistical analysis packages were developed usually by universities and Research Institutes. As these packages matured, data and file handling facilities were added in order to ease the burden for the researcher in that respect. In the commercial environment, on the other hand, data base management packages were developed in order to ease the burden in organizing, storing, documenting and accessing data in a flexible manner. As these systems matured, flexible retrieval and analytical software of statistical nature were added in order to explore the full information potential contained in those data bases.

Today these distinctions are of no significance anymore as both, the data base management system developer and the statistical package developer have realized the importance of a smooth interface between both environments.

CANSIM, the Canadian Socio-economic Information Management system of Statistics Canada fulfills both needs: an organized data base of statistical information, and analytical tools for statistical and economical modelling and interpretation.

CONCEPT AND PURPOSE

The concept of CANSIM was born in the late '60's as a mechanism to store and make available to the public key statistics to economists and statisticians in Canada. CANSIM brings together under one umbrella data from Statistics Canada as well as other organizations such as Federal Departments, Provincial Governments and the Bank of Canada. Today, in a publicly accessible data base, socio-economic data of time series as well as cross-classified nature are stored, documented and disseminated to the statistical community in Canada. From a modest start CANSIM has grown in volume and in importance and is now an integral part of socio-economic analysis in Canada.

The original concept placed primarily emphasis on a simple data organization such that data from different sources could be related to each other. Simple access routine allowed the retrieval of selected time series which then were taken by the user into his own environment for further computation. Over the years much software was developed or interfaced with CANSIM for increasing complex analysis.

EVOLUTION AND OPERATION

Initially CANSIM was synonymous with time series. In 1968 the foundation

was laid with 2,500 series, today the time series module carries over 350,000 time series. In order to handle cross-classified data efficiently, a cross-classified module was added to CANSIM which since 1976 has been used to store data of multi-dimensional nature, referred to as tables, where a table can be either retrieved in whole or only parts of it. CANSIM Cross-Classified now carries data from a variety of predominantly social statistical areas, such as health, justice, education, and demography.

Recently a third module was added, referred to as CANSIM Summary Data which allows access and selective retrieval of data which are also available in the form of User Summary Tapes. At present our Census data aggregated to small areas are available in this module. That we now have three distinct information systems is not the result of design but rather historical evolution. At some future point unification has to be attempted such that one data model can be used to describe all data regardless of whether they are predominantly of time series or predominantly of cross-classified nature.

The CANSIM systems are maintained and operated by Statistics Canada. They are maintained under contract at a commercial computing service organization and they are updated daily. The public has access to these data bases under separate, individual contracts with the supplier. However CANSIM data are also available under the trade name CANSIM Mini Base, through other computing services companies which we call Secondary Distributors. Secondary Distributors obtain daily updates to a standard sub-set of the main base. In addition they can obtain supplementary time series as requested by their clientele.

This delivery mechanism of Host Service Bureau and Secondary Distributors offers our users a choice in access and analytical software. Some of the Secondary Distributors are in the information base business and make CANSIM data part of a larger set of economic and statistical information, while others are general purpose computing services companies who had been asked by specific customers to make CANSIM data available at their computing centres.

In Statistics Canada the CANSIM Division is responsible for all development and operational aspects concerning this information dissemination approach. The CANSIM Division adds new data and maintains existing data in all data bases at the Host Service Bureau, and it produces and distributes printed data directories which describe the data and provide the access identifications. CANSIM Division undertakes a marketing and training program and as well provides a marketing and training program and as well provides consultation to all users of CANSIM data who are searching for particular data or have difficulties in interpreting them. Last but not least CANSIM Division maintains and develops new software to explore further the CANSIM data bases for analytical purposes.

ANALYTICAL SOFTWARE

Originally Statistics Canada had to develop software for retrieval, manipulation, statistical analysis, and representation of results. However over the years many packages have been interfaced to CANSIM either by Statistics Canada as part of the Main Base, or by Secondary Distributors who took the CANSIM Mini Base and integrated it into their own software facilities. This was only possible because the data model for the time series data is very simple and follows common conventions within the economic and statistical community. In the beginning of the CANSIM development the linking of data to software tools needed a lot of attention for reasons of limitations in speed and size of hardware, as well as lack of adequate packages. Today, computing resources are much cheaper and the concern has shifted to developing an environment in which packages can be used at ease by providing smooth and transparent interfaces between data storage and access systems and analytical tools.

Software tools for the exploitation of statistical data bases can be categorized broadly as follows:

a) Basic retrieval and selection

This function is usually provided by the access software to a particular data base environment.

b) Normalization and Transformation

Having retrieved selected data, they often need to be normalized and transformed to make them compatible. An example would be two time series which both contain price indexes, but based on different base years. Before they can be compared they need to be adjusted to a common base year.

c) Statistical/Econometric Analysis

Here we have a variety of packages. For example CANSIM data have been interfaced to TROLL for econometric analysis and SAS for statistical analysis. In addition APL has become the defacto standard as the fall-back software package for manipulation which other packages cannot provide.

d) Reports

A convenient and powerful tool to present data, particularly larger amounts of data is essential. We have used two particular approaches.

On one hand we have interfaced the package TPL as a convenient way to provide cleanly labelled tables, on the other hand we have developed what we call a chinese menu which provides 20 standard options under which time series data can be presented together with calculations such as percentage change over periods of time.

e) Graphs

There are many plotting and charting packages available. As well, several APL macros have been developed for that purpose. Our newest addition is a service called TELICHART which will be described later on.

In analytical software the saying holds true: "Different strokes for different folks". CANSIM data are not only used by experts in econometric analysis and statistical analysis, they are used increasingly by less trained users. Thus there must be a range of software tools which strike a balance between power and flexibility on one side and complexity of use on the other. We found it convenient, for purposes of discussions, to plot these analytical software tools on a chart with axis corresponding to the above two terms. Thus we can discuss in which area further work needs to be done.

CURRENT DEVELOPMENTS

CANSIM has become a vital part in the tool kits of statisticians and economists in the public and private sector. This is the result of 14 years of development which started out with a simple basic idea and a modest beginning. Enhancements and further developments were undertaken as the result of users requests and market pressure. We see no reason to change the basic thrust of our approach with CANSIM, however, adjustments in terms of data contents and analytical software may need to be made in order to lay the foundation for future growth. The following following are our current activities in that respect.

TELICHART

During recent months we have introduced a new graphic on-line display service which we call TELICHART. TELICHART utilizes low cost videotex (TELIDON) display terminals and is linked to a subset of our CANSIM Data Base. The user enters simple commands, such as "CPI" and sees immediately on the screen the Consumer Price Index time series displayed as a curve over time. Other commands are available to adjust the automatically provided scaling, to window in on specific time periods, to extend the time period, or to add other time series as curves or bar charts on the same screen. Also, the screen can be divided into a bottom chart and top chart in which different time series can be displayed.

The attractive feature of this new service is the depths of the CANSIM data base, as well as the low cost graphic access and display facilities

provided through the TELIDON technology. The service was introduced in May 1983 and has met very favourable acceptance. While similar plotting and charting facilities have existed before, for the first time ease of use and much lower costs are offered for such a service. We anticipate growing demand for this particular service which we offer currently in market test mode.

Micro Computer Interface

The raison d'etre for CANSIM is to provide key statistical information to users on their screens on their desks. Thus we are servicing the market needs of those users who are familiar with computer equipment and have the necessary training to operate them. Rapid changes are taking place in this market through the installation of an increasing number of micro computers. We anticipate that users of these micro computers wish to receive CANSIM type information via public communication lines and we are currently developing a streamlined interface between CANSIM and micro computers. The market for such services seems to be developing very rapidly. We receive an increasing number of inquiries about such a service. Already, we have created an interface between our CANSIM data base and micro computers in allowing to download selected time series such that they can be manipulated using standard micro computer software such as spread sheet calculation, statistical packages, plotting software and other manipulation facilities.

Another interesting development is that micro computers become videotex compatible. Through additional hardware or software a micro computer can be made to behave like a videotex terminal. The whole area of micro computers is a most exciting development and we feel that it will increase significantly the market for electronic information in the near future.

Electronic Mail Services

Public electronic mail services have been introduced in Canada. Again, our orientation here is to support screens on our users desks with relevant information. The information could be of textual nature, such as the announcements of the most recent results of our statistical collections, but could also be of more specific nature, such as selected statistical tables. Through public electronic mail services we will enhance our electronic information services from pure statistical numbers to general type information.

Meta Data

The concept for documenting time series data in CANSIM was established in the late 60's. CANSIM documents each time series as an individual data item. As a result, our documentation of currently 350,000 time series has become rather bulky. While we have introduced keyword searching facilities on the existing documentation, we are in the process of re-evaluating our whole approach towards documenting statistical data. This we do with the view of simplyfing the concept and making the documentation tight and precise. In particular, the matrix or table approach has to be considered where an individual time series would be referenced as a particular data cell within a larger cross-classified table.

CONCLUSION

Statistics Canada as a central statistical agency is in a key position to develop the infra-structure in which national statistical information and analytical tools are combined to address the needs of the right spectrum of users, from a presentation of statistics in chart form, to fact sheets containing data from a variety of sources in easy to view form, to powerful econometric modules. With the move of our society to supplement paper based communication with electronic communiction means we will continue paying attention to this part of our communication obligations.

Diversification in Statistical Data Bases and its Consequences

Helen C. Poot

Abstract

In this brief report the author discusses the consequences of the growth and diversification in statistical data bases as evidenced at Data Resources, Inc. (DRI). In general, one sees time-series being used in closer conjunction with cross-sectional and textual information. This has a great impact on the types of software used. The variety of data available requires more diverse methods of quality control. The amount and scope of information now online requires that descriptive and reference materials be made available online, particularly in the area of current awareness.

I. Introduction

The rapid growth and diversification which has taken place in statistical databases over the last decade has put greater demands upon data base producers and vendors. Originally the information contained within these data bases consisted primarily of macroeconomic time series. The software applied was almost exclusively analytical, and the end users, working with a limited number of variables with which they were usually already familiar, did not require the assistance of information or data specialists. Much has changed since that time, as is evidenced by the current situation at Data Resources, Inc. (DRI). Macroeconomic information is now only a part of the data base system; data coverage extends into a variety of areas. Time-series are now used in conjunction with cross-sectional data and textual information. This has required the development of new software, facilitating the retrieval of these new data and their integration with or conversion to time series. The expansion of data offerings in terms of the subject areas covered and the actual form which the data may take has led to more active interaction between the data or information specialist and the user and has required the development of more descriptive information online.

II. Growth in Data Base Coverage

In 1969 DRI offered one data base containing one to two thousand time series on the U.S. economy. This was supplemented in 1971 by a regional data base, providing information at the Census Region, State and SMSA level. Between 1973-1978 these data bases had themselves grown dramatically, and were supplemented by 36 additional data bases in the international, financial, microeconomic/interindustry and energy areas. In 1983, DRI's data bases exceed 80 and contain over 10 million time series.

Although this growth in the amount and range of statistical information is important, of perhaps more significance is the fact that included in the current roster of DRI's data bases are those which do not, strictly speaking, consist of time series. Cross-sectional data and textual information represent a growing part of the data base system. Cross-sectional data would range from the results of the 1980 Census to fundamental, descriptive information on securities. Textual data bases available online include DATAPRO, which provides information on computer software and hardware, and DMS/ONLINE, which describes the defense and aerospace industries. The situation is compounded by the fact that statistical information may be embedded in a textual data base and that, on the other hand, some data, maintained in time-series mode, also lend themselves to cross-sectional analysis.

III. Software Diversification: Data Requirements

The growth in the number and types of information available has necessitated a diversification in the software used in accessing them. Statistical data bases were in the past associated almost exclusively with a powerful, analytical language permitting efficient access, display and analysis of data. This remains the case for the most part; clients use EPS Plus, DRI's proprietary language, to conduct 80 percent of their work. Because they were originally designed for time series analysis, however, the analytical languages have had to be expanded or supplemented to include search capabilities and to handle better cross-sectional information.

The addition of textual data bases to the system required that DRI develop a new language, TEXT, which permits full text searching using Boolean operators. In this area statistical data base producers have had much to learn from the original bibliographic and textual data base vendors, such as BRS and Lockhead.

The development of a language with extensive search capabilities for text data bases, then, does not represent a dramatic innovation; such languages have existed before. The ability to

search through a group of statistical data bases in order to determine what data are available on a certain topic is, however, a much needed development. The amount of statistical information currently available is too great for most users to fully fathom, much less to be able to access readily.

In order to meet the consequences of the explosion in the number of statistical data bases, DRI developed another search language, ABSTRACT. In one instance, key word searching is performed on major groups of data bases, rather than on a single bank; search strategies need not be saved and input repeatedly as one goes from one bank to another. Searching is possible on retrieval code, prefixes, the online documentation associated with the time series (on the DRI system each time series may have up to 5 lines of documentation), or a particular data base. Currently it is possible to search through DRI's 22 international data bases at once; it will be expanded to include most other data bases in the near future.

Ultimately, however, the data retrieved become the subject for analysis, as the 80 percent figure noted above indicates. There has to be, therefore, some integration between the searching language and the analytical one.

The ABSTRACT program, then, was adapted to include the ability to route search results to namelists in EPS format which may then be used to retrieve and display data. Although this is a two-step process, involving ABSTRACT and EPS Plus, the user can readily search a vast number of data bases at one time, determine those series pertinent for his or her purposes and then display them and analyze them with very little effort.

To better deal with cross-sectional data, DRI developed RETRIEVE. Unlike ABSTRACT, RETRIEVE is an EPS-based system; it is not necessary to exit one program and go to another. RETRIEVE allows the user to screen the data base so that only data meeting specified criteria will be brought into the workspace. Once there, the data can be analyzed or displayed using EPS Plus.

One of the interesting consequences of dealing with cross-sectional data is that users now have the ability to screen through data bases consisting of time series more efficiently using the same software.

Another consequence is that statistical data may be extracted from quasi-textual data bases using the ABSTRACT and RETRIEVE programs and then manipulated using more analytical languages.

IV. Software Diversification: Applications Requirements

The previous section dealt with the various software required to access the information efficiently. Once this had been done, one finds that the applications made of the data have also grown dramatically.

Initially, the major historical data bases were closely associated with econometric forecasting models, the former being viewed primarily as vehicles for supporting and generating the latter. DRI today has 36 models available for its users. On the other hand, there are over 80 historical data bases on the system, clearly the majority. The number of specialized software packages, however, has proliferated. For the high-frequency Financial and Credit Statistics data base (DRI-FACS), the number of software routines is over 20. The direction is away from providing general, probable scenarios of the future to providing very specific tools and applications which are flexible enough to allow the user to perform almost any function necessary and to pursue very individual interests or queries.

V. Software Diversification: Quality Control

One of the major concerns of statistical data bases has always been quality control. How accurate are the data being provided by sources? If the data are entered manually, how accurately is the task accomplished?

Testing has always been an important part of maintaining a statistical data base. Data sets are checked automatically to verify that, within certain tolerances, the total really does equal the sum of the parts. Indices may be evaluated by displaying percent changes and comparing them to those published by the source. If a source agency typically publishes the change between the current observation and the previous one, this difference is calculated on the system and also compared with the published.

In the latter two instances, sight proofing remains a major ingredient of quality control. As the amount of statistical information available in a data base system increases, the need for automatic checking routines will become more essential; it will become humanly impossible to keep up.

Not all statistics lend themselves to these types of testing. Financial data, such as interest rates, futures prices or bond yields, are among the most significant members of the "difficult-to-test" group. The financial area, however, has exhibited the greatest growth among statistical data bases. New quality control techniques have had to be developed and still need refinement. Currently, testing may include verifying that highs are always higher than any other price quotation for that period, and that the lows are lower; other checks involve running standard deviations and setting certain range limits. This kind of testing will determine outlyers, but it will not catch all errors. An important question for statistical data base producers in the future will be the degree to which data accuracy and data base integrity can be guaranteed.

VI. Diversification in On-line Aids: Current-Awareness

The proliferation of statistical data bases has necessitated development of online tools which permit the optimal use of the information. This need is underscored by the fact that much of the data are subject to frequent and extensive revision due to changes in seasonal factors, benchmarking, definitional or methodological changes.

Stored with each time series on the DRI system are potentially five lines of documentation containing a full name (GROSS NATIONAL PRODUCT instead of GNP), units, seasonality, source and source document. Also stored as part of the series' intrinsics are the conversion method and, if applicable, if the data have any embedded gaps (due to disclosure, or weekends and holidays for daily data). Due to the fact that data are not constant and are being revised so frequently, another feature has been added to the series' intrinsics, the revision status. It is now possible to "flag" a series as being in the process of being revised; if accessed, a notation to that effect is sent to the user.

The number of current-awareness files has increased greatly. In 1970 DRI had a text service providing online access to major statistical news releases from major government sources on such subjects as consumer prices, industrial production, and so on. Another text service provided information on any additions, deletions or revisions which may have occurred with the data base system. Neither text service had full-text searching, and searches could only be limited by specified time parameters (before or after a certain date).

In order to meet the needs of a massive data base system, these first, inflexible current-awareness services are in the process of being enhanced. Full text searching is now possible for the services monitoring changes within the data base system (additions, deletions, name changes). A user may screen on a series mnemonic, bank name, or subject area using the TEXT program discussed above. The new service will be similarily enhanced in the near future.

Current awareness needs vary from data base to data base, depending on the kind of information stored within them. There has been in recent years the addition of several current awareness files to the system which are very specific and unique to certain data bases. For high frequency or high priority data bases these files would include those listing everything that had been updated during that day. For data bases where the statistics are extremely volatile, files listing data sets under revision would be maintained online. Calendars are accessible online so that users may know when to expect certain data to become available.

VII. Diversification in On-line Aids: Reference

Because of the growth in size and diversity of the statistical data bases available, the need for more descriptive information online about the data is becoming more important. One has seen for many years the online dictionaries for programming languages, where by entering 'HELP' or 'EXPLAIN' the use and proper syntax of certain commands would be described; this is now becoming more important for the data themselves. Within the U.S. macroeconomic data bases, for example, one may have three different definitions of inventories and five different trade balance variables. The user needs ready access to these definitions so as to make the best and most appropriate choice. Financial analysts need ready access to more descriptive information about financial instruments: dates of issue, amount issued, yield to maturity, etc.

Not all these tools are currently available. In the short term, what is seen with statistical data bases is the increased participation of data specialists. Data consulting is now an integral part of DRI's service, where the user may address his or her questions to someone who works closely with that information.

VIII. Conclusion

The scope and size of statistical data bases has grown dramatically in recent years. This has increased the responsibilities of the data specialists both directly and indirectly. More online tools, in terms of new software and current-awareness or reference materials, are needed. The data specialist must take a more active role in working with the end user.

Econometric Time Series on DIALOG

Robert T. Lundy
DIALOG Information Services, Inc.
3460 Hillview Avenue
Palo Alto, CA 94304

Abstract

DIALOG Information Services, Inc., the leading commercial provider of online non-numeric database services, has also been involved in the development of numeric databases, particularly time series. This report describes the efforts being made to adapt an essentially text-oriented retrieval system to the needs of users of econometric time series.

The DIALOG Information Service has for some time been interested in the problems inherent in the handling of econometric time series. This report describes the service currently being developed to deal with this interesting and valuable class of databases.

The DIALOG service originated as a bibliographically-oriented system. All search and display functions have been oriented towards picking up text strings based on identity and proximity to other defined strings and displaying them in a manner appropriate to text handling. Numeric items have been treated as just another text field. They can be searched for and displayed, but no computations or other analysis can be performed.

This approach works well as long as the number of numeric items remains small and algebraic comparisons are not often required. DIALOG has for years had a number of databases in which numeric items have been available for display only. These have usually been simple scalar items, and no computational or other analytic capabilities have been available for them. However, with the advent of major time series such as the Bureau of Labor Statistics time series, DIALOG has had to revise its approach to numeric data in general and time series in particular.

There are several issues to be considered when planning a database for online search and analysis:

1. Finding the desired data.

   In the context of the BLS database, for example, this might be one of the hundreds of price index series available.
   It is in this area that DIALOG's original query scheme has its greatest value. Instead of having to memorize an obscurely named keyword, derived from the painstaking study of a thick codebook, the DIALOG user can select the desired (group of) series by means of a set of meaningful keywords or a context search through a descriptive paragraph. This feature is especially useful when searching for series that do not have well-known acronyms such as GNP.

2. Displaying the data once found.

   Formatting the data in a meaningful way is not a severe problem conceptually. However, two problems can arise when dealing with time series:

   a. How much of the series should be displayed.

      Users may or may not be interested in seeing the whole file. Generally, DIALOG's display scheme calls for the user to select one of a limited number of formats in which the data could be displayed, and none of the formats permitted modification. This is being modified to permit the user to display only the time range of interest.

   b. How much ancillary information should be displayed ?

      There may be footnotes and qualifications that are vital to the understanding and interpretation of the data. Methods need to be developed to minimize the chance that this information will escape the user's notice without cluttering up the display to the degree that the user will abandon it and go off to a competitor.

3. Generating Reports

   It often happens that the data in the form of a single series is far more meaningful if displayed in conjunction with one or more other series or other forms of data. For this purpose a report generating program is being developed. However, for a general service such as DIALOG, most report generators are either too restrictive or too complicated to be useful. We are attempting to solve this problem with a system that will operate in three different modes depending on the sophistication of the user. These modes range from a very terse and arcane command with very few prompts or other attempts at 'user-friendliness' to a verbose prompted menu-oriented interactive mode in which the user is led very carefully through the specification of a report with detailed explanations at every step.

4. Predefined Reports In many cases, users may want the same kind of report. For example, a listing for selected products of the previous 3 year's exports arranged by country of destination. As such ,commonly desired reports are identified, we will try to set them up as 'canned' report formats that can be generated simply by giving their names as part of a terse command.

5. Interfacing with a computational analysis system.

   DIALOG is strictly a search-and-retrieval system with no intrinsic computational facilities of its own. Consequently, a mechanism must be found to enable users to integrate and use the various data items in meaningful ways. A group of price series may be useful as a set of independent tables, but it is more useful still when combined into a single table or processed through an econometric model.

   At DIALOG we are addressing this problem by setting up an interface between the DIALOG time series files and the SAS system.

6. Interfacing with Microcomputers

   An increasing number of users are interested in downloading data from services such as DIALOG and processing it on their own machines. To answer this need, we are developing an interface that prints out the selected series in the Visicalc DIF format. When captured on the user's micro, it is then ready to use with any program (and there are several "visiclones" around now) that can read it.

7. Support of User Equipment

   The customer base at DIALOG is characterized by extreme heterogeneity in the kinds of hardware that is available. DIALOG has traditionally dealt with this problem by aiming for the lowest common hardward denominator - no assumptions whatever are made about the hardware configuration, nor are any attempts made to support any devices with even slightly non-standard features. Only the page size (for CRTs) and line length may be reset to take advantages of differences between terminals.

   For the econometric time series service, however, graphics and complex reports will inevitably be required, and this will require that the extended DIALOG system be able to support a variety of graphics terminals.

Biographical Note

Robert T. Lundy has a B.A. in Human Ecology from the University of California at Santa Barbara and an M.A. and Ph.D. in Demography from the University of California at Berkeley. His research interests, in addition to the development of online databases in demographic and related areas, include microdemographic computer simulation, the health impacts of energy systems and the demography of Japan during the Tokugawa period.

Evolution in Storage and Retrieval: The LABSTAT Data Base and Software System

Gwendolyn L. Harllee

Bureau of Labor Statistics

June 1983

ABSTRACT

The use of an integrated data bank for time series data was initially introduced at the Bureau of Labor Statistics two decades ago. The data bank has been available on line since LABSTAT was introduced in 1977. This paper summarizes some of the major developments in the data bank and access software and discusses some enhancements currently being considered.

The Bureau of Labor Statistics is responsible for the production of a number of economic statistics, including national and local area estimates of employment and unemployment, the Consumer Price Index, Producer Price Index, Employment Cost Index, measures of productivity, and others.

The surveys that produce these statistics are processed by Bureau staff who do much of their work from specific data bases designed to handle the screening, estimation, and other processing in the particular survey. Each survey, or program, has a data base and processing system designed to perform efficiently in that survey. Once the estimates are calculated and become part of a time series, there is more similarity than difference in processing requirements for analysis and publication.

Since 1963, the BLS has used an integrated data bank for time series analysis and tabulation. The initial system, the BLS Information System, was a tape system with tabulation capabilities and interfaces to statistical processors. Over the next decade and a half, the Bureau's systems moved to on-line data bases.

In 1977, a small group of systems analysts at the Bureau developed a time series data base system which allows on-line storage, manipulation, and display of the Bureau's summary data. This system, called LABSTAT, rapidly became established as a useful tool for economic data analysis.

From its beginning, the LABSTAT design provided a standard time series record format, a common central data entry facility and a variety of retrieval options. The data entry procedure was designed with a central main program handling common functions and a group of separate subroutines for processing the various input formats produced by the Bureau's survey systems. Each survey subroutine processed the appropriate survey estimates and produced the standard LABSTAT update format. The retrieval options available included the Macro Data Language (MDL), which was developed at the Federal Reserve Board and adopted for use in LABSTAT. MDL provided processing capabilities ranging from simple percent changes to limited regression or seasonal adjustment. Most of these operations could be done interactively as well as in batch operations. In addition to MDL, other retrieval options were available through TRIM, a BLS-developed facility which provided generalized data manipulation capabilities for data bases such as LABSTAT which use the TOTAL Data Base Management System. Because of the flexibility of TRIM, data could be extracted from the data base for generalized statistical processors including SAS and SPSS. The user could also develop tailored formats for use with other programs.

One of the earliest enhancements provided security. The security mechanism in LABSTAT restricts access to protected data. All non-zero data are encrypted as they enter the system. The data reside on disk in permanently encrypted form. LABSTAT's security mechanism decrypts data only after determining that the requesting user is authorized to access the requested data.

Security protection of two types is provided, observation protection and date protection. Observation protection applies to statistics which are permanently withheld from public release for such reasons as confidentiality. These data are made available only to analysts authorized by the survey manager. Observation protection may be defined at different levels, with access to the most secure data limited to a few individuals, while other data may be made available to a still restricted, but larger group of analysts. Date protection provides security for data which are stored in the data base prior to public release. This facility allows survey managers to store estimates as soon as calculations are complete and use

the LABSTAT facilities to produce publication tables and charts for release. The security mechanism protects the data up to the specified release date and time, allowing access only to those individuals authorized by the program manager. Unlike observation protection, date protection is not permanently encoded with the data estimates. For each survey using date protection, the system stores the date of the earliest observations to be protected, the release date and time, and the code identifiers of users who are authorized early access.

For example, estimates for a survey may be updated into LABSTAT several hours before public release. The date protection mechanism will prevent release to the general user community, while allowing the authorized analysts to use LABSTAT facilities to produce publication tables and charts. At the specified release time, the data are released to all users without intervention of any kind.

The observation and date protection facilities have enhanced the utility of LABSTAT by making it feasible to store data which are available only to an appropriately authorized user community. Other users, attempting to access protected data, may have their listings footnoted to show the reason for withholding the data.

Perhaps the most significant enhancement of the system's analytical power is the development of an integral data access mechanism in LABSTAT. The TRIM step, which extracted data for many applications, has been replaced by a data retrieval module which can access a user's private working file as well as the LABSTAT data base. With the extension of LABSTAT's data retrieval module to interface directly with SAS and other processors it became unnecessary to execute a separate data extraction step for most LABSTAT applications. The expansion of the data access module also makes it possible to calculate new series, store them in an MDL working file, and use the calculated series together with LABSTAT series in charting or other applications. This working file facility also allows users to enter data from outside sources and use LABSTAT's analytical and display tools on the full set of data from the data base and the user's own working file. The pre-release protection and working file extensions work together to facilitate the production of publication tables and charts.

The introduction of the LABSTAT Press Release Service increased system utilization significantly by making available on-line the text and tables of Bureau Press Releases. The Press Releases are stored in a separate TOTAL data base with its own access software and security system. The BLS Regional Offices, in particular, find this service to be of value in getting information on a timely basis. Partly because of the LABSTAT Press Release Service, the Bureau is now able to offer on-line public access to its Press Releases. This service was introduced last year, and while it is not a part of the LABSTAT system, the existence of LABSTAT made it relatively simple to establish machine-readable versions of Press Releases.

Data coverage has more than doubled from approximately 60,000 series in the beginning to approximately 140,000 series containing 1.5 million yearly records. Data from more than twenty different BLS surveys and some statistics from the Commerce Department's Business Conditions Digest are now available in LABSTAT. This expanded data coverage facilitates analysis of data from different surveys and also increases the number of BLS surveys which are available to the public on the standard LABSTAT export tapes. All BLS survey data in LABSTAT may be extracted for public dissemination at relatively low cost. Many of these data tapes are prepared on a regular basis for researchers in other

government agencies, universities, information services, and a variety of other organizations.

LABSTAT system usage has sometimes doubled in a single year. Overall usage has increased ten-fold since the first year of operation. A large portion of this access is for information retrieval or special research projects. There is an increasing usage of LABSTAT to produce publication products, including tables and charts. The charting facility in LABSTAT is provided by a Bureau-developed system known as SCS, or the Statistical Charting System. This system produces publication quality line charts of time series data with simple user input instructions. Using the Bureau's Table Producing Language, TPL, users have available a direct interface to the LABSTAT data base for TPL's wide range of calculation and cross-tabulation capabilities as well as photo-composition.

Many further enhancements are desirable. Users of LABSTAT must still request series using 16 character identifiers which sometimes seem clumsy or arbitrary. This means, in most cases, that thick series directories must be available for reference. This inconvenience to the users limits the utility of the system. Further, the cost of producing and printing the series directories is increasing significantly. For these reasons, we are beginning to develop a data access option which will allow users to query the system for data using English-like descriptors or by responding to system prompts. When this facility is in operation, users will be able to enter requests for classes of data such as "employment in New York's apparel industry." the processing of queries will be possible because the system will include directory or dictionary information and users can get an up-to-date description of data base content on-line.

Another requested enhancement is incorporation of more detailed footnote information. Statistical publications typically include a variety of footnotes giving significant explanatory information. The most frequently occurring footnotes, such as "Rounded to zero", "Continuity break", or "Preliminary estimate" are coded in the LABSTAT data base and may be displayed automatically with the data. Other notes are combined in a single category and the LABSTAT footnote will simply read "Footnoted in Publication." The user must go to the printed material to obtain the text of the applicable footnote. The inclusion of the full footnote text is an enhancement which would be of significant value to analysts using the data base.

The use of an integrated data bank is firmly implanted in BLS operations. The data bank has, in many cases, reduced costs and increased timeliness of delivery. With improved facilities for accessing, manipulating, and displaying data, the system becomes useful to a broader audience and meets an expanding set of data requirements.

INTERACTIVE INFORMATION MANAGEMENT WITH EPS

Stephen R. Childs
Data Resources, Inc.

This paper provides an illustration of interactive statistical data base management. Its focus is on the use of the EPS (Econometric Programming System) software language as a manager of information. EPS is usually employed to analyze time series information using econometric techniques. The paper will illustrate some EPS capabilities using several data bases developed by Gnostic Concepts, Inc., a subsidiary of DRI (Data Resources, Inc.) specializing in market analysis of the Electronics Industry.

INTRODUCTION TO EPS EPS is a proprietary software developed and maintained by DRI for a real time analysis of economic and financial information. It is an interpretive, command driven language which mimics English sentence structure. Its syntax rule is simple and consistent:

Command Verb	<	Option block Adverb	>	Command Body Object Expression

eg.,

PRINT <DOWN> %CHANGE(GNP)

There are 130 commands and 250 options which combine to give the user a vast menu of possible requests. The command body expressions may utilize information objects (items) as well as some 60 operators or 275 functions to tailor user-specific information transfer, manipulation or presentation.

EPS has been focused for use by quantitative business analysts in econometric model building. Clients access DRI's projections of aggregate time series as exogenous inputs to their own satellite business models. There is an historical bias in EPS documentation toward time series structures which can be easily accessed, powerfully manipulated, and beautifully presented with either high resolution graphics or tabular report generation.

Time series might be thought of as particular slices from multidimensioned arrays. EPS is less well-known for its complete array manipulation capabilities. Arrays in EPS are multidimensional information objects having homogeneously classed elements. When an array is defined, the structure requires each dimension to be indexed, or mapped, by labels or numbers so that selection of partitioned subarrays is simple and display is meaningful. The cell values of arrays most often are scalar (numeric) valued but may alternatively be string (alphanumeric) valued, or boolean (true/false) valued, etc. For a list of EPS classes see Appendix I.

COMPOSITELY-CLASSED EPS ARRAYS Recently, DRI released some significantly improved capabilities for data base management with EPS. Formerly, users were required to house alphanumeric information in one array and numeric information in another. This necessitated careful software development to ensure that the different arrays were consistently maintained and manipulated. It is now possible to construct an element-class which is composed of scalars, strings, dates, booleans, time stamps, etc. This enables the analyst to produce one structure for all cross-sectional attributes related to each sample observation.

REGIONAL MARKET PATTERNS (RMP) DATA BASE We will illustrate composite classsed arrays with RMP, an annual survey of electronic equipment production facilities. This snapshot records the geographic distribution of production by equipment type for currently 2474 sites. These data have been housed in a one-dimensional EPS array with 2474 records. The composite class contains the following eight fields:

```
COMPANY : STRING
DIVISION : STRING
CITY : STRING
STATE : STRING
ZIP : STRING
PHONE : STRING
GCICODE : SCALAR
PRODUCTION81 : SCALAR
```

PRODUCTION81 represents the 1981 value of equipment production in millions of dollars. It is paired with GCICODE, a product classification scheme developed by Gnostic Concepts., Inc., which is far more detailed than government product classifications. There are over 250 unique products classified within RMP's 2474 sites. The GCICODE is a six-digit hierarchical coding system summarized to the second level below:

```
?DO FAMILYTREE<DEPTH=2>(500000)


500000  ELECTRONIC EQUIPMENT
510000    BUS/RETAIL/EDUC
520000    COMMUNICATION
530000    CONSUMER ELEX
540000    COMPUTER EQUIPMENT
550000    GOVERNMENT/MIL.
560000    INDUSTRIAL ELEC
570000    INSTRUMENT
```

FAMILYTREE is an EPS application program or routine which discloses the codes associated with equipment descriptors. Another routine, HIERARCHY, is used to detail an exact code's product description.

```
?DO HIERARCHY(545221)

(545221)

 5       ELECTRONIC EQUIPMENT
  4       COMPUTER EQUIPMENT
   5       TERMINAL/WORKSTATION
    2       CRT TERMINALS
     2       GRAPHIC TERMINALS
      1       STORAGE TUBE DISPLAY
```

REPORT WRITING Arrays provide natural structures for report writing within EPS. The user simply prints the array, or a specific partition of the array. The syntax for selecting a partition of the RMP array by records and by fields is straightforward:

PRINT RMP [recordnumber] <<fieldname>>

```
?P RMP[1927]

                    COMPANY          DIVISION      CITY      STATE   ZIP

RMP[1927]  ROBERTSHAW CONTROLS  NEW STANTON  YOUNGWOOD     PA    15697


                   PHONE      GCICODE  PRODUCTION81

RMP[1927]  412-925-7211    561200      38.400

?PRINT RMP[1787]<<GCICODE>>

RMP[1787]<<GCICODE>> = 511500
```

The first example illustrates the reporting of one record for all attributes, while the second selects one record and one field. Finally, a selection of fieldnames can be requested as illustrated below:

```
?PRINT<DOWN>RMP[12]<<NAMELIST(COMPANY,GCICODE,PRODUCTION81)>>

                RMP[12]<<NAMELIST(COMPANY,GCICODE,PRODUCTION81)>>

COMPANY                          ACE RADIO CONTROL
GCICODE                                 531900
PRODUCTION81                             1.800
```

RECORD SELECTION AND SORTING Partitions of the data base follow selection rules established by the user. EPS offers some functions which are easy to use. The COMPRESSMAP function produces a list of records satisfying the user's selection criterion.

```
?CALIF=COMPRESSMAP(RMP[*]<<STATE>> EQL "CA")
```

The argument passed to COMPRESSMAP asks that all of RMP's sites [*] be compared in the state field, to the string ("CA"). For each site's result that is true, its record number is placed into a vector of record numbers called CALIF. We can discover how many California sites there are in RMP by typing:

```
?WHATS CALIF

   CALIF          ARRAY(FROM 1 TO 713)
                      DECS: 0
                      713 SCALAR ELEMENTS
```

We can further partition RMP into California producers of computer equipment, where GCICODE begins with 54 in its first two digits. The DIV operator allows us to truncate the six digit GCICODE to its first two digits. We compare this two digit product scheme to the value 54 to produce a new vector of record numbers restricting California producers to those who produce electronic data processing (EDP) equipment.

```
?EDP@CA=COMPRESSMAP(RMP[CALIF]<<GCICODE>> DIV 10000 EQL 54)

?WHATS EDP@CA

   EDP@CA         ARRAY(FROM 1 TO 264)
                      DECS: 0
                      264 SCALAR ELEMENTS
```

Let us further restrict our subset of 264 California EDP sites to those whose 1981 value of production exceeds $150 million. We can also sort this final subset by the field "CITY" using the KEYSORT function.

```
?LARGE=COMPRESSMAP(RMP[EDP@CA]<<PRODUCTION81>> GTR 150)

?WHATS LARGE

   LARGE          ARRAY(FROM 1 TO 9)
                      DECS: 0
                      9 SCALAR ELEMENTS

?SORTED=KEYSORT(RMP[LARGE]<<CITY>>)
```

Finally, we wish to display some fields for our nine large California EDP producers, sorted alphabetically by city. The leftmost column of the display indicates the record number within the RMP data base. Note that a variety of equipment types is indicated in the column labeled GCICODE. These are particular computer equipment classifications.

```
?P RMP[SORTED]<<NL(COMPANY,CITY,GCICODE)>>

RMP[SORTED]<<NL(COMPANY,CITY,GCICODE)>>
```

	COMPANY	CITY	GCICODE
1148	HEWLETT-PACKARD	CUPERTINO	542300
1150	HEWLETT-PACKARD	CUPERTINO	541000
1580	MOTOROLA	CUPERTINO	541200
2445	XEROX	HAYWARD	545210
2084	SPERRY	IRVINE	541200
1243	IBM	SAN JOSE	542400
1244	IBM	SAN JOSE	544130
1521	MEMOREX	SANTA CLARA	542300
1614	NATIONAL SEMICONDUCTOR	SANTA CLARA	540000

LINKING TIME SERIES FORECASTS TO RMP Gnostic Concepts also provides forecasts which can link to and augment the RMP data base. Econometric models are run each quarter to update projections of electronic equipment production. These forecasts are also indexed by the product category codes found in RMP's GCICODE. As a base scenario, each site's 1981 value of production is grown by the national growth factor for its specific equipment type.

Following the procedures outlined for using the routine PROD@SITE, we rebase the equipment forecasts in the array E832@VAL. These results are stored in the growth factor array EQUIP%GR, whose rows identify equipment types and whose columns indicate forecast years.

```
?DESCRIBE PROD@SITE

--- PROD@SITE ---
CLASS: ROUTINE
LONG:   PROD@SITE ..... This routine forecasts the
    production by RMP sites.  It requires a vector of growth factors
    (use routine PCTBASE on retrieved ECON equipment fcst) by equipment
    type.  'PRODUCTION84= DO PROD@SITE(EQUIP833GRFACTORS)'

4 LINES, 1 THRU 4
```

```
?DESCRIBE PCTBASE,E832@VAL

--- PCTBASE ---
CLASS: ROUTINE
LONG:   PCTBASE ..................... This routine indexes an N-
    dimensional array to one of its column vectors. ARGUMENTS: an array
    and a string containing a 'selector' from the column map. e.g.
    %SALES=DO PCTBASE(COMPDATA,"SALES")*100

21 LINES, 1 THRU 21

--- E832@VAL ---
CLASS: ARRAY(LISTED(SCALARS,V(200000,210000,211000,211100,211110,211120,
    211130,211140,211141,211142,211143,211150,211160,211200,211210,
    211220,211250,211300,211310,211320,212000,212100,212110,212120,
    212130,212140,212150,212160,212180,212200,212210,...,...) ),...)

DECS: 0
6140 SCALAR ELEMENTS

?EQUIP%GR=DO PCTBASE(E832@VAL,"A(81)")
```

We again use the routine HIERARCHY to describe the military simulators and trainers product codes. We then display the rebased growth factor array selected for those categories over the specified subinterval.

```
?DO HIERARCHY(555000)

(555000)

 5       ELECTRONIC EQUIPMENT
  5       GOVERNMENT/MIL.
   5       SIMUL & TRAIN

?PRINT EQUIP%GR[V(500000,550000,555000),81 TO 85]

EQUIP%GR[V(500000,550000,555000),81 TO 85]

          1981    1982    1983    1984    1985

500000   1.000   1.117   1.267   1.459   1.684
550000   1.000   1.199   1.367   1.577   1.812
555000   1.000   1.328   1.565   2.001   2.655
```

Finally, we invoke the PROD@SITE routine to construct the forecasted production value, by site, for 1985.

```
?PROD85@SITE=DO PROD@SITE(EQUIP%GR[*,85])

?WHATS PROD85@SITE

   PROD85@SITE  ARRAY(FROM 1 TO 2474)
                     DECS: UNSPECIFIED
                     2474 SCALAR ELEMENTS
```

LINKING TECHNOLOGICAL CONSUMPTION FACTORS TO RMP Gnostic Concepts also provides forecasts of technological consumption factors for over 500 electronic components which are used in the production of particular equipment. These are value-based input/output (I/O) ratios relating the derived demand for a particular component per dollar of production for a particular equipment.

Again, following the instructions described for the routine VAL@SITE, we access the I/O array I0736100. This array contains the consumption factors by 31 types of equipment for a particular type of fiber optic connector over a ten year forecast horizon.

```
?DESCRIBE VAL@SITE


--- VAL@SITE ---
CLASS: ROUTINE
LONG:   VAL@SITE ..... This routine produces the nominal
    dollar demand for a particular component at each site.  It requires
    a vector of production by site and an I/O vector by equipment type.
    'VAL84=DO VAL@SITE(PRODUCTION84,I0842216[*,84])'


4 LINES, 1 THRU 4


?DO HIERARCHY(736100)

(736100)

 7       PASS/ELECTROMECH COMP
  3       CONNECTOR & SOCKET
   6       FIBER OPTIC
    1       CABLE TERMINATION
```

```
?WHATS IO736100

IO736100      ARRAY(LISTED(SCALARS,V(500000,510000,511000,520000,
                521000,523000,526000,529000,530000,535000,540000,
                541000,543000,544000,545000,550000,551000,552000,
                553000,554000,555000,556000,557000,558000,559000,
                560000,561000,562000,564000,570000,571000)),DATED ...)
                  DECS: 5
                  310 SCALAR ELEMENTS
```

Imbedded in these ratios are patterns of technological change over time for particular markets. We note below both the small level of demand as well as the high rate of growth. Finally, we invoke the routine VAL@SITE to produce FO85@SITE, a forecast of demand for fiber optic cable termination connectors, by site, in 1985.

```
?P<ROWMARGIN "%GR">IO736100[V(500000,550000,555000),81 TO 85]

IO736100[V(500000,550000,555000),81 TO 85]

            1981      1982      1983      1984      1985       %GR

500000   0.00003   0.00004   0.00006   0.00008   0.00013     45.47
550000   0.00006   0.00008   0.00010   0.00013   0.00016     25.51
555000   0.00003   0.00004   0.00006   0.00008   0.00011     41.31

?FO85@SITE=DO VAL@SITE(PROD85@SITE,IO736100[*,85])

?WHATS FO85@SITE

   FO85@SITE     ARRAY(FROM 1 TO 2474)
                     DECS: UNSPECIFIED
                     2474 SCALAR ELEMENTS
```

REGIONAL AGGREGATION OF RMP SITES Each market analyst working with the RMP data base and forecasting extensions of it will be interested in particularizing its regionality to his/her own marketing regions. These regions may be developed by utilizing groups of zip codes, states, or phone area codes. In our final exhibit, we illustrate the use of a routine called AGGRMP. This routine requires, as its second argument, some allocation rule for grouping 2474 sites into predefined regions. As an illustration, we build a rule, ZIP1, containing the first digit of a site's zip code. We then execute AGGRMP for both equipment and component demand forecasts for 1985.

```
?DESCRIBE AGGRMP

--- AGGRMP ---
CLASS: ROUTINE
LONG:   AGGRMP ..................... This routine summarizes RMP sites
    into aggregate groupings.  It requires two conformable arguments:
    the data to be aggregated and the aggregation rule.  'MYREG=DO
    AGGRMP(PRODUCTION85,STATERULE)'


9 LINES, 1 THRU 9

?ZIP1=RMP[*]<<ZIP>> DIV 10000

?PROD85@ZIP1=DO AGGRMP(PROD85@SITE,ZIP1)

?FO85@ZIP1=DO AGGRMP(FO85@SITE,ZIP1)

?P<DOWN,COMMAS>PROD85@ZIP1,FO85@ZIP1

   PROD85@ZIP1  FO85@ZIP1

0    29,400.341     5.751
1    20,277.052     2.576
2    11,608.108     3.050
3    11,557.422     1.719
4    14,104.646     4.199
5    11,512.881     0.827
6    15,972.038     7.403
7    18,047.690     4.476
8    14,445.429     1.683
9    52,809.158     5.993
```

```
*  AMSYNOPSIS (AMSYNOPSES)
   ARRAY (ARRAYS)
*  ATOMIC (ATOMICS)
   BOOLEAN (BOOLEANS)
*  BUNDLE (BUNDLES)
   CLASS (CLASSES)
*  CONVERSION (CONVERSIONS)
*  COVERING (COVERINGS)
   DATE (DATES)
*  DISTRIBUTION (DISTRIBUTIONS)
*  DRAWING (DRAWINGS)
*  ENDMARKER (ENDMARKERS)
*  EQSYNOPSIS (EQSYNOPSES)
   EQUATION (EQUATIONS)
*  EXECKEYWORD (EXECKEYWORDS)
*  EXTRACTION (EXTRACTIONS)
*  FIELDFORMAT (FIELDFORMATS)
*  FIELDTYPE (FIELDTYPES)
*  FILETITLE (FILETITLES)
   FREQUENCY (FREQUENCIES)
*  GROUP (GROUPS)
*  INTERPOLATION (INTERPOLATIONS)
   INTERVAL (INTERVALS)
   KNOWNWORD (KNOWNWORDS)
   LSSYNOPSIS (LSSYNOPSES)
*  MAP (MAPS)
*  MATHTERM (MATHTERMS)
*  MATRIX (MATRICES)
   MEMO (MEMOS)
   NAMELIST (NAMELISTS)
*  NULLMARKER (NULLMARKERS)
*  NUMERIC (NUMERICS)
*  PADTRIMTYPE (PADTRIMTYPES)
   PICTURE (PICTURES)
*  PLOTSPEC (PLOTSPECS)
*  POLYNOMIALDISTRIBUTEDLAG (POLYNOMIALDISTRIBUTEDLAGS)
   PORT (PORTS)
*  PORTATTRIBUTE (PORTATTRIBUTES)
*  PORTKIND (PORTKINDS)
*  PORTUSE (PORTUSES)
*  RANGE (RANGES)
*  RECORD (RECORDS)
*  REFERENCE (REFERENCES)
   ROUTINE (ROUTINES)
*  ROWLIST (ROWLISTS)
*  SASYNOPSIS (SASYNOPSES)
   SCALAR (SCALARS)
   SERIES (SERIES)
*  SLASHING (SLASHINGS)
   STRING (STRINGS)
   STUBLIST (STUBLISTS)
   SYNOPSIS (SYNOPSES)
*  SYSTEMFILEKIND (SYSTEMFILEKINDS)
*  TABLEAU (TABLEAUX)
   TEMPLATE (TEMPLATES)
*  TICKER (TICKERS)
*  TIMESTAMP (TIMESTAMPS)
   TOOL (TOOLS)
*  UNSPECIFIED (UNSPECIFIEDS)
*  VALUELIST (VALUELISTS)
   VECTOR (VECTORS)
   VERSION (VERSIONS)
```

Meta data : an experience of its uses and management

Roger E. CUBITT

Data Processing Management Unit, Statistical Office of the European Communities, Luxembourg, Grand Duché

Abstract

This short paper presents some of the problems surrounding the management and use of the meta data which is associated with most types of statistical data. Current approaches to the problems which are being taken within the Statistical Office of the EC are described.

1. CONTEXT

The Statistical Office of the EC (SOEC) is a service Directorate General with responsibilities for the collection, supply and analysis of data for the Institutions of the EEC and in particular the Commission of the European Communities. It is a very large user of computer processing power (20 to 25 Terra-instructions/year) and is responsible for one of the largest Socio-Economic Time Series Databases in Europe called CRONOS. This database, which was constructed in-house, contains of the order of 1.3 million time series covering a large range of statistical domains for all EEC countries and in some cases other developed and developing countries. This database represents only a small part of the total data processed by the SOEC but is at the present time the most visible part in that it is distributed both via data service companies to the public and via X25 networks to the contributory member country statistical offices and governments.

2. THE PROBLEM

Since the implementation of the network based SOEC distribution policy some two years ago it has become increasingly obvious that the major restriction on the use of our data has been the problem of description. In other words "How do I (the user) find out what data is available in the Database, select the data I require and ensure that when I find it, it represents what I want?". The SOEC has, like all statistical environments, a classification scheme which is partly defined internally and partly dictated by external constraints (e.g. International Common Agreements). This classification is inevitably hierarchical which in practice means that a search procedure using the classification structure is only of any real use to people with an a priori knowledge of the classification scheme and procedure. It is in this light that the SOEC began some investigation work into the possibilities of using meta-data to provide both alternative non-hierarchical access paths to data and descriptions of what had been found.

3. INITIAL ORIENTATIONS

Two parallel investigations were initiated at about the same time which eventually cross-fertilized to provide a hybrid approach to our immediate problems. One investigation centred on a theoretical data modelling/data dictionary approach, the other was based on a pragmatic examination of the particular instance represented by the CRONOS system. The former led to proposals for a classification orientated hierarchical data model with an associated data description language; the latter resulted in proposals to use keyword based documentation with associated interrogation facilities. As it was obvious that a keyword facility would become very cumbersome if applied to all the time series in CRONOS, the data model provided a means to describe a level in conceptual terms which was logically higher than that of a single series, but which did not

exist in the CRONOS system as implemented. CRONOS also provided a specific case of data identification which enabled the attribute description and classification facilities in the theoretical model to be refined and developed.

4. THE CURRENT STATE

Once the keyword approach had been accepted, a pilot application was developed to demonstrate how such a facility would work for both interrogation and documentation purposes. To do this the structure and content of a specific sector of the CRONOS data base was analysed by a professional documentalist, particularly to clarify the problems resulting from the use of this approach to statistical data (e.g. the use of classification codes and abbreviations as keywords). There were also particular local problems which resulted from the multi-lingual environment in which the system had to operate. This had particular effects on the provision of synonyms and descriptors. The problem of description was tackled by comments facilities which could be associated with each of the elements in the logical model and which were themselves classified by type. The result of this work was a prototype facility which enabled a user with no prior knowledge of the data to discover the contents of the base either via a specification of keywords or hierarchical interrogation or both. Both access paths lead to the definition of a logical group of data items from which specific identifiers can be constructed. Meta-data relating to individual time-series can be presented in tabular form at the logical group level and descriptions are accumulated through the hierarchy as required. Classification plans can be produced automatically both in printed form and on magnetic media for external clients. A spin-off benefit has been the ability to carry out audit activities on the content of the data base.

5. PRODUCTION IMPLEMENTATION

The pilot facility appears to provide a potentially useful service for a number of types of user. The major outstanding problems are the following.

Meta-Data Management

In practice the management of the meta-data as it has been extended presents a task which is of non-trivial size and complexity. Experience to date has led us into the consideration of a DBMS solely for meta-data. This management task can be alleviated to some considerable extent if the modifications to data content in the CRONOS database are submitted via or in parallel to modifications to the meta data. This contradicts an original objective from the data model investigations which was to provide a more loosely coupled facility which could be applied to more than one type of data base. In fact the documentation investment made from the keywork point of view would appear to demand application to more than just one single data base.

Data access

At present the pilot facility indicates the location of data described by the user. Access to that data is then obtained via the particular application facilities. Closer direct integration again presents problems of coupling particularly in the multi-machine environment which prevails in the Commission. Advances in distributed data-base facilities may provide answers here.

Confidentiality/Security

Application of a meta-data management system to a number of different data bases is leading us towards a centralised control of access rights for all SOEC system users. This again presents particular problems in a multi-machine environment. Not all of our data is fully public and at the present time we avoid disclosure by only documenting via the meta-data those sectors which are truly public. This has disadvantages for both internal users and also for the completeness of the audit procedures mentioned earlier. At present no convenient solution presents itself to ensure that accidental disclosure of data existence does not occur via the meta-data for numerical data which is confidential.

PROBLEMS, PLANS AND ACTIVITIES CONCERNING THE ECONOMIC DATABASES AT STATISTICS SWEDEN

Lars Nordbäck

Head of generalized software development and marketing
at Statistics Sweden

Abstract

In this paper some of the issues concerning the move towards the terminalized dissemination of statistics from a central statistical office will be highlighted. The implications of the new techniques on the in-house activities. The way to assimilate this new technique into the different statistics consumers' hardware and software environment. The problems to suite different levels of knowledge of data processing in the man - machine interface.

INTRODUCTION

Statistics Sweden has statistical databases managed by the in-house developed AXIS-system. Originally AXIS was developed to serve regional planning authorities with data from a regional statistical database. The experiences inspired us to use this tool for the dissemination of statistics to other users as well.

After an introductory period when we had several different organizations linked to our computer free of charge, we went smoothly into the business of commercial database services in the spring of 1982. Now we have some 70 paying customers, ranging from manufacturing companies, banks and labour market organizations to local, regional and central government authorities.

Below I'll mention some issues of interest in the management of commercial statistical databases from the point of view of a central statistical office.

MOVING TOWARDS COMPUTERIZED DISSEMINATION OF STATISTICAL DATA - AN INTERNAL MARKETING PROBLEM

Statistical database services consist of at least two principal parts, viz. the database management system and the database(s). Disregarding the DBMS for the moment, in a statistical office the data needed to solve the customers' problems are not only - or even mainly - bought from outside the office. They are to a great extent produced within the office. Of course, this is an advantage, but not an advantage gained without a lot of work.

Some factors that influence the possibility to get statistical data into a database are:

a) The organization of the office, in particular the autonomy of the different statistical "products". Very operative decisions have to be made to ensure a rapid development of the databases.

b) How well the DBMS fits into the traditional production process. Does it merely mean extra efforts to satisfy new requirements, or does the software yield some advantages for the traditional production of statistical publications as well? Within Statistics Sweden we are studying the possibility to develop general software to produce publications from data stored in the statistical databases, i.e. not only to supply the figures but also the metadata.

c) Information to and training of the people responsible for the different products to make them aware of the advantages of the new way of disseminating statistics.

d) Training of the database management staff in how to structure and describe the data.

e) Existence of standard definitions of variables. Harmonization of definitions which for a long time has been a problem to discuss is in this environment a problem to solve.

All the above issues need to be penetrated. The amount of work required depend on the internal marketing of the statistical databases.

STATISTICAL DATABASES VS THE USER'S HARDWARE ENVIRONMENT

This issue should not be neglected. In the work of establishing the statistical database services, we have confronted many different users, all with their own specific EDP-background. There are differences both in knowledge and in the equipment they use or intend to use to link up with our databases. Some examples of user hardware are 3270 compatible terminals closely connected to a service bureau, word-processing machines, micro computers and, which suits our computing centre best, normal TTY-compatible terminals. Our service now is 8 TTY-lines 300 baud, 6 TTY lines 1200 baud, one 2741 line 134 baud, and one manual 3270 line 2400 baud. When customers use equipment with facilities for storing data retrieved from our computer, the uncertainty in the normal TTY protocol with its lack of a data transmission check becomes clearly inconvenient. We have to find a safer way of data transmission for this kind of customers.

Those of our customers who are closely tied to a service bureau could, if they are 'cluster customers' to the same service bureau, get linked up to our computer through the service bureau. We have not yet any arrangements of this kind, but users connected to the same bureau exist among organisations on both local and regional level

and among governmental authorities.

Another connection to our databases tried on a test level, is through the Swedish teledata net called the Datavision. This is a way to get in contact with our customers in a gateway with very small equipment requirements. I do no think the gateway effect should be underestimated.

STATISTICAL DATABASES VS THE USER'S SOFTWARE ENVIRONMENT

The problems in this area are of at least the same magnitude as in the hardware side. Of course, the different users use different software for their various projection and econometric work. The reasons are different hardware and different ideas of how to handle the data from the point of both the econometric work and the computing technique. On one hand there is no conformity in different users' software, on the other hand there is no conformity in different statistical databases as regards user interface and technical format of data (both figures and meta data). The number of problems equals the product of the number of users and the number of statistical database vendors.

I think this meeting has the responsibility to initiate efforts to minimize the negative effects that the users of several databases experience.

Some of the users do not have any former experience of EDP in their econometric work, having previously tried a manual approach. This category of clients are interested in software development work at our office and want to use our computer as a service bureau. For that reason, as well as for internal use, we intend to develop more functions within our DBMS, as well as interfaces to some external software packages, e.g. SAS/ETS, X-11-Arima and APL.

THE MAN - MACHINE INTERFACE AND THE CURVE OF LEARNING

We have learnt a lot from our experiences with the statistical databases handled by the AXIS system. One thing is that AXIS is very easy to learn for the unexperienced user, but somewhat heavy to use for the familiar user. Consequently, we are now developing an alternate way to access the data, viz. by means of a command language. This command language will be integrated with the system for the additional functions mentioned above. The best composition of a generally used system like AXIS would probably be a menu-driven system where every use of the menu should be some kind of training an underlying command language. When the user has achieved a certain level of proficiency he will be able to abandon the menues and continue in the command language. Then answers in terms of the number in front of an alternative in the menu should be forbidden. When using the touch method it is less convenient to enter a figure from the key-board than a short word.

STATISTICAL DATABASE SERVICES AND THE STATISTICAL OFFICES

Most of the statistical database services are supplied by commercial service bureaus. The data in these databases are bought from international organizations and in some cases directly from the federal or national statistical offices. Some of the statistical offices have decided to use commercial bureaus for the statistical database services. This implies that the user will have access to some data and perhaps some publications from the institutions responsible for the data. When access is made to a certain set of data, it is up to the user to try to find some information on these very data to get an idea of the quality of the retrieved figures. In the AXIS system at Statistics Sweden much work is done in the metadata area. Just to mention one function, a compulsory comment is displayed just before the result of a retrieval is displayed. This comment can sometimes include references to the responsible department including telephone numbers.

What I want to stress is the fact that statistical databases in commercial bureaus in general are not supported in the sense of knowledge of the data. Since it is easy for a statistical office to make their statistical data accessible by a commercial bureau, it is also easy to run the risk of misuse of data.

PROPOSAL FOR WORKSHOP ON LARGE ECONOMIC DATA BASES
By Phyllis Levioff

Outline for discussion purposes concerning problems confronting data base managers in their role as conduits between data sources and end users

I. PROBLEMS ASSOCIATED WITH CHANGES IN CONCEPTUAL TREATMENT OF DATA

- A. How to integrate conceptual changes into a macroeconomic data base
- B. How to communicate these changes to users
  1. in the short-term, i.e., online
  2. in the long-term, i.e., in hard copy documentation

II. PROBLEMS ASSOCIATED WITH CHANGES IN DATA COLLECTION

- A. How to deal with a definitive break in a time series
- B. How to deal with discontinued series
  1. when there are substitutions
  2. when there are no substitutions
- C. How to deal with series in terms of documentation which are no longer published but are still available in unpublished form or on tape

III. PROBLEMS ASSOCIATED WITH TIME AS A CONCEPT

- A. How to present data series that are really not a true time series because of the inconsistency of the data over time, but that are presented as time series online
- B. How to communicate to the end user the nature of the above problem without destroying credibility in the data

IV. PROBLEMS ASSOCIATED WITH DEFINITIONS OF DATA TYPE

- A. How to field technical questions from end users
- B. How to utilize source materials most effectively
  1. What sources are available, e.g.,
     - Dictionary of Economic and Statistical Terms, Bureau of Economic Analysis (out of print)
     - BLS Handbook of Methods, Bureau of Labor Statistics
     - Handbook of Cyclical Indicators, Supplement to Business Conditions Digest, Bureau of Economic Analysis
     - Statfacts, New York Federal Reserve
     - Other
  2. What reference materials should be included in a comprehensive bibliography for data base managers

SAS APPLIED TO STATISTICAL DATABANKS VIA A COMMAND LANGUAGE.

Inger Nilsson, System Engineer

I/S Datacentralen af 1959, Copenhagen, Denmark

Abstract

This paper describes how a command language, DC-TIME Series Management System, is used as an interface between the user, a databank and SAS, Statistical Analysis System. The aim is to give easy access to time series in a large databank to a great variety of users ranging from those just in need for the data to the ones requiring an analysis tool like SAS.
The paper gives an account of the structure of DC-TIME, which is actually a combination of a command language and a dialogue. To illustrate the application and the usefulness of the system some examples are given using the databank CRONOS-Eurostat, which contains macro-economic time series concerning national accounts, production, trade etc. for a large number of countries. Various ways of presenting the data, using the facilities in SAS, are also given.

1. INTRODUCTION

Statistical data are conveniently stored in large databanks in the form of time series. To make efficient use of such data there is a need for a simple method which allows various types of users - specifically those without previous programming experience - to carry out advanced statistical analysis on data and having the results presented in an easy and clear way, for example as tables, graphs or histograms.

The command language, DC-TIME, links the databank, CRONOS-Eurostat, to all the facilities in SAS. It has been developed so as to enable the user, who is interested in the data but not in the programming problems, to make use of all the possibilities in SAS.

DC-TIME is used to select time series from the databank and build up a data set, so the various SAS procedures can be applied by simply writing the proper SAS statements.

2. THE CONTENTS OF THE DATABANK, CRONOS-EUROSTAT

CRONOS-Eurostat is a macro-economic databank, containing more than 700,000 time series. The databank is subdivided by subject into six main topics, as briefly described below.

General Statistics.

This main topic covers statistical data regarding short-term economic figures for the European Community within the following subject areas: population and employment, industry, agriculture, prices, services, transport, finance and national accounts. Also available are macro-economic indicators of the developing countries, concerning demography, social and economic indicators, transport and services, industrial and agricultural production, external trade, national accounts and balance of payment.

National Accounts, Finance and Balance of Payments.

Within this main topic the European System of integrated economic Accounts (ESA) is used. ESA consists of a coherent and detailed set of accounts and input-output tables, which are intended to provide a systematic, comparable and - as far as possible - complete picture of the economic activity within each of the member countries.

Information on transactions in goods and services directly related to the formation of the Gross Domestic Product are available and further the balance of payments of the European countries, USA and Japan.

Industry and Services.

This heading covers: annual surveys of a number of economic variables - f.ex. turnover, production, value, value added, number of employees - for companies with more than 20 employees. Also included are figures for production, import and export of textiles, footwear, paper, computers and electric appliances to name some examples. Data on energy production, export and import are available as well as information on production, trade and employment within the iron- and steel industry.

Agriculture, Forestry and Fisheries.

Here, agricultural prices and price indices are available on crop products, animal products and the means of agricultural production. Also provided are fisheries statistics on annual catches by fishing region for 300 species, monthly data on landings, and annual data on foreign trade.

Foreign Trade.

For foreign trade more than 300.000 time series are available. It provides information on import and export, in values and quantities, either by product (300 SITC headings) or by trading partner (200 countries).

Mischellaneous.

This group provides information on government expenditure on research and development concerning: earth and atmosphere, human health, energy, agriculture, industrial technology etc.

3. EUROSTAT

CRONOS-Eurostat is produced by the Statistical Office of the European Communities, EUROSTAT. The aim of the office is to measure and analyse the inter-European economic and social activities, and the Common Market's relations with the rest of the World.

Data are collected from - amongst other sources - the national statistical offices of the member states of EC and stored as monthly, quarterly and yearly time series.

Geographically is covered EC, other industrial countries such as USA and Japan plus about 160 developing countries.

4. CONSIDERATIONS FOR A COMMAND LANGUAGE

In November 1981 Datacentralen implemented CRONOS-Eurostat, with the aim of offering the content to any user, interested in such macro-economic data. It was foreseen that many of the potential users, f.ex. on a management level, might be without previous experience in the use of computer systems and therefore reluctant to accept this type of service. On the other hand, some would be familiar with programming and therefore inclined to carry out further processing of the data - like regression analysis, forcasting etc. The system to be developed should therefore be flexible enough to satisfy a wide range of users. As regards the statistical analysis it was obvious to make use of SAS, which, in addition to a number of statistical procedures, offers a number of data presentation facilities.

Consequently a suitable solution for satisfying all types of users was to develop a command language, which acts as an interface to the databank and to SAS. This enables the user to apply all the facilities in SAS on the data, without having to engage in a study of the SAS language. This approach would further have the advantage of enabling the experienced SAS user to apply all the SAS procedures on the data. The command language developed, named DC-TIME, is described below.

5. FACILITIES IN DC-TIME

The command language DC-TIME is a time series management system, offering the following facilities:

- display of the time series, as tables, graphs and histograms
- creation of derived time series
- storage and maintenance of own time series
- statistical processing of time series

6. THE USE OF SAS

The diagram, figure 1, shows how DC-TIME acts as an interface between the user, the databank and SAS. In a dialogue with DC-TIME the user specifies a command and the time series to be analysed, as described in detail in the following section. The command and the time series numbers are transferred to the SAS program via SAS macro's which are built up by DC-TIME and linked to the SAS program.

The time series specified are selected from the databank by DC-TIME, stored in a file and transferred to a SAS data set by the DATA step in the SAS program. The SAS procedures, corresponding to the command, are now executed without any further action from the user. The output is presented directly on the terminal or, should the user want so, printed off line at Datacentralen.

7. COMMAND AND DIALOGUE STRUCTURE IN DC-TIME

The principle of the dialogue with DC-TIME is shown in figure 2. The user types in one of the available commands - f.ex. TABle, GRAph or HIStogram - and the codes for the time series to be analysed. The system replies by writing the name of the command and a list of parameters attached to this command. The parameters have initially been assigned some default values, for which reason

the user only has to specify desired changes. Also, the code numbers of selected time series are listed, allowing the user to add or delete time series before execution. When the command, the parameters and the time series list are acceptable, the command is executed by entering (CR).

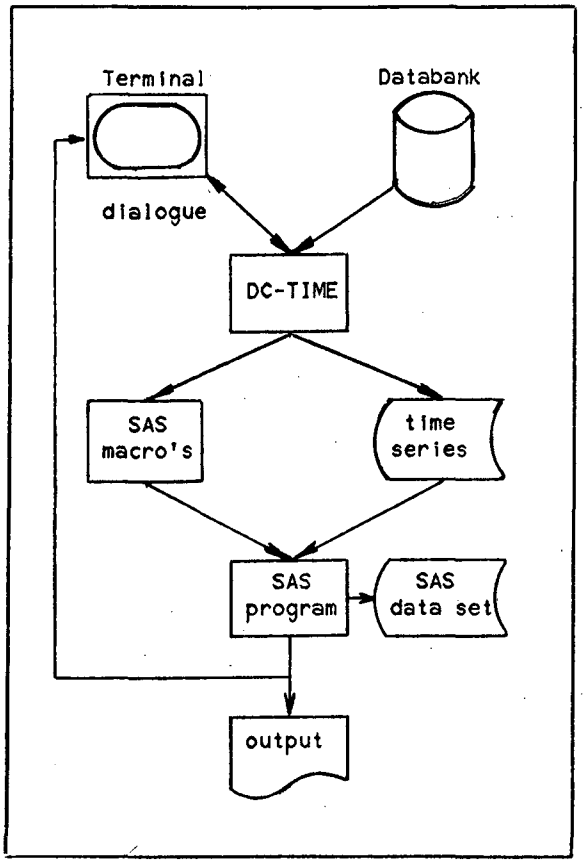


Fig. 1

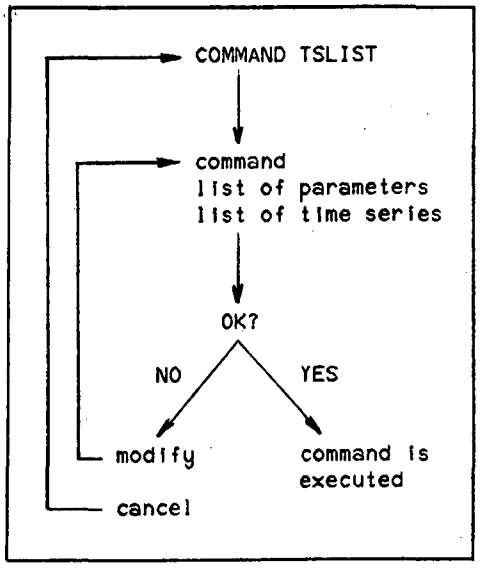


Fig. 2

8. DISPLAY OF TIME SERIES

The following examples show different ways of displaying the registration of new passenger cars in Germany, France and U.K.

The table, figure 3, can be obtained by typing:

```
TAB TS=ICG:124251006,14,26
```

to which the system replies:

```
TABLE:
DECIMAL=      0
INTERVAL=     6801,8101
LINESIZE=     72
PAGESIZE=     24
TITLE=        ENGLISH
OUT=          LIST

TIME SERIES:
ICG   124251006 144251006 264251006
MODIFY/EXECUTE/CANCEL ?
```

```
TS001=ICG :124251006,REGISTRATION OF NEW PASSENGER CARS                    1
TS002=ICG :144251006,REGISTRATION OF NEW PASSENGER CARS
TS003=ICG :264251006,REGISTRATION OF NEW PASSENGER CARS

                    DATE_     TS001     TS002     TS003

                    6801      1425      1240
                    6901      1841      1365
                    7001      2107      1342
                    7101      2152      1469      1335
                    7201      2143      1638      1702
                    7301      2027      1696      1688
                    7401      1693      1525      1274
                    7501      2106      1482      1198
                    7601      2312      1858      1288
                    7701      2561      1907      1335
                    7801      2664      1927      1618
                    7901      2623      1976      1732
                    8001      2426      1873      1536
                    8101      2330      1879      1514
```

Fig. 3

A graph showing the same time series can now be obtained by just typing:

GRA GO

where GO is used to avoid repeating all the parameters and the time series list. (Figure 4).

The histogram in figure 5 can be obtained by using the command HIS.

As an illustration of the links between DC-TIME and SAS, figure 6 shows the principles underlying the generation of the table.

The user only has to be concerned with the DC-TIME part, while the SAS steps are automatically activated by DC-TIME.

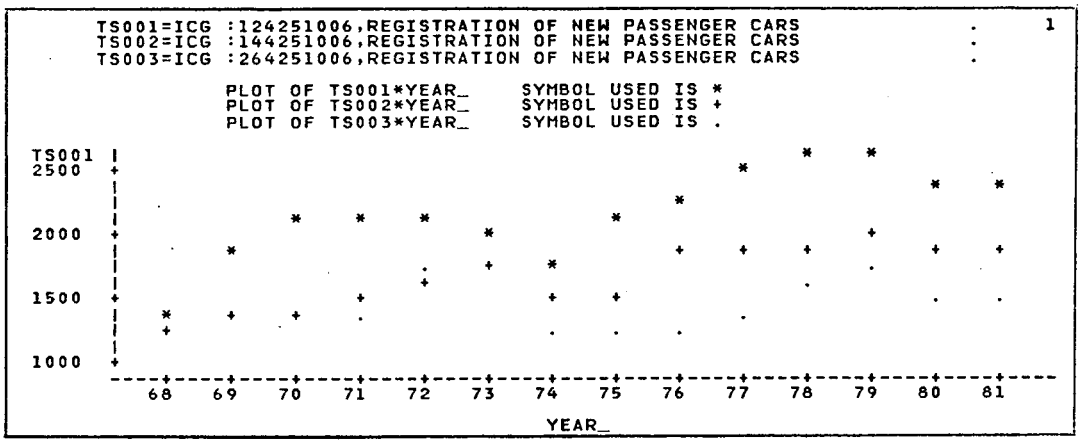


Fig. 4

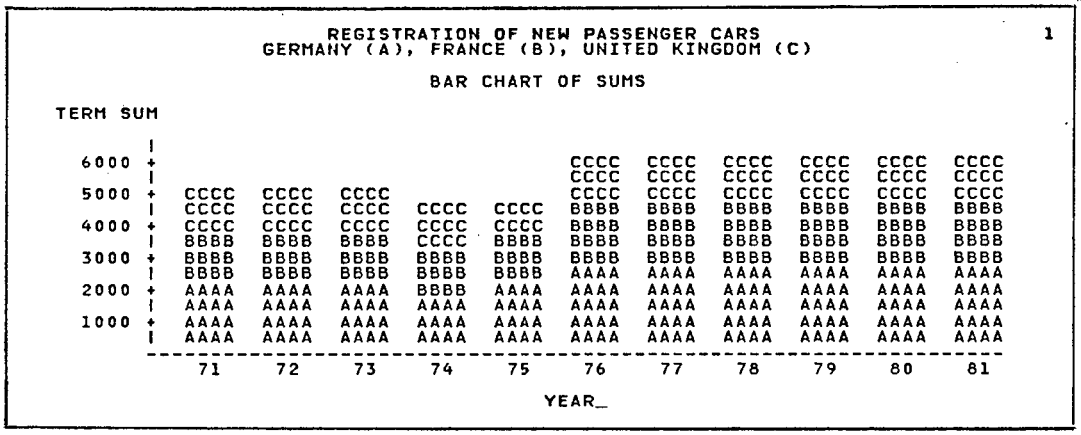


Fig. 5

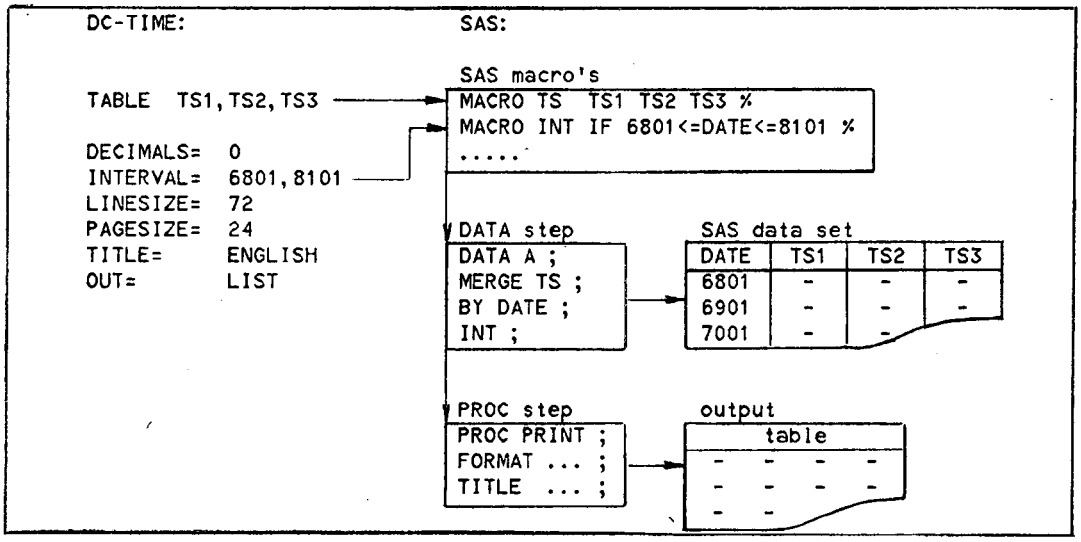


Fig. 6

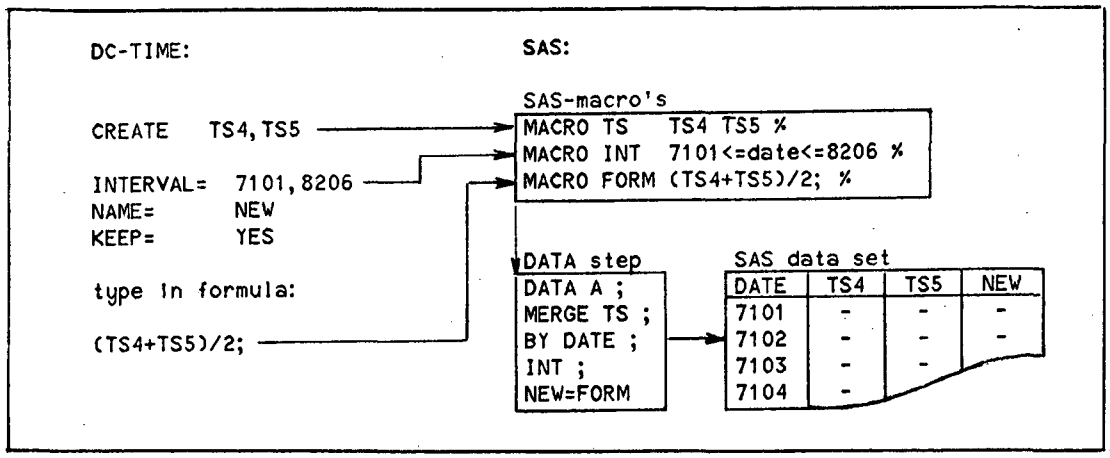


Fig. 7

9. CREATION OF DERIVED TIME SERIES

Some users might wish to create derived time series on the basis of existing data. The command CREate is used for this purpose. For creating a derived time series, the user has to type CRE, the codes of the time series from which the new series is to be calculated and the formula defining the derived time series. The new time series may now be used together with any other time series in the databank. The formula and the derived data may even be kept for later usage.

The principles in this command is illustrated in figure 7.

10. STATISTICAL ANALYSIS

The powerful facility of DC-TIME is that it enables the user to apply SAS for carrying out advanced statistical analysis on the data in the databank.

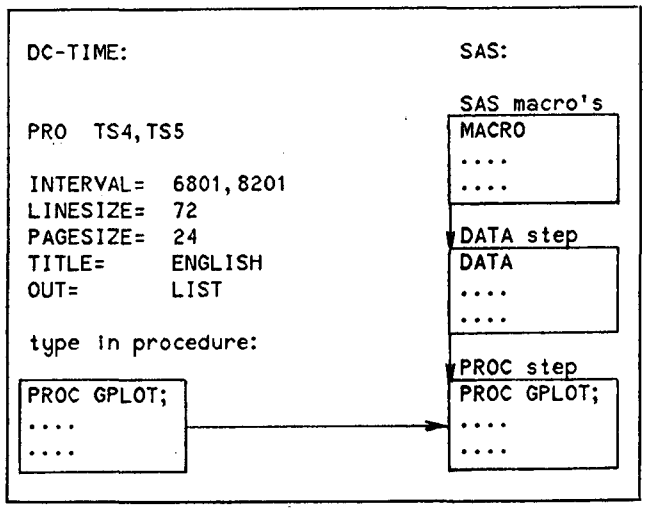


Fig. 8

The most obvious procedures to be applied to these data are regression analysis, forecast and seasonal adjustment, but any SAS procedure may be used.

The DC-TIME command for this facility is PROcedure, the principles of which are illustrated in figure 8. The only action of the user is to type PRO and the codes of the time series to be analysed. The system replies: 'Type in procedure', to which the user has to respond with the actual SAS procedures.

11. GRAPHICS

SAS contains a graphical part named SAS/GRAPH enabling a user with a graphical terminal to obtain the output in a graphical form by just typing the proper SAS statements.

12. CONCLUSION

The system has now been operational since April 1982 and used by various types of users. The original aim of developing a flexible system has been fulfilled by the application of a command language acting as an interface to SAS. Users without preceding experience in programming can take advantage of many of the facilities in SAS, and the experienced user can make full benefit of the advanced statistical procedures available in SAS.

A STATISTICAL DATA MANIPULATION LANGUAGE

G.Barsottini, Systems and Management, Torino
presently working in Luxemburg under contract with EEC

JC.Farget, EEC, Informatics Engineering, Brussels

Abstract

A new Data Model based on a set theoretic approach is proposed here; it is intended to fit the statisticians requirements by giving them the ability to describe data in terms of classes of objects which are ordered according to users needs. Classes are identified by the handle which is a set of attributes; they are gathered in collections.

Based on that Model a Statistical Data Manipulation Language (SDML) is presented to provide tools for Statistical Manipulations on Collections. Basic classes of operations are Retrieve, partition, internal or inter-object arithmetical operations and join of collections. The SDML has been designed to be concise, user-friendly and extensible.

1. INTRODUCTION

In many cases, the work of the data processing statistician consists in extracting data either from raw files or more or less classical data bases and entering them into a processing system which can be either APL, a statistical package or ad hoc programs; the results of the processing may in turn be reentered in the storage pool.

This implies, as stated in (R5), that the greatest part of the work is dedicated to the data formatting and transfers instead of insisting on data processing itself.

We therefore propose a new Data Base Management System where Data Storage and Data Manipulations as needed by statisticians are integrated into one unique system. This system called CROSIBASE is based on two main components, as far as the user interface is concerned: the first one is the catalogue which holds all data and processing descriptions; this catalogue is described in another paper (R6). The second one is the Statistical Data Manipulation Language (SDML) that we present here. We shall insist in this paper on the problems of derivation. In (R1) Adiba gives a good framework for the introduction of derived data; however the operations that he proposes are still too closely related to the Relational Model and are not powerful enough to take statistical operations into account. In statistics derived data are not only selected or projected or joined they are also aggregated by some statistical functions, or computed in different ways. We assume that it is important to the statistician to have these tools at the same level of availability as other data management tools.

We do not discuss here the problem of updating derived data, as it has been done in (R2) and (R7); but it is obvious that the problem is even more complex in the case of statistical derivation.In CROSIBASE we handle that problem by giving strong limits to the automatic updating process and by storing in the catalogue all information concerning both data and derivation descriptions.

Section 2 gives the Data Model we use in our system and a conceptual specification of operations.

Section 3 presents a detailed description of the proposed SDML.

2.DATA MODEL

Before describing any Data Manipulation Language it is fundamental to give the description of the basic features of the data model.

Recent works on the subject deal with the analysis of data Models by considering three fundamental and disjoint components :

1) Model data Structure
2) Type of executable operations on such Data (Data Manipulations)
3) Constraints on Data Structure and type of operations

On the basis of these criteria of analysis we describe the CROSIBASE Data Model presenting the Data structure, then the constraints imposed on them, then the categories of operations and finally the constraints which link operations and Data structures.

2.1 Data Structure

Definition: The CROSIBASE Data Structure can be described by a six-tuple

(A , DOM , LINK , C , Coll , I)

where

A : is a non empty set of names called attributes

DOM : is a function $A \longrightarrow T$ which associates for each attribute Ai a set of elementary values Ti; Ti is said to be the type of Ai; different attributes may have the same type.

LINK : is a function which associates a particular attribute and a particular value giving couples called atoms :

(Ai , Vj) $\quad Ai \in A,\ Vj \in Ti$

C : is the name of a collection

Coll : is a function which associates to a collection name C a non empty ordered set of attributes $Y = \{A_1, \ldots An\}$, $Ai \in A$, in which each attribute occurs only once.

According to that definition, we can associate to C the following collection structure :

$$C = \{A_1 : T_1 , A_2 : T_2 , \ldots An : Tn\}$$

I : is a set of points; each point is a set of atoms, each set having the same structure defined by Coll.

An instance I of the collection C as previously defined can be the following set of points

$$\{(A_1 , a_{1,A1}) \ (A_2 , a_{1,A2}) \ldots \ (A_n , a_{1,An}),$$
$$(A_1 , a_{2,A1}) \ (A_2 , a_{2,A2}) \ldots \ (A_n , a_{2,An}),$$
$$\vdots$$
$$(A_1 , a_{p,A1}) \ (A \ , a_{p,A2}) \ldots \ (A_n , a_{p,An})\}$$

2.2 Constraints on Data Structure

Following our schema of structure description, we give now the constraints on the previously described structures, i.e., identify the possible types of points and collections, through the semantics which can be attached to attributes or collections by the users. This is intended to provide a unified view of the concepts of sets of points and sets of classes of points.

Definitions :

point :
set of atoms i.e couples (attribute, value) in which there exists only one occurence of each attribute.

elementary object :
a point divided into two classes of atoms: one is called the handle and is supposed to contain all identifying atoms of the point; the other is called the content and is supposed to hold the quantifying or qualifying atoms of the point.

simple object :
set of elementary objects with the same handle value; a simple object can be viewed as a class of elementary objects, all having the same value for the handle attributes.

simple collection :
set of simple objects having the same set of attributes, handle attributes and content attributes.

elementary collection :
set of elementary objects, having the same set of attributes, handle attributes and content attributes. In this case the handle is a key for the collection.

One important feature in CROSIBASE is the concept of order because statisticians usually produce ordered data sets : the order is defined:

a) on the set of values defined for each type
b) on the handle attributes within an object
c) on the objects using the order defined by the handle

Type of collections :
What we will usually call collection in CROSIBASE is in fact an ordered elementarycollection or 'OE-collection' (figure a). To simplify the following of the paper we will continue to write 'collection' instead of 'ordered elementary collection'.

But we will also have to use ordered simple collections which we will call 'OS-collections' (figure b), unordered simple collections that we will call 'US-collections' (figure c), and unordered elementary collection that we will call 'UE-collections' (figure d).

It is important to note that an ordered elementary collection can be also viewed as a n-dimensional matrix, n being the number of handle attributes, each cell holding a record composed of the contents values.

This view will help the user understanding vertical operations (see 3.3.3) and also in the table lay-out process (see 3.5).

Notations : For each collection C we have the handle set of attributes or handle H and the Content Set of attributes or Content Q; we call form of the collection C the couple (H,Q) and we note it

$$\cup C = H \mid Q$$

In terms of attributes the form of a collection can be described as

$$A_1 . A_2 \ldots . A_h \mid A_{h+1}, A_{h+2}, \ldots . A_{h+q}$$

If H is composed of 2 subsets, namely H1 for the first attributes (from left to right) and H2 for the last ones, the form of C can be written as

$$\cup C = H1 . H2 \mid Q$$

In the same way it can be written

$$\cup C = H \mid Q1 , Q2 \qquad \text{if } Q = Q1 , Q2$$

The difference of notation between handle and content comes from the fact that the sequence of handle attributes is used to identify objects in an n-dimensional space, which is not the case for Content attributes.

The form is defined in the same way for OS-collections and US-collections and UE-collections.

Example

Let us consider the following handle attributes

A : T1 (a_1 , a_2 , a_3)
B : T2 (b_1 , b_2)
C : T3 (c_1 , c_2)

and the following content attributes

D : T4 (d_1 , d_2 , d_3 , d_4)
E : Integer

Let us consider one of the possible forms induced by these attributes :

A . B . C | D , E

We may have the following types of collections :

figure - a: ordered elementary collection

A	B	C	D	E
a_1	b_1	c_2	d_1	e_1
a_1	b_2	c_2	d_1	e_2
a_1	b_2	c_3	d_2	e_3
a_2	b_1	c_1	d_4	e_4

Handle values are different for each point; the order of the points is defined by the order of values in the types and the order of attributes in the handle

figure - b: ordered simple collection

A	B	C	D	E
a_1	b_1	c_2	d_1	e_1
a_1	b_1	c_2	d_1	e_2
a_1	b_1	c_3	d_2	e_3
a_2	b_1	c_1	d_4	e_4

The first two objects have the same handle values; the objects are ordered

figure - c: unordered simple collection

A	B	C	D	E
a_1	b_1	c_2	d_1	e_1
a_2	b_1	c_1	d_4	e_2
a_1	b_1	c_3	d_2	e_3
a_1	b_1	c_2	d_1	e_4

The first and the fourth objects have the same handle value

figure - d: unordered elementary collection

A	B	C	D	E
a_1	b_1	c_2	d_1	e_3
a_2	b_1	c_1	d_4	e_1
a_1	b_2	c_3	d_2	e_2
a_1	b_2	c_2	d_1	e_3

No two objects with the same handle value

2.3 categories of basic operations

We assume there exists five fundamental categories of basic operations needed by statisticians to operate on these data:

1) retrieval operations :

they are monadic operations for retrieval of information contained in the collections through the use of the 2 modes known in the Relational Model : Restriction and Projection. Restriction is the retrieval of the points of a collection which satisfy a given condition.Projection is the retrieval of sub-points of a collection (i.e. subsets of the points determined by a subset of attributes).

2) classification operations :

these monadic operations are used for the constitution of classes on the collections, based on the use of generalized equivalence relations. There are 2 types of classifications: partitions when disjoint classes are needed and distributions for non disjoint ones. The result is always a simple collection composed of the defined classes.

3) arithmetical computation operations :

these monadic operations are of two possible types :

3.1 computation on each object of the input collection giving a new value of an existing content attribute or creating a new content atom; this is called horizontal computation; it never changes the type of a collection but eventually extends its form by creating new content attributes;

3.2 computation implying several objects of the input collection : the computation is performed on one or several content attributes giving either a new content value of an existing object or a new object created by the operation; this is called vertical computation; these operations are called vertical operations because, in the case of creation of new objects, they extend vertically the tabular representation of collections. The process of aggregation is one particular type of vertical computation. Vertical computation implies no change neither on the type nor on the form of the operand collection.

4) collection join operations :

these operations are dyadic and allow 3 types of join between collections according to relational properties defined on subsets of type compatible attributes. These operations can be viewed as an extension of the relational join. These operations are authorized only on ordered elementary collections and the result is also an ordered elementary collection.

5) sort operations :

monadic operation to redefine the order of a collection or to put into order an unordered one.

3.Statistical Data Manipulation Language (SDML)

3.1 Objectives and Concepts for SDML

SDML is a set of highlevel commands having collections as operands.

They map the functionalities previously described into a user's language.We tried to describe a Relational-like language because the relational approach is widely known and understood, and also because collections are in someway equivalent to relations in first normal form.

In order to be more concise, we have grouped the executable operations into 3 main categories:

a) Retrieval operations : operator RETRIEVE, monadic
b) Computation operations : operators H-COMPUTE,V-COMPUTE,AGGREGATE, monadic
c) Connection operations : operators LOWER-JOIN, UPPER-JOIN, REDUCED-JOIN dyadic

As said earlier in the paper the main objectives of SDML will be to provide the user with a unified view of sets of objects and sets of classes of objects. Any collection in the system will be able to be considered as a set of objects (ordered or not) or as a set of classes of objects, the classes being determined by the user, according to properties of the objects.

To achieve these objectives the language has the following structure:
each operation is described by a command; each command is functionally divided into 3 parts:

1) data input definition
2) command body or processing body: operator, parameters...
3) output data format definition

The general form is the following :

1) { [FROM < collection-name >]$_1^2$ [WHERE < condition >]
   { [pre-structuring clauses]

2) { < operator >
   { [< operational description >]

3) [GIVING <collection-name>][<post structuring clause>]

Comments :

FROM : gives the name(s) of the input collection(s)

WHERE : gives the restriction to be applied on that input

< pre structuring clause > : for instance partitioning of the points of the input collection into classes.

< operator > : name of the operator

< operational description > : for instance list of attributes of a join or list of elementary statements in a computation.

GIVING : name of the output collection

< post structuring clause > : instructions for output data structuring;
for example : order of the objects or deletion of some objects.

For each command we introduce, we shall give all the functionalities it allows;we can see already that the WHERE clause is present in all commands, which involves that all functions will allow restriction on input data.

We must say a few words now of the processing environment of our SDML: all named collections are referenced in the CROSIBASE catalogue.The set of SDML commands which creates collection Coll2 from collection Coll1 is called a DERIVATION and Coll2 is said to be a derived collection. The derivation description is itself stored in the catalogue. However, for more flexibility, one derivation can create several collections.
Each command is executed sequentially and can be either complete i.e. with input and output operands or uncomplete in which case the output operand of command i is the input operand of command i+1; such commands are linked by a meta-operator "/".

Example

```
DERIVATION D1

  FROM Coll1 OP1/OP2/.../OPi GIVING Coll2;
  FROM Coll2 OP_f GIVING Coll3;
  FROM Coll2, Coll3 OP_k /OP_l  ... GIVING Coll4

END
```

This derivation has Coll1 as input collection and creates Coll2, Coll3 and Coll4.

We limit the use of this facility in the sense that any referenced collection in the catalogue must be ordered and elementary; this implies that once the name of the output collection is given in the GIVING clause, the type and order of the output collection are fixed; in absence of the GIVING clause the ORDERED and UNIQUE clauses will always give the possibility to produce an ordered elementary collection.

In all the following examples we will consider coll-input as :

A . B . C . D . E . | F , G

A : T1 (a_1,a_2)
B : T2 (b_1,b_2,b_3)
C : T3 (c_1,c_2,c_3)
D : T4 (d_1,d_2,d_3)
E : T5 (e_1,e_2)
F : T6 (1 to 1000)
G : T7 (1 to 50)

A	B	C	D	E	F	G
a_1	b_1	c_1	d_1	e_1	1	1
a_1	b_1	c_1	d_2	e_1	16	4
a_1	b_2	c_2	d_2	e_1	3	7
a_1	b_2	c_2	d_3	e_2	27	8
a_1	b_3	c_1	d_2	e_1	1	9
a_1	b_3	c_2	d_1	e_2	32	8
a_2	b_2	c_3	d_1	e_1	100	2
a_2	b_2	c_3	d_3	e_1	15	6

3.2 retrieve operations

General form :

I [FROM < collection-name>] [WHERE < predicate>]
II RETRIEVE [<attribute-list>]
III [GIVING < collection-name> [ORDERED] [UNIQUE]

This statement represents the formalization of the functionality of the 2 previously introduced operations : RESTRICT and PROJECT.

Restriction is done through the optional WHERE clause, followed by a predicate which the selected input objects must satisfy.

The Projection is done through the attribute list which gives the attributes necessary for the output collection. The "UNIQUE" clause implies that resulting objects having the same handle will be reduced automatically to one object, which will produce an elementary collection.

In the case where no output collection is specified, the "ORDERED" clause will produce an intermediate sorted collection.

Example

a) The following derivation

FROM coll-input WHERE C = c_2 AND D $\in (d_1, d_3)$
RETRIEVE A , B , D , F
GIVING coll-output

will give the following result :

A	B	D	F
a_1	b_2	d_3	27
a_1	b_3	d_1	32

b) Projection giving an output collection without contents

FROM col-input WHERE E = e_1
RETRIEVE A , B
GIVING coll-output UNIQUE
result :

A	B
a_1	b_1
a_1	b_2
a_1	b_3
a_2	b_2

3.3 Compute operations

We have three types of computations:

a) horizontal computation on the basis of object by object processing
b) aggregation which computes statistical aggregates based on classes of objects
c) vertical computation which derives new objects on the basis of a class by class processing

Remark

Aggregation is a special case of verticalcomputation but it is so commonly used by statisticians that it was important to make it a district operation.

3.3.1 horizontal computaion

General form :

I [FROM < collection-name>] [WHERE < predicate >]
II H-COMPUTE < block >
III [GIVING<collection-name>][ORDERED by<attribute-list>]

The block in the processing body, introduced by the keyword "H-COMPUTE", is composed of statements; each statement is either a simple statement or a conditional statement; conditional statements have the general form

IF < predicate> THEN < statement >
ELSE < statement >

A simple statement is an assignement of the form

<attribute-name > ::= <Arithmetical expression >

The authorized operators of the arithmetical expression are +,-,*,/; operands are content attribute names. If the attribute name on the left part of the assignment is a new attribute and the GIVING clause is omitted, the type of this new attribute will result of the types of the operands.

The structure of the output collection is determined either by the GIVING clause or by the computed content attributes in the processing body where new attributes may be introduced; the handle of the result is the same as that of the input.

Examples

a) FROM coll-input WHERE D $\in (d_1, d_2)$
H-COMPUTE
begin
F1 := F * G
end
GIVING coll-output

result of the restriction clause "WHERE":

A	B	C	D	E	F	G
a_1	b_1	c_1	d_1	e_1	1	1
a_1	b_1	c_1	d_2	e_1	16	4
a_1	b_2	c_2	d_2	e_1	3	7
a_1	b_3	c_1	d_2	e_1	1	9
a_1	b_3	c_2	d_1	e_2	32	8
a_2	b_2	c_3	d_1	e_1	100	2

result of the computation :

A	B	C	D	E	F	G	F1
a_1	b_1	c_1	d_1	e_1	1	1	1
a_1	b_1	c_1	d_1	e_1	16	4	64
a_1	b_2	c_2	d_2	e_1	3	7	21
a_1	b_3	c_1	d_2	e_1	1	9	9
a_1	b_3	c_2	d_1	e_2	32	8	256
a_2	b_2	c_3	d_1	e_1	100	2	200

b)
```
FROM coll-input WHERE  D∈(d1,d2)
    H-COMPUTE
        IF  C = c2
        THEN
            F1 := F * G
        ELSE
            F1 := F * 5
    GIVING coll-output
```

result :

A	B	C	D	E	F	G	F1
a_1	b_1	c_1	d_1	e_1	1	1	5
a_1	b_1	c_1	d_2	e_1	16	4	80
a_1	b_2	c_2	d_2	e_1	3	7	21
a_1	b_3	c_1	d_2	e_1	1	9	5
a_1	b_3	c_2	d_1	e_2	32	8	256
a_2	b_2	c_3	d_1	e_1	100	2	500

3.3.2 aggregation

General form :

```
I   { [FROM   collection-name ] [WHERE <predicate>]
    { [GROUPING BY  <class-attribute-list > ]
    { [HAVING <predicate> ]
II   AGGREGATE  < aggregate block >
III [GIVING <collection-name > ]
```

In this case the input definition is completed by a "GROUPING BY" clause which helps the user to define the classes that will be reduced to a single object by the aggregation process.
The class-attribute-list is a list of items which are either attributes of the input collection, or new handle attributes derived from the ones of the input collection by the Distribution operation. This clause can be omitted if there has had a previous phase which has produced a simple collection.

In this case the handle of the input collection defines by itself the classes necessary for the aggregation. The predicate of the HAVING-clause concerns the classes defined by the GROUPING BY clause, and not the points themselves.

The processing body of this command contains the definitions of the statistical functions to be applied to the contents of the objects of each class defined by the GROUPING BY in the input collection; for instance these functions will be COUNT, SUM, MEAN,...etc...:for each class determined by the GROUPING BY, the contents will be aggregated on the basis of the specified functions.

The resulting collection is always an elementary collection ordered according to the order of attributes given either by the GIVING clause or by the GROUPING BY clause, if the GIVING clause is omitted.

Example

```
FROM coll-input WHERE  B∈(b1,b2)
    GROUPING BY  C, A,
        H DISTRIBUTE D//(d1,d2),(d2,d3),(d1,d3)//
      HAVING  F ≥1  OR  F ≤16
AGGREGATE
    G1 SUM G
GIVING coll-output
```

evaluation of the "GROUPING BY" clause :

C	A	H	F	G	B	D	E
c_1	a_1	1	1	1	b_1	d_1	e_1
c_1	a_1	1	16	4	b_1	d_2	e_1
c_1	a_1	2	16	4	b_1	d_2	e_1
c_1	a_1	3	1	1	b_1	d_1	e_1
c_2	a_1	1	3	7	b_2	d_2	e_1
c_2	a_1	1	3	7	b_2	d_2	e_1
c_2	a_1	2	27	8	b_2	d_3	e_2
c_2	a_1	3	27	8	b_2	d_3	e_2
c_3	a_2	1	100	2	b_2	d_1	e_1
c_3	a_2	2	15	6	b_2	d_3	e_1
c_3	a_2	3	100	2	b_2	d_1	e_1
c_3	a_2	3	15	6	b_2	d_3	e_1

evaluation of the HAVING clause :

C	A	H	F	G	B	D	E
c_1	a_1	1	1	1	b_1	d_1	e_1
c_1	a_1	3	1	1	b_1	d_1	e_1
c_1	a_1	1	16	4	b_1	d_2	e_1
c_1	a_1	2	16	4	b_1	d_2	e_1
c_2	a_1	1	3	7	b_2	d_2	e_1
c_2	a_1	2	3	7	b_2	d_2	e_1
c_3	a_2	2	15	6	b_2	d_3	e_1
c_3	a_2	3	15	6	b_2	d_3	e_1

final result :

C	A	H	G1
c_1	a_1	1	5
c_1	a_1	2	4
c_1	a_1	3	1
c_2	a_1	1	3
c_2	a_1	2	15
c_2	a_1	3	8
c_3	a_2	2	6
c_3	a_2	3	6

3.3.3 vertical operations

General form :

```
   { [FROM <collection-name>] [WHERE <predicate>]
I  { GROUPING BY <attribute-list>
   { HAVING <object identification list>
   {V-COMPUTE [ON <content-attribute-list>]
II {           <vertical block>
   {EXCEPTION <value-list>
III [ GIVING <collection-name> ]
```

In the case of vertical operations, the GROUPING BY clause is mandatory because vertical operations are allowed only on elementary collections; the GROUPING BY clause is based on handle attributes of the input collection; distribution operations are not allowed here for simplification purpose.

The HAVING clause is also mandatory because its aim is to identify objects within each class; this identification is done by a predicate of the simplified following form

$$Ai = vi \wedge Aj = vj \wedge \ldots$$

where Ai,Aj are handle attributes not used in the definition of the classes. The implicit constraint linked to that formulation is that the union of the set of attributes used inthe GROUPING clause and of the set of attributes used in the HAVING clause, is equal to the set of handle attributes.

This constraints garantees that vertical operations have elementary objects as operands.

Each object identified by this predicate is assigned a name which is the name of the object within each processed class. The final syntax for this part of the language is the following:

<object identification> ::=
    <simple predicate> AS <object-name>[TARGET]

So, when we enter the processing body we have identified classes and within each class objects on which the computation takes place.The computation is made on the content attributes of these objects; the contents are those specified in the ON clause or all of them if the clause is omitted.

The vertical block is a list of statements of the same kind as in the horizontal operations; however, in this case, the variables or operands are the object names instead of the attributes names for the horizontal operations.

The EXCEPTION clause gives default values for contents in the case some of the objects are missing in the class and the computation cannot be performed; default values can be replaced by a call to a predefined procedure.

Example

```
FROM coll-input
  GROUPING BY   A,B
    HAVING C = c1 AND  D = d2 AND
                       E = e1  AS  X1  TARGET
           C = c1 AND  D ∈ (d2,d3)  AND
                         E = e1  AS  X2
           C = c2 AND  D ∈ (d1,d3)  AND
                         E = e2  AS  X3
V-COMPUTE
    ON G
    X1 := X2+X3
    EXCEPTION (X2,3),(X3,6)
GIVING coll-output
```

result :

A	B	C	D	E	F	G
a_1	b_1	c_1	d_1	e_1	1	1
a_1	b_1	c_1	d_2	e_1	16	10 = (4+6)
a_1	b_2	c_1	d_2	e_1	Φ	11 = (3+8)
a_1	b_2	c_2	d_2	e_1	3	7
a_1	b_2	c_2	d_3	e_2	27	8
a_1	b_3	c_1	d_2	e_1	1	17 = (9+8)
a_1	b_3	c_2	d_1	e_2	32	8
a_2	b_2	c_1	d_2	e_1	Φ	9 = (3+6)
a_2	b_2	c_3	d_1	e_1	100	2
a_2	b_2	c_3	d_3	e_1	15	6

3.4 JOIN Operations

In this case we have two operand collections :

coll1 : H1 | Q1 and coll2 : H2 | Q2

We assume that H1 = X U H'1
H2 = Y U H'2

where we assume that X and Y are 2 sets of type compatible attributes element by element. First Coll1 is projected on X as PROJ1 and Coll2 on Y as PROJ2; Set-op is a set operator i.e. union, intersection, difference.

We call coprojection the set PR12 defined as PR12 = PROJ1 set-op PROJ2.
For each object X of PR12 we build a new object of the form

X . H'1 . H'2 | Q1 , Q2

where X.H'1|Q1 ∈ Coll1, and X.H'2|Q2 ∈ Coll2

If one of the objects is missing in one of the two collections, empty values will be provided for the relevant attributes.

General form

I { FROM <collection-name> [WHERE <predicate>]
AND <collection-name> [WHERE <predicate>]
II { <operator>
ON <attribute list> AND <attribute list>
III [GIVING <collection-name>]

The two input collections may be filtered by a WHERE clause.
There are 3 operations corresponding to the 3 types of operator used in the coprojection:

- LOWER-JOIN for the intersection of the projections
- UPPER-JOIN for the union of these projections
- REDUCED-JOIN for the difference

The two optional attribute lists in the ON clause will indicate, if necessary, the attributes which have to be matched in the join; the default option is that the join is based on attributes with the same name. These attributes define the projection for each collection.

Examples

- coll-input1 - A . B . C | F

A : T1 (a_1, a_2)
B : T2 (b_1,b_2,b_3)
C : T3 (c_1,c_2)
F : T4 ($1 \leqslant I \leqslant 1000$)

- coll-input2 - B . C . D | G

B : T2
C : T3
D : T5 (d_1,d_2)
G : T6 ($1 \leqslant I \leqslant 50$)

coll-input1:

A	B	C	F
a_1	b_1	c_1	1
a_1	b_1	c_2	3
a_1	b_2	c_2	10
a_2	b_2	c_1	20
a_2	b_2	c_2	31
a_2	b_3	c_2	4

coll-input2:

B	C	D	G
b_1	c_1	d_1	4
b_1	c_2	d_1	7
b_1	c_2	d_2	9
b_2	c_1	d_1	14
b_2	c_1	d_2	6
b_2	c_2	d_2	1

a) FROM coll-input1 AND coll-input2
LOWER-JOIN
GIVING coll-output

B	C	A	D	F	G
b_1	c_1	a_1	d_1	1	4
b_1	c_2	a_1	d_1	3	7
b_1	c_2	a_1	d_2	3	9
b_2	c_1	a_2	d_1	20	14
b_2	c_1	a_2	d_2	20	6
b_2	c_2	a_1	d_2	10	1
b_2	c_2	a_2	d_2	31	1

The join is implicitly based on attributes having the same name i.e. B and C here

b) FROM coll-input1 AND Coll-input2
LOWER-JOIN
ON B AND B
GIVING coll-output

		Coll	Col2			
B	A	C	C	D	F	G
b_1	a_1	c_1	c_1	d_1	1	4
b_1	a_1	c_1	c_2	d_1	1	7
b_1	a_1	c_1	c_2	d_2	1	9
b_1	a_1	c_2	c_1	d_1	3	4
b_1	a_1	c_2	c_2	d_1	3	7
b_1	a_1	c_2	c_2	d_2	3	9
b_2	a_1	c_2	c_1	d_1	10	14
b_2	a_1	c_2	c_1	d_2	10	6
b_2	a_1	c_2	c_2	d_2	10	1
b_2	a_2	c_1	c_1	d_1	20	14
b_2	a_2	c_1	c_1	d_2	20	6
b_2	a_2	c_1	c_2	d_2	20	1
b_2	a_2	c_2	c_1	d_1	31	14
b_2	a_2	c_2	c_1	d_2	31	6
b_2	a_2	c_2	c_2	d_2	31	1

in this case the base of the join is limited to attribute B

c) FROM coll-input1 AND coll-input2
UPPER-JOIN
GIVING coll-output

B	C	A	D	F	G
b_1	c_1	a_1	d_1	1	4
b_1	c_2	a_1	d_1	3	7
b_1	c_2	a_1	d_2	3	9
b_2	c_1	a_2	d_1	20	14
b_2	c_1	a_2	d_2	20	6
b_2	c_2	a_1	d_2	10	1
b_2	c_2	a_2	d_2	31	1
b_3	c_2	a_2	Φ	4	Φ

d) FROM coll-input1 AND coll-input2
REDUCED-JOIN
GIVING coll-output

A	B	C	D	F	G
a_2	b_3	c_2	Φ	4	Φ

3.5 Data Presentation

It is important to say a few words on the presentation facilities available in CROSIBASE : there are 2 possibilities: DISPLAY and TABLE.

Any collection referenced in the catalogue may be displayed at screen (eventually copied on paper); in the case data are stored in a public domain, the wordings associated to data description items (collection, attributes, values) will be automatically displayed. It is possible also to store in the catalogue and link to the collection a Table lay-out which will be used for any display required on that particular collection.

Moreover as any ordered elementary collection can be viewed as an n-dimensional table, n being the number of handle attributes, the user may use the TABLE command to give a specific table lay-out; there exists also the possibility of merging several collections to produce a table with complex structure, as was the case with the previous OSIRIS (*) table generator.

Note however that the data issued from the presentation process is no longer homogenous to the concept of collection and thus cannot be re-used for subsequent processing (except through the interface with a photocomposer).

4. CONCLUSION

We have introduced in this paper a new Data Manipulation Language to be used by statisticians.

It is based on a Data Model where objects and classes of objects have the same representation; homogenous sets of objects/classes are grouped into collections. The Statistical Data Manipulation Language (SDML) has been designed to integrate basic statistical Data Manipulation tools; the result of each operation can be used in turn by a new operation; such a set of operations is called a derivation and is stored in the catalogue as are data descriptions.

The basic functional components are restriction, projection, classification, sort, aggregation, arithmetical computation within objects or within classes, and connection. These functions are grouped together into three main classes of operations : retrieval, computation and join.

The proposed language is assumed to be user friendly and concise.

(*) OSIRIS was developped within EEC and made operational since 1976

References

(R1) - M.Adiba : "Derived relations : a unified mechanism for views, snapshots and distributed data"
Proc. of Very Large Data Bases, Cannes sept.1981 p.293

(R2) - Astrahan et al : "System R : Relational approach to Data Base Management"
ACM Transaction on Database Systems Vol 1, n°2, June 1976

(R3) - A.W.Bragg : "Data Manipulation Languages for Statistical Databases: the Statistical Analysis System (SAS)"
Proc. of the first LBL Workshop on Statistical Database Management
Menlo Park, Ca, dec 1981

(R4) - E.F.Codd : "Extending the Relational Model to capture more Meaning"
ACM Transaction on Database Vol 4, n°4 dec.1979

(R5) - A.M.Parkhurst : "A Statistician's View of the Requirements of a Host Database to support a Central Query Language"
Proc. of the first LBL Workshop on Statistical Database Management"
Menlo Park, Ca, dec.1981

(R6) - U.Rugani : "The Catalogue of the Crosibase Statistical Data Base"
abstract, april 1983

(R7) - M.R.Stonebrakes, E.Wong, P.Kreps, G.Held "The design and implementation of INGRES"
ACM Transaction on Database Systems Vol 1, n°3, sept.1976

5. Special Data Types and Operators for Statistical Data and Metadata

See Also. . . .

COMPLEX DATA TYPES AND A DATA MANIPULATION LANGUAGE FOR SCIENTIFIC AND STATISTICAL DATABASES+

V. A. Brown++, S. B. Navathe, and S. Y. W. Su
Database Systems Research and Development Center
University of Florida, Gainesville, FL 32611

Abstract

A Scientific and Statistical DBMS needs to recognize a greater variety of data types than those currently supported by conventional DBMSs. Using the concept of abstract data types, we propose a set of extended data types (Complex Data Types) to be supported directly by the DBMS. The Complex Data Types presently recognized include: set, vector, ordered set, matrix, time, time series, text, and generalized relation. A data manipulation language designed specifically for scientific and statistical data processing is presented using these types as a basis.

1. INTRODUCTION

Scientific and Statistical Databases (SSDs) define a class of databases which are intended for statistical analyses. Demographic and geographic databases are two common examples of SSDs. This class of databases has several characteristics with which conventional DBMSs and statistical packages cannot cope effectively. Facilities for managing data to generate statistics as well as for appropriate user interfaces for statistical analyses are lacking in these DBMSs:

A. Conventional DBMSs do not provide either the necessary tools for complex statistical analysis, or provide the environment for the analysis process [BOR82].

B. SSDs consist of sparse data, making the physical design and processing requirements quite different from conventional DBMSs [TUR79, BAT82].

C. The distinction between parameter and measured data becomes quite important in SSDs [JOH81, SHO82, SU82]; conventional DBMSs do not make a distinction between these types.

D. Conventional DBMSs model business or corporate databases. However, the data found in SSDs do not fit well into this "Supplier-Parts paradigm". Data exists not as integers and alphanumeric strings but in more complex forms such as matrices, time series, and sets.

E. Statistical packages have historically been flat file systems geared towards number crunching. Such systems have little data management capabilities and are not oriented towards providing processing environments.

The number of data-specific management systems[1] that have resulted from these differences demonstrates the need for new techniques. A research project was initiated at the University of Florida in 1981 to address the areas of logical and physical data modeling for SSDs. This paper reports the results of some of that research. It specifically addresses the need for incorporating an extended set of data types into a general semantic data model for SSDs. Such incorporation offers several advantages:

1. *High-level Interface.* The user can manipulate data at his[2] natural level of abstraction, rather than decomposing his view to accommodate the system.

2. *Representation Independence.* By using the abstract data typing concepts to represent these objects, the user is separated from the details of implementation.

3. *System-enforced Integrity.* The explicit modeling of CDTs transfers the responsibility of ensuring the correct implementation of abstract objects and operators from the application programmer to the DBMS.

We have developed an extended set of data types which we call Complex Data Types (CDTs) for use in a generalized DBMS for scientific and statistical database management. A CDT is a structured generic data type which corresponds to an abstract object commonly found in the user's view of data. To enable users to manipulate the CDTs, one must provide a language vehicle. A possible language interface called SSDL, which falls somewhere between a query language and a procedural language, is introduced in this paper. In addition to illustrating the usefulness of CDTs, SSDL represents the beginning of our work on a "statistician-friendly" data manipulation language.

+ This work was supported by the Department of Energy under contract DE-A505-81ER10977

++ Now at American Bell Inc., Lincroft , NJ 07738

1. Such as SEEDIS--the Social, Economic, Environmental, and Demographic Information System--which was developed at the Lawrence Berkeley Laboratory to provide SSD users with an integrated access to a specific set of databases (MCC81).

2. For brevity the masculine gender is used as a generic pronoun reference. No bias is intended.

2. BACKGROUND

In programming languages, data abstraction is done through data typing. The user views a data type simply as a set of values and the operations permissible on those values. Details of implementation--how the data is stored and how operations are performed--are considered noise and remain hidden. Abstract data typing, which allows users to define their own data types, extends this data abstraction capability even further. In languages that support abstract data typing, such as CLU [LIS74] and ADA [WEG80], users have the ability to model the language interface, customizing it to fit a specific application.

Previous work on applying abstraction techniques to database management has focused on module abstraction [ROW78, SMI78, BAR81]. In the module approach, each relationship is modeled as a unique "type" bound to its application processes. An employee, for example, is of type EMPLOYEE, and is bound to its associated operators of HIRE and FIRE. Complex data typing uses abstraction at a lower level. Using the CDT approach, an employee may be modeled as a relation with attributes which may be of type date, name, money. Our Complex Data Typing is not used to define the processes on employee, but to create an environment that makes those processes easier to specify.

The CDTs defined in this paper interface both with the user and a semantic association model for SSDs called SAM* [SU82]. The SAM* model defines a database in terms of a set of interrelated associations: membership, aggregation, generalization, interaction, composition, cross-product, and summarization. These associations are represented by one or more generalized relation (G-Relations).

A G-Relation provides the distinction between the identifying and summary attributes. Identifying attributes are attributes that qualify what the corresponding data is about. Summary attributes are attributes that constitute the measurements and needed values. For example,

State	County	population

State and County are identifiers while population is the summary attribute. A G-relation can be defined over a set of complex domains; a complex domain may be of the type G-relation itself.

3. COMPLEX DATA TYPES FOR SSDS

Operations on a data type may be separated into primitive and high-level operators. Primitive operators define the meaning of the data type and would be included in any implementation of the data language. A list of primitive operators for corresponding CDTs are given in the appendix. High-level operators are derivable from primitives. They are defined as those operations that would be convenient for the user to have, and will differ according to the user population. In this section, each of the identified CDTs is defined, and examples of high-level operators are given.

G-RELATION: A Generalized or G-Relation is a relation defined over a set of simple or complex domains. Domains are further classified as being either identifying or summary domains, as described in [SU82].

Operations on a G-Relation are performed at two levels. At the lower level, the operators correspond to the data type of the domain. At a higher level, the G-Relation is simply a set of uniform tuples. As such, all set operators previously described are applicable. Traditional relational operators, such as join, select, project, can also be used with some modification. For example, the following statement selects all experimental data on a subject in which the average water retention for the first measured parameter is greater than five percent.

```
IN pilot:
 IF ((subject=5547) and
     (water_ret(1;ave) .GT. .05)
 THEN SELECT *
```

SET: A set is a collection of elements, all of the same type, in which no duplicates are allowed.

The CDT set corresponds directly to the mathematical notion of set. Operators should include finding the union, intersection, difference between two sets. E.g.,

```
Given a: <<Chicago, Denver, Lincroft>>
      b: <<Denver, Detroit>>

      a CONTAINS 'Chicago'        returns <T>
      a INTERSECT b          returns <<DENVER>>
```

ORDERED SET: An ordered set is a set in which the elements are ordered and may be indexed.

Elements of a set are related to each other by membership only. Members of any ordered set are additionally related by order. As such precedence operators are added to both set and vector operations.

```
example:
      fc1: {Davis, Chen, Sandburg, Mondale}

      IN fc1: 'Chen' BEFORE 'Mondale'
            returns <T>
      IN fc1:  PREDECESSOR(Sandburg)
            returns  {Davis, Chen}
```

VECTOR: A vector, V, is a collection of homogeneous elements. There is an ordered set of indices, I; and a one-to-one mapping between I and V associated with a vector. Subscripting is defined as the mapping of I into V.

Operations on vector may be either position or content dependent. Content dependent operators include:

- Information retrieval operations such as checking for inclusion or exclusion of a single element, or multiple occurrences of either; E.g., given:
  a : <-1,2,-3,5,8,9>
  b : <-3,5>
  a INCLUDES b returns the index (3) denoting the position from which b is included in a. To search for the location of a value, the "?" operator is used.
  s:= INDEX(a(?) > 0) returns the indexset <(2),(4),(5),(6)>.
  To return the actual values greater than zero, this indexset may be used
  a(s) returns the vector <2,5,8,9>.

- Cleaning a data vector for further analysis through replacement or deletion of specified components. For example, to replace all values less than zero by zero:
  IN a:
  REPLACE a(?) < 0 BY 0

Position dependent operators utilize subscripting to provide flexible access of an entire vector, a subvector, or a single component.
a(1,3,4) returns <-1,2,5>

MATRIX: A matrix is a multidimensional collection of elements of the same type.

The use of a matrix in a traditional DBMS system is typically made possible via an application program in which all processing is done one cell at a time. When a matrix is recognized as a data type, operations can be performed on it taking multidimensionality into account. The system SAS, for example, now has a matrix language which provides for non-procedural aggregation and linear algebra functions.

In the following example each null (unknown) data element in a matrix is replaced by the average of existing values. First, an indexset containing the position of each missing element is defined as t; next, an indexset of all existing elements is defined as u; finally, the missing elements in z are redefined as the average of all existing values.

```
t:=INDEX(z(?;?)=NULL)
u:=INDEX(z(?;?)=EXISTS)
z(t):=AVE(z(u))
```

TIME: Time is a value representing a point of reference.

Operations on time include:

1. Accessing by interval:
   DURING Jan
   BETWEEN 10:10 AND 10:15

2. Accessing by temporal order:
   BEFORE 12/20/82

TIME SERIES: A time series is a two-dimensional matrix in which rows represent cases (measurements), and columns represent observations (identifiers), indexed by an ordered set of times.

The CDT time series is modeled as a special case of the two-dimensional matrix, so that multiseries operations can be included. A time series differs from a matrix in that the semantics of the rows and columns are explicitly defined. Consider a time series of rainfall for 1982 by county.
rainfall(Alachua) returns a single series on Alachua county;
rainfall(BETWEEN 3/15/82 AND 9/15/82) subsets the series by time.

Data in a time series is not necessarily periodic; however, for statistical analysis, periodicity is required. The BY operator is used for implicit aggregation or disaggregation.
monthly_rainfall:=rainfall BY MONTH

TEXT: A text is a vector of characters.

Text allows for a free-formatted field in the data. Traditional DBMSs require the user to implement text through application programming using characters and substrings. Realistically, this means that unformated fields are not included. Comments or descriptions are either encoded (e.g., medication:=44 means "Arithromycin, 10mg, 5 days") or are recorded off-line.

Common text processing operations to be performed include:

1. The use of variable or fixed length don't cares:
   'pollut*' matches
   pollute, pollution, pollutants

2. Threshold matches: for example, to select data in which a descriptive field contains at least 60% (threshold) of the keywords described:
   BY WORD(desc CONTAINS AT LEAST .6 OF keywords)

3.1 RELATED WORK: STATISTICAL ANALYSIS SYSTEMS

We surveyed nine statistical analysis packages: CONSISTENT (DAW80), GENISYS (DIN80), MINITAB (MIN81), P-STAT (BUH79), S (S80), SAS (SAS79), SIR (SIR80), TPL (TPL80). and TROLL (TROL79). Specifically we were interested in finding out how well our notion of complex data types is supported in statistical analysis packages. Although the packages do not necessary support the notion of type, the usefulness of defining these complex types and the facility to manipulate them directly is recognized.

ata Type	*Concept Supported By*
SET	-
VECTOR	S, MINITAB, TROLL
ORDERED SET	-
MATRIX	SAS, MINITAB, TROLL
TIME	SAS,SIR,TPL,CONSISTENT
TIME SERIES	SAS,MINITAB,TPL
TEXT	CONSISTENT
G-RELATION	-

Figure 1. CDT Concepts Support by Statistical Packages

4. AN SSD LANGUAGE

In designing an SSD Language (SSDL), we made the following design decisions:

1. The language should operate in an environment which allows a user to create and save temporary files. A SSD is generally static in nature; the average user will not be allowed to make alterations to the main database. At the same time, a user may often want to alter the data experimentally, or need to massage it into a specific form for future analysis. We assume that it is possible to define the view (subschema) of the database for each user and that users can only perform retrievals against local views. Updates are authorized to DBA only.

2. SSDL should include tools for descriptive analysis and data manipulation, but should not duplicate the efforts of complex analysis (e.g., mathematical modeling) and display packages. It should, instead, provide the user with a common interface to those packages. This decision may cause integration problems for the implementor, but will give the user the most advanced tools available without reinvention.

3. "Friendliness" must not get in the way of power. A language which does not allow the user to perform a large range of semantically valid data operations does not provide sufficient support for exploratory analysis. In SSDL we include a complete set of primitives for such exploration in addition to the commonly used high-level operators for a more casual user.

4. For statisticians, a structured, procedural language may be more natural than a non-procedural language. We first patterned SSDL after SQL [CHA76]. A reaction from statisticians was that things were done out of order--that output should be specified last, not first as in the SELECT statement; that nested commands force the user to get all the way into the middle of the procedure before he understands what is going on. If the user is going to proceduralize a language to understand it, one might just as well begin with a procedural language.

   Preliminary studies by Welty and Stemple [WEL81] suggest that performance on complex queries is better in a procedural language and no worse on simple queries.

Consider a simple aggregation using the relation emp(empno, dno, sal):

Query: List the departments in which the average salary is less than $10,000.

Figure 2 gives the SQL solution; to the right of the query is its architectural structure.

```
SELECT dno                          operation
FROM emp                            environment
GROUP BY dno                        level
 HAVING AVE(sal) < 10,000           condition
```

Figure 2. SQL Version of Query

What we suggest as a more natural ordering is based on CASDAL [SU78] and is shown in Figure 3. The user specifies first what he is working on; second, how he is going to work with it; third, any restrictions; and fourth, the operation to be performed.

```
IN emp:                             environment
 FOREACH dno                        level
  IF (AVE(sal) .LT. 10,000)         condition
  THEN OUTPUT (dno)                 operation
ENDEACH
```

Figure 3. SSDL Version of Query

4.1 THE STRUCTURE OF SSDL

A SSDL program consists of one or more program statements. A program statement is composed of the following structures.

1. *Environment.* Environment specifies the context under which operations are going to be performed. Conceptually, the specification of a G-Relation in the environment statement retrieves that table of data into the user's workspace.

2. *Level.* The level specifies the range over which the data operations are to be performed. Since aggregation over subsets of data is a prevalent operation in SSD processing, this structure must be straightforward and easy to use. If no level is specified, it is assumed to be tuple at a time.

3. *Condition*. The condition structure specifies under what conditions data retrieval and manipulations are made. The constructs that make up this structure are similar to those found in PASCAL. Condition statements, the FOR construct, and statement blocks are all supported. Additionally, a set of boolean system operators which apply to all CDTs are specified in the language: *ANY*, *ALL*, *NO*, *EXISTS*, *NULL*.

4. *Operation*. The operation structure specifies the retrievals and manipulations necessary for output or preparation for further processing. Operations available to the user include:
   - *Output a variable, expression, or special function.* SSDL borrows non-procedural output statements from existing statistical packages for aggregation (the BY operator); cross tabulation, e.g.,
     occupation BY education USING (MIN(sal), MAX(sal));
     and aggregation of quantitative variable, e.g., sal BY CUT(age,5).
   - *Create temporary variables or new views.* The user can define variables or G-Relations by example. Using the emp G-Relation, the user could write:
     ```
     IN emp:
       CREATE asal(dept AS dno, ave_sal
                   AS sal)
     ```
     Synonyms are available to simplify a given program. Thus,
     ```
     LET adp==auto.date_purchased
     ```
     allows the user to type the shorter version for the duration of the environment (similar to [DINT80]).
   - *Perform a subprogram.* A complex query often involves the manipulation of a G-relation at different levels of processing. The BEGIN-END structure is used to specify environment within an environment processing. An example of this is given in Figure 5.
   - *Perform a macro.* Users are given the power of procedural abstraction.
   - *Call a statistical package for performing complex analysis.*

5. EXAMPLES

Two examples of SSDL programs are given below to demonstrate the use of CDTs and the flavor of SSDL as a high-level procedural language. Numbers to the right of program lines correspond to comments following the program.

The first example involves the G-Relation:
energy_use(state,county||pop,oil_consumption)
where state and country are identifying attributes; pop and oil_consumption are summary attributes; state, and county are simple string variables, pop is an integer, and oil_consumption is a single time series.

```
query: List the average monthly per capita oil
       consumption for each state during last
       winter (11/82 - 3/83).
IN energy_use:
   LET oc==oil_consump

   ADD month_woc AS oc(1)                          [1]

 DO                                                [2]

   oc:=oc(BETWEEN 11/81 AND 3/82) BY MONTH         [3]

   month_woc:=AVE(oc)

 END

OUTPUT(SUM(month_woc)/SUM(pop) BY state)
```

Figure 4. Time Series Example

The bracketed notes [1], [2], [3] are explained below.

1. For the duration of the program, the G-Relation energy_use is appended with a summary field to hold average monthly oil consumption for each county. Note the use of "definition-by-example" with the AS construct.

2. The DO-END structure indicates the statements to be performed on each tuple of the G-Relation.

3. This statement combines vertical subsetting and aggregation. The time series o_c is now periodic by month and contains only data for November through March.

The second example is taken from Teitel's volume testing paper[3] [TEI81]. Consider the following G-Relation:
persons(id, birthyr, educ, sex, mo_id, fa_id)
Each tuple contains information on an individual; data on that individual's parents (e.g., birthyr, educ) may or may not be included in the data as a separate entry.

```
query: Give a frequency distribution of
       offspring by the educational level of
       each parent.

IN persons(X):                                     [1]

  CREATE parents(child AS id,
           mo_ed AS educ, f_ed AS educ)
```

3. Volume is defined as width: number of variables; length: number of cases; and depth: query complexity.

```
DO
  parents:= parents UNION
                <id,NULL,NULL>        [2]
  BEGIN:
    IN persons(Y):                    [3]
      IF (X.id =Y.mo_id)              [4]
      THEN parents.mo_ed :=Y.educ
      ELSE
       IF (X.id = y.fa_id)
       THEN parents.fa_ed := Y.educ
  END
END
IN parents:
   OUTPUT(mo_ed BY fa_ed
          USING COUNT(child_id)       [5]
```

Figure 5. Embedded program example

1. The X and Y prefixes distinguishes between two scans of the same G-relation.

2. The G-relation PARENTS is created then filled tuple at a time. NULL is a SSDL keyword.

3. For each tuple the entire G-Relation is searched again to find and record parental data. The BEGIN-END block indicate a SSDL sub-program.

4. The notion of a currency pointer is retained within a subprogram.

5. The BY...USING construct fills in cells of a table using the specified function COUNT [TPL80].

6. CONCLUSIONS AND FUTURE WORK

A user language for SSDs must optimize user friendliness while providing power and flexibility. In SSDL this is done by providing two levels of processing. For common tasks such as aggregation or frequency distributions, non-procedural operators and built-in functions are available. We have integrated the work of existing statistical systems to provide high-level procedures for descriptive analysis. To ensure that users can perform any semantically valid operation on CDTs, primitive operators are also available. User aids, such as synonyms, macros, and definition-by-example, are included to make low-level operators easier to use.

Work continues at the University of Florida in the area of statistical database languages. Shen [SHE83] uses the notion of CDTs and the basic SSDL structures in a formal design of a database language for statistical and scientific users.

SSDL is being developed as a data language for retrieval and manipulation of SSDs. A data definition language (DDL) must also be developed to be used in conjunction with SSDL. A DDL to support SSDL should include:

1. *Domain Definition*: A domain is a CDT in which the value set may be further constrained, either through enumeration or range specification. E.g.,
   inches_rain: REAL <00..60>
   counties: SET OF <(list of counties)>

2. *CDT Definition*: A parametric approach to data definition might be most useful.
   rainfall: *TSERIES OF* inches_rain
   CASE: counties
   SDATE: 1/82
   PERIOD: daily

3. *Function Specification*: To perform implicit aggregation in SSDL, aggregation procedures must be defined in the DDL. This includes standard summarization (e.g., population BY county) as well as specially defined aggregates (e.g., profit BY corporate_quarter.)

4. *Units of Measure*: A SSD must be able to handle units of measure cleanly. In our opinion data typing is not a good solution, since the problem centers around conversion between units rather than operations on them. Some work has been done in this area (e.g., KAR78, GEH82), but the problem is not yet solved. We look upon this as an area for future study.

Complex data typing has been presented as a means of supporting the manipulations necessary for SSD processing at the user's level of abstraction. We feel that incorporating CDTs directly into the DBMS will result in more efficient and effective management of SSDs.

7. REFERENCES

BAR81 Baroody, A., and DeWitt, D. An Object-Oriented Approach to Database System Implementation. *ACM Transactions on Database Systems*, Vol. 6, No. 4, 1981.

BAT82 *Index Encoding: A Compression Technique for Large* Statistical Databases, CIS Technical Report 8182-9, 1982. To appear in *Proceedings of the Second International Workshop on* Statistical Database Management, Los Altos, CA, September 1983.

BUH79 Buhler, S., and Buhler, R. *P-STAT 78 User's Manual*, P_STAT, Inc., Princeton, NJ , 1979.

CHA76 Chamberlin, D., Astrahan, M., Eswaran, K., Griffiths, P., Lorie, R., Mehl, J., Resiner, P., and Wade, B. SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control. *IBM Journal of Research and Development*, Vol. 20, No. 6, 1976.

DAW80 Dawson, R., Klensin, J., and Yntema, D. The Consistent System. *The American Statistician*, Vol. 34, No. 3, 1980.

DIN80 Dintelman, S., Mannes, A., Skolnick, M., and Dean, L. GENISYS: A Genealogical Information System. *Genealogical Demography*, Academic Press, New York, NY, 1980.

JOH81 Johnson, R. Modelling Summary Data. *Proceeding of ACM/SIGMOD International Conference on the* Management of Data, Ann Arbor, MI, April 1981.

LIS74 Liskov, B., and Zilles, S. Programming with Abstract Data Types. *SIGPLAN Notices*, Vol. 9, No. 4, 1974.

MCC81 McCarthy, J., Merrille, D., Marcus, A., Benson, W., Gey, F., Holmes, H., and Quong, C. *The SEEDIS Project: A Summary Overview*, Technical Paper PUB-424, September 1981.

MIN81 *MINITAB Reference Manual*, Version 81.1, Duxbury Press, Boston, MA 1981.

ROW78 Rowe, L. Data Abstraction from a Programming Language Point of View. *Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Models*, Pingree Park, CO, June 1980.

S80 *S User's Guide*, Bell Labs, Murray Hill, NJ, 1980.

SAS79 *SAS User's Guide*, 1979 Edition, SAS Institute, Inc., Cary, NC, 1979.

SHE83 Shen, K., Navathe, S. B., and Su, S. Y. W. *Toward a Programming Language for Statistical and Scientific Databases*, working paper, CIS Technical Report, Database Systems Research & Development Center, University of Florida, 1983.

SHO82 Shoshani, A. *A Research and Development Plan in Data Management and Distributed Systems*, Technical Report, Lawrence Berkeley Laboratory, Berkeley, CA, 1978.

SIR80 *SIR User's Manual*, Version 2, SIR, Inc., Evanston, IL, 1980.

SMI78 Smith, J., and Smith, D. Conceptual Database Design. *Proceedings of the NYU Symposium on Database Design*, New York, NY, May 1978.

SU78 Su, S., and Eman, A. CASDAL: CASSM's DAta Language. *ACM Transactions on Database Systems*, Vol. 3, No. 1, 1978.

SU82 Su, S. *SAM**: *A Semantic Association Model for Corporate and Scientific/Statistical Databases*, CIS Technical Report 8182-6, Database Systems Research & Development Center, University of Florida, 1982.

TEI81 Teitel, R. Volume Testing of Statistical/Database Software. *Computer Science & Statistics: Proceedings of the Thirteenth Symposium on the Interface*, Pittsburg, PA, March, 1981.

TPL80 *Table Producing Language System, Language Guide*, Versin 5, U.S. Bureau of Labor Statistics, Washington, DC, 1980.

TROL79 *TROLL User's Manual*, MIT, Cambridge, MA, 1979.

TUR79 Turner, M., Hammond, R., and Cotton, P. A DBMS for Large Statistical Databases. *Proceedings of the 5th International Conference on VLDB*, Rio De Janeiro, Brazil, 1979.

WEG80 Wegner, P. *Programming with ADA: An Introduction by Means of Graduated Examples*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

WEL81 Welty, C., and Stemple, W. Human Factors Comparisons of a Procedural and a Nonprocedural Query Language. *ACM Transactions on Database Systems*, Vol. 6, No. 4, 1981.

8. APPENDIX

Primitive Operations for Complex Data Types:

SET: s1, s2: set

- ELEMENT(s1) : single element
- INTERSECT(s1,s2): elements common to both s1 and s2
- UNION(s1,s2): elements in either set, remove duplicates
- DIFFERENCE(s1,s2): elements in s1 that are not in s2
- CARDINALITY(s1): number of elements in s1

VECTOR: v1, v2: vector; p_ex: predicate expression; i: vector index

- ELEMENT(v1,i): element located at position i
- INDEX(v1,p_ex): indexset of position matching p_ex
- INSERT(v1,e,i): vector with element e inserted after position i
- DELETE(v1,i): vector with element located at position i deleted

ORDERED SET: os: ordered set; e: element of os; i: index of os

- SET primitives: CARDINALITY; INTERSECT
- VECTOR primitives: ELEMENT; DELETE
- INSERT(os,e,i): ordered set with element e inserted after position i IF e was not a member of os, ELSE no effect.

MATRIX: m1,m2: matrix; e: matrix element; i: matrix index; d: dimension

- ELEMENT(m1,i): element at position i
- INDEX(m1,e): set of indices specifying the positions of the element in the matrix
- SHAPE(m1): vector containing dimensions of m1
- JOIN(m1,m2,d): matrix which is the concatenation of m1 and m2 over dimension d
- DELETE(m1,i): matrix in which the element at position i is logically deleted.
- REPLACE(m1,i,e): matrix in which element at position i is replaced by e

TIME: t1,t2: time

- TIME: date and time of day
- Component extractor (t1): e.g., YEAR, DAY, HOUR
- BEFORE(t1,t2)/AFTER(t1,t2): boolean variable
- DIFFERENCE(t1,t2): duration value

TIME SERIES: s1,s2: time series; agg_fnct: built-in or user defined function for specifying periodicity

- All MATRIX primitives.
- MERGE(s1,s2): time series representing the merge of s1 and s2 along *both* case and observation indices
- TIME_COLLAPSE(s1,agg_fnct): time series which has been either aggregated or disaggregated to a form specified by the agg_fnct

TEXT t:text; i: text index; s: string

- All VECTOR primitives apply to text
- CHARACTER(t,i): ith character in t
- WORD(t,i): ith word in t

G_RELATION: All SET primitives

- RELATIONAL ALGEBRA primitives (e.g., DAT81) modified to include the semantics of summary and identifying domains. These operators, and their modifications, are described fully in [SU82].

DATA STRUCTURES FOR SCIENTIFIC SIMULATION PROGRAMS

Jean Bell

Computer Science Department, University of Colorado, Boulder

Abstract

This research investigates data management, data structures, and constraints on data structure design in a cross section of programs known as "scientific models". Scientific models are large programs that use numerical approximation techniques to simulate physical phenomena for which exact solution is impossible, e.g. weather forecasting models. Scientific models, as a class, use remarkably similar data structures. The generic structures and access patterns in scientific models are the basis for functional specification of a data management system tailored to this application class.

This article is an extended abstract of dissertation research about data management in scientific simulation programs. Scientific simulation programs, also called "scientific models", are programs used for prediction of physical phenomena; weather prediction, nuclear reaction, and climatology simulations are examples of programs in this class. Of particular interest are the data structures and data structuring principles in such programs, especially the content, organization, and access patterns for data retrieval.

The main research activities are a survey of data management characteristics and a codification of the survey findings. We investigate what structures are used in individual scientific models, and whether generic structures are applicable to a wide class of models. Generic data structures are identified. These structures provide the necessary basis for building efficient, general purpose data management tools for scientific models. A data management tool tailored for scientific models is specified in the dissertation.

The scientific models in the survey were carefully selected to be representative of the entire class. Twenty-seven models were investigated in the survey, and eleven of those were the subject of extremely detailed analysis. Many previously unknown similarities in data management across a wide range of scientific disciplines and numerical methods were found.

Factors that influence data structure design include characteristics of the physical phenomena, the mathematical equations, the numerical approximation and solution methods, and the computer environment. Many of these characteristics constrain choice of data structure, and hence are called "constraints". The analysis of constraints focuses on application characteristics, rather than on computer characteristics. The understanding of constraints on access patterns and data arrangements can therefore be used within different computer environments or in the design of future computer environments for scientific models.

In summary, the research reported here addresses the following questions:

(1) What data structures are implemented in scientific simulation programs?

(2) What constraints influence the data structure design in such programs?

(3) Can the access patterns in such programs be categorized such that a small set of data structures integrates the access patterns in many such models?

1. NEED FOR RESEARCH INTO SCIENTIFIC DATA MANAGEMENT

Scientific applications often stress the computer environment to its limits, so that in order to do the calculation at all, the scientist must make optimum use of his resources. Sequential file processing predominates, because it minimizes the expense and the difficulties of data management across memories. Standard database management systems are not used for two reasons: they are highly inefficient, and hence too expensive, for scientific applications; and, a scientist often views his own computing problems as unique, so that no general purpose tool would be applicable.

Because scientists do not use data management systems, there is a natural tendency for data management researchers to ignore scientific applications. However, it has never been determined whether scientists avoid general purpose tools because they are not needed, or because tools that fit scientists' needs are not available. Standard database management systems are optimized for com-

mercial applications that need interactive retrieval of random requests. Scientific applications, on the other hand, are often batch jobs with predefined retrieval patterns. Virtual memory is inefficient for the data access patterns in large arrays that are typical in scientific applications [Mura80]. A few specialized data management packages for scientific applications have been constructed, but none is widely used. Data management in scientific applications must thoroughly investigated, in order to understand what tools are suitable.

2. SCIENTIFIC MODELS

Scientific models are programs used for approximate prediction of physical phenomena, such as weather, ocean currents, and nuclear reactor dynamics. A large system of partial differential equations describe the physical relationships in the model. The systems of equations cannot usually be solved exactly, so numerical analysis techniques are used to approximate their solution. Efficient data management is critical in scientific models because they are computationally very intensive AND they have a high volume of data.

The physical domain in a scientific model is discretized into a large number of small areas by superimposing a grid on the domain (see Figure 1-1). The phenomena of interest are predicted for each grid cell, e.g. temperature at some future time in each cell is predicted as a function of the temperature and humidity in the cell and surrounding cells at a previous time. The value of a physical quantity for one grid cell may be the average value for the entire space in the cell, or it may be the value at a particular point in the cell. The values of physical variables across all cells in the grid forms the composite prediction (e.g. a weather map).

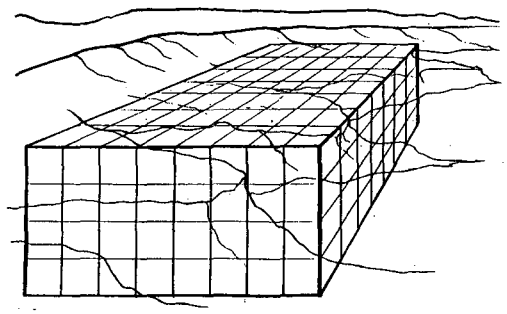


Figure 1-1. Superposition of a three dimensional rectangular grid on a spatial area.

Some models also track particles (e.g. electrons) that represent a discrete unit of matter across the spatial domain. By tracking a large number of particles, the model obtains a very accurate picture of material movement. In particle models, physical properties of both the grid cells and the particles are predicted.

Time is also discretized, into many small "timesteps". The state of the physical system is recorded for all grid cells and particles at the end of each timestep. The predicted values at the current and/or the previous timesteps are the input to the next prediction (see Figure 1-2). The timestep is a basic unit of computation, in that the same sequence of calculations is performed in each timestep. Within timestep, data about grid cells are updated in a fixed sequence. The access pattern to individual grid cells, and within grid cell to individual variables, is almost always totally defined before the program is compiled.

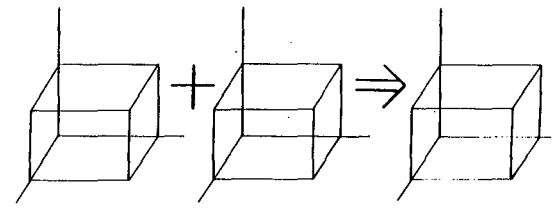


Figure 1-2. Timesteps in the predictive process: example shows the "leapfrog" explicit time integration method, in which predictions of new grid values are a function of the current and the previous predictions.

In a typical scientific model, data about the grid cells comprises the bulk of the stored data. The amount of grid data is a function of the grid resolution, the number of physical phenomena being predicted, and the domain size. In the FTELT model, for example, the grid typically contains 20,000 cells, and 150 variables are stored about each cell, for a total of about three million numbers of grid data. Those three million numbers will ALL be updated in each timestep. Dozens of timesteps are made in one model execution, so over one hundred million updates will be performed in one simulation.

The amount of data precludes its storage in central memory, but the data is accessed repeatedly by the calculations. Under these circumstances, efficient data management is a primary performance criterion. The structures for data on the external storage device must be designed to avoid

unnecessary block transfers. Partitioning into blocks must take into account the calculational dependencies; that is, all variables needed to update each cell in a block should be in central memory at the same time. On the other hand, the amount of storage in central memory for buffer space, the transfer rates, and other characteristics of the computer hardware, must be taken into account to determine block size for efficient transfer. Storage structure design is constrained by the complex interrelationships among these and other decision variables.

3. MAJOR SURVEY FINDINGS

The dissertation research cuts across subject areas and numerical methods to illuminate the larger issues in data structure design in a way previously obscured by the details of computer implementations. Many similarities in data structures among scientific models were found. For example, a major research task was the classification of data structures in scientific models. The classification shows that a small set of data structures are sufficient for supporting the access patterns in a wide range of scientific models. This finding is of great significance because it permits the building of tools which are both generally useful and specifically efficient.

Many of the results in the dissertation analyze constraints on data structure design. For example, the desirability of using a rectangular grid for modeling a variety of physical domains is analyzed. The effects of computer environment on the details of storage structure design, such as block size, stratification into files, etc., are presented. The constraints placed by numerical method are also presented, particularly order of calculations and dependencies in the calculations. Understanding the factors that influence the data structure design adds to the ability to manage data in scientific simulation programs AS A CLASS. Such an understanding also contributes to the generalizability of the results to other application programs and computer environments.

The three most significant findings about implemented data structures in scientific models are summarized in the following subsections.

3.1. CLASSIFICATION OF STORAGE STRUCTURES FOR DISK FILES

All programs in the survey used only three types of storage structures for organizing disk files into blocks: planes, pencils, and sequential lists. The majority of scientific models have a rectangular three dimensional grid, i.e. a parallelepiped whose individual cells are parallelepipeds. For models with rectangular grids, the grid is partitioned into geometric portions, either "planes" and "pencils", and each partitioning defines a class of storage structures. For non-rectangular grids and for particle data, sequential list storage structures are used.

The choice of storage structure is constrained by the update dependencies in a scientific model. Spatial dependencies predominate in scientific models. That is, the calculations to update one cell require data simultaneously from some set of other cells. The set of cells usually has a characteristic geometric shape, and hence is called a "stencil". The associations between categories of stencils and categories of storage structures is explained below.

3.1.1. Planes

A plane contains all of the grid cells on a plane perpendicular to one of the grid axes (see Figure 1-3). In the plane storage structure, each block holds exactly one plane of the grid. The file consists of an ordered list of consecutive planes. The blocks are accessed in sequential order in a "moving window" sweep across the file, always retaining a certain number of consecutive planes (e.g. 5 planes) in central memory at a given time. In a typical sweep through a file, the first several higher level I/O operations build up the number of planes in central memory, i.e. "Get next block AND retain all previous blocks". After the requisite number of planes are in memory, the higher level I/O operation is "Get next block AND replace oldest block".

There are some variations on these higher level operations. For instance, a backwards sweep moves through planes from last plane to first. Also, there may be minor perturbations in the order of access to the first and last planes because of special calculations at the edges of the domain. However, the variants are classified together

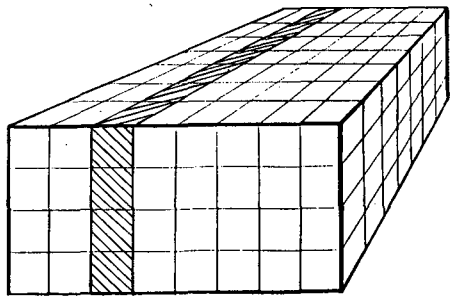


Figure 1-3. Rectangular three dimensional grid subdivided into planes: hatched area is one plane.

because of the characteristic moving window sweep through the majority of planes, and the geometric partition of cells into disk blocks by planar location.

The plane data structure is used in all computations in which the stencil for updating a particular cell includes a local set of adjacent cells in all three dimensions within the grid (e.g., see Figure 1-4). In calculations with a local, three dimensional stencil, the plane data structure minimizes the coupling between blocks because of the simple fact that all other geometric subdivisions have more sides. That is, all cells on a plane are adjacent to cells in only its two adjacent planes; for any other geometric subdivision, cells in one subdivision are adjacent to cells in more than two other subdivisions. Furthermore, any other partition would require re- reading of at least some blocks during the update of all cells in the grid.

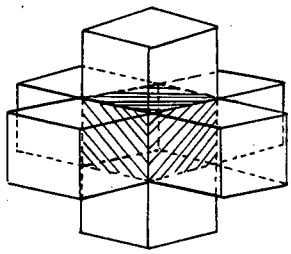


Figure 1-4. Stencil for a three dimensional second order finite difference calculation: hatched area is the cell to be updated, as a function of data about the other cells in the stencil.

3.1.2. Pencils

A pencil is a bundle of adjacent lines, where each line spans the grid in the dimension of its length. The pencil data structure is particularly suited to algorithms that have a one dimensional stencil. That is, in order to update a particular cell, other cells on a line (perhaps the whole line) are needed simultaneously, but there is no coupling to cells on any other line.

Scientific models that use the pencil storage structure often use a stencil that has been "factored". That is, the stencil may contain cells in all three dimensions, but the stencil has been broken up into a set of stencils, each of which contains cells in only one dimension. The algorithm, then, uses several access patterns within a timestep, each of which sweep through the lines in the grid in a different dimension. Therefore, pencils are further subdivided into "cubes" (not

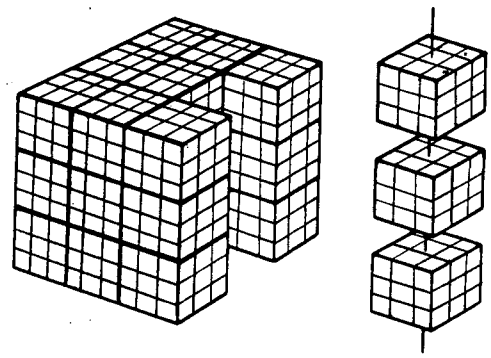


Figure 1-5. Rectangular three dimensional grid subdivided into cubes: stack of cubes is one pencil.

necessarily perfectly cubic in shape), where a cube is the intersection of pencils in three orthogonal dimensions (see Figure 1-5). All of the cubes in a pencil in a particular dimension must be in memory when the cells in lines in that pencil are being updated, but no other cubes are required.

The pencil storage structure partitions the grid into cubes and each cube is stored in one block. The file is a three dimensional array of blocks, where each block has a linearized index. There is one secondary index to cubes within each pencil in each dimension. For example, for pencils in the "x" dimension, there is a list of the indices of cubes in the first pencil in that dimension, a list of indices of cubes in the second pencil in that dimension, etc. The higher level I/O operation, similar to an indexed sequential access primitive, is "Get all the blocks whose indices are on this list".

3.1.3. Sequential lists

The third storage structure, called an "sequential list", is used for particles, for irregularly shaped grids, and for other files where there are few update dependencies among instances of a single entity type. For example, no data about any other particle is needed when data about a particular particle is updated. Also, the particles can be updated in any order. Thus, no coupling among particles must be preserved in the partitioning of particle data into blocks. Only one block is needed in central memory at one time, since there is no coupling between blocks. There is sometimes an order to the blocks. For example, particles may be partially ordered by location to facilitate access to cell data. However, the blocks are always accessed in sequential order, so the appropriate access operation is a simple sequential fetch, i.e. "Get next block".

3.2. SIMPLE USER LEVEL DATA STRUCTURES

Scientific models store data about only a few types of entities: grid cells, other subregions of space, particles, and materials. Grid cells are the subdivisions of space defined by superimposing a grid on the spatial domain. Other subregions are aggregations of grid cells based on common properties, e.g. all cells in the subregion defined by the vessel part of a nuclear reactor. Particles are units of material, e.g. neutrons. Materials (e.g. steel) and material-particle interactions (e.g. neutrons hitting steel) are together classified as "handbook" data because only static facts are recorded about these entities; thus, this type of data resembles a computerized chemistry handbook.

The user database is simple in other ways: few facts ("attributes") about most entities, few types of relationships, and few types of allowable operations in the user data structures. The simplicity of user data structures implies that full generality of data management is not needed. Data management tools for scientific simulation programs can be tailored to handle the specific user data structures without loss of applicability. For example, each entity type has a characteristic form of identifier. Particles have no unique identifier, since their individual properties are not of interest. Grid cells and other subregions have a implied location identifier, usually an array index. By knowing the full range of identifier types, the user interface can be greatly simplified.

3.3. PREDEFINED ORDER OF ACCESS

Order of access to disk files is predefined in scientific models; the order in which files are read, and the order in which blocks are read within file, is totally predetermined before compile time. Truly random access, i.e. where the decision on what entity to access next depends on a updatable value in the data itself, is not used in any model in the survey. The dissertation shows that random access would introduce inefficiencies because of the interaction between the stencil, the volume of stored data, and the fact that all values of all records are updated in every timestep. Other inefficiencies, such as redundant data storage, introduced in order to avoid random access are also analyzed. The analysis shows that random access is the source of greater inefficiency.

The significance of this discovery is that data transfers can be highly optimized by prefetching. A data management tool would need some dynamic information about timing of transfers, but order of transfer can be entirely prespecified. In sharp contrast, standard database management systems assume that truly random access will be used heavily, so they require a much different approach to the problem of data transfer.

4. A DATA MANAGEMENT TOOL FOR SCIENTIFIC MODELS

The dissertation includes a specification of the functional requirements for a "scientific model data management system" (SMDMS) from the user perspective, and also from the perspective of storage management. In the proposed tool, the user specifies certain parameters of his application, e.g. the size and shape of his grid, the use of particle data, etc. In the main part of his program, the user specifies one or more sets of calculations, called datapasses, to be performed on each instance of an entity type. A datapass retrieves all instances, performing the set of calculations in the datapass on each instance in its turn. The user specifies parameters of the datapass, e.g. the order of update and the stencil to be used. Based on these user parameters, the appropriate storage management and data transfer commands are automatically activated by the SMDMS.

The heart of the data management system specification is the user interface. The interface is tailored for the scientist user by taking into account access patterns in scientific models as a class. The datapass concept, which specifies a set of calculations to be applied to every instance of a relation, is equivalent to a series of queries and updates in a standard database management system. Since the same calculations are performed on each instance, the datapass provides a convenient and succinct specification of the entire set of queries. Furthermore, it allows the data management system to optimize storage organization and disk transfers for the entire series of updates.

5. CONCLUSION

The most general statement of the original goal of this research was to assist physical scientists and hardware manufacturers in providing data management facilities for large scientific models. The first step in the process of tool building is to achieve the understanding of functional requirements for the tools. To this end, the investigation focused on the data structures, both user visible and underlying storage structures, that support the access and data patterns in scientific models.

Investigation of data structures used in current implementations of scientific models, and of con-

straints on their design, formed the main part of this dissertation research. The data collected in the investigation provided a rich and detailed description of all phases of data management in scientific models.

Of special interest in the analysis is data structure at the level of file storage: the partition and organization of data into records, blocks, and files so that access across memories is efficient for individual models. Storage structures for file organization into blocks fall into only a few categories: planes, pencils, and sequential lists. The three categories are well defined by both their storage arrangements and higher level I/O primitives.

The dissertation presents a data management tool (SMDMS) tailored to the access patterns and data content in scientific models. The functional requirements for the SMDMS data management system are similar to a standard data management system, but the characteristic kinds of retrievals and updates for scientific models are quite different. Efficiency requirements are critical because of the very large number of updates in a scientific model execution. Efficient implementation is based on the categories of storage structures discovered in the research.

A full database management system can now be designed, using the functional specification in the dissertation. Implementation of a SMDMS system is necessary to test the usability and the practical efficiency of the data structures and data management system developed in this dissertation research. The final product, i.e. the data management system system itself, will directly benefit the computer user community of scientific modelers. In addition, it will provide data management researchers with an interesting tool for studying the usefulness and efficiency of specialized data management systems.

Acknowledgement

Special gratitude goes to my three main advisors in this research: H. Paul Zeiger, G. Stuart Patterson, Jr., and Richard Hackathorn. The research was sponsored by Cray Laboratories, Inc., Boulder, and the National Center for Atmospheric Research, Boulder.

References

[Mura80]
Muramatsu, H. and Negishi, H. "Page Replacement Algorithm for Large-array Manipulation," Software -- Practice and Experience 10, 7 (July 1980), 575-587.

AN EXTENSION OF RELATIONAL ALGEBRA FOR SUMMARY TABLES*

Z. Meral Ozsoyoglu and Gultekin Ozsoyoglu

Department of Computer Engineering and Science
Case Western Reserve University
Cleveland, OH 44106

Abstract

A summary table is one of the useful data structures used in statistical databases. For an algebraic summary table manipulation language, we first extend relational algebra for nested relations and aggregate functions, then propose a summary table manipulation language based on the extended algebra. A new operator, called aggregation-by-template is introduced, and other operators of the relational algebra are modified to apply nested relations. A special case of summary tables, called primitive summary table, is distinquished since it can be directly represented by a nested relation. Primitive summary tables are viewed as building blocks of summary tables. Operators for constructing and manipulating summary tables, and their properties are also discussed.

1. INTRODUCTION

Tabular representations of summary data, hereafter called summary tables, are widely used in various application areas such as management decision making, health care, economic planning and census data evaluation. Figure 1 contains an example for a summary table.

SUM-SALARY-OF EMPLOYEES			DIV: divl		div2
	DIV: divl	div2	DEPT: man	personnel	DEPT: acct
AGE:	SUM-SAL:		SUM-SAL:		
[18,30]	250K	290K	100K	150K	290K
[31,40]	500K	400K	200K	300K	400K
[41,60]	600K	250K	250K	350K	250K

Figure 1: An example summary table: SUM-SALARY-OF-EMPLOYEES

The use of summary tables is not restricted to output formatting; they are maintained for bookkeeping, compared and evaluated perhaps over a time span. Thus, starting from the premise that summary tables are proper logical modeling tools, we need a data manipulation language for summary tables.

Current statistical software packages have only summary table creation capabilities, and usually have a list of commands to manipulate only a single flat file. A notable exception is the Table Producing Language system (TPL) of the U.S. Bureau of Labor Statistics [Uslb 80] that has powerful facilities to produce summary tables. However TPL does not manipulate or store summary tables, and is executed as a stand-alone system in batch mode. Therefore, it is not part of an integrated data manipulation language.

This paper extends relational algebra [Codd 72] to handle summary tables. More specifically, we allow set-valued columns in relations, and add or modify relational algebra operators for manipulating summary data maintained in relations.

* This research is supported in part by the National Science Foundation under Grant MCS-8306616.

Recently Klug [Klug 82] extended relational algebra by incorporating aggregate functions, and proposed a new operator, called aggregate formation. This operator uses the concept of partitioning a relation and applies an aggregate function to each partition. For summary table manipulation, we define another operator, called aggregation-by-template, based on the concept of grouping (not partitioning). Jaesche and Schek [Jaes 82] define an extension of relational algebra for nonfirst normal form relations (i.e. a tuple component may be a set or a set of sets, etc.).

We use nested relations to represent a special case of summary tables which are called primitive summary tables. Informally, a nested relation is a relation where tuple components for zero or more of its attributes are sets of simple values. Primitive summary tables are used as building blocks of summary tables. That is, a summary table can be decomposed into primitive summary tables, and two or more primitive summary tables can be combined into a larger summary table. The relational algebra is extended for aggregate functions and nested relations so that it forms a basis for a summary table manipulation language.

In Section 2, we introduce the terminology and definitions. Operators that define the algebra of nested relations are given in Section 3. The operations for arithmetic on nested relations are discussed in Section 4 as an additional feature of the extended algebra. Section 5 discusses operators to construct and to manipulate arbitrarily general summary tables. Section 6 is the conclusion.

2. TERMINOLOGY AND DEFINITIONS

2.1 Relational Model and Nested Relations

A relation is a set of n-tuples, for some fixed n>0. Each component of a tuple in relation R has a name, which is called an attribute of R. A relation scheme is a set of attributes. As a notation we deal with column-ordered relations and refer to attributes by column numbers as in [Codd 72, Klug 82, Ullm 82] instead of attribute names unless explicitly stated. The attributes of a relation R of degree n (i.e. R with n attributes) are identified by integers 1,2,...,n. The set of attributes of R is denoted Atr(R). Each attribute of a relation has an associated domain of values. Let U be the set of all values regarded as atomic (such as integers, reals, character strings, etc.) and a value null denoted '-'. Let $D_1,\ldots,D_n$ be subsets of U. A first normal form relation R is a subset of $D_1 x \ldots x D_n$ where x is the cartesian product [Codd 72].

An attribute is called atomic if its domain is a subset of U. If the domain of an attribute is a subset of P(U) (i.e. power set of U) then it is called a nested attribute. A nested relation R is a relation whose attributes are either atomic or nested.

In this paper, we use functional dependencies and embedded join dependencies of relational model [Codd 72, Ullman 82]. Functional dependencies are defined for nested relations without any change. The projection operator is directly applicable to nested relations. For the natural join on nested relations, the join attributes in the relations joined are either both nested or both atomic attributes. Thus, the join dependency and the embedded join dependency generalize to nested relations straightforwardly.

2.2 Summary Tables

Informally, a summary table scheme is a two dimensional table of cells. Each cell can be considered as an element in a two dimensional array. The rows and columns of a summary table have some attributes called category attributes which correspond to array subscripts in a sense. Category attributes in a row or in a column may be structured as forests of trees whose nodes are attributes. In a summary table, attributes which appear in a root-to-leaf path in a row or column category attribute tree are called row category attributes or column category attributes, respectively, of a cell. In addition to category attributes, a cell also has an attribute called cell attribute. The following example illustrates a summary table scheme.

Example 2.1: There are two cells in the summary table below. The row category attribute is AGE for both cells and column category attribute DIV belongs to the first cell, and DIV and DEPT belong to the second cell. The cell attribute, SUM-SAL, is the same for both cells.

SUM-SALARY-OF-EMPLOYEES	DIV	DIV / DEPT
AGE	SUM-SAL	SUM-SAL

More formally, a summary table scheme is a 4-tuple $S(F_r,F_c,A_c,M)$ where F_r and F_c are row and column category attribute forests, A_c is an ordered set of cell attributes and M is a mapping function. F_r and F_c are ordered forests of ordered trees (F_r or F_c may be empty but not both). A nonempty category attribute forest F is denoted as $F=<T_1,\ldots,T_k>$ where each $T_i=(V_i,E_i)$ is an ordered tree whose vertex set is V_i and edge set is E_i. If m and n are the total number of leaves in F_r and F_c, respectively, then there are m*n cells in S for $m\neq 0$ and $n\neq 0$. If m or n is zero then the number of cells is n or m respectively. Each cell in S has a pair of ordered sets of attributes as row and column category attributes. The row and column category attributes of a cell appear in a root-to-leaf path of a tree in F_r and F_c respectively. The function M maps a pair of category attribute sets for a cell into an attribute in A_c, which is the cell attribute of the cell. The function M is one-to-one and onto. Cell attributes in A_c are not necessarily distinct, i.e., A_c is an ordered multiset.

For notational convenience, we omit the function M and assume that the mapping from pairs of category attribute sets to cell attributes is done as follows. Let leaves in F_r and F_c be numbered from 1 to m and from 1 to n respectively. (Top leaf of F_r and leftmost leaf of F_c are both numbered as 1.) Then category attributes in the path from a root to the i^{th} leaf in F_r and category attributes in the path from a root to the j^{th} leaf in F_c map to the $(i*(n-1)+j)$ th cell attribute in A_c, where $1\leq i\leq m$, $1\leq j\leq n$. Having fixed the mapping function M, we use $S(F_r,F_c,A_c)$ to denote a summary table scheme.

A category attribute may be nested or atomic, i.e. its domain is a subset of P(U) or U. A cell attribute is an atomic attribute. A Summary table instance for $S(F_r,F_c,A_c)$ is a two dimensional table of cell instances corresponding to cells in S. A cell instance has row and column category attribute values and a cell attribute value. Let T=(V,E) be a tree in F_r or F_c of S. An instance t of T is an ordered tree, defined recursively as follows:

If T is null (i.e. $V=\emptyset$, $E=\emptyset$) then t is null. Otherwise,
(a) the root of t is a value for the attribute V_0 which is the root of T.
(b) let $V_1,\ldots,V_k$ be the children of V_0 in T and $T_1,\ldots,T_k$ be the subtrees of T such that V_i is the root of T_i, $1\leq i\leq k$. Then, subtrees of t are instances of T_i, $1\leq i\leq k$, such that instances of T_{i+1} follow instances of T_i, $1\leq i\leq k$. (i.e. ordered).

Example 2.2: Consider the column category attribute tree T=(V,E) where V={DIV,DEPT}, E={(DIV,DEPT)} in the summary table scheme given in Example 2.1. There are two instances, say t_1, t_2, of T in the summary table instance shown in Figure 1, where

$t_1=(V_1,E_1)=(\{div1,man,personnel\},\{(div1,man),(div1,personnel)\})$, and

$t_2=(V_2,E_2)=(\{div2,acct\},\{(div2,acct)\})$.

An instance of a category attribute forest F is an ordered set of instances of the trees in F. An instance for a summary table $S(F_r,F_c,A_c)$ has instances of F_r and F_c as row and column forests respectively. A pair of root-to-leaf paths in an instance of F_r and in an instance of F_c defines a cell instance. Each such cell instance has a cell attribute value. We will use term "cell" (and also the term "summary table") to denote a scheme or an instance interchangeably whenever there is no ambiguity.

Semantically, each cell in a summary table corresponds to a group of individuals in a given population such that category attribute values of the cell define a group of individuals and the cell attribute value is the result of an aggregation applied over this group.

2.3 Primitive Summary Tables

A summary table $S(F_r,F_c,A_c)$ where F_r and F_c each consists of a single chain of

attributes (i.e. a tree with one leaf) is called a primitive summary table. In other words, a primitive summary table has exactly one cell. Primitive summary tables are basic building blocks of summary tables since each cell in a summary table S corresponds to a primitive summary table. Figure 2 shows the schema and the instance of one of the two primitive summary tables for the summary table given in Figure 1.

ST2	DIV
	DEPT
AGE	SUM-SAL

ST2	DIV:		
	divl		div2
	DEPT:		DEPT:
	man	personnel	acct
AGE:	SUM-SAL:		SUM-SAL:
[18-30]	100K	150K	290K
[31-40]	200K	300K	400K
[41-60]	250K	350K	250K

Figure 2: One of the primitive summary tables for SUM-SALARY-OF-EMPLOYEES.

Our approach for defining operations for summary tables is first to define operations to construct and manipulate primitive summary tables, then to extend the language to deal with arbitrary summary tables in terms of these operations.

A nested relation can be used to represent a primitive summary table excluding the ordering and the type (row or column) of category attributes. Let $S(F_r,F_c,A_c)$ be a summary table where X and Y are sets of attributes in F_r and F_c respectively, $X \cap Y = \emptyset$, and $A_c=\{C\}$. Then a nested relation R where $Atr(R)=X \cup Y \cup C$ can be used to represent S such that each tuple t in R corresponds to a cell occurrence in S whose row and column category attribute values are tuple components in t[X] and t[Y] respectively and whose cell attribute value is t[C]. Let R be such a relation representing a primitive summary table S. Then the f.d. XY->C holds in R since each cell occurrence in S is uniquely identified by category attribute values. Moreover, the e.j.d. *(X,Y) holds in R since there is a cell occurrence in S for every pair of X,Y values. If R is such a nested relation representing a primitive summary table S we say that R and S are information equivalent. Similarly, given a relation R and a primitive summary table scheme $S(F_r,F_c,A_c)$ where X and Y are sets of attributes in F_r and F_c, $A_c=\{C\}$ and $Atr(R)=X \cup Y \cup C$, an instance for S can be directly constructed from R if f.d. XY->C and e.j.d. *(X,Y) hold in R.

Consider a nested relation R representing a primitive summary table S. Attributes of R corresponding to cell and category attributes of S should be semantically distinguished. For example, a relation R' obtained from R by projecting out some category attributes may not represent a meaningful summary table. That is because changing category attributes effects the underlying populations over which the aggregation is done. For R' to represent a summary table, its column representing the cell attribute should be changed appropriately whenever columns for category attributes are changed.

2.4 Aggregate Functions

An aggregate function, given a set of tuples and a column number (attribute) i, returns a simple value obtained from the ith components of tuples in the given set. The attribute i should be an atomic attribute. In general, the domain of attribute i should be compatible with the aggregate function, i.e. the result of the aggregate function when applied to column i should not be undefined. When an aggregate function is applied over an empty set of tuples, the result is null, denoted by '-'. The null value is an atomic value and is included in the set U of all atomic values. The meaning of the null value is 'nonexistent'. When an aggregate function is applied to attribute i of a nonempty set of tuples, tuples t with $t[i] \neq null$ contribute to the result, and tuples with t[i]=null are simply ignored. If all tuples, over which the aggregate function is applied have t[i]=null then the result is also null, i.e., the same as the case when the aggregate function is applied over an empty set of tuples. The result of an aggregate function over a nonempty set of tuples with one or more tuples t having $t[i] \neq null$ is not null. In this paper, we consider only aggregate functions MAX,MIN,SUM and COUNT. However, results can be extended with minor modifications to include any aggregate function.

3. AN ALGEBRA OF NESTED RELATIONS

In this section we first give operators for an extension of relational algebra to

nested relations; and then identify the basic set of operators of the extended algebra. The operands of the extended algebra are either constant relations or variables denoting relations with fixed number of columns. We use the term relation for nested relations as well as 1NF relations. The following notation is used throughout the paper.

Notation: Let t_1, t_2 be two tuples having components for a set of attributes X. Then $t_1[X]=t_2[X]$ denotes $t_1[X_i]=t_2[X_i]$ for each attribute X_i in X where '=' is simple equality if X_i is an atomic attribute and it is a set equality if X_i is a nested attribute.

3.1 Aggregation Operators

Aggregate Formation: Let R be a relation with attributes Atr(R) and $X \subseteq Atr(R)$, $|X|=k$. Let f be an aggregate function and A be an atomic attribute of R. Then $R<X,f_A>$ is a relation with degree k+1 and is defined as

$$R<X,f_A>=\{t[X]oy \mid t\in R \wedge y=f_A(\{t' \mid t'\in R \wedge t'[X]=t[X]\})\}$$

where "o" denotes concatenation.

The aggregate formation operator first partitions tuples of relation R such that tuples having the same X component are in the same partition. Then, the function f is applied to component A of tuples in each partition, and the X-value and the associated value produced by the aggregate function are output for each partition. The aggregate formation operator for 1NF relations is defined by Klug [Klug 82]. The definition given above is a straightforward extension of this operator to nested relations.

Aggregation-by-Template: Let R_1 and R_2 be two relations with attributes $Atr(R_1)$ and $Atr(R_2)=Y \subseteq Atr(R_1)$, $Z=Atr(R_2)$ where $|Y|=|Z|\geq 1$ and each attribute in Z is a nested attribute. Let $X \subseteq Atr(R_1)$ such that if X is nonempty then X and Y are disjoint, and A be an atomic attribute of R_1. For notational convenience let Y_a denote the set (possibly empty) of column numbers (attributes) in Y that are atomic in R_1, and $Y_n=Y-Y_a$. Let Z_a and Z_n denote those attributes in Z that correspond to Y_a and Y_n. Then $R_1<X,Y,f_A>R_2$ is a relation with degree $|X|+|Y|+1$ and is defined as follows.

$$\begin{aligned} R_1<X,Y,f_A>R_2 = \{toy \mid\ & (\exists t_1)(\exists t_2)(t_1\in R_1 \wedge t_2\in R_2 \wedge \\ & t[X]=t_1[X] \wedge t[Z]=t_2[Z] \wedge \\ & y=f_A(\{t' \mid t'\in R_1 \wedge t'[X]=t[X] \wedge \\ & t'[Y_a]\in t[Z_a] \wedge t'[Y_n]\subseteq t[Z_n]\}))\}. \end{aligned}$$

The aggregation-by-template operator $R_1<X,Y,f_A>R_2$ groups tuples of R_1 as follows: Let t be a tuple over the attributes $X \cup Z$, where

$t[X]=t_1[X]$ for some tuple t_1 in R_1, and $t[Z]=t_2[Z]$ for some tuple t_2 in R_2.

Each such tuple t defines a group G_t of tuples of R_1 such that a tuple v of R_1 is in G_t if

$$v[X] = t[X] \ , \ v[Y_a] \in t[Z_a], \text{ and } v[Y_n] \subseteq t[Z_n].$$

Then the aggregation function f is applied on attribute A of tuples of R_1 in each group. Finally, the X-value, the Z-value, and the associated aggregate value is output for each group. Thus the number of tuples in $R_1<X,Y,f_A>R_2$ is the product of the number of tuples in $R_1[X]$ and the number of tuples in R_2. In aggregation-by-template, tuples of R_2 direct the grouping (i.e., R_2 is the "template"). Consequently, there may be some empty groups. The value returned by the aggregate function f applied over an empty group is null, which is denoted by "-". The treatment of null values is discussed in section 2.3.

The aggregation-by-template is more convenient than the aggregate formation when there are prespecified groupings attached to category attributes. For example, sum salary of employees by a specified set of age groups can be tabulated by a single aggregation-by-template. When dealing with summary data it is common to have aggregations over populations defined by prespecified groupings of category attribute values.

Furthermore, the aggregation-by-template is based on grouping tuples of a relation while the aggregate formation is based on partitioning the tuples.

Example 3.1: As a running example, consider the following relation

SKIER:	NAME	RACE-TYPES	NO-OF-INJURIES	MISSED-SEASONS	NO-OF-RACES WON
	I.Stenmark	{S,GS}	2	{1976}	7
	E.Dodge	{D}	2	{1973,1976}	6
	E.Halsnes	{D}	4	{1976}	6

where S,GS and D denote Slalom, Giant-Slalom and Downhill respectively. We may want to find the total number of races won by each group of skiers who compete in the same set of race types and either
(a) had 1 or 2 injuries, and missed 1975 or 1976 seasons, or
(b) had 2 or 3 injuries, and missed 1976 or 1977 seasons.

The template relation is

TEMP:	NO-OF-INJURIES	MISSED-SEASONS
	{1,2}	{1975,1976}
	{2,3}	{1976,1977}

The query is SKIER<{2},{3,4},SUM_5>TEMP, resulting

RACE-TYPES	NO-OF-INJURIES	MISSED-SEASONS	TOTAL
{S,GS}	{1,2}	{1975,1976}	7
{S,GS}	{2,3}	{1976,1977}	7
{D}	{1,2}	{1975,1976}	null
{D}	{2,3}	{1976,1977}	null

Note that the tuple for skier I.Stenmark belongs to two partitions whereas the tuples for E.Dodge and E.Halsnes do not belong to any partition.

3.2 Pack, Unpack and Set Formation Operators

In this section we give operators that change the nesting depth of attributes in a relation.

Pack: Let R be a relation with $|Atr(R)|=n$, $A \in Atr(R)$ and $C_A=Atr(R)-\{A\}$. For each (n-1)-tuple g in $\Pi_{C_A}(R)$, an n-tuple W_g is defined as follows:

$$W_g[C_A] = g$$

$$W_g[A] = \{t[A] \mid t \in R \wedge t[C_A]=g\} \quad \text{if A is an atomic attribute}$$

$$\{x \mid (\exists t)(t \in R \wedge t[C_A]=g \wedge x \in t[A])\} \quad \text{otherwise.}$$

Then $P_A(R) = \{W_g \mid g \in \Pi_{C_A}(R)\}$.

The pack operator $P_A(R)$ maps (packs) sets of tuples in R, whose n-1 components for attributes in C_A are the same, into single tuples. The C_A-component of the packed tuple is the same as the C_A-component of those tuples that are packed. The A-value of the packed tuple is the set of A-values of the corresponding tuples if A is an atomic attribute in R. If A is a nested attribute in R then the A-value of the packed tuple is the union of A-values of the corresponding tuples. The pack operator is similar to one-attribute nest operator described in [Jaes 82].

Unpack: Let R be a relation with attributes Atr(R), A Atr(R), and $C_A=Atr(R)-\{A\}$. For each tuple t R, a set of tuples $U_A(\{t\})$ is defined as follows:

$$U_A(\{t\}) = \begin{cases} \{t\} & \text{if A is an atomic attribute} \\ \{t' \mid t'[A] \in t[A] \wedge t'[C_A] = t[C_A]\} & \text{otherwise.} \end{cases}$$

Then $U_A(R) = \bigcup_{t \in R} U_A(\{t\})$.

If A is a nested attribute, $U_A(R)$ maps each tuple t in R into a set of tuples such that each element in t[A] becomes the A-value of one of the resulting tuples and the tuple components for the attributes C_A are the same as $t[C_A]$. If A is an atomic attribute then $U_A(R)=R$. The unpack operator is the same as the unnest operator in [Jaes 82].

Set Formation: Let R be a relation with attributes Atr(R), $A \in Atr(R)$, and $C_A = Atr(R) - \{A\}$. Then

$$\Psi_A(R) = \begin{cases} R & \text{if A is a nested attribute} \\ \{t' \mid (\exists t)(t \in R \wedge t'[A] = \{t[A]\} \wedge t'[C_A] = t[C_A])\} & \text{otherwise.} \end{cases}$$

This is a trivial operator. If the attribute A is an atomic attribute, for each tuple of R, $\Psi_A(R)$ replaces the tuple component for A by its singleton set. If the attribute A is nested, $\Psi_A(R)=R$. The set formation is required for other extended algebra operations.

3.3 The Standard Relational Algebra Operators

The relational algebra operators [Codd 72] include cartesian product, project, select, join, set union, set intersection, set difference and quotient. The cartesian product (x) and project (Π) apply directly to nested relations. For the union ($\cup$), the intersection ($\cap$), the set difference (-), and the quotient ($\div$) the corresponding attributes in both relations must have the same nesting depth (i.e. both of the ith attributes in the two relations are atomic or both are nested). The selection operator, the natural join and θ-join [Codd 70, Ullm 82] of two relations can be extended to nested relations with minor modifications [OzsO 83]. Recently a new operator called natural join by intersection has been introduced for nonfirst normal form relations [Jaes 82]. This operator also applies to nested relations without any modification.

3.4 Basic Set of Extended Algebra Operators

Basic set of operations of the extended algebra include the five basic operations of the relational algebra (union, set difference, cartesian product, selection and projection) extended for nested relations, and the operators aggregation-by-template, pack and unpack. All other operators can be expressed by the above eight operators and the aggregation-by-template in this basic set of operations can be replaced by the aggregate formation to obtain another basic set of operations [Ozso 83]. The algebraic laws involving the aggregation-by-template and other operators of the extended algebra, which are useful in query optimization, along with their proofs can be found in [OzsO 83].

4. ARITHMETIC CAPABILITIES

When dealing with summary data, arithmetic operations are frequently used. For example, we may want to tabulate the difference in sum salaries of employees by divisions and departments with respect to years 1981 and 1982, given the 1981 sum salaries and the 1982 sum salaries of those employees. As another example, we may want to tabulate sum salaries of employees multiplied by a constant for security reasons. For such operations, we add arithmetic capabilities to the algebra extended for nested relations. These arithmetic capabilities are limited since we use as operands of arithmetic expressions only components of the same tuple in a given nested relation. Note that the query language, SQL [Cham 1976] of System R [Cham 1981] allows arithmetic operations in much the same way (i.e. in SELECT clause of SQL).

Let R be a relation with $Atr(R) \supseteq X = \{X_1, \ldots, X_r\}$ and $Y = Atr(R) - X$. Let g be a valid arithmetic expression involving +, -, /, * as operators and X_i X as operands. Then

$$R\langle g(X)\rangle = \{t[Y]o y \mid t \in R \wedge y = g(t[X])\}$$

where g(t[X]) denotes $g(t[X_1], t[X_2],\ldots,t[X_n])$. Note that if R is degree n, $|Y|=k<n$ then the result of $R\langle g(X)\rangle$ is of degree k+1. Also, if $t[X_i]$ is null for some i, $1 \leq i \leq r$ then g(t[X]) is also null.

5. OPERATIONS ON SUMMARY TABLES

In this section we describe operations involving nested relations, primitive and complex summary tables. Basically, these operations facilitate utilization of the algebra of nested relations for summary table manipulation. Due to space limitations, only brief descriptions are included in this paper. Formal definitions of the operators and examples can be found in [OzsO 83].

Primitive Summary Table Formation: This operation produces a primitive summary table from a nested relation. Let R be a nested relation, $Atr(R)=X \cup Y \cup \{C\}$ where X and Y are disjoint sets of attributes, C is an atomic attribute and the f.d. XY->C holds in R. Then

$$ST_{(T_r,T_c,C)}(R)$$

produces a primitive summary table whose row and column category attribute trees are $T_r=(X,E_r)$ and $T_c=(Y,E_c)$ respectively and whose cell attribute is C. E_r and E_c are sets of ordered edges such that each of T_r and T_c is an ordered chain of attributes in X

and Y respectively. The primitive summary table formation maps each tuple in R into a cell instance in the summary table produced. This mapping is one-to-one.

Let $S=ST_{(T_r,T_c,C)}(R)$ where $Atr(R)=X\ Y\ \{C\}$, and vertex sets of T_r and T_c are X and Y respectively. Let x and y be ordered sets of values (vertices) in some instances of Tr and T_c in R respectively. If there is a tuple u in R such that u[X]=x and u[Y]=y then u[C] is the cell attribute value of the cell instance in S whose category attribute values are x and y respectively. If there is no such tuple in R then the cell attribute value is null ('-') which stands for "nonexistent". The f.d. XY->C in R guarantees that there is at most one cell attribute value in S for each cell. In addition, if the e.j.d. *(X,Y) holds in R then the mapping between tuples of R and cell instances of S is one-to-one and onto. That is, there is no cell instance, with '-' cell value, in S unless there are tuples in R whose C components are '-'.

Relation Formation: Let S be a primitive summary table where X and Y are row and column category attributes and C is the cell attribute, $X \cap Y=\emptyset$. Then REL(S) produces a relation R where $Atr(R)=X \cup Y \cup \{C\}$ and for each cell in S, there is a tuple t in R such that t[X] and t[Y] are the same as the row and the column category attribute values of the cell and t[C] in the cell attribute value of the cell. The mapping between cell instances in S and tuples in R is one-to-one and onto. That is, if R=REL(S) then the f.d. XY->C and the e.j.d. *(X,Y) hold in R.

Concatenation of Primitive Summary Tables: Let $S_1(F_{r1},F_{c1},A_1)$ and $S_2(F_{r2},F_{c2},A_2)$ be two summary tables, where F_{c1} and F_{c2} are not null. If $F_{r1}=F_{r2}$ and all tree instances of F_{r1} are the same as all tree instances of F_{r2} (denoted $F_{r1} \doteq F_{r2}$) then a larger summary table can be obtained by concatenating S_1 and S_2 such that the resulting summary table has $F_r \doteq F_{r1} \doteq F_{r2}$ as row category attribute forest; and has column category attribute forest F_c which is obtained by concatenating F_{c1} and F_{c2}. We call this operation column-concatenate summary tables S_1 and S_2, denoted as

$$CC(S_1,S_2).$$

The column-concatenate $CC(S_1,S_2)$ produces a summary table $S(F_r,F_c,A)$ where $F_r \doteq F_{r1} \doteq F_{c1}$ and $F_c \doteq F_{c1} o F_{c2}$ (o denotes concatenation of ordered sets). Instances of F_r in S are the same as instances of F_{r1} in S_1 and instances of F_c in S are obtained by concatenating instances of F_{c1} in S_1 and instances of F_{c2} in S_2. A is an ordered (multi) set of cell attributes obtained from A_1 and A_2 such that for each cell X in S, if X is in S_1 the cell attribute of X is the same as that in A_1, otherwise it is the same as that in A_2. If $F_{r1}=F_{r2}=\emptyset$ then $F_r=\emptyset$. The column concatenate $CC(S_1,S_2)$ is undefined if F_{c1} or F_{c2} is null.

Similarly, if F_{r1} and F_{r2} are not null and $F_{c1} \doteq F_{c2}$ then row concatenate summary tables S_1 and S_2, denoted

$RC(S_1,S_2)$

produces a summary table $S(F_r,F_c,A)$ where $F_r \doteq F_{r1} \circ F_{r2}$, $F_c \doteq F_{c1} \doteq F_{c2}$ and A is obtained as described above.

Extract Summary Tables: Let $S(F_r,F_c,A_c)$ be a summary table where F_r and F_c are row and column category attribute forests and A_c is the ordered set of cell attributes. Let T_r be a tree in F_r and T_c be a tree in F_c. Then

$EX_{(T_r,T_c)}(S)$

produces a summary table whose row and column category attribute forests are T_r and T_c respectively, and whose cell attributes and those attributes in A_c corresponding to T_r and T_c. Instances of T_r and T_c in S are also the instances of the row and the column category trees of in the resulting summary table. Every cell in S whose row and column category attributes are root-to-leaf paths in an instance of T_r and in an instance of T_c in S is also in the resulting summary table and the cell attribute value is the same as that of the corresponding cell in S.

The extract summary table operation is in a sense the inverse of concatenate summary table operation.

Attribute Splitting: Extended relational algebra operates on nested relations. Given a primitive summary table the corresponding nested relation can be obtained by operation REL as discussed before. Given a summary table whose row and column category attribute forests are ordered sets of chains, its primitive summary tables can be extracted by a sequence of extract operations. However, in order to decompose an arbitrary summary table into a number of primitive summary tables, operations which will transform row and column category forests into sets of chains are required. The following operations, row-split and column split, are defined for this purpose. These operators take the attribute to be splitted and the summary table as arguments, and produce a summary table whose row (or column) category attribute tree has the specified attribute splitted.

Attribute Merging: Given a nested relation, summary table formation operator produces the corresponding primitive summary table. Using concatenate operators, two or more primitive summary tables can be combined to obtain larger summary tables. However, category attribute forests of such summary tables always are some ordered sets of chains. In a category attribute forest of a summary table, suppose root attributes in two trees have the same instances. It may be desirable to merge these attributes into a single attribute. Similarly, when two attributes having the same parent in a tree also have the same instances it may be meaningful to merge them into a single attribute. The two operations, row-merge and column-merge are defined for this purpose. These operators take the attributes to be merged and the summary table as arguments, and produce a new summary table whose row (or column) attribute tree has the specified attributes merged.

6. CONCLUSION

In this paper, the relational algebra is extended for nested relations and for aggregate functions so that it can be used for summary table manipulation. Nested relations are used to represent a special case of summary tables, called primitive summary tables. Primitive summary tables are in turn used as building blocks of summary tables in general. A summary table manipulation language, based on the extended algebra of nested relations, is discussed. Basically, a query on summary tables is expressed in terms of (i)operations to transform summary tables into corresponding primitive summary tables, (ii)operations to transform these primitive summary tables into nested relations, and (iii)the extended algebra operations on these nested relations. The summary table manipulation language discussed in this paper will be implemented underneath another screen oriented, relational calculus-based language, called Summary-Table-By-Example [OzsO 82a]. Further investigations will be done in the direction of extending the relational calculus for aggregate functions [Klug 82] and nested relations, and showing that the extended algebra and the extended calculus of nested relations are equivalent in expressive power.

REFERENCES:

[Cham 76] Chamberlin, D.D. et. al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation,and Control," IBM J. Research and Development, Vol 20, No. 6, 1976.

[Cham 81] Chamberlin, D.D. et.al., "A History and Evaluation of System R," Comm. ACM Vol. 24, No. 10, 1981.

[Codd 72] Codd, E.F., "Relational Completeness of Database Sublanguages," in Database Systems, Courant Computer Science Symposia Series, Vol. 6, Englewood Cliffs, Prentice Hall, 1972.

[Jaes 82] Jaeschke, G. and Schek, H.J., "Remarks on the Algebra of NonFirst Normal Form Relations," Proc., ACM Symposium on Principles of Database Systems, 1982.

[Klug 82] Klug, A., "Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions," JACM, Vol. 29, No. 3, July 1982.

[OzsO 82a] Ozsoyoglu, Z.M., Ozsoyoglu, G., "STBE--A Database Query Language for Manipulating Summary Data," Technical Report CES-82-2, Computer Engineering and Science Dept., Case Western Reserve University, July 1982.

[OzsO 82b] Ozsoyoglu, G., Ozsoyoglu, Z.M., "SSDB--An Architecture for Statistical Databases," Technical Report, CES-82-11, Computer Engineering and Science Dept., Case Western Reserve University, October 1982.

[OzsO 83] Ozsoyoglu, Z.M., Ozsoyoglu, G., "An Extended Algebra of Nested Relations for Summary Tables," Technical Report, CES-83-3, Computer Engineering and Science Dept., Case Western Reserve University, March 1983.

[Ullm 82] Ullman, J.D., Principles of Database Systems, Second Edition, Computer Science Press, Potomac, MA 1982.

[Uslb 80] Bureau of Labor Statistics, "Table Producing Language System," Version 5, Washington, D.C., July 1980.

How Baroque Should a Statistical Database Management System Be?

Frank Olken

Computer Science and Mathematics Dept.
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

ABSTRACT

How elaborate should the functionality, data types, and data models of a statistical database management system be? In this paper we consider several criteria to be used in deciding the matter: efficiency, data semantics, standardization, integrity , security, and useability. On the basis of these criteria we construct a taxonomy of proposals for enhanced functionality of statistical database management systems.

1. Criteria for Extensions

The first generation of DBMS research was built around the relational data model paradigm. Recent years have seen a growing interest in more elaborate data models in many areas, e.g. entity-relationship models, aggregation operators, and new data types to support CAD/CAM systems. In this paper we are concerned with the question of how far designers of statistical database management systems (SDBMSs) should proceed in the direction of more elaborate (baroque) functions and data types in SDBMSs. Six basic types of criteria for judging the desirability of such enhancements will be considered:

(1) efficiency

(2) data semantics

(3) standardization

(4) integrity preservation.

(5) security

(6) useability

On each of these grounds it appears that substantially more elaborate database management systems are warranted than presently exist. In a few areas simplifications appear possible.

In certain respects this paper parallels that of Shoshani [Shos81]. However, the papers have different purposes. Shoshani's paper was intended as a tutorial, this paper is intended primarily as a taxonomy. Whereas Shoshani was heavily concerned with describing the characteristics of statistical databases and their problems, we are primarily concerned with constructing a taxonomy of the many possible enhancements which could be included in statistical database management systems according to the criteria by which they are justified. We have attempted to truncate the discussion of topics treated extensively by Shoshani, such as physical organization and security.

Throughout the paper we generally compare SDBMSs to commercial DBMSs (CDBMSs) rather than statistical analysis systems.

2. Efficiency

One obvious consideration is efficiency of the computation. Operations should be integrated into the SDBMS if they can thereby be more efficiently performed. There is a cost associated with moving data into and out of the SDBMS. Hence it is typically worthwhile to move computationally simple operations into the SDBMS.

In general operators which are data intensive, rather than computationally intensive, are plausible candidates for incorporation into the SDBMSs. Thus operators such as sorting or ranking may be appropriate to include in a SDBMS, while a computationally intensive iterative maximum likelihood procedure such as Cox regression would gain little in efficiency by inclusion within a SDBMS. Similarly, there seems little to be gained in terms of efficiency by including graphics capabilities within the SDBMS.

2.1. Sampling, Subsets and Versions

Various types of sampling operations (simple random sampling with and without replacement, stratified sampling, etc.) are clear candidates for inclusion into the SDBMS. It makes no sense to extract an entire set of data from a SDBMS in order to take a 1% sample.

It has been observed that statistical data analysis tends to generate numerous subsets or versions of the data as the analyst discards outliers, or focuses attention on "interesting" subsets. Decisions must be made as to how to store the various versions and subsets of the data. Redundant storage offers fast access to subsets at the potential cost of increased total storage requirements.

Issued as LBL report LBL-15765(rev.)

This work was supported by the Director, Office of Energy Research, Office of Basic Research Sciences, Division of Engineering, Mathematical and Geosciences of the U.S. Department of Energy under Contract DE-AC03-76SF00098.

Some systems (e.g. [Burn83]) have opted for "virtual subsets". Finding the useful subsets which might accelerate query response has been addressed in [Fink82].

2.2. Ordering and Aggregation

Traditionally, arithmetic operators and sorting have been provided in conventional database management systems (DBMS). The aggregation operators (MIN, MAX, SUM, COUNT, AVG) which have been included in conventional DBMSs exhibit simple computation structures. These aggregation operators distribute easily over set union operations. Thus, for example, the minimum of A union B is min(min(A),min(B)). Hence these "incremental" aggregation operators can be easily moved within the query computation strategy and lend themselves to parallel evaluation. If the number of groups for which aggregations are being computed is small (i.e., fits within memory) then all of the statistics can be computed in a single pass over the unordered raw data set. Other statistics (such as median) effectively require that we first sort the raw data by group and then process one group at a time. Klug [Klug82] has argued that integrating aggregation and output ordering during query evaluation can reduce costs.

Some authors [Swar83] have suggested that users be supplied with a simple programming language rather than a set of aggregation operators, citing better flexibility. While such a proposal has merits, we tend to favor the provision of at least a library of commonly used aggregation operators. One problem with user-written ad hoc aggregation operators, is that they may require non-trivial code analyses by the query optimizer to determine:

(1) opportunities to exploit parallelism or alternative orderings of the data.

(2) opportunities to infer some aggregate statistics from other previously computed statistics as in [Rowe81].

(3) procedures for maintaining integrity of aggregated data when underlying data is altered.

Certainly, some of the these problems can be eased by careful design of the user aggregation language.

2.3. Lag and Binning

Other related candidate operators for inclusion in SDBMSs on efficiency grounds include LAG operators which return the preceding element of sequences or time series, and BIN operators which convert continuous domains into discrete domains for histogramming and contingency table analyses [Iked81].

2.4. Abstract Data Types

Efficiency considerations may limit the utility of abstract data type mechanisms for extending SDBMSs [Ston82]. Too much hiding of the internal structure of abstract data types may preclude effective query optimization. It may be more useful to think of new data types as views, i.e., as constructions which are translated during query parsing into elementary data management operations.

2.5. Physical Organization

As discussed in [Shos81] and the references cited therein, SDBMSs lend themselves to data compression, array linearization access methods, and vertical partitioning of relations. The static nature of many statistical databases suggests that it may be practical to keep redundant, differently transposed copies of the data to improve the efficiency of retrievals.

2.6. Storage Hierarchy Management

Typical CDBMSs manage a two level storage hierarchy: disk and main memory. DBMS designers [Sacc82] have argued that the DBMS should control the buffering and caching of disk pages rather the operating system. Similarly, in distributed DBMS designs one typically wants the DBMS to have explicit knowledge of the locations of various relations (or copies) to facilitate query optimization.

In contrast to CDBMSs which typically manage two-level storage hierarchies, SDBMSs will often have three-level storage hierarchies. The tertiary store may be comprised of an automatic tape library, a magnetic mass store, or an optical disk juke box. We think it likely that the SDBMS will want to explicitly manage the movement of data between secondary (disk) and tertiary (e.g. tape) storage, rather than treating all secondary and tertiary storage as a giant black box to be managed by the operating system. We expect that information on the location of data within the storage hierarchy may be incorporated into the query optimization process in a manner akin to that used in distributed DBMS query optimization.

3. Data Semantics

Unannotated data is useless. Inadequately annotated data may be worse than useless, it may be misleading. Hence the ability of a database management system to adequately capture the semantics of the data it contains is a fundamental criterion for judging the acceptability of database management systems. Thus the specification and management of metadata has been a major research area in SDBMSs. We contend that such metadata should include statistically important semantic information.

3.1. Data Models

The classical relational data model is concerned with sets of records, which are in turn composed of fields. Other common data models (hierarchical and network) are also concerned with sets of records. These models are sometimes inadequate for statisticians, who are often concerned with sequences, time series (uniformly spaced sequences), and bags (i.e., samples which may contain multiple instances of the same value). The objects which comprise these sequences, time series and

bags are often vectors or matrices. Sets of records are not enough. Some statistical analysis systems (SAS [Helw79], EPS [Wan82], S [Beck81]) have begun to provide more elaborate data structures and data types. Typically they have weak data management facilities, e.g., requirements that the data structures fit into memory, weak query languages, and little or no query optimization.

3.2. Summary Data

Summary (or aggregated) data is commonplace in statistical databases [Shos81], [Gey81]. A considerable amount of work has been done in modelling this kind of data [John81b], [Krep82], etc. Such work has been motivated by three reasons. One is simply to adequately convey the meaning and inter-relationships of various datasets. Modelling summary data can also aid efficiency of query answering (as noted above), facilitate the framing of queries, and permit automatic maintenance of consistent derived data in the presence of updates (as discussed below). This has been one of the most intensively studied areas of statistical database management.

3.3. Statistical Models

For some statistical databases much of the data will be estimates of the values of parameters of statistical models, e.g. regression coefficients, distributional parameters, etc. For such data to be meaningful it is important that the corresponding statistical model be recorded in the SDBMS. In contrast to the recent attention given summary data (very simple nonparametric statistics), much less attention has be given to representing, indexing, retrieving or manipulating statistical models. An exception is EPS [Wan82]. Examples of statistical models range from simple fitted probability distributions to general nonlinear regression models. The estimated parameters of these models are useless without the distribution or terms of the regression model to which they refer.

3.4. Samples and Subsets

Earlier we discussed efficiency considerations of sampling and subsetting operations, here we are concerned with semantic aspects of specifying samples and subsets.

3.4.1. Sample Design

For some types of statistical analyses it is important to know whether the data constitutes the entire population, a simple random sample, or a more complex sample. Administrative records usually include the entire population of interest, scientific data are often simple random samples, while social science data are frequently the product of complex sample designs used to minimize the cost and sampling errors of the study. See [Coch77].

Some of the sample designs used are:

(1) stratified sampling

(2) multi-stage clustered sampling

(3) attribute based sampling

The sample design affects how certain summary statistics are calculated and the estimation of sampling errors. Hence the sample design needs to be recorded in the metadata in a manner accessible to both humans and programs. The PSALMS procedure in the OSIRIS IV statistical analysis package [Van79] has provisions for describing and analysing complex samples. However, OSIRIS does not provide any means of storing the sample design with the data. Few other statistical analysis or database management systems have any provisions for dealing with sample designs. The HODM system proposed in [Ozso82b] provides for specification that one dataset is a sample of another, but it is unclear how one goes about specifying the structure of a complex sample.

3.4.2. Subsets

Often the analyst will extract subsets of data from larger datasets for analysis [Thom81]. One would like to record the fact that one data set is a subset of another, and perhaps record the predicate (query) used to select the subset.

3.5. Graphics

We believe that it will be desirable to be able to store the *specifications* of graphs in the database. One reason for doing this is to provide standardized graphics to permit comparability. Examples include [Ozso82b] and [RTI83]. However, we do not believe it is necessary to integrate entire graphics *packages* into the SDBMS. They could instead be applications programs which call the SDBMS.

3.6. Level of Measurement

It is not enough to know that a data item is an integer. We also need to know the *level of measurement* of the data item, i.e., whether the integer represents a *nominal* (unordered categorical) variable, an *ordinal* (ordered categorical) variable, or an *interval* (metric) variable. The types of statistical operations which make sense depend on the level of measurement as well as the data type. It makes no more sense to compute the average religion of a population, than to compute the average name. Yet most data management systems will compute the average religion if religion has been coded as an integer. This HODM system proposed in [Ozso82b] includes mechanisms for specifying the level of measurement.

3.7. Units of Measurement

Similarly, it has been recognized that SDBMSs should record the units of measurement of the data and provide automatic conversion where necessary [Spar82],[McCa82b]. Clearly, if we want to compare two measures of length we need to take note whether they

are recorded in feet or meters. Pure unit conversions (e.g. inches to centimeters) convert between different measures of the same property. These conversions are invariant with respect to context.

However, the choice of preferred units may vary from application to application. In some (nonlinear) statistical models, the choice of units (i.e., scaling factors) may affect the conclusions, as in the case of multidimensional scaling or clustering. Hence it may be necessary to specify the units (scaling) for variables in statistical models, as well as data in the database.

Some unit conversions, such as force to mass (e.g. pounds to kilograms), or volume to density (e.g. bushels of wheat to tons of wheat or barrels of oil to tons of oil), vary with the context in which they occur. They are actually inferences of one property from another. Such inferences depend upon other properties of the entity, or assumptions about the context in which the conversion occurs. Thus converting the volume of oil to mass of oil depends on the type of oil, and the temperature of the oil. Currency conversions may depend on the time and place at which the conversion is presumed to occur.

Default procedures for such inferential conversions might plausibly be specified in the metadata description of an entity or perhaps inherited from a more general class of which the entity is an instance. Such default procedures could be explicitly overridden in the query specification if necessary. Conversion procedures might include parameters which could be bound during query evaluation, such as temperature.

3.8. Coordinate Systems

Another common type of data conversion is between different coordinate systems (e.g. from polar to Cartesian or between different frames of reference). In contrast to unit conversions, which are typically scalar operations, coordinate transformations are usually operations on vectors. Frames of reference may vary with the object referenced or with time. Coordinate conversions arise in physics, astronomy, and geodesy.

3.9. Humanly-Interpretable Metadata

Thus far we have argued for the necessity of including machine interpretable metadata to assure that only statistically meaningful operations are applied to the SDBMS. In addition one will often want to include information intended for human interpretation concerning the meaning of the data. Such annotations may include bibliographic citations concerning experimental techniques, the quality of the data, etc. While the SDBMS need only know that it should not add apples and oranges together, the user may want to know how to tell the difference.

In practical terms, this suggests that although the SDBMS will be primarily concerned with numeric data, provisions for managing text will be necessary. For example, if it is discovered that a particular experimental procedure or instrument produces unreliable results, one might wish to determine all of the datasets which use that procedure or instrument. Automatic propagation (or inheritance) of footnotes will be desirable, so that important caveats do not get lost during data analysis or aggregation.

4. Standardization

Statistical databases are often composed of data from diverse sources. The diversity of sources often results in a diversity of terms, codes, and formats for similar or even identical items. Inconsistent codes or formats may preclude comparability of data domains. Examples include inconsistent formats for specifying dates, and different sets of abbreviations for states. Hence it is important to provide mechanisms to facilitate the standardization of the data and the metadata which describes it.

Code and format conversions, like unit conversions, should be provided automatically by the SDBMS. Provisions need to be made in the metadata specification to specify the base (canonical) type (e.g. states) and mappings ("views") between variant and canonical sets of codes or formats. Type generalization and inheritance mechanisms [Smit77], [Sato81],[Krep82] and libraries of standard type definitions can be used to facilitate standardization.

5. Integrity

Assuring the integrity (i.e., correctness or at least consistency) of a database is important for both commercial and statistical DBMSs. Below we discuss some salient SDBMS integrity issues: derived data, concurrency control, and statistical input validation.

5.1. Derived Data

Much of the data in a SDBMS may be derived data - aggregations, descriptive statistics, model parameter estimates, etc. If one wants to maintain the consistency of the derived data with the underlying raw data across updates, then one will have to tell the SDBMS about the relationship of derived data to raw data [Koen81]. Such consistency is a kind of integrity constraint on the database. It seems clear that the responsibility of maintaining such consistency should be entrusted to the SDBMS, not to some external analysis package or to the user. Alternatively one can think of derived data as a type of view of the database. Preserving the consistency of the derived data is thus akin to updating views, i.e., one needs to store both some sort of definition of the view and updating procedures.

A variety of strategies have been proposed, some involving complete recomputation of the derived data (possibly on demand), and some involving incremental updating of derived data [Koen81]. Whereas integrity constraints are customarily specified individually for each data domain, it would be very useful if the "view specifications" of the derived data written in terms of generic statistical operators could be used to infer the

consistency maintenance procedures. As noted above, there has been considerable work on incorporating certain simple types of summary statistics into the data models (e.g. total sums and counts, averages, min, max). This work needs to be extended to encompass more complex parametric statistics. Thus, for example, one would want the SDBMS to know about covariance operators, and the appropriate procedures to update the covariance matrix when the underlying dataset was modified [Koen81], [Rowe82].

The point is that maintaining the consistency of derived data will require that the SDBMS know about statistical operators it would otherwise not need to understand. This is a strong argument for inclusion of a fairly elaborate statistical analysis capability within the SDBMS. Such a library of statistical analysis routines has been proposed for the HODM system in [Ozso82b] to permit security analyses of queries.

5.2. Concurrency Control

Many statistical databases are fairly static, e.g. updates may occur only monthly or more infrequently. Most such updates append rather than modify data. Furthermore, many statistical queries process a large portion of the records for a particular domain. This suggests that simpler concurrency control mechanisms which lock domains may be more appropriate than the fine grained record level locking found in some commercial database management systems.

5.3. Statistical Input Validation

It is commonplace in CDBMSs to enforce integrity constraints on the database by means of input validation procedures, which are invoked before updates to the database are permitted. Typically such validation procedures check that the input data are in allowable ranges, or from allowable code sets. Users of SDBMSs may also want to specify additional statistical input validation procedures (e.g., outlier detection) to be used on the input data. Thus if the data item is more than 2 standard deviations from the average for a domain, the data entry clerk might be warned. More elaborate statistical consistency checks using regression equations might be used on entire records to detect improbable combinations of height and weight, job and salary, etc. This would require some means of specifying such statistical constraints and maintaining the distributional and regression parameters.

6. Security

Often one wishes to permit statistical analyses of datasets while precluding access to individual records so as to protect the privacy of individuals. Common examples include medical and census records. This has been an area of intensive investigation. For overviews and bibliographies see [Denn83] and [Shos81]. Security concerns may have profound effects on the architecture of the data management system. As Ozsoyoglu [Ozso82b] and others have noted, the data management systems require considerable knowledge of the statistical procedures being used in order to prevent insecure inferences.

7. Useability

Ease of use is the last criterion we will consider for assessing SDBMS features. We examine both the user interface and the external interface to other software systems.

7.1. User Interfaces

Useability requirements for SDBMSs are in some ways more severe than for commercial DBMSs. Commercial DBMSs are typically used to process a few types of transactions by personnel who perform many of the same type of transactions daily. Such users tend to be familiar with the data and often the transactions can be compiled so that the user need only fill out an electronic form.

In contrast, the user of a SDBMS is often confronted by an enormous collection of data items (e.g. the U.S. Census) of which he wishes to extract a small portion. Just finding the name of the data he wants may be a major undertaking. Systems such as SUBJECT [Chan81] and SEEDIS [McCa82a] have been created to facilitate browsing.

Queries are often ad hoc and frequently involve aggregation. A great deal of work [Klug81a], [John81a], [John81b], [Ozso82a] has been done in providing query languages which support aggregation. It has been observed that richer semantic models may permit simpler, terser queries. The CABLE language [Shos78] represents an attempt to exploit an entity-relationship model to permit implicit specification of some joins.

The user may be a social scientist who is not interested in becoming a computer expert. Systems such as GUIDE [Wong81] have been created to provide simple ad hoc graphical query interfaces. Forms-based query languages for aggregative and summary tables queries have been proposed in [Klug81] and [Ozso82b].

7.2. External Interfaces

CDBMSs tend to provide a query language, a programming language interface, and perhaps a report writer. Loading and unloading databases and the attendant database conversion problems tend to be infrequent occurrences performed by professionals. In contrast the users of a SDBMS are often loading new databases into the SDBMS from a wide variety of sources. They may also need to move large amounts of data *and the attendant metadata* to and from statistical analysis, graphics or data management systems. SDBMSs need to provide tools which easily and efficiently permit users to do this. Designers of statistical analysis and graphics packages need to design reasonable interfaces. At present it is often necessary to convert numeric data to ASCII character representation to exchange with other systems, a

fairly expensive operation. Metadata conversion is even more clumsy.

8. Conclusions

We have proposed several criteria for evaluating proposed enhancements of SDBMSs: efficiency, capturing data semantics, standardization, integrity preservation, security, and useability. We believe that these considerations justify substantial expansion in the data types, data models, access methods, operators, metadata, metadata browsing mechanisms provided by SDMSs over those provided by typical present day commercial DBMSs. Existing and proposed SDBMSs include some of these features, none includes all.

We are not convinced that SDBMSs need to do everything imaginable. We believe that specialized functions, such as graphics, may be built as application modules which invoke the SDBMS. Furthermore, SDBMSs may not require as sophisticated concurrency control mechanisms as CDBMSs.

Much of what seems to distinguish the desired SDBMS from present day commercial DBMSs is the inclusion of much more intensionsal information (i.e., information specified by formula rather than enumeration). Examples include unit conversion procedures, input validation procedures, statistical models, predicates which define subsets, and definitions of derived data as aggregations or samples.

9. Acknowledgements

The author is indebted to his colleagues Arie Shoshani, Peter Kreps, John McCarthy, Harry K.T. Wong, Fred Gey, and Paula Hawthorn for their comments and encouragement. Special thanks are due John McCarthy for his extensive editorial assistance.

Bibliography

Beck81 Becker, Richard A., and Chambers, John M., *S, A Language and System for Data Analysis* , Bell Labs , Murray Hill, N.J. , 1981

Burn81 Burnett, Robert A. and Thomas, James J., "Data Management Support for Statistical Data Editing and Subset Selection" in *Proceedings of the First LBL Workshop on Statistical Database Management* , Lawrence Berkeley Lab , Berkeley , 1981 , pp. 88-102

Chan80 Chan, P., and Shoshani, A., "Subject: A Directory driven System for Organizing and Accessing Large Statistical Databases" in *Proceedings of the International Conference on Very Large Data Bases (VLDB)* , VLDB Endowment , Saratoga, Calif. , 1980 , pp. 553-563.

Coch77 Cochran, W.G., *Survey Sampling, 3rd ed.* , Wiley , 1977

Denn83 Denning, Dorothy E., "A Security Model for the Statistical Database Problem" in *Proceedings of the Second International Workshop on Statistical Database Management* , Lawrence Berkeley Lab , Berkeley , 1983

Fink82 Finkelstein, Sheldon, "Common Subexpression Analysis in Database Applications" in *Proceedings of the 1982 International Conference on the Management of Data* , ACM/SIGMOD , 1982 , pp. 235-245

Gey81 Gey, Fred, "Data Definition for Statistical Summary Data or Appearances Can Be Deceiving" in *Proceedings of the First LBL Workshop on Statistical Database Management* , Lawrence Berkeley Lab , Berkeley , 1981 , pp. 3-18

Helw79 Helwig, Jane T. and Council, Kathryn A., (ed.) *SAS User's Guide, 1979 Edition* , SAS Institute Inc. , Raleigh, N.C. , 1979

Iked81 Ikeda, Hideto, and Kobayashi, Yasuyuki, "Additonal Facilities of a Conventional DBMS to Support Interactive Statistical Analysis" in *Proceedings of the First LBL Workshop on Statistical Database Management* , Lawrence Berkeley Lab , Berkeley , 1981 , pp. 25-38

John81a Johnson, Rowland, "Modelling Summary Data" in *Proceedings of the 1981 International Conference on the Management of Data* , , 1981

John81b Johnson, Rowland, "A Data Model for Integrating Statistical Interpretations" in *Proceedings of the First LBL Workshop on Statistical Database Management* , Lawrence Berkeley Lab , Berkeley , 1981 , pp. 176-189

Klug81 Klug, A., "Abe -- A Query Language for Constructing Aggregates-by-example" in *Proceedings of the First LBL Workshop on Statistical Database Management* , Lawrence Berkeley Lab , Berkeley , Dec. 1981 , pp. 190-205

Klug82 Klug, Anthony, "Access Paths in the "Abe" Statistical Query Facility" in *Proceedings of the 1982 International Conference on the Management of Data* , ACM/SIGMOD , 1982 , pp. 161-173

Koen81 Koenig, Shaye and Paige, Robert, "A Transformational Framework for the Automatic Control of Derived Data" in *Proceedings 1981 International Conference on Very Large Databases* , VLDB Endowment , Saratoga, Calif. , 1981 , pp. 306-318

Krep82 Kreps, Peter, "A Semantic Core Model for Statistical and Scientific Databases" in *A LBL Perspective on Statistical Database Management* , LBL-15393 , Lawrence Berkeley Lab , Berkeley , 1982 , pp. 129-157

McCa82a McCarthy, John, *The SEEDIS Project: A Summary Overview of the Social, Economic, Environmental, Demographic Information System* , LBL-PUB-424 (rev.) , Lawrence Berkeley Lab , Berkeley

McCa82b McCarthy, John, "Metadata Management for Large Statistical Databases" in *Proceedings 1982 International Conference on Very Large Databases* , VLDB Endowment , Saratoga, Calif. , 1982 , pp. 234-243

Merr83 Merrill, Deane, et al., "Distributed Data Management in a Mini-computer Network: The SEEDIS Experience" in *Proceedings of the Second International Workshop on Statistical Database Management* , Lawrence Berkeley Lab , 1983

Ozso82a Ozsoyoglu, Z.M., and Ozsoyoglu, G., "STBE -- A Database Query Language for Manipulating Summary Data" , Technical Report CES-82-2 , Computer Engineering and Science Dept., Case Western Reserve Univ. , July 1982

Ozso82a Ozsoyoglu, Z.M., and Ozsoyoglu, G., "SSDB -- An Architecture for Statistical Databases" , Technical Report CES-82-11 , Computer Engineering and Science Dept., Case Western Reserve Univ. , Oct. 1982

Rowe82 Rowe, Neil C., "Rule-base Statistical Calculations on a "Database Abstract"" in *Proceedings of the First LBL Workshop on Statistical Database Management* , Lawrence Berkeley Lab , 1982 , pp. 163-175

RTI83 RTI, *INGRES GBF (Graph by Forms) User's Guide* , Relational Technology Inc. , Berkeley , 1983

Sacc82 Sacco, G.M., and Schkolnick, M., "A Mechanism for Managing the Buffer Pool in a Relational Database System Using the Hot Set Model" in *Proceedings of the International Conference on Very Large Data Bases (VLDB)* , VLDB Endowment , Saratoga, Calif. , 1982

Sato81 Sato, Hideto and Hotaka, Ryosuke, "For Large Meta Information of National Integrated Statistics" in *Proceedings of the First LBL Workshop on Statistical Database Management* , Lawrence Berkeley Lab , 1982 , pp. 206-223

Shos78 Shoshani, Arie, "CABLE: A Language Based on the Entity-Relationship Model" in *Proceedings of the International Conference on Databases, Improving Usability and Responsiveness* , Jerusalem , also as UCID-8004 , Lawrence Berkeley Lab , August 1978

Smit77 Smith, J.M., and D.C.P. Smith, "Database Abstractions: Aggregations and Generalization" , *ACM Transactions on Database Systems*, vol. 2, no. 2 , pp. 105-133

Spar82 Sparr, Ted M., "Units and Accuracy in Statistical Databases" in *Proceedings of the First LBL Workshop on Statistical Database Management* , Lawrence Berkeley Lab , 1982 , pp. 59-60

Ston82 Stonebraker, Michael, "Application of Artificial Intelligence Techniques to Database Systems" , UCB/ERL M82/31 , Electronics Research Lab, U.C. Berkeley , Berkeley , May 1982

Swar83 Swartout, Don, "How Far Should a Database System Go? (to Support a Statistical One)" in *Proceedings of the Second International Workshop on Statistical Database Management* , Lawrence Berkeley Lab , 1983

Thom81 Thomas, James, et al., "Data Editing on Large Data Sets" in *Proceedings of the 13th Symposium on the Interface: Computers and Statistics* William F. Eddy (ed.) , Springer-Verlag , 1981

Van79 Van Eck, Neal A., et al., *OSIRIS IV User's Manual, 4th edition* , Institute for Social Research, Univ. of Michigan , Ann Arbor, Michigan , May 1979

Wank82 Wanka, Jane, et al., *EPS Plus Resources* , Data Resources Inc. , 1982

Wong82 Wong, H.K.T., and Kuo, I., "A Graphical User Interface for Database Exploration" in *Proceedings of the International Conference on Very Large Data Bases (VLDB)* , VLDB Endowment , Saratoga, Calif. , 1982

How Far Should a Database System Go? (to Support a Statistical One)

Don Swartwout

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

This note discusses some issues that were addressed in adapting a prototype database management system to enable it to supply data to a statistical system. The issues include the choice of statistically-oriented features for the database system, the "right" way to do aggregate computations, and some side effects of decisions that were made in this case.

In mid-1981 I was working on an experimental database management system intended to support distributed query processing in the UNIX* operating system. The database system, known as Datastream, was about six months old and working in rough prototype form, but the high-speed network and distributed database applications that would have been required to give it a serious test had not materialized as expected. However, several non-distributed, predominantly statistical applications were available, and I decided to try Datastream on their data. About the same time I met John Chambers, an originator of the S system for data analysis [1]. Discussions with him quickly made it clear that both systems would benefit from an interface that would permit Datastream to extract interesting data sets from a large database and pass them to S in a convenient way. Details of the physical transfer of data and control between the two systems were easily arranged; several more-or-less equivalent schemes have been implemented since then. Datastream's repertoire of functions was not adequate to support its new users, though, so a variety of enhancements were made. This note outlines the issues that had to be addressed as Datastream was modified to support statistically-oriented queries. The discussion covers two facilities, both present in S, that were added to Datastream, and several others that were not. Some impressions of what makes a feature appropriate for use in the database system are given.

A few words of background information on data analysis in a UNIX environment are in order. Large scale statistical databases seldom exist in such an environment. I take "large scale" to mean something like a hundred megabytes or more. Data analysts working in UNIX systems typically negotiate for data from other sources. A common arrangement involves a member of a data processing staff, who writes and runs an ad-hoc program to extract and re-format a subset of some database. The output is put on a magnetic tape, which is read and re-formated again on the UNIX system. If enough data is involved, or if the data is sufficiently complex, it may be stored in a UNIX database constructed from scratch to support the analysis. The analyst usually has complete control of the data after it arrives from the remote source; if he or she is not satisfied with the way it is structured, a new structure can be tried. The volumes of data involved are seldom large enough to seriously impede the analyst's ability to restructure it. For example, S normally supports data sets of up to a megabyte effectively; Datastream has been used to store databases of up to fifteen megabytes.

Datastream supports a variant of the entity-relationship data model [2]. The query language is a blend of ideas from UNIX programming and relational database languages. By comparison with most relational query languages, it has a procedural flavor. For example, the following Datastream query† selects certain rows from a table of data on telephone customers and prints the values of certain attributes.

```
get each Customer
    such that num_calls > 17
        print num_calls, most_recent_bill ;
```

The keyword **print** introduces a list of expressions whose values are computed, formated appropriately and "printed" as a stream of bytes that can be saved in a file, displayed on a terminal or line printer, or passed as input to another program (S, for example).

The first change made to support analytical data was the introduction of conditional expressions. A conditional expression consists of a condition and two expressions. If the condition is true, then the first expression is evaluated and its value becomes the value of the conditional expression. If the condition is false, the value of the other expression is used. Such expressions have many applications in statistical queries. For example, the following example converts arbitrarily-designated numbers into intelligible strings:

* UNIX is a trademark of Bell Laboratories

† For simplicity in this note, Datastream syntax will be modified slightly, and examples will deal only with single entities (i.e. flat files).

```
get each Customer
   print if ( sex = 0 ) then "MALE" else "FEMALE" ;
```

Datastream is written in C [3], and it uses native arithmetic and comparisons. This makes arithmetic fast and easy to program, but it exposes the query writer to problems such as division by zero. Conditional expressions allow the query writer to guard against such operations, since only the selected expression is actually evaluated. For example, to compute customers' average length of telephone calls, in the presence of customers who made no calls, one can write:

```
get each Customer
   print
      if ( num_calls > 0 )
      then total_time / num_calls
      else 0 ;
```

The second enhancement was on a larger scale: the addition of support for aggregate computations. When it was conceived as a system for distributed query processing, Datastream did not appear to need more than the ability to do simple arithmetic on the values in a single record (e.g. conversion from English to metric units). However, aggregate operations such as sums, counts, and so on proved to be necessary to support statistical applications. The principal design decision involved was that Datastream would *not* support specific aggregate functions. Instead, it would have a general facility for iterative computations that would support the usual aggregate functions and a wide variety of others as well. For example, the following counts customers and finds the average, maximum, and minimum number of calls.

```
initialize count = 0, total_calls = 0,
   max_calls = 0, min_calls = 0;

collect each Customer
   count = count + 1,
   total_calls = total_calls + Customer.num_calls,
   max_calls =
      if (Customer.num_calls > max_calls or count = 1)
      then Customer.num_calls
      else max_calls,
   min_calls =
      if (Customer.num_calls < min_calls or count = 1)
      then Customer.num_calls
      else min_calls,
then
   print count, total_calls / count, max_calls, min_calls ;
```

The first line of the query consists of initializations. It is executed once. Then the list of assignments between **collect** and **then** is executed for each row in the Customer table. Finally, the **print** operation is performed.

A common impression of this example is that the query writer has considerably more to say than he or she would if the language supported such expressions as

```
count(Customer), mean(Customer.calls), ...
```

Furthermore, the query has a strong programming flavor. The user must understand the initialization and the programming-language-style assignments, and so on.

Both of those impressions are true, but it is also true that a large percentage of the data analysts Datastream supports are experienced (if not always sophisticated) programmers. They seldom have trouble with the computational portions of the programs they write; input/output causes most of the problems for data analysts who write programs. That is, the programming aspects of writing Datastream queries are mostly limited to a set of constructs that analytical users find easy enough to handle. Users have not objected to the size of their queries, either; I object to voluminous queries more than they do. Furthermore, an analysis often involves running a succession of queries that differ only in small respects such as the values of certain constants or the structure of certain conditions. Query sequences like this are usually created by writing the first from scratch and then editing a little to produce the others.

The most important advantage of a general facility as opposed to built-in functions is that it provides a considerable increase in computational power without a major increase in the basic constructs of the language. Datastream's aggregate computation facility required two new types of control flow: initialization statements and the **collect ... then** style of iteration. No new operators, expressions, or other notations were required. As it turned out, it was necessary to overhaul the system's internal mechanism for handling variables, but this was invisible to query writers. In the following query, initialization and **collect ... then** are used to compute a stratified sum.

```
initialize morning_calls = 0,
   afternoon_calls = 0, night_calls = 0;

collect each Call
   morning_calls = morning_calls
      + if ( Call.start_time < 1200 ) then 1 else 0,
   afternoon_calls = afternoon_calls
      + if ( Call.start_time >= 1200 and
         Call.start_time < 1800 ) then 1 else 0,
   night_calls = night_calls
      + if ( Call.start_time >= 1800 ) then 1 else 0
then
   print morning_calls, afternoon_calls, night_calls ;
```

Strictly speaking, both the conditional expressions and the **collect ... then** feature duplicate facilities available in S. The architecture of S at the time made it impossible to use the existing software, however. Before S can operate on some data, it must first be set up as an S data set (i.e. a disk file in a certain format). But this makes it impossible to use the computation facilities of S on anything that is too big to be reasonably representable as a data set, and Datastream databases exist precisely because they are too big to be stored as S data sets.

How far should such duplication of function go before it is stopped in the name of cleanliness and modularity? Perversely, the duplication I have discussed exists because the systems interact across a fairly clean interface. S regards Datastream as one of many ways to build a data set; Datastream sees S as one of many things that can be done with the output of a query. Neither system meddles in the other's business. This requires the database system to replicate enough of the functions of the statistical system to permit users to describe interesting subsets of databases. Unless the database and statistical functions are so tightly coupled that it is difficult to describe them as separate

systems, one should expect to find most of the statistical system's standard selection facilities duplicated one way or another in the database system. From this point of view, one would not expect to find regression functions in the database system; data analysts seldom use regression to select a small subset of a larger data set.

Another S facility not duplicated in Datastream was general looping. Datastream's **collect** construct sets up loops based on structures in the database. One can sum the values of some field in all the records of a certain type, but one cannot write a loop of the form

```
for i in 1 .. n
do
   something involving i and
   the fields of a Customer record
end
```

Several types of general loops are available in S, but they have not been implemented in Datastream for two primary reasons.

1. Most users do not seem to need them.
2. If general loops were available, serious users would probably need a debugger.

Data and control flow in Datastream queries are fairly transparent. No one has written a query whose behavior could not be understood by reasonable "desk checking". A general loop facility would probably lead to complex queries whose behavior could not be deciphered without a debugger.

Syntax is an area in which little was done but much might have been. The first user to explore a database must deal with at least two distinct and largely incompatible languages: the Datastream query language and the S command language. In addition, the program which constructs databases has to be given instructions, in yet another language. The only thing that makes this arrangement palatable is the following hierarchy.

1. The typical data analyst spends much more time working in S than in Datastream.
2. The typical Datastream user spends much more time working with queries than with build instructions.

Of course, queries written and saved in an appropriate place make it possible for exploratory users to protect some of those who come later from dealing directly with the database system.

The modifications I have described were intended to make Datastream more useful as a tool for constructing S data sets. As it turned out, some users were able to accomplish their analysis without using S at all. These were not elaborate analyses; rather, they were excursions intended to discover what kinds of information could be extracted from certain databases. For this type of analytical work the principal advantages of Datastream were its ability to process queries against large files efficiently and its ability to traverse hierarchically-structured data while doing straightforward computations.

In summary, the following is a (non-exhaustive) list of my conclusions from experience adjusting a database system to serve statistical users.

1. Conditional expressions are well worth having.
2. Aggregate computations including but not limited to sums, counts, and extrema are indispensable, but they do not have to be offered as "canned" sum, count, and extrema functions. A general aggregate computation facility is more powerful than a typical set of specific functions. It may also be less trouble to implement and not much more trouble to use.
3. A database system's query facilities should not grow so powerful that they require a debugger. Database queries can be regarded as programs in special-purpose languages; query languages that need debuggers have probably ceased to be special-purpose languages.
4. A uniform interface to both the statistical and database facilities sounds wonderful. Unfortunately, it may prove too good to be true if one is dealing with existing systems.
5. A database system that can feed useful data to a statistical system and stand by itself as well must inevitably duplicate some of the basic arithmetic and selection functions of the statistical system. When this happens, some users may be able to work entirely within the database system.

References.

[1] Chambers, J. and Becker, R., "S: A Language and System for Data Analysis", Computing Information Service, Bell Laboratories, Murray Hill, NJ.

[2] Chen, P., "The Entity-Relationship Model -- Toward a Unified View of Data," *ACM Transactions on Database Systems 1*, 1(1976), 9-36.

[3] Kernighan, B. and Ritchie, D., *The C Programming Language*, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

AN INTEGRATED MACRO-ECONOMIC DATA MANAGEMENT SYSTEM BASED ON MULTI-DIMENSIONAL ARRAYS

M. GIBBONS and M.DAVID
OECD, France

Abstract: The paper describes the system currently being developed at the OECD for the management of macro-economic data. The system provides a general framework for economists and statisticians to structure and manipulate large volumes of data in the form of multi-dimensional arrays. The dimensions of the arrays are hierarchical lists of elements corresponding to criteria of classification, TIME typically being one of them. Derivation of data can occur either automatically during extraction or explicitly through a flexible Data Manipulation Language. The system fully incorporates the management of qualitative information associated with the data at several levels. A number of interfaces are provided including an extension to the Research and Analysis Language (RAL).

1. INTRODUCTION.

The Organisation for Economic Cooperation and Development (OECD) objective to promote policies for sustained economic growth and expansion of the world trade entails the collection, on a regular basis, of data from its 24 member countries, for analysis by statisticians and economists. As a contribution to this on-going effort, the Systems Development Division of this Organisation is implementing a generalised database management system, designed for the computerised storage and manipulation of macro-economic statistical data.(1)

The system has been conceived to meet three equally important requirements:

-- to provide the producer of statistical information with all the tools necessary to process the data from receipt to publication;
-- to provide analysts with the means to explore relationships between the statistics which surpasses that possible with a purely time-series oriented approach to the classification and storage of data;
-- to incorporate the management of qualitative information, so that users can be fully aware of the important characteristics of the data.

This paper describes the structural concepts, the main features of the Data Manipulation Language and the functional implementation of the system. Particular emphasis is placed on the facilities for treating qualitative information.

---

(1) Basic ideas for the system are borrowed from ISIS, developed by the Austrian Statistical Office.

2. STRUCTURAL CONCEPTS.

Statistical data are organised as sets of arrays within SECTORS which correspond to broad categories of economic interest, such as foreign trade or national accounts.

A set of structurally homogeneous arrays is known as a SEGMENT. Each dimension of an array is defined by a CRITERION OF CLASSIFICATION, which is a list of ELEMENTS. An element of a criterion may itself be a list of elements known as a SUB-CRITERION, providing the possibility of tree-structured or COMPLEX criterion. TIME is a special criterion whose elements depend upon the frequency of observations.

Each segment consists of one or more base arrays, depending on the existence of complex criteria within the segment definition, and of all the arrays implicitly derived from the base arrays by reduction over one or more dimensions. For a segment defined with N simple criteria, there exists one base array and

$$\sum_{i=1..N} N! \,/\, i!\,(N-i)!$$

derived arrays, counting the scalar generated by total reduction.

In the segment definition, criteria may be PARALLEL where an alternative classification of the same statistical item is available, or OBLIGATORY when no reduction over the criterion is possible. To promote the use of standardised nomenclatures, criteria can be declared as GLOBAL and are then available to all sectors for segment definition. Global criteria are maintained by the database administrator.

To clarify these concepts, the following example for foreign trade data is given:

SEGMENT definition:

```
FOREIGN-TRADE
Frequency: Annual
Criteria:  OECD,
           PARTNERS,
           UNITS obligatory,
           SITC,
           ISIC parallel to SITC,
           DIRECTION.
```

CRITERION definition:

```
OECD: Member Countries (GLOBAL). Elements: USA, CANADA,... (Size=24)
PARTNERS: Trade partners. Elements: ... (Size=200)
UNITS: Elements: $US, TONS. (Size=2)
SITC: Classification of products. Complex with 5 levels. (Size=2665)
ISIC: Alternative classification of products. (Size=192)
DIRECTION: Elements: IMPORTS (factor=-1), EXPORTS. (Size=2)
```

3. DATA RETRIEVAL.

The retrieval of data for display, manipulation or reporting, is based on the data-specification construct which, in a concise way, permits the selection of the level of data-aggregation desired, the order of the dimensions and elements within dimensions.

Derivation by reduction can occur automatically at extraction time, depending on the data-specification entered by the user. If one or more criteria are omitted from the data-specification, reduction is performed on the corresponding dimensions. During this process, pre-specified factors are applied to the elements of the dimension being reduced.

Example: FOREIGN-TRADE UNITS 1 OECD [USA CANADA] PARTNERS 1 TO 100
ISIC 1 2(1 TO 5 *) TIME 78 TO 83

where ISIC 1 2(1 TO 5 *) stands for the first product of the ISIC classification and then the five first sub-products of the second product, followed by their sum.

This specification corresponds to a 2 X 100 X 7 X 6 array (i.e. a logical group of 1400 time-series of 6 observations each). It is obtained by reduction over the criterion DIRECTION. With a factor of (-1) for IMPORTS and a default factor of (+1) for EXPORTS, the reduction yields the trade balance.

Automatic frequency conversion (e.g. from monthly to yearly data by averaging) may also occur during the extraction process, depending on the frequency of the data stored and the frequency requested.

Once extracted, data are available for manipulation in what is called the current-array. Data from other parts of the database can be further extracted and combined with, or appended to, the current data. The current-array is itself a component of a broader concept, called the current environment. Among the other components of this environment are the trees of identifiers and titles associated with each dimension; structural information and documentation; and a set of arrays of note-attachments.

4. QUALITATIVE INFORMATION.

Two kinds of qualitative information are considered: the structural documentation, describing the features of the structures (SECTOR, SEGMENT, CRITERION or ELEMENT) and the notes, providing background information about the quality of the data.

Notes are pieces of textual information. They are given an identifier and can be attached to, or detached from, data-cells in the database (permanent attachments) or in the current-array (temporary attachments), at all possible levels (individual cell, group of cells, array or segment).

In the case of database cells, an extended form of the data-specification construct is used to describe the scope of the attachment, where an infinite TIME dimension is allowed. In the case of current-array cells, a region-specification is used. A current-array region can be physical, i.e. expressed in terms of the physical positions of the cells, or logical, i.e. expressed in terms of a logical condition to be met by the cells.

Example of physical region: ROWS 1 TO 5 COLS 3 7 9 TO 16
Example of logical region: (VALUE > 0) AND (ESTIMATED)

Estimated values, ruptures within time-series and computed totals involving missing values are but a few examples of the data characteristics which can be highlighted by using the notes facilities. At data extraction time, note-attachments are optionally extracted and transferred to the current environment where they can be displayed or updated with the data-values.

During computation or derivation of data, new note-attachments can be generated according to a default or user-specified logic. Possible options are: OR, AND or no-logic. For example, with the OR logic, an estimated value combined with a non-estimated value yields an estimated value; with the AND logic the result is a non-estimated value; with the no-logic option, the note-attachments associated with the operands are ignored.

A few notes are global. Their texts are pre-defined by the Database Administrator and can be used by all users of the database.

5. THE DATA MANIPULATION LANGUAGE (DML).

Typically, the first task performed by an interactive database user is to establish a current environment by extracting both quantitative and qualitative information from the database. This can be done by using the DML statement GET. The data-specification for the GET always refer to a group of data within a single segment of the database. The environment can be completed with data from other segments by using the EXTEND statement. Extension occurs along one specified dimension of the current-array, all the other dimensions being compatible in size with the matching dimensions of the extension array.

The other array-manipulation statements do not involve the database, but instead process the current environment directly. DELETE is used to remove parts of the array; TRANSPOSE is used to modify the order of the dimensions; PERMUTE is used to change the order of elements within the dimensions; MAP is used to modify the shape of the current-array by combining dimensions, e.g. to transform a 6 X 2 X 4 array into a 6 X 8 array. During the MAP function, the title trees of two combined dimensions are merged by inserting the tree corresponding to the second combined dimension at each leaf of the tree for the first combined dimension.

The DISPLAY and UPDATE statements are used to produce a display on the terminal of a portion of the current environment, according to default or user-specified format parameters. The data-array is partitionned by the system into two-dimensional pages. The user can control at any point in time which page will be displayed next by indicating the starting element for each dimension. In update mode, the data-values or note-attachments can be modified in the current environment. The changes become permanent (i.e. they are reflected in the database) if a STORE statement is issued subsequently.

Computational facilities are available through the COMPUTE statement and include simple arithmetic operations and functions such as growth rates, moving averages, etc...

Sample DML session:

```
GET <data-specification>
EXTEND ROWS <data-specification 2>
COMPUTE ROW 10 = SUM (ROWS 1 TO 9)
DELETE ROWS 2 TO 5
DISPLAY
ATTACH ANOTE TO (VALUE=MISSING) AND (ROWS 10 TO 15)
PRINT USING <format>
(...)
```

6. FUNCTIONAL IMPLEMENTATION.

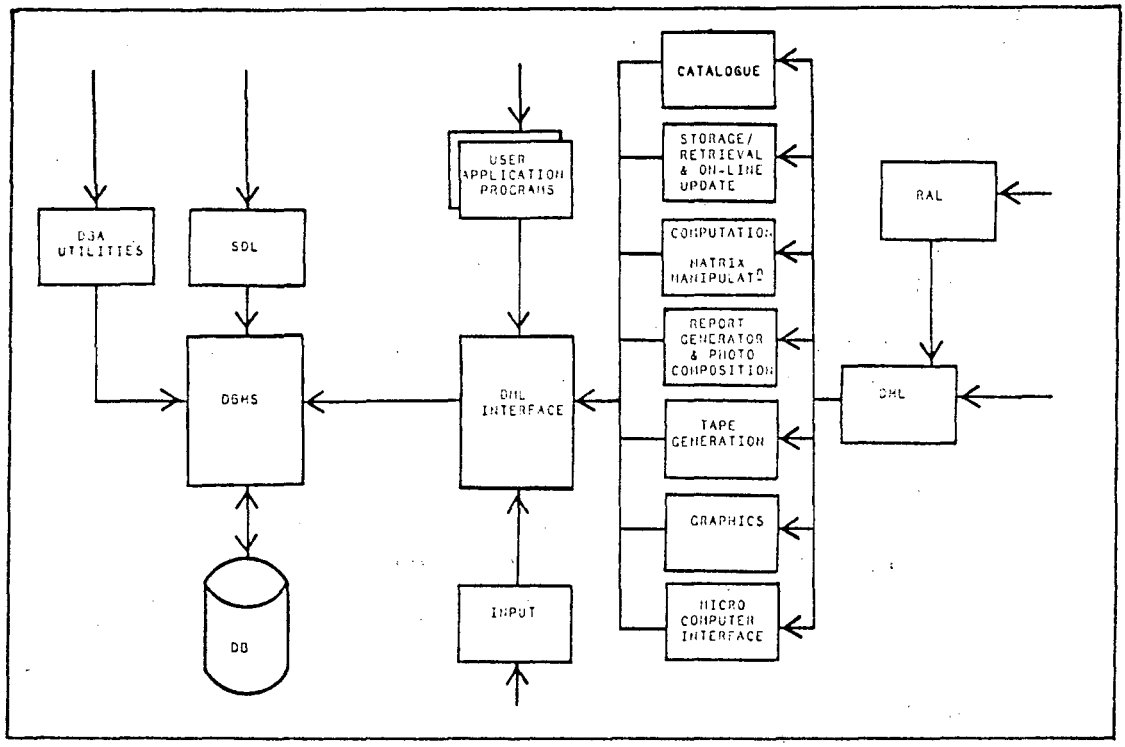
The system is built on top of a CODASYL-type DBMS (Burroughs DMS-II) providing efficient storage/retrieval techniques and a first level of data compaction and access-right control. Logical views are created via the Structure Definition Language (SDL) processor. A unified DML interface, in the form of a library of procedures which can be dynamically called, handles all requests for data manipulation. It establishes a clear boundary between the DBMS structures and the logical structures as perceived by the users.

The interactive Data Manipulation Language processor itself consists of a driver and a set of modules. It acts as an interpreter. Commands can be issued one at a time or programs can be written in DML and executed either directly or from within other DML programs. The modules are shown in Table 1. Typically, they process the current environment so that they can be chained, each function building up on the previous ones. In addition to the facilities discussed above, they include:

-- A Report Generator allowing for very flexible table layouts. Variable data-value formats are specified by using the region specification construct.
-- An interface with photo-composition utilities, making the production of publications from the database a highly automatic process.
-- A graphics interface.

-- A facility to produce magnetic tapes for outside clients. The tape may include qualitative information, with automatically generated note references.
-- A link to micro-computer software, in particular a facility to generate files compatible with MULTIPLAN (from Microsoft Corp.).
-- A catalogue enabling the user to browse through the database logical structures using keywords.

Table 1. FUNCTIONAL IMPLEMENTATION DIAGRAM.



The DML processor can be accessed from the Research and Analysis Language (RAL) developed by the Federal Reserve Bank of New-York. Control can be switched back and forth from RAL to DML and the transfer of data is possible in both directions. As a result, the RAL statistical routines are readily available to database users and all the DML facilities can be used to process RAL data.

Apart from the DML modules, the DML Interface can be called from user-application programs, written in ALGOL, FORTRAN or COBOL (e.g. special-purpose simulation or model-solving programs) and from the bulk input utility, the purpose of which is to provide a generalised entry point for large volumes of data originating from other systems.

6. Logical models, Metadata, and Data Transformation

See Also. . . .

CLASSIFICATION OF METADATA

Dr. Yvonne M. Bishop and Dr. Stanley R. Freedman

Office of Statistical Standards, Energy Information Administration
Department of Energy, Washington, D.C.

Abstract

The Energy Information Administration (EIA) has developed a search tool, the Data Resources Directory, that tracks the energy information obtained on source data collection forms to its final publication. The metadata is indexed according to a specially developed hierarchical classification scheme. Experience so far has indicated good features of this approach but problems still remain to be solved, particularly in determining optimum classification strategies for metadata.

The Department of Energy collects information about energy by means of over 200 surveys of industry participants. This information is disseminated by means of weekly, monthly, quarterly and annual periodicals and forms the basis for analytic reports which project the likely future trend under differing scenarios. A Data Resources Directory has been developed to keep track of the available information. The original plan was to implement the DRD system by system and eventually have a mechanism whereby each data element on a form could be traced through all intermediate steps to the final publication, and its numerical value identified. Some subsystems of the DRD are in place and experience so far has raised a number of issues:

- Hierarchical classification of data elements is time consuming and costly.
- The Federal Energy Data Index (FEDEX), the publications subsystem currently in place, uses an Energy Data Base (EDB) thesaurus for indexing, which is different from the classification scheme used in the DRD.
- An unambiguous vocabulary, such as used in the DRD, requires many synonym linkages if it is to be a useful search tool for a specialist in a specific industry.

DRD Subsystems in Place

The subsystems currently in place are derived mainly from data collection forms with the notable exception of the Public Use Energy Statistics Data Base (PUESDB). Others are in the process of being implemented. Those in place are as follows.

Forms/Frames Subsystem

The forms/frames subsystem contains metadata on DOE data collection forms and the lists and survey frames that are used to identify respondents and draw samples. Seventy-five metadata attributes about each form are maintained. They can be divided into four categories: identification attributes, management attributes, linkage attributes, and frame attributes. Identification attributes include form number, title, prior form numbers, indexing terms used to describe the form and an abstract. Management attributes include metadata used in EIA's forms clearance activities such as average burden per response, expiration data, voluntary mandatory reporting requirements and similar types of information. Sponsoring agency of the data collection, forms manager's name and phone number, computer system processing the data and publications in which the data appear are considered linkage attributes. Finally, type of respondent and their SIC code, source, size and name of frame, sample methodology

type and frame update frequency are included under the category of frames metadata.

User oriented products of the form/frames subsystem are:

- o on line file of data collection forms searchable by metadata attribute;
- o on line and hardcopy display of forms and their attributes;
- o Photocomposed EIA Directory of Energy Data Collection Forms; and
- o Periodic reports for forms clearance activities.

Glossary Subsystem

The glossary is an inventory of energy terms in use by EIA. The major source of input are active data collection forms, selected major publications such as the Monthly Energy Review and existing standard definitions. For each technical term (such as jet fuel, lignite coal or average dealer margin) the definition, and source of the definition are identified. The glossary is used in EIA as a tool to achieve standardization of energy terminology, to eliminate duplicate and overlapping terms and to respond to inquiries from the public. Outside EIA, the glossary is particularly helpful to users for locating the precise definition of an energy statistic. Products available through the glossary of energy terms include:

- o On line searching and retrieval of terms by energy subjects and source;
- o On line display of definitions and source by subject and source; and
- o Hardcopy glossary reports.

Data Element Subsystem

The largest component of the DRD is the data element subsystem. This subsystem is a description, inventory and index to statistical energy data elements on active data collection forms. A data element is a block or blank on a form. Each data element is assigned a unique serial number and contains five additional attributes. These are: (1) the form number on which the data element is found; (2) the physical location of the data element on the form; (3) the index terms used to catalog the data element; (4) the description of the data element using actual wording from the form; and (5) notes which contain elaboration or clarification information about that data element. Products from the data element subsystem include:

- o On line searching of data elements by subject categories, source and source descriptions;
- o On line display of data elements and their descriptions;
- o Hardcopy classified catalog of data elements;
- o Linkage of data elements to their source.

As with other DRD components, the data element descriptions are linked to the other subsystems in an integrated data base.

Public Use Energy Statistics Data Base (PUESDB)

Over 450 time series taken from major publications such as the Annual Report to Congress and the Monthly Energy Review are contained in the PUESDB. These series are accessible through retrieval and display screens using the DRD software. The source publication and form number for each series is stored as well as its detailed documentation. This provides linkages between the time series and the metadata subsystems. The PUESDB is also available for sale to the public through the National Technical Information Service.

DRD Software

The DRD is made up of three software systems: online information retrieval and display; hardcopy reports; and file maintenance. Each of these systems operates on the metadata base which is stored in the ADABAS data base management system. The online information retrieval and display system allows the user to search the data base to locate information, display information retrieved, and scan alphabetically and hierarchically the indexing terms in the vocabulary. When searching the data base, two subsystems can be linked or coupled together. This allows the user to first search one subsystem for information and then locate related information in a second subsystem without formulating a new search query. The metadata base can be searched using not only the indexing terms, but also the other metadata attributes. No special programming knowledge is required to use the software. Each of the subsystems has associated with it predefined display formats that present subsets of metadata in logically related groups. One format displays all metadata attributes for each subsystem. The online information retrieval and display was designed for IBM 3270-type intelligent terminals, and operates in a full-screen form fill-in mode. Each function -- search, display and scan of the indexing term file -- are presented on one screen. The user then provides search or display criteria, edits the screen if necessary and finally sends that function to the computer for processing. Hardcopy reports are requested from the DRD online and are then processed in batch. There are 30 reports available to users, most of which are used as part of EIA's forms clearance activities. These reports provide the user some flexibility in terms of search criteria and satisfy the bulk of user needs. More specialized reports can be generated using the NATURAL programming language which was developed for use with ADABAS.

Finally, specialized maintenance software for the DRD metadata base has been developed. As with the retrieval and display system, the maintenance system also operates online with full screen display and text editing.

Issues Needing Resolution

All of the data elements, forms and glossary terms in the DRD are indexed and described using a highly structured, hierarchical vocabulary which is organized into eight broad categories or facets. These are:

- Energy Source--the subject matter or product being measured such as coal, unleaded motor gasoline or electricity;
- Source Qualifiers--terms used to further define energy sources such as sulphur content range, imported or finished;
- Energy Function--the stage of the energy production life cycle and the management of energy firms such as mining, refinery processing, purchases or consumption;
- Assets--equipment or resources such as drills, fields, coke ovens or tankers;
- Participants/Facilities--both the physical and corporate facilities and agents such as subsidiaries, refineries, suppliers or shippers;
- Location--the geographic, administrative, political, geological or topological place terms used to describe energy data such as United States, DOE Region 1, OPEC, basin or offshore;
- Frequency--periodicity of the data such as monthly, quarterly or daily; and

- Measurement--the physical and accounting units that describe the data such as barrels, BTUs, short tons, feet or dollars.

This form of indexing using a highly structured hierarchical scheme ensures that each data element is uniquely defined. The searcher can be assured that all pertinent references are obtained from a search. It also has disadvantages, namely:

1. It is costly to index all the data elements on a form using this scheme--a form of 100 elements takes about 40 hours at a cost of approximately $2000. The current process is for the indexer to fill out a form with the appropriate facets for indexing each item. Metadata from this form are entered into the terminal and then a batch editing procedure picks up illegal terms and other errors which are subsequently corrected. Varying degrees of interactive indexing have been considered. It seems highly desirable to speed up the indexing process, but we hesitate to invest in the extra programming necessary without knowing other people's experience with such an approach.

2. Currently, EIA uses the Energy Data Base (EDB) thesaurus for indexing our publications, in common with other agencies. As there is a strong move afoot to coordinate all such efforts nationally, it seems unreasonable to change our mode of indexing publications, but there are sufficient differences from the DRD classification scheme that integrating the two indexing systems presents problems. The FEDEX hierarchy has multiple routes to the same entry. For example, if references to "coal" are sought in the EDB thesaurus, there are three routes: (1) materials→carbonaceous materials→coal; (2) fuels→fossil fuels→coal; or (3) energy sources→fossil fuels→coal. When we get to one lower level of coal in the DRD, we have separate entries for anthracite, bituminous lignite and peat, whereas in the EDB thesaurus the entries for anthracite and bituminous coal are subsidiary to the entry "black coal" and peat is parallel to coal under fossil fuels. In other instances, the DRD is more specific than the EDB. It would be feasible to relate these structural differences for words currently in the DRD, but it would complicate the process of adding additional words.

3. The vocabulary for the DRD was chosen so that each word was unambiguous. Current efforts are underway to add synonyms and cross references to assist a specialist in a particular meaning in the context of this industry. The following are examples in the petroleum industry. The term "stocks" is cross-referenced to the DRD term "inventories (energy sources)," and "shipments" is cross-referenced to the DRD term "deliveries" as there is a one-to-one correspondence in meaning under current usage at EIA. We refer to these types of cross-references as translations.

In other instances words used by petroleum specialists can have different meanings according to their context. An example is "inputs." This item can refer to (1) refinery processing in general, (2) distillation (refining), (3) distillation (natural gas processing), and (4) the crude oil and other products that are input into a refinery. There are several approaches possible for handling this problem, of which the most feasible seems to be a translation to the term "processing functions." This approach would enable the searcher to go to the vocabulary BROWSE feature and select the most appropriate of the precisely-defined terms available.

Other translations that have been installed are more obviously synonymous or could even be thought of as alternate spellings. For example, "mix," "mixture," and "mixtures" are equivalent when used in such strings as "ethene-propane mix."

CONCLUSION

Keeping abreast of what information is available is an important function of an agency whose purpose is to provide the public with energy information. The DRD systems provide effective means of searching for information to satisfy a variety of needs such as verifying that the information requested on a new survey form is not already available, or determining where a particular item of information is published and from whom it is obtained. EIA is proceeding to link together many disparate subsystems and so increase the search capabilities. We are constantly looking for improved strategies and would much appreciate learning about other experiences in this area. In particular, we would like to consider the advantages and disadvantages of a highly structured hierarchical thesaurus compared with less rigid constructions; we would like to know whether tools exist that would speed up the lengthy indexing process and we would value any insights that can be shed on the problems of indexing terms that are used with different meanings in different contexts.

Some experiments in evaluation of an expert system for statistical estimation on databases

Neil C. Rowe[1]
Department of Computer Science
Stanford University
Stanford, CA 94305

Abstract

In our paper for the first of these Workshops [1] we claimed advantages of a new approach to estimation of statistics on a databases. We now back up these claims with quantitative experimental evidence of the comparative performance of our approach versus several simpler alternatives.

This work is part of the Knowledge Base Management Systems Project, under contract #N00039-82-G-0250 from the Defense Advanced Research Projects Agency of the United States Department of Defense. The views and conclusions contained in this document are those of the author and should not be interpreted as representative of the official policies of DARPA or the US Government.

[1]Current address: Department of Computer Science, Code 52, Naval Postgraduate School, Monterey, CA 93940

1. Introduction

We have been constructing a rule-based system for top-down estimation of the values of statistics on the contents of databases [2, 3, 1]. Our approach has two parts, a "database abstract" consisting of precomputed statistics on a particular database, and a set of inference rules for estimating the values of other statistics not stored in the abstract. This approach has quite different advantages and disadvantages than the main competing technique of random sampling. To help understand these advantages and disadvantages a quantitative evaluation is helpful. We validate performance here on a quite different database than that on which the system was originally developed, demonstrating a degree of portability of our ideas.

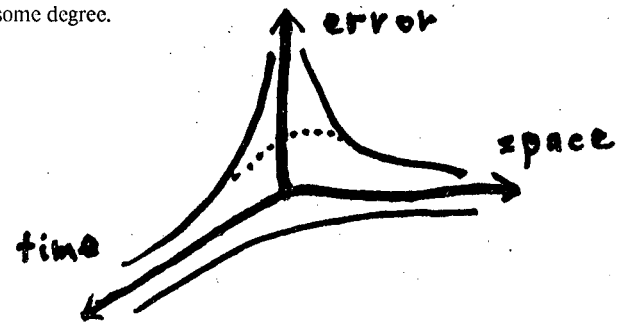
After an overview in section 2, we explain our method for comparing answers and estimates in section 3. Section 4 introduces the four basic control experiments needed to validate performance. Section 5 introduces results tables for the medical database. Section 6 discusses these results in regard to space and accuracy performance, and section 7 in regard to time performance. Further experiments with this and another (the original) database are covered in [3].

2. The evaluation problem

Our work is in the tradition of many rule-based "expert systems" developed in the field of artificial intelligence [4]. But unlike, say, a medical expert system designed to choose the most appropriate medical treatment for a patient, there is a quite rigorous way in which our statistical estimation expert can be evaluated: comparison

of estimated numeric values with the actual values of the same statistic found by going back to the original data. We can thus quantify the disparity, and study the relative effectiveness of the estimation for different kinds of statistics, database sets, and attributes of those sets.

Evaluation of a statistical estimation system is tricky because there is a three-way tradeoff between space required, time required, and accuracy obtained. Plotted as a three-dimensional surface (see below), it is roughly a hyperboloid. (We expect this from the information-theoretic assumption that the total information quantity transmitted across a channel is constant, that is, SAT=K where S=space, A=accuracy, T=time, and K is some constant.) For a fixed level of accuracy, the curve is a hyperbola relating time vs. space. For less accuracy, the hyperbola moves closer to the vertical axis; moving this way "up" the surface is essentially what our system is trying to do. We would like to parameterize this surface to some degree.



3. Comparing answers and estimates

To quantify "how close" an estimate is to an actual answer we use the number of consecutive high-order bits in common between the estimate and the answer, $-\log_2|(\text{est-ans})/\text{ans}|$. We wish to compare space and accuracy, and representing both in bits allows this. We could have used other metrics, as for instance an extremum of estimation rather than a normative summary of estimation, but we felt (1) extrema are harder to compare to space measurements, and (2) a good normative measure seems to correspond better to an intuitive performance assessment of an estimation system.

There are three problems with this bit-accuracy formula, however. First, the actual answer may be zero or negative; we ignored this, since much statistical data represents positive sums and counts anyway. (To handle this we would need to measure absolute accuracy or something else other than bits in common.) Second, the estimate may be exactly equal to the answer, giving an infinite number of bits in common; we handle this by putting a maximum on all such measures equal to the bit accuracy of the attribute whose statistics are being estimated. Third, the estimate may be greater than twice the answer, or less than half the answer, in which case the formula becomes negative; we handle this by arbitrarily rounding all negative results to zero.

We thus tabulate this performance metric on a series of random queries to our system. These abrupt changes for very close and very far apart values, however, are somewhat arbitrary, and so we also tabulate the breakdown of items in each of three categories (very close, reasonable, and very far) for a set of random queries.

Estimation performance depends on the database abstract as well as the inference rules, and so is database-dependent. In particular, it will not work well for data with complicated correlations between attributes. And since statistics on very small sets are not usually very meaningful, and subject to large variances, we impose the restriction that we only check our system performance on queries ten items or more in size. In addition, performance depends on the particular statistic queried, the form of the query set, and the query attribute. To avoid averaging these factors out over many queries we tabulate performance separately for major categories.

4. Control experiments

In order to demonstrate that our estimation approach is advantageous we must have a standard of comparison, a "control" experiment. Matters are complex because there are at least four such controls. We present them in approximate order of increasing challenge to our methods.

1. answering the query from a database abstract without using any inference rules (the "null rules" control)
2. answering the query with a minimal abstract (just statistics on each relation as a whole), but a full set of inference rules (the "null abstract" control)
3. running the query on the full original database, calculating the exact answer (the "full database" control)
4. "upwards" inference from a random sample the same size as a particular database abstract (the "sampling" control)

Our system must perform at least as well as all four of these in order for it to be judged a "success". By "at least as well" we mean that if any two of the three factors of space, time, and accuracy are held constant, performance will be better in regard to the third factor. To put it in terms of the hyperboloid, the hyperboloid representing behavior for the experimental scheme must be "below" the hyperboloid representing behavior for the control scheme for a reasonable range of parameters. In some cases, only one factor needs to be held constant. For instance, for the first and fourth controls above, we shall show that both time and accuracy are better when space is held constant.

Restrictions that the database, abstract, or sample be of a certain size or type, are fair when stated explicitly. Significance can be checked by the standard deviation of the bit accuracy and usual hypothesis-testing methods.

5. Some results

We developed an implementation of our ideas originally for four attributes of a small subset of the merchant shipping database of the KBMS project at Stanford, doing debugging and preliminary evaluation with this data. To demonstrate the generality of our ideas more convincingly we needed a different database, and we chose a random subset of the database of the RX project at Stanford [5], itself a subset of the ARAMIS (American Rheumatology Association Medical Information System) database of information about rheumatology patients. We chose six attributes to analyze for 28 patients with a total of 1000 visit records: patient number, sex, disease activity level, temperature, measured cholesterol, and administered prednisone. (Occasional missing values for the last four attributes were filled with values on previous visits.) We created a vocabulary of 18 named (or "first-order" sets) whose statistics would be stored in the abstract, representing partitions into two parts on sex, four parts on disease activity, and three parts for the other four attributes.

We summarize our results in tables at the end of this paper. Results of thirty different experiments are displayed. Each experiment is numbered, and contains three lines of result data. Each experiment involved testing of estimation performance for ten random queries on six statistics: count (set size), mean, max (maximum), sigma (standard deviation), median, and modefreq (mode frequency). Results for each statistic are tabulated separately, and are spaced horizontally across the page in six columns, in that order.

Results for each statistic in an experiment are summarized in seven numbers presented on three lines, in this format:

<bits of accuracy>(<standard deviation of accuracy>)
<# of exact answers>-<# reasonable estimates>-<# poor estimates>
<average range narrowing>(<standard deviation of narrowing>)

where:

- <bits of accuracy> is the average number of bits in common between the estimate (EST) and the actual statistic value, in ten random queries, computed according to the formula in section 3. 10 bits is assumed the maximum accuracy for all these experiments, since it is the accuracy of the numeric data.
- <standard deviation of accuracy> is the standard deviation of those numbers for the ten queries
- <# of exact answers> is the number of queries, in the ten, that can be answered to at least 10 bits of accuracy, the accuracy of the data
- <# of reasonable estimates> is the number of estimates, in the ten, that were not near-exact, but no worse than twice the actual answer or half the actual answer
- <# of poor estimates> is the number of estimates, in the ten, that were more than twice the actual answer of half the actual answer
- <average range narrowing> is the average ratio, in ten random queries, of the range between the bounds on the estimate to the possible range of that statistic
- <standard deviation of the narrowing> is the standard deviation of the preceding in ten random queries

Experiments 1 through 6 show results for statistics on the temperature attribute, experiments 7 through 12 prednisone dosage, experiments 13 through 16 results for two arithmetic operations between attributes, and experiments 17 through 20 results for set unions. Experiments tested particular query set forms, attributes or "fields", and abstracts. The same query sets were tested for each of the six statistics. "Exact" means that exact rules give a certain value for the answer, not an estimate; this is implies in our table a 10.0 bits of accuracy, a 10-0-0 answer breakdown, and .00 range narrowing. For set intersections, only those larger than 10 items were used for

tests, since statistics on smaller sets fluctuate widely, and the statistics are not particularly significant anyway. For set unions, because of time constraints, we disenabled the "backwards" reasoning or relaxation-style analysis used in all the other tests here; hence results are not as good.

6. Discussion: space and accuracy

Section 4 gave four separate control experiments we must compare performance against. Our experiments do not provide a complete comparison to each, in part because of time and space limitations, but they do cover most of the issues. (We used about 50 hours of CPU time on a DEC-20 at SRI International to perform this evaluation, coming close to the space limitations of single-user Interlisp in the process. 90% of the time expended was calculation of the database abstract values when needed from the actual data.) The basic philosophy of this evaluation is determination of "the value of rules" in the style of [6].

6.1. Control 1: Abstract, no rules

Clearly we can answer many more queries with rules on an abstract than without. An abstract can only contain contain a finite number of query answers, whereas rules can give statistics on arbitrarily large intersections and unions of sets in the abstract. The space for the rules is negligible compared to the size of the abstract because (a) rules can be coded efficiently since they contain few different symbols, and (b) we are interested primarily in large data sets where the abstract (as well as the database itself) is likely to be considerably larger than the rule storage.

6.2. Control 2: Rules, no abstract

We study this by experiment. For "no abstract" we still mean to include statistics on entire relations, information which it seems reasonable to assume is accessible to a user without the computer -- technically, a "null abstract". These conditions apply to the even-numbered rows in our first four tables. As one can see by comparing the figures with those for corresponding queries with a first-order abstract (experiment 2 with experiment 1, 4 with 3, 6 with 5, etc.), performance is usually significantly better. We can quantify the level of significance by the standard deviations given in parentheses on the first lines of the entries.

6.3. Control 3: calculation on full database

The third control experiment is getting the exact answer by running the query on the full database. The data is 1000 tuples with 6 attributes, a total of 1000 * 6 * 16 = 96,000 bits. The first-order abstract used in the experiments consists of 19 first-order sets plus the universe, with 14 statistics tabulated for numeric attributes and 5 for nonnumeric, for a total of 5*14 + 5 = 75 attributes, each with 10 bits of accuracy, for a total of 19 * 75 * 10 = 14,250 bits, or about 14.8% of the size of the database. (We ignore here the size of the program to manipulate the database abstract, as it is fixed in size independent of the database and database abstract, and its rules can be coded highly efficiently in few bits if desired.)

The main difference, however, is between the exact answers given by full-database querying and the limited accuracy of estimates. The tests we have run give average bits of accuracy for particular query forms. If multiply this by the number of possible queries of a given type, and sum up over all query types, we can get an estimate of a "virtual database size" due to the inclusion of inference rules along with the abstract. Of course there are an infinite number of queries since intersections and unions can be embedded arbitrarily deep, but one can set reasonable limits on query size (or better yet, weight query types as per their frequency of occurrence). As an example, consider just our estimates of the intersections of two sets. There are about 18 * 15 = 270 such sets, and the six statistics computed on these sets in experiments 1 cover 8.24 + 9.09 + 7.23 + 3.41 + 3.22 + 8.17 = 39.36 bits on the average per set, so there is virtual storage for about 270 * 39.36 = 10,600 bits, representing near to a doubling of the database abstract size. Similar figures can be summed over all experiments for all the common query forms. The total sum represents how well the rules are extending a given database abstract, and may be roughly compared to the size of the original database.

6.4. Control 4: Random sampling

The fourth and last control experiment is to extrapolate from a random sample the same size as the database abstract. We studied this experimentally by constructing a random sample the same size as our first-order (18-set) database abstract, 148 sample items out of 1000 in this case, and inferring upwards from statistics on the sample to statistics of the population. Results are contained in experiments 21 through 30 listed at the end of this paper. Experiments should be compared as follows:

- experiment 21 with experiment 1
- experiment 22 with experiment 7
- experiment 23 with experiment 17
- experiment 24 with experiment 19
- experiment 25 with experiment 3
- experiment 26 with experiment 9
- experiment 27 with experiment 5
- experiment 28 with experiment 11
- experiment 29 with experiment 13
- experiment 30 with experiment 15

Our rule-based method is about the same or better in most comparisons, while at the same time being likely to have much better access time in terms of page retrievals for all but very small databases, as discussed in [2], and while avoiding the "brittleness" mentioned there in regard to sets of related queries.

7. Discussion: time

These limited experiments do not well address the tradeoff between time and the other factors of space and accuracy, because significant advantages do not accrue unless the database is several orders of magnitude larger. A primary motivation for the database abstract architecture is the improvements in paging performance over random sampling and full-database-access methods, and 1000 sextuples should be easy to fit into most any computer's primary memory. But if we pretend that we have very limited primary memory and that all large datasets (database abstract and random sample as well as full-database tuples) are kept in pages (say 1000 16-bit words) in secondary storage, we can make the following comparisons for each of the four control experiments. (We assume, as with most large databases, that page accesses are the only significant time cost.)

7.1. Control 1: abstract, no rules

We assume rules can be coded efficiently and kept in core, hence they add no paging cost. Hence there is no difference in time.

7.2. Control 2: rules, no abstract

This alternative does obviate paging of the abstract, but that is only one page (14,250 bits = 890 16-bit words) for these experiments. For larger abstracts we assume all statistics on the same set are placed on the same page, and so an upper bound on the number of page accesses is the number of different sets queried. This number is constant for all queries of a given form. For intersections of two sets it is three: the first set, the second set, and their intersection. For unions it is four: all the preceding plus the union of the two sets. For unary and binary operations on simple attributes it is one since only the set actually queried need be accessed. So the number of page accesses needed to estimate a statistic with the database abstract is a small constant independent of the size of the abstract, rule set, or database.

7.3. Control 3: calculation of answer on full database

The database is stored on 1000 * 6 / 1000 = 6 pages. Even if there is an index pointing to every tuple of a given set, unless the sets are very small (say, 10 items or less) it is likely that at least one item of the set is on every page, except for the rare case where the database is clustered with respect to the partitioning that defines the set. Hence all six pages will need to be fetched nearly all the time, whereas only one page in these experiments, or a small constant number of pages in general, will need to be fetched to use the database abstract, a clear cost savings during query answering.

The database abstract does require setting up, however, which in turn requires accessing these very same six pages. But we only wish to use the database abstract architecture when setup work is small compared to query answering, and setup cost can be amortized to insignificance over a large number of queries. Setup can be made page-efficient, too, by implementation as a single-pass algorithm through the database.

7.4. Control 4: random sampling

For setup, this has essentially the same paging costs as the use of the abstract, since for reasonably-sized samples and more than just a few tuples per page, nearly every page must be retrieved for at least one tuple. In this particular case, the random sample is 148 items, and the odds are very high that each of the six pages will be represented. For a larger database with p items per page, and a random sample of size m of the database, the number of pages

looked at will be $p(1-e^{-m/p})$ from a Poisson model, assuming independence of page placement. Since our approach must always look at every page during setup, the net paging advantage of random sampling during setup is $(1-e^{-m/p})$. For $m=p$, i.e. the number of sample points being equal to the number of database pages, this is only a savings of $e^{-1}=36.8\%$ over our approach, and for most databases this represents a very small random sample, too small to be useful.

Once the random sample and the database abstract are created, they both fit into the same amount of space, and the same number of pages. But answering queries with the sample will likely require accessing most pages of it, because even if there is an index (which may require additional paging to obtain), one has similar paging inefficiencies with random placement of records as with random sampling of the full database. Thus as the size of the sample increases, the necessary paging to answer queries will increase nearly proportionately. At the same time, answering queries with the abstract will require a fixed number of page accesses (bounded by the total number of pages) depending on the form of the query, as discussed in section 7.2. In addition, new random samples usually need to be fetched from the database if a user is interested in another data set, whereas the database abstract is general-purpose. In a distributed architecture where the database abstract and/or random sample are separated from the database by a low-bandwidth connection, these additional fetches may be intolerable.

8. Conclusions

We have shown that our approach, applied to a particular database, does as well as random sampling, and in some cases significantly better, for the same amount of storage space. Our approach also does better than three other control methodologies.

References

1. Neil C. Rowe, "Rule-Based Statistical Calculations on a Database Abstract," *Proceedings*, First LBL Workshop on Statistical Database Management, Menlo Park CA, December 1981, pp. 163-176.

2. Neil C. Rowe, "Top-down Statistical Estimation on a Database," *Proceedings of the International Conference on Management of Data*, ACM-SIGMOD, May 1983, pp. 135-145.

3. Neil C. Rowe, *Rule-based Statistical Calculations on a Database Abstract*, PhD dissertation, Stanford University, June 1983.

4. Bruce G. Buchanan and Richard O. Duda, "Principles of Rule-Based Expert Systems," in *Advances in Computers*, M. Yovits, ed., Academic Press, New York, 1982.

5. Robert L. Blum, "Discovery, Confirmation, and Incorporation of Causal Relationships from a Large Time-Oriented Clincial Data Base: The RX Project," *Computers and Biomedical Research*, Vol. 15, 1982, pp. 164-187.

6. Donald Michie, "A Theory of Advice," in *Machine Intelligence 8*, E. W. Elcock and D. Michie, eds., Wiley, New York, 1976, pp. 151-168.

count	mean	max	sigma	modcfreq	median
Experiment 1: statistics on the intersection of two first-order sets, with respect to the temperature attribute, for a first-order abstract					
8.24(3.6)	9.09(1.2)	7.23(2.0)	3.41(1.9)	3.22(2.9)	8.17(1.3)
8-1-1	1-9-0	0-10-0	0-10-0	1-9-0	0-10-0
.01(.01)	.32(.19)	.28(.16)	.10(.07)	.07(.10)	.29(.16)
Experiment 2: same as experiment #1 but for null abstract					
.34(.70)	7.00(1.6)	3.95(1.1)	1.87(2.0)	.02(.04)	6.59(1.0)
0-2-8	0-10-0	0-10-0	0-7-3	0-1-9	0-10-0
1.0	1.0	1.0	1.0	1.0	1.0
Experiment 3: statistics on a first-order set, for the square root of temperature, first-order abstract					
exact	10.0(0)	exact	2.59(2.0)	exact	exact
	10-0-0		0-8-2		
	.002(.01)		.12(.09)		

Experiment 4: same as #3 but for null abstract					
.50(.90)	8.97(.9)	5.82(1.4)	0(0)	.52(1.4)	8.17(1.2)
0-3-7	1-9-0	0-10-0	0-0-10	0-2-8	0-10-0
1.0	1.0	1.0	1.0	1.0	1.0
Experiment 5: statistics on a first-order set, for the square of temperature, first-order abstract					
exact	exact	exact	7.38(1.7)	exact	exact
			0-10-0		
			.24(.14)		
Experiment 6: same as #5 but for null abstract					
1.21(1.7)	7.26(1.3)	3.93(1.3)	2.33(1.5)	.09(.18)	6.33(1.1)
0-4-6	1-9-0	0-10-0	0-8-2	0-3-7	0-10-0
1.0	1.0	1.0	1.0	1.0	1.0

count	mean	max	sigma	modefreq	median
Experiment 7: statistics on the intersection of two first-order sets, with respect to the prednisone dosage attribute, for a first-order abstract					
6.27(3.9)	3.51(1.9)	3.56(3.1)	1.49(1.6)	2.57(3.2)	4.99(3.4)
5-5-0	0-10-0	0-8-2	0-7-3	1-7-2	3-7-0
.03(.04)	.34(.31)	.34(.31)	.14(.08)	.08(.05)	.34(.31)
Experiment 8: the same as #7 but for a null abstract					
.73(1.0)	2.20(2.1)	1.90(3.3)	1.40(1.6)	.40(.92)	6.25(4.6)
0-4-6	0-7-3	0-6-4	0-7-3	0-2-8	0-8-2
1.0	1.0	1.0	1.0	1.0	1.0
Experiment 9: statistics on the square root of prednisone, first-order abstract					
exact	7.77(1.4)	exact	2.51(3.0)	exact	exact
	0-10-0		0-6-4		
	.05(.04)		.13(.05)		
Experiment 10: same as #9 but for a null abstract					
.76(.73)	4.11(2.6)	2.72(3.7)	0(0)	.79(1.6)	6.59(4.2)
0-6-4	0-10-0	0-6-4	0-0-10	0-2-8	6-4-0
1.0	1.0	1.0	1.0	1.0	1.0
Experiment 11: statistics on the square of prednisone, first-order abstract					
exact	exact	exact	1.90(1.1)	exact	exact
			0-10-0		
			.07(.06)		
Experiment 12: same as #11 but for a null abstract					
.86(1.4)	1.22(1.5)	2.99(4.6)	.16(.5)	.54(1.2)	6.14(4.7)
0-3-7	0-7-3	0-3-7	0-1-9	0-2-8	6-3-1
1.0	1.0	1.0	1.0	1.0	1.0

Estimation of statistics on prednisone dosages

count	mean	max	sigma	modefreq	median

Experiment 13: statistics on a first-order set, with respect to the sum of corresponding values for prednisone and cholesterol, with first-order abstract

exact	exact	5.59(1.8)	5.52(2.2)	.65(.8)	5.97(3.0)
		1-9-0	1-9-0	0-6-4	3-7-0
		.11(.08)	.04(.03)	.61(.89)	.08(.07)

Experiment 14: same as #13 but with null abstract

.38(1.1)	4.45(1.7)	1.50(2.2)	.62(1.2)	.39(1.2)	6.46(3.1)
0-1-9	0-10-0	0-7-3	0-3-7	0-1-9	4-6-0
1.0	1.0	1.0	1.0	1.0	1.0

Experiment 15: statistics on a first-order set, with respect to the product of corresponding values for prednisone and cholesterol, with first-order abstract

Some virtual-attribute statistics

exact	6.76(2.1)	3.42(3.4)	1.25(1.8)	.15(.3)	3.63(3.3)
	2-8-0	2-8-0	0-5-5	0-3-7	2-8-0
	.57(.36)	.34(.31)	.27(.18)	.30(.49)	.07(.05)

Experiment 16: same as #15 but for null abstract

1.11(1.1)	1.99(1.6)	.20(.40)	.29(.77)	.40(.76)	0(0)
0-6-4	0-8-2	0-2-8	0-2-8	0-2-8	0-0-10
1.0	1.0	1.0	1.0	1.0	1.0

Experiment 17: statistics on the union of two first-order sets, for the temperature attribute, with first-order abstract

6.84(2.4)	6.56(2.4)	exact	3.54(4.3)	7.12(2.5)	6.26(2.1)
3-7-0	3-7-0		3-3-4	4-6-0	1-9-0
.11(.14)	.37(.36)		.28(.25)	.07(.11)	.42(.35)

Experiment 18: same as #17 but for null abstract

.39(.9)	7.29(1.4)	4.45(1.1)	.87(1.6)	.23(.50)	7.00(1.3)
0-3-7	0-10-0	0-10-0	0-4-6	0-2-8	0-10-0
1.0	1.0	1.0	1.0	1.0	1.0

Results for set unions, without relaxation

Experiment 19: statistics on the union of two first-order sets, for the prednisone attribute, with first-order abstract

7.91(1.9)	6.59(2.6)	exact	6.63(2.5)	6.49(2.8)	2.19(3.3)
3-7-0	3-7-0		3-7-0	3-7-0	1-6-3
.05(.09)	.01(.02)		.04(.04)	.03(.04)	.22(.25)

Experiment 20: same as #19 but with null abstract

1.12(1.4)	3.58(1.9)	5.25(4.8)	3.02(3.2)	.92(.95)	0(0)
0-8-2	0-10-0	0-10-0	0-9-1	0-6-4	0-0-10
1.0	1.0	1.0	1.0	1.0	1.0

count	mean	max	sigma	modefreq	median
Experiment 21: statistics on the intersection of two sets and the temperature attribute					
2.00(1.8)	8.14(2.9)	7.38(3.3)	3.07(2.8)	2.51(3.8)	8.55(3.2)
0-8-2	1-8-1	5-4-1	0-7-3	0-7-3	8-1-1
Experiment 22: statistics on the intersection of two sets and the prednisone attribute					
2.30(1.4)	4.15(2.6)	3.35(4.4)	1.80(1.4)	2.36(2.1)	4.74(4.3)
0-9-1	0-9-1	3-3-4	0-7-3	0-7-3	4-5-1
Experiment 23: statistics on the union of two sets and the temperature attribute					
2.60(1.3)	9.51(.6)	4.73(1.1)	2.94(1.5)	2.62(2.3)	8.96(.7)
0-10-0	0-10-0	0-10-0	0-10-0	0-9-1	3-7-0
Experiment 24: statistics on the union of two sets and the prednisone attribute					
3.29(1.8)	3.84(2.2)	2.50(3.8)	3.52(1.5)	2.74(1.2)	8.45(3.1)
0-10-0	0-10-0	2-6-2	0-10-0	0-10-0	8-2-0
Experiment 25: statistics on a first-order set, of the square root of the temperature attribute					
2.57(1.5)	9.09(1.2)	6.78(2.2)	1.45(1.4)	3.08(1.9)	8.80(1.3)
0-9-1	0-10-0	3-7-0	0-7-3	0-10-0	3-7-0
Experiment 26: statistics on a first-order set, of the square root of the prednisone attribute					
2.75(1.7)	4.59(2.2)	3.29(3.4)	2.65(1.9)	2.38(1.5)	8.27(3.5)
0-9-1	0-10-0	2-7-1	0-8-2	0-10-0	8-2-0
Experiment 27: statistics on a first-order set, of the square of the temperature attribute					
2.82(1.6)	7.95(1.5)	6.48(3.0)	3.44(2.4)	2.59(2.9)	8.07(1.8)
0-10-0	0-10-0	4-6-0	0-9-1	0-8-2	4-6-0
Experiment 28: statistics on a first-order set, of the square of the prednisone attribute					
2.86(1.1)	3.28(1.8)	6.0(4.9)	2.00(1.5)	2.71(.6)	8.28(3.4)
0-10-0	0-10-0	6-0-4	0-10-0	0-10-0	8-2-0
Experiment 29: statistics on a first-order set, of the sum of corresponding values for prednisone and cholesterol					
1.75(1.6)	4.66(2.8)	4.34(3.9)	1.33(1.8)	1.64(1.8)	5.99(3.8)
0-6-4	0-8-2	3-5-2	0-4-6	0-7-3	4-4-2
Experiment 30: statistics on a first-order set, of the product of corresponding values for prednisone and cholesterol					
2.34(1.6)	3.50(2.0)	2.69(3.8)	2.22(1.6)	1.98(1.5)	5.07(3.6)
0-9-1	0-9-1	2-4-4	0-8-2	0-7-3	3-6-1

Results for extrapolation from a random sample.

The GENISYS Data Definition Facilities

A. Timothy Maness and Sue M. Dintelman
University of Utah

Abstract

This paper briefly describes the data definition facilities of the Genealogical Information System, GENISYS, which include the ability to define and to modify the definitions of the various database elements such as files, fields, logical links between files and forms to input, display and modify data. The use of a source definition and a target definition for data translation is presented. Also included is a list of proposed improvements and extensions to the current facilities.

Keywords - Data definition, data description, data dictionaries, meta-data.

I. Introduction

GENISYS, the GENealogical Information SYStem was developed to meet the needs of a multidisciplinary research project involved in historical demography studies and genetic studies of cancer and heart disease. The project database consists of demographic information for individuals associated in pedigrees and several riles of medical and vital statistics data. A detailed discussion of the design goals for GENISYS may be found in Maness and Dintelman (1982).

Because the first goal was to improve access to our existing data the initial focus of GENISYS development was the query language. One of the key features of GQL (the GENISYS Query Language) is the use of defined links between files to make complex, multi-file queries easy to formulate (Dintelman and Maness, 1982). For example, for a database consisting of an Individual file and a Household file, where the Individual file contains demographic information for a set of individuals and the Household file contains information such as location, type of dwelling, etc., for each household, possible links that could be defined are HEAD_OF_HOUSEHOLD, a 1 to 1 link from the Household file to the Individual file; HOUSEHOLD_MEMBERS, a 1 to N link from the Household file to the Individual file; and 1880_HOUSEHOLD, a 1 to 1 link from the Individual file to the Household file. In a query these links may be used to access information in one file based on criteria in another, for example:

```
SELECT individual_id, birthyear
WHERE 1880_HOUSEHOLD
dwelling_type = single
```

This query would list the identifiers and birth years (both fields from the Individual file) for each member of a single family type dwelling (a characteristic of a household).

GQL has provided us a convenient mechanism for formulating data access requests. Currently we are shifting our emphasis to providing more convenient ways for users to populate a database from both digitized and non-digitized sources, to access and modify the data dictionary to define new files and links and to allow users to interactively browse through their data and through the data dictionary. The next section describes the current data definition capabilities and the final section lists some of the extensions that are currently being added.

II. Current Data Definition Facilities

The following example illustrates the use of the data definition language and GENISYS commands to (1) create the definition of a newly acquired data set, (2) translate the data set into a more compact form, (3) generate a form to use to view, modify or add to the data and (4) place the definition of the file into the GENISYS data dictionary where it may be accessed by the query language and other system utilities.

Following is a simulation of an actual GENISYS session, with explanatory comments enclosed in braces.

{The GENISYS user creates a definition file and uses a standard system editor to input the definition of the newly acquired data file, which is a text file consisting of a subject identifier, the sex of the subject, a single data value followed by 4 readings with the dates of each reading.}

```
CREATE/DEFINITION  NEW_RESULTS
*append
1    SUBJECT_ID string length  5
2    SEX string length 1 recode
     ("M" "0", "F" "1")
3    XVALUE string length 5
4    MONTH1 string length 2
5    DAY1 string length 2
6    YEAR1 string length 4
7    READING1 string length 5
8    MONTH2 string length 2
9    DAY2 string length 2
10   YEAR2 string length 4
11   READING2 string length 5
12   MONTH3 string length 2
13   DAY3 string length 2
14   YEAR3 string length 4
15   READING3 string length 5
16   MONTH4 string length 2
17   DAY4 string length 2
18   YEAR4 string length 4
19   READING4 string length 5
20

*bye
```

{It is often desirable to translate data into a different format for analysis. One reason may be to compact the data to save space and access time, another may be to alter a coding scheme. For the purpose of our example the user now creates a second definition.}

```
.CREATE/DEFINITION LAB4_RESULTS
*append
1    INDEX FIELD
2      ID label "Subject Number:" integer
       length 5
3    SECTION
4      SEX label "Sex:" byte length
       1 recode ("M" 1, "F" 2)
5      XVALUE real length 5
6    ENDSECTION
7    SECTION repeats 4 times
8      MONTH label "Reading date:"
       byte length 2 range (0...12)
9      DAY byte length 2 range
       (0..31)
10     YEAR integer length 4 range
       (0,(>1975 and <1980))
11     READING real length 5 range
       (>0.0, <100.0)
12   ENDSECTION
13
*bye
```

{A detailed explanation of the features illustrated in these definitions is below. To continue the example, the user may now translate the original data file into the new format using the TRANSLATE command where the arguments are the source definition and the target definition.}

```
.TRANSLATE NEW_RESULTS LAB4_RESULTS
```

{Another command allows a GENISYS user to generate a form which can be used to display the converted data file, add or delete records or modify existing records.}

```
.GENERATE LAB4_RESULTS
```

{When the user is satisfied with the definition, the GENISYS DEFINE command is used to add the definition of the LAB4_RESULTS data set to the data dictionary.}

```
.DEFINE LAB4_RESULTS
.BYE
```

The two definitions above illustrate some of the features of the GENISYS data definition language. These are discussed below.

Index Fields. The first section of a description is used to designate the field or fields to be used as a primary key for the file. Indexes for other fields may be indicated using the INDEX keyword.

Data Types. GENISYS currently allows the following data types: bit, byte, integer, integer4, real, double precision, string, soundex (this is a special coded string used in our application). There are translation routines for all the reasonable mappings so that the TRANSLATION command may be used to translate from one type of data to another conveniently. Note that the length specified using the LENGTH keyword applies to the length of the field for display purposes. If the data type is anything but string the length of the stored data is based on the data type. There are default display lengths for all data types.

Undefined Values. The default undefined value is a zero (null) value, but the user may specify another value using the UNDEFINED VALUE keyword. This is useful in instances when zero is a valid reponse.

Range Checking. Following the RANGE keyword a specification of valid responses is listed. The example above illustrates the type of range specifications that are allowed. The ranges are checked during the initial input of a record whether during a batch operation using TRANSLATE or using an input form. Ranges are also checked whenever a record is modified. To apply a new consistency constraint to an existing file the GENISYS command CHECK_CONSISTENCY will find all records which do not meet the range requirements.

Recoding. The above example illustrates two uses of recoding. One use is to allow a change from one coding scheme to another. In the example above male was originally coded as 0 and female as 1. In the new file male is 1 and female is 2. The use of recoding also allows users to use an uncoded description of all types of descriptive data items such as race, religion, household type, etc., when doing data entry and data access. Using an uncoded description is much more intuitive and results in fewer errors due to misuse of coding schemes.

Record Layout. The physical layout of the two files described above was determined by the system from the relative position and data type of each field description. Specific byte (or bit) locations may be given using the LOCATION keyword to override the default position value.

Logical Data Structure. The SECTION breaks may be used to separate logical groups of data items, although currently the only use of non-repeating sections is to allow the use of the NEXT SECTION function key in a form generated from the definition. Repeating sections (such as the reading information in the second definition) and repeating fields are common in many of our applications. The internal structure of a logical file containing repeating information may include multiple physical files in order to save storage space. For example, a questionnaire may allow up to 10 treatment descriptions and the average used in the data collected to date may be only 2. Considerable space savings may be realized by using a separate file for the treatment descriptions. The translation command allows the physical structure to be changed as well as the logical structure, i.e. in the example above the readings were initially considered to be a flat list of 16 data items and in the translated file are considered to be four repeats of four data items. The physical structure of a new file may be specified by the user or determined by the system, as in the example, based on the trade off between the length of the repeating information, the maximum number of repeats, and the length of the system information required to implement a separate file.

III. Extensions

This section describes some of the features we are currently designing and implementing to expand the data description and data handling capabilities of GENISYS.

Consistency Constraints

The definition of consistency constraints is important for finding inconsistencies in existing data, for preventing errors in newly input data and preventing modifications that would introduce inconsistencies. Consistency constraints are particularly important for global files, i.e., data files that are utilized by the entire project. It is important that updates to global files meet the consistency constraints required by the entire group of users. We currently support the range checking of a single field and intra-record constraints, but are working on adding the ability to enforce inter-record constraints. As part of the work with our large genealogy file we have a list of 85 rules which must be met before a pedigree is considered consistent. We are very interested in making the specification of these complex constraints more convenient by allowing the use of the same link names and types of expressions as in GQL.

The consistency constraints will be implemented as a rules system (Stonebraker 1982). Rules take the general form of:

if <condition>
then <action>

where <condition> is a list of things to watch for and <action> is a list

of things to do. This same paradigm has been used extensively by the AI community in the development of expert systems (MYCIN (Davis, 1977)) and rules systems have been called "production systems, rule based systems, pattern-directed inference systems" (Nilsson, 1980).

An example of a simple rule that may be associated with a record for an individual is:

```
IF (BIRTHYEAR > DEATHYEAR  AND
DEATHYEAR ^ = 0)
THEN WARNING ("Birthyear is greater
than Deathyear")
```

If a request to change a record results in the condition that an individual's birth year becomes greater than his death year then a warning is returned to the process requesting the change. The actions associated with conditions may range from warnings, to refusal to perform the requested update, notification of systems personnel, or logging the fact that the condition of the rule was met for reporting purposes. The parsed form of the rules will be stored in the data dictionary as a hierarchical structure.

Abstract Date Types

Use of abstract data types provides a convenient way to deal with **complex data attributes**, that is, attributes which consist of elementary attributes. "Date" is an example of a data type which consists of "month", "day" and "year" pieces. Users should be able to use any of the following equivalent representations for a date value:

```
01/04/1983, 01-04-83, or January
4, 1983
```

It would also be convenient to specify in a query an expression such as

```
DEATHDATE > BIRTHDATE + "8 YEARS"

or

DATEOFDIAGNOSIS BETWEEN "1/1968"
AND "12/1974"
```

An abstract data type includes the definition of the internal representation of the data type and conversion routines to convert from the abstract data type to other types, such as strings for input or output. Also associated with the data type may be definitions of comparison routines, arithmetic operators, aggregate functions and special functions for the data type.

An additional example of the use of an abstract data type is the definition of a pedigree data type for general use in our project. The ability to define abstract data types will also make dealing with different coding schemes much more convenient. For example, death certificate records which contain ICD-O coding for cause of death use many different revisions of the coding scheme. In order to ask about a specific cause of death it is currently necessary to know the cause of death (COD) code for each ICD revision and ask for:

```
ICD_REVISION = 6 AND COD = V1
OR ICD_REVISION = 7 AND COD = V2
OR ICD_REVISION = 8 AND COD = V3
```

It would be much more convenient to define an abstract data type "CAUSE_OF_DEATH" which includes both the revision number and the code and conversion routines for causes of death that will be used such as "CANCER" or "BREAST CANCER".

Implementing abstract data types will require extensions to the data dictionary and type checking during the parsing phase becomes more complex and potentially time consuming. Preliminary implementations of abstract data types (Overmyer, 1982) indicate these are not insurmountable problems. Because our queries are currently compiled, inclusion of any of the functions associated with abstract data types present no difficulty. These types of functions will be included in our plans for a more interactive system by implementing them as separate processes using available interprocess communication facilities.

Use of Links and Forms.

The current implementation of GENISYS allows us to define forms which can be used to display, add and modify data in a single logical file. We would like to take advantage of the features of the currently supported system and include the ability to display, add and modify

links between different logical files.

In order to support these capabilities in a general way we will extend the data dictionary to contain the definition of forms which use path expressions to select data to be displayed on a form and to allow the definition of commands to move between forms. These commands will allow a user to define several forms and to specify the mechanism (usually some key value) to retrieve a new record (or records) and the form used to display it (them). Our current design of the use of forms in GENISYS has been influenced by the frames discussed by Catell in (Catell, 1980) and the forms discussed by Rowe in (Rowe, 1982).

The ability to quickly specify a form and use it to display records will be a useful companion to the use of GQL for preliminary analysis. In other words the use of forms will not be restricted to input operators, but will be useful for researchers who, we hope, will be creating their own customized forms.

Unified treatment of data dictionary and data

The importance of integrating meta-data facilities with database management facilities is discussed by several people, for example, see (Codd, 1982 and McCarthy, 1982). It is our goal to move closer to the ideal situation where the data base management system manages the data dictionary as a database, no different than any other. The feature we are missing is the ability to have self describing files and fields, that is, using the value of one field to interpret the contents of another and the use of a field to determine the type of record. Once implementation of this feature is complete, we will be able to define the data dictionary to itself and use GQL to query the data dictionary and define forms and associated commands to browse through the database. This will be useful for new researchers or researchers investigating new areas as they will be able to see what data is available, who has run queries or modified the data they are interested in, etc.

IV. Conclusion

This has been a very brief description of the type of data definition capabilities we currently have as part of GENISYS and the type of capability we are planning. Our main goal in all GENISYS development has been to implement tools which will be useful to and used directly by the researchers who are analyzing the data.

References

[1] R. G. G. Catell, "An Entity-based Database User Interface," Proc. ACM-SIGMOD 1980 International Conference on Management of Data, pp. 144-150, May 1980.

[2] E. F. Codd, "Relational Database: A Practical Foundation for Productivity," Communications of the ACM, pp. 109-117, February 1982.

[3] R. Davis, B. Buchanan, E. Shortliffe, "Production Rules as a Representation for a Knowledge-Based Consultation Program," Artificial Intelligence, vol. 8, pp. 15-45, 1977.

[4] S. M. Dintelman, A. T. Maness, "An Implementation of a Query Language Supporting Path Expressions," Proc. ACM-SIGMOD 1982 International Conference on Management of Data, pp. 87-94, June 1982.

[5] A. T. Maness, S. M. Dintelman, "Design of the Genealogical Information System," Proc. First International Workshop on Statistical Database Management, pp. 44-59, March 1982.

[6] J. L. McCarthy, "Metadata Management For Large Statistical Database," Proc. Eighth International Conference on Very Large Databases, 1982.

[7] N. J. Nilsson, Principles of Artificial Intelligence, Tioga Publishing Co., 1980.

[8] R. Overmyer, M. Stonebraker, "Implementation of a Time Expert in a Database System," SIGMOD RECORD, vol. 12, no 3, pp. 51-60, April 1982.

[9] L. A. Rowe, K. A. Shoens, "A Form Application Development System," Proc. ACM-SIGMOD 1982 International Conference on Management of Data, pp. 18-39, June 1982.

[10] M. Stonebraker, R. Johnson, S. Rosenberg, "A Rules System for a Relational Database Management System," Improving Database Usability and Responsiveness, pp. 323-335, 1982.

Acknowledgments

This research was supported by NIH grants HL-24855-04, CA-28854-02, HL-21088-06, and HD-15455-02, Public Health Services, DHEW.

LOGICAL AND PHYSICAL MODELING OF STATISTICAL/SCIENTIFIC DATABASES*

Stanley Y. W. Su, Sham B. Navathe, and Don S. Batory

Database Systems Research and Development Center
University of Florida, Gainesville, Florida 32611 U.S.A.

Abstract

This paper describes the major research tasks currently being undertaken at the Database Systems Research and Development Center of the University of Florida in the area of management of statistical databases. It presents the progress made to date and accomplishments related to the following: 1) the development of a semantic association model and its data language for use in statistical/scientific applications, 2) the study of (a) data mapping between a semantic model and the model used by a particular DBMS and between a logical model and its physical implementation, and (b) view integration problems in database design, 3) the investigation of data compression techniques and the development of a general model of database implementation, and 4) the study of parallel algorithms and database machine techniques for the efficient processing of statistical/scientific databases. The research project has been supported by the Applied Mathematical Science Program of the Department of Energy under contract #DE-AS05-81ER10977.

1. INTRODUCTION

In our complex, technologically-oriented society, the success of many human endeavors relies very much on the availability of data relevant to decision making. Examples of these endeavors abound in energy-related research, operations and management. In organizations involved in energy production/distribution/management, technical staffs and management personnel often need to access diverse and interdisciplinary statistical and scientific databases (SSDs) containing energy, census, geographical, environmental, and socio-economic data. These databases have been gathered by different governmental agencies and DOE laboratories and offices. There are several factors that make the access to these databases very difficult, if not impossible:

(1) In general, different databases are implemented on different hardware using different data processing systems. Consequently, there are no common conventions for naming, describing, formatting, representing, and structuring data. Also, the languages used for accessing and manipulating different databases are also different. Access to data in multiple databases is, thus, rendered very difficult.

It is, therefore, necessary to investigate the techniques and rules for mapping and interpreting data stored under different representations and translating queries issued for one system into queries suitable for another system. The data mapping and query translation problems also exist in relating the users' views (external models) to the community users' view (conceptual model) and to the internal structures (internal model) of a database. Furthermore, better language interfaces should be designed and developed to make it easy for the users who lack computer training to access the diverse databases.

(2) It is advantageous to have a generalized database management system (DBMS) to facilitate the accessing and sharing of valuable data among energy-related organizations. Unfortunately, the existing relational, hierarchical and network models used in business-oriented DBMSs are not adequate for defining and processing SSDs. Several recently proposed "semantic models" [CHE76, SMI77, HAM78, NAV78, COD79, SU79, HAM81, SHI81, KRE82], though richer in semantics, are not designed for SSD applications. SSDs have characteristics which are quite different from those of business-oriented databases [BIR78, HAP78, SZC78, CHA81, SHO82, BOR82]. They contain a large variety of data types: numeric data, such as fixed point, floating point and double-precision numbers, bit strings, vectors, matrices and arrays, as well as non-numeric data, including text, formatted and unformatted character strings. The operators useful for the manipulation of these data types are quite different from those used in the processing of formatted business data. For example, the operators for matrix manipulation and text processing needed for SSDs are generally not available in business-oriented DBMSs. The processing of SSDs often involves the use of aggregate functions and statistical routines to obtain summary data which provides a proper context for interpretation, extrapolation, and prediction. The operations required to aggregate and extract data from SSDs are different from those available in business DBMSs. They are a part of the semantics of SSDs and need to be explicitly modeled. Due to these and several other differences, there is a definite need for a data model that is tailored for SSD applications. The model should not only provide for modeling of a variety of high-level data types, aggregate functions and statistical summary data, but also allow an explicit definition of the semantic properties of SSDs so that meaningful operations associated with the data may be predefined and carried out by a DBMS. The latter property 1) eliminates the need for a redundant specification of these operations via users' queries, 2) allows a high-level data language to be designed and, thus, 3) simplifies the users' tasks in accessing and manipulating the databases.

(3) Inefficiency of data access and manipulation is always a problem in statistical/scientific applications. This is because of the sheer size of

*This project is funded under Department of Energy contract #DE-AS05-81ER10977.

databases involved and the time-consuming operations required in these applications. There is a strong need for the development of implementation techniques which allow (a) an efficient processing of SSDs in terms of retrievals, updates, aggregations, etc., and (b) facilities for handling compressed data, variable-length data, data types, which characterize SSDs, etc. The implementation techniques should take advantage of the special characteristics of physical data as well as operations of SSDs to achieve the needed efficiency. They should also take into consideration the ease of mapping the logical data representation and operations into its physical counterparts. It is expected that implementation techniques for SSDs will be quite different from those for the business-oriented databases.

Software techniques work well but often at the expense of generating additional system overhead and other software problems. The indexing technique for speeding up data search at the expense of ease of updating is an example. It is, therefore, important to seek alternative hardware solutions to the efficiency problem. Hardware architectures or special purpose hardware for supporting statistical/scientific processing can be expected to be quite different from the existing database machines designed for managing business-oriented databases.

Under the support of the Division of Engineering, Mathematical and Geosciences, Office of Basic Energy Sciences, Department of Energy, we have for the past two years (since July 1981) concentrated on the following research tasks in logical and physical modeling and design of SSDs:

1. The design of a semantic association model for the conceptual design of SSDs.

2. The identification of complex data types and their associated operations which are useful to the SSD users and are to be used as a basis for the design of a high-level, non-procedural data manipulation language.

3. The development and analysis of two data compression techniques: dynamic index encoding and the vertical elimination of repeating characters (VERC).

4. The design of a general model of physical databases to describe and define the physical structures of existing scientific and statistical database management systems.

5. A preliminary investigation into the mapping of semantic models into the logical and physical structures in a DBMS.

6. A framework for integrating the user views during the "view integration" phase of logical database design of SSDs.

7. Algorithms for sorting and statistical aggregations and their implementation in a microcomputer network.

2. OVERVIEW OF CURRENT EFFORTS AND ACCOMPLISHMENTS

In this section, we shall give an overview of our research effort and accomplishments during the past two years of study under the support of the DOE. The results of our research are detailed in [BAT82, BRO82, CHE82, FEI82, NAV82a, SU82a, SU82b, SU82c, BAT83a, BAT83b, NAV83, SU83a, SU83b]. The following three areas will be covered:

a) Semantic Modeling and Language Design
b) Database Mapping and Integration
c) Physical Design.

2.1 SEMANTIC MODELING AND LANGUAGE DESIGN

The design of a powerful data model is perhaps the most important and essential step in statistical and scientific database management. This is because a model is necessary to define and describe the logical views of users (external models) as well as the integrated view (conceptual model) of the user community. It is also essential for the development of a generalized DBMS for SSDs since the design of a data model would determine the data definition language and the data manipulation language which are the key facilities in a DBMS. The data model used for logical database modeling would also affect the physical database modeling and design because the physical database structure depends on the logical database structure and the mapping between logical and physical databases is one of the key considerations in efficient design.

The data models used in the existing commercial DBMSs and the current research works on semantic models are designed for modeling corporate databases rather than scientific/statistical databases (SSDs). SSDs have many characteristics that are very different from business databases. Most notably, SSD applications regularly deal with complex data types, such as matrices, time-series, set, vector, variable length text strings, date, etc. These are generally not recognized in existing DBMSs but are handled by application programs written in high-level programming languages. They involve operations such as 1) statistical aggregations and disaggregations (e.g., aggregate petroleum production by country, by state/province, by petroleum type), 2) conversion of data to suit statistical packages, 3) modification of data to produce the needed periodicity for statistical analysis of time series, etc., and 4) the regular database management functions, such as retrieval, update, insert, and delete. Because of the above and many other differences, a different data model is needed to adequately define a database. Also, new language constructs must augment the associated data language to make it easy for the user to manipulate the database. By expanding upon the different types of associations in the original semantic association model - SAM [SU79], we have developed a semantic association model SAM* which is tailored for SSD applications [SU82a, SU83a].

In SAM*, an SSD is modeled by a network of atomic and non-atomic concepts represented as nodes. The interconnections of these nodes (i.e., the arcs) specify how concepts are semantically related and are grouped to describe other concepts. Seven general grouping strategies called "associations" are recognized in the model in terms of which complex semantic information of a database can be explicitly defined. They are 1) membership, 2) aggregation, 3) generalization, 4) interaction, 5) composition, 6) cross-product, and

7) summarization associations. The distinctions among these seven association types are made based on the differences in structural properties, semantic constraints, and operations. The conceptual model of a database is graphically represented by the recursive use and nested structuring of these constructs. A simple example is given below. Figure 1 models an air pollution database which is a part of the air quality data of the South Coast Air Quality Management District of Los Angeles. As defined, AIR_POLLUTION_DATA is an entity type formed by an Aggregation Association and is explicitly labeled as an A node in the graph. It is characterized by a pollutant identification code, the locations where pollutant measurements are taken, and the measurement by hours and by days. POLLUTANT_D is the unique identifying attribute type (primary key) and is thus underlined. Attribute types are concepts formed by a Membership Association and are explicitly specified as M nodes. LOCATION is an entity type formed by aggregating attribute types LATITUDE, LONGITUDE, STATION_ID, and CITY. It has an existence dependency relationship with AIR_POLLUTION_DATA in that the existence of location data in the database depends on the existence of the pollutant entity which the location data characterize. Each pollutant type is measured in many locations at a given time. The times that pollutant measurements are taken are defined by the Cross Product of a set of hours (the M node HOUR) and a set of dates (the M node DATE). Thus, TIME is explicitly labeled as an X node (cross-product association). Each hour-and-date pair defines a time at which all measurements of a pollutant are taken at various locations. The attribute MEASURE is a summary of these measurements and is connected to TIME by HOURLY_MEASURE which represents a Summarization association, thus labeled as an S node in the graph.

In retrieving and updating the database, the access paths through the network and operations on the data represented by the network nodes are guided by the association types at the nodes. Thus, the enforcement of semantic constraints and the manipulation operations can be made implicit and need not appear as explicit commands in the users' application programs or queries. This greatly increases the power of a DBMS and its data language and simplifies the users' task in interacting with the database.

A tabular form for representing the data modeled by the association types is introduced in SAM*. It is called a Generalized Relation (G-relation) which is the mathematical notion of a relation defined over a set of attributes belonging to different data types, such as integer, real, matrix, set, ordered set, vector, time-series, text, etc., as well as the type G-relation. Thus, a G-relation is recursively defined allowing G-relations to be nested in other G-relations to an arbitrary level. Furthermore, two general types of attributes are distinguished: identification attributes and summary attributes. The former type identifies or characterizes entities or categories of entities and the latter type specifies the summary information over categories of entities. An example G-relation is shown in Figure 2. It is a tabular representation of the AIR_POLLUTION_DATA shown in Figure 1. In the figure, AIR_POLLUTION_DATA, LOCATION, HOURLY_MEASURE and TIME are G-relations in a nested structure. The association types associated with the G-relations are specified following their names. The summary attribute MEASURE is separated from the identification attribute TIME in G-relation HOURLY_MEASURE by two vertical lines.

A set of algebraic operators has been defined for processing G-relations. These operators are dif-

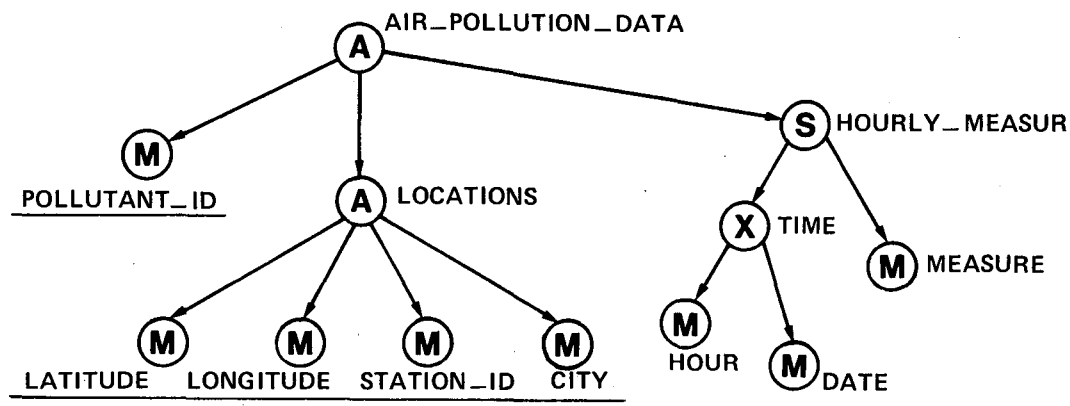


Figure 1. A Model for Air Quality Database.

AIR_POLLUTION_DATA(A)

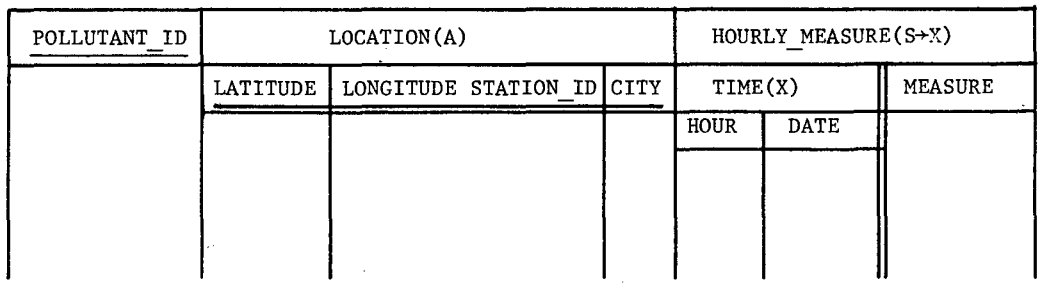


Figure 2. An Example G-Relation.

ferent from the usual relational operators in that they operate on data with complex data types and are subject to the semantic restrictions of complex data types. A number of operators for the statistical aggregation and disaggregation are included. They are not available in the existing relational systems.

2.1.1 COMPLEX DATA TYPES

Existing DBMSs support only a few primitive data types: simple numerics and character strings. For SSDs, which require extensive data manipulation, many more data types need to be recognized. Currently, an SSD user must perform the necessary operations through application programs outside of the DBMS. Not only is this a burden to the user, but the system has no control over whether the operations performed are semantically valid. A solution to this problem is to directly support an extended set of data types within the DBMS.

A complex data type (CDT) is a structured data type which corresponds to an abstract object commonly found in the user's view of data. We propose that a DBMS for SSD processing should recognize the following CDTs: set, vector, ordered set, matrix, date, time, time series, and G-relation. They are each defined and briefly described below.

1. Set. A set is a collection of elements, all of the same type, in which no duplicates are allowed. Some operations which should be included for set are:
   - finding the union, intersection, or differences of two sets;
   - adding, deleting, or replacing elements of a set.

2. Vector. A vector is a collection of elements, all of the same type, which are ordered. Since a vector is ordered, operations may be either position or content dependent. Content dependent operators include information retrieval operations, such as checking for inclusion/exclusion of a single element, a sequence of elements, or multiple occurrences of either. Position dependent operators utilize subscripting to provide flexible access of an entire vector, a subvector, or a single component.

3. Ordered Set. An ordered set is a set in which the elements are ordered and may be indexed. In addition to standard set operations, position dependent operations, such as retrieving all elements preceeding a specified value, are included.

4. Matrix. A matrix is a multidimensional collection of elements, all of the same type. The operations on matrix include those available for vector as well as higher level operations which take multidimensioning into account. For example:
   - insertion, deletion, or comparison of rows or columns of a matrix;
   - checking for the inclusion of a submatrix.

5. Time. Time is a coded value for duration from a fixed reference point. Operations available are: accessing by interval or accessing by temporal order.

6. Time Series. A time series is a two-dimensional matrix in which rows represent cases, and columns represent observations over time. Operators include basic matrix manipulations, as well as special modification operators, such as merging two time series.

7. Text. A text is a vector of characters. The inclusion of the data type text allows for a free-formatted field in the data. Common text processing operations include the following: recognition of partial matches (e.g., text contains at least 5 of the given list of words; fixed) and variable length Don't Cares (e.g., "match pollut*" should match pollutant, pollution, polluting, etc.).

8. G-relation. Collections of related data are represented as a tabular structure called

a generalized relation or G-relation. Unlike the relational model, the G-relation allows attribute domains to be any CDT, including another G-relation. Thus, hierarchical modeling is directly supported. Operations on a G-relation are performed at two levels of abstraction. At the low level, operations correspond to the CDT of the attribute domain. At a higher level, a collection of attributes forms a uniform tuple; and a G-relation is simply a set of tuples. As such, all set operations previously described are applicable to a G-relation. Traditional relational operators, such as join, selection, projection, etc., can be used for G-relations with some modifications.

A language specifically designed for SSD processing has been developed [BRO82] and is currently being revised. This Statistical and Scientific Data Language (SSDL) incorporates the following features:

1) Complex data types for direct modeling and high-level manipulation of data objects which correspond to the user's view of data.

2) A straightforward language structure for query specification. This structure, modeled after CASDAL [SU78b], allows for a natural way of specifying aggregate processing and of decomposing complex queries into simpler subqueries.

3) The integration of high-level procedures from existing statistical processing systems for descriptive analysis.

Complex data typing is a means of supporting the manipulations necessary for SSD processing at the user's level of abstraction. We feel that incorporating CDTs directly into the DBMS will result in more efficient and effective management of scientific and statistical databases.

2.2 DATABASE MAPPING AND INTEGRATION

The database mapping problem is related to the use of different data models for different purposes in the context of any database. In particular, the mappings of specific interest to us are the following:

1) from a semantic model to the model used by a particular DBMS

2) from a logical model to its possible physical implementations.

The first type of mapping stated above arises during any database design after a community view of data is formulated in terms of a high-level data model, and is to be implemented using a specific database management system. The second type of mapping belongs to the area of "physical design" and will be discussed in section 2.3.

A related problem to the above database mappings is the problem of "view integration." Navathe, in his previous work [NAV78, YAO78, NAV82b] has defined a framework for database design consisting of the following phases:

a) requirements analysis: gathering the general requirements of users for data and applications.

b) view modeling: modeling of individual users/application area's views.

c) view integration: integration of multiple users' views into a single global view.

d) schema analysis and mapping: mapping a community view into a logical database schema in the target database management system.

e) physical design: mapping the logical database schema into a physical database schema.

In the present project, it is apparent that we are trying to cover phases (b) through (e) of database design mentioned above. Requirements analysis, or phase (a), is not being explicitly addressed in our present scope of work.

To concretely apply the above framework to the design of SSDs would imply the following scenario. Each step in this scenario provides us with a research problem:

A) Individual user's views would be expressed in a semantic model.

B) These views would be integrated into a single "community view."

C) The community view would be mapped into a specific DBMS's logical schema.

D) The logical schema would be implemented in the form of a physical (or implementation) schema.

While A) was dealt with in the previous subsection, and D) is dealt with in the following subsection, here we shall focus our attention on problem B), view integration, and problem C), schema mapping. So far, we have researched these problems in a fairly general sense without limiting ourselves to just statistical and scientific databases. The general results of our research can be easily applied to special cases involving SSDs.

View Integration

Our work on view integration [NAV82a] takes the Navathe and Schkolnick (N-S) data model [NAV78] as its basis and discusses how a View Integrator would operate if user views in the N-S model were input to it for integration. Figure 3 shows a schematic diagram of the view integrator. The salient features of the view integration method-

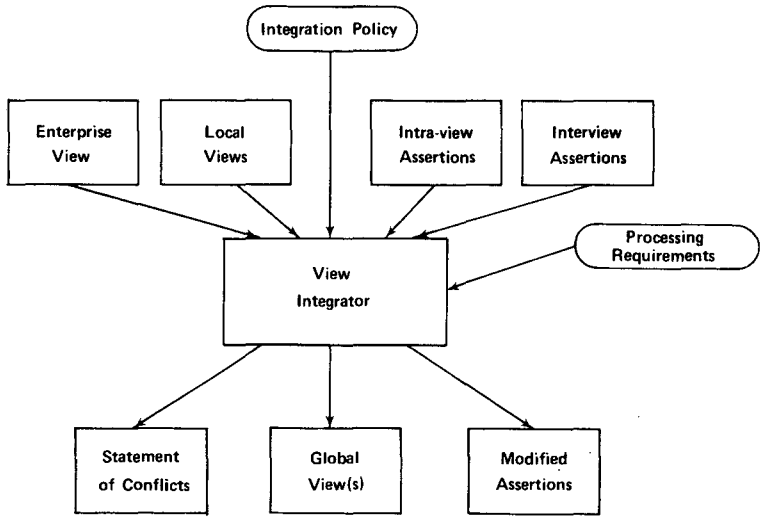


Figure 3. A Model for View Integration.

dology are as follows:

a) We allow for assertions or constraints to be expressed in an assertion language, which is an addition to the definition of a view. Both intra-view and inter-view constraints are allowed.

b) There is a notion of "equivalent views" which are views containing the same information, but having different structures.

c) In case of equivalent views, we allow for a quantification of the preference of a view, so that the integrator can process views in a decreasing order of preference scores.

c) In [NAV82a], a general procedure for view integration is defined which consists of separating views into equivalence classes and then performing integration on each class by a process of "matching." During view integration, the assertions are modified as views change and the designer is supposed to be constantly informed or consulted to resolve conflicts.

e) The matching process has been analysed in detail by considering how two different data objects may have a match on the name, key attributes and non-key attributes under various combinations.

f) A series of view integration operations have been defined to deal with the different constructs from the N-S model.

Previous work by researchers (e.g., [ELM79, ARO82]) has considered the view integration operations without considering conflicts regarding naming, identification, etc., which are so common in any real application. They also have not tried to specify how the view integration should actually be carried out in an interactive semi-automated manner as we do. The only other work similar to ours that we are aware of in this area is that of Batini [BATI82].

Our general framework can be made to apply to the SAM* model [SU83a] by providing an assertion language to describe the intra- and inter-view assertions, defining various types of equivalence, etc., with respect to that data model. This activity will be undertaken during the next phase of our project.

Schema Mapping

Mapping of database schemas among dissimilar models has interested several researchers in the past few years. Just the problem of mappings between the relational model on one hand, and the network or hierarchical model on the other, has been addressed by many (e.g., [KLU77, KLU78, KLU81, NAV80, ZAN79a, ZAN79b]). In the present context, our interest is mainly to address the mappings of a global community view of data into the logical schema of a specific DBMS. Barring a few exceptions [SAK80], this problem has not been specifically dealt with. We, therefore, decided to address this problem in a concrete way by assuming that the community view is available in a certain well-defined, usable, and fairly well-accepted semantic data model. For the community view, we selected the extended entity relationship model [CHE76, SCH80], and as the target model we chose the hierarchical data model. The methodology for schema mapping in this specific context is reported in [CHE82, NAV83]. The highlights of this work are summarized below. Our effort complements that of the Lawrence Berkeley Laboratory group, which is also experimenting with an extended entity-relationship model for better user interfaces through graphics (e.g., [WON82]).

a) It is assumed that the community (of users) view is modeled in the extended entity relationship (E-E-R) model using the entities and relationships as originally defined by Chen [CHEN76] and incorporating additionally three semantic constructs: subset hierarchies, generalization hierarchies and relationship of relationships [SCH80].

b) We divide the schema translation process into local translation, evaluation of alternate structures and global translation. The information about the E-E-R model input to the schema translation comprises:

- hierarchical dependencies and subsequent first order and general hierarchical decomposition as defined by Delobel [DEL78].
- quantitative parameters in the E-E-R model, such as the number of occurrences of an entity, average ratios of members to owners in different relationships, etc.
- transaction specifications in terms of the access paths used by each transaction and the relative frequency of each transaction.
- cost information pertaining to storage cost factors, relative cost of different types of accesses, etc.

c) Local translation algorithms are constructed to map individual constructs from E-E-R into the hierarchical model. These algorithms produce a unique schema if the given transaction specification clearly favors one alternative structure over all others. Otherwise, several target structures are produced and subjected to further evaluation of the total cost of processing the given transactions by using estimated cost factors. We realize that this method of evaluation does not yield "the optimal" schema as a result; however, at this stage of database design, one cannot conduct a more realistic analysis/evaluation. The resulting schema may be considered as a preliminary design or a guideline.

d) The target structures in the hierarchical model are then subjected to a merging process called "global translation." This actually is an integration of subschemas and is DBMS specific. For example, a system like IMS [IBM75] would use "logical pointers" to model a relationship between two segments; whereas, another system like S 2000 [MRI74] may use redundancy.

The above work has shown that the schema mapping between a semantic model and an implemented data model can be dealt with in a comprehensive way and optimization of target structures can be attempted at the logical level. We have also indicated in the above work how the real benefit of such mapping algorithms can be realized in a practical way for users by developing an automated schema mapping tool. Similar research may be performed on the mapping of semantic models, such as SAM*, into existing DBMSs.

2.3 PHYSICAL DESIGN

Our research on physical design has concerned investigations of new data compression techniques and the development of a general model of database implementation.

Data Compression Techniques

SSD files are characterized by large quantities of numeric and alpha-numeric data. Processing these files normally requires an examination of every record. Data compression is quite useful in this connection, for a reduction in storage volume is proportional to the increase in speed at which a file can be processed. For this reason, data compression plays an important role in SSD implementation. We have studied two compression techniques: dynamic index encoding and vertical elimination of repeating characters (VERC).

Index encoding is used to some extend in almost all SSDs. The basic idea is to identify the set of all distinct values that an attribute assumes in a data file. The elements of this set are sorted lexically and the index position at which an element appears becomes its index code. The data file is then encoded by replacing attribute values with their corresponding index codes. Since the storage requirements of codes are less than their data value counterparts, a sizable compression often results.

An important feature of index encoding is the identity of the index and lexical order of (code, value) pairs. Because of this identity, the costly process of translating index codes to data values can be eliminated during the data searching, sorting, and processing phases of most file operations [ALS75]; this enables operations to be performed directly and efficiently on compressed data.

An obvious problem with index encoding is the addition of a new data value to an attribute's domain; index codes must be recomputed and the data file must be recoded. Clearly, if data values are added frequently, the overhead of file recoding becomes significant. An obvious way around this problem is not to assign consecutive index codes. For example, if (A,I,O,U) is the domain of an attribute, one might assign the codes (0,8,16,24) rather than (0,1,2,3). Because codes are nondense, new data values can be added to a domain and unused index codes can be assigned in a way that preserves the lexical and index ordering identity. Doing so elimiates (or significantly reduces) the need to recode. For example, the data value 'E' could be added to the above domain and assigned index code '4' without altering previously assigned index codes or violating the lexical and index order identity.

The initial assignment of index codes and the method by which unused codes are selected and

assigned to new data values influence the overall performance of generalized (or "dynamic") index encoding algorithms. Several algorithms have been analyzed and a practical methodology for their application has been proposed [BAT82].

Perhaps the most elementary data compression technique is the elimination of repeating characters (ERC); a string of five A's "AAAAA" is replaced by the two-byte string "5A", where "5" is the repeat count and "A" is the repeat character. Versions of this technique have appeared in a number of significant commercial and specialized DBMSs: ADABAS [GES76, SOF77, KRO77], IDMS [KRO77, CUL81], and RAPID [TUR79, STA81]. Actually, ERC is only one of several data compression methods employed by these systems. However, experimental evidence shows that ERC accounts for 75% or more of the reduction in storage volume afforded by these DBMSs, so ERC is certainly important.

A survey of the compression techniques in use today reveals that they are primarily used to compress individual records; redundancies which might occur across consecutively stored records are rarely eliminated. Some exceptions do exist; e.g., see [EGG80, EGG81]. The Vertical ERC (VERC) was developed to eliminate such redundancies. The idea is to place a collection of records in a two-dimensional character array, where each row contains a single record. The array is then transposed, so that row i contains a single record. The array is then transposed, so that row i contains the ith character of each record. The ERC is then applied to each row, and the compressed rows are stored. From empirical studies at least 40 records should be compressed in this manner if the technique is to be effective. Consequently, the VERC may not be well-suited for conventional database processing where individual records are examined, but it is well-suited for statistical and sequential processing where all records are examined.

Experimental results show that there is a surprising amount of coherence in consecutive records. The VERC was found to have an equivalent or superior performance to that of the data compression algorithms used in ADABAS and IDMS, and to that of a commercially available compression package which is based on Huffman encoding [INF78]. Moreover, the program that implements the VERC algorithm was significantly less complex than those for ADABAS and IDMS. For these reasons, it is believed that the VERC technique will be useful in future statistical database implementations [BAT83a].

A General Model of Database Implementation

In addition to our investigations on data compression, we have also been developing a general model of statistical database implementation; the model applies to nonstatistical databases as well. There is a strong need for such a model. With few exceptions (e.g., [SCH77, SEL79, CAS81]), most of the research on physical design does not describe how theoretical models have been used to improve the performance of real database systems. Instead, generic problems of hypothetical databases utilizing generic structures have been studied. Certainly, such research is important, but the results are still remote from practice. In order to tie theory to practice, and to address the problem of improving the performance of real systems, the underlying structures and operations of real databases must be examined.

Presently, there are no models of physical databases that are general enough to account for the diversity and variety of structures (and their associated algorithms) found in commercial and specialized (i.e., statistical) databases. Although existing models have been used as starting points, considerable effort is needed to adapt and extend these models just to describe a single DBMS [CAS81]. The difficulty in using existing models clearly suggests that fundamental principles of physical database design and implementation are not well-understood and have been inadequately represented. Moreover, to improve existing models does not simply involve enlarging the spectrum of structures and operations that they describe. It requires much more.

To illustrate the disparity, consider how index records of inverted files are described in theory and how they are realized in practice. Database texts and research papers define the contents of an index record as a data value and an inverted list containing a variable number of pointers [AND77, KRO77]. The implicit structure of an index record is shown in Figure 4 [DAT81].

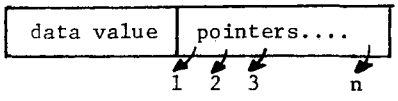


Figure 4. An Abstract Index Record

Commercial and specialized database systems rarely implement index records directly as in Figure 4. The reason is that the underlying file structures of most databases require records to have a uniform and fixed length; index records of Figure 4 have variable lengths. What Figure 4 really shows is an abstract representation of an index record (henceforth called an abstract index record). Actual database systems materialize abstract index records in a variety of different ways. Here are some examples:

RAPID [TUR79, STA81] and IMS [DAT81] materialize an abstract index record by pairing the data value with each pointer in the inverted list. Each (data value, pointer) pair defines a "concrete" index record, i.e., a record that is actually stored (Figure 5).

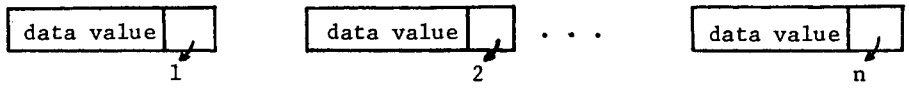


Figure 5. RAPID and IMS Realization of an Abstract Index Record.

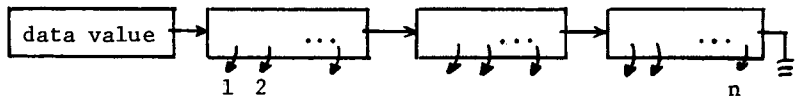


Figure 6. SYSTEM 2000 Realization of an Abstract Index Record.

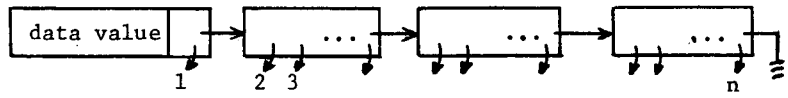


Figure 7. MRS Realization of an Abstract Index Record.

SYSTEM 2000 [KRO77, CAS81] materializes an abstract index record by storing the data value in a separate record and the inverted list in one or more additional records. A linear list chains these records together (Figure 6).

MRS [KOR79] materializes an abstract index record similar to SYSTEM 2000, except that the first pointer of the inverted list is stored in the record containing the data value (Figure 7). This was done so that if the data value was an identifier, no inverted list records would need to be accessed.

Each of the above materializations are functionally equivalent. That is, they all realize the same concept: an abstract index record. The idea of functional equivalence has a much broader application than this simple example suggests. In fact, it has been used as the basis of a new model of physical databases, called the functional equivalence model (FM). The modeling approach is to start with the generic logical record type that is supported by a statistical database system (SDBMS), along with a description of the types of fields that can be contained within it (e.g., single-valued attributes, repeating groups, matrices, etc.). Each logical record type is an abstract record whose materialization is to be determined. The materialization can be specified by a derivation involving the application of one or more elementary transformations, where at least one transformation is applied in each step of the derivation. Every transformation introduces a certain amount of physical detail to an abstract record. The product of applying a well defined sequence of transformations - the result of a derivation - is a collection of concrete record types, i.e., the physical record types that are actually stored, and their interconnections. In this way, for example, each of the materializations in Figures 5-7 is seen as a result of applying a different sequence of transformations to an abstract index record (Figure 4). The derivation is completed with the specification of the file structures that are used to organize the records of each concrete type.

This approach can handle not only the materialization of logical record types, but also the relationships between types. We start with a "generic" logical database diagram of the generic types, fields, and relationships actually supported by a statistical DBMS. Transformations are then applied to this logical or abstract description until its materialization has been defined. Thus, our approach embodies the idea of logical-to-physical mappings.

Although it may seem that the number of transformations is enormous, only nine distinct transformation types have been identified so far. These nine types have been sufficient to accurately describe the complex physical structures of the RAPID, INQUIRE [INF79], ADABAS, and SYSTEM 2000 DBMSs. (These DBMSs were selected primarily because of the immediate availability of information on their internal structures; models of other SDBMSs, such as ALDS, SYSTEM S, and SEEDIS, will be completed once sufficient information on their internals is gathered.) Details of the model and the physical structures of real SDBMSs are given in [BAT83b].

2.4 PARALLEL ALGORITHMS AND DATABASE MACHINES

Our effort on parallel algorithms and dataabse machines has been in the design and evaluation of sorting algorithms for a microcomputer network system called MICRONET [SU78a, NIC80, SU82b, SU83] and in the design and evaluation of a dynam-

ically partitionable network using the concept of shared main memory modules (SM3) [FEI83, BAR83].

The work reported in [SU82b] presents a key broadcasting algorithm for sorting a distributed file in a local area network, a parallel algorithm for finding the global maximal or minimal value of a data field or a distributed file and parallel algorithms for traditional data management operations, such as "join", "selection", and "projection". The key broadcasting algorithm was analyzed and compared with several other algorithms using the timing information of the existing MICRONET prototype system [NIC82]. Experiments have also been conducted using other sets of parameter values in an analytic study. The results are reported in a forthcoming paper [SU83b]. Our work shows that algorithms which are suitable for one architecture may not be suitable to (or optimal for) another. There is a close relationship between algorithm design and hardware implementation. Our effort is to find the relationship and use the knowledge to design better algorithms and hardware for supporting statistical/scientific applications.

Another major thrust is the study of a partitionable network system called SM3 which uses the concept of shared main memory modules [FEI83]. The main idea is to eliminate the network data transfer time (one of the main bottlenecks of a network system when large quantities of distributed data are to be transferred among processors) by transferring data through some shared main memory modules. The shared main memory modules can be switched electronically to a processor for data loading and be switched to another for data access, thus reducing the usual network transfer time to module switching time. The partition of processors into clusters is also achieved by setting or resetting switches. The clusters can carry out parallel processing of multiple database transactions. An analysis of the SM3 system has been carried out and will be reported in a forthcoming paper [BAR83]. Our analysis shows that statistical aggregation operations and common database management operations can take advantage of this shared main memory feature to gain efficiency.

2.5 CONCLUSION

We have highlighted above the problems that we have dealt with related to statistical database management in the last two years. Our main thrust in the past has been in the modeling, design, and mapping areas. In the future, we are likely to concentrate more on the architectural issues.

3. REFERENCES

[ALS75] Alsberg, P.A., "Space and Time Savings Through Large Database Compression and Dynamic Restructuring," Proc. of IEEE, Vol. 63, No. 8, 1975, pp. 1114-1122.

[AND77] Anderson, H.D. and Berra, P.B., "Minimum Cost Selection of Secondary Indices for Formatted Files," ACM Transactions on Database Systems, Vol. 2, No. 1, March 1977, pp. 68-90.

[ARO82] Arora, A.K. and Carlson, C.R., "On the Flexibility Provided by Conflict-free Normalization," Proc. of COMPSAC 82, Chicago.

[BAR83] Baru, C.K. and Su, S.Y.W., "Performance Evaluation of Statistical Aggregations in the SM3 System," Database Systems Research and Development Center, Univ. of Florida, manuscript, 1983.

[BATI82] Batini, C., Lenzerini, M., Santucci, G., "A Computer-aided Methodology for Conceptual Data-base Design," Information Systems, Vol. 7, No. 3, 1982, pp. 265-280.

[BAT82] Batory, D.S., "Index Encoding: A Compression Technique for Large Statistical Databases," CIS Technical Report #8182-9, Dept. of Computer and Information Sciences, Univ. of Florida, 1982.

[BAT83a] Batory, D.S. "A Data Compression Technique for Sequential Files," manuscript in preparation, 1983.

[BAT83b] Batory, D.S., "The Functional Equivalence Model of Physical Databases," manuscript in preparation, 1983.

[BIR78] Birss, E., Jones, S., Ries, D., and Yeh, J., "Scientific Data Base Management at Lawrence Livermore Laboratory: Needs and a Prototype System," in a special report on Generalized Data Management Systems and Scientific Information, OECD Nuclear Energy Agency, Paris, 1978, pp. 132-144.

[BOR82] Boral, H., DeWitt, D., and Bates, D., "A Framework for Research in Database Management for Statistical Analysis," Proc. of ACM/SIGMOD 1982 Conference, June 1982.

[BRO82] Brown, V.A., Navathe, S.B., and Su, S.Y.W., "Complex Data Types and a Data Manipulation Language for Scientific and Statistical Databases," CIS Technical Report #8182-7, Database Systems Research and Development Center, Univ. of Florida, June 1982; a revised version to appear

in the Proc. of the Second Int. Workshop on Statistical Database Management, Sept. 27-29, 1983.

[CAS81] Casas-Raposo, I., "Analytic Modeling of Database Systems: The Design of a System 2000 Performance Predictor," M.Sc. Thesis, Department of Computer Science, University of Toronto, 1981.

[CHA81] Chan, P., and Shoshani, A., "SUBJECT: A Directory Driven System for Organizing and Accessing Large Statistical Databases," Proc. of VLDB, 1981, pp. 553-563.

[CHE76] Chen, P.P.S., "The Entity-relationship Model: Towards a Unified View of Data," ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976.

[CHE82] Cheng, A.C., "Database Mapping from an Extended-Entity-Relationship into the Hierarchical Model," Master's Thesis, University of Florida, Dept. of Computer and Info. Sci., Dec. 1982.

[COD79] Codd, E.F., "Extending the Database Relational Model to Capture More Meaning," ACM Transactions on Database Systems, Vol. 4, No. 4, Dec. 1979, pp. 397-434.

[CUL81] Cullinane Database Systems, Inc., "The Internals of IDMS: Classroom Aids," April 1981.

[DAT81] Date, C.J., An Introduction to Database Systems, Third Edition, Addison Wesley, 1981.

[DEL78] Delobel, C., "Normalization and Hierarchical Dependencies in the Relational Data Model," ACM Transactions on Database Systems, Vol. 3, No. 3, Sept. 1978, pp. 201-222.

[EGG80] Eggers, S., and Shoshani, A., "Efficient Access of Compressed Data," Proc. VLDB, 1980, pp. 205-211.

[EGG81] Eggers, S., Olken, F., and Shoshani, A., "A Compression Technique for Large Statistical Databases," Proc. VLDB, 1981, pp. 424-434.

[ELM79] El-Masri, R., and Wiederhold, G., "Data Model Integration using the Structural Model," Proc. 1979 ACM/SIGMOD Conference, Boston, pp. 191-202.

[FEI82] Fei, T.H., Baru, C.K., and Su, S.Y.W., "The Shared Main Memory Modules (SM3) System," Database Systems Research and Development Center, Univ. of Florida, manuscript, 1982.

[GES76] Gesellshaft fur Mathematik und Datenverarbeitung, "ADABAS: Database Systems Investigation Report, Vol. 2, Part 1," Institute fur Informationssysteme, Bonn, W. Germany, 1976.

[HAM78] Hammer, M., and McLeod, D., "The Semantic Data Model: A Modeling Mechanism for Database Application," Proc. ACM SIGMOD International Conference on Management of Data, Austin, TX., May 1978, pp. 26-34.

[HAM81] Hammer, M., and McLeod, D., "Database Description with SDM: A Semantic Database Model," ACM Transactions on Database Systems, Vol. 6, No. 3, Sept. 1981, pp. 351-386.

[HAP78] Hampel, V.E., and Ries, D.R., "Requirements for the Design of a Scientific Data Base Management System," in special report on Generalized Data Management Systems and Scientific Information, OECD Nuclear Energy Agency, Paris, 1978, pp. 111-131.

[IBM75] IBM Corp., Information Management System (IMS/VS) Publications: General Information Manual (GH20-1260-3), System/Application Design Guide (SH20-9025-2); White Plains, N.Y.

[INF78] Informatics, Inc., "Shrink/2 Users' Guide," Canoga Park, CA., 1978.

[INF79] Infodata Systems, Inc., "INQUIRE Basic Training Course," Pittsford, New York, 1979.

[KLU77] Klug, A., and Tsichritzis, D.C., "Multiple View Support within the ASNI/SPARC Framework," Proc. 3rd VLDB, Tokyo, Japan, 1977, pp. 477-488.

[KLU78] Klug, A., "Theory of Database Mappings," Ph.D. Thesis, Dept. of Computer Science, Univ. of Toronto, 1978.

[KLU81] Klug, A., "Multiple View, Multiple Data Model Support in the CHEOPS Database Management System," Technical Report 418, Comp. Sci. Dept., Univ. of Wisconsin, 1981.

[KOR79] Kornatowski, J.Z., "The MRS User's Manual," Computer Systems Research Group, Univ. of Toronto, 1979.

[KRE82] Kreps, P. "A Semantic Core Model for Statistical and Scientific Databases," Research Proposal to DOE, Lawrence Berkeley Laboratory, 1982.

[KRO77] Kroenke, D., Database Processing, S.R.A., Inc., Chicago, 1977.

[MRI74] MRI System Corporation (Intel), System 2000 Reference Manual, Austin, TX., 1974.

[NAV78] Navathe, S., and Schkolnick, M., "View Representation in Logical Database Design," Proc. 1978 ACM/SIGMOD Conference, Austin, TX., May 1978, pp. 144-156.

[NAV80] Navathe, S.B., "An Intuitive Approach to Normalize Network Structured Data," Proc. 6th VLDB, Montreal, Canada, Oct. 1980, pp. 350-358.

[NAV82a] Navathe, S.B., and Gadgil, S.G., "A Methodology for View Integration in Logical Database Design," Proc. 8th VLDB Conference, Mexico City, Sept. 1982, pp. 142-164.

[NAV82b] Navathe, S.B., et al., "Logical Database Design," in Database Directions: Information Resource Management - Strategies and Tools, NBS Special Publication 500-92, Alan Goldfine (ed.), U.S. Dept. of Commerce, Sept. 1982, pp. 73-140.

[NAV83] Navathe, S.B., and Cheng, A.C., "A Methodology for Database Schema Mapping from Extended Entity Relationship Models into the Hierarchical Data Model," Proc. 3rd Int. Conf. on Entity Relationship Approach, Oct. 5-7, 1983

[NIC80] Nickens, D.O., Genduso, T.B., and Su, S.Y.W., "The Architecture and Hardware Implementation of a Prototype MICRONET," Proc. of 5th Conference on Local Computer Networks, Oct. 1980, pp. 56-64.

[SAK80] Sakai, H., "A Unified Approach to the Logical Design of a Hierarchical Data Model," Entity Relationship Approach to Systems Analysis and Design, P. Chen (ed.), North Holland, Amsterdam, 1980, pp. 61-74.

[SCH77] Schkolnick, M., "A Clustering Algorithm for Hierarchical Structures," ACM Transactions on Database Systems, Vol. 2, No. 1, March 1977, pp. 27-44.

[SCH80] Scheuermann, P., Schiffner, G., and Weber, H., "Abstraction Capabilities and Inveriant Properties Modeling within the Entity-Relationship Approach," Entity-Relationship Approach to Systems Analysis and Design, North Holland, Amsterdam, 1980, pp. 121-140.

[SEL79] Selinger, P., et al., "Access Path Selection in a Relational Database Management System," ACM/SIGMOD Conference, 1979, pp. 23-34.

[SHI81] Shipman, D.W., "The Functional Data Model and the Data Language DAPLEX, ACM Transactions on Database Systems, Vol. 6, No. 1, March 1981, pp. 140-173.

[SHO82] Shoshani, A., "Statistical Databases: Characteristics, Problems, and Some Solutions," Proc. 8th VLDB, Mexico City, Sept. 1982, pp. 208-222.

[SMI77] Smith, J.M., and Smith, D.C.P., "Database Abstractions: Aggregation and Generalization, ACM Transactions on Database Systems, Vol. 2, No. 2, June 1977, pp. 105-133.

[SOF77] Software AG of North America, Inc., "ADABAS: Introduction," Reston, VA., 1977.

[STA81] Statistics Canada, "RAPID Internals Manual," Ottawa, 1981.

[SU78a] Su, S.Y.W., Lupkiweicz, S., Lee, C., Lo, D.H., and Doty, K.L., "MICRONET - A Microcomputer Network System for Managing Distributed Relational Databases," Proc. 4th International Conference on Very Large Data Bases, Berlin, W. Germany, Sept. 1978, pp. 288-298.

[SU78b] Su, S.Y.W., and Emam, A., "CASDAL: CASSM's Data Language," ACM Transactions on Database Systems, Vol. 3, No. 1, March 1978, pp. 57-91.

[SU79] Su, S.Y.W., and Lo, D.H., "A Semantic Association Model for Conceptual Database Design," Proc. of the International Conference on Entity-Relationship Approach to Systems Analysis nad Design, Dec. 1979, pp. 147-171.

[SU82a] Su, S.Y.W., "SAM*: A Semantic Association Model for Corporate and Scientific Databases," CIS Technical Report, University of Florida, 1982.

[SU82b] Su, S.Y.W. and Mikkilineni, K.P., "Parallel Algorithms and Their Implementation in MICRONET," Proc. of the 8th VLDB, Mexico City, Sept. 1982.

[SU82c] Su, S.Y.W. and Mikkilineni, K.P., "Sorting Algorithms for Common-bus Local Networks," CIS Technical Report, Database Systems Research and Development Center, University of Florida, 1982.

[SU83a] Su, S.Y.W., "SAM*: A Semantic Association Model for Corporate and Scientific/ Statistical Databases," Revised version to appear in the Journal of Information Sciences, 1983.

[SU83b] Su, S.Y.W. and Mikkilineni, K.P., "An Evaluation of Sorting Algorithms for Common-bus Local Networks," paper submitted for publication, 1983.

[SU83c] Su, S.Y.W., "A Microcomputer Network System for Distributed Relational Databases: Design, Implementation and Analy-

sis," to appear in Journal of Telecommunication Networks, 1983.

[SZC78] Szczesny, K., and Gersbacher, W., "The Capabilities Required in a Generalized Data Base Management System for Handling Scientific and Technical Data," in a special report on Generalized Data Management Systems and Scientific Information, OECD Nuclear Energy Agency, Paris, 1978, pp. 106-110.

[TUR79] Turner, M.J., Hammond, R., and Cotton, P., "A DBMS for Large Statistical Data Bases," Proc. VLDB 1979, pp. 319-327.

[WON82] Wong, H.K.T., and Kuo, I., "GUIDE: Graphical User Interface for Database Exploration," Proc. of 8th VLDB Conference, Mexico City, Sept. 1982, pp. 22-32.

[YAO78] Yao, S.B., Navathe, S.B., Weldon, J.L., "An Integrated Approach to Logical Database Design," Proc. of 1978 NYU Database Symposium on Database Design, published as Vol. 132, Lecture Notes in Computer Science, Springer-Verlag, New York, 1982.

[ZAN79a] Zaniolo, C., "Design of Relational Views over Network Schemas," Proc. 1979 ACM/SIGMOD Conference, Toronto, pp. 179-190.

[ZAN79b] Zaniolo, C., "Multi-model External Schemas for CODASYL Database Management Systems, in Data Base Architecture, G. Bracchi and G.M. Nijssen (eds.), North Holland, Amsterdam, 1979.

PROPOSAL OF A LOGICAL MODEL FOR STATISTICAL DATA BASE.

Maurizio RAFANELLI(+), Fabrizio L. RICCI(-)

(+) Ist. Analisi dei Sistemi ed Informatica, Viale Manzoni 30
00185 Roma
(-) Ist. Studi e Ricerche Documentazione Scientifica, Via De
Lollis 12, 00185 Roma
National Research Council, Italy

Abstract - In this paper the Authors propose a logical model (called GRASS) for representing both the properties of a Statistical Data Base (S.D.B.) and the tables view, which represent the peculiar reality of statistical user.
The proposed model consists of a marked, labeled, direct, connected, acyclic, partially ordered graph. For this graph nodes semantics and connection and branching rules are provided. A cognitive and selective approach to how navigate through category attributes and summary data are given too.
Finally, some facilities of the model are discussed and a comment is made.

1. INTRODUCTION

The term "Statistical Data Base" (S.D.B.) refers, in scientific literature, to Database (D.B.) that represent statistical or summary information and are used for statistical analysis. They can be described in terms of the type of data they contain and their use.
S.D.B.s contain quantitative information (such as County business patterns, population census, economical data as production and consuction of fuels, etc...) and a combination of descriptive information (such as age, sex, race, average income, etc...) for each quantitative measure [1]. They tipically contain both parameter data and measured data for these parameters.
Parameter data consist , for example, of different values for varying conditions in an experiment; measured data are the measurements taken in an experiment under these varying conditions. These data bases are usually organized into "flat files„ or "tables„ [2]. Besides such S.D.B.s tend to be static because the stored data represent consolidated events.

We think that the term "S.D.B.„ is used to describe various situations [3]: a typical situation is to use conventional D.B.s, on which mainly transactions of the statistical type are effected and on which statistical packages therefore operate. But there are various reasons for the fact that conventional, commercial data menagement systems have not been wirely used for S.D.B.s, in that they are often inadeguate to manage efficiently these D.B.s. The main reasons are discussed in [4].

In this paper the S.D.B. term will refer to a particular class of D.B.s which has some characteristics enphasized; e.g., data are numbers, often they represent already statistical or summary information of elementary (disaggregate) data, they contain quantitative informations, data typology is various (e.g., integer or real values, averages, rates, etc...) and they contain a variety of data structures (such as matrix, vector, set, etc...); they are usually large D.B., in which some values may be missing and sparse data are very often contained.

Moreover they exhibit also the following characteristics:
a) attributes can be classified as category attributes (usually small) and summary attributes (usually large);
b) for each summary attribute a "cross product„ of category attributes is usually required;
c) it is possible, with proper techniques, to store the range values, related to the involved category attributes, once and to use known computational techniques to find the position of the corresponding summary set values. This obviates the fact that there is no reason to change it. Indeed such data represent consolidated (i.e. stabilized) events.

In the Section 2 we describe the type of data so as can be thought in a S.D.B.; in the Section 3 we report the logic rappresentation of data file by means of a graph, so as it is discussed in [5]; in the Section 4 we propose a logical model, giving the semantic description of the five types of nodes of such graph; in the Section 5 we illustre the connection rules for the above mentioned nodes, giving some examples; in the Section 6 we give the branching rules, discussing some significant cases; finally, in the Section 7 we discusse the proposed model and the future developments.

2. CATEGORIES AND SUMMARY ATTRIBUTES

Most S.D.B.s can be thought of as having two types of data: measured or quantitative data and parameter or descriptive data [6].
The first *quantitative data* have referred to numeric attributes on which statistical analysis is performed.
The second *qualitative data* have referred to as selection category attributes describe the measured data.
Therefore attributes for parameter data are referred to as "category„ attributes (into which the numeric attributes are classified), since they contain categories for the measured data.
The attributes for the measured data are referred to as "summary„ attributes, since they contain on which statistical summaries and analysis are applied.
In a S.D.B. there is usually a combination of category values for each summary value.
The measured data are usually "numbers„, while category attributes (being more descriptive in nature) tend to be "character„.
Often category attributes ranges are grouped together (such as using "age range„ rather than "age„) to form new category values.
Category attributes represent a *cross-product* of a n-dimentional space, since each combination corre-

SUMMARY TABLE.

COUNTRY= ITALY. UNITS=MTOE.

CONSUMPTION

	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990
SOLID FUELS	NA	10.5	12.9	14.0	14.9	15.6	16.6	17.4	18.6	20.1	21.0	22.7	24.6
NAT. GAS	NA	23.1	23.1	22.6	24.1	24.3	25.5	26.3	27.2	28.1	29.1	30.0	30.9
OIL	NA	102.2	98.3	94.6	92.7	94.5	98.6	101.0	102.6	103.2	103.6	105.1	106.7
NUCLEAR	NA	0.5	0.5	0.5	1.2	1.7	1.7	1.7	1.7	2.6	4.1	4.1	4.1
HYD. GEOTH.	NA	10.0	11.3	10.8	10.4	10.8	10.8	10.8	10.8	10.9	10.9	11.0	11.2
TOTAL	NA	146.3	146.0	142.5	143.3	147.0	153.2	157.2	160.9	165.0	168.7	172.9	177.5

PERCENT CHANGES

	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990
SOLID FUELS	NA	NA	23.0	8.2	6.3	5.2	6.2	5.0	6.8	8.2	4.2	7.9	8.5
NAT. GAS	NA	NA	-0.2	-2.2	6.9	0.9	4.8	3.1	3.4	3.5	3.5	3.1	3.1
OIL	NA	NA	-3.8	-3.7	-2.0	2.0	4.3	2.5	1.6	0.5	0.4	1.5	1.4
NUCLEAR	NA	NA	-9.1	3.5	142.9	37.0	-0.6	1.2	1.2	51.8	56.4	-0.3	-0.1
HYD. GEOTH.	NA	NA	12.5	-4.1	-3.7	4.1	-0.4	-0.1	0.0	0.9	0.6	0.6	2.2
TOTAL	NA	NA	-0.2	-2.4	0.6	2.6	4.2	2.7	2.3	2.5	2.3	2.5	2.7

SHARES OF TOTAL CONSUMPTION

	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990
SOLID FUELS	NA	7.2	8.8	9.8	10.4	10.6	10.8	11.1	11.6	12.2	12.4	13.1	13.8
NAT. GAS	NA	15.8	15.8	15.8	16.8	16.5	16.6	16.7	16.9	17.1	17.3	17.4	17.4
OIL	NA	69.8	67.3	66.4	64.7	64.3	64.4	64.2	63.8	62.5	61.4	60.8	60.1
NUCLEAR	NA	0.4	0.3	0.4	0.9	1.2	1.1	1.1	1.1	1.6	2.4	2.4	2.3
HYD. GEOTH.	NA	6.8	7.7	7.6	7.3	7.4	7.0	6.8	6.7	6.6	6.5	6.4	6.3

% DEGREE OF SELF SUFFICIENCY

	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990
SOLID FUELS	NA	3.5	3.7	3.6	3.4	3.4	3.3	3.1	3.0	3.2	3.2	3.5	4.1
NAT. GAS	NA	48.6	45.3	52.3	51.3	49.9	46.0	43.0	39.6	36.5	33.7	31.2	28.7
OIL	NA	1.7	1.9	1.5	1.7	1.7	1.6	1.6	1.6	1.6	1.6	1.6	1.6
TOTAL	NA	16.3	16.8	17.6	18.2	18.2	17.2	16.5	15.8	15.8	16.1	15.6	15.2

Fig. 1

sponds to one summary value [5].
The n-tuple of category attributes that selects the numeric attributes can also be seen as *primary key* of a *m-th* relation in a relational data base (with m > n).

We consider, for example, the summary table of Fig. 1, achieved from the Data Resourches Inc. Database [7]. This database reportes economical data on energetical sector, regarding production and consuction of fuels and energetic budget (prices, import-export, etc...); in particular this table shows data related to percent changes, shared of total consuption and degree of self sufficiency (in percent) in Italy from 1978 to 1990. Data supply is OCSE and AIE.

If we consider, for semplicity, that part of table related to "shared of total consuption„, such table is represented by the relation (in the relational model) *<shared of total consuption>*, that has the following attributes: *country, type of energy, year, percent change*. The classic representation of such relation is the table of Fig. 2.

<Shared of total consuption>

Country	Type of en.	Year	Percent change
Italy	solid fuel	1978	---
Italy	solid fuel	1979	7,2
Italy	solid fuel	1980	8,8
...	.....	....	...
Italy	nat. gas	1981	15,8
...	.....	....	...
Italy	hyd geoth.	1990	6,3

Fig. 2

We note that no distinction is made between parameter data and measured data (both of them described in terms of the attributes). The distinction between category attributes and summary attributes can be made only considering the category attributes as relation keys.
In this case the keys are *country*, *type of energy* and *year*. Besides, a very large redundance is introduced in those coloumns which are "parameter data„ (country, type of energy and year). We will see, in following sections, how such a redundancy is reduced using the proposed logical model.

3. GRAPH RAPPRESENTATION

One possibility to represent the semantic concepts mentioned above is described in [5]. SUBJECT is a system that represents such semantic concepts internally as a graph, so that they are invisible to user. In addition the concepts of "cross product nodes„ (X-nodes) and "cluster nodes„ (C-nodes) are associated to the ones of subject nodes, file nodes, data nodes and terminal nodes.
Cross-product abstraction refers to the multi-dimentional nature of the category attributes of a S.D.B.; it corresponds to logically divide the set of attributes into a set of n-dimentional spaces of data and can be used as a tool to represent more concisely multiple events of cluster abstractions.
Cluster abstraction is an organization mechanism at several levels in order to select category attributes and to reduce the complexity of a large cross-product.
Subject nodes (which belong to the class of cluster nodes) are used to describe subject categories of files in the system. *File nodes* (which belong to the class of cross-product nodes) represent the physical or virtual files present in the D.B..
Data nodes (which can be either cross-product or cluster nodes) represent the selection categories for statistical data. *Terminal nodes* (which belong to neither of the two above mentioned classes) represent the assumable instances for data nodes.

In order to access the summary values, a logical rappresentation of data file is achieved by using cross product and cluster nodes to describe the actual structure of a file.
The above mentioned type of nodes can be connected by edges to form a direct acyclic graph.
Both subject nodes and data nodes can be organized into multiple levels; all nodes can be shared, except the root node.

As example we consider a way of organizing data in a table form as shown in Fig. 3. The relative SUBJECT-graph is given in Fig. 4.

Year = 1982

Sex	Age	Number of passengers of plane	N. of pass. of railw.	N. of pass. of ship
male	1	1,500	....	....
	2	1,621	....	....
	.			
	.			
	.			
	35	163,438	....	....
	.			
	.			
	.			
	100	....	....	....
female	1	....	....	....
	2	....	....	....
	.			
	.			
	.			
	100	....	....	....

Fig. 3

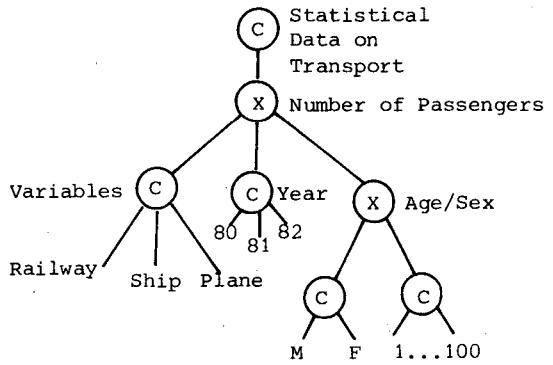


Fig. 4

As it may be seen, the C-node with label "statistical data on transport„ is a subject node; the X-node with label "number of passengers„ is a file node. All the nodes, except the leaf nodes under such file node, are data nodes. The leaf nodes are "terminal nodes„.

In [5] the semantics of "abstraction nodes„ (cross product and cluster node) and the main functions available in that system are given.
This system has also made some important first steps in the problem of navigating through the "metadata„. Metadata and their management is a non-trivial problem. Since an S.D.B. may consist of several thousand tables (each one with many attributes), just understanding the logical structure of such a D.B. is a complex task [8], [9].

4. THE GRASS LOGICAL MODEL

Also the GRASS *(GRaphical Approach for Statistical Summaries)* model proposed [3] in this paper is based on the concepts of category and summary attributes. The GRASS model endowes to the user a tool to know in what manner the S.D.B. is logically organized.
In the classic approach to design a D.B. it passes from the phase of "requirements analysis„ to the logical and phisical project.
The tools for the description of the reality are data models [10].
But for the analysts and the statisticians the *reality* is usually formed by *tables*.
They are the objects that must be described by logical model. Such a description bases itself just on the concepts of category and summary attributes. It introduces five new types of nodes that represent their meaning to the user (from the semantic point of view, with regard to the above mentioned concepts).
These nodes are *marked* (to distinguish the type) and *labeled* (to identify each node).
The edges are *oriented*, being the graph a *direct, acyclic, partially ordered, connected* graph. Such orientation is drawn by means of an arrow. This one is necessary to avoid ambiguity in the graph.

We give now the semantic description of the five types of nodes.

S-node: it represents the indicator of tables Selection paths; therefore it defines the meaning of tables combination of which it is the root and from which statistical data (selected by means of selection category attributes) will be subsequently obtained.
This means that it does not contribute (like the selection category attributes) to select numeric data, neither it does represent data that are phisically present in the D.B..
The root node of a graph that represents a "statistical view„ of data is always as S-node.

T-node: it represents logically the information (i.e. the statistical Tables) physically present in the D.B.; it is the *only* node of the graph able to represent the stored data.

C-node: it represents the single selection Category attribute for the statistical tables. A set I of instances, for example the domain of assumable values, is associated to it.
C-nodes can be organized at various levels of abstraction (and then they also represent aggregations of selection categories).
Let C be a generic node representative of a selection category and let I be the associated set of assumable instances, this node is equivalent to a C'-node, having as its branches nodes $C'_1, C'_2, \ldots, C'_k$ (with respective sets of assumable instances $I'_1, I'_2, \ldots, I'_k$), if:

(1) $I = I'_1 \cup I'_2 \cup \ldots \cup I'_k$

(2) $I'_i \cap I'_k = \emptyset \quad with\ i \neq k$

It follows that (see Fig. 5):

(3) $card\ (1) = \sum_{1}^{k_i} card\ (I'_1)$

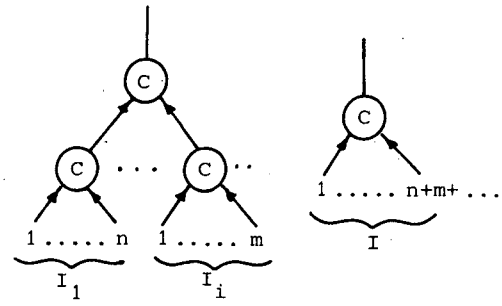


Fig. 5

A-node: it represents another selection category attribute for statistical tables, that Aggregates other category attributes in a determinate manner; also this node can be organized at various levels of abstraction.
Let A be a generic node, representative of a selection category, and let $C_1, C_2, \ldots, C_k$ be the branch nodes (with respective sets of assumable instances $I_1, I_2, \ldots, I_k$), the set I of instances that this node assumes is given by the *cartesian product* of the sets of instances associated to the branch nodes, i.e.:

(4) $I = I_1 \times I_2 \times \ldots \times I_k$

(5) $card\ (I) = \prod_{1}^{k_i} card\ (I_i)$

Node A, therefore, differs conceptually from node C in that this one represents merely a way of aggregating among them various selection category attributes. In fact we note that, although the two graphs of Fig. 6 are equivalent, the drawing of the graph without the use of A-nodes would produce an

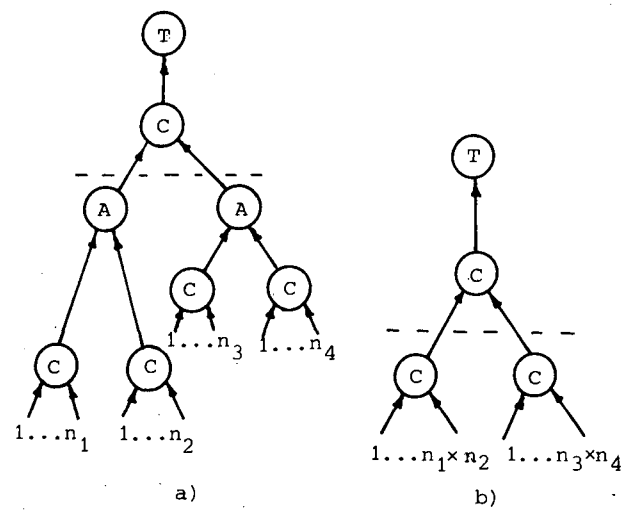


Fig. 6

excessive number of t_n nodes.

In Fig. 6a) we see that the number of terminal nodes is $n_1+n_2+n_3+n_4$, while if Fig. 6b) it is seen how the elimination of the A-nodes leads the number of t_n nodes to:

$$(n_1 \times n_2) + (n_3 \times n_4).$$

t_n-node: it represents the instance of the C-node at the more low level of abstraction. The Terminal Nodes (leaves) of a graph are only t_n nodes. For semplicity of drawing such instances are given directly in the graph.

5. CONNECTION RULES

The connection rules for the nodes of the GRASS model are now described.
Given a graph G, representing an S.D.B., we have:

Rule 1: a minimum graph G' is composed of the following chain:

$$S \rightarrow T \rightarrow C \rightarrow t_n$$

Rule 2: an S-node has as its branches S-nodes or T-nodes.

Rule 3: a T-node has as its branches C-nodes and/or A-nodes.

Rule 4: an A-node has as its branches A-nodes or/and C-nodes.

Rule 5: a C-node has as its branches C-nodes and/or A-nodes or only t_n-nodes.

Resuming the example of the summary table of Fig. 2, we can represent this situation by means of the GRASS-graph of Fig. 7.

ECONOMICAL DATA ON ENERGETICAL SECTOR FROM 1978 to 1990

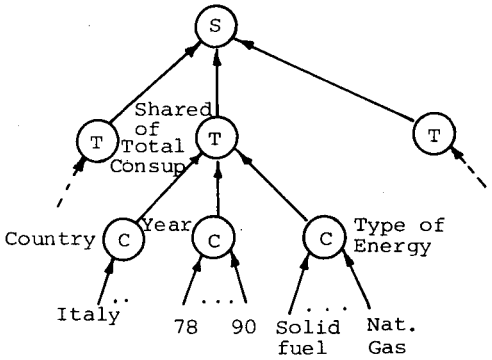


Fig. 7

We now consider another example (shown in Fig. 8) relative to the situation of the psychiatric hospitals in the city of Rome and to the years from 1978 to 1980 [11].

In this figure the available statistical tables, relative to the two T-nodes of the graph, are shown. These data give the number of hospitalizations by year and by Hospital, and the number of patients by sex, age-range and district of residence.

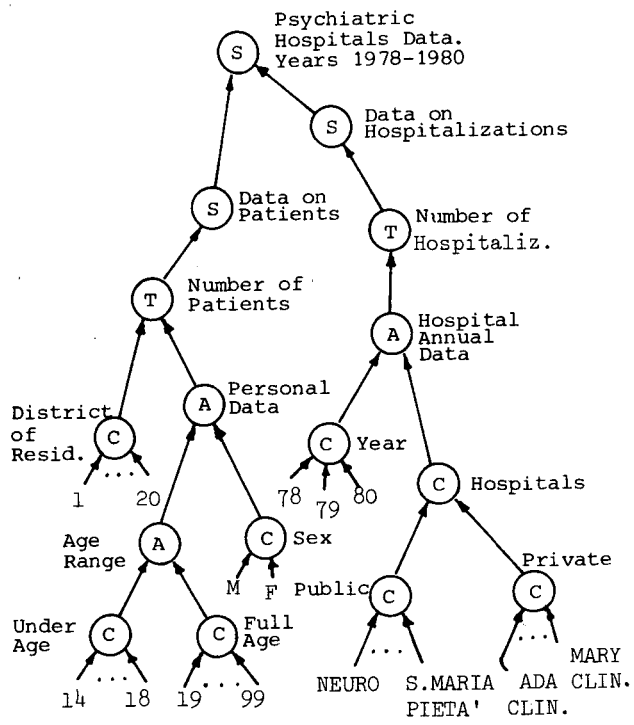


Fig. 8

These tables can easily be identified by means of the root S-node, which represents the "name„ of the S.D.B., and of the successive S-nodes, according to a cognitive approach, that identifies the table(s) (and both the related category attributes and the terminal nodes used to select the table(s)) object of the search.
In particular, in Fig. 8 the information regarding data on patients can be obtained through the selection category attributes "sex„, "age-range„ and "district of residence„.
It may be noted that the first two category attributes are organized at a higher level of abstraction by means of the A-node "personal data„. At the same manner, the information related to the "number of hospitalizations„ can be obtained by means of the "year„ and "hospital„ selection category attributes.
The last category attribute is a *generalization* of the other two category attributes "public„ and "private„, while the category attribute "personal data„ is the *aggregation* of "sex„ and "age-range„ category attributes [12].

6. BRANCHING RULES

We examine now some rules regarding the braching of subgraphs from the original graph.
Every time a statistical query is made, it identifies a subgraph branchable from the original graph (if data are available in the D.B.).

The rules to obtain this subgraph from logical schema are the following:

Rule 1: to select an S-node means to report the whole subgraph of which the S-node is the root.

Rule 2: the category attributes directly connected with a T-node must be reported in the subgraph with all their "terminals„, if they are not expressed.

Rule 3: if a category attribute is part of a "hierarchy of generalization„ and has not been expressed, it is not reported in the subgraph.

Rule 4: the nodes connected with an A-node which have not been expressed, must be reported in the subgraph with all their terminals.

Rule 5: if a C-node is selected and if only some branch nodes are mentioned, only they must be reported in the subgraph.

For example, we examine the graph of Fig. 9, in which data relating to railway transport are shown.

Wishing to obtain statistical information concerning the "number of passengers that have utilized railway transport„, under particular requirements, the chain of S-nodes which leads to the involved T-node from the root node will be selected.
For example, if the statistical query is the following:
"I wish to know the *number of passengers, of male sex, belonging to age range = 1E and that have travelles in 1st class.*„, the subgraph (obtained by applying the branching rules) is shown in Fig. 10.

The following observations may be made:

a) the "type„ category attribute (which is not mentioned in the statistical query) is considered in the branched subgraph, with all its terminal nodes;
b) the category attributes relating to age range = 2E,...,6E (which are not mentioned too) are not considered in branched subgraph.

When a statistical query does not mention a category attribute, two alternative situations can be happen:

1) all the selected data are showed in output;
2) output is the result of *summarization* of selected data.

For example, we consider the Fig. 11.
A statistical query of the type: *"I wish to know the number of passengers that have travelled by train in the 1st class„* selects the first three rows and all the columns of the table of Fig. 11.

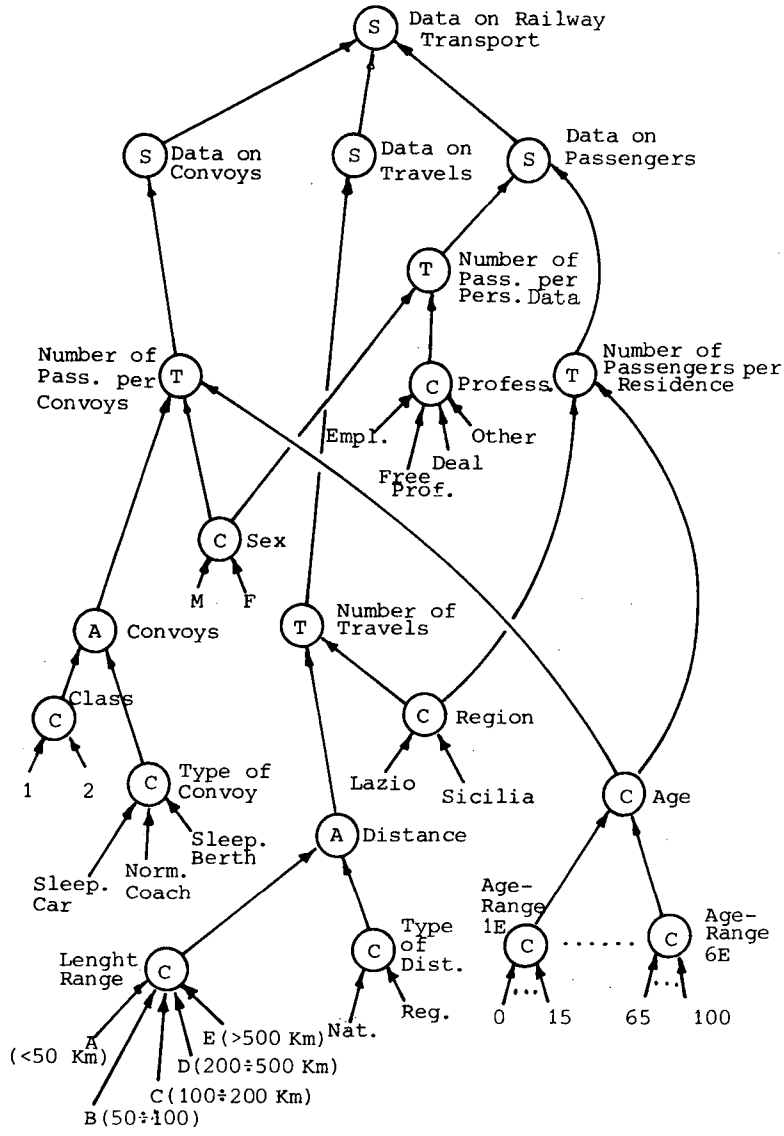


Fig. 9

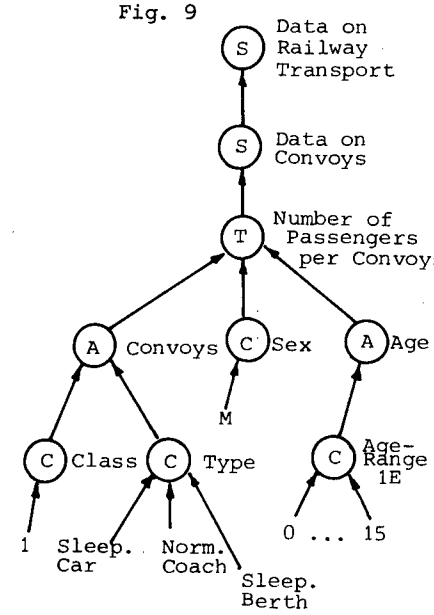


Fig. 10

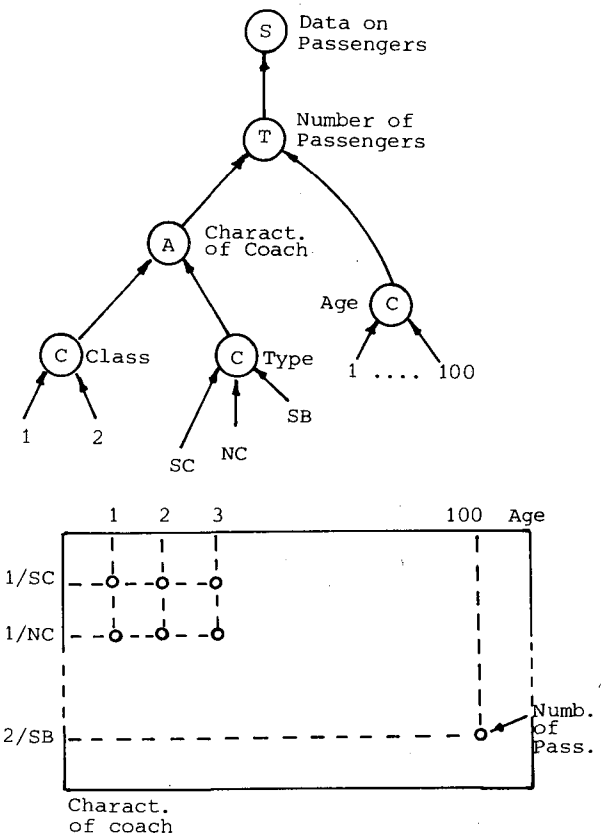
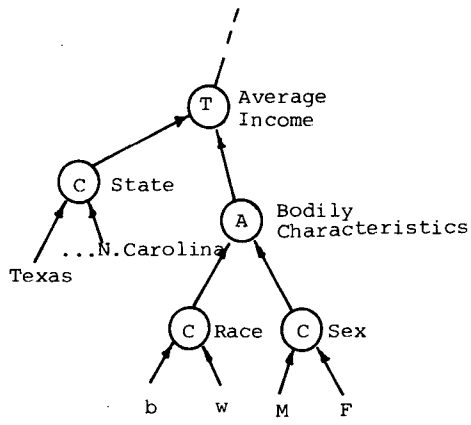


Fig. 11

The significant datum is obtained by "summarizing„ (i.e. adding) the selected data to one another. It is not always true that summarization is achieved by means of a sum, but such a function will depend on the particular type of data [13].

When the user expresses a statistical query, he does not know *a priori* if it is satisfiable. That can happen because:
a) data are missing in the D.B.;
b) classes (or groups) of category attributes (with regard to respective t_n) cannot be mutually exclusive, for which, if a "total„ are required, it can turn out as impossible;
c) a query requests a "summarization„ of data that is not possible to perform.
As simple example [14] we consider the case shown in Fig. 12. Here category attributes are "race„ and "sex„ and summary attribute is "average income„. Table is referred to TEXAS state.
It a user is interested only in population average income broken down by race but not by sex, such query has not answer, because "count data„ are not available.



TEXAS

race	sex	average income
b	m	24,000
b	f	21,000
w	m	28,000
w	f	25,000

Fig. 12

7. DISCUSSION AND FUTURE DEVELOPMENT

During the design process (from requirements analysis to phisical implementation) of S.D.B. it is necessary to keep in mind the particular typology of users. These users are not "casual„, also if they are not expert in computer science (rather in economical or epidemiological or other science) and accustomed "to see„ reality described in terms of *tables*.
It follows that the mapping process is not the classic process of management D.B. [3], but it will consider this "reality„.

The GRASS model offers a clear and compact view of the table that form the S.D.B..
The clarity refers mainly to different type of nodes (and to their semantic meaning): in fact, e.g., summary data (i.e. the tables) can be represented by means of a T-node and a T-node can represent only summary data.
The compactness is evident if we compare the GRASS graph of Fig. 7 with the relational model of Fig. 2; both of them represent the table of Fig. 1.
We note that to know in the GRASS model all the values of the definition domains of the k selection category attributes, n terminal nodes are necessary, with

$$(6) \quad n = \sum_{1}^{k} \dim(C_i)$$

(being $\dim(C_i)$ the cardinality of definition domain of C_i category), while m tuples of rela-

tional model are necessary, with

$$(7) \quad n = \prod_{1i}^{k} \dim(C_i)$$

Moreover, with regard to relational model, the GRASS model put in evidence the category attributes involved, without applying to the concept of "key„. Another advantage is the maior semplicity to occur when we will carry out a simple manipulation (with regard to a value of a category definition set).
For instance, the insertion of a new value requires the addition of only one t_n node, while the update requires only to modify a label. A further advantage of the proposal model is that it is used to describe meta-data that refer tables of an S.D.B. [15]. Moreover it allows a hierarchic description of the category attributes, for which user can see the whished detail level. Besides the navigation in the GRASS graph allows to explore data present in the S.D.B..
It is possible to navigate through the GRASS graph according to two approaches:
a) *cognitive approach*, in which user runs the graph from an S-node (the root or an S-node of another lower level) to one (or more) node(s); such an approach allows to know which attributes define a fixed table (or a fixed group of tables).
b) *selective approach*, in which user runs the graph from a t_n node to a T-node; such an approach allows to know which tables (and, than, which users view) are defined by fixed attributes.

We consider now the graph of Fig. 13, where the sets of terminal nodes (related to the C_i-nodes, with i = 1,2,...,k, which is connected to the A-node) are indicated with $I_1,...,I_k$.

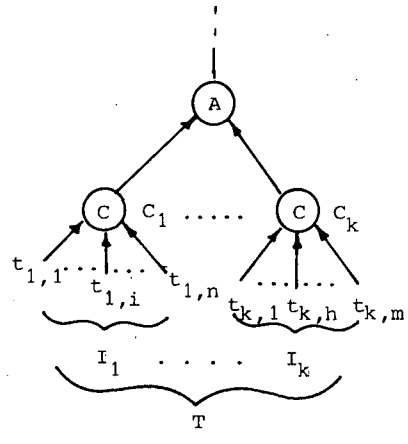


Fig. 13

Let ζ be the set of the "real„ terminal nodes of the A-node, it coincides with the *cartesian product* of sets I_i (with i = 1,2, ...,k), that is:

$$(8) \quad \zeta = I_1 \times I_2 \times ..... \times I_k$$

Therefore, every instance of selection category is a k-tuple $(i_{1,i},...,i_{k,j})$; rows or columns of statistical tables may correspond to each of these k-tuples. When no numeric attributes correspond to one or more instances of selection category, no distinction is made between unavailable values (not taked off) and inadmissible values. In both cases we will refer to them as *null values*.
Let us now suppose that only one subset, $\zeta' \subset \zeta$, is significant for the statistical query.
Then the cardinality of the terminal nodes set ζ' will be less than card(ζ), that is:

$$(9) \quad \text{card}(\zeta') < \text{card}(\zeta) = \text{card}(I_1) \times \text{card}(I_2) \times ... \times ... \times \text{card}(I_k)$$

otherwise some "integrity constraints„ would be violated.
This means that there are k-tuples of T $(t_{1,i},...,t_{k,h})$ that are not admitted.
For example, we consider again the graf of Fig. 10. Let us suppose that, with regard to the A-node "convoys„, the pair of instances "2nd„ of the C-node "class„ and "Sleeper choach„ of the C-node "type„ is not admitted. The cardinality of A-node "convoys„ then changes from 6 to 5.
Hence, such part of the graph will be changed as Fig. 14.

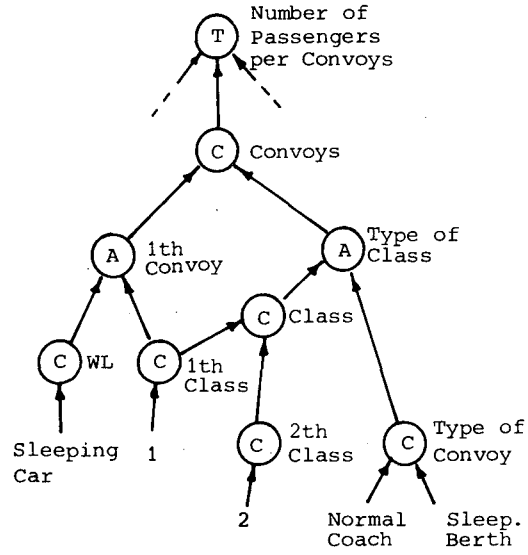


Fig. 14

We note as this solution is not the only one. In fact, we can choose also the solution of Fig.15. That does not mean *ambiguity* but *flexibility of the model*, in that is the user that can choose the more satisfactory solution for him.

The search developed has permitted to identify the following points (object of work in progress) as aims of future searches:
a) definition (and implementation) of a Data Definition Language [16] and of the related Data Dictionary, for the GRASS model;
b) formal definition of such D.D.L. and D.D.;
c) integration of user views (with the consequent

treatment of problems, as inconsistencies and conflicts);
d) definition of the primitives of a languages for the handling of the schemes by statistical users.

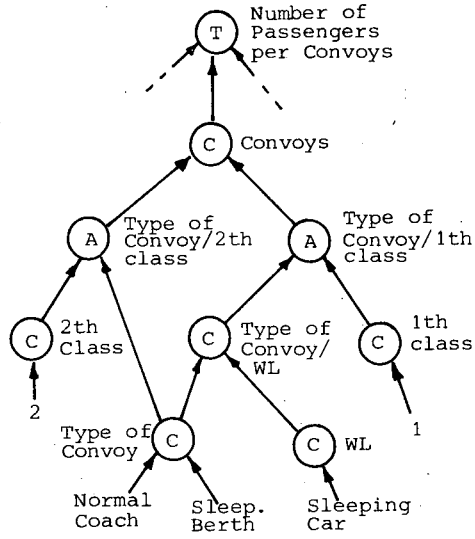


Fig. 15

REFERENCES

[1] H.K.Wong:"*Statistical Database Management*„ International Conference on Management of Data. ACM-SIGMOD, 2-4 June 1982, Orlando, Florida.

[2] A.Shoshani:"*Statistical Databases: characteristics, problems and some solutions*„ Eighth International Conference on Very Large Data Bases, 8-10 September 1982, Mexico City, Mexico.

[3] A.Balsamini, M.Rafanelli, F.L.Ricci: "*GRASS: a Logical Model for Statistical Databases*„ Technical Report I.A.S.I. - National Research Council, n. R-39, October 1982.

[4] E.Cohen, R.A.Hay Jr.:"*Why are commercial Database management systems rarely used for research data?*„ Proceedings of the First LBL Workshop on Statistical Database Management, 2-4 December 1981, Menlo Park, California.

[5] P.Chan, A. Shoshani:"*SUBJECT: a directory driven system for organizing and accessing large statistical databases*„ Proceedings of VII International Conference on Very Large Data Bases, 9-11 September 1981, Cannes, France.

[6] M.J.Turner, R.Hammond, F.Cotton:"*A DBMS for Large Statistical Data Bases*„ Proceedings of the V International Conference on Very Large Data Bases, 3-5 October 1979, Rio de Janeiro, Brazil.

[7] A.Sanò:"*Basi di Dati Numeriche e possibilità di elaborazione*„ Tech. Rep. E.N.E.A., Italy, N. 82, October 1982.

[8] D.Bates, H.Boral, D.J.De Witt:"*A framework for research in database management for statistical analysis*„ Proceedings of the International Conference on Management of Data, ACM-SIGMOD, 2-4 June 1982, Orlando, Florida.

[9] R.Hammond:"*Metadata in the RAPID DBMS*„ Proceedings of the First LBL Workshop on Statistical Database Management, 2-4 December 1981, Menlo Park, California.

[10] S.B.Yao, S.B.Navathe, J.L. Weldon:"*An Integrate Approach to Database Design*„ Proceedings on Data Base Design Techniques, Lecture Notes in Computer Science, May 1979, New York.

[11] V.Mirizio, M.Rafanelli, F.L.Ricci:"*A DBMS for an Epidemiological research about mental diseases and relative hospitalization: initial results and deductions of a preliminary study*„ Third Congress of Medical Informatics Europe, MIE 81, 9-13 March 1981, Toulouse, France.

[12] J.M.Smith, C.P.Smith:"*Database abstraction: aggregation and generalization*„ Transactions on Database Systems, Vol. 2, N.2, June 1977.

[13] H.K.Wong, I.Kuo:"*A Graphical User Interface for Database Exploration*„ Proceedings of the VIII International Conference on Very Large Data Bases, 8-10 September 1982, Mexico City, Mexico.

[14] R.R.Johnson:"*Modelling Summary Data*„ Proceedings of the ACM-SIGMOD 1981, Ann. Arbor, Michigan.

[15] J.L.Mc Carty:"*Metadata management for large statistical databases*„ Proceedings of the VII International Conference on Very Large Data Bases, 8-10 September 1982, Mexico City, Mexico.

[16] M.Rafanelli, F.L.Ricci:"*A Data Definition Language for a Statistical Data Base*„ Technical Report I.A.S.I. - National Research Council (to appear).

STATISTICAL DATA MANAGEMENT RESEARCH AT LAWRENCE BERKELEY LABORATORY

P. Chan, S. Eggers, F. Gey, H. Holmes, P. Kreps,
J. McCarthy, D. Merrill, F. Olken, A. Shoshani, H. Wong

Lawrence Berkeley Laboratory
Berkeley, California 94720

Abstract

This paper summarizes current research on statistical database management issues at the Lawrence Berkeley Laboratory. This research includes development of prototype systems as well as analytic work on user interfaces, physical organization, hardware architecture, and database modeling.

1. Introduction

Lawrence Berkeley Laboratory's data management research program deals with the topic of "scientific and statistical databases" [SSDB's]. Databases created and collected for scientific, socio-economic, and other types of statistical analysis, have requirements that cannot be easily supported by existing commercial data management systems. Such data are prevalent in government (e.g., energy, census, pollution) and in scientific environments (e.g., seismic data, experimental data), but they also exist in industry (e.g., clinical trials, economic time series).

SSDB's have characteristics and usage patterns that require special data management techniques. SSDB's typically contain descriptive information measured data values. SSDB's tend to be large, both in volume and in their number of distinct data elements (entities, attributes). The measured data may contain a large number of nulls (missing data), thus, requiring special techniques for sparse data (such as compression). Repetitive descriptive information and widely varying magnitudes of data values present further compression possibilities.

For large SDB's, users often extract subsets or summaries of the data or summaries over the data for their analytic purposes. Furthermore, data analysis involves many iterations of examining samples, refining the data, and comparison of multiple subsets. This tends to generate a large number of smaller data sets which need to be managed. There is need for techniques to associate subsets of data with the original data, to keep track of their history, and to maintain consistent naming conventions. The large number of data elements presents problems for users. There is too much to remember in terms of data element names, acronyms, codes for data values, permissible data formats, and syntax for data retrieval. Thus, it is necessary to develop user interfaces that can alleviate these problems. Information about the data and its support is referred to as "meta-data management."

The data management research program has four major components that address the specialized problems of SSDB's. In the "user interface" area we are exploring several approaches suitable to users with different needs. The "physical organization" area is concerned with compressing data and accessing them efficiently, as well as managing files on secondary storage. The "hardware architecture" area deals with specialized hardware for SSDB's. The "modeling" area bridges the user interface for physical organization and hardware architecture areas, by dealing with models of both logical and physical data structures. These models allow multiple user interfaces, physical data structures, and access techniques to co-exist in a single system, and they provide the basis for query optimization and processing. The following four sections describe our work in these areas.

2. User Interfaces

Several different projects at LBL concern different aspects of and approaches to user interfaces. The first three subsections below describe prototype user interfaces that eliminate the need for users to remember names, acronyms, formats, and complex syntax rules. The first subsection discusses our work on a graphical user interface for data exploration (GUIDE). The second subsection describes a system (SUBJECT) which is based on the representation of statistical databases as logical graphs. The third describes the user interface for SEEDIS, which uses an "on-line codebook" approach. The fourth subsection describes research on application of graphic design principles to user interfaces. The fifth subsection summarizes a more procedural interface for expert users and system integrators, which is based on self-describing data files and "software tools" that manipulate such files.

2.1. User Interfaces for Data Exploration

The purpose of this research is the organization of information associated with large databases for presentation to users who are not computer experts. It is based on the premise that guided exploration of this information using rich and flexible graphics tools will lead to an effective, easy-to-use user interface for finding and displaying data.

The research focuses on the problem of dealing with databases that are not necessarily large in size, but rather have a large number of data elements and complex

This work was supported by the Director, Office of Energy Research, Office of Basic Research Sciences, Division of Engineering, Mathematical and Geosciences of the U.S. Department of Energy under Contract DE-AC03-76SF00098.

semantic relationships. It has been applied to the area of statistical databases, since they provide good examples of this complexity.

The research is motivated by technological developments with respect to networks and work stations. Large databases are often centralized resources with non-expert users attempting to access them via work stations. In addition, recent developments in the database modeling area lend themselves to the use of graphics for representation of the semantics of data.

The graphical user interface under development is called GUIDE (Graphical User Interface for Database Exploration). There are three major differences between existing work in the area and the GUIDE approach: the GUIDE approach uses a rich underlying model that can be displayed graphically; it supports partial queries, thus enabling the user to issue progressively more complex queries rather than specifying a long complex query in a single step; and it explicitly represents the description of the data, called meta-data.

Meta-data is especially important for the management and exploration of complex databases because these databases often contain hundreds of data element types. Users, even experienced users, cannot remember all the names and descriptions of these data elements. The approach taken in GUIDE is to expressly present the meta-data in graphical form so the user can explore the database without needing extensive knowledge of the structure and types of the data.

2.2. The SUBJECT System

SUBJECT is a continuing project whose purpose is to provide users with simple but effective means of accessing statistical databases. This is achieved by using a specially designed graph structure to represent the logical content of statistical databases. A novice user can browse through the graph for descriptive information about databases, select a database to explore, and continue on to express queries. The user interface represents data in a menu format, thus eliminating the need to remember names, values and formats of data elements.

An alternative to the browsing capability is provided for experienced users, who can search for a data file by keys. They may quickly locate a desired data file, and then proceed to express query conditions in the usual fashion (i.e., by moving around the directed graph). The system also provides access to documentation associated with nodes in the graph.

An important concept of the graph representation is that of "node sharing," which permits more than one arc to point to the same node, thus forming directed acyclic graphs. Node sharing allows for attribute domains to be shared between different files, providing several advantages: eliminating duplicate data values; consistency of naming (where items that are the same, but reside in different files, are forced to have the same name); and allowing the specification of join domains between files, permitting multiple physical files or fractions of these files to be viewed jointly as a single logical entity.

The system provides an interactive facility for specifying the structure of a SUBJECT graph, including the specification of shared nodes. This facility is integrated with the browsing facility to let users browse existing portions of the graph. In addition to supporting the construction of multiple levels, it is possible to connect to a substructure of an existing graph. This provides a convenient way to join files.

Another facility is the "data editor" which allows interactive data entry and modification of data values. The editor prompts the user with the combination of parameters (category values) for which a data value has to be entered.

2.3. On-line Dictionaries in SEEDIS

Recent research on user interface design suggests that novice users find it easier to use systems which simulate or employ analogies to more familiar non-computer objects (e.g., desk tops, filing cabinets, folders, etc.). Data item selection in SEEDIS employs a similar approach, which has proved quite popular and successful with users.

To select data items, SEEDIS users browse through an on-line data dictionary which resembles a printed codebook, complete with page numbers, table of contents, indexes, and footnotes. In addition to data item information, SEEDIS dictionaries contain documentation on data sources, how data were collected and installed, persons to contact for help, etc. Users can browse data dictionaries in any order, using the carriage return key to move ahead page by page, a page number to skip back and forth, and line letters to choose data items during the course of browsing. Users select data by typing single letter codes corresponding to line identifiers on the current screen of information. Software translates these temporary line identifiers into database and data item codes. Users can thus select data items from a number of different databases, and the retrieval software automatically takes care of combining them into a single working dataset.

2.4. User Interface Design Principles

Many user interfaces for computer systems contain unintended oversights and errors in visual communication, such as confused composition, poor typographic hierarchies, and color combinations that inhibit legibility. Application of graphic design principles to the visible language of an interface (i.e., typography, symbolism, spatial organization, sequencing, and color) can improve communication of information to users. As part of the SEEDIS project, professional graphic designers with experience in computer techniques have sought to apply graphic design principles from print and film media to computer output on video display terminals and hard copy devices. This research has identified goals for user interface design (e.g., to aid learning of complex information, to facilitate memorization of key procedures, to encourage accurate decision-making, to build a clear

conceptual image of the system), as well as general guidelines to meet those goals. The set of guidelines being developed are based on the modern Swiss utilitarian (programmic) design tradition, a grid-oriented approach eminently suited to information display in which many complex relationships must be distinguished clearly and carefully.

User interface design involves selection of symbols and formats for the standard functional components of a system: menus, prompts, help messages, status reviews, etc. It also involves detailed specification of standards at a lower level: determination of a layout grid; selection of typographic styles, sizing, spacing, and means of emphasis; standard treatment for continuous prose (e.g., help messages), interrupted prose (error messages, system status reports, etc.), and tables/lists (menus, data dictionaries, etc.).

Unlike conventional prose texts, user interfaces have many components and corresponding layouts. These include tables, indices, lists, numbered items, diagrammatic presentations, explanatory notes, and pictorial images. The user interface is not intended for continuous reading as for prose text, but rather is a framework for complex movement with constant shifts in levels of instruction to the viewer and frequent distractions to the viewer's attention.

These design considerations are more than just "cosmetics." By carefully considering not only what to show, but also when, how, and why to show it, a better understanding of the functionality of the system emerges in the minds of the builders, the users and the viewers of a system and its information.

2.5. Codata Tools for Expert Users

Another aspect of the SEEDIS project has been research on tools for expert users of statistical databases. This research has explored the application of user interface ideas from UNIX and the Software Tools to self-describing data files.

The Codata Tools are a set of programs which read, write, and restructure self-describing Codata (common data format) files. These tools manipulate both data and data description, so that the the output of any operation is itself a Codata file. Semantics of results and descriptions of derived Codata files are inherited from descriptions of input Codata files. Following the Software Tools philosophy, the Codata tools are modular - each tool performs a specific limited task. They follow the UNIX and Software Tools conventions of standard input and output. The output of any module can automatically serve as the input of another, and they can be "pipelined" or "chained" together.

Codata tools can be used to extract specified rows and/or columns from a file, to sort a file, to perform relational joins, to perform tabulations by aggregating on common key values, and to perform other operations. The Codata tools are written in RATFOR (a transportable FORTRAN preprocessor), and can be easily adapted to run on any computer where the Software Tools have been implemented. Work is currently under way on substantial enhancements to the Codata file format and the Codata tools to provide for more efficient physical storage formats, more complex data structures, and more extensive, open-ended data description.

3. Database Modeling

Three areas of database modeling are discussed below. The first area described in the section entitled "Semantic Core Model" is concerned with the representation of conceptual entities and their relationships. The second area described in the section entitled "physical modeling," is concerned with the representation of physical database structures and their characterization. The third area described in the section entitled "meta-data management" is concerned with representation of data about data, or meta-data.

3.1. Semantic Core Model

Scientific and statistical database (SSDB) applications present formidable modeling requirements that tax the power of conventional record-oriented database models (such as the relational or CODASYL models). Since data are viewed in terms of the structures in which they are defined and the operations performed on them, the available model exerts a powerful influence on the ability of applications to deal naturally with the data. At issue is the size of the gap between a naturalistic model and the database system-imposed model.

In SSDB's many entities of interest are actually inferred or derived from data pertaining to other more concrete or measurable entities in the world. The boundaries of such abstract entities often cannot be well defined.

Our own experience with statistical applications (mostly socio-economic, environmental, and demographic data) and with scientific applications (seismic data) confirms this observation. For example, information about seismic events is generally not gathered directly at the source. It is derived by an elaborate process of analysis from ground motion signal data measured at numerous sensor sites remote from the seismic source. Furthermore, this analysis can be a multi-stage iterative process. First the sample (time-series) data streaming from the sensors are analyzed to abstract inferences about the arrival of propagated signals at the sensor stations. These "arrivals" are then collectively analyzed together with a model of waveform propagation through regions of the earth to determine the existence and approximate parameters of an hypothesized seismic event. The hypothesized event parameters can be used to iteratively refine the set of arrivals, which are then used to further refine event parameters, and so on. Because steps in this process are so computationally intensive, there is a need to preserve computed values and intermediate results whose relationship to the raw data must be maintained.

SSDB's are also often the product of heterogeneous data sources requiring integration and assimilation. It is

often not possible or practical to force all data into a common format. It may also be a requirement to preserve (or recover) the original view of the data. Thus, integration may be best implemented via some virtual interface mechanism where data is mapped into one or another view as appropriate.

Based on our observations of SSDB's in practice and as reported, we have concluded that conventional data models and database systems provide inadequate representational tools for these applications. These and other considerations have led us to develop a new model for representing SSDB's which we call the *Semantic Core Model* (SCM). This model is based largely on the semantic data abstraction models found in the literature. However, none of the extant models address all of the issues raised earlier regarding SSDB's.

A data definition syntax for most of the constructs proposed in the SCM has been developed through several iterations. These constructs have been used to experimentally model several applications of SSDB's.

3.2. Physical Database Modeling

This work is intended to support implementation of the semantic core model. There are four phases to physical database modeling: descriptive, analytic, query optimization, and database synthesis.

The descriptive phase consists of constructing a parameterized taxonomy of physical data structures used to implement the database. The descriptions must specify both the structure of physical data structures and their placement, since the placement can affect also the cost of i/o which needs to be done. While there has been considerable work done on the structural specifications of physical database structures in the literature, placement specifications have received much less attention.

The analytic phase consists of constructing formulas to estimate the cost of processing a specific query, using a known computational strategy, against a particular set of data structures.

The query optimization phase is concerned with searching the space of query computation strategies to find the cheapest one. In addition to models of query computation and cost, the query optimizer requires a search strategy.

The final phase of physical database modeling is concerned with the physical design of databases.

Work in this area is only in initial stages. We will first concentrate on the descriptive and analytic areas. In particular, the descriptive work will consist of extending the semantic core model specification language to include specifications of the structure and placement of physical data structures. We are particularly concerned with describing "clustered transposed files" and compressed array linearization storage structures (these structures are described in the section below on Physical Organization).

Our analytic work will build on the existing extensive literature. Different data structures and query operators for statistical applications will require attention to issues often neglected in conventional analyses. The analytic model will include decompression, recompression, and tuple assembly costs, aggregation operators, and output reordering.

3.3. Meta-Data Management

Statistical databases frequently contain hundreds of distinct attributes or variables. Many new variables are created during the course of data exploration, manipulation, and analysis. Users and software need many different kinds of data description or meta-data (e.g., data type specifications, missing data codes, attribute names, category value labels, etc.) in order to deal effectively with statistical data. Since this meta-data can be quite voluminous and since it differs considerably from statistical data in terms of content and structure, meta-data management presents a number of important questions.

LBL research in this area is concerned with content, physical characteristics, operations, and users of meta-data. It seeks to identify different types of meta-data, particularly those used in statistical analysis. It analyzes different ways that meta-data are used by people and programs -- end users, database administrators, database management software, and application programs. It explores the logical and physical structure of meta-data, which spans a whole range of data types including numbers, text, mathematical equations, etc. It also studies types of operations which are especially important for meta-data in general and statistical meta-data in particular (e.g., keyword indexing, category set mapping, attribute inheritance).

The SEEDIS Project has provided an opportunity to test various meta-data management ideas. A prototype extensible data definition language has recently been implemented in SEEDIS to provide a standardized, unified source for program information (e.g., data types, physical storage locations, and missing data specifications), user documentation (on-line and printed data dictionaries) and data labeling (e.g., variable names, data value labels). Programs for data loading, compression, extraction, manipulation, report generation, and codebook creation all make use of the same basic data definition language. SEEDIS has also demonstrated the importance and utility for statistical databases of self-describing data files and software tools to manipulate such files, as described above under "User Interfaces."

4. Physical Organization

The following sections describe work in compression and file management. The first section discusses continuing work on compression for statistical databases. It is concerned with achieving high level of compression while providing fast access to the compressed data. The second section deals with algorithms to rearrange database attributes in order to maximize the compression factor. The

third section discussion file management issues, including tuple partitioning, data caching, and distributed file management. The fourth section describes automatic file migration replacement policies for moving files from secondary to tertiary storage devices and vica-versa.

4.1. Compression for Statistical Databases

The storage and transmission of very large databases often constitutes a significant portion of the cost of managing them. Compression of data becomes increasingly important as the volume of data grows. Numerous techniques have been devised which are capable of compressing a variety of databases to varying extents.

The compression techniques originally developed for SEEDIS (described elsewhere in this Proceedings) combine a form of run-length encoding with a computer-independent, variable-length representation of data values. These techniques permit SEEDIS to reduce the amount of disk or tape storage required for statistical data files, such as those from the U.S. Census Bureau, to from 20 to 50 percent of their original volume. Partitioning and indexing of SEEDIS compressed data files is based upon geographic areas to which the data pertain, so retrieval is efficient for geographic selection criteria.

However, if selection criteria pertain to attributes which are not indexed, most compression techniques, including those currently used in SEEDIS, require that a database be serially decoded before being searched, a process which requires linear time. In order to overcome this limitation, techniques have been developed which achieve compression ratios comparable to linear time algorithms but in which the access time required for searching non-indexed values is logarithmic.

The new compression scheme, called header compression, has the capability of both eliminating multiple types of constants from the database and compressing each stored value to its minimal byte length. The scheme is a variation of run-length encoding, in which modified run-lengths are extracted from the data stream and stored in a header. The header is used to form the base level of a B-tree index into the database. The run-lengths are cumulative, and therefore a logarithmic search algorithm can be used to obtain an access time which is logarithmic in the size of the header. Two versions of the header compression scheme have been implemented, called the basic and general versions. The basic version compresses only a single type of constant (e.g., zero), while the general version compresses both multiple types of constants and stored data to their minimum byte length. These particular versions of header compression were chosen because they represent the extreme cases in the tradeoff between functional capability and degree of compression and access time, and because they have the widest applicability.

An integrated design of the two versions of header compression takes advantage of many overlapping functions. As a result, about 60% of the code is shared between both versions. The implementation includes modules for loading the data and building the corresponding system catalogues and B-trees, for accessing the B-trees and partitions of the data, and for reformatting of the data and storage allocation for the output.

Future plans include benchmarking of the compression scheme against the current SEEDIS compression scheme and others.

4.2. Rearranging Data to Enhance Data Compression Efficiency

In the course of the work on data compression, we noticed that vertical partitioning of the data (i.e., storing each domain (or column) in a separate file) markedly increased the effectiveness of run-length type data compression schemes such as header compression. If one thinks of the data as an array indexed by tuple-id and domain-id then one can think of vertical partitioning as a transposition operation. These observations led us to consider the impact of other rearrangements of the data upon the efficiency of data compression. For example, if we are storing population counts in an array indexed by race, sex, age, and county then it would be reasonable to assign adjacent codes to the least populous races and counties, because it tends to maximize the probability of consecutive zero counts. The header compression scheme mentioned above, can then be used to compress the consecutive zero counts more efficiently. Furthermore, race or county would be plausible variables for the least rapidly varying index and sex for the most rapidly varying array index.

There are two subproblems involved. One is the assignment of ordering to category values, (i.e., reordering rows or columns of the matrix). The second is the ordering of the category attributes within the indexing function (i.e., transposing the matrix). Thus far we have only made progress on the first problem.

For the sake of tractability we have turned to probabilistic models of the data instead of exact deterministic models. We were able to obtain two analytical results for the value ordering problem. Both results, which are achieved under different assumptions, show that the value ordering should be according to the familiar "pipe organ" assignment, or a variation of it.

We were not able to achieve analytical results for the attribute ordering problem. We plan to use simulation techniques for this problem.

4.3. File Management

Because statistical databases are frequently large, and because they sometimes must be partitioned into separate physical files, at different locations, distributed file management is another important research area at LBL.

Statistical data often are organized in nested hierarchies of entities (such as geographic areas, or types of patients). These hierarchies provide a natural way of partitioning data using simple naming conventions within a standard hierarchical file system. SEEDIS incorporates a hierarchical file system as part of its over- all data model. This hierarchical file organization also extends to

files stored on tape; tapes contained tables of contents which permit retrieval of files or entire directories by logical pathname rather than sequential file number.

In order to extend the hierarchical file model to a multi-site system, SEEDIS has been enhanced to provide automatic access and disk caching of tape files from a remote automatic tape library as well as distributed data elsewhere on the network. Each SEEDIS node on the network is independent. System managers at each site can decide where to put their own physical files and which files should be shared with users at other nodes. Databases dictionaries and file location tables are automatically distributed to other sites periodically. System logs are kept to provide information on data request file migration, cache purging, etc,; these will help provide empirical evidence to test analytic results regarding optimal file migration strategies.

4.4. Automatic File Migration

This work is concerned with modeling and managing the automatic movement of files between secondary (disk) and tertiary (tape) storage. We assume that files are brought into the disk *cache* from tape when they are referenced, i.e., *demand fetch*. A *replacement policy* is used to choose files to be evicted from the cache so that there will be sufficient room for incoming files.

This work is based upon a replacement policy STOCHOPT originally proposed by A.J. Smith. In Smith's model one assumes a fixed rental charge per unit of disk space and a charge to fetch a file from tape. Smith also assumes that the time intervals between successive references to the same file are characterized by known probability distributions, referred to as inter-reference time (IRT) distributions. The STOCHOPT policy chooses the time to hold the file in the cache so as to minimize the expected cost of the next reference (i.e., the storage rental charge for holding the file in the cache until referenced or evicted, plus the tape fetch cost if the file is referenced after it has been evicted). Smith constructed empirical distributions for various classes of files and performed the minimization numerically.

In our work we have shown that for a certain classes of inter-reference distributions one can analytically determine the optimal cache holding time for STOCHOPT for a particular file. Furthermore we have shown that this result holds even for improper inter-reference time distributions (i.e., those for which there is a nonzero probability that the file will never be referenced again).

The *hazard rate* of the IRT distribution at time t is the conditional access rate, i.e., the probability (rate) that the file is referenced at time t since last reference given that it has not been referenced again up to time t. In reliability theory the *hazard rate* is often referred to as the *failure rate* because it denotes the rate at which components which survive t time units fail.

Our analysis indicates that if the hazard rate is monotonically decreasing, then the optimal holding time can be characterized in terms of a scaled hazard rate (i.e., the hazard rate divided by the file size). Files should be held in the disk cache as long as the scaled hazard rate exceeds the unit storage rental charge. Empirical studies by Smith suggest that the assumption of monotonically decreasing hazard rates is reasonable, i.e., the longer the time since last reference to file the lower the rate at which it is referenced. This HOPT replacement policy has fixed space analogue HMIN which simply evicts the file with the smallest hazard rate whenever it needs some space. HMIN reduces to LRU if all files have the same size and IRT distribution.

5. Hardware for Statistical Databases

The Microprocessor Assist System (MAS) is a research database machine that is especially designed for statistical data management. The MAS consists of (one or more) trees of microprocessors that are at the bottom level connected one to each disk (the *leaf* microprocessors) and at the highest level (the *root* microprocessor) connected to the front-end computer. A single root microprocessor directs the activities of, and receives data from, its child processors. The tree can be more than two levels, and there may be more than one tree connected to the same front-end. However, for simplicity, in its initial implementation, a two level single tree design is used. The front-end system is assumed to be a minicomputer with equivalent functionality of a DEC VAX 11/780.

The major functionality of the microprocessors is to implement compression and attribute partitioning techniques for the statistical data management system (SDS) running in the front-end computer. We are exploring the use of multiple microprocessors in a database machine to handle two I/O related functions: compression/decompression and attribute assembly/disassembly.

The MAS obtains its performance benefits from the parallel operation of multiple microprocessors. The leaf microprocessors schedule the disk reads, read the required blocks, decompress the data, and send the required data up to the level above; the higher-level microprocessor assembles attributes that are spread across multiple disks (hence multiple microprocessors). Fully assembled tuples are sent from the root microprocessor to the main computer. The fundamental function provided by the MAS is to make compression and partitioned attributes invisible to the front-end system.

Software for the MAS has been designed. It is a relational data management system, where the functions performed at the disk-level microprocessors are simple restrictions, attribute assembly and decompression (for retrieval); disassembly and compression (for storage). Two possible prototype applications are being explored: time-series, instrumental physical data from the Time Projection Chamber (TPC), and mixed textual and numeric data, from the Particle Data Group. It is expected that one, or both, applications will be brought up in prototype form on the MAS.

Hardware for the MAS has been partially defined. The microprocessors will be M68000's; the decision of which 68000 systems to use depends on results of modeling the system, and on the potential vendors actually delivering promised systems to customers. It is our purpose to develop generally useful methodologies for building and using multi-microprocessor systems, so we are not interested in one-of-a-kind systems; hence, we will use commercial, well functioning hardware.

6. Conclusion

The area of SSDBs offers interesting and challenging opportunities in data management research. Most of the traditional issues in data management still apply here, but because of the nature and characteristics of SSDBs, different (sometimes drastically) techniques are required in data modelling, user interfaces, physical structures, as well as in hardware.

7. References

1. 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.
2. Denning, D., Nicholson, W., Sande, G., Shoshani, A., "Panel Report on Statistical Database Management," *Second International Workshop on Statistical Database Management, 1983.*
3. Eggers, S. J., Shoshani, A. "Efficient Access of Compressed Data," *Proceedings of the International Conference on Very Large Databases*, 6, 1980, pp. 205-211.
4. Eggers, S., Olken, F., Shoshani, A., "A Compression Technique for Large Statistical Databases," *Proceedings of the International Conference on Very Large Data Base (VLDB)*, 1981, pp. 424-434.
5. 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.
6. Gey, F. G., J. L. McCarthy, D. Merrill, H. Holmes, "Computer-Independent Data Compression: A Space-Efficient, Cost-Effective Storage Technique for Large Statistical Databases, *Proceedings of the Second International Workshop on Statistical Database Management* September, 1983.
7. Hawthorn, P., "Microprocessor Assisted tuple access, decompression and assembly for statistical database systems," *Proceedings of the International Conference on Very Large Data Base (VLDB)*, 1982.
8. Johnson, R. R., "Modeling Summary Data," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1981, pp. 93-97.
9. Johnson, R. R., "A Data Model for Integrating Statistical Interpretations," *Proceedings of the First LBL Workshop on Statistical Database Management*, December, 1981, pp. 176-189.
10. Kreps, P. "A semantic core model for statistical and scientific databases," in *LBL perspective on statistical database management*, H.K.T. Wong, ed., Lawrence Berkeley Laboratory report LBL-15393.
11. Marcus, A., "Graphic Design for Computer-Based Information Management: A Case Study of SEEDIS," Lawrence Berkeley Laboratory Technical Report #16330.
12. McCarthy, J. L., "Meta-Data Management for Large Statistical Databases," *Proceedings of the International Conference on Very Large Data Bases, (VLDB), September, 1982*, pp. 234-243.
13. 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.
14. Merrill, D., J. L. McCarthy, F. Gey, H. Holmes, "Distributed Data Management in a Minicomputer Network: The SEEDIS Experience," LBL Tech. Report #15705, *Proceedings of the Second International Workshop on Statistical Database Management* September, 1983.
15. Merrill, D., and J. L. McCarthy, "Codata Tools: Poetable Software for Self-Describing Data Files," *Proceedings of Computer Science and Statistics: 15th Symposium on the Interface*, Houston, Texas, March, 1983.
16. Olken, F., "HOPT: A Myopic Version of the STOCHOPT Automatic File Migration Policy," *1983 SIGMETRICS Conference*, (LBL-15554).
17. Shoshani, A., "Statistical Databases: Characteristics, Problems, and some Solutions," invited paper in the *Proceedings of the International Conference on Very Large Data Bases, (VLDB), Sept. 1982*, pp. 208-222.
18. Wong, H.K.T. and Kuo, I., "GUIDE: Graphical User Interface for Database Exploration," *Proceedings of the International Conference on Very Large Data Bases, (VLDB), Sept. 1982*, pp. 22-32.
19. Wong, H.K.T., "A Graphical Query System for Complex Statistical Database," *Computer Science and Statistics: Fifteenth Symposium on the Interface*, March, 1983.
20. Wong, H.K.T., Editor, "A LBL Perspective on Statistical Database Management" LBL-15393, December 1982 (Contains most of the publications mentioned here plus few additional papers).

A Statistical Database Component of a Data Analysis and Modelling System: Lessons from eight years of user experience.

John C. Klensin
Laboratory of Architecture and Planning
Massachusetts Institute of Technology

When development of the Consistent System, a large-scale data analysis and modelling system for the social, policy, and behavioral sciences, began, it was determined that database management facilities would be necessary to handle the variety of complex data that were anticipated. The primary data base management component of that system, called Janus, has been in use, in two prototypes and then in production form, at several sites and by a variety of users, since about 1975. This paper reviews some of the original design considerations about Janus (including its relationship to the rest of the Consistent System) and reflects on them in the light of user experiences and comments in the subsequent years.

In 1969, MIT started a major effort to consider issues in the design of tools and environment for social and behavioral scientists. That effort was very active for about eight years, and also involved researchers from several other institutions, especially Harvard, and a variety of disciplines. A summer study held in 1970 produced a menu of recommendations about the facilities that systems would require in the future. Among the more important of those conclusions was that there was a need for serious facilities to manage the types of data that would eventually be processed statistically -- facilities that, to different researchers, meant

- Processing of "waves" of surveys
- Management and retrieval of very large bodies of information
- Making inferences and doing analysis from data bases containing data at multiple levels of aggregation, moving back and forth among aggregation levels (rather than simply looking at the top or bottom level of a hierarchy).
- Dealing with missing values in ways that remained consistent across different datasets and sources of information.
- Handling data of different types - nominal, multiple-response nominal, integer, real, dates and times, and text.
- Handling aggregate data types and variable number of responses per subject in all the usual modes: statistical summaries, counting, adding, taking the first (or the last) and so forth.
- Being able to make arbitrarily structured queries, retrieving on any attribute, or combination of attributes, in the database.
- And, being able to take any data, or combination of data, and subject them to a full range of transformations and statistical analysis techniques.

A data management system, known as "Janus" was developed to meet these goals as a "data management front end" for a more comprehensive analysis and modelling environment that is now known as the "Consistent System" (Dawson, Klensin, and Yntema 1980). Janus went through two complete prototyping periods with user and technical review before work on the present system began. The present version went into active use in about 1976 and has now survived all of its original designers and developers.

The author of this paper and his current colleagues in maintaining and developing the Consistent System were not among the designers or developers although they were associated with the project during that period. We were not and, indeed are not, very happy about several aspects of the design. We find ourselves today in the somewhat uncomfortable position of writing the history, not by being the victors, but by having more staying power for reasons that are perhaps artificial.

In any event, we have now accumulated several years of experience with a variety of users of statistical databases -- students, academic and private researchers, governmental and commercial analysts, and even some users of commercial databases - and a variety of databases ranging from simple surveys to personnel files and from records of municipal employment and finance to records of medical incidents to records of radio listening and international crises. We have learned which facilities are heavily used and which lightly, which capabilities are important and which ones are absent or clumsy, and where the major problems lie.

In this paper, we intend to look through the design of Janus as it has evolved from the original criteria. We will focus on those aspects of the design that relate directly to the manipulation of data, e.g,. operations on datasets and how the data are made available for statistical analysis. As we examine a feature in this area, we will discuss the reasons for it and the degree to which it appears to us to have succeeded or failed.

We do not intend to spend much time on the internal organization of Janus or of the Consistent System more broadly. The former topic was discussed in some length in a preliminary paper on Janus by the designers, some time ago (Stamen and Wallace 1974). While the terminology now in use has evolved from that in the paper, and some of the features discussed were not implemented initially (a few never have been), the paper gives a reasonable overview of the internal operation of the system. Indeed, we still use a marked up copy of that paper to introduce newly-hired maintainers to the system.

The author would like to acknowledge the helpful critical comments of several colleagues, especially Ree Dawson, and users of the Consistent System, without which the paper would have been less complete and less clear. The inferences drawn remain the author's responsibility.

A few characteristics of the system's style and design may be helpful in understanding our perspective. First of all, and most important, Janus is command-driven rather than a host language system. It has the ability to be called from a host language, but those capabilities were added very late in the implementation, are less powerful than the command-level capabilities, and have been little-used. Not only are queries, reports, and construction of new data elements done with commands, but database definition and creation are also. Second, the designers were convinced, based on prior experience, that use of a statistical database was characterized primarily by two types of operations, both of which were less common in commercial databases: creating a new variable in an existing dataset as a transformation or combination of existing variables and retrieving (for either analysis, computation, or display) all or most of the values of a few variables (compared to the total number in the database) rather than all or most of the values of a few records (compared to the total number in the database). (1) Third, while the system utilizes some unusual terminology and operation forms in deference to its audience and the preferences of its designers, it is essentially relational in nature -- another one of those "relational, but" and "relational plus" systems. The "but" here is the absence of either host language capability or tools for the DBA, and some peculiar terminology. The "plus" is the ability to use some unnormalized forms explicitly and the presence of some operators and functions from the statistical, rather than database, domain. The first of these brings with it the attendant need to be able to name the mappings between relations where more than one mapping can exist between a pair of relations. In addition, for a variety of reasons, it is often convenient to treat the mappings themselves as objects. We will come back to the user appearance of these features later. (2)

The User's View

The user of data with simple structure sees a single dataset that looks very much like a rectangular data matrix. When things are displayed, the entities look like subjects and the attributes look like columns of that traditional matrix. The underlying relational scheme, when contrasted with experience with traditional statistical systems, causes two related surprises that some users have found disconcerting: there is no assumption of either ordering in the attributes (a "next to" relationship) nor in the entities (an "after" relationship). The former leads to the desire to talk about "variables 10 through 30", for which Janus has no equivalent concept. (3) The latter leads to two things: Users want to talk about entities 35 through 40, which is only rarely substantively meaningful. (4) More important, users keep perceiving the need to sort a dataset prior to making an analysis involving a selection or subset of cases. This is never actually necessary in conventional analysis and, since it requires creating a new relation (dataset), is fairly costly compared to just selecting the appropriate entities. But, when people have learned for years that this sort of operation is necessary, we have a great deal of trouble with ingrained habits.

The cells of the data matrix or relation may themselves be drawn from the domains of vectors or matrices of values. This feature has been important in several applications, but is not heavily used. It would have been more heavily used, we believe today, had the vectors been able to carry enough label and periodicity information to make effective representations of time series of moderate length. By use of lower bounds on vectors that are other than one, that capability has been simulated effectively in a few databases, but never satisfactorily.

In addition, the cells may contain variable-length lists of values (an unnormalized form with all of the associated problems). Those lists can be used to represent variable numbers of responses per subject. They also come into being as a result of some Janus operations, as discussed below. They have, as one might reasonably predict from the relational database literature, caused a great deal of trouble for both users and implementers. A complete and satisfactory set of operations is extremely difficult to define for them. At the same time, they have provided some capability, such as the ability to use the system efficiently for information retrieval queries by keywords and one model for handling multiple-response data, that is of great value.

The operations and commands

We could have accommodated multiple types of data representations -- scalars, vectors, and lists, nominal, integer, real, text, and date-time values -- without the considerable investment required to build a DBMS. Some of the variability and heterogeneity of types are a nuisance in a conventional statistical system, but it is possible to incorporate provision for them, as some statistical systems have started to do within the last few years. The project specified a stronger database capability because its participants saw a need to deal with multiple datasets at the same time: datasets at potentially different levels of aggregation, or drawn from different sources, or requiring substantive, rather than mechanical, merging processes. Codd's work on the relational algebra (Codd 1971) was just becoming known, and the designers of Janus drew on one of the early relational experiments (Goldstein and Strnad 1971) as well as the successes and problems of three systems with which they had had extensive experience: Data-Text (Armor 1969), ADMINS (McIntosh and Griffel 1970), and DATANAL (Miller 1967, Miller 1968). One of the principal difficulties with those systems was precisely the inability to deal with multiple datasets simultaneously. This was a need that kept arising no matter what sorts of "merging" facilities were designed for these systems.

Janus's real capabilities arise when a second dataset (relation) is introduced. The second, and subsequent, datasets, can be introduced in any of three ways:

- As additional externally-produced values to be merged with those in an existing dataset on (substantially) an entity-by-entity basis.
- As additional externally-produced values collected at a different level of aggregation from those in existing datasets.
- As a derivative of existing datasets.

These operations and their implementation in Janus are discussed in the next few sections.

Adding groups of new values

In the original implementation, new cases were added by the use of basic dataset creation operations (taking the union of the two existing relations with respect to one or

more attributes). Users requested that we add specialized commands for the purpose in order to reduce typing and cognitive aggreviation. The resulting commands permit specification of sets of attribute values to be appended (other values are set to missing), while the union operation copies only those attributes involved in the set operation; other attributes had to be copied individually by separate commands.

More specifically, when values are to be added to a Janus database that represent some or all of the existing attributes for a new set of subjects, a new dataset is created (from the raw data) representing those subjects and attributes and then appended to the existing dataset for the existing subjects. If the new dataset does not contain some of the attributes in the existing one, the values for those attributes are automatically set to missing in the appending process. Users have had no difficulties with this feature, although the fact that the appending process is not sensitive to the order and column positions in which raw data appear continues to astonish some of them for a long time.

Merging and updating information

Janus deviates from the traditional statistical package model in its approach to merging data. When values from two sources, but representing the same subjects, are to be merged or updated, separate datasets are again created, each one representing one of the sources and each one containing subject identification information. (5) A mapping is then established between the two datasets and the attributes of one copied or updated from the other. This operation is approximately equivalent to a traditional "join", but has seemed more sensible to the casual user-analyst. Since relations are stored transposed (by attribute), there is no performance penalty. If any subjects do not appear in the new (update) dataset who do appear in the source one (the one to which attributes are being added or updated), the values of the attributes for those entities are set to missing. It is possible to query the database and display entities in either dataset that do not participate in the mapping (that would, therefore, lead to either missing values or lost information). It is also possible to form new relations as the intersection or difference between a pair of existing datasets as a means of detecting these problems. The query, display, and inspect capability is used by almost all users involved in updating and merging operations; the intersection and differencing operations have, to our surprise, been little used.

Again, these are relatively recent facilities. Prior to their addition, users went through a series of steps to locate and update values. Based on complaints from users before the aggregate commands were added, designers of future systems should probably include similar capabilities. This appears to be true even though only the more primitive ability to infer individual attributes from one relation is logically necessary.

None of these facilities should be confused with those for simply editing individual data values, or sets of values with common characteristics, by specifying a replacement value. Simple editing facilities of that sort have existed in Janus from the first prototype and are important for many purposes, especially in an interactive system. They cannot be used to substitute for the dataset-merging facilities discussed above, nor can the dataset-merging facilities be reasonably used as substitutes for simple editing.

Changing the level of aggregation

Statistical data are typically updated less frequently than data managed for commercial purposes and the operations described above are of less importance to traditional statistical applications than to commercial or mixed approaches. More crucial to the needs of the analyst faced with multiple sets of data from different sources is the ability to reconcile different levels of aggregation either between relations or within a single relation. These different levels are often the traditional strict hierarchies -- people, in buildings, in city blocks, in cities, in states -- but may also represent much more complex (and non-hierarchical) relationships. For example, we have been involved recently in the creation of a database of all international crises since approximately the end of the second world war (Farris et. al. 1980). That database includes a relation whose entities are crises and one whose entities are phases of the crises (a strict hierarchy). But it also requires, for analysis, a relation whose entities are actors -- parties to, or participants in, the crises -- and that relation has many-to-many characteristics with respect to information about crises and phases of crises. Janus handles the mappings that reflect relationships among relations in an explicit fashion, permitting naming these mappings and treating them as objects. The map-forming operation is, essentially, a matching one between attributes or tuples in different relations. The principal user so far of the database described above found it surprisingly easy to use, given the intrinsic complexity of the data, but often has trouble figuring out what to map onto what. It is not terribly unusual to have several different maps between a pair of relations in the effort to isolate specific information.

Once a map or maps are formed, the user determines the level of aggregation at which analysis is to be performed. Asking questions in a relation whose entities are households is likely to yield information about households; asking the same type of questions in a relation whose entities are communities will probably yield information about communities. To reduce the user confusion that might otherwise result, Janus utilizes the notion of what we might consider a reference relation -- a "default dataset" relative to which operations are being performed. While operations mapping between existing relations, and operations that create new ones are, of course, exceptions, the user thinks of himself as operating "in" one relation at a time. We have noticed that this seems less confusing than the usual situation in systems based more closely on the relational algebra. (6)

An example may help clarify this level of aggregation situation. Assume that a database contains information on cities, information on households (including what city each is included in), and information on persons (including the income of each and what household each is associated with). By fairly conventional joining and tallying operations, we can determine how many people are in each household, what the aggregate (and average) household income are, and the same information for each city. The average per-capita city income (a question we would want to address in the relation whose entities are cities) is going to be quite different from the average per-capita household income (a question for the relation whose entities are

households). Assuming that the necessary mappings exist (they can be created or composed easily), it is feasible to ask either question, but they have different substantive meanings.

In this example, we are moving information and aggregating along a many-one mapping. There are two sets of possible complications here. If we permit an individual to be listed in more than one household (consider the answer to the question "where were you yesterday?"), we may or may not want him counted twice in averaging incomes. To avoid double-counting, we must reduce the relation of individuals (i.e., person-household pairs) to one that contains only unique people. Assuming that relation ends up with fewer entities than the person-household one, we must choose how to treat the other information on the subject before mapping them into the new relation of individuals. In the original design, Janus permitted adding the information up, taking the mean of the values being mapped onto each entity, and taking the maximum and minimum of those values. We have found it necessary to add two new choice functions -- one arbitrary and one that selects a value (such the income for a subject) only if all candidate incomes are the same. These permit dealing with the problem that arises when several entities in one relation map onto an entity of another relation and we wish neither to aggregate nor to end up with a list of values.

We have spoken about the use of these facilities strictly in the direction of transferring information "down" many-one mappings, which we have found to be the most common case, (or when I discussed merging and updating information, transferring information "across" one-one mappings). The need periodically arises to move information "up" a many-one mapping, or "across" a many-many one. The first of these two operations involves disaggregating or "spreading" information from one relation onto the subgroups of another relation to which that information applies. In the election example that I will discuss later, it was necessary to spread biographic information on candidates onto the relation of votes (where most candidates appeared multiple times) because the information level of the vote values proved useful in looking at that information.

More important, we sometimes do not wish to aggregate or disaggregate as we move up or down a many-one mapping. If we were interested in the number of households containing at least one person over 65, we could take the maximum age of the people who mapped into each household and then count the number of households in which the maximum age was greater than 65. If, however, we wanted to know how many households contained someone named "Fred" we would have two choices: we could create an attribute in the relation of people whose value was, say, one if the person's name was Fred and zero otherwise, add this up (or take its maximum) through the mapping, and then proceed as if Fred-ness were age. Or, we could copy into each entity of the household relation the first name of all of the people in that household. That would give us a variable-length list of names for each entity, the problematic nasty unnormalized form mentioned earlier. Are the two approaches equivalent? This depends on what you intend to do next. We have observed that, for the social scientists and statistical analysts who make up much of our user community, "tell me about those households that contain a 'Fred'" is immediately followed by "tell me about those households that contain a 'Tom'", and that is followed immediately by queries about "Dick" and "Harry". Creating all of the attributes needed for the first approach is, at best, tedious. The use of these lists instead permits the user to ask the questions in a more direct way without moving back and forth repeatedly between the two relations.

As mentioned before, there are problems with a scheme that permits the results of inferring through mappings to be stored explicitly in the relation. They are problems to which, at various times, we have thought we understood the solutions but were lacking in the resources to implement them, or thought we had the resources but were lacking in ideas to which we could not immediately find objections. Often, we have had neither resources nor reasonable ideas. It seems to us today that problems that arise when we *start* with, to continue with the example above, a list of names and want

- to find out how many distinct names appear in our population, and how many times each appears;
- to form a mapping, using the names, between two relations containing lists of names;
- to form a mapping, using the names, between one relation containing the list of names and another containing not more than one name per entity;
- to create a new relation whose entities are unique names (assuming that such a dataset did not exist already).

The need for these operations illustrates the difficulties here and lays out the requirements for an acceptable solution.

Multiple mappings among relations

The requirement for multiple mappings in a statistical database does not arise very often but, when it does, it proves extremely helpful. It could perhaps be replicated by several joins and subsets, but at greater (logical) cost and complexity to the user. An example from one of the first major political science research efforts with the system may be more helpful than an abstract presentation.

In a voting study, the user (Deber 1977). had a collection of data containing candidate names, party affiliation, districts, election years,, and votes for a particular set of congressional districts. The data were not identified with who had won each election, much less vote percentages and other information useful for trend analysis. It was ultimately necessary to aggregate and compare the information in a variety of ways, and to connect vote information with candidate biographic and personal electoral and party history information, but it was first necessary to identify the elections and winners from these data.

We first formed a new relation, using uniqueness of the date-district tuple as the creating condition. We inferred into that relation the maximum value (for each contributing entity) of the vote, and the sums of the votes. That gave us, in the new relation, four attributes: the year and district, the total votes cast, and the vote for the winner. Completing the winner's percentage of vote (in that relation also) is trivial. But our goal was to identify the

winners. To do this, we built another mapping between the original and new relations that associated matching triples of district, year, and vote with district, year, and (maximum) vote. The entities of the original relation for which this mapping exists are "winners", other entities represent non-winners. (7)

This example is, of course, fairly trivial but it does illustrate the problem.

Other derived relations

Finally, we often see a need to create new relations from existing ones. To our surprise, the facilities of Janus for doing operations within a single relation are apparently powerful enough that this requirement arises much more often in statistical uses than in commercial ones. The reasons are similar to the uses of a hierarchical set in the examples above: a means of aggregating information in ways that accumulate information for a selected grouping rule on the entities. The most common of these is the creation of a new relation whose entities are the unique values of an attribute or tuple in some other relation. Janus also supports creation of new relations by union, intersection, and difference of sets of others, but these more complex operations have, as mentioned above, received very little use in the community of analysts of statistical and quasi-statistical databases.

The statistical interface

Janus does not contain any serious statistical facilities, although it can be used to perform simple summaries -- means, standard deviations, and the like -- and data transformations -- logs, sines, cosines, and so forth. The reasons why it does not are partially philosophical and tied up with the mechanisms used in the Consistent System to keep its components from becoming complex to the point of unmaintainability. (8) The technical reasons are more important to the statistical database question. One of our primary motivations with the Consistent System was to insure that the outputs of operations would be self-describing files that could be manipulated by the system: reformatted, passed to further analysis routines, and the like. That requirement implied that the outputs of statistical procedures embedded in Janus would, of necessity, be either values that could be embedded in existing relations (9) or in new relations. The value plan works fairly well for the sorts of simple summaries mentioned earlier -- we can easily store the mean of a set of attribute values. (10) However, assume that we are computing a regression on some of the attributes in a relation. The results of that regression are represented in several differently-shaped objects (e.g., regression coefficients, partial correlations, analysis of variance information), few of which bear any particular relationship to the number of attributes or entities of the original relation. Only the residuals lend themselves naturally to being stored back as additional attributes in that relation. We could create new relations for each of the outputs but there would be no obvious way to create mappings to the existing relations.

A crosstabulation that produces a multidimensional table is another excellent example. Recall that we are committed to producing that table as a file, not just printing it, and that we are committed to actually producing the contingency table, not just statistics about what is going on inside it. For a variety of reasons, the best form for that file (again, in most cases) is as a multidimensional array with one dimension per variable, and the extents along each dimension matching the number of codes. That form cannot be represented in Janus at all if the number of dimensions (variables) exceed four and cannot reasonably be represented if the number exceeds two (Janus cannot associate labels with the rows and columns of the cells of matrix-valued attributes).

So, we push the results of regressions and tabulations off into the numeric array files used by the rest of the Consistent System, unless the user only wants the results printed and discarded.

To reduce inconvenience to people doing simple and obvious things, there is a connection in Janus to the language form used in the rest of the Consistent System for numerical and statistical computations. It will convert values automatically and invisibly as needed, and will, on request, print results. If the resulting values can reasonably be represented in Janus and the user wishes, it can convert the results and add them to the current working relation. In more complex cases, the user uses Janus commands to put the desired values into system-standard files, and then invokes the desired commands directly (using, however, the same linguistic constructions). These commands may be issued by "escaping" from Janus, or by terminating the Janus session and returning to Consistent System command level; the choice depends on the preferences of the user.

Relationship to commercial uses

Janus was designed specifically to manage statistical databases. Its data storage schemes, ways of presenting information, and, most important, lack of locking strategies, tools for frequent updates, detailed audit trails, and transaction facilities, are consequences of that goal. There have been several attempts to use it in more conventional commercial data base applications. Depending on how one looks at the world, those attempts have either been extremely successful, considering the circumstances, or rather unfortunate. The commands that have been added to Janus to support merging and updating of information have never worked as well or as cleanly as those that perform more conventional statistical manipulations. We have an ongoing internal technical discussion about whether it is really practical to develop record-level locking in a database system in which most information is stored transposed and this author, at least, is deeply skeptical. If it is possible to develop a single system that will serve, and serve equally well, both the needs of the statistical database user and those of the conventional/commercial user, especially on the same databases, we have not invented it.

Notes

1 This argument, which leads to the suggestion that information in statistical databases be stored in transposed form, has apparently risen up as the conventional wisdom in many places at different times. Important examples include PICKLE (Baker 1976), SCSS (Nie 1977), and RAPID (Turner 1980). There is some experimental evidence that supports it. See, for example, Tjoa and Wagner (1980) or Batory (1978).

2 At the risk of confusing any casual Janus user who encounters this paper, we are going to try to stay with conventional terminology in this discussion except in the Janus language examples. The terminology used in Janus documentation uses "dataset" (the common term among data analysts) instead of "relation" or "table" and the term "relation" to describe the mapping among a pair (or more) of relations ("datasets") "Entities" and "attributes" have their usual meanings. There are conceptually several types of functions: those that create new relations ("datasets"), those that define new mappings ("relations"), those that are used in forming new values for each entity in a relation (e.g., log10, upper_case), those that aggregate values within a relation ("dataset") (e.g., mean or sum), and those that move or infer information from one relation to another which information was not involved as arguments to the relation-creating function. The style of the language and documentation have permitted the notion of a "tuple", and analogous terms, to be avoided entirely.

3 It is possible to get around this in some circumstances, but doing so often leads to more confusion.

4 This is, however, easy and convenient since Janus maintains a sequential entity number as a key that can be used in selection expressions.

5 In the rare case in which subjects are identified by order alone, the entity sequence numbers, mentioned above, are coerced into subject identifiers.

6 Other commentators have also assumed that this sort of convention is necessary. See, for example, Teitel (1981).

7 A complication could, in principle, arise if two cases had exactly the same vote in the same election. It was easy to test for and did not, empirically, occur -- these were, after all, election results and someone was elected in all cases.

8 A non-technical discussion of the modular organization of the Consistent System and the intellectual motivation for it appears in another paper (Klensin and Yntema 1981). Time and space do not permit replicating that discussion here.

9 Janus provides the capability of representing so-called dataset-level values that logically represent all of the entities of a relation, rather than being different for each entity. Such values as the mean or standard deviation of an attribute fall into this category. The alternative, dictated by a strict relational model, would have been to create a new relation representing the appropriate aggregation of cases and compute summary values using that aggregation. While that facility exists in Janus, and is heavily used when aggregate summaries are needed for several subsets of the entities in a relation, being able to store values that are conceptually scalars associated with a given relation with that relation is intellectually easier in many cases.

10 If the attributes are vector- or matrix-valued, there is an additional small problem of selecting the correct value: whether the mean (or whatever) is to be of all of the values, or is itself is to be a matrix, or some collapse of that matrix, or whether the mean is wanted for each entity, for the values of that entity. Functions such as "mean" take additional syntax when applied to matrix-valued attributes to select among these possibilities. That arrangement is not available when values are being projected or aggregated across relations; something we frequently regret.

Bibliography

Armor 1969
Armor, David J., et. al., *The Data-Text System*, Preliminary Manual, Department of Social Relations, Harvard University, 1969.

Baker 1976
Baker, M., "User's Guide to the Berkeley Transposed File Statistical System: PICKLE", Survey Research Center, University of California, 1976.

Batory 1978
Batory, D.S., "On Searching Transposed Files", *Fourth International Conference on Very Large Databases*, 1978.

Codd 1971
Codd, E. F., "A Data Base Sublanguage founded on the Relational Calculus", *Proceedings of the 1971 ACM-SIGFIDET Workshop on Data Description, Access, and Control*, San Diego, Calif., 1971.

Dawson, Klensin, and Yntema 1980
Dawson, Ree, John C. Klensin, and Douwe B. Yntema, "The Consistent System", *The American Statistician*, 35, 3 (August 1980), pp. 169-176.

Deber 1977
Deber, Raisa, *Who runs: Congress and realignment sequences*, unpublished Ph.D. dissertation, Department of Political Science, Massachusetts Institute of Technology, 1977.

Farris et. al. 1980
Farris, Lee, H. R. Alker, Jr., Kathleen Carley, and Frank L. Sherman, "Phase/actor disaggregated Butterworth-Scranton codebook", Project working paper 13, project on Reflective Logics for Resolving Insecurity Dilemmas, Center for International Studies, MIT, 1980.

Goldstein and Strnad 1971
Goldstein, Robert C. and Alois J. Strnad, "The MacAIMS Data Management System", MIT Project

MAC technical memorandum MAC-TM-24, April 1971.

Klensin and Yntema 1981
Klensin, John C. and Douwe B. Yntema, "Beyond the package: A new approach to behavioral science computing", *Social Science Information*, 20, 4/5 (1981), pp. 787-815.

McIntosh and Griffel 1970
McIntosh, Stuart D. and David M. Griffel, "Admins Mark III - The user's manual", MIT Center for International Studies, March 1970.

Miller 1967
Miller, James R. III, "Datanal: An interpretive language for on-line analysis of empirical data", MIT Sloan School working paper 275-67, August 1967.

Miller 1968
Miller, James R. III, "On-line analysis for social scientists", *Social Science Information*, April 1968.

Nie 1977
Nie, Norman H. and C. Hadlai Hull, *SCSS: SPSS Conversational Statistical System*, Chicago: SPSS, Inc., 1977.

Stamen and Wallace 1974
Stamen, Jeffery and Robert Wallace, "Social Science Goes Online", *Computer Decisions*, April 1974.

Teitel 1981
Teitel, Robert, "User Interface with a Relational Model of Data", *SIGSOC Bulletin*, 13, 2-3 (January 1982).

Tjoa and Wagner 1980
Tjoa, A. M. and R. R. Wagner, "A general concept for the simulation of interaction on statistical databases" in Barritt, M. M. and D. Wishart, eds., *COMPSTAT 80: Proceedings in Computational Statistics*, Vienna: Physica-Verlag, 1980, pp. 115-121.

Turner 1980
Turner, M. J., R. Hammond, P. Cotton, "RAPID: A DBMS for Large Statistical Databases," Statistics Canada, 1980.

SYSTEM/K: A KNOWLEDGE BASE MANAGEMENT SYSTEM

Mauro MAIER, Claudio CIRILLI

IBM Scientific Center, Via S.Maria 67, 56100 Pisa

Abstract

System/K is a Knowledge Base Management System designed to offer a set of facilities for knowledge representation and usage at the conceptual level by means of three descriptive mechanisms ("Aggregate", "Derive" and "Collect").
Two specific object-types (Assertions and Sets) are defined to represent the "part-of", "is-a" and "member-of" relationships.
An "object-oriented" cross-reference logic is defined, that saves users from having to be constantly aware of "keys".
System/K refers to SQL/DS (a relational DBMS) to maintain information concerning both the conceptual relationships (meta-database) and the description of the entities in the real world (database).
A logic is defined to generate the appropriate relation schemes starting from the conceptual definitions.

1. - Introduction

This paper describes the basic concepts of System/K, a research prototype for knowledge representation and usage at the conceptual level, developed by the IBM Scientific Center at Pisa, in the framework of a research project on Territory Information Systems (TIS project) [7].
The terms "knowledge representation", "knowledge base" and "semantics" are usually referred to with different meanings, depending on the topics and the research communities which use them (artificial intelligence, linguistics, database systems, programming languages) [6] [4].
Although the exchange between these communities has been growing over the last few years, it may be worthwhile to define, at least informally, how these terms are used here.

In the context of complex information systems "knowledge representation" means that designers and users are provided with adequate and integrated tools, in order to describe and manage what they know about the real world concerned, with special reference to the high level of description and to the basic requirement that the knowledge inserted be understandable both for the whole community of users and for the system itself (that is, an adequate and sufficiently precise notation is needed) [13].

In addition to this, a high degree of "semantic capacity" (as a measure of the correspondence between the representation and the meanings of the real world) is required, both for static (data) and dynamic (process) aspects which must be representable by taking into account mechanisms that human beings naturally use to organize their knowledge (such as classification, aggregation and generalization) [13] [20] [16] [17].

To fulfil these requirements "semantic models" (DB corresponding to what is usually referred to in AI as "representation schemes") [3] [5] [10] can be used as basic tools of "structured knowledge representation": the application of a given model to a slice of reality gives the relevant "knowledge base".
Thus a knowledge base includes both "abstract knowledge" (information on general concepts, types, descriptors etc.) and "concrete knowledge" (information on individual facts) [13].
In terms of stored data, the knowledge base is split up into two subsets: the database, in its usual meaning, and the "meta-database", that is the structured collection of data which materialize the abstract knowledge, together with control information (data dictionaries, database schemes etc.) [11].

In the following we describe a proposal for a Knowledge Base Management System.
In section 2, an overview of System/K is presented.
In section 3, the basic concepts of System/K are presented with special emphasys on the three description mechanisms ("Aggregate", "Derive" and "Collect"), that make it possible to compose descriptions of elementary facts of knowledge (called "Assertions") to build the descriptions of complex entities.
In section 4, some implementative aspects are illustrated such as the categories of objects the system manages, the logic the

system follows to navigate through the database, the relation schemes the system adopts to materialize the meta-database (the set of descriptive structures) and the logic the system uses to automatically build the relation schemes to materialize and maintain the descriptions of the specific objects.
In the conclusive section, an attempt is made to emphasize the advantages offered by a conceptual interface to the database in terms of semantic capacity and ease of use.

2. - System/K General Overview

System/K is a Knowledge Base Management System designed to offer a set of facilities for knowledge representation and usage at the conceptual level. It is based on an "object-oriented" approach, i.e. it deals with objects, addressing them by means of system identifiers which are completely transparent to the user. In this sense, System/K enables the user to refer to certain properties of an entity (e.g. the personal code of an employee) to find the required entity (the required employee); from that moment on, the system uses the appropriate identifier to present the concerned entity in its entirety (the employee with all his/her properties).

System/K is the basic component of a system for Territory Administration and planning, developed in the framework of the TIS project.
The gross architecture of this macro-system is to be found in Fig.1.

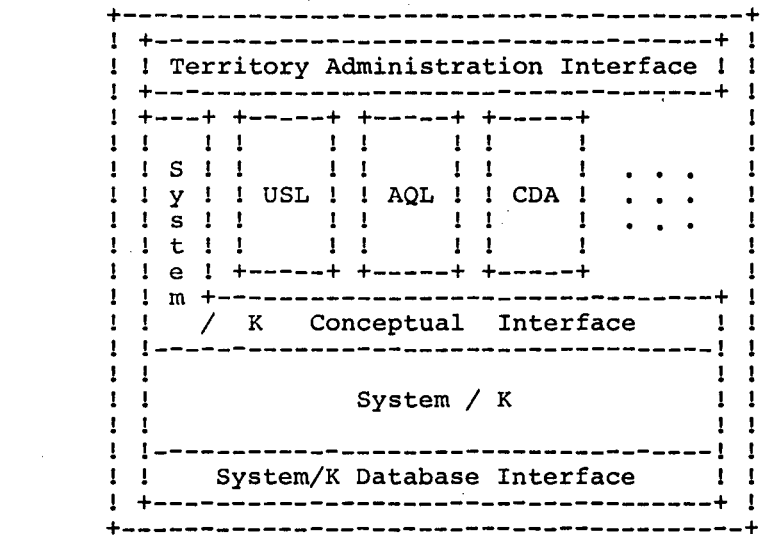


Fig.1 - Architecture of a System for Territory Administration

The facilities offered by System/K are presented to the user by means of the System/K Conceptual Interface (KCI). Certain specific languages and components can be located on top of this basic interface, such as USL (User Specialty Language) [14], developed at the IBM Heidelberg Scientific Center, to offer System/K facilities through a quasi-natural language; AQL (APL Query Language) [2], developed at the IBM Rome Scientific Center, to offer System/K facilities in the APL environment; Conceptual Design Aid, to be developed at the IBM Pisa Scientific Center, to offer a specific set of administration facilities, especially for database optimization purposes.
All these languages and components will be presented to the final user through a general purpose interface, based mainly on menus. A Database Interface (KDI) is used by System/K to map assertion values into specific DBMS structures, files and libraries. At the moment, KDI refers to SQL/DS [19] and to core-image libraries, as far as data management and program management are concerned respectively.

3. - Basic Concepts

System/K is based on the assumption that a significant portion of the knowledge, acquired on the real world, is representable in a computer by means of elementary statements. These elementary facts of knowledge can be combined, then, by means of appropriate mechanisms to represent more complex concepts and entities.

3.1 - Assertions

System/K is based on the assumption that representable knowledge is made up of elementary facts, each describing a specific characteristic of an entity by means of another associated entity. Elementary facts are, then, described according to a protostructure which consists of three components: the described entity, the descriptive role (or property) and the describing entity.
Instances of this protostructure will hereafter be called "assertions", and will represent single elementary facts of knowledge.
The set of all the assertions known by the system will be called the "Knowledge Base".

Examples of assertions are:

(1) "a character string is the name of a person";
(2) "an integer is the population of a city";
(3) "Mauro Maier is the name of a specific person";
(4) "92000 is the population of the city named Fidenza".

It may be noted that some assertions look like "variables" (refer to (1), (2) in the examples above) with a set of possible values (domain) associated to them, while others look like "constants" (refer to (3), (4) in the examples above). Moreover, constant assertions are often obtained selecting precise values from the domain of the appropriate variable assertions. Variable assertions capture the modalities of representing analogous elementary facts, while, constant assertions represent specific facts existing in the real world.

3.2 - Descriptive Mechanisms

Three descriptive mechanisms ("Aggregate", "Derive", and "Collect") are defined, to capture, represent and use knowledge in System/K on the basis of certain abstract concepts and relationships [5].

The "Aggregate" Mechanism

The "Aggregate" mechanism makes it possible to represent an entity in terms of other entities, each of which describes a property of the former one. The "Aggregate" mechanism makes it possible to represent an entity of the real world as a group of assertions, which describe how other entities concur in this representation with specific descriptime roles. The "Aggregate" mechanism groups variable assertions to build the descriptive formats (structures) of classes of entities, and constant assertions to build the description of specific entities (as instances of the appropriate structures, in general). The "Aggregate" mechanism is related to the abstraction mechanism "Aggregation", as defined in [17] [13] [5] [1], and makes it possible to materialize "part-of" relationships [5].
The groups of assertions produced by the "Aggregate" mechanism are called "objects" in System/K.

An example of the use of the "Aggregate" mechanism to define a structure is:

```
DEFINE Person::
       name:: surname: Character;
              first-name: Character;;
       birthdate: Date;
       age: Age(birthdate);
       actimity: Character;;
```

The "Derive" Mechanism

The "Derive" mechanism makes it possible to define new structures as views or, together with the "Aggregate" mechanism, as "specializations" and "extensions" of others, already defined in the knowledge base. The "Derive" mechanism is related to the abstraction mechanism "Generalization", as defined in [17] [13] [5] [1], and makes it possible to materialize "is-a" relationships [5].
An example of the use of the "Derive" mechanism to define a new structure is:

```
DERIVE Employee FROM Person
  WITH activity = 'employee'
  AND::code: Integer;
       job: Character;
       salary: Money;;
```

As far as the concept of "view" is concerned, the "Derive" mechanism makes it possible to define structures derived from others, for the sole purpose of describing the same object in a different way or presenting other assertions with different, but equivalent, describing components.
In order to make it possible to use the same name, views must be defined inside a different user environment from the one containing the originating structure. Several views can be grouped into "contexts" so as to supply overall personalized views of the knowledge base and authorization schemes as far as access to, and handling of, information is concerned.

The "Collect" Mechanism

The "Collect" mechanism makes it possible the definition of objects as sets of other objects in the knowledge base. A "set-object" (more simply, a "set") is defined by a "collection criterion" which determines whether or not an object belongs to the set in question. An object of the knowledge base belongs to a set if and only if it complies with the relevant collection criterion. Each object in a set plays solely the role of being a member of that set, without any other specific meaning. In this sense, the "Collect" mechanism is related to the abstraction mechanism "Association" (or "Collection"), as defined in [5] [1], and makes it possible to materialize "member-of" relationships [5].

In the part above on the "Aggregate" mechanism, the concept of class was used to refer to all the entities described in terms of the same group of properties (the same descriptive format). The concept of class, in System/K, is a special case of the more general concept of "set" as produced by the "Collect" mechanism, the relevant collection criterion being: "all the objects produced as instances of the same structure". The following are examples of sets that are not classes: an examining board, an agricultural community, a football team, etc.

3.3 - Basic Structures

Two specific structures ("Assertion" and "Set") are defined in System/K in order to materialize the conceptual relationships ("part-of", "is-a" and "member-of") on the basis of which information about the real world is acquired, represented and maintained in the "Knowledge Base".

Assertions are the basic elements of System/K. They are defined as follows:

```
DEFINE Assertion::
   role: Word;
   target: Class;
   domain: Object;
   type: Character
      ('owned'     ! 'derived',
       'inherited' ! 'restricted',
       'explicit'  ! 'implicit',
       'unique'    ! 'multiple',
       'constant'  ! 'variable');
   default: Object in domain;;
```

Many types of assertions may take place; the meaning of all these types will be the subject of a future document, together with a detailed description of all the assertions in the above structure. Among the others, the types "owned", "derived", "inherited" and "restricted" will be considered now.
We call "owned" those assertions owned by the object under definition, while we call "derived" those assertions acquired by the object in question from a more general object. To make an example, consider the entity "employee" defined as a specialization of "person" (an employee is a person in any case). If the entity "person" has the property "name" described by the assertion "the name of a person is a character string", also the entity "employee" will have the property "name" described by the assertion "the name of an employee is a character string". In this example, the assertion concerning the property "name" is owned by the object "person", but is acquired by the object "employee".
The types "inherited" and "restricted" concern the way an assertion is derived from another. The example above concerns an "inherited" assertion: the rules to form the name of an employee are exactly the same as those to form the name of a person (both of them are represented by a character string). A "restricted" assertion is derived from an existing assertion with an additional restriction on the domain (the describing component). An example of restricted assertion can be the definition of the property "children" of a person with the distinction between daughters and sons. In this case three assertions take place: (1) "the children of a person are persons"; (2) "the daughters of a person are children of this person with female sex"; (3) "the sons of a person are children of this person with male sex".

Sets are the basic tool of System/K for organizing knowledge. There are two main types of sets: "Classes" and "Groups". With class we refer to a set containing objects generated on the basis of the same structure, while groups are all the sets produced by the "Collect" mechanism on the basis of a defined criterion (enumeration included). Sets represent "member-of" relationships and are defined by the following structure:

```
DEFINE Set::
   name: Word;
   criterion: Predicate;
   type: Character
      ('homogeneous'  ! 'heterogeneous',
       'independent'  ! 'characteristic',
       'structured'   ! 'unstructured',
       'class'        ! 'collection',
       'base'         ! 'derived',
       'view'         ! 'specialization');
   preconditions:   (Operator,Predicate) ;
   postconditions:  (Operator,Predicate) ;
   domain-operators:  Operator ;;
```

As well as for the assertions, a detailed description of all the possible types of sets and of all the assertions describing a set will be the subject of a future document.

4. - Implementative Considerations

System/K is a knowledge base management system realized on top of SQL/DS database management system [19]. The implementation of System/K is based on:

- a certain number of categories of objects the system is expected to provide;
- a logic to navigate through the knowledge base following the paths stated by the descriptive mechanisms;
- a logic to automatically map the objects of the knowledge base into relation schemes and relations;
- a logic to modify the relation schemes and the relevant relations (for optimization purposes), preserving the congruence of the database with the conceptual scheme.

4.1 - Categories of Objects

The objects of the knowledge base are classified in two main categories: "Unstructured" and "Structured".
The unstructured objects are then subdivided into "Elementary" and "Special", while the structured objects are divided into "Independent" and "Characteristic".
For the construction of an object of any category, the starting point is the

concept of "symbol", that is something with no meaning in itself, except for its "mnemonic" value for the human beings.
Symbols are represented by means of bit-strings and are used as the target of the knowledge mapping functions. Operators are provided to compare bit-strings and to transform bit-strings into other bit-strings. In this sense, "symbols", together with their operators, constitute the "primitive objects" for knowledge representation and usage.
The first step towards capturing the meanings of the real world is the definition of elementary and special objects in terms of symbols. These objects never exist in their own right, but only in order to describe other objects (for this reason, they may have multiple occurrences in the database, each occurrence being completely independent from the others). Elementary objects aim at defining classes of primitive concepts (like numbers, characters, etc.), in order to establish the rules for mapping those concepts into symbols and for associating appropriate sets of operators with those classes.
Elementary objects are described by means of a structure consisting of a singleassertion (for this reason, elementary objects, together with the special objects, are called "unstructured") with "symbol" as the domain. The following are examples of elementary objects:

```
DEFINE ELEMENTARY Character::
       value: Symbol;;

DEFINE ELEMENTARY Real::
       value: Symbol;;

DEFINE ELEMENTARY Module::
       value: Symbol;;
```

Special objects are defined to represent singular concepts:

UNKNOWN: to represent the fact that the value of a property is not known at the moment and that it may, then, correspond to any one of the possible objects;

NULL/EMPTY: to represent the fact that the value of a property does not correspond to any object in the knowledge base, but to a non-existent object which has the same properties as "ZERO", or as the "EMPTY SET", in relation to the "SUM" and "PRODUCT" operators;

NON-SENSE: to represent the fact that a property is not applicable, i.e. its value corresponds to an object which attributes a lack of meaning to the assertion in which it appears as the describing component.

Structured objects are distinguished from the unstructured objects mainly because they are not related to certain instruments used by human beings to acquire and represent knowledge about the real world, but constitute a "structured" organization of that knowledge.
A distinction is made between "Independent" and "Characteristic" objects in the sense that: independent objects are those objects existing in their own right, while characteristic objects (as well as elementary and special objects) exist only in order to describe other objects [15].
Independent objects will have one occurrence alone in the database, and references to them are made by means of system identifiers, as explained below.
Characteristic objects will have as many occurrences in the database as necessary (i.e., the same number as the number of objects in whose description they concur).
The following are examples of characteristic objects:

```
DEFINE CHARACTERISTIC Name::
       first name: Word;
       second name: Word;
       family name: Word;;

DEFINE CHARACTERISTIC Address::
       street: Word;
       city: Word;
       ZIP-code: Integer;;
```

4.2 - System/K Navigation Logic

System/K, as an "object-oriented" system, deals with objects, addressing them by means of system identifiers which are totally transparent for the user. In this sense, the user refers to certain properties of an entity (e.g. the personal code af an employee) in order to find the required object (the required employee). From that moment on, the system uses the appropriate identifier to present the object to the user in its entirety (the employee with all his/her properties).

The structure and the logic of using system identifiers make up the "System/K Navigation Logic" (KNL), that addresses all the objects in the knowledge base (assertions and sets included) by means of bit-strings called KNLIDs. The appropriate KNLID is used every time a reference is needed to any object in the knowledge base. A KNLID can include the addressed object or give the access path to reach it: in the first case we talk about "Immediate References", while in the second case we talk about "Pointers".

Immediate references may address "Special" or "Elementary" objects, the distinction being made by the target type. Pointers refer to single objects or to a multiplicity of objects. A "Single" pointer makes it possible to reach all the properties of the object addressed. Sets are treated as

objects and their members are not selectable. A "Multiple" pointer makes it possible to reach all the properties of the objects addressed, but sets are treated as collections of objects, thereby allowing the selection of specific members.

Pointers consists of a couple of system identifiers: the "class identifier" and the "object identifier". The first identifier addresses the (base or derived) class to which the concerned object belongs, while the second one identifies that object inside its "base" class (the class origin of the IS-A hierarchy to which the above mentioned class pertains). The identifiers are generated by counters (one for each class) wothout reusing values previously assigned to deleted objects; this is made to avoid problems arising from not deleted references (but invalid after the deletion of the addressed object). A procedure of reference validation and redefinition is provided to solve the deadlock due to the exhaustion of a counter.

Classes are considered at any rate as objects of the knowledge base belonging to the class "Sets", then they have the value "zero" as class identifier and a generic value as object identifier (the class "Sets" has the value "zero" both as class identifier and as object identifier).
The object identifier of a class become the class identifier in any pointer addressing an object belonging to that class.
In this way KNL makes it possible to address any object in the knowledge base following the same logic, independently for the fact that it be a "Set" (group or class) or an "Object" (in the strict sense of member of a class), the distinction being made by the value ("zero" vs "not-zero") of the class identifier.

Every time a new independent object is added to the knowledge base, an identifier is automatically chosen and assigned to it, in such a way that the same symbol (the value of the identifier), used for an object, cannot be assigned to another object even if the first owner has been cancelled from the system. This choice prevents any possible misxse due to references not being cleared at cancel time, but, on the other side, it implies a finite life-cycle of the system: when identifiers are exhausted, a check of all the references in the system is made, in order to clear out those which are invalid, and a new life-cycle is set up with usable identigiers available.

4.3 - Meta-database and Database

The term "meta-database" refers to the structured collection of information about both the abstract knowledge that comes from a conceptual representation of the real world (e.g. classes of objects), and the description of the objects used by the system itself.
The two basic structures of the system (assertions and sets defined above) define two classes of objects, and their instances are a portion of the stored data that materialize the meta-database, which in turn "describes" the relevant concrete objects stored in the database.
Every class of objects from the real world is thus represented in the meta-database by a specific set, and by as many assertions as there are relevant properties, according to the description inserted by means of the conceptual language.

The actual management of any kind of stored data (meta-database included) is performed by means of the underlying relational DBMS (SQL/DS), while the nucleus of the system takes care of any distinction between abstract and concrete knowledge, together with the management and control of the conceptual aspects of the meta-database (e.g. property inheritance coming from IS-A relationships).

In terms of stored data, the ultimate representation, for the meta-database too, is by means of relations, but it is worthwhile noting that these are completely transparent to the users, who deal only with objects and their properties: the system uses appropriate internal identifiers to create and manage the relevant references (both in the database and the meta-database) and to return the objects in their entirety.

The relation schemes are automatically generated starting from the structures defined by the user (or predefined into the system) and can be changed by the Knowledge Base Administrator by means of a set of administration facilities.
As a starting point we assume that all the objects of the same base class are materialized by means of a unique SQL/DS relation scheme and by the relevant relation.
Objects of the same derived class are materialized by a unique additional relation scheme (and by the relevant relation), as far as additional properties (owned assertions) are concerned. This relation scheme is created as an extension of the relation scheme of the class from which the concerned class is derived.

A relation scheme R for System/K consists of an attribute for the system identifier and of as many attributes as are necessary to represent the owned assertions in the structure generating the relevant class.

A relation scheme R, materializing the owned assertions of a structure, can be decomposed vertically into a set of

relation schemes S = R1, R2, ... , Rn , on the assumption that each scheme Ri of S is a projection of R, necessarily containing the attribute for the KNLID.

Horizontal decomposition can be achieved by decomposing a class, at the conceptual level, into a number of derived classes, according to distinct selection criteria, and presenting the original class as the union of the new ones. The user of the knowledge base may ignore this decomposition, even if it is represented at the conceptual level.

Horizontal and vertical decompositions are reversible, and, then, original relations can be reconstructed.
Moreover, vertical compositions can be operated, combining several relation schemes into one whose attributes are the union of the attributes of the combined schemes.
The relation produced will have "NON-SENSE" values in those columns materializing properties that are "inapplicable" for the corresponding object. On this basis, owned assertirns of a derived class can be materialized by adding new attributes to the relation scheme owned by the originating class.

There are certain exceptions to the criteria for automatically generating relation schemes from th ϵ conceptual definitions. These exceptions are related to the characteristic objects, which exist only in order to describe other objects and, then, may have multiple occurrences in the database, and to the multimalued assertions, which originate repeating groups of values. These exceptions will be analyzed in a technical note (to appear) specific on the implementation of the database interface.

5. - Conclusions

System/K has been presented as a knowledge base management system running on top of a relational DBMS (SQL/DS [19]). Its purpose is to offer a set of facilities to the user for knowledge representation and use at the conceptual level. This means that the user of System/K is not required to know the structures into which data are logically and physically organized by the DBMS, but is able to dedicate his/her efforts to the conceptual description of the real world of interest.

The user may then define conceptual relationships between the entities of the real world to capture more semantics than with the traditional DB systems.

The conceptual description produced can be appropriately formalized to represent a basic instrument for integrating knowledge coming from different applicative sectors. Such a description has a great adaptability to changes, being highly independent of the specific applicative sectors and of the database into which it is automatically mapped.

Moreover, it seems less difficult and more systematic to approach quasi-natural interfaces (languages) on top of a conceptual representation of knowledge [12], rather than traditional data structures (hierarchic, network or relational models [9] [18]). Natural languages are one of the most important requirements to gain the favour of casual users (especially in the public administration environment).

On the side of the DB systems, the availability of a knowledge base management system will offer a great clearness of roles. The existence of a system offering appropriate facilities for representing complex relationships at a higher level, with respect to data, will better focus the requirements to be satisfied by the database system: flexibility, logic and performance. In this sense, the relational approach seems to be the more quoted. Its flexibility (ease of use of the relation schemes) and logic (the power of the relational algebra) are getting appreciated more and more.

REFERENCES

[1] ALBANO A., OCCHIUTO E., ORSINI R., "GALILEO: a conceptual language for database applications", Tech. Rep. RT-8-ISIPI-1, Computer Science Dept., University of Pisa, 1981

[2] ANTONACCI F.,BARTOLO L.,DELL'ORCO P., SPADAVECCHIA V., "AQL: a relational DBMS and its geographic applications", Lecture Notes in Computer Science 81, Data Base Techniques for Pictorial Applications, Springer Verlag, 1980

[3] BORGIDA A., MYLOPOULOS J., "Semantic models in databases: some formal aspects", Advanced Seminar on Theoretical Issues in Data Bases, Cosenza, Italy, Sept. 1981

[4] BRACHMAN R.,SMITH B.,(Eds), "Special issue on Knowledge Representation", SIGART, No.70, Feb. 1980

[5] BRODIE M.L., "On modeling behavioural semantics of databases", Proceedings of the Seventh International Conference on Very Large Data Bases, Cannes, Sept. 1981

[6] BRODIE M.L., ZILLES S.N., (Eds), "Proceedings of Workshop on data abstraction, databases and conceptual modeling", SIGART/SIGMOD/SIGPLAN Special Issue, 1981

[7] CASAZZA I., CIRILLI C., MAIER M., "Conceptual modeling in territory management: a proposal for a semantic subsystem", IBM Tech. Rep., G513-3590, Pisa, 1982

[8] CODD E.F., "Extending the relational model to capture more meaning", ACM Transaction on Database Systems, Vol. 4, No. 4, December 1979

[9] DATE C.J., "An introduction to database systems", Addison-Wesley Publ., 1975

[10] HAMMER M., MCLEOD D., "The Semantic Data Model: a modeling mechanism for database applications", SIGMOD Conf., Austin 1978

[11] KILOV K. I.,POPOVA I. A., "Meta-database architecture for Relational DBMS", SIGMOD RECORD, Vol.12,No.1, October 1981

[12] MYLOPOULOS J. et al., "TORUS: a step towards bridging the gap between data bases and the casual user", Information Systems, Vol.2, N.2, 1976

[13] MYLOPOULOS J., "An overview of knowledge representation", Workshop on Data Abstraction, Pingree Park, Colorado, 1980

[14] OTT N., ZOEPPRITZ M., "USL: an experimental information system based on natural language", in Natural Communication with Computers, Carl Hanser Verlag, 1979

[15] SCHMID H.A., SWENSON J.R., "On the semantic of the relational model", SIGMOD Conf., San Jose, 1975

[16] SCHMID H.A., "An analysis of some constructs for conceptual model", IFIP Conf., Nice 1977

[17] SMITH J.M., SMITH D.C.P., "Database abstractions: aggregation and generalization", ACM Trans., 1977

[18] ULLMAN J.D., "Principles of database systems", Computer Science Press, Potomac, Md., 1980

[19] IBM Corp., "SQL/Data System: General Information", GH24-5012, 1981

[20] BUBENKO J.A., "Information modeling in the context of system development", IFIP Congress, Tokio, 1980

7. Data Compression, Storage, and File Organization

See Also. . . .

Computer-Independent Data Compression for Large Statistical Databases

Fredric Gey, John L. McCarthy, Deane Merrill, Harvard Holmes

Computer Science and Mathematics Department
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Abstract

This paper describes a dictionary-driven, hardware-independent data compression scheme for archival storage of large statistical databases. It discusses motivations for this development, storage format requirements, implementation details, access considerations, and possible extensions of the technique. It also analyzes the degree of compression achieved for different types of statistical data files.

1. Introduction and Motivation

SEEDIS is a research and development project on Social, Economic, Environmental, and Demographic Information Systems [MCCA82C], [GEY 81]. The project was initiated in the early 1970's to provide quick, low cost access to databases including the 1970 U. S. Census -- 1.6 billion individual data values for some 400,000 geographic areas. Over the past decade, SEEDIS has evolved from a batch-processing system on Control Data Corporation (CDC) computers to an interactive system on a distributed network of Digital Equipment Corporation VAX 11/780 computers running the DECNET communication system.

By 1977, SEEDIS databases contained nearly 25 billion characters of information, primarily 1970 census data for numerous geographic summary levels, including states, counties, census tracts, enumeration districts, and others. Data were stored in a variety of physical formats, usually a different format for each major data set. Some had been converted to binary representation on the CDC 6000-7000 computers at Lawrence Berkeley Laboratory (LBL). Others had been converted to the CDC display code character set (a 64 character alphabet, with ten six-bit characters per computer word).

Data were stored, for the most part, on an unusual mass storage device, the IBM 1360 photodigital chip store [GEY 75]. In 1977, IBM announced it would no longer maintain this storage device after October 1, 1979, and LBL undertook to search for a replacement. The LBL Computer Center eventually settled on a Cal Comp automatic tape library (ATL), with a 3300 tape reel capacity, robot-assisted tape retrieval, and mounting without operator intervention.

The SEEDIS project was faced with re-archiving its entire database on the new mass storage device. The actual conversion effort consumed 2.5 staff years over a period of 20 months. Given the magnitude of this conversion effort for existing archived data, the project decided to develop a standard archival format in order to minimize the cost of current and future reconversions which might be necessary.

2. Storage Format Requirements

As the project reviewed its long-term archival storage requirements, several common characteristics of statistical databases helped narrow the storage format design goals. First, SEEDIS databases were archival rather than transactional; updates were infrequent and limited in scope. Second, large new databases were continually being added, thus requiring ever-increasing amounts of storage space. Given these basic parameters, several basic design goals were developed, as follows:

- Computer-independent, binary physical storage format for multiple data types
- Dictionary-driven data definition files for data specification and access
- Data compression for efficient storage, retrieval, and transmission

2.1. Computer-Independent Data Formats

For large archival databases such as those in SEEDIS, the data may have a much longer life than the hardware and software used to store and manage them. Since conversion is an expensive and time-consuming process, data need to be stored in formats that do not have

This work was supported by the Office of Health and Environmental Research and the Office of Basic Energy Sciences of the U.S. Department of Energy under Contract DE-AC03-763F00098; and the Department of Labor, Employment and Training Administration under Interagency Agreement No. 06-2063-36.

to be changed as hardware and software change over time.

Unfortunately, most simple standard formats (e.g., fixed length ASCII records) are too inefficient for large numeric databases. Fixed length records require more storage space, more disk accesses for retrieval, and more overhead for data transmission. Furthermore, numeric data represented as characters must be converted to binary before the data can be used in calculations.

In order to provide a format that was simple, efficient, and computer-independent, SEEDIS staff developed a binary storage scheme based on a "virtual machine" for portability of data [HEAL 78]. The commonality of characteristics which constitute this "virtual machine" are as follows:

- storage is divided into 8-bit byte segments
- character (or string) data are stored with ASCII encoding

2.2. Dictionary-Driven Data Definition

Machine-readable data require data specifications that retrieval and applications programs can use. They also require codebooks that human beings can read. The SEEDIS project decided to meet both these requirements simultaneously, by developing a data definition language that could be used to describe both compressed data files and their uncompressed, fixed-length ASCII equivalents.

The basic approach is similar to that of data dictionaries in other database systems and statistical packages. Each self-describing dataset consists of two logical components -- a data definition file (DDF) and a data file (DF). The logical data view is that of a table (or flat file) with a fixed number of rows and columns. Data are arranged so each row of the table contains all the attributes (data elements or columns) of a named entity (e.g., Alameda County), as well as a row label or stub plus any keys necessary for data access and matching. The number of rows is equal to the number of entities in the data file, and the number of columns is equal to the number of data elements in each row.

The basic structure of meta-data elements in the DDF is:

<keyword> = <value(s)>

with one "keyword=value" pair per unit record (line). Keywords occurring before the first data element definition have global effect. That is, they hold for all data elements, unless specifically overridden by keyword definitions within the local environment of a data element definition.

SEEDIS COMPRESSED files, which will be described in Section 3, separate data and description into two distinct physical files. The data file (DF) is in a binary format, while the data definition file (DDF) is in human-readable ASCII representation. One DDF can describe an unlimited number of compressed data files.

Another similar data format known as CODATA (COmmon DATA Format) was also developed by SEEDIS staff [MERR81], [MERR82]. CODATA files contain both data definition and an uncompressed, fixed-length, ASCII representation of the data in a single physical file. These self-describing CODATA files are used to communicate between various independent modules within the SEEDIS system (retrieval, analysis, data entry, graphic display, data export). A software library is available to translate COMPRESSED datasets to CODATA files and vice versa, as well as to extract particular pieces of meta-data information from data definition files.

2.3. Data Compression

Since the volume of SEEDIS databases was quite large (over 25 billion characters of information in 1977 and growing), it was clear that compression techniques were required to minimize costs of data storage, retrieval, and transmission. Initial experimentation revolved around a modified form of packed-decimal format, a data compression technique commonly used by COBOL programmers. This technique, however, was soon rejected because it would save only a fixed amount of space (approximately 50%) for each numeric data item.

Drawing on previous experience with the data, project staff surmised that more substantial compression could be achieved by exploiting several characteristics common to many of the databases in SEEDIS:

- although fixed data fields on source tapes allow enough space for the maximum possible values, most data values require only one or two digits
- many values (particularly zero and missing data codes) are repeated in sequence

The project staff therefore undertook to develop a format which takes maximum advantage of these characteristics, and yet retains the basic objective of hardware independence. The details of this format are described in the following section.

3. Compressed Format Specifications

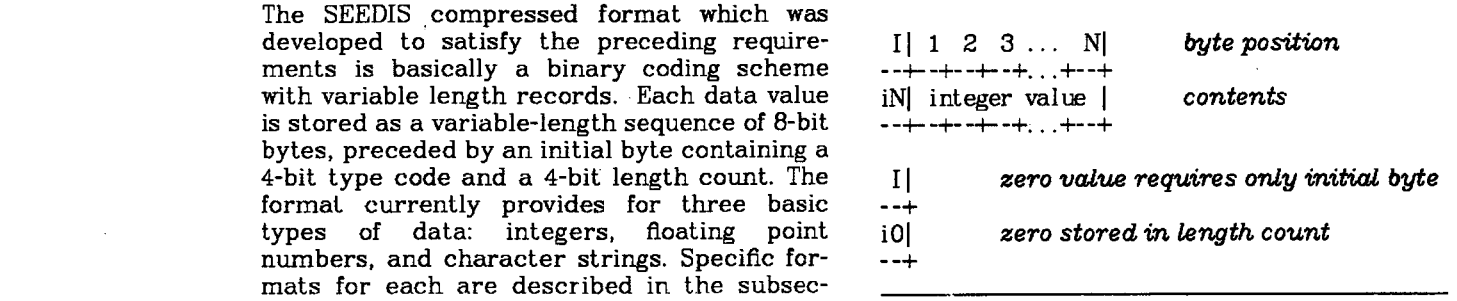
The SEEDIS compressed format which was developed to satisfy the preceding requirements is basically a binary coding scheme with variable length records. Each data value is stored as a variable-length sequence of 8-bit bytes, preceded by an initial byte containing a 4-bit type code and a 4-bit length count. The format currently provides for three basic types of data: integers, floating point numbers, and character strings. Specific formats for each are described in the subsections below.

3.1. Integer and Fixed Decimal Numbers

Both integers and fixed point decimal data values are handled by a single compressed data storage format. The only difference is in the data definition file, where the "type" for a data element (field) may be defined as either decimal or integer, and a scale factor may be included for decimal data. If a scale factor is indicated, that constant value is multiplied by the stored data value at retrieval time. Scale factors may have any positive or negative value; they may be used to convert units (for example feet to meters) at the time data are retrieved from archived files.

For integer and fixed point decimal data values (type = i), the first four bits of the initial byte contain a code indicating type "i," and the second four bits specify a "length count," the number of bytes required to store the integer in its signed binary representation. The initial byte is followed by the "length count" number of bytes containing the signed binary representation of the integer data value. The integer value 0 (zero) is stored in a special way in the initial byte itself, with a "length count" of zero. Exhibit 1 presents a schematic representation of this compression scheme for integers and values of zero. In this schematic representation, the initial byte is shown as byte position "I", while the variable number of succeeding bytes are labeled "1" through "N". Thus the total number of bytes required for a non zero integer data value is N + 1, while a zero value requires only a single byte of storage.

Exhibit 1: Integer Compression

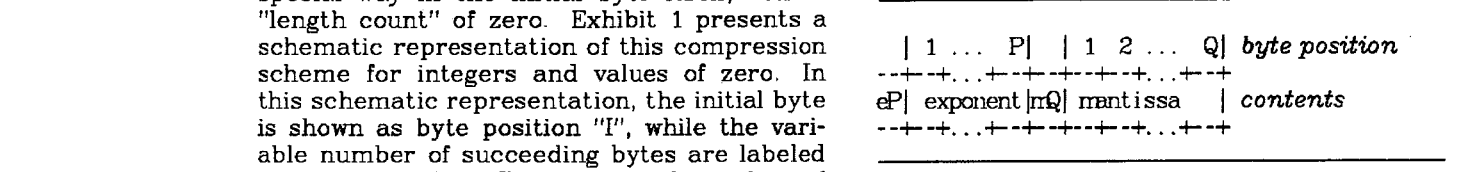


Since many computers cannot handle integers larger than can be stored in 32 bits, integers larger than that are automatically stored in ASCII character string form. The DDF plus the access software can automatically translate such data to floating point values on retrieval.

3.2. Floating Point Decimal Numbers

Floating point numbers are stored as two successive integers representing the exponent and mantissa. The first four bits of the initial byte of the exponent contain a code type indicating "e," and the second four bits contain a "length count," the number of bytes required to store the exponent in its signed integer binary representation. The initial exponent byte is followed by "length count" bytes containing the value of the exponent. These are followed by the initial byte of the mantissa, with four bits indicating type "m" and four bits for the "length count" of the mantissa. Finally, there are the "length count" bytes containing the value of the mantissa itself. Exhibit 2 summarizes the compression scheme for floating point numbers in schematic form.

Exhibit 2: Floating Point Compression



As the schematic representation shows, the amount of storage required for floating point numbers is 2 + P + Q, where P and Q are the number of bytes required to store the integers representing the exponent and mantissa, respectively. The values of P and Q, in turn, are N + 1 and M + 1, respectively, where N is the the number of bytes required to represent the signed integer value of the exponent, and M is the number of bytes required to represent the signed integer value

of the mantissa.

This compressed format for floating point numbers is, in general, less efficient than standard 32-bit binary representations. It requires a minimum of four bytes (32 bits), and frequently may require five or six bytes. On the other hand, it provides a single representation for decimal numbers of virtually unlimited precision. Because of different formats and precision limits for single and double precision on 32, 60, and 64 bit machines, and because most SEEDIS data are integers or fixed decimal numbers, the staff decided to trade compression efficiency for computer hardware independence. Depending upon the data and applications, if large amounts of floating point data became the rule rather than the exception, it might be desirable to add more data types to achieve greater compression at the expense of precision.

Scale factors may be used with the floating point decimal format just as with the fixed decimal format.

3.3. Character Strings

For character string data values (type = a), the first four bits of the initial byte contain a code indicating type "a," and the second four bits contain a "length count," the number of bytes required to store the binary integer representation of the ***length*** of the character string. The initial byte is followed by "length count" bytes containing a binary integer, "S," and then S bytes containing an ASCII representation of the character string itself. Character strings are further compressed by removing trailing blanks.

For example, if we wish to store a 30-byte character field which contains 6 trailing blanks, the initial byte will contain a code for type "a" and a value of 1, indicating that the length of the character string will be stored in the next 1 byte. The second byte will contain a binary integer representation of the value "24," and that will be followed by 24 bytes of the actual character string, represented in ASCII. Exhibit 3 shows a schematic representation of the SEEDIS compressed format for character strings.

Exhibit 3: Character String Compression

```
 I| 1  2 ...  R| 1  2 ...  S|   byte position
--+--+--+...+--+--+--+--+...+--+
aR| integer "S"|string value|   contents
--+--+--+...+--+--+--+--+...+--+
```

3.4. Repetition of Data Values

The SEEDIS compression scheme is rounded out by a fourth data "type" which specifies a repeat count (type = r) of the data value which follows it. Repetition of data values is an extremely important consideration in statistical summary data, which often consist of multi-dimensional cross tabulations of micro-data files. For example, 1970 Fourth Count census data for each geographic area consist of 1178 data items for each of five race categories (total, white, black, hispanic-origin, other). For a large number of geographic areas, all data for the latter three race groups are zero or suppressed. Thus, for those three racial categories, groups of 1178 data values can be replaced by six bytes of actual storage, assuming the appropriate arrangement has been made for physical contiguity of the data (which sometimes requires transposition of the data matrix). Exhibit 4 presents a schematic representation of the repeat data type, using zero as an example of the repeated value. The first four bits of the initial byte contain a code indicating typle "r," and the second four bits contain a "length count," the number of bytes required to store the binary integer representation of the repeat count W (the number of times the data value is repeated).

Exhibit 4: Repetitive Value Compression

```
 I| 1  2  3 ...  W| I|  byte position
--+--+--+--+...+--+--+
rW| repeat count  |i0|  contents
--+--+--+--+...+--+--+
```

Since repeated values of zeros and missing data codes occur frequently in scientific and statistical databases, this type of compression is particularly effective. The repeated value need not be zero, of course. It can be any of the currently recognized data types -- integer, floating point, or character string.

4. A Simple Example

Using the basic building blocks outlined above, we illustrate in this section a brief example using fragments of the data definition file and a single abbreviated data record from the 1980 Census Summary Tape File 1 (STF1).

Exhibit 5 shows a slightly simplified version of the global portion of a data definition file plus definitions for several individual tabulations. This data definition file contains information for both the compressed data file and its fixed field ASCII equivalent. The various meta-data elements appearing in Exhibit 5 are fully defined in [MCCA82A]. The meta-data element "POSITION" gives the sequential field position of the data element within each compressed data record, while the corresponding "START" and "LENGTH" meta-data elements give the physical position and field length in the corresponding fixed-length ASCII CODATA file.

Exhibit 5: Example Data Definition File

```
DATABASE = 1980 Census STF1 Fragment
NDE = 17
AREAS = 4
RECORD_LENGTH = 40
TYPE = A
DE = FIPS.STATE
   START = 1
   LENGTH = 2
   POSITION = 1
DE = FIPS.COUNTY80
   START = 4
   LENGTH = 3
   POSITION = 2
DE = STUB.GEO
   START = 8
   LENGTH = 33
   POSITION = 3
DE = TAB3(1)
   TYPE = I
   START = 42
   LENGTH = 9
   POSITION = 4
   UNIVERSE = 100-Percent Count of Persons
   HEADER =#100-Percent Count of Persons#
DE = TAB74
   TYPE = D
   START = 52
   LENGTH = 9
   POSITION = 5
   SCALE = 0.001
   UNIVERSE = Families
   HEADER =#Median Family Income In 1979#
   HEADER =#(in thousands of dollars)#
DE = TAB12
   STRUCTURE = ARRAY
   DIMENSION = RACE(12)
   TYPE = I
   START = 61
   LENGTH = 10
   POSITION = 6
   UNIVERSE = Persons
CATEGORY_SET = RACE(12)
   CATEGORY = White
   CATEGORY = Black
   CATEGORY = American_Indian
   CATEGORY = Eskimo
   CATEGORY = Aleut
   CATEGORY = Japanese
   CATEGORY = Chinese
   CATEGORY = Filipino
   CATEGORY = Korean
   CATEGORY = Asian_Indian
   CATEGORY = Vietnamese
   CATEGORY = Hawaiian
END DDF
```

There are 17 data elements in Exhibit 5, namely FIPS.STATE, FIPS.COUNTY80, ..., TAB74, and the 12 elements of TAB12. Note that the last DE entry, TAB12, is a vector (or one-dimensional matrix) rather than a simple single-valued field. This fact is denoted by the meta-data information "STRUCTURE = ARRAY." Information about the 12 components of TAB12 is located in the category set "RACE(12)" named in the "DIMENSION" statement. This notation considerably simplifies specification of multi-dimensional arrays, which frequently occur in scientific and statistical data. It also saves space and processing time in handling large data definition files. For such array data elements, the "POSITION" given is the position of the first cell in the array. Each element of an array is stored exactly as a simple data element. Retrieval software computes the linearized position number of other cells from a standard array notation such as "TAB51(3,12,4)". Similarly, the "START" for such data elements gives the starting location of the first cell, while "LENGTH" gives the length of each successive cell in the array.

Exhibit 6 shows a fixed format ASCII representation of a single data record as defined by the DDF in exhibit 5. As implied by the "RECORD_LENGTH" global meta-data item of the DDF, each logical record is comprised of five 40-character physical records (lines), with 20 characters of padding at the end of the last line. This is the type of format that constitutes the data portion of a CODATA file, and it typifies formats used for data export by many

agencies such as the U.S. Census Bureau. Note how fields are allocated to accommodate the largest possible value that might occur in that field -- even though most of the actual values are much smaller. The indication "SCALE=0.001" under "DE=TAB74" specifices that the stored value (17240), when multiplied by the scale factor (0.001) will yield the median family income in thousands of dollars (17.240).

Exhibit 6: Fixed-length ASCII Data Record

byte position in physical record

```
         10        20        30        40
+++++++++|+++++++++|+++++++++|+++++++++|
06 003 CA ALPINE
      1097     17240       912         0
       169         0         0         0
         0         0         0         0
         0         5
```

Exhibit 7 presents a schematic representation of the compressed form of the data record from Exhibit 6. It shows how 200 bytes of original data compresses to 38 bytes of output data, or 19 percent of the original record size.

Exhibit 7: Compressed Data Record

byte position

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
a1  2  0  6 a1  3  0  0  3 a1  9  C  A     A  L  P  I  N  E
-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

byte position (continued)

```
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
i2 1097 i2 17240 i2  912 i0 i2  169 r1  8 i0 i1  5
-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

A close comparison of Exhibits 6 and 7 shows that space savings occur in four major ways: (1) truncation of the area name by removing trailing blanks; (2) binary encoding of integer and fixed point decimal data (e.g. one byte will now hold numbers from -127 to +127, two bytes will now hold numbers from -32,767 to +32,767, and these can be scaled by an arbitrary constant in the data definition file for decimal fields); (3) run-length encoding of repeated values; and (4) elimination of padding at the end of the last physical record. The largest single savings comes from the repetition of 8 successive zeros, which originally occupied 80 bytes, but takes only three bytes in compressed form (byte positions 34-36).

The next section provides more extensive empirical data about the degree of compression achieved over large numbers of records for some actual SEEDIS datasets.

5. Analysis of Compression in SEEDIS

Previous sections have described SEEDIS compression methods and an example of how these compression techniques work for a single data record. This section analyzes the results of compression on entire data files and data bases. In general, data from public agencies such as the U. S. Census Bureau are formatted into fixed-size records for each file, regardless of the contents of the data. Thus, for each data base, a single constant number of bytes is associated with each record as received by LBL. Of interest, therefore, are:

- The distribution of sizes of output records after compression
- The ratio of compressed record size to original record size for different databases
- The cumulative percentage of all records in a data base whose compressed size is less than some percent of the original size
- Whether compression correlates with any particular data item contained within the data record

Exhibit 8 shows the distribution of records for a single data base (1980 Census Summary Tape File 3 (STF3), county records) whose compressed size is expressed as a percentage of the original size. For this database (and level of geography) the median compressed record was 25 percent of the size of the original record.

Exhibit 9 summarizes these distributions over several data bases, showing the cumulative percentage of records with compressed size less than some fraction of original size. The best compression is achieved on the 1980 Census Equal Employment Opportunity (EEO) data file, which contains counts of labor force for 514 detailed occupational categories. The high degree of compression is due to the large number of repeating data values of zero in this file. The worst cases are the air quality and mortality datasets, which contain a high proportion of floating point data.

Finally, we conjectured that the compression might be related to the total population of the geographic area corresponding to the data record, since the proportion of data with values of zero or suppression (missing data)

codes increases for smaller areas. Exhibit 10 is a scatter plot of compressed record size versus the logarithm (base 10) of 1980 total population for the STF3 data base.

A straight-line fit is a remarkably good one, which is especially significant because it suggests that one can sometimes make quantitative estimates of storage requirements for each record or set of records based upon a single value contained within the data.

Our quantitative exploration of compression results is continuing and we hope to use these results in developing further analytic models of compression.

Exhibit 9: Compression in Selected Data Files

Cumulative Percent of County Level Records						
Fraction of Original	EEO	STF3	CDB	STF1	AQ	MOR
0 to 5%	58	0	0	0	0	0
5 to 10%	93	0	0	0	0	0
10 to 15%	98	0	0	0	0	0
15 to 20%	100	3	1	0	0	0
20 to 25%	100	43	5	1	2	0
25 to 30%	100	92	94	18	7	0
30 to 35%	100	100	100	87	20	1
35 to 40%	100	100	100	100	35	3
40 to 45%	100	100	100	100	46	8
45 to 50%	100	100	100	100	59	15
50 to 55%	100	100	100	100	73	28
55 to 60%	100	100	100	100	81	49
60 to 65%	100	100	100	100	88	64
65 to 70%	100	100	100	100	92	75
70 to 75%	100	100	100	100	95	85
75 to 80%	100	100	100	100	98	92
80 to 85%	100	100	100	100	99	97
85 to 90%	100	100	100	100	99	100
90 to 95%	100	100	100	100	100	100
95 to 100%	100	100	100	100	100	100
Mean %	5	25	27	32	47	61

EEO - 1980 Census Equal Employment Opportunity
STF3 - 1980 Census Summary Tape File 3
CDB - 1940-1977 City County Data Book
STF1 - 1980 Census Summary Tape File 1
AQ - 1974-1976 Air Quality
MOR - 1968-1972 Age-Adjusted Mortality

Exhibit 8: Size of Compressed Records

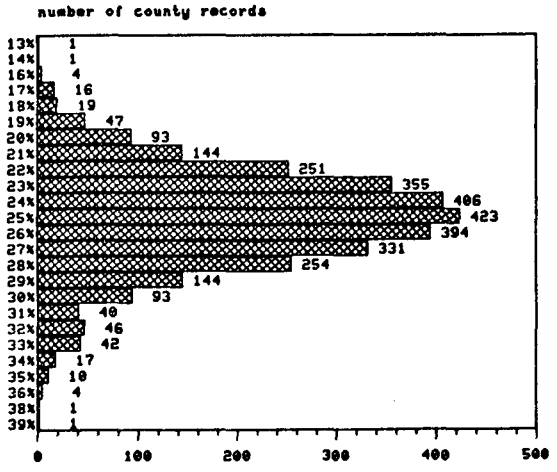
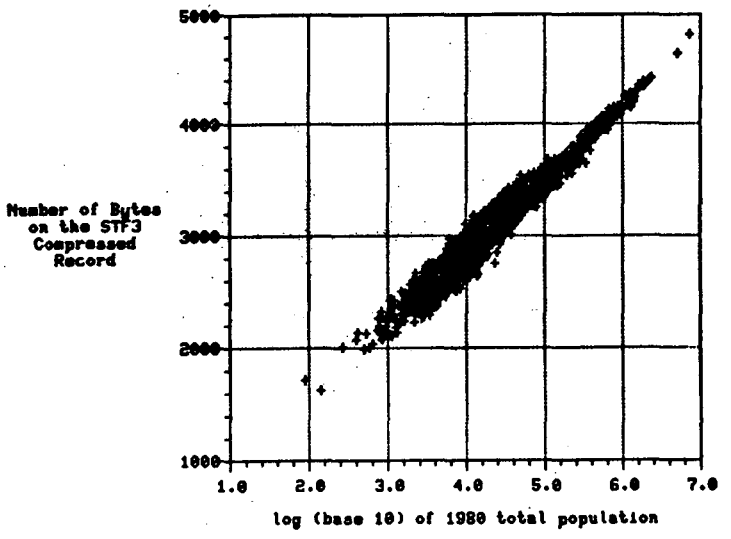


Exhibit 10: Compressed Size vs. Population



6. Access Methods for Compressed Data

In the simplest case, a SEEDIS compressed dataset consists of four physical files. A dataset known as MYFILE, for example, might consist of files having the following names:

MYFILE.DAT: Compressed binary data file (DF)

MYFILE.NDX: Index file containing pointers to individual records (entities) in MYFILE.DAT

MYFILE.DDF: ASCII data definition file (DDF) describing individual named attributes (data elements) in the DF

MYFILE.DDX: Index file containing pointers (sequence numbers) corresponding to each data element in the DDF.

The multi-file scheme permits considerable flexibility in the way data are stored and accessed. For example, large county-level data sets are normally stored with one DAT and its corresponding NDX file for each state, e.g.:

S01.DAT: Alabama DF
S01.NDX: Index for S01.DAT
S02.DAT: Alaska DF
S02.NDX: Index for S02.DAT
MYFILE.DDF: Same as before
MYFILE.DDX: Same as before

Corrections, if necessary, can be easily made to one state at a time. Furthermore, not all states need to be stored on the same disk pack or even on the same node (host computer). Automatic schemes in SEEDIS [MERR83] provide for selective caching of only those files actually required by the user. Because the meta-data (DDF and DDX) are physically separate from the data files (DAT and NDX), meta-data elements (for example data element labels or scale factors) can be conveniently changed without the need to recompress the data.

6.1. Record Access Mechanisms

As stated above, the NDX file provides pointers to the individual records (entities) of the compressed SEEDIS data file (DAT file). In the example above, the file S01.NDX is itself a simple CODATA file containing four data elements for each county in Alabama: the FIPS (Federal Information Processing System) state code, the FIPS county code, the size in bytes of the compressed record for that county, and the physical block location of the start of that county in the file S01.DAT. Now suppose that at some future date the FIPS county numbering scheme for Alabama is changed to accommodate the splitting of a county or the combination of two present counties. Without modifying the file S01.DAT, one can easily modify S01.NDX so that the existing data can be retrieved via the revised county codes. New county codes not corresponding to existing data are simply omitted from the new NDX file, resulting in a missing data indication when data are extracted. Because the NDX files are much smaller than the compressed DAT files, multiple sets of NDX files corresponding to various county definitions can easily be created, all pointing into a single set of large DAT files. Entities (counties) not in the original DAT files can be left missing, or can be created by aggregation or disaggregation and appended to the original DAT file. Each set of NDX files provides pointers to a complete and non-overlapping set of entities, for example 1960 FIPS counties, 1980 FIPS counties, etc., with only a small increase in stored data.

6.2. Intra-Record Access

When retrieving a particular data value from a particular record, the access software must search from the beginning of the record to the particular data value. Thus, although some time can be saved by computing intra-record positions over repeated data items, access time is generally linear, with the last data item in a record requiring the maximum access time. This has not posed severe problems for SEEDIS data records containing up to 1000 data items but it could for much larger data records. The 1980 Census EEO (Equal Employment Opportunity) database, for example, contains over 12,000 items per record. Access time for some data elements will be slow if the current scheme is not enhanced.

Recognition of the limitations of linear access has provided impetus for additional research on compression methods. This research has lead to a variety of general results [EGGE 81]. In the modified approach, which has yet to be incorporated in SEEDIS, all counts are removed from the data file and stored in a separate header. The counts are cumulative, allowing the header to be searched in logarithmic time. The header is used to form the base level of a B-tree index into the data record, which further improves the access time by increasing the rate of the logarithmic search.

An even simpler scheme involving less storage overhead would be to include in the NDX file not only the starting block location of data element 1 in the DAT file record, but also (for

example) the starting block location of data element number 1001, data element number 2001, etc. Considerable time would be saved in retrieving from large files like the 1980 Census EEO file.

6.3. Other Access Considerations

A minor problem which arose during software implementation was the fact that most machines store data in contiguous byte sequential format, but DEC equipment (PDP-11 and VAX) stores numeric data in inverted order within each word of the machine. This fact has been noted in articles on portable software [NEAL 78]. Thus the VAX implementation of the access software had to be slightly different from what it might be on IBM or other hardware.

One other compromise to portability was made within the data records themselves. In order to efficiently use the FORTRAN language for accessing variable-length data records on CDC machines, the beginning of each record contains the CDC word count as well as the total length of the record in bytes.

7. Current and Future Developments

During the past two years a major development effort in SEEDIS has been the design of extensions to meta-data structures for more efficient processing of large summary databases [MCCA82A, MCCA82B]. These enhancements include:

- matrix data elements
- category set specifications
- comprehensive handling of missing data

We are currently designing an enhanced compressed interchange file wherein meta-data as well as data will be stored in the same binary compressed format. Compressed files will replace CODATA files as the standard format for internal interchange of data between SEEDIS modules, in order to provide for more efficient data transfer and conversion. This will be an upwardly compatible enhancement to the original compressed data format, adding new compressed data types for specification of arrays, recursive hierarchical structures, multiply occurring data values, multi-valued missing data codes, etc.

8. Conclusions

With minor modifications and compromises, the computer-independent compressed data storage format described above has remained the SEEDIS standard format for six years. We are currently adding the 1980 Census to the archive, bringing it from 3 to 6 billion data values within a three-year period. We are able to access and decompress on VAX computers data which were archived on tape by CDC computers in 1977.

These compression methods have been very successful from the standpoint of reduction in storage space. Most SEEDIS files occupy from twenty to fifty percent of their original space. Compression of integer and fixed decimal fields to variable-length sequences of bytes, and run-length encoding of repeated values, have accounted for the majority of space saved.

Although the methods currently used to access SEEDIS compressed data files have been adequate for retrieval of data for specified geographic areas, they are not efficient for queries based on data values. Analysis indicates that changes in access methods as well as changes in the compression scheme itself could considerably improve performance for such queries.

Work is currently under way to implement additional compression techniques, different access methods, and compressed data types to accommodate meta-data (e.g., data description files) as well as data. The SEEDIS project plans to use the compressed data format as a medium for internal exchange as well as for archival storage, in order to improve data transmission efficiency between application modules.

9. Acknowledgments

Carl Quong, head of the LBL Computer Science and Mathematics Department, is responsible for the research environment in which this work was conducted. Others who made these results possible include Harvard Holmes, who has directed the SEEDIS project and guided design of the compression methodology, and Bob Healey, who contributed to the design and implemented the initial versions of compression. Wayne Graves made substantial contributions to improving the efficiency of the compression programs, and provided ideas for intra-record access enhancements.

10. References

EGGE 81 Eggers, S., Olken, F., and Shoshani, A., "A Compression Technique for Large Statistical Databases," *Proceedings of the Seventh International Conference on Very Large Databases*, Cannes, France, September, 1981, 424-434.

GEY 75 Gey, F. and Mantei, M. "Keyword Access to a Mass Storage Device at the Record Level," *Proceedings of the First International Conference on Very Large Databases*, Framingham, Massachusetts, September, 1975, 572-588.

GEY 81 Gey, F., "A Beginner's Guide to SEEDIS," Lawrence Berkeley Laboratory Report LBL-11198, January, 1981.

HALL 81 Hall, D., Scherrer, D., and Sventek, J., "A Virtual Operating System," *23 Comm. ACM 9*, September, 1980, 495-502.

HEAL 78 Healey, R., "BYTER and DBYTE," Lawrence Berkeley Laboratory, Internal Document, May, 1978.

MCCA 82A McCarthy, J., "Enhancements to the Codata Data Definition Language," Lawrence Berkeley Laboratory, LBL-14083, February, 1982.

MCCA 82B "Metadata Management for Large Statistical Databases," *Proceedings of the Eighth International Conference on Very Large Databases*, Mexico City, September, 1982.

MCCA 82C McCarthy, J. L., et al., "The SEEDIS Project: A Summary Overview," Lawrence Berkeley Laboratory, PUB-424, May, 1982.

MERR 81 Merrill, D., "CODATA Users' Manual," LBID-021, revised October, 1981.

MERR 82 Merrill, D., "CODATA Tools: Portable Software for Managing Self-Describing Data Files," Lawrence Berkeley Laboratory, LBL-15441, *Proceedings of Computer Science and Statistics: Fifteenth Symposium on the Interface;* Houston, Texas, March 16-19, 1983.

MERR 83 Merrill, D., McCarthy, J., Gey, F., and Holmes, H.; "Distributed Data Management in a Minicomputer Network: The Seedis Experience;" *Elsewhere in the proceedings of this workshop.*

NEAL 78 D. Neal and V. Wallentine, "Experiences with the Portability of Concurrent Pascal," *Software Practice and Experience*, V.8, NO. 3, 1978, pp. 341-353 (specifically p.346).

INDEX CODING: A COMPRESSION TECHNIQUE FOR LARGE STATISTICAL DATABASES[1]

D. S. Batory[2]
Database Systems Research and Development Center
University of Florida
Gainesville, Florida 32611

ABSTRACT

Index encoding is a compression technique that involves the substitution of numeric codes for data values. Current methods of index encoding are suited only for attributes whose underlying domains are small or static. In this paper, general methods to encode dynamic domains are proposed and analyzed. A practical methodology for their application is presented. We also compare and contrast our methods with another that is now being used in a commercial file management system.

1. INTRODUCTION

Scientific and Statistical Databases (SDDs) is a generic name given to databases that are routinely subjected to statistical analyses. SDDs are prevalent in scientific, socio-economic, and business applications. Examples are geologic measurements and observations recorded over a period of time, demographic databases such as the U.S. census, and business ledgers.

SDDs differ from 'corporate' or nonstatistical databases in a number of different ways [HaRi77], [THC79], [Brag81], [Gey81], [ChSh81], [BBD82], [Sho82]. SDDs usually have a significant numerical content with tens or hundreds of attributes per file. Queries tend to be ad hoc and statistical in nature, requiring the examination, sorting, and summarization of selected attribute data of all records in a file. Record deletions and modifications are relatively infrequent, so SDDs tend to be either static or growing. Because of these peculiarities, special techniques are often used to store and process SDDs. Index encoding is one of them.

Index encoding is a data compression technique that is used to some extent in almost all SDDs. Commercial and specialized database management systems, such as RAPID [THC79], IRIS [Alsb75], and CREATABASE [NDX81], have facilities to support it. Index encoding involves the use of numeric codes to represent data values. The idea is to identify the set of all distinct values that an attribute assumes in a data file. The elements of this set are sorted lexically and the index position at which an element appears becomes its index code. The data file is encoded by replacing attribute values with their corresponding index codes. Since the storage requirements of codes are less than their data value counterparts, a sizable compression often results. In general, if an attribute assumes D distinct values, the minimum number of bits needed to represent an index code is $\lceil \log_2 D \rceil$.

A dictionary is maintained for each attribute in order to translate between index codes and data values. For example, a dictionary for the attribute STATE is shown in Table 1. This dictionary enables the 14 byte string "North Carolina" (or any other state name) to be encoded into $\lceil \log_2 50 \rceil = 6$ bits.

0 Alabama	25 Montana
1 Alaska	26 Nebraska
2 Arizona	27 Nevada
3 Arkansas	28 New Hampshire
4 California	29 New Jersey
5 Colorado	30 New Mexico
6 Connecticut	31 New York
7 Delaware	32 North Carolina
8 Florida	33 North Dakota
9 Georgia	34 Ohio
10 Hawaii	35 Oklahoma
11 Idaho	36 Oregon
12 Illinois	37 Pennsylvania
13 Indiana	38 Rhode Island
14 Iowa	39 South Carolina
15 Kansas	40 South Dakota
16 Kentucky	41 Tennessee
17 Louisiana	42 Texas
18 Maine	43 Utah
19 Maryland	44 Vermont
20 Massachusetts	45 Virginia
21 Michigan	46 Washington
22 Minnesota	47 West Virginia
23 Mississippi	48 Wisconsin
24 Missouri	49 Wyoming

Table 1. A Dictionary for STATE

An important feature of index encoding is the identity of the index order and lexical order of (index code, data value) pairs. Because of this identity, the costly process of translating index codes to data values can be eliminated during the data searching, sorting, and processing phases of most file operations; this enables operations to be performed directly and efficiently on compressed data [Alsb75].

As an example, suppose a compressed file is to be sorted lexically on STATE names. One way to do this would be to expand the STATE field for each compressed record, and then sort the expanded file. A more efficient way would be to sort the compressed file on compressed STATE codes, and then refer to the dictionary in Table 1 for expansion.

An obvious problem arises with index encoding when a new data value is added to an attribute's domain; index codes must be reassigned and the data file must be recoded.

Clearly, if data values are added frequently, the cost overhead of recoding becomes significant. An obvious remedy to this problem is not to assign consecutive index codes to attribute values. For example, if (A, E, R, Z) is the domain of an attribute, one might assign the codes (0, 4, 8, 12) rather than (0, 1, 2, 3). Because the codes are nonconsecutive, new data values can be added to the domain and be assigned unused index codes in a way that preserves the lexical and index ordering identity. The data value 'C', for example, could be added to the above domain and assigned index code '2' without altering previously assigned index codes or invalidating the lexical and index ordering identity. The intended benefit of this scheme is achieved; a limited number of domain insertions can be accommodated before the dictionary and data file must be recoded. The penalty that is paid for a reduced recoding frequency is slightly longer index codes.

The initial assignment of index codes and the method by which unused codes are selected and assigned to new data values influences the overall performance of generalized index encoding schemes. In this paper, we will present and analyze two algorithms that index encode dynamic domains. A practical methodology for their application is proposed. We will also compare and contrast our methods with another method that is presently in use.

2. ALGORITHMS TO INDEX ENCODE DYNAMIC DOMAINS

Suppose a domain has D data values initially. Let A_0 ... A_{D-1} be the lexically ordered sequence of these values, where the lowest value is A_0 and the highest is A_{D-1}.[3] It is expected that subsequent record insertions and modifications will cause new data values to be added to the domain.

Index codes can be assigned to A_0 ... A_{D-1} in the following way. Data value A_i is given the index code $i \cdot 2^k$, where k is a nonnegative integer. 2^k is a scaling factor which allows data values to be assigned nonconsecutive (but uniformly spaced) index codes. For example, the codes for A_0, A_1, ... A_{D-1} would be 0, 8, ... $(D-1) \cdot 2^3$ if k=3. Whenever k>0, unused index codes will be present. These codes will be given to subsequent domain insertions. By assigning codes in this manner, the number of bits needed to represent an index code is $\lceil \log_2 D \rceil + k$.

We will consider two algorithms for assigning unused index codes to domain insertions. The simplest is the *true order* algorithm. It works by assigning a new domain value an unused index code whose numerical value is the rounded average of the index codes of the data value's lexical predecessor and successor.

More formally, let V be a new domain value. Let V' and V" be data values in the dictionary which are the lexical predecessor and successor of V. Let C' and C" be their index codes. The index code given to V is $C = \lceil (C'+C")/2 \rceil$. Normally C is an unused code, but there is one exception. If C=C", which occurs when C"=C'+1, there is no unused code that can be assigned to V that preserves the lexical and index order identity or dictionary entries. In this case, the dictionary and data file must be recoded in order to accommodate V.

Figure 1 illustrates the true order algorithm from the time a dictionary was created to the time at which it needs to be recoded. In this Figure, D=4 and k=2.

1a. Initial Dictionary		1b. Insert M		1c. Insert H		1d. Insert W		1e. Insert L	
0	A	0	A	0	A	0	A		
4	E	4	E	4	E	4	E	recode	
8	R	6	M	5	H	5	H		
12	Z	8	R	6	M	6	M	is	
		12	Z	8	R	8	R		
				12	Z	10	W	triggered	
						12	Z		

Note: Index codes are stored as 4-bit numbers
Domain values are single characters

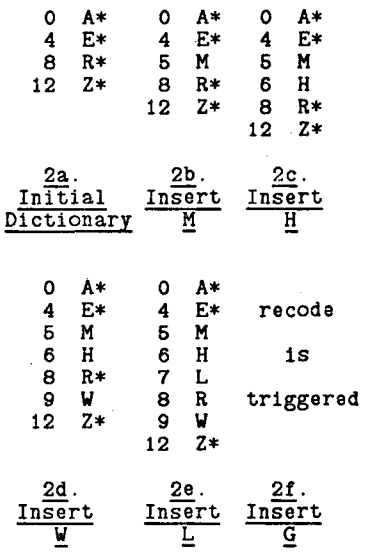
Figure 1. Dictionary Insertions Using the True Order Algorithm

As a general rule, the true order algorithm only allows a certain number of domain insertions to be made before a recoding is triggered. Another algorithm, called the *interval order* algorithm, causes recoding to be delayed to the latest possible time. However, it is more complicated. To understand how it works, again let A_0 ... A_{D-1} be the sorted sequence of values that define a domain initially. Observe that each pair of consecutive initial values defines an interval of domain values. That is, pair (A_i, A_{i+1}) defines the value interval starting with A_i and ending with A_{i+1}. These intervals are fixed and do *not* change with subsequent domain insertions. Once index codes are assigned to A_0 ... A_{D-1}, a distinct interval of unused index codes can be identified with each value interval. For example, if (A, E, R, Z) are the initial values of a domain and (0, 8, 16, 24) are their assigned index codes, value interval (A, E) is identified with the code interval (0, 8). When a domain value is to be inserted, the interval order algorithm finds the corresponding value interval and assigns the new domain value the first unused index code in the corresponding code interval. In general, the lexical and index order of dictionary entries is *not* preserved.

More formally, let G_i be the number of domain insertions whose values fall in the interval (A_i, A_{i+1}). C_i is the index code for A_i. A new value V which satisfies $A_i < V < A_{i+1}$ is assigned the index code $C = C_i + G_i + 1$. Normally C is an unused index code, but there is one exception. If $C = C_{i+1}$, which occurs when 2^k-1 values have already been inserted into the interval, there is no unused code that can be assigned to V. In this case, the dictionary and data file must be recoded in order to accommodate V.[4]

Figure 2 illustrates the interval order algorithm from

the time a dictionary was created to the time at which it needs to be recoded. In this Figure, D=4 and K=2. Because of the special role of the initial domain values, they are indicated by stars '*'.



Note: Index codes are stored as 4-bit numbers
Domain values are single characters
Original domain values are indicated by stars '*'

Figure 2. Dictionary Insertions Using the Interval Order Algorithm

As mentioned before, the interval order algorithm does *not* preserve the index and lexical order identity of dictionary entries. It does, however, preserve a ranking of domain values by their assignment to value intervals. That is, all domain values in the interval (A_i, A_{i+1}) have index codes that are strictly less than the index codes assigned to domain values in an interval (A_j, A_{j+1}) where $i < j$. (Hence, data values are ranked according to an "interval order.") Only within the same interval does the index order differ from the lexical order.

Whenever a difference exists between the index and lexical order, dictionary references will need to be made in order to process certain comparison and sorting operations on compressed files. Such additional complications, however, can be kept to a minimum. In processing an inequality of the form (Attribute > V), for example, only the dictionary entries of other values that belong to the same interval as V need be kept in main memory; all other comparisons can be made using index codes. The number of such entries that are needed at any one time is 2^k. As we will see in the following section, typical values of k will range from 2 to 8, so the storage requirements for these entries is minimal. Similar minor alterations would be needed to handle other operations.

The overall performance of the true order and interval order algorithms depends, in part, on the value k. In the following section, we analyze these algorithms. The results of the analysis provide a guideline for selecting an appropriate value for k.

3. ALGORITHM ANALYSIS

The length of an index code assigned by the true order and interval order algorithms is $\lceil \log_2 D \rceil + k$ bits. We will refer to the most significant $\lceil \log_2 D \rceil$ bits as the *base* of an index code; the least significant k bits will be called the *extension*. The objective of our analysis is to determine an appropriate value for k.

Consider any value interval. Let $T_k(I)$ be the probability that I values can be inserted into the interval without triggering a recode, given that index codes have k-bit extensions. For the interval order algorithm, $T_k(I)$ has a particularly simple form:

$$T_k(I) = \begin{cases} 1 & \text{if } 0 \leq I < 2^k \\ 0 & \text{otherwise} \end{cases} \qquad (1)$$

$T_k(I)$ for the true order algorithm is more complicated. Each time a new value is inserted, the true order algorithm generates an unused code by taking the midpoint of an interval of index codes. The history of interval splitting caused by a sequence of insertions can be modeled by the binary tree that results from these insertions (see Figure 3). The height of this tree is related to the function $T_k(I)$ in the following way. When a new code is generated, the minimum number of bits that are necessary to represent its extension is precisely one more than that which is needed to represent the extensions of the index codes from which it was derived. This means that the number of levels in a binary tree resulting from a sequence of insertions equals the minimum number of extension bits that are needed to encode this sequence. Figure 3 illustrates these relationships.[5]

Accordingly, $T_k(I)$ for the true order algorithm can be interpreted as the probability that a binary tree of I nodes has k or fewer levels. Assuming that all possible domain values are equally likely to be inserted, the following well-studied recurrence relation defines $T_k(I)$:

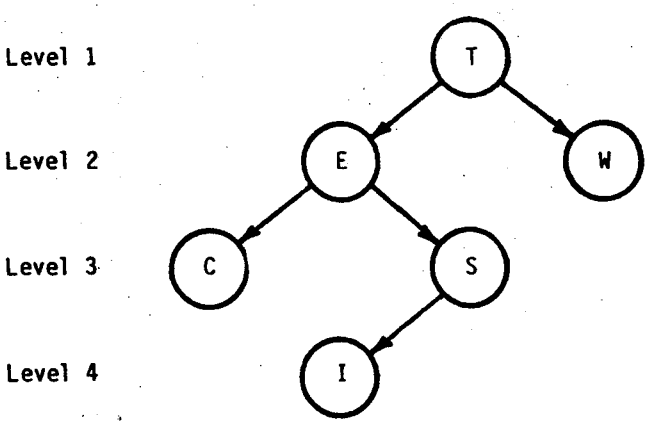
$$T_k(I) = \frac{1}{I} \cdot \sum_{j=0}^{I-1} T_{k-1}(j) \cdot T_{k-1}(I-1-j)$$

$$T_1(I) = \begin{cases} 1 & \text{if } I = 0 \text{ or } 1 \\ 0 & \text{otherwise} \end{cases} \qquad (2)$$

No closed form solution to eqn. (2) is known (see [BKR72], [FL0d81]). However, this does not present a problem since values of $T_k(I)$ can be obtained easily by iteration. (As we will see shortly, practical values for k do not exceed 8.)

Suppose a dictionary initially contains D distinct values. Let $R_k(D,I)$ be the probability that the dictionary will be able to accommodate I domain insertions without triggering a recode, given that index codes have k-bit extensions.

Binary tree resulting from the insertion sequence T,E,S,C,W,I into interval (A,Z).



data value	Binary 1-bit base	Index Code 4-bit extension	Minimum # of Extension bits per Index Code
A	0	0000	0
Z	1	0000	0
T	0	1000	1
E	0	0100	2
W	0	1100	2
C	0	0010	3
S	0	0110	3
I	0	0101	4

Figure 3. Correspondence of the Levels of a Binary Tree and the Minimum Number of Extension Bits per Index Code

Clearly, when a dictionary contains only the lowest and highest possible domain values:

$$R_k(2,I) = T_k(I) \qquad (3a)$$

To determine R_k for $D > 2$, let $A_0 \ldots A_{D-1}$ be the ordered sequence that defines the initial domain values of a dictionary. Without loss of generality, suppose D is odd where $D = 2N + 1$ for some integer N. Let H(D,I,j) be the probability that j of I insertions will have values which fall within the interval $(A_0 \ldots A_N)$, i.e., the first half of the dictionary.[6] Assuming domain insertions have values that are randomly chosen without replacement from a large and static, but not necessarily lexicographically uniform, collection of values, H(D,I,j) is found to be:[7]

$$H(D,I,j) = \frac{(2N-1) \cdot \binom{2N-2}{N-1} \cdot \binom{I}{j}}{(2N-1+I) \cdot \binom{2N-2+I}{N-1+j}}$$

where $\binom{a}{b}$ is a binomial coefficient and, again, $D = 2N + 1$. A derivation of H(D,I,j) is given in the Appendix.

Doubling a dictionary in size and considering all possible sequences of I insertions yields a general recurrence relation for R_k:

$$R_k(2D-1,I) = \sum_{j=0}^{I} R_k(D,j) \cdot R_k(D,I-j) \cdot H(2D-1,I,j) \qquad (3b)$$

The term 2D-1 arises since each 'half' of a dictionary of 2D-1 values contains D values, and one value (namely the D^{th} value) is found in both halves.

The probability $P_k(D,I)$ that a recode will occur on the i^{th} insertion into a dictionary is:

$$P_k(D,I) = \begin{cases} 0 & \text{if } I = 0 \\ R_k(D,I) - R_k(D,I+1) & \text{otherwise} \end{cases} \qquad (4)$$

The expected number of insertions a dictionary can accommodate before a recode $E_k(D)$, and the standard deviation of this quantity $S_k(D)$ follow from eqn. (4):

$$E_k(D) = \sum_{I>0} I \cdot P_k(D,I)$$

$$S_k(D) = \left(\sum_{I>0} I^2 \cdot P_k(D,I) - E_k^2(D) \right)^{1/2}$$

No closed form solution to eqn. (4) is known. Values for $R_k(D,I)$ can be obtained in a straightforward manner by iterating equation (3b). This, as it turns out, is practical only if I is small. For large I, values of R_k can be approximated by calculating $R_k(D,i)$ only for selected values of i; all other values are obtained by linear interpolation. Because R_k is a smooth and slowly changing cumulative probability function, this approach to approximating R_k was found to work well.

The results of selected computations are listed in Tables 2 and 3. The entries of these tables are indexed by D, the initial size of the dictionary, and k, the number of extension bits per index code. The values listed in the tables are the expected growth and standard deviation expressed as a percentage of the initial dictionary size:

$$\%E_k(D) = 100 \cdot E_k(D)/D$$

$$\%S_k(D) = 100 \cdot S_k(D)/D$$

Note that values for some entries could not be determined because the time required for their computation was excessive.

A number of conclusions can be drawn from these tables. It is not difficult to see that the expected dictionary growth before recoding (%E) for the interval order algorithm is uniformly better than that for the true order algorithm. This in itself is not surprising, for only in the best possible circumstances (i.e., for certain input sequences) will the true order algorithm be able to accommodate dictionary growth as the interval order algorithm. What is important is that the performance of the interval order algorithm provides a *theoretical upper bound* on the expected case performance of any algorithm which maintains the index order and lexical order identity of dictionary entries.[8] For example, if insertions to a domain were batched (which can occur if record insertions are batched), it is possible to order these insertions so that each could be assigned an index code with the minimum number of extension bits. While such optimization would significantly complicate an insertion algorithm, the resulting gain in performance would never surpass that of the interval order algorithm.

The primary objective of introducing extension bits to index codes is to reduce the frequency of recoding. *Providing that domain insertions are random*, it is clear in Tables 2 and 3 that allocating just a few bits achieves this objective. For example, if a dictionary initially contains 8193 values and 2-bit extensions are used, the dictionary would be expected to grow by 5.3% if the true order algorithm is used, or by 10.5% if the interval order algorithm is used. This effectively reduces the expected frequency of recoding to every 434 and 860 insertions, respectively. (The corresponding standard deviations of 2% and 3.3% indicate that the expected growth rates are not guaranteed; a rather wide range of possible growth rates centered about these means should be expected.) Except for highly dynamic domains, recoding at these rates should not pose a significant problem. Even higher percentage growth rates can be expected for larger k. If four extension bits were allocated, for example, the corresponding increases in expected dictionary growth are 27% and 114.1%.

	k=2		k=4	
D	%E	%S	%E	%S
2	233.3	47.1	653.3	183.1
3	156.3	51.6	437.3	147.9
5	109.0	41.5	307.5	110.7
9	78.1	31.4	224.8	81.5
17	57.2	23.4	169.4	60.3
33	42.5	17.4	130.8	45.1
65	31.9	13.0	103.1	34.3
129	24.3	9.7	82.6	26.4
257	18.6	7.4	67.1	20.7
513	14.3	5.6	55.2	16.4
1025	11.1	4.3	45.8	13.0
2049	8.7	3.3	38.2	10.7
4097	6.8	2.6	32.0	8.9
8193	5.3	2.0	27.0	7.4
16385	4.1	1.6	22.7	6.4
32769	3.2	1.2	19.1	5.6
65537	2.5	1.0	16.0	5.1
131073	2.0	0.8	13.1	4.8

	k=6		k=8	
D	%E	%S	%E	%S
2	1401.9	452.2	2713.1	957.0
3	933.7	333.0	1796.5	673.5
5	655.6	239.4	1255.9	472.5
9	480.4	172.5	918.0	335.2
17	364.5	126.0	695.9	242.1
33	284.4	93.6	543.6	178.4
65	227.2	70.9	435.1	134.2
129	184.9	54.7	355.4	102.9
257	152.8	42.9	295.0	79.8
513	127.9	34.0	247.0	66.1
1025	107.7	28.5	208.7	55.7
2049	91.3	24.2	177.0	48.6
4097	77.6	21.1	149.6	44.4
8193	65.7	19.1	124.8	42.3
16385	55.0	18.0	101.0	41.1
32769	44.9	17.4	77.7	39.3
65537	35.1	16.7	54.9	34.9
131073	25.5	15.2	-	-

Legend: D initial number of entries in dictionary
k number of extension bits per index code
%E expected growth of dictionary before recoding given as a percentage of D
%S standard deviation of expected growth as a percentage of D

Table 2. Performance Measures of the True Order Algorithm

	k=2		k=4	
D	%E	%S	%E	%S
2	300.0	0.0	1500.0	0.0
3	203.8	53.8	1034.4	223.2
5	144.6	47.3	753.8	197.9
9	105.9	37.2	574.0	157.7
17	79.5	28.5	453.9	122.8
33	60.8	21.7	367.8	95.5
65	47.3	16.1	305.9	74.9
129	37.2	12.8	259.4	59.2
257	29.6	10.0	222.9	49.2
513	23.8	7.8	194.0	41.3
1025	19.3	6.2	170.1	36.0
2049	15.7	5.0	149.7	33.0
4097	12.8	4.0	131.4	32.1
8193	10.5	3.3	114.1	32.8
16385	8.6	2.7	96.5	34.4
32769	7.1	2.3	77.5	35.5
65537	5.7	2.0	57.3	33.8

	k=6		k=8	
D	%E	%S	%E	%S
2	6300.0	0.0	25500.0	0.0
3	4361.2	894.7	17669.6	3579.1
5	3199.7	793.5	12985.3	3174.4
9	2460.2	632.5	10012.3	2504.9
17	1965.0	490.8	7999.9	1977.4
33	1618.6	378.0	6556.7	1637.9
65	1367.4	286.4	5475.6	1400.6
129	1167.5	255.7	4600.1	1284.3
257	1011.5	226.7	3828.5	1256.7
513	880.4	212.7	3087.1	1262.8
1025	761.4	213.7	2333.8	1225.1
2049	644.1	223.8	1597.1	1066.4
4097	519.9	232.6	-	-
8193	386.2	225.5	-	-
16385	-	-	-	-
32769	-	-	-	-
65537	-	-	-	-

Legend: D initial number of entries in dictionary
k number of extension bits per index code
%E expected growth of dictionary before recoding given as a percentage of D
%S standard deviation of expected growth as a percentage of D

Table 3. Performance Measures of the Interval Order Algorithm

As a suggested guideline for choosing a value for k, two to four bit extensions should be adequate for most purposes (again provided that domain insertions are random). Extensions of these lengths can be obtained easily by rounding the number of bits in the base of an index code to the nearest byte or half byte boundary. (Doing so allows one to take advantage of a CPU's byte addressing capability.) More than four bits may be allocated per extension. However, Tables 2 and 3 suggest that eight bits is a practical upper.

Our analysis has been based on the assumption that insertions were random. In real files, domain insertions need not be random. Nonrandom insertions would make the predictions in Tables 2 and 3 optimistic; clustered domain insertions would likely cause recodes to be triggered more frequently. To test the effect of possible clustering, some experiments on real files were conducted. It was observed that a minimal amount of clustering does occur naturally, but the observed deviation of measured values from theoretical predictions was only a few percent. For this reason, it is believed that Tables 2 and 3 provide reasonable guidelines for the selection of k.

A final observation concerns an unexpected trend in Tables 2 and 3. Namely, the expected factor by which large dictionaries grow before they need to be recoded is less than that for smaller dictionaries, i.e., %E declines with increasing D. To explain this trend, it is shown in the Appendix that the probability Q(D,I,j) that a value interval will receive j of I insertions given that D values were in the dictionary initially is:

$$Q(D,I,j) = \frac{(D-2)\cdot\binom{I}{j}}{(j+1)\cdot\binom{D-2+I}{j+1}}$$

Although it is difficult to prove analytically, it is easy to show by example that the value of Q is essentially constant for all dictionaries that have grown by the same factor (i.e., the same I/D ratio):

D	I	j	Q(D,I,j)	Q(10D,10I,j)	Q(100D,100I,j)
256	50	4	.00056	.00059	.00060
256	300	4	.03935	.03906	.03903
512	256	7	.00030	.00030	.00030
512	1024	7	.01959	.01952	.01951
2048	512	5	.00025	.00026	.00026
2048	8192	5	.06556	.06554	.06554

From the above rule it is reasonable to define a growth factor r and a function Q'(r,j) such that $Q'(r,j) \simeq Q(D,I,j)$ and $r = I/D$.

The relationship between %E and D can now be understood. As a general rule, if an interval experiences a large number of insertions, the probability that a recode will be triggered is large. The expected number of intervals with j insertions is $(D-1)\cdot Q'(r,j)$. As D increases, there is a greater expectation that one or more intervals with j values will appear. Consequently, the expected factor by which large dictionaries will grow before they need to be recoded is less than that for smaller dictionaries, i.e., %E declines with increasing D.

4. COMPARISON WITH ANOTHER METHOD

An alternative approach to index encoding dynamic domains is employed by CREATABASE [NDX81], [BrTo76]. The basic idea (not necessarily an exact description) is to assign index codes incrementally, ignoring the lexical and index ordering identity of dictionary entries. That is, if there are N values in a domain, the next domain insertion will be assigned index code N. Initially an attribute will have D domain values; $\lceil \log_2 D \rceil$ bits are allocated per index code. Domain recoding occurs only after I insertions, where the number of bits needed to represent D+I distinct index codes exceeds $\lceil \log_2 D \rceil$ bits. This does not happen very often.

CREATABASE performs operations directly on compressed files. Some operations, such as file sorting and processing range queries, require the identity of the lexical order and index order of (data value, index code) pairs. In such cases, a surrogate index code is assigned to each data value. This surrogate coding supports the lexical and index ordering identity. As compressed records are processed, surrogate codes are substituted for the index codes that were originally assigned. (Note: the compressed file is *not* updated with surrogate codes; substitution occurs only during record processing and is temporary.) In this way, CREATABASE index encodes dynamic domains.

This method does have its limitations. As long as the dictionaries are small enough to be main memory resident, this scheme works very well. The constant dictionary references that are needed for surrogate substitution can be done quickly with little overhead. However, if a domain contains many data values, which happens when an attribute is an identifier or primary key, it does not work as well. The dictionary references caused by surrogate substitution would create an enormous overhead. (CREATABASE, in fact, does not encode such domains for this reason.)

Our method of index encoding should exhibit a comparable performance when dictionaries are main memory resident. It might be less efficient because of longer index codes; it might be more efficient because code substitutions are unnecessary. However, it is primarily for this latter reason that our method should be more efficient when large domains are encoded. Another reason is given below.

When a dictionary is large, it must be organized on secondary storage by a file structure. An appropriate and practical file structure for efficiently maintaining and querying dynamic directories has been proposed by Eggers, Olken, and Shoshani [EOS81]. This is a slightly modified B+ tree structure which accommodates dictionary growth by node splitting and allows records (i.e., dictionary entries) to be ac-

cessed efficiently given either their data values or index codes. (This is possible because of the coincidence of the index and lexical orderings.) Thus, a single and simple directory structure suffices. If the index encoding method of CREATABASE were to be applied to a large domain, several directory structures would probably need to be maintained. One would be needed to quickly translate index codes into data values, another for translating data values into index codes, and possibly a third to support surrogate codes.

5. CONCLUSIONS

Practical techniques to index encode attributes with dynamic domains have been proposed and analyzed. The goal of these techniques is to reduce the frequency of recoding whenever new data values are added to an attribute's domain. Under the conditions that domain insertions are random, recoding frequencies have been shown to be reduced significantly simply by enlarging the length of an index code by a few extra bits. Experimental results on real data support our findings.

It is believed that the index encoding techniques presented here are practical alternatives to those used in specialized and commercial DBMSs today.

Acknowledgements. I gratefully acknowledge the help of Ignacio Casas, Arie Shoshani, and Won Kim for their constructive comments on an earlier draft of this paper. I also thank J. Stevens, D. Carter, R. Hammond, M. Conlon, S. Su, S. Navathe, and S. Kundu for their contributions to the development and clarification of the ideas in this paper.

REFERENCES

[Alsb75] Alsberg, P. A., "Space and Time Savings Through Large Database Compression and Dynamic Restructuring," *Proc. IEEE* Vol. 63 #8 (Aug. 1975), pp. 1114-1122.

[Bat82] Batory, D. S., "Optimal File Designs and Reorganization Points," *ACM Trans. Database Syst.* Vol. 7 #1 (Mar. 1982), pp. 60-81.

[BBD82] Bates, D., Boral, H., DeWitt, D. J., "A Framework for Research in Database Management for Statistical Analysis," *Proc. SIGMOD 1982*, pp. 69-78.

[BKR72] de Bruijn, N., Knuth, D., and Rice, O., "The Average Height of Planted Plane Trees," in *Graph Theory and Computing*, R-C. Read, Editor, Academic Press, New York, 1972, pp. 15-22.

[Brag81] Brag, A. W., "Data Manipulation Languages for Statistical Databases--The Statistical Analysis System (SAS)," *Proc. LBL Workshop on Statistical Database Management 1981*, pp. 147-150.

[BrTo76] Brill, R. C., and Tolkin, S. E., "Subset Selection by Boolean Calculation," 1976.

[ChSo81] Chan, P. and Shoshani, A., "SUBJECT: A Directory Driven System for Organizing and Accessing Large Statistical Databases," *Proc. VLDB 1981*, pp. 553-563.

[EOS81] Eggers, S., Olken, F., and Shoshani, A., "A Compression Technique for Large Statistical Databases," *Proc. VLDB 1981*, pp. 424-434.

[FlOd81] Flajolet, P. and Odlyzko, A., "The Average Height of Binary Trees and other Simple Trees," INRIA Research Report #56, Feb. 1981.

[FNPS79] Fagin, R., Nievergelt, J., Pippenger, N., and Strong, H. R., "Extendible Hashing--A Fast Access Method for Dynamic Files," *ACM Trans. Database Syst.* Vol. 4 #3 (Sept. 1979), pp. 315-344.

[Gey81] Gey, F. G., "Data Definition for Statistical Summary Data or Appearances can be Deceiving," *Proc. LBL Workshop on Statistical Database Management 1981*, pp. 3-18.

[HaRi77] Hampel, V. and Ries, D., "Requirements for the Design of a Scientific Database Management System," working paper, Lawrence Livermore Laboratory, Berkeley, CA, 1977.

[Lar78] Larson, P., "Dynamic Hashing," *BIT* 18 (1978), pp. 184-201.

[Lit78] Litwin, W., "Virtual Hashing: A Dynamically Changing Hashing," *Proc. VLDB 1978*, pp. 517-523.

[NDX81] NDX Retrieval Systems, Inc., "CREATABASE: Performance Manual," NDX Retrieval Systems, Houston, TX, 1981.

[Sho82] Shoshani, A., "Statistical Databases: Characteristics, Problems, and Some Solutions," *Proc. VLDB 1982.*

[THC79] Turner, M. J., Hammond, R., and Cotton, P., "A DBMS for Large Statistical Data Bases," *Proc. VLDB 1979*, pp. 319-327.

APPENDIX. DERIVATION OF H(D,I,j) AND Q(D,I,j)

The ordered sequence of all possible domain values can be modeled by the real number interval [0..1]. Let $A^* = (A_0 \ldots A_{D-1})$ be the ordered sequence that defines the initial values of a dictionary. A^* consists of the endpoints 0, 1 and D-2 random points in (0..1). Without loss of generality, suppose D is odd so that $D = 2N + 1$.

Let f(w) be the probability density function that the value interval $(A_0 \ldots A_N)$ has length w. f(w) is obtained by

observing that 1 point is at [0,0] with probability 1, N-1 points are in (0,w) with probability w^{N-1}, 1 point is at [w,w+dw] with probability dw, N-1 points are in (w+dw,1) with probability $(1-w)^{N-1}$, and 1 point is at [1,1] with probability 1. The probability of all events occurring simultaneously is proportional to the product of their individual probabilities. Integrating this product from 0 to 1 and normalizing yields:

$$f(w) = (2N-1) \cdot \binom{2N-2}{N-1} \cdot w^{N-1} \cdot (1-w)^{N-1}$$

where $\binom{a}{b}$ is a binomial coefficient and

$$(2N-1) \cdot \binom{2N-2}{N-1}$$

is the normalization constant.

After the insertion of I values, the probability b(I,j,w) that the interval $(A_0 \ldots A_N)$ with a length of w will contain j additional values is binomially distributed:

$$b(I,j,w) = \binom{I}{j} \cdot w^j \cdot (1-w)^{I-j}$$

It follows that the probability H(D,I,j) that j of I insertions will have values which fall within the interval $(A_0 \ldots A_N)$, i.e., the first half of the dictionary, is:

$$H(D,I,j) = \int_0^1 b(I,j,w) \cdot f(w) \cdot dw$$

$$= \frac{(2N-1) \cdot \binom{2N-2}{N-1} \cdot \binom{I}{j}}{(2N-1+I) \cdot \binom{2N-2+I}{N-1+j}}$$

Q(D,I,j) is derived in a similar manner. Let g(w) be the probability density function that a value interval has length w. g(w) is obtained by observing that 1 point is at [0,0] with probability 1, 1 point is at [w,w+dw] with probability dw, D-3 points are in (w+dw,1) with probability $(1-w)^{D-3}$, and 1 point is at [1,1] with probability 1. Taking the product of these probabilities, integrating from 0 to 1, and normalizing yields:

$$g(w) = (D-2) \cdot (1-w)^{D-3}$$

The probability Q(D,I,j) that an interval will receive j of I insertions is:

$$Q(D,I,j) = \int_0^1 b(I,j,w) \cdot g(w) \cdot dw$$

$$= \frac{(D-2) \cdot \binom{I}{j}}{(j+1) \cdot \binom{D-2+I}{j+1}}$$

NOTES

[1]This work was supported by the U.S. Department of Energy under contract DE-A505-81ER10977.

[2]Author's current address: Department of Computer Sciences, The University of Texas at Austin, Austin, Texas 78712.

[3]A_0 and A_{D-1} should be the lexically lowest and highest possible values that can occur in the domain. If these values are not present when the file is first examined, they are added to the domain prior to index encoding.

[4]As an implementation note, it is unnecessary to maintain an array of G_i's. Instead, an unused code within a particular code interval can be found easily by searching the directory.

[5]Readers may note a strong similarity between the true order algorithm and its conditions for dictionary recoding, and the node splitting algorithm and its conditions for directory reorganization in dynamic hash-based files [Lar78], [Lit78], [FNPS79]. In spite of this similarity, the analyses which model their behaviors are quite different. Explanations of these differences, too long to be included here, can be found in [Bat82, pp. 65-66].

[6]By symmetry, H(D,I,j) is also the probability that I-j of I insertions will fall within the interval $(A_N \ldots A_{D-1})$, i.e., the last half of the dictionary.

[7]This assumption is identical to that used to derive eqn. (2).

[8]A second possible qualification is that the algorithm does not recode subsections of a dictionary to delay the complete recoding of the dictionary. While this extra qualification may be needed to further specify when the performance of the interval order algorithm provides a theoretic bound, the idea of partial recoding may not be practical. Preliminary investigations suggest that complete recoding is not much more expensive than partial recoding, and complete recoding occurs less frequently.

AN OVERVIEW OF CANTOR - A NEW SYSTEM FOR DATA ANALYSIS

Ilkka Karasalo and Per Svensson
Swedish National Defense Research Institute, Stockholm, Sweden

ABSTRACT

A transportable system for the analysis of large sets of data, forming complex information structures, is being developed at the Swedish National Defense Research Institute, with financial support also from some civilian Swedish government agencies. The system is based on a relational data base handler of new design, permitting efficient data storage and fast evaluation of complex, spontaneous queries. A query language, based on set algebra and oriented towards scientist users, was developed for the system. To this kernel will be added subsystems for interactive sublanguage definition and user communication, data loading, tabular and graphic data presentation, statistical analyses, and data base backup and recovery. The paper presents an overview of the system version now completed.

1. INTRODUCTION

Techniques and tools for the analysis of data have developed rapidly during the last decade. Much of this development effort has been directed towards developing large libraries of statistical analysis procedures and integrating them into user-controlled statistical systems. In recent years, several of these systems (e.g., SAS[1)]) have been supplied with data management and data presentation capabilities, turning them into quite powerful general-purpose data analysis tools. Users of statistical systems seem to be quite satisfied with the facilities provided. So why bother to develop yet another system, which in this light seems to stand a small chance of becoming commercially accepted?

Existing statistical systems serve their purpose well, as long as the data bases to be processed have a simple, static logical structure and are not too large, and moreover, stay that way during the course of the analysis. In many scientific fields, however, the steps in an analysis involve the creation of complex data structures and large data sets. Typically, such situations arise when looking for interactions between several separately monitored classes of phenomena, for example the incidence rate of a disease and various environmental factors. If the kind of interaction is unknown a priori, the data space in which to look rapidly becomes unmanageable.

The last decade's developments in the theory and technology of data base management systems have led to new possibilities for the design of general-purpose software useful for the detection and analysis of unknown kinds of interaction between loosely related classes of data.

The system to be described below was designed with the goal of being able to analyze in depth large and complex data bases, without recourse to problem-specific programming in a general purpose programming language.

The system contains a user language based on an algebra of relations. In this language, a wide class of data transformations may

be expressed and stored as virtual data extensions, or "views", in the data base itself. The evaluation of views is carried out in the framework of a relational data base management system, designed for efficient storage and access of data in a user environment where the "queries" are statistical, i.e., whose results depend on large subsets of the data. The few commercially available data base systems which have enough functional power for this task are optimized for queries, whose answers depend on small subsets of the data base.

2. PROJECT OBJECTIVES

The following design goals were formulated after an analysis of the requirements on a data analysis support system in a scientific, large data base environment[2]:

i) To allow the scientist users to work directly with their data, the system must be highly automatic. In particular, it must not require any tuning or extensive maintenance during its use.

ii) The system had to contain a very high level user language (query language) suitable for the succinct expression and gradual accumulation of complex data transformations. On the other hand, it was considered reasonable to expect from the scientist user an ability and willingness to express these transformations in a formal language.

iii) Complex transformations easily lead to unacceptable execution times unless efficient methods of query optimization are found, not least when expressed in a very high level, "non-procedural" language. Certain classes of queries are inherently impossible to evaluate efficiently. Query classes which could be efficiently evaluated would have to be defined and appropriate algorithms developed[3,4].

iv) The typical user was envisaged as member of a small group of analysts, working either with a dedicated, comparatively small computer system, or with a large, time-shared central computer. The system should allow for easy sharing of results within the group. To access data not originating within the group, conventional copying and loading was considered adequate. Requirements for simultaneous reading and updating of data would therefore exist but did not impose stringent restrictions on the system design.

v) Within the user community many different types of computer would exist. To allow the system to be used on different computer types without unacceptable conversion costs, it would have to be transportable, i.e., written mainly in a high level language subset acceptable by most compilers for the chosen language.

vi) Economy of data base storage space was found important both directly, to reduce storage space costs for large data sets, and indirectly, as a means of reducing the processing time of a query. The more compactly data can be stored, the less data transportation between primary and secondary storage will obviously be required.

vii) A set of general-purpose application functions should be included in the basic system. Requirements for subsystems for

descriptive statistics, basic statistical analysis, data presentation, and bulk data test, input, and transfer were formulated.

viii) A programming user must be able to add new application functions, written in some commonly used programming language, to the system. Such additions always require additions to the user language as well. To support a controlled language growth, a language definition subsystem allowing the incremental addition of new grammatical rules was found highly desirable. Language and function extensibility were thus required.

ix) To enable adequate documentation and retrieval of data and views, a meta-data subsystem would have to contain both system-created and user-provided information about the elements of the data base.

3. SYSTEM STATUS AND ENVIRONMENT

The system, previously called Datalab but for trade-mark reasons renamed to Cantor in this paper, is in its present form a single-user, interactive relational dbms with an unusually powerful query language and storage and access performance characteristics designed to fit the intended application area.

A few facilities remain to be implemented in order to bring the database management system to the intended level of usefulness[5]. Subsystems for descriptive statistics, statistical analysis, and data presentation also remain to be designed and implemented. We will describe here mainly those functions which are already more or less in their final form.

Cantor is designed to run under a time-sharing operating system (or, obviously, on a single-user computer). All details of terminal interfacing, transaction queueing, and communication network management are assumed to be handled by the host operating system, as are the allocation to different users of common resources, such as primary memory and processor time.

It is intended to eventually allow several simultaneous users to share data in its data bases. A user who wants to update a copy of a data (relation) table can do so, as long as he is authorized by the operating system to access the data base which contains it. An update of a nonprivate data table must wait, however, until the user has been granted exclusive access to it by the Cantor system. In this way, a crude but in our opinion sufficient multiple user facility is planned for later versions of the system.

The existing system consists of almost 70 000 lines of Pascal code and 3 500 lines of Assembly language code, comments included.

At present, the system runs only on the Dec-10 computer under the operating system Tops-10. The system is being transported to Tops-20 and Vax/VMS. An implementation for a powerful personal computer ("workstation") is planned.

An independent evaluation of Cantor was recently completed by the Swedish Bureau of Statistics[6]. Allowing for the fact that some parts of the system are still provisional, the evaluation report states that the system permits very complex queries to be formula-

ted and evaluated without difficulty. Also, the report recognizes that although no system tuning is necessary or possible (without recompiling the system), data storage and subset selection are very efficient.

Plans have now been made for a second development phase, with the goal of making the system a marketable product. System security, language facilities, and performance will be improved. Subsystems for data presentation and descriptive statistics will be added. The system will be priced, documented, and maintained so as to allow widespread distribution.

4. DATA TYPES AND DATA OBJECTS

Objects recognized by the system have unique names and a well-defined although not always explicitly declared type. They are either flat objects viz. scalars or tuples or set objects. Sets of tuples of a common type are called relations.

Each kind of data object can have two modes: value and view. A value consists of explicitly stored data. A view is an expression which can be evaluated, ultimately in terms of stored values, to form a new value. Views may have parameters.

Scalar types are predefined: integer, float (optionally with restricted precision or constant exponent), logical, literal (used for names of objects), and text, all extended by the special value UNDEFINED. An unordered set of pairs (attribute name: scalar type) determine a tuple type. A tuple value is an instance of its type, i.e., a set of pairs (attribute name: scalar value). In the same way, a set type is defined as the set of sets of values of a given flat type. A relation is a special case of a set, namely a set formed on a tuple type (even if this tuple type has only one attribute).

Associated with each data base in the system is a meta database, which contains all information required by the system to keep track of stored data and its properties. The meta database is organised as three relations, maintained automatically by the system. Views may be defined whose value depend on the meta database relations.

5. BASIC COMMANDS

The terminal user language is planned to consist of a standard part used for predefined operations and a variable set of private extensions to this language. In the existing version of Cantor, only the standard language is recognized. It contains commands for declaring the name, attributes, and key of a base relation, defining a view, evaluating a parameterless view and storing its value, inserting, removing, and updating tuples of a base relation, adding and deleting non-key attributes of a relation, loading and printing data objects, etc.

6. THE QUERY LANGUAGE SAL

The view concept in Cantor is a natural generalization of the concept of a function procedure (subroutine) in conventional program-

ming languages. A view may have a number of parameters of arbitrary type.

Recursive view references, which significantly extend the expressive power of relational languages[7,8] are not allowed in the present version of the system.

Views are formulated in SAL[9], the query language of Cantor. It was developed under explicit assumptions about the users' educational background. A user with a basic mathematical education at university level should be able to use the language with fairly little training, because all its important concepts are already in his repertoire. In the requirements definition phase of our project a number of existing query languages were studied, but for various reasons no one was considered suitable[5].

The SAL language was designed according to the following basic objectives:

i) It is only used to compute new data from existing data. For example, no input-output statements are part of the language.

ii) The query language is not intended as a general-purpose computing facility. Many kinds of computation will have to be done by special programs interfaced to the dbms. A user should not be misled to use the query language for purposes where conventional programming is the only adequate technique. Control and data structures proper to algorithmic languages were therefore excluded.

iii) A simple and formal structure was desired rather than similarity with natural language, with its many subtle ambiguities.

iv) The intended users should be familiar with the formalisms of elementary algebra and set theory. The language should not introduce concepts outside of this domain unless necessary.

v) Data transformations are usually derived using the same stepwise abstraction process found indispensable when solving nontrivial problems in other domains, such as programming or elementary calculus. The view definition mechanism is suitable for this purpose and was therefore made central to the language.

The language has been successfully used in several pilot applications. Among these are a cancer epidemiology study and a geographical database application. With very few exceptions, the computational problems involved in these applications have been solved within the language. The queries in a test set defined by M. Lacroix and A. Pirotte[10] have also been expressed and evaluated without difficulty.

Operators and expressions in SAL

Operators are functions from operand values to result values, and the result type is uniquely determined by the operator and the operand type(s). If the result can not be computed, the result is UNDEFINED of the appropriate type.

Any syntactically correct non-recursive expression will also have a semantic interpretation provided that referred objects are defined at execution time and provide appropriate argument types for all operators involved.

Unary operators are NOT, ISUNDEF, ROUND, TRUNC, CARDINAL, and the arithmetic functions ABS, SQR, SIN, COS, EXP, LN, SQRT, ARCTAN. They are written in functional notation, e.g., NOT (a).

The binary operators +, -, *, /, DIV, MOD, =, <>, >, <, >=, <=, OR, AND have their usual meaning. They are valid for the appropriate scalar type, except = and <>, which are valid for tuple types as well. Binary operators acting on sets are EQUALS, CONTAINS, CONTAINEDIN, UNION, INTERSECTION, DIFFERENCE, and MEMBER with the conventional set algebra interpretation. Binary operators have fixed priorities and association rules. To override these rules, parentheses may be used.

To operate on tuples, the identification (:), catenation (.(,,,)), and extraction (.) operators exist, as well as the relational operators (=, <>) already mentioned. The purpose of the first three operators is to provide adequate facilities for naming attributes which occur as intermediate or final results of a sequence of operations on relations.

There are three kinds of operators acting on relations, namely functional form operators, the cartesian product operator, and the partitioning operator.

Functional form operators are binary operators with a relation expression as left argument and a flat expression, whose type depends on the operator, as right argument. The right argument must be enclosed in brackets [ ]. They are:

restriction	- WHERE
generalized projection	- no keyword
selection	- SELECT, SELECTMAX, SELECTMIN
aggregation	- COMPUTE, SUM, PRODUCT, MAX, MIN, AVERAGE, EXISTS, ALL, COUNT, CARDINAL.

Restriction and projection produce results of relation or set type. Selection and aggregation produce results of tuple or scalar type.

The cartesian product operator *(,,,) takes one or more relations (factors) as argument and produces a result relation, whose attributes are the union of the sets of attributes of the factors, provided that all attribute names of the factors are different. If necessary, the identification operator is used to achieve this. The set of tuples of the cartesian product is the set of all catenations of tuples of the factors.

The partitioning operator BY has a special form: R BY [e] agop[s] where e is a tuple expression, s a scalar expression and agop stands for any aggregation operator. R BY [e] may be viewed as a

functional form operator with a set of relations as result, which is however not an allowed object type. Aggregation over each of the relations in the set is needed to obtain a result of set type.

7. SYSTEM DESIGN ASPECTS

The system currently consists of five main subsystems: storage and access, search and sort, grammatical analysis, evaluator, and command handler.

The storage and access subsystem

The storage structure of Cantor is a development of the concept of a fully transposed file[11]. Additional principles that have been employed are ordering, dynamic data compression, and B-list structure[5], i.e., an adaptation of the B-tree principle to linear lists of varying length data.

Since transposed files are not well suited for fast random access to single tuples, a special "cache" buffer is kept for the meta database. A record in this buffer corresponds to a meta database tuple.

The tuples of a base relation are ordered according to a sort order implied by the relation key given by the user in the BASERELATION command. While storing a relation, the system applies a data compression algorithm to fixed-length segments of each attribute. This algorithm reduces the number of bits used for each object, and suppresses the storage of repeating values within an attribute subfile ("run length compression"). The combined effect of sorting and run-length compression provides for compact storage of relation tables with multidimensional keys. Other desirable properties of the chosen storage structure are fast sequential and direct read access and ability to update, insert, and delete values with good efficiency and without the accumulation of garbage.

The performance of a prototype system was measured and compared with a commercially available data base system of good quality[12]. The results showed that very significant performance gains could be achieved by combining transposed file storage and data compression techniques.

The search and sort subsystem

Batory[11] showed that search algorithms designed for use with transposed files could outperform commonly used techniques such as the use of inverted files (indexes). In [13], one of us presented theoretical and empirical results showing that a certain class of associative queries, called conjunctive queries, may often be evaluated even more efficiently if the transposed file structure is combined with sorting and run-length data compression. These results led to the design decision not to implement indexes in Cantor, although there exist special cases where the availability of an index would have improved search performance.

The search and sort subsystem of Cantor was designed to take advantage of these observations as far as possible. It contains algorithms for internal and external sorting, duplicate tuple detection, conjunctive query search ("box search"), key lookup, equijoin, set union, difference, and intersection, all designed to work one (or a few) attribute(s) at a time, to match the transposed file principle. Only the conjunctive search algorithm has yet been critically evaluated.

The grammatical analysis subsystem

This subsystem performs syntactic and semantic analysis of command language and query language syntax. When evoked by a STORE q statement, where q is the name of a parameterless view, it produces either error messages or a syntax tree where all references to stored data, to attributes of relations in enclosing expressions (similar to the referencing of non-local identifiers in a block structured language), and to views have been resolved. The types of each partial result are calculated, checked for consistency, and stored in the nodes of the tree.

In the case of a reference to an unevaluated view, its syntax tree is generated and connected to the result tree. Reference to a previously evaluated view is resolved as a reference to stored data rather than to an expression, providing the user with a degree of control which can be used to avoid the generation of very large syntax trees through repeated view substitutions.

The evaluator subsystem

The evaluator subsystem performs optimization, "dataflow net generation" and "dataflow net interpretation".

The optimizer works by transforming the syntax tree into a logically equivalent one, corresponding to a different (more efficient) query formulation, and with special-case information added to certain nodes.

Two important special cases of expression, corresponding to box and equijoin search are detected. Restrictions on factors and subproducts of a cartesian product are analysed and when possible moved so as to become evaluated before the full product is formed, thus avoiding the formation of unnecessarily large cartesian products The optimizer detects such common subtrees of the syntax tree which need to be evaluated only once, using a fast hashing technique.

The optimizer will be extended with certain rewriting rules which reduce the nesting depth of certain functional form operator expressions , enabling faster evaluation of important query classes. These rules are similar to, but more general than, those proposed by Kim[15] for use with IBM:s query language SQL.

Using the possibly optimized syntax tree as input, the dataflow net generator builds a hierarchy of static dataflow graphs. These graphs, in which adjacent operator nodes are separated by buffer nodes, describe the order in which the different operators are to be executed. An operator node is either a simple operator node representing other than functional form operators of SAL, or a composite operator node, i. e. a subgraph representing a functional

form operator. Each edge in the net signifies a dataflow for one attribute or temporary data stream. The dataflow net generator generates edges only for attributes which are part of an expression and thus are needed to produce the required result.

Cantor uses an interpretation technique which is analogous to the operation of a vectorized dataflow computer[16] except of course that only one processor is available. In a multiprocessor environment, several operators could execute concurrently, since the order of execution of operators is not rigidly determined. The use of vector operators distributes interpretation overhead over many elementary operations, invalidating the "folklore theorem" which states that interpretation of query language expressions is fundamentally inefficient.

The interpretation program has two main tasks. When initializing a composite operator, it assigns space to its buffers. Then, it calls the component operators of this operator in some order until all its input has been consumed.

The system contains a large number (about 70) of operators, categorized as constant, scalar, tuple, set, relation, aggregation, and transfer operators. Transfer operators move data between stream buffers and B-lists. Scalar operators perform unary and binary operations, cf. sec. 6, on (streams of) scalars. Set and relation operators are used where the stream technique is inadequate, i.e., for operations which require the entire set as input. Dependent on metadata about their operands, such as cardinality, sort order, relation key, and extreme values for attributes, several of these operators dynamically select which algorithm to use, usually by calling different procedures in the search and sort subsystem. Also, if the optimizer has detected that a restriction, or a part of it, has the special property of a box search, or that a restriction of a cartesian product has components of equijoin type, special relation operators will perform the required function.

As an example, the projection operator in general sorts its input and removes non-unique tuples. However, analysis of relation key information is done in the projection operator, which may thus detect that sorting and duplicate removal are not necessary.

8. SUMMARY AND CONCLUSIONS

A relational data base management system, designed for the analysis of complex statistical data was presented. The system shows several unusual design features, motivated by the intended application area which in many respects poses different problems from more conventional dbms applications.

The system has a powerful, formal query language whose concepts closely follow those of elementary set algebra. Its design is strongly oriented towards fast evaluation of complex queries. Basic design decisions of the storage, search, and query evaluation subsystems were made to this end. The use of ordered transposed files and data compression techniques provide both economic utilization of available storage and fast data access through mechanisms discussed in this paper. Query optimization is performed in several levels of the system.

9. ACKNOWLEDGMENTS

Many individuals and organizations have contributed to this project in different ways. Among the latter, the Swedish Bureau of Statistics deserves special mentioning for its continuing support.

We are directly indebted for parts of this paper to our former colleague, Professor Stefan Arnborg of the Royal Institute of Technology in Stockholm.

REFERENCES

1) SAS User's Guide, 1979 Edition. SAS Institute, Cary, NC, USA.
2) Svensson, P.: Om forskarens datormiljo (On the scientist's computer environment), in Swedish. FOA Rapport C20215-D8, Jan. 1978. Swedish National Defense Research Institute, Stockholm.
3) Arnborg, S. and Svensson, P.: Fast multivariable query evaluation. FOA Rapport C20189-D8, Aug. 1977. Swedish National Defense Research Institute, Stockholm.
4) Arnborg, S.: On the complexity of multivariable query evaluation. FOA Rapport C20292-D8, March 1978. Swedish National Defense Research Institute, Stockholm.
5) Arnborg, S., Elvers, E. and Svensson, P.: Design specification for Datalab - a system for data analysis based on the relational model of data. FOA Rapport C20326-D8, Oct. 1979. Swedish National Defense Research Institute, Stockholm.
6) Mourgues, K. and Strid, P.-O.: SCB:s utvardering av den forsta etappen i utvecklingen av Datalab (Evaluation by Statistics Sweden of the first stage in the development of Datalab), in Swedish. March 1983. Swedish Bureau of Statistics, Stockholm.
7) Aho, A. V. and Ullman, J. D.: Universality of data retrieval languages. Proc. 6th ACM Symp. on Principles of Programming Languages. ACM, Inc., New York 1979.
8) Chandra, A. K. and Harel, D.: Structure and complexity of relational queries. Proc. IEEE Symp. on Foundations of Computer Science 1980. IEEE Inc., New York 1980.
9) Arnborg, S.: A simple query language based on set algebra. BIT 20 (1980), 266-278.
10) Lacroix, M. and Pirotte, A.: Example queries in relational languages. MBLE technical note N107, Jan. 1976. MBLE Research Laboratory, Brussels.
11) Batory, D. S.: On searching transposed files. ACM Trans. on Data Base Systems, 4, 4 (Dec. 1979), 531-544.
12) Svensson, P.: Performance evaluation of a prototype relational data base handler for technical and scientific data processing. FOA Rapport C20281-D8, Dec. 1978. Swedish National Defense Research Institute, Stockholm.
13) Svensson, P.: On search performance for conjunctive queries in compressed, fully transposed ordered files. Proc. Fifth Int. Conf. on Very Large Data Bases, IEEE Inc., New York 1979.
14) Svensson, P.: Contributions to the design of efficient relational data base systems. TRITA-NA-7909, April 1979. Dept. of numerical analysis and computer science, Royal Inst. of Technology, Stockholm.
15) Kim, W.: On optimizing an SQL-like nested query. ACM Trans. on Data Base Systems 7, 3 (Sept. 1982), 443-469.
16) Giloi, W. K.: Towards a taxonomy of computer architecture based on the machine data type view. Proc. 10th Ann. Symp. on Computer Architecture, Stockholm 1983. IEEE, New York 1983.

Statistical Database Research Project in Japan

and the CAS SDB Project

Kohji Shibano
Tokyo Scientific Center
IBM Japan

Hideto Sato
Economic Planning Agency

Abstract

In this paper, we describe working statistical database systems in Japan. Most are special purpose systems. In 1983, a comprehensive statistical database research project MUSE began in academic societies. In conjunction with it, EPA is going on an SDB development project (CAS SDB project). CAS SDB plans to handle a large amount of statistical summary data and to support a variety of social scientific uses. We describe statistical meta-data problems and a current design for a statistical meta database. We also describe a current implementation of CAS SDB. It was implemented on a relational DBMS and has an interface with SAS. We are now testing the capabilities of CAS SDB in real statistical applications.

1. Introduction: Large Statistical Databases in Japan

There are a number of large statistical databases and database systems in Japan. Many government organizations, such as the Economic Planning Agency, Ministry of International Trade and Industry, Bureau of Statistics, National Land Agency, etc., are interested in developing statistical databases.

Most current working large statistical database systems in Japan are special purpose, e.g., SDB of PPIS (Policy Planning Information System: MITI) [FUJI83] for industry and trade data, ISLAND (NLA) system for regional grid data. HSDB (Hiroshima University) [IKED82] is another type of statistical database system designed for the management of summary data which is consistent with micro data. However, there are few statistical database systems constructed for multi-purpose statistical use.

The comprehensive statistical database research project MUSE (Multi-Use Socio-Economic Statistical Data Bank) began in academic societies in 1983 [SHIS83]. The center of the MUSE project is the University of Tsukuba. In conjunction with the MUSE project, the Economic Planning Agency has begun to develop a new statistical database management system, named CAS SDB, which would be the core software of EPA. This project is going on with the help of the University of Tsukuba, Mitsui Knowledge Industry, and Tokyo Scientific Center (IBM Japan).

In the Japanese environment, large statistical databases are usually databases of statistical summary data because governmental statistics in micro data form are not open to public use. Our first object is to be able to handle large volumes of statistical summary tables in an easy way.

2. The MUSE Project

The MUSE project includes the following components and corresponding research groups:

1. Database management research group
2. Distributed database and econometric modelling software research group
3. Database construction and research on multiple uses of MUSE SDB
    - SNA (System of National Accounts) and micro data set research group
    - Multi-sector economic data research group
    - Social and political data research group
    - Regional economic data research group
    - Econometric data research group

The database construction group plans to collect most Japanese machine-readable statistics.

The MUSE project is intended to support social science research. MUSE SDB will handle a variety of summary statistics from heterogeneous sources collected by many different organizations. It will support a variety of different research and planning uses. We have a difficult task to describe, store and utilize many kinds of published statistics.

Hereinafter, we shall explain our treatment of these problems in the CAS SDB which is developed with the cooperation of the MUSE database management research group.

3. Meta Database Problems of CAS Statistical Database

In a statistical database unlike most conventional business databases, the amount of information concerning an object database often is too large for users to comprehend for the following reasons:

- Abstract entities may consist of many entities
- There may be a large number of entity types, attributes, domains, and relations
- Many candidate schemata for the same information are possible [KENT82, HOTA83]
- The amount of descriptive or meta-data information may itself be very large.

As a result, statistical database systems need to be able to handle many files, meta-data information (the code book problem) and also to handle many alternative representations. Few database systems can provide such an environment. For example, it is difficult to transform a set of time series data to a cross-sectional form where the set of series are compounded according to the same attribute, TIME. In fact, a set of income time series for prefectures can be regarded as cross sectional income data categorized by time by prefecture. Conversely, a cross-sectional statistical data designed for a single survey, such as population census data, is difficult to treat as a set of time series. For example, in order to retrieve time series population data for Hiroshima Prefecture from 1961 to 1980, we need to access 20 files and select Hiroshima Prefecture from each; then to combine 20 results of the selection. This is a very complex and cumbersome task for the user. As a result, we need a database system which can handle both time series and cross-sectional data.

CAS SDB plans to allow users to have many kinds of external views. Thus we must take special account of how to represent and manage complex meta-data information. In order to do this, we must first describe the meta-data in a schematic manner and to give precise descriptions of statistical data processing operations that may be expected.

4. Design of Meta Database

Although record-based representations of data have some limitations [KREP83], they are still an easy-to-understand representation for statisticians. Statisticians' terms 'samples' and 'items', naturally correspond to 'entities' and 'attributes', in terms of database researchers.

We think that the best way to deal with statistical data is not to abandon the record-based representation of data but to add statistical meta-data to it.

Schematic representation of a meta database is shown in Figure 1. In order to manipulate meta-data information and the data itself in a similar manner, we made a meta database and its object database, both of which have the same structure.

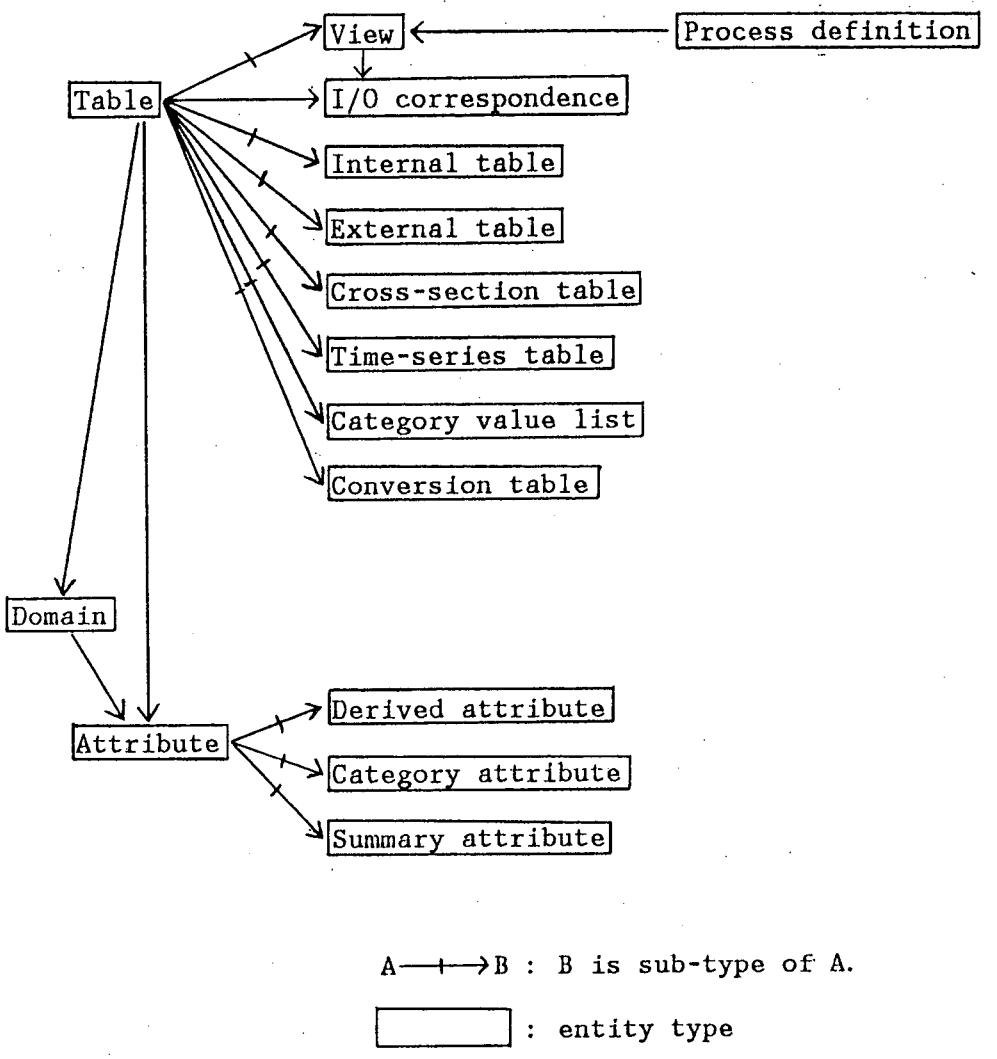


Figure 1. Schematic representation of meta database

The main entity types of the meta database are 'table', 'attribute', and 'domain'. 'Category value list' is a table of values of a category domain. 'Conversion table' describes the correspondences between the values of two category domains, e.g., a reclassification rule from minor categories to major categories. The 'table' and 'attribute' files describe the current status of the object database. The domain file describes the range of domain values and/or the name of the 'category value list' table.

In a statistical database, there is sometimes confusion between two types of abstraction: value levels and type levels. For example, when we think of a certain category, such as 'California' as a state in the U.S., we recognize that 'California' is an entity or an entity value. On the other hand, when we think of 'California' as a set of counties in California State, we recognize 'California' is an entity type or an set of entities.

We need to distinguish between an entity value and an entity type, but statisticians always recognize this difference by means of direct mentioning or from context. We plan to have the database system recognize this difference in a similar way.

In addition, enumeration of domain values is not sufficient to distinguish real values and missing value codes. Many kinds of exceptional values are required in statistics and statisticians always need to distinguish different types of exceptional values (such as impossible, secret, lack of continuity, tentative values). For example, UN energy statistics includes "impossible" and "missing" values in different codes. A statistical database system should support these different missing data codes. In CAS SDB, such differences can be described in the 'domain' file of the meta database.

5. Current Implementation of CAS SDB and Future Plan

A preliminary version of the experimental CAS statistical database management system was implemented at EPA. It supports some meta database facilities, and it was implemented on a relational DBMS. It has an interface with SAS [SAS79]. We are now testing the capabilities of CAS SDB for real statistical applications.

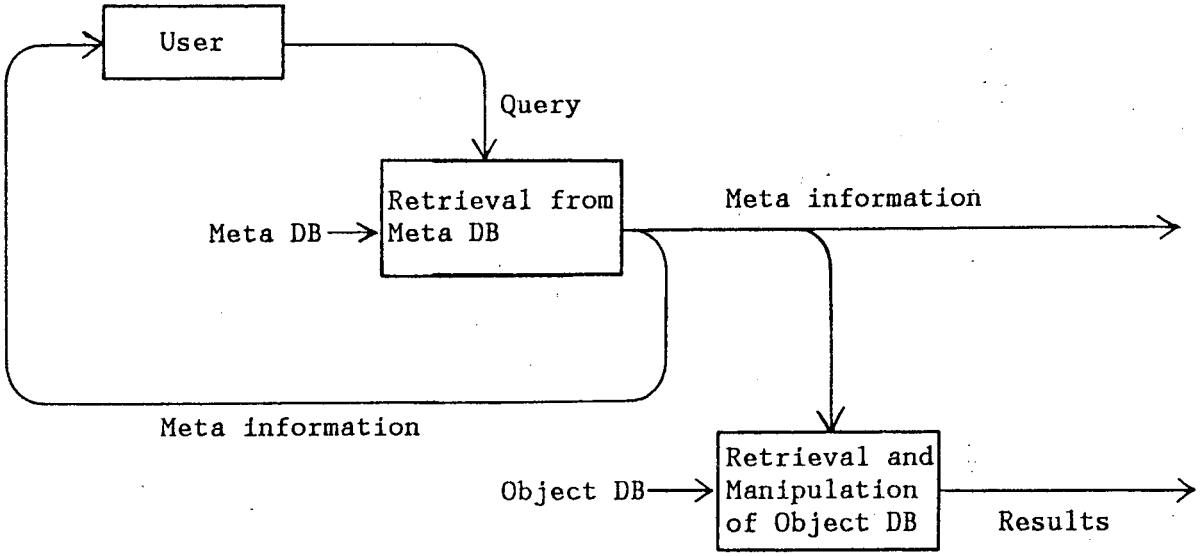


Figure 2. Schematic Representation of Operations of CAS SDB

The preliminary version also supports restriction, projection, reclassification of categories using conversion tables, and automatic aggregation. But it does not yet support view realization, automatic join or automatic aggregation directly inferred from meta information (directory driven).

The user interface of our test version DBMS is a full screen menu interface like IBM's SPF (System Productivity Facility). Its operation is similar to SPF member list operations. First, the user retrieves or browses meta information from the meta database and selects a desired table from the table list or the attribute list displayed on the screen. Next, the user specifies operations on the table. In such a manner, he can manipulate the object database itself. Information once entered never has to be re-entered. As shown in Figure 2, a user does not need to enter table names or attribute names in the meta database. The user does not need to look through a code book but only to browse a certain domain value list.

The above consideration is highly necessary in Japan, because to input Japanese characters is still cumbersome even using computers. So to decrease input is more important in the Japanese environment than in the U.S..

Alphabetic or English expression of information is not sufficient for Japanese users. Japanese users need precise definitions of meta objects of statistical data in Japanese. Alphabetic or English expressions only play the role of aliases.

For the next implementation of CAS SDB, self-descriptive database management facilities are planned. We plan not only to manage the meta database (dictionary), but also to support operations on statistical data which is abstract and has special semantics.

6. Concluding Remarks

CAS SDB is a computer system for operations, such as searching statistical tables, joining and transforming the tables, and preparing data for statistical analyses or modelling.

However, meta database facilities which are the core of CAS SDB, require the users to input additional information. Probably they feel CAS SDB is cumbersome. Hence we have to pay closer attention to the user interface, in order to simplify its use. The most urgent issue for us is to devise an interface with which users can do their complex jobs with the least effort or less effort than in a conventional way.

Another important problem for us is what can be a conceptual schema for a statistical database. A conceptual schema lets users know what information is in the database, and what kinds of views may be allowed.

We are now studying a type of conceptual schema which will permit us to describe heterogeneous statistical data in an integrated manner, and in which alternative representations of equivalent information can be treated as synonymous.

Acknowledgment

Our research has been carried on by the CAS SDB development group organized by the Economic Planning Agency. We have received several helpful comments and suggestions from the committee members of the group. Especially, we would like to thank Professor R. Hotaka of the University of Tsukuba for his continuing encouragement and insightful suggestions, and Professor H. Ikeda of Hiroshima University. We also thank S. Saeda of Mitsui Knowledge Industry, and J. Sakamaki and Y. Toda of Fujitsu Corpo-

ration, who developed the early and current verions of CAS SDB with us. We are also grateful to J. McCarthy of LBL for many helpful suggestions.

REFERENCES

[EPA82] Economic Planning Agency, "Design of Cross-Section Database" (in Japanese), March, 1982.

[FUJI83] I. Fujimori, "Statistical Database of MITI" (in Japanese), Proceedings of the 2nd Conference on the Advancement of Statistical Data Processing -Statistical Database-, Tokyo Scientific Center, IBM Japan, January, 1983.

[HOTA83] R. Hotaka, "Statistical Data from the Viewpoint of Database," Proceedings of the 2nd Conference on the Advancement of Statistical Data Processing -Statistical Database-, Tokyo Scientific Center, IBM Japan, January, 1983.

[IKED82] H. Ikeda, Y. Kobayashi, "Additional Facilities of a Conventional DBMS to Support Interactive Statistical Analysis," Proceedings of the First LBL Workshop on Statistical Database Management, Lawrence Berkeley Laboratory University of California, March, 1982.

[KENT82] W. Kent, "Choices in Practical Data Design," Proceedings of the International Conference on Very Large Data Base (VLDB), 1982.

[KREP82] P. Kreps, "Semantic Core Model for Statistical and Scientific Databases," A LBL Perspective on Statistical Database Management, December, 1982.

[SHIS83] S. Shishido, "Multi-Use Socio Economic Statistical Databank" (in Japanese), Tokei, Vol. 34, No. 1, January, 1983.

[SATO81] H. Sato, "Handling Summary Information in Database," SIGMOD 81, 1981.

[SATO83] H. Sato, N. Tamachi "Meta Database Management System for Statistical Database" (in Japanese), Proceedings of the 28th Conference of IPSJ, March, 1983.

A STRATEGY FOR IMPLEMENTATING A COMPUTER EFFICIENT DATABASE MANAGEMENT SYSTEM PRELIMINARY RESEARCH REPORT

John Dixie, Philip Wake

THE OFFICE OF POPULATION CENSUSES AND SURVEYS, TITCHFIELD, FAREHAM, HANTS UNITED KINGDOM

Abstract
The United Kingdom census office has a need to improve access to large data sets such as the population census, registrations of births, marriages, deaths and diseases, and major social surveys. In general terms the need is to access a small number of data fields from a large number of records (with or without filtering) for applications in which the data volume is encormous but stable (say 100Mb or greater). A DBMS with transposed file structure (like RAPID) would appear to be ideal for this.

A strategy is proposed for implementing such a system on ICL 2900 range computers. Questions are raised conerning enhanceability, programming languages, data packing and file structure, storage of meta-data, and the use of the operating system. A recently implemented secondary (macro) data TDF (Transposed Datastore File) is described and the possibility of using the same structure for primary (micro) data is discussed.

1. OPCS' REQUIREMENT

1.1 The Office of Population Censuses and Surveys (OPCS) is one of the main collectors of data for statistical analysis in the UK. It has responsibility for conducting the population census, for processing registration data on births, marriages, deaths and diseases, and for government social surveys.

1.2 A need for better access to data has been identified in three main areas:

CENSUS DATA: The 1981 census of population primary data comprises some 8,000Mb (albeit with some duplication under our current system of processing). The next census may possibly be held in 1986, but more likely in 1991, and it is necessary to begin planning the overall strategy now. The problem to be solved is the inflexibility of the current datastream approach in which supposedly small selections of data fields are serially extracted on to magnetic tape for the purpose of tabulating a series of related tables. These datastreams have to be planned well in advance for the efficiency of the approach to be realised. In practice this planning can go wrong, and it is not known for more than one of the datastreams to have to be accessed for one table. Last minute adhoc tabulations are a particular problem, and ther is a tendency to design the datastreams to contain more data fields than they should in an attempt to avoid later problems. There is no doubt that if an economic system could be developed to avoid the necessity for creating datastreams (or make it possible for datastreams to be created less far in advance where the data has to be sorted) many census data users would more than pleased.

REGISTRATION AND MEDICAL DATA: Data for a single dataset, eg births for a single year, is relatively manageable. But there is a need to be able to make links between data sets, for example to analyse and tabulate deaths data over a ten year period (the life of a particular version of ICD). Even for a single dataset (around 100Mb) a transposed file storage system which would make it feasible for ad hoc tabulations to be run at the terminal in a few minutes would be more than welcomed by OPCS statisticians, who currently find that a 10% sample is all they can manage on their assigned computer budgets. We also have a major longitudinal analysis project which

involves following the progress of a 1% sample of population over time starting from the 1971 Census. This data will amount to some 750Mb, from the 1971 and 1981 Censuses and the intervening events, and will continue to grow.

DATA COLLECTED BY SOCIAL SURVEYS: The larger surveys, such as the Continuous Manpower Survey and the General Household Survey, involve substantial volumes of data and computer efficiency is a consideration. The ad hoc surveys are much smaller, and do not present the same problems (but might benefit from a system developed for the above applications).

1.3 Our research indicated that a relational database system using a transposed file structure, such as RAPID, would be most appropriate for our needs. However, as we do not operate IBM equipment, the option to use RAPID is not open to us. We therefore studied commerically available software in order to see if any of it could provide an acceptable degree of efficiency. During 1982 we carried out a full trial of the most promising relational database management system, called RAPPORT.

We found that the relational structure was well suited to our work and that relational databases seem to be relatively easy to understand and design. But we found that processing costs for statistical work were exceedingly high.

1.4 Finally we made a preliminary study of the problem of implementing RAPID on ICL 2900 computers. The task was seen to be daunting, to say the least, but we noticed that much of the package was concerned with updating the database with new data (rather than just amending current data). It seemed that this concern with updating created much complexity in the package, as also did the concern with packing data down to the smallest number of BITS by recoding. It occurred to us that if we assumed the data was totally stable (a fair assumption considering the way we process our data at the moment) and that packing below the BYTE level was too heavy on mill time to make the data storage savings worthwhile, we could build a new system from scratch in a reasonable period of time. As far as data packing is concerned we felt that if recoding to a sequential code is to be done, it should be done visibly and logically by the user (as if deriving a new variable) because then our TAU tabulation package could take advantage of the recoding by using direct table look-ups.

1.5 From this study evolved the stategy for implementation described in this report. As one might expect at this stage in the project we have rather more questions than answers, but the position looks hopeful.

2. A POSSIBLE STRATEGY

2.1 Our proposed strategy for implementing a statistical database management system is strongly influenced by our circumstances. We need to obtain as early a return as possible on our investment, and to limit our rate of investment to what can be provided in the current economic climate. This implies starting with a simple system, but one which has been designed so that more advanced facilities can be added as resources become available. We propose therefore to implement first a system to hold static edited microdata. We can then develop by adding utilities for expanding or contracting the data set, querying and altering ad hoc records, extracting subsets and PERHAPS interfacing to our editing systems. The facility to join two ordered relations to the form of a simple hierarchy (as used for census data) will certainly be an early enhancement to the software. We feel very strongly at the moment that the system will NOT be enhanced later to directly add new records to a datastore in amongst current records (either physically or logically). But there will be a facility to amend current data or create a new relation (these do not cause file design problems).

2.2 The overall concept will be one of loading data into the database format from whatever data collection and editing system is currently used and using the datastore software to access the data efficiently. Of course there would be nothing to stop the user reserving space for derived data fields and computing and storing these in the datastore for the period during which they were required. We would expect users to work in this way.

2.3 For ease and speed of first implementation, we propose to use a high level language. This may sound illogical when the ultimate aim is high computer efficiency, but we hope to be able to contain the key processes within a limited set of modules, which can subsequently be rewritten in a low level language if absolutely necessary (a full system in the high level language would always be maintained for both maintainability and forward compatibility).

* What languages should we consider in our design experiments for the first implementation?

* In particular, is there any facility which would limit the choice to those languages that have it?

There is another reason for beginning in a high level language. That is to ensure the portability of the system. In the UK the statistical function is not centralised in the sam way as in many other countries, and we expect a variety of machines to come into use. The use of common software is a major issue at the present time, and in our view a worthwhile cause.

2.4 The use of a high level language would appear to allow packing only to the byte, rather than bit, level in the initial implementation.

* Is packing to the type level adequate for the envisaged production systems?

* Should we deviate from a fully transposed structure to allow grouping of short codelist data fields in order to minimise storage, and if so how should they be grouped?

* Should data recoding to facilitate reduction to te minimum number of bytes be part of the system, or should we just load the data given? (Certainly the latter in the first version, but should allowance be made for that kind of enhancement?)

2.5 It might be advantageous for various reasons (statistical, managerial and economic) to link the database system to our data dictionary system, which has been developed specifically for statistical work. We would certainly aim to make the loaded TDF version of data invisible to the user.

* Can we avoid the need to store meta-data along with the data in the database?

* What would be the disadvantages of doing this?

* What are the implications of separating the data from the meta-data for privacy protection? The latter is of prime importance to us, particularly for the census and the longitudinal study.

2.6 We would like to use the operating system as little as possible, or at least contain its use to within specific parts of the system, in order to minimise the problems of portability.

* Is there a conflict between this and the facilities needed for a TDF system?

3. A DESIGN BASIS FOR INITIAL DEVELOPMENT

3.1 The initial development of this TDF (Transposed Datastore File) system for primary data will be based on the experience gained on the secondary data TDF system which has been successfully used with the OPCS mapping system STATMAP. The data storage element was the word (instead of the byte) and each data element occupied precisely one word (instead of a variable number of bytes) but it is believed that the same simple technique is applicable in the more complex world of primary data fields by:

- using VME to ensure efficiency at the byte (instead of word) level
- not packing data below the byte level
- making the data element the byte to correspond with the data storage element
- let the software worry about which bytes need

retrieving to make up a requested data field (ie the TDF structure should admit nothing about groups of bytes)

3.2 It is thought to be worthwhile to make the above 'adjustments' to contrive a TDF system for primary data which is as single and uncluttered with complications as is the secondary data TDF system because of the results achieved. In the mapping TDF we have stored (in a 100Mb file) 188 SAS (Small Area Statistics) 1981 Census derived variables for each of 130,000 EDs (Enumeration Districts) in Great Britain. For most mapping purposes the user needs to extract 4 variables for all the EDs in a chosen window of CB. The process of extracting 4 variables for the whole GB window takes between 36 and 108 elapsed seconds. The amount of mill time used is less than 10% of the average elapsed time (important if elapsed time estimates are to be meaningful in the context of a busy multi-program computer).

3.3 If the same simplicity (and therefore efficiency) can be attained, the access speeds achievable, in applications where such speed is USEFUL, are illustrated by the following examples: (What is meant by USEFUL is explained after the examples.)

A census tabuation of 4 variables (data fields) might involve accessing only 4 bytes per person. The 100,000 blocks of data (assuming 2Kb per block, there would be 25,000 segments in a census TDF) would be accessed in 60 to 180 minutes elapsed time (12 mill mins). That's less than 3 minutes (12 mill secs) per county. Where filtering is required, only the variable to be filtered need be accessed at first, the other variables only being accessed when the filter test is positive.

A single year's deaths tabulations involving similar variables would require the access of 1000 2Kb blocks in 36 to 108 seconds (7 mill secs). Accessing ten year's worth of deaths at the terminal becomes a feasible proposition when the elapsed time reduces to 6 to 18 minutes.

* What does USEFUL (above) mean?
The measure used (elapsed time) does not necessarily mean that the intention is to make access to data quick in a while-you-wait fashion. Where there is no need for terminal access to data, the facility for short access times has two main beneficial effects: The effective work-capacity of the computer is increased because more access runs can be done in a day, and the need for dump-and-restart is reduced because of the lesser risk of the computer going down during a particular run.

3.4 We will now describe how the secondary data TDF system works and then show how a primary data TDF system might be designed to take advantage of the simple structure and superb efficiency offered.

4. TDF FILE STRUCTURE CONSIDERATIONS

(The mapping TDF is a file set up for the OPCS STATMAP thematic mapping system currently being used to produce point maps of 1981 census data)

4.1 The mapping TDF is designed as follows:

A 100Mb file on an EDS 200 is divided into 261 segments. Each segment is precisely one cylinder containing 188 2Kb blocks. Each block contains the data for 1 variable (a single SAS derived count) for 500 consecutive records (Enumeration Districts). The 500 four-byte integer counts in a block represent the smallest unit of data transfer between computer memory and disc. All the blocks in a particular segment contain the counts for the same 500 areas, but represent different variables. Thus up to 188 variables for each of 130,500 EDs are held in the TDF (block 89 in segment 2 contains the data for variable 89 for the 501th to the 1000th ED).

This partially transposed structure has the advantage that any number of variables for 500 consecutive records can be accessed without disc read-head movement. It also has the advantage that the data loading process is perfectly serial and needs a program data core size of only 188*2Kb. Conversely, entire-record access (eg for editting) is also achievable in a serial fashion, again involving a program data core size of only 188*2Kb.

Unlike the completely transposed and non-aligned (conventional) structure, the partially transposed and aligned structure enables the fastest possible access speeds to be achieved and enables entire-record access to be achieved without loss of efficiency compared with using the original serial file from which the datastore was loaded.

4.2 For a primary TDF:

Precisely the same datastore structure could be used to hold 522,000 primary records with 188 bytes per record. Each block would contain the data for one part-variable for 2000 consecutive records. A sequence of blocks would hold the data for a whole variable, the number of blocks being the same as the number of bytes needed to hold the variable - 1 block for a 1 byte variable, 2 blocks for a 2 byte variable, and so on. Only the software needs to know which blocks represent which parts of which variable.

The objective would be for the user to deal with the same Cobol-like record layout that he would normally deal with for accessing his data. The only difference would be that if the data is TDF loaded only the parts of the record layout being used would be 'filled in' by the software each time the read-next-record subroutine is called. An initialising subroutine would have to be called to set the scene.

It is envisaged that the TDF access and update subroutines would be used in the TAU tabulation package. A separate utility program will be needed to load data into TDF format, given its meta-data and an appropriate disc file.

4.3 The initial development will involve the loading of a specific set of data into TDF format (yet to be chosen, but 1 years worth of deaths data is a likely candidate). An ad hoc facility will be put into TAU to enable the TDF to be accessed for nominated variables, and the facility tried out on interested users, and empirical data collected to assist further development.

UTILIZATION OF CHARACTER REFERENCE LOCALITY FOR EFFICIENT STORAGE OF DATA BASE

M. A. Bassiouni
Department of Computer Science
University of Central Florida
and
K. A. Hazboun
Pennsylvania State University

Abstract

This paper describes an efficient character encoding method based on practically observed properties of character occurrences within files. The method is specially designed for the encoding of files containing both numeric and alphabetic fields. It is therefore particularly attractive for the storage of many large database files, encountered in practice, which are amenable to statistical analysis. The technique of using m-grams is also incorporated to enhance the compression efficiency. Numerical tests have given favorable results for the proposed method.

Index Terms: encoding techniques, data compression, character reference behavior, storage efficiency, binary trees.

I. INTRODUCTION

Many encoding methods have been developed in the literature [1-9] for representing data with fewer bits than are required by a conventional fixed-length character representation. These methods, usually called data compression techniques, have wide applications in information processing systems, where character representation has a considerable effect on the efficiency of file storage on magnetic media and its transmission down telecommunication channels.

One of the popular data compression techniques is that developed by Huffman [3] who took advantage of the fact that characters do not occur with equal frequency. Accordingly the most frequent characters are assigned to the shortest codes and all larger codes are constructed so that shorter codes do not appear as prefixes. Simply, the Huffman's method is to build a decode tree (i.e., a binary tree in which external nodes represent characters) having minimal external path length. If p(i) is the expected relative frequency (or the weight) of the ith character and d(i) is the distance of the external node for the ith character from the root node, then the Huffman's decode tree minimizes the quantity

$$\sum_{i=1}^{N} p(i)*d(i)$$

where N is the size of the character set. A recent implementation of Huffman's method along with compression statistics are reported in [6].

Another observed property of character occurrence within files is the arbitrary alternation of alphabetic, numeric data, and spaces within predefined fields of one or more characters. The record layout of most commercial files is designed such that the majority of fields are dedicated to contain a sequence of numeric or alphabetical data consisting of several characters in each field. In turn, the numeric field may be zero filled, while the

alphabetic field may contain a succession of blanks. Consequently, the majority of characters within each field are limited to a subset of the global character set (we call this subset a locality set). Such locality of character reference behavior may extend over two or more adjacent fields. A recent compression technique [2] that makes use of the distributional as well as the correlational characteristics of character reference has been proposed by the authors of this paper. The technique is a two-level hierarchy of Huffman's type binary trees. The trees in the first level (called the local trees) are identified and constructed based on the divisions (groups) of characters as induced by the property of locality of character reference. The trees in the second level (called the failure trees) are used to indicate the transition from one group to another when there is a change of the locality set.

In addition to the variability in frequency and the locality of reference, it has been also observed that certain sequences of characters occur more frequently than others. One approach to data compression is to replace high-frequency variable-length fragments of words by fixed-length codes pointing to a compression table containing these high-frequency fragments. Mulford and Ridell [5] , Ruth and Kreutzer [8] , and Schwartz and Kleiboemer [9] have used Huffman's encoding with frequently occurring bigrams (sequences of two characters) or m-grams (sequences of m characters) added to the character set. This technique achieves tighter compression, but there is a tractability problem in finding the optimal m-grams for a given text.

In the following sections, we describe an encoding technique that combines the advantages of the techniques mentioned above. The method, which uses one level of Huffman's type trees, will be explained in stages, then a formal decoding algorithm is presented. Next, results of the numerical tests are reported.

2. UTILIZATION OF CHARACTER REFERENCE LOCALITY

For simplicity, we shall use a restricted character set to illustrate the technique. Assume that we have a set of 6 characters consisting of 3 alphabets and 3 digits. Table I gives the relative frequencies of these characters along with the binary code obtained from applying the Huffman's method on these statistics as shown in Figure 1 (the value inside each node represents the relative frequency of all leaf nodes in the subtree whose root is this node).

Table I

Character	Relative Frequency	Huffman's Code
A	9	01
B	6	101
C	5	111
1	9	00
2	6	100
3	5	110

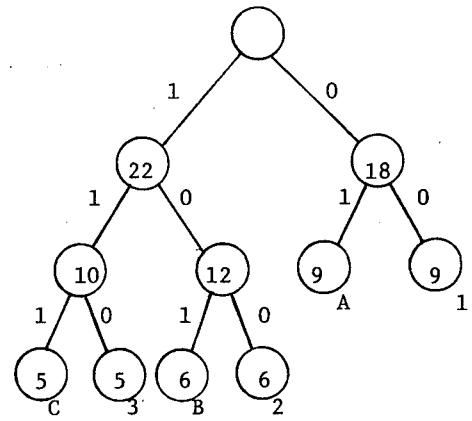
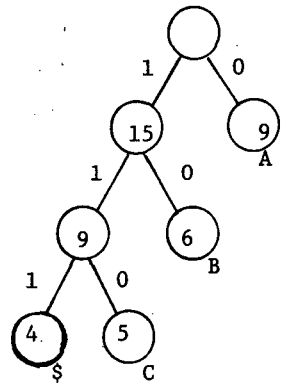


Figure 1. Huffman tree for the character set of Table I

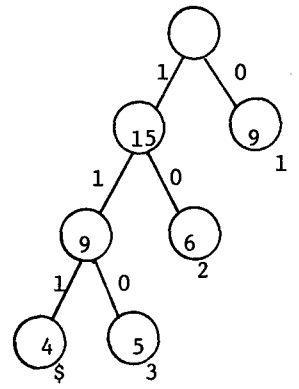
Now let us assume that in addition to the statistics of Table I, we also know that adjacent characters within text tend to fall within one of two groups (locality sets): alphabets or digits. Let us define the average sequential-run length of a group to be the expected number of consecutive characters of this group before a character from a different group appears in the text. The reciprocal of the average sequential-run length of a group indicates the frequency of character switching (i.e., change of locality) from this group.

For the example of Table I, let us assume that the average sequential-run length of both the alphabet and the digit group is 5, i.e., on the average 5 characters of the same group will appear consecutively before a character from the other group interrupts the current locality. To make use of this alternating behavior, we construct two Huffman's type decode trees: the alphabet tree and the digit tree. In each tree, we introduce an extra imaginary character, $, called the switch indicator, which is merely used to indicate that the next character is from a different group and thus the other decode tree must be consulted.

Figure 2 shows the alphabet and digit trees constructed using the statistics of Table I and an average sequential-run length of 5 as explained above. The relative frequency of the switch indicator $ within the alphabet tree is given by (9+6+5)/5 = 4, i.e., every 20 alphabet characters will contain on the average 4 switches to the digit group. Note that in general the relative frequency of $ within the two trees can be different.



alphabet tree



digit tree

Figure 2. Alphabet and digit trees for the character set of Table I

As an example, the string ABAAC11123AABCB will be decoded in 31 bits (assuming we start from the alphabet tree) as follows:

0	10	0	0	110	111	0	0	0	10
A	B	A	A	C	$	1	1	1	2

110	111	0	0	10	110	10
3	$	A	A	B	C	B

The Huffman's scheme (Figure 1) requires 37 bits for the same string. It is easy to show (assuming the statistics of Table I and the average sequential-run length reflect the true figures found in practice) that the average number of bits per character for the Huffman's method is 2.55, while that of the scheme of Figure 2 is 2.4. This means that the savings achieved by using shorter representation (as a result of

locality) exceeds the overhead introduced by the switch indicator.

3. OVERLAPPING LOCALITY SETS

In general, the locality sets do not have to be mutually disjoint, i.e., a given character can belong to more than one locality set. As an example, let us add an extra character, the blank character ␣, to the set of Table I and assume that it has a relative frequency of 10. Furthermore, assume that the blank character has equal probability to occur in any group. Figure 3 shows the Huffman's tree for this set and Figure 4 gives the alphabet and digit trees using an average sequential-run length of 10 in both trees (Note that the relative frequency of $ is 2.5 in this case).

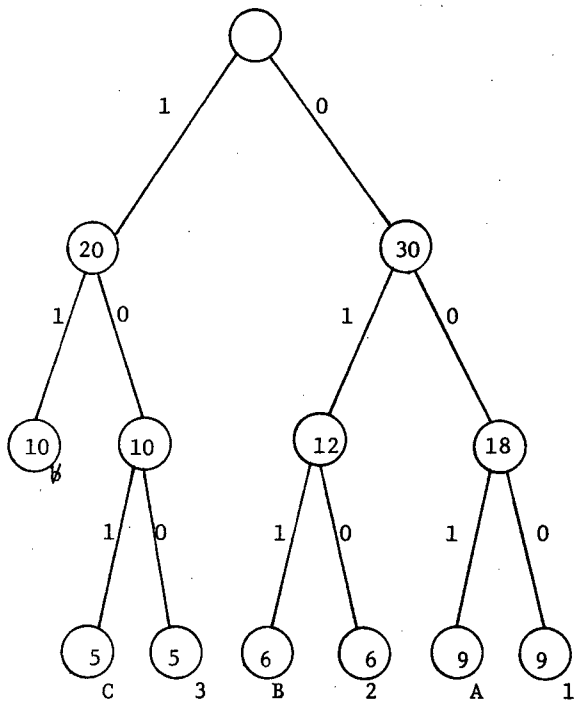
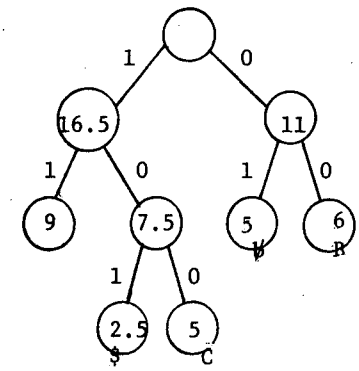
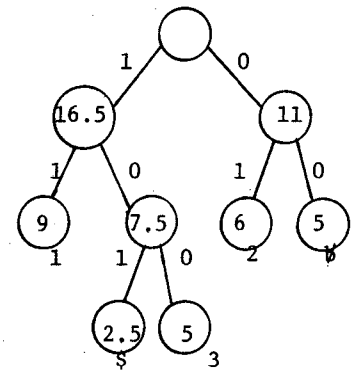


Figure 3. Huffman's tree for the modified set



alphabet tree



digit tree

Figure 4. Locality trees for the modified set

For example the string AA␣B␣C11␣21␣ is represented by 28 bits as follows

11	11	01	00	01	100	101	11
A	A	␣	B	␣	C	$	1

11	00	01	11	00
1	␣	2	1	␣

Note that we deliberately made the code for the blank character be different in the two trees to emphasize that no ambiguity is introduced as a result of having the same character in more than one locality set. This is because only one locality tree is used at a time to control the decoding process, with control being transferred to the other tree whenever the code of the switch indicator is encountered. Using straightforward calculation it is easy to show that the average

number of bits per character for the Huffman's scheme (Figure 3) is 140/50 = 2.8, while that of the scheme of Figure 4 is 125/50 = 2.5.

4. MULTIPLE LOCALITY SETS

So far, we have dealt with only two locality sets. In practice, it might be preferable to have more than two locality sets. A general scheme could use four locality sets: alphabets, digits, successive blanks, and the special characters.

Assume that the character set is divided into n locality sets (not necessarily disjoint). In the locality tree of the ith group, there will be a switch indicator leaf node, $ (i,j), to indicate a switch from group i to group j, $i \neq j$, and $1 \leq i,j \leq n$. The relative frequency of $(i,j) within the ith tree is obtained by collecting statistics about the average sequential-run length and transition frequency from group i to group j.

As an example, suppose that we would like to divide the character set of Figure 3 into 3 groups such that the blank character is in a separate group. Figure 5 shows the three locality trees: the alphabet, the digit, and the blank tree, using appropriate relative frequencies for the switch indicator nodes.

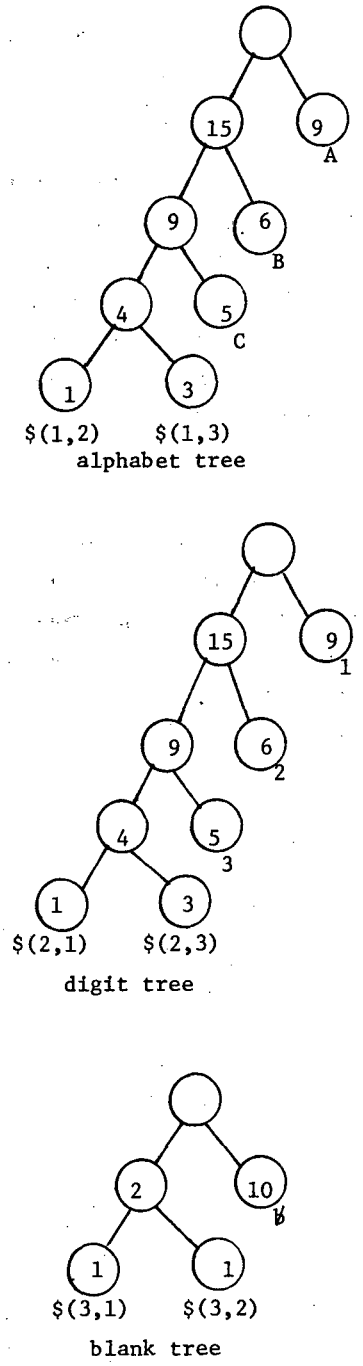


Figure 5. Three locality trees

Note that the saving obtained from shorter representation of characters and the clustering of characters of the same type in adjacent positions of the input text outweighs the extra space due

to the less frequent occurring switch indicators.

5. USE OF LOOK-AHEAD

The idea of using bigrams (or m-grams) can be easily incorporated into the previous technique if a look-ahead capability is added to the encoding process. For example, if in the alphabet tree of Figure 2, we know that the bigram 'BC' occurs frequently, e.g., 50% of the occurrences of the character B are followed by the character C. Then we can add a leaf node for the bigram BC as shown in Figure 6. Whenever the character B is encountered in the text, the next character is also examined to see if the code of the bigram BC can be used to encode the two consecutive input characters. It is easy to see that the use of the scheme of Figure 6, rather than that of Figure 2, will save an average of 0.5 bit for each occurrence of the character B.

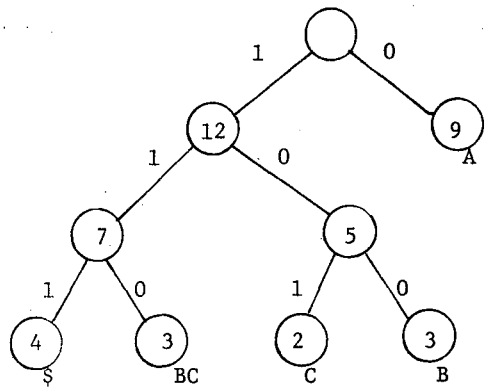


Figure 6. An alphabet tree with a bigram leaf

Another application of look-ahead is when a separate locality set is used for the blank character, yet the blank character is also included in the alphabet tree in order to handle the occurrence of a single blank between two alphabet words. Whenever a blank character is encountered after an alphabet, the next few characters are examined. If this blank is a member of a succession of blanks, control is transferred to the blank tree using the appropriate switch indicator. Otherwise, the local code of the blank character in the alphabet tree is used.

6. THE COMPRESSION SUBSYSTEM

A typical compression subsystem, based on our proposed scheme, would consist of three principal components: the statistics gathering function, the data compression function, and the data expansion (decoding) function.

The Statistical Gathering Function: A statistically significant portion of the file is processed and pertinent statistical information is collected (e.g., frequencies of characters, transition frequencies among locality groups, frequency of bigrams, etc.)

The Data Compression Function: This function builds the decode trees using the statistics obtained by the statistical gathering function.

The Data Expansion Function: This function restores the original text from the compressed data. The following is the decoding algorithm used by this function. We assume that the pointer R is initialized to point to the root of one of the locality trees.

Decoding Algorithm:

```
P ← R
/* R points to the root of current tree*/
/* P points to current node */

LOOP
   IF node P is not a leaf THEN
      CASE
      :input=o ; P ← rightchild(P) ;
      :input=1 : P ← leftchild(P) ;
      ENDCASE;
   ELSE   /* P is a leaf node */
       IF node P is a switch indicator
       THEN  R=P ← DATA(P)
             /*New root */

       ELSEDO;
          Print output character
          or m-gram of node P;
          P ← R
          /* Start from the root */
       ENDIF;
   ENDIF;
UNTIL end of compressed data;
```

7. NUMERICAL TESTS

Two separate implementations (one on an IBM 370/3033 machine and the other on a VAX 11/780 machine) have been used to compare our proposed scheme and the Huffman's method. The extensive tests performed so far have shown a consistent superiority over the Huffman's method and an improvement over the results reported in [2]. The average improvement in compression over Huffman's method is 24%. The processing time during the decoding phase for the proposed scheme was 9% smaller (on the average) than that of the Huffman's method (due to shorter search path, i.e., less number of bits in the compressed text). The files under consideration were files containing both numeric fields (thus they are amenable to statistical analysis) and alphabetic fields (e.g., names, addresses, etc.). These files are common in business, academic, as well as some research environments. The implementation of m-grams has been limited to few (statically defined) bigrams. A dynamic (more sophisticated) gathering statistics component, i.e., one in which the decision about the number of locality groups and their character members will be based on the statistics collected, is being planned.

REFERENCES

[1] Cooper, D. and Lynch, M. "Text compression using variable to fixed length encodings" JASIS, Vol. 33, No. 1, January 1982.

[2] Hazboun, K. and Bassiouni, M. "A multi-group technique for data compression" Proceedings of the ACM SIGMOD 1982 Int. Conf. on Management of Data, Orlando, Fl, June 1982, pp. 284-292.

[3] Huffman, D. "A method for the construction of minimum redundancy codes", Proc. IRE, Vol. 40, 1952, pp. 1098-1101.

[4] Mommers, J. and Raviv, J. "Coding for data compaction" IBM Research Report RC 5150, T. J. Watson Res. Center, Yorktown Heights, NY, Nov. 1974.

[5] Mulford, J. and Ridell, R. "Data compression techniques for economic processing of large commercial files" Proc. ACM Symp. on Information Storage and Retrieval, 1971, pp. 207-215.

[6] Pechura, M. "File archival techniques using data compression" CACM, Vol. 25, No. 9, 1982, pp. 605-609.

[7] Rubin, F. "Experiments in text file compression" CACM, Vol. 19, No. 11, Nov. 1976, pp. 617-623.

[8] Ruth, S. and Kreutzer, P. "Data compression for large business files" Datamation, Sept. 1972, pp. 62-66.

[9] Schwartz and Kleiboemer, A. "A language element for compression coding" Information and Control, Vol. 10, 1967, pp. 315-333.

8. Security and Integrity Issues

See Also. . . .

AUTOMATED CELL SUPPRESSION TO PRESERVE CONFIDENTIALITY OF BUSINESS STATISTICS*

G. Sande, Structural Analysis Division, Statistics Canada
Ottawa, Ontario, K1A 0T6 Canada

Abstract

A Statistical Agency must balance the competing requirements of preserving the confidentiality of the data obtained from respondents and of publishing statistical summaries of the data obtained from respondents. This note will describe the components of an experimental suite of software which seeks to resolve this conflict in the case of economic censuses. The components described will be those which identify sensitive statistics, determine complementary suppressions and audit the suppressed publications. These components are supported by an infrastructure of utility programmes. Experience with the software on various economic censuses and opportunities for further work will be discussed.

Introduction

A Statistical Agency must balance the competing requirements of preserving the confidentiality of the data obtained from respondents and of publishing statistical summaries of the data obtained from respondents<2>. This note will describe the components of an experimental suite of software which seeks to resolve this conflict in the case of economic censuses. The components described will be those which identify sensitive statistics, determine complementary suppressions and audit the suppressed publications. These components are supported by an infrastructure of utility programmes. Experience with the software on various economic censuses and opportunities for further work will be discussed.

Basic Data

The foundation for all of the activities is the data supplied by the respondents. For business respondents we assume that the existence of the economic activity is known and that any interested observer can make a reasonable guess as to the amount of the economic activity. In precise terms, this means that the geographic location (SGC or Standard Geographical Code), the type of business (SIC or Standard Industrial Classification) and the owners identity (IDN or Identifier Number) are all known with certainty. The value of the economic activity may be reasonably guessed and we will assume that an approximation of between 50 and 150 percent of the true value may be readily obtained. The details of whether the approximation should be 50 to 150 percent or 60 to 140 percent do not much matter as only various ratios are important. The 50 to 150 percent assumption yields various coefficients of 1/2 which will arise later.

The most striking feature of this data is the great diversity of sizes of the economic units. There are many units such as "Sam's Corner Store" with activity of $10,000 and a few units such as "Multinational Manufacturing, Inc." with activity of $500,000,000. A published total of these two would for all practical purposes be the value of the larger unit. An aggregation would require many more units before it is no longer just a minor perturbation of this large unit.

*presented to Conference of European Statisticians, Working Party on Electronic Data Processing, March 21-25, 1983, Geneva

Identifying Sensitive Statistics

The data supplied by the respondents is used to determine many related statistics. Examples would include total shipments of furniture manufactures in Saskatchewan, all manufactures in Saskatchewan, furniture manufactures in Canada or all manufactures in Canada. These illustrate various levels of aggregation (Saskatchewan and Canada or furniture manufacturing and all manufacturing) in both the geographic and industrial classifications. Standard rolling up schemes would allow the calculation of the various totals to be carried out.

The problem is more involved as we would like to identify the large units in each of the totals we form. If we seek to aggregate

10	50	60	120
Tom	Sue	Dick	Total

with

5	45	75	125
Harry	Sue	Joe	Total

we get

5	10	60	75	95	245
Harry	Tom	Dick	Joe	Sue	Total

The six separate units become five under aggregation as "Sue" is common and the size ordering under aggregation is quite unstable. The size ordered list of unique respondents is used to determine which totals are sensitive. The respondents contributions to a particular total are identified directly, unduplicated for ownership and then sorted to produce the desired list with no attempt made to roll up aggregations because of the difficulty illustrated above. Determining the respondents for a particular total is an example of a "range query" and may be processed efficiently with an appropriate data structure such as a k-d tree<1>.

A typical rule to identify sensitive totals is if the three largest respondents contribute more that 75 percent of the total then the total is sensitive. Dick, Joe and Sue, above, with combined value of 230 contribute more than 75% of the total of 245 so the example aggregation is sensitive. We would write this as

$$x(1) + x(2) + x(3) > 3/4 \{ x(1) + x(2) + x(3) + x(4+) \}$$

where x(1) is the largest unit, and x(4+) is the sum of the fourth and smaller units.

$$s'(x) = 1/4 \{ x(1) + x(2) + x(3) \} - 3/4 \; x(4+) > 0$$

is the same formulae where s'(x) > 0 has become the rule for identifying sensitive totals. We prefer to have the coefficient of x(4+) be -1 so we have the equivalent rule

$$s(x) = 1/3 \{ x(1) + x(2) + x(3) \} - x(4+) > 0$$

We may demonstrate that

$$s(x+y) \leq s(x) + s(y) \qquad (1)$$

$$s(x+y) \geq s(x) - t(y) \qquad (2)$$

where x+y is the aggregation of two separate totals and t(.) is the total displayed in the same notation<3>. Equation (1) indicates that s(.), which we will call the sensitivity criterion, is subadditive and expresses precisely the intuitive notion that aggregation provides protection of the confidentiality of the respondents' data. Equation (2) shows that small totals cannot provide protection for very sensitive totals. The -1 coefficient of t(.) in (2) is a result of our choice of -1 as the coefficient for x(4+) in s(.). We are more interested in the upper tolerance

$$u(x) = t(x) + 1/2\ s(x) \qquad (3)$$

for the identification of the complementary suppressions. The 1/2 is a reflection of the 50 to 150 percent approximation assumption made earlier.

In practice the sensitivity criterion in use will have other percentage thresholds and numbers of units although the algebra will be formally the same. The details of the sensitivity criterion are often held to be as sensitive as the data it is used to protect.

Determining Complementary Suppressions

The identification of sensitive totals gives us a cleanly stated problem which can now be solved. We do this by changing a sensitive total, temporarily, to its upper tolerance (3) and seeking to rebalance the table.

Suppose

2	2	4
2	10	12
4	12	16

is a table with the upper tolerance of 11 for the sensitive cell with value 10. We would get

2	2	4
2	10+1	12
4	12	16

which doesn't add up but the table

2+1	2-1	4
2-1	10+1	12
4	12	16

does add up. Mathematical programming is designed to maintain the equations which indicate that the total is the sum of its parts. From this modified table we conclude that the three additional totals should be suppressed as complementary suppressions. The amount of change permitted in the totals of value 2 would be from 1 to 3 by our 50 to 150 percent assumption.

When the examples are larger there are many alternate patterns which will do the job so we must specify how to choose between the many alternatives. We rate each pattern by summing the

product of the size of the total being changed and the amount of change. This yields a linear objective function for the mathematical programming problem. This "size rule" tends to favour the suppression of small totals in order to preserve large totals. The mathematical programming obtains the best pattern, under the objective function, without having to examine all possible patterns.

An objective function which summed the size of the totals being suppressed, without regard to the amount of change, would yield a discrete optimization problem well known to be much more difficult to solve. The solution algorithm in use applies the temporary modification to the most sensitive total first, and then to the second most sensitive total not protected with the previous complements used with no cost in the objective function. This is repeated until all sensitive totals have been protected. The result is a simple sequential heuristic which works surprisingly well and which can be enhanced with minor manual intervention.

The resulting operation may be described as the completion of the pattern of complementary suppressions. The smallest initial pattern of suppressions would be the sensitive totals with no presupplied complementary suppressions and leads to a completely automatic determination of all required complementary suppressions. Many presupplied complementary suppression may result in the sensitive cells being already protected with no new complementary suppressions would be required. An example of this would be the use of the January suppression pattern for the February data of a monthly business survey.

Auditing Suppressed Publications

The results of the automated calculation of a suppression pattern are, of course, correct. To increase our confidence in the correctness or to examine a pattern prepared elsewhere we would like to audit the pattern for correctness. The aggregation structure of the table can be represented as a system of equations which can be used as constraints in a mathematical programming problem.

The table

x	x	4
x	x	12
4	12	16

yields the equivalent table

0-4	0-4	4
0-4	8-12	12
4	12	16

The ranges for the individual totals cannot be usefully added in general as illustrated by the first row has a total of 4 but has the ranges adding to the range 0-8.

The ranges provide no information that

is not already present in the table and could well be constructed by any user of the table. Providing this information may render the table more useful to the users who do not want to become well versed in the techniques of using suppressed tables.

The example

x	x	2	2	16
x	x	x	2	8
2	2	x	x	8
2	2	x	x	16
16	8	8	16	48

is equivalent to

8-12	0-4	2	2	16
0-4	0-4	2-2	2	8
2	2	0-4	0-4	8
2	2	0-4	8-12	16
16	8	8	16	48

The range 2-2 is an example of what we want to avoid. This example illustrates that the rule of two suppressions in every row and columne is not enough to prevent residual disclosures. Such tables are not produced by the automated techniques discussed above as they are based on a much stronger analysis of the problem.

Support Utilities

The three substantial components discussed above are supported by a collection of ten utility programmes. The rest of this section is a brief description of this infrastructure and illustrates the additional support required to extend the substantial components to a functional suite of software. All of the programmes use a self describing free format file for both input and output so they may be run in any order which makes sense in terms of the operations performed. Occasionally, some very unforeseen execution sequences have made sense.

The tabulation program which identifies sensitive totals, called Build, has a limit of 35,000 micro-records because it keeps the data in main memory. When this is a restriction, a utility program, Merge, allows separate segments of the data to be tabulated and the results combined so that the 35,000 limit becomes a nuisance rather than a severe limitation.

There are four variations on update utilities with Merge also having some update functions. The main update utility, Updat, allows the specification of publish or suppressed status for individual totals as well as modification, including deletion, of upper tolerances when waivers from respondents allow changes in the apparent sensitivity status of totals. A bulk update, Trnsf, allows transfer of a suppression pattern from one tabulation to another related tabulation such as the January to February transfer for monthly surveys. Another bulk update, Mask, allows repetitive specification of publish or suppressed status as might be done to suppress all four digit SIC detail in a very small province. Segments of a tabulation may be extracted with a utility, Sbset, to break the computation into subproblems for large tabulations.

There are three variations on reformat utilities. Most of the components deal with a tabulation having totals with a status code while others, most importantly the audit component Bound, use a tabulation with totals present or absent. The conversion utility, Reles, that does this transformation is said to release the tabulation as the ouput contains no sensitive information. The conversion of the files from the self describing free format to a fixed format record is done by Refmt, with the ouput serving as a control file for the regular publication system. Transfer of files between the two classification variable suite and the three classification variable suite is done by the utility Face with the output typically serving as a bulk update for the transfer utility.

Externally prepared tabulations often have empty totals omitted. The pattern is completed, by Cmplt, so that empty totals are explicitly represented with a value of zero. The resulting table may not exactly add and must be adjusted, by Adjst, to correct for independent rounding or other errors before it may be audited. This adjustment is done by mathematical programming which is internally similar to the Suprs component which calculates the complementary suppressions. The adjustment facility, which hardly deserves to be classified as a support utility, is of use in its own right to adjust independent rounding and works even when there are suppressed totals.

Useage Experience

The major benefit of the software is the discipline imposed in proceeding from various notions expressed in terms of convex spaces<5> to functioning software. The same discipline also applies to the preparation of tables from economic census takers for use with the software.

An early problem was the geography where an economic region contains several counties which contain several municipalities as well as the economic region containing a metropolitan area containing several municipalities. The counties and metropolitan areas are equally valid disaggregations of the economic region and are built up from the same municipalities but in different groupings. The two classification variable suite addresses this problem with little external, but moderate internal, complication. In applications, it is easy to forget the metropolitan area implicitly defined by the remainder of the economic region after the regular metropolitan region is defined. (The first sentence of this paragraph illustrates how natural this mistake is.) Similar problems occasionally arise in the industry codes with non-standard aggregations. Included subtotals of some tables must be explicitly represented in the classification hierarchy.

Live problems are much larger than the illustrations that are included in expository notes. A typical economic

census will have a geographic structure of 12 provinces or territories and 4 regional or national groups for a total of 16 geographic codes. The industrial classification structure may have 200 industrial codes which are 4 digit codes (industries), 3 digit codes (industry groups), 2 digits codes (major industry groups) or the grand total. These two classifications yield 3000 totals organized into 200 tables at varying levels of aggregation. The tabulation of 30,000 manufacturing records to identify which of the 3000 totals are sensitive takes around 2 minutes. The calculation of complementary suppressions takes about 20 minutes and the auditing takes about 3 minutes. The operation is as smooth as anything involving 200 pages of output can be reasonably expected to be. For 60,000 employment records, the tabulation is done in two segments, followed by a merge, with the similar overall timings. For 360,000 financial records, the breaking of the tabulation into 15 segments becomes annoying with the other timings as before. For 20,000 food service records classified by three variables for 1000 totals, the overall timings totalled 10 minutes and was successfully completed on its first attempt. The census of agriculture was a problem in controlling the operation as 25 different agricultural attributes; mostly crops, were processed for each of the 10 provinces in several weeks.

Very large problems which must be broken into pieces are not smooth operations. The manufacturing records classified additionally by country of control yield about 12,000 totals or by destination of shipments yield more than 50,000 totals. The labour records classified down to the city and county level yield more than 50,000 totals. These very large problems have a high proportion of zero totals. A typical scheme is to determine the suppression pattern for the labour records classified to the province only and then to disaggregate each province in turn.

Future Work

The purpose of the suite of software was to provide a testbed for research into automated cell suppression techniques. It has demonstrated working techniques and the range of support required of a full collection of software. One train of development would be to reimplement the facility but with a stronger production orientation. For example, the existing software prints tables with labels of numeric codes only while a production oriented system would use descriptive text stubs. Various changes could be made in the existing software to reduce operational annoyances. For example, the status flagging mechanism should be able to indicate that a subtotal introduced for convenience purposes and not intended for inclusion in any publication may have its value determined exactly without drawing residual disclosure error messages as it does in the current system. A new implementation of Build to process

more that 35,000 records without the help of Merge would relieve much annoyance.

The real opportunities for future work are in enhancements in understanding, facility and capability. Use of degeneracy exploiting linear programming inplace of the existing steepest descent linear programming may make smooth operation possible for larger problems. Perhaps the 3000 totals boundary may become 4500 with this internal tuning. A better segmentation scheme may make the very large problems more like smooth operations. An experimental version of the complementary suppression calculator which uses ranges, such as might be used to represent error estimates, on all totals has been tested with encouraging results although elaboration on how to publish its results are required. A good understanding to the relevant matrix theory is only available for single two variable tables<4>. A corresponding understanding for hierarchies of tables and for three variable tables would provide insight into the pragmatic success of the methods and may lead to improved algorithms. Some interesting but isolated matrix theory facts have been discovered by exhaustive machine aided searching.

Conclusion

A suite of software has been implemented to carry out the automated cell suppressed required to protect the confidentiality of respondent data in economic censuses. The original notions arose in the context of convex spaces. The result is a functioning suite of software which has been used by several users, some of whom stretched the limits beyond the development intentions. The very brief description here only illustrates the major notion of the main components. For the foreign control breakout of the census of manufactures, the user reports an increase of timeliness of ten months, in a biennial program, concurrent with a five-fold increase in the number of totals processed. Considerable opportunity exists for extending understanding and capability in what is already a successful development.

References

1. J.L. Bentley and J.H. Friedman, Data Structures for Range Searching, pp 397-409, Vol. 11, no. 4, Computing Surveys, 1979

2. L.H. Cox and G. Sande, Techniques for Preserving Statistical Confidentiality, Proceedings of the 42nd Session of the International Statistical Institute, (in press), Manilla, 1979

3. G. Sande, Towards Automated Disclosure Analysis for Establishment Based Statistics, Statistics Canada, 1976

4. G. Sande, A Theorem Concerning Elementary Aggregations in Simple Tables, Statistics Canada, 1978

5. G. Sande, Confidentiality and Polyhedra - An Analysis of Suppressed Entries in Cross-Tabulations, presented American Statistical Assoc., San Diego, 1978

An Information Theoretic Approach to Statistical Databases and their Security: A Preliminary Report

Mary McLeish
University of Guelph

Abstract

This paper considers a statistical database model used by J. Kam and J. Ullman in [5] to study database security. Results for the transmitter-receiver problem studied in information theory are applied to the database model to provide a measure of the information in the query-record system. A slightly different information theoretic approach is then used to provide a more effective model for the security problem. Results are obtained which give the conditions on queries necessary to minimize the information gained by making a query. Minimizing this information function corresponds to increasing the chance of a security breach. Statistical methods can then be used to determine if a series of queries are being used which have properties significantly close to those required to endanger the security.

INTRODUCTION

A statistical database is a collection of records about which primarily summary data, such as means, medians, standard deviations etc., is required. Information about a particular individual is to be protected and the security problem consists of preventing such information from being deduced from collections of summary data. Several researchers, [2,5,6] have considered this problem under a variety of circumstances. In this paper, the model used by J. Kam and J. Ullman [5] is further investigated and information theory is used to provide guidelines for restricting queries to reduce the likelihood of breaking the security.

1.1 The Database model:

In their model, a statistical database is a function f from strings of k bits to the positive and negative integers. The keys become the domain of f. The range of f is usually considered to be finite. An (s,k) query is a sequence of length k consisting of 0's, 1's and *'s with exactly s 0's and 1's. (If k is known, this is simply called an s-query.) The symbol * stands for "don't care", in the sense that it matches either 0 or 1. Thus, the query *010* matches the keys 00100, 00101, 10100, 10101. An (s,k) query matches 2^{k-s} keys, forming a cube of dimension $k-s$ in the Boolean k-cube. The result of a query q on a statistical database f is then $\sum_{q \text{ matches key } i} f(i)$. As an example, consider a database of hotel workers salaries in a city. The key could consist of 10 bits $xxxxxyyyzz$ as follows:

i) $xxxx$ is a code for the hotel

ii) yy is a code for the type of employee, e.g. 100 is the manager, 000 is a bell-hop, 001 is the ball tender etc.

iii) zz is a code representing which chain the hotel is part of, and is 00 if the hotel is part of no chain.

Then 1000100001 would represent the sum of all bell-hops salaries at hotel 10001 of chain 01 and *****10001 would be the sum of all the managers' salaries in chain 01.

If one can deduce the value of $f(i)$ for some i by knowing the result of a set of s-queries the database is said to be compromisable by s-queries.

Using this model, several security results are obtained in [5]. Finite and infinite ranges are considered separately. Further results involving sparse database (ones in which many keys might have no records associated with them) have been proven by Chin in [2].

1.2 Some Basic Definitions in Information Theory:

In this study, an attempt is made to measure the information received when queries are made.

Let X be a random variable with sample space Ω. Suppose Ω is partitioned into a finite number of mutually exclusive events E_k with associated probabilites p_k.
Then $I(E_k) = -\log p_k$ (usually to base 2) is called the amount of self-information associated with the event E_k. A bitof information $=-\log\frac{1}{2}$ is the amount of information associated with the selection of one of two equiprobable events. The average amount of information or entropy associated with the random variable X and the events E_k is $H(X) = \overline{I(E_k)} = -\sum_{k=1}^{n} p_k \log p_k$. This then represents the expected value of the uncertainty associated with the probability scheme. The function H considered as a function of the p_k has certain well-known properties, such as continuity, additivity, symmetry etc., which will be recalled if needed.

This measure of information can be extended to a two-dimensional scheme with two finite discrete sample spaces. If $\{x_i\}_{i=1}^{n}$ represents the possible values of the random variable X and $\{y_i\}_{i=1}^{m}$ those of another random variable Y, then the five quantities of interest are:

(i)

$$H(X|Y) = -\sum_{j=1}^{m}\sum_{k=1}^{n} p\{y_j\}p\{x_k|y_j\}\log p\{x_k|y_j\},$$

where $p(x_k|y_j)$ is the conditional probability of x_k given y_j and $p(y_j)$ (or(x_k)) represent marginal probabilities.

(ii) $$H(X) = -\sum_{k=1}^{n} p\{x_k\}\log p\{x_k\},$$

(iii)

$$H(X,Y) = -\sum_{k=1}^{k=n}\sum_{j=1}^{k=m} p\{x_k,y_j\}\log p\{x_k,y_j\},$$

where $p(x_k,y_j)$ is the joint probability of the pair (x_k,y_j),

(iv) $H(Y)$, defined similarly to $H(X)$ and finally,

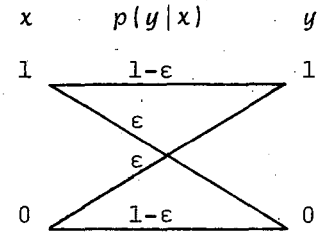
(v) $H(Y|X)$, as for $H(X|Y)$ with x_k and y_j and the order of summation interchanged.

Another useful concept is that of mutual information, $I(x_i,y_j)$ or the information about the event x_i by the occurrence of the event y_j. This is defined to be $\log \frac{p\{x_i,y_j\}}{p\{x_i\}}$. It can be shown that $I(x_i,y_j) = I(y_j,x_i)$, whence the word mutual. If $x_i = y_j$, one obtains $-\log p(x_i)$ or the earlier self-information of the event x_i. $I(X,Y)$ stands for the average mutual information and is the expect -ed value of $I(x_i,y_j)$. It can be shown that $I(X,Y) = H(X) - H(X|Y)$ or $H(X)+H(Y)-H(X,Y)$ or $H(Y)-H(Y|X)$. When the system under consideration is that of a source and receiver connected by a channel, with $\{x_1,\ldots x_N\}$ and $\{y_1,\ldots y_M\}$ the sets of possible inputs and outputs to the channel, $C = \max I(X,Y)$ is called the Channel capacity. This is the maximum taken over all possible sets of input probabilities and it can be shown to exist. (c.f. 1,3,6 for further details.)

§2 Measuring Information for a Statistical Database System

Consider again the concept of a transmitter and a receiver, both with given finite alphabets. Let our individual records be strings of independent binary bits which are receiving queries. If we look at one bit position at a time and assume the probability of a * in that position is p and that the 0's and 1's are equally probable, then the probability of a match is $\frac{1+p}{2}$.

We could then assume a model like that for a binary symmetric channel of the form: (c.f. 3)



The value of ε will be $1-(\frac{1+p}{2})$ or $\frac{1-p}{2}$.

Here we have the query on the left, represented by x, but the possibility of obtaining a * is incorporated into the conditional probability of a match. On the right is the record, y. Here $P_x(1) = P_x(0) = P_y(1) = P_y(0)$ are all taken to be ½. But $P(1|1)$ is now $\frac{1+p}{2}$ or $1-\varepsilon$ and similarly for $(0|0)$ and $(0|1)$. This is assuming that in fact the pair (1,1) can be obtained either by having (1,1), whose probability is $(\frac{1-p}{2} \times \frac{1}{2})$ or by having (*,1), whose probability is $\frac{p}{2}$.

Thus $p(1,1)$ is $\frac{1+p}{4}$ and

$$P(1|1) = \frac{P(1,1)}{P_x(1)} = \frac{1+p}{2}.$$

In this scheme, the conditional entropy $H(Y|X) = -((1-\varepsilon)\log(1-\varepsilon) + \varepsilon\log\varepsilon)$.

$I(X,Y) = H(Y) - H(Y|X) = H(Y) + (1-\varepsilon)\log(1-\varepsilon) + \varepsilon\log\varepsilon)$.

But $H(Y) = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2} = 1$. Thus the mutual information is then $1+(1-\varepsilon)\log(1-\varepsilon)+\varepsilon\log\varepsilon$, which is also in

fact the channel capacity. The derivative of H or I with respect to ε is 0 when $\log \frac{\varepsilon}{1-\varepsilon} = 0$, giving $\varepsilon=\frac{1}{2}$ or $p=0$. This would correspond to a completely noisy channel. In other words, input and output are statistically independent. In our situation, there are no *'s and the value of $P(x,y)$ is always $\frac{1}{2}$. The mutual information is 0 in this case. At the other extreme, when $p=1$ or $\varepsilon=0$, the channel is noiseless or the query specifies the record with certainty and the mutual information is 1.

With this model, the extreme values occur when p is either 0 or 1. We actually wish, for security purposes, to increase the probability of a match. This means the sum of the $f(i)$ over the matched records includes more terms and there is a smaller chance of determining a single $f(i)$, with this or subsequent queries. This corresponds to keeping the value of the mutual information high. A series of experiments would help to determine an optimal value of the mutual information or conditional entropy function for a specific application. An investigation of the effect of subsequent, not necessarily independent, queries on the total entropy in order to bound it appropriately, is being made. In section 2.2, a slightly different model will be presented from which a cut-off point can be found directly. However, if the security problem is not of primary concern, model 2.1 does provide a convenient measure of the information contained in a query-record system, when extended to sequences of bits. This extension is purely additive for a memoryless system (no dependence between successive bits) and the extremal results possess the same characteristics.

§3 A Revised Model for the Security Problem

3.1 Definition of the Mutual Information Function:

Consider our system as consisting of one query at a time being made against a set of n records. Let the probability $p(x)$ of a match be simply $\frac{1}{n}$; i.e. all the records are equiprobable. Then the entropy function or the average self-information of the system

$= E\{-\log p(x)\}$ (the expected value)

$= -\sum_x p(x) \log p(x)$

$= \log n.$

Now each query partitions the record space into two parts: S, the set of records matching the query and S^c, the complement of S. In actual fact, usually the total sum of the $f(i)$ for all records is known or can be easily found by querying with all *'s.

Thus, when a query is made, not only is information gained about the records in S, but also about the set S^c, by subtracting from the total sum for all the records. Suppose the number of records in S is n_1, then $\log n_1$ and $\log(n-n_1)$ represent the self-information possessed by a record lying in S and S^c respectively. The security is more easily violated the smaller the amount of information returned by a query. Therefore, the appropriate quantity to consider here is $\min\{\log n_1, \log(n-n_1)\}$. Then the information gained by making a query becomes $\log n - \min(\log n_1, \log n - n_1)$ or $\log\left(\frac{n}{\min(n_1, n-n_1)}\right)$. (If Y represents the query and X an individual record, this quantity is really $I(X,Y)=H(X)-H(X|Y)$ or the mutual information function.) This function will be bounded at the extremes by 0 or $\log n$. Thus, if a query with no *'s matches no record, $I(X,Y)$ will be simply 0 and if $n_1=n$ the function will be equal to $\log n$. That is, the mutual information will not be allowed to exceed the original self-information of X.

Formally, we have the following definition:

Definition 3.1.1.

Given a statistical database as described in Section 1.1, containing n records and upon which a single query is made, the information gained by making the query is defined to be:

$$\begin{cases} \log\left\{\frac{n}{\min(n_1, n-n_1)}\right\} & \text{Where } n_1 = \text{the number of records matched by the query, provided } n_1 \neq n \text{ or } 0; \\ \log n, & \text{if } n_1 = n; \ 0 \text{ otherwise.} \end{cases}$$

3.2 Determination of the Expected Mutual Information for a Random Query

Assume that the variable n_1 is uniformly distributed on the set $\{0, 1, \ldots n\}$ (c.f. 4). If n is odd $= 2m+1$ for some integer m,

then $\log \frac{n}{\min(n_1, n-n_1)}$ has expected value $\log n - \frac{2}{n+1} \sum_{j=1}^{m} \log j - \frac{1}{n+1} \log n$

$= \log\{\frac{n}{(nm!^2)^{1/n+1}}\}$

In the even case, if $n=2m$, one obtains:

$\log n - \sum_{i=1}^{m-1} \frac{2}{n+1} \log i - \frac{1}{n+1} \log n - \frac{1}{n+1} \log m$

$= \log\{\frac{n}{(nm!(m-1)!)^{1/n+1}}\}$.

In both cases, the bounds $\log (\frac{n}{m+1}) < E(\log \frac{n}{\min(n_1, n-n_1)}) \leq \log n$ may be established. One wishes to investigate situations in which the values produced by queries are regularly falling below the expected value or its lower bound. This could correspond to a user attempting to break the security. Whether the regularity is statistically significant can be tested to further determine the likelihood of this being tried.

Now consider the above expected value when we are specifically in the situation of queries being k-bit strings of 0's,1's or *'s. Let us also assume that the distribution of the number of *'s in a query is binomial. Let p be the probability, $p(*)$, of a * occurring in any given bit position. That is, $p(n_1=2^j) =$

$(\frac{k}{j})\ p^j\ (1-p)^{k-j}$, $j=0,1,2,\ldots k$ is the probability of j stars, which we will call $p(j)$. Also suppose the full record set is available. Therefore $n=2^k$. Then we may prove the following theorem:

Theorem 3.2.1

The expected value of the information gained by making a query of length k to a statistical database of 2^k records is minimized when $p = (\frac{1}{k})^{1/k-1}$, when $k>1$.

Proof: Now the possible values of $\min(n, n-n_1)$ are all the powers of 2 up to and including 2^{k-1}, with respective probabilities $p(j)$. Thus the expected value sought is

$p(0) \log (\frac{n}{2^0}) + p(1) \log (\frac{n}{2^1}) + \ldots\ldots$

$p(k-1) \log \frac{n}{2^{k-1}} + p(k) \log n.$

This equals $\log 2 \{\sum_{i=0}^{k} p(i)(k-i) + k\ p(k)\}$,

$= E(k-X) + k\ p(k)$, where X is a binomial random variable

$= k(1-p) + k\ p^k$.

Now the derivative of this last expression with respect to p is $k + k^2\ p^{k-1}$, which equals 0 when $p=(\frac{1}{k})^{\frac{1}{k-1}}$.

The second derivative is $k^2(k-1)p^{k-2}$ which is greater than 0, as long as the strings are at least of length 2. (If $k=1$, the expected value is constant with value 1 bit, regardless of p.) □

The minimum expected value becomes $k-(k-1)(\frac{1}{k})^{\frac{1}{k-1}}$. This will be less than k except when $k=1$. When $p=0$ or 1, we have the maximum value of k bits.

Again, one would wish to observe the empirical value of p from a sequence of queries and determine if it was significantly close to the minimization value given here. The information function given by Definition 3.1.1 could be evaluated for each individual query and its deviation from the minimum value just given, studied.

§4 Concluding Remarks

Some of the areas requiring further investigation are:

(i) Studying distributions other than the uniform distribution for the model of section 3.

(ii) Constructing a test for the independence of successive queries.

(iii) Measuring the information gain after a sequence of possibly dependent queries.

(iv) Finding the distribution of the number of queries required to be made until a fixed amount of information is obtained. Also, determining the probability of obtaining more than a certain amount of information in a fixed number of queries.

(v) Other moments of the information function could give further knowledge of the behaviour of the information function.

(vi) Finally, simulation experiments should be run to help substantiate the ideas given in the theory.

REFERENCES:

1. Ash, R.B., "Information Theory", Interscience Publishers, (1967)
2. Chin, F.Y., "Security in Statistical Batabases for Queries with Small Counts", ACM Transactions on Database Systems, Vol.3, No.1, (1978), pp. 92-104.
3. Gallager, R.G., "Information Theory and Reliable Communication", John Wiley and Sons, (1968).
4. Hodges, J.L. and Lehmann, E.L., "Basic Concepts of Probability and Statistics", Holden Day (1970).
5. Kam, John B. and Ullman, J.D., "A Model of Statistical Databases and Their Security", ACM TODS, Vol.2, No.1, (1977), pp.1-10.
6. Reza, F.M., "An Introduction to Information Theory", McGraw-Hill, (1961).

An Introduction To Sampling To Estimate Database Integrity

Rick Greer
Bell Laboratories
Murray Hill, New Jersey

ABSTRACT

In order to create a sound and meaningful sampling plan for estimating measures of database integrity, a statistician must first have appropriate theoretical tools

(i) for understanding the nature of databases and their integrity constraints,

(ii) for developing reasonable numerical measures of database integrity, and

(iii) for determining the sampling unit and its relationship to the integrity measures.

This paper uses database theory, mathematical logic, and sampling theory to provide such tools.

1. Introduction

The practicing statistician can use the theory presented in this paper to define measures of database integrity and to develop sampling plans to estimate them.

Section 2 presents a way to understand databases that fosters the selection of both appropriate integrity measures and appropriate sampling units.

Section 3 defines integrity constraints to be assertions describing some database that one would like to have be true. Symbolic logic can be used to partially formalize integrity constraints so that their new structure easily suggests meaningful numerical measures for measuring their proximity to truth.

These measures form a family of database integrity measures called the V criterion family which is the subject of Section 4. The first step in creating a V criterion is to use the structure of logic-formalized integrity constraints to "decompose" them into collections of simpler assertions called test assertions; this is done in such a way that any original integrity constraint is true if and only if all of its test assertions are true. Then, for atomic test assertions, one possible V criterion is a positive weighted average of the 0-1 numerical truth values of the given test assertions. In its simplest form, a V criterion is that fraction of a group of atomic test assertions which are true.

The last section of this paper discusses the application of sampling theory to the estimation of V criteria. Part of this discussion is concerned with delineating the relationship between the sampling unit and the chosen integrity measure. The role of ratio estimation and cluster sampling in estimating V criteria is briefly mentioned.

While most of this paper lies squarely in the realm of computer science, the author believes that it nonetheless comprises essential knowledge for the statistician practicing in this area. It is the responsibility of the statistician to know enough about an application in order to ensure that the statistical methods he or she chooses will yield meaningful results. In particular, in the case of database integrity, it is the responsibility of the statistician to make sure that appropriate database integrity measures are selected since it is pointless to develop a sampling plan for estimating meaningless parameters.

Although this paper is introductory in character, it does assume some familiarity with mathematical logic for Sections 3 and 4 (see Mendelson [3]) as well as some familiarity with sampling for Section 5 (see Cochran [1]). Some familiarity with database theory is assumed; see Date [2] for more about this topic. Due to space limitations, points are sometimes sketched and other relevant points are not discussed at all.

2. The Nature Of Databases

A `database` can be thought of as having three components: assertions, symbols, and implementations. The `assertions` of a database are the meanings of its principal symbols; they are the information content of the database. The `symbols` are merely abstractions that represent or stand for the assertions whereas the `implementations` of the symbols are those constructs which model the way the symbols are stored in some machine (or in some other medium).

In the way of examples, a commonly used database symbol is that of a `segment` which is a sequence of values for a corresponding sequence of fields. A segment's field sequence is called its `format`. Suppose segments are the principal

symbols of a database and that the format for parent segments is (MOTHER, AGE_OF_MOTHER, FATHER, AGE_OF_FATHER). Then the assertion represented by the segment (Joanna, 32, Arthur, 35) is "The mother of a family is Joanna who is 32 years old and the father of this family is Arthur who is 35 years old". A computer implementation of a parent segment might be a record; here a `record` is defined to be a sequence of groups of bytes (corresponding to the sequence of fields) where each byte consists of a pattern of eight 0's and 1's that represents some character like "J".

In contrast to the previous definition, the most common conception of a database considers it to be *only* the implementation of a collection of segments which themselves may or may not be linked together in some fashion indicating their inter-relationships. This common database conception will probably also specify the access path pointers among the segment implementations that the machine uses in order to retrieve them. In addition, it will probably confuse these pointers with the relationship links.

Note that this paper's database definition is not given in terms of implementations alone. Instead, it emphasizes that the assertions do exist and are, in fact, the raison d'être for the symbols and their implementations. Furthermore, it distinguishes between the symbols and their implementations. The usual database definition fails to recognize that symbols and implementations are merely convenient tools for working with assertions; it considers them to be fundamental in character and thereby takes them too seriously.

These distinctions among assertions, symbols, and implementations are useful ones to make and will be consistently made here. In particular, relative to database integrity, thinking solely in terms of the implementations of symbols can lead to two specific problems. The first is that it is likely that one's integrity measures will be based on the assumption that the validity of particular implementations can be determined. This is often much easier said than done. This problem does not arise for the integrity measures defined in this paper since they are defined to be direct functions of integrity constraints, not of symbols or implementations. Secondly, another good reason not to think in terms of implementations alone is that it may very well be that the best sampling unit for a good integrity measure is not some portion of an implementation.

As for terminology, the `symbol` (component of a) `database` is the database's collection of symbols and similarly for the `assertion database` and the `implementation database`.

-- an example database --

A discussion of a hypothetical family database will serve to illustrate the preceding philosophy concerning the nature of databases. Suppose it is desired to structure, represent, and store age and relationship information on a number of biological families. (A biological family consists of a mother, a father, and the children they produce.)

There are a number of symbolic systems that can be used to represent such information. Other than natural languages, these include the hierarchical, network, and relational database models. Since the main production database systems in use today are based on the hierarchical model, this model will be used to provide the symbols for the family symbol database.

The structure of each symbol in a hierarchical database model is that of a tree obtained by linking segments together in a hierarchical fashion.

Figure 1 shows the structure of the segment trees used here in the family symbol database to represent age and relationship information on biological families. The first level of one of these two level trees consists of a single segment which, as the format indicates, has seven field values, one for each of the fields SEG_TYPE, SEG_ID, MOTHER, AGE_OF_MOTHER, FATHER, AGE_OF_FATHER, and NBR_OF_CHILDREN. Figure 2 portrays this paper's family symbol database. (This symbol database does contain errors.) By definition, the root segment's value for the SEG_TYPE field is P. This root segment is linked to a list of segments, all of which have the indicated format for C type segments.

So far, only the structure of these segment trees has been discussed, not what they might mean. Their meaning is specified by stating (i) what each segment in a given tree means and (ii) what it means for a segment of one type to be linked to a list of segments of another type.

In general, each segment is taken to represent an assertion about the attribute values or characteristics of an entity. (An `attribute` is a function mapping entities to their properties or characteristics.) Each of the non-SEG_TYPE, non-SEG_ID fields in the format is taken to be an attribute of the entity being described. In the family example, the P format is used for the creation of assertions which describe the parents of a family. By virtue of giving values for the appropriate attributes (i.e., fields), each P type segment in a family symbol database represents an assertion about a family's parents: this assertion states the mother, mother's age, father, father's age, and the number of children associated with the given parents entity. SEG_TYPE and SEG_ID will be discussed in a moment. A C type segment represents an assertion about a

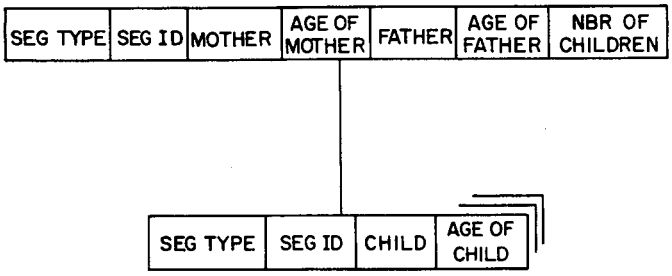


FIGURE 1: THE FIELD CONFIGURATION FOR SEGMENT TREES IN THE FAMILY SYMBOL DATABASE

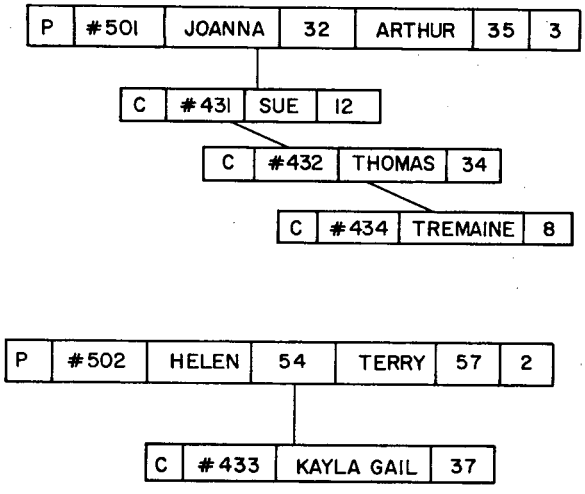


FIGURE 2: THE FAMILY SYMBOL DATABASE IN PICTORIAL FORM

child entity that states the child and its age.

In general, linking a segment of one type to a list of segments of another type is taken to mean an assertion that the entity described by the first segment is in some specified relationship to each of the entities described by the segments in the list. The nature of this relationship is required to be determinable from the SEG_TYPE and SEG_ID values occurring in the segments involved.

Relative to a tree of segments from the family symbol database, the linking of a P segment to a list of C segments represents an assertion that the parents being described by the P segment are, in fact, the parents of the children being described by the C segments in the list.

In more detail, the SEG_TYPE value for a segment has two functions. First, it tells how to decode the sequence of values in the segment by indicating that the appropriate sequence of fields is the one that corresponds to that type. Second, it helps to identify the role that the associated entity plays in the relationship indicated by any linkings.

The SEG_ID field value for a segment gives an identifier for that segment and serves to mark the involvement of that segment's entity in the indicated relationship, regardless of what other relationships that entity is involved in.

Possible implementations of the family symbolic database will not be discussed here.

3. The Nature Of Integrity Constraints

An `integrity constraint` is an assertion about a database that one would like to have be true. As such, it may refer to any of the three components of a database. This section is concerned with categorizing integrity constraints and with discussing how to use symbolic logic to formalize their expression and to reveal their structure.

Integrity constraints can be dichotomized into internal and external constraints. `Internal integrity constraints` assert the internal consistency of a database component whereas `external integrity constraints` assert that a database component is accurate in that it agrees with reality. Due to space limitations, only internal integrity constraints will be dealt with at length in this paper. As it turns out, it is possible to view external constraints as internal constraints and therefore to treat them both with the same methods.

As an example of an internal constraint for an assertion database, consider the following one for the family assertion database: "The assertions in the family assertion database are such that the stated ages of the children are less than the

stated ages of their parents."

The analog for the family symbol database of the preceding family assertion database constraint is: "For each segment tree in the family symbol database, the values for the AGE_OF_CHILD field for the C type segments in the tree are each less than the values for the AGE_OF_MOTHER and AGE_OF_FATHER fields in the corresponding P type segment." Note the difference in character: the family symbol database constraint is asserting something about database symbols, not about the assertions they represent. The family assertion database constraint, on the other hand, is an assertion about the assertions in the family assertion database: it is totally oblivious to *any* symbolic system for representing family information as well as to any physical system (paper or computer) that may be used for storing such symbols.

The implementation database analog of the preceding family assertion and symbol database age constraints makes reference to following various record pointer chains in such a way as to verify that the bit patterns of various fields are in the proper relationships. The details are omitted.

This brings up an interesting observation. Database integrity issues typically arise when people are concerned about the integrity of an implementation database. But observe that when people express their constraints in natural language, they are often found to be stating assertion or symbol database constraints. The reason is clear: it is a real nuisance to mention all of the intricate implementation details necessarily involved in the complete specification of an implementation database constraint. Consequently, it is often convenient to state instead a truth-value equivalent assertion or symbol database constraint.

As an example of an internal family implementation database integrity constraint that has no assertion or symbol database equivalent, consider: "Beginning at the first record tree in the family implementation database and (by following the P pointer chain) continuing through each succeeding record tree, the chain of C type records in each record tree is accessed in order of increasing value of the bit patterns implementing the values of the AGE_OF_CHILD field."

As an example of an external family assertion database integrity constraint, consider: "The parent-child relationships asserted by the family assertion database are consistent with what can be independently verified about the actual families."

-- integrity constraints expressed in logic --

Integrity constraints can be expressed in a number of languages. It will now be shown how a simple language based on first order logic can be used to illumine the structure of constraints, to achieve a certain precision in their expression, and ultimately, to formally suggest measures of database integrity and sampling units.

The basic building blocks of the logic language used here consist of (i) sets of entities called `domains`, (ii) variables which range over domains and whose names begin with "v_", and (iii) `predicates` which are either domains or subsets of Cartesian products of domains. The atomic declarative statements in the language are the `membership claims` which assert that domain entities or vectors of domain entities are in predicates. Other statements are generated from the membership claims by using universal and existential quantification and the usual connectives such as "not", "and", "if ... then ...", etc.

The following logic language statement AGE is equivalent to the preceding English family symbol database parent-child age integrity constraint (an explanation of the symbols follows):

```
for each value of v_P_ID, v_age_mom, v_age_dad(
    if P_SEG_ID( v_P_ID )
        and SEG:AGE_OF_MOTHER( v_P_ID, v_age_mom )
        and SEG:AGE_OF_FATHER( v_P_ID, v_age_dad )
    then for each value of v_C_ID, v_age_kid (
        if C_SEG_ID( v_C_ID )
            and LINKED_TO( v_P_ID, v_C_ID )
            and SEG:AGE_OF_CHILD( v_C_ID, v_age_kid )
        then v_age_kid < v_age_mom
            and v_age_kid < v_age_dad ))
```

Here P_SEG_ID is a domain consisting of all segment ID's for P type segments in the family symbol database. The membership claim "P_SEG_ID(v_P_ID)" in the constraint is an assertion that the value of the variable v_P_ID is in the domain P_SEG_ID. The membership claim "SEG:AGE_OF_MOTHER(v_P_ID, v_age_mom)" is an assertion that the value of v_age_mom is the value of the AGE_OF_MOTHER field for the P type segment whose SEG_ID value is the value of v_P_ID. The membership claim "LINKED_TO(v_P_ID, v_C_ID)" is an assertion that the segment whose ID is v_P_ID's value is in the same relationship chain as the one whose ID is v_C_ID's value.

There are other logically and model equivalent ways to write this constraint in logic.

By using the formula manipulation theorems of symbolic logic, one can often transform logic language integrity constraints into logically equivalent constraints that are of a particular form. This form is called `universal implication form` and is that of

for each value of v_1, ..., v_n
(if $U[\ v_1, ..., v_n\ ]$ then $T[\ v_1, ..., v_n\ ]$)

Here "$U[\ v_1, ..., v_n\ ]$" represents a logic language statement whose free variables are exactly v_1, ..., v_n and "$T[\ v_1, ..., v_n\ ]$" represents a logic language statement whose free variables are among v_1, ..., v_n.

Define the `jurisdiction J(C)` of a constraint C having this form to be the set of all n-tuples (e_1, ..., e_n) of domain entities such that $U[\ e_1/v_1, ..., e_n/v_n\ ]$ is a true statement. (This latter statement is formed from $U[\ v_1, ..., v_n\ ]$ by replacing all free occurrences of v_k with e_k for $1 \leqslant k \leqslant n$.)

$T[\ e_1/v_1, ..., e_n/v_n\ ]$ is a `test assertion` for constraint C if and only if (e_1, ..., e_n) is in J(C).

The age constraint above is already in this form. Relative to the family symbol database, J(AGE) equals { (#501, 32, 35), (#502, 54, 57) }, where the ordering of the free variables is as in (v_P_ID, v_age_mom, v_age_dad). The (false) test assertion for jurisdiction element (#501, 32, 35) is:

```
for each value of v_C_ID, v_age_kid(
    if C_SEG_ID( v_C_ID )
        and LINKED_TO( #501, v_C_ID )
        and SEG:AGE_OF_CHILD( v_C_ID, v_age_kid )
    then v_age_kid < 32
        and v_age_kid < 35 )
```

As a final comment, observe that the integrity constraints for a database actually form an assertion database themselves for which one corresponding symbol database consists of the associated logic language sentences. The corresponding implementation database is very likely the way one would write or type the sentences. Thus the "integrity constraints" for a database can be seen to be a database in their own right; one wants the original database to be such that its integrity constraint database accurately describes it.

4. The Design Of Database Integrity Measures

A common way to define a database integrity measure is to define it as a function mapping databases to numerical indices of their integrity. This paper takes a somewhat different approach. The guiding idea is this: Suppose there is a function π which maps statements to numbers between 0 and 1 which indicate the proximity of those statements to being true. Then in order to measure the integrity of a database, it suffices to apply π to the one big assertion one would like to have be true for the database, namely the conjunction of all of the basic integrity constraints for that database. Call this conjunction the database's `meta-constraint`. Consequently, a `measure of database integrity` is just a π-type function which is evaluated primarily on database integrity constraints. As will be seen, the highly structured, recursive form of integrity constraints expressed in the logic language virtually automates the definition of measures of database integrity.

The first step in the recursive definition of a truth proximity measure π is to specify the values that π takes on the membership claims associated with various predicates. Here it will suffice to consider only those π such that, for all membership claims M, $\pi(M) = 1$ if M is true and $\pi(M) = 0$ if M is false.

The next step is to extend π to the measurement of `compound logic language statements` which are those that are formed from one or more membership claims by the use of connectives and quantification. In general, π measures of compound statements will be functions not only of the π measures of their constituent statements but also of positive weights that have been assigned to these constituent statements.

Like π, the weight function w will be recursively defined. For each membership claim M, w(M) is defined to be some positive number, possibly solely dependent on the predicate involved in the claim. In the `equi-weighting scheme`, w(M) is a constant (say, 1) for all membership claims M.

The most predictable extensions for π and w are to define for statements S, π(not S) to be $1 - \pi$(S) and w(not S) to be w(S).

Letting w(S_1) and w(S_2) be the positive weights associated with statements S_1 and S_2, here is the weighted average definition for π(S_1 and S_2):

(weighted average) $$\frac{w(S_1)\pi(S_1) + w(S_2)\pi(S_2)}{w(S_1) + w(S_2)}$$

In support of this definition, observe that π(S_1 and S_2) = 1 if and only if both π(S_1) and π(S_2) are 1. Also, as π(S_1) increases (decreases), so also does π(S_1 and S_2). w(S_1 and S_2) is defined to be w(S_1) + w(S_2).

Since a statement in universal implication form with non-empty jurisdiction is true if and only if all of its test assertions are true, the π value of such a statement with a finite jurisdiction is defined to be the π-value of the conjunction of all of its test assertions. In the special case that the

jurisdiction is empty, the π-value is defined to be 1. The w value of a statement in universal implication form with finite jurisdiction is 1 if the jurisdiction is empty and is the w value of the conjunction of the test assertions if otherwise.

For example, if conjunctions are measured by averages under the equi-weighting scheme, then (by applying π recursively), π(AGE) = 7/8 (not 1/2).

-- V criteria --

A V criterion measure of database integrity is a π-type truth proximity measure which measures conjunctions with weighted averages and which is used to measure constraints for databases all of whose basic constraints are in universal implication form. Hence, the V criterion maps the meta-constraint for a database to a weighted average of the values it takes on the test assertions associated with *all* of the basic integrity constraints. (The "V" is for "validity".) In symbols, for basic constraints $C_1, \ldots, C_m$ and designated weight function, $V(C_1 \text{ and } \cdots \text{ and } C_m) =$

$$\frac{\sum_{i=1}^{m} \sum_{\underline{e} \in J(C_i)} w(T_i[\underline{e} / \underline{v}]) \cdot V(T_i[\underline{e} / \underline{v}])}{\sum_{i=1}^{m} \sum_{\underline{e} \in J(C_i)} w(T_i[\underline{e} / \underline{v}])}$$

where $\underline{e}$ is a vector of domain entities and $\underline{v}$ is an appropriate vector of variables. π(AGE) above is a V criterion value.

It is worthwhile examining how V's value on a conjunction of m statements is determined by its value on a conjunction of two statements. In this regard, note that V's value on the meta-constraint is not an average of V's values on the basic constraints. Finally, observe that, under the equi-weighting scheme, if each test assertion is a membership claim, then the value of the V criterion is that fraction of all the test assertions which are true.

-- the U criterion --

The nature of the V criterion is illumined by contrasting it with what is probably the most widely used measure of database integrity, namely a measure here called the U (for unreliability) criterion. U criteria are typically defined for record-based implementation databases. In its simplest form, a U criterion for such a database is determined by decomposing that database's records and/or pointers into pieces and then computing that fraction of those pieces which are in error (or, alternatively, which are unreliable in some sense). In the case of a hierarchical implementation database, such pieces may be record trees or records or sub-record byte groups or access-path pointers.

The essential difficulty with the U criterion is not that it doesn't measure something of some interest but rather that in general, it is very hard to compute. The computation of the U criterion presupposes that it is possible to identify exactly which implementation pieces are incorrect when a test assertion is false. This is frequently impossible if two or more such pieces refer to values from a single jurisdiction tuple generating a false test assertion. For example, suppose a test assertion for AGE fails: one doesn't necessarily know if it failed because the AGE_OF_CHILD value is incorrect or because the AGE_OF_MOTHER (or AGE_OF_FATHER) value is incorrect. What is even more insidious here is that the incorrect implementation pieces may be the ones referring to values from the jurisdiction tuple that do *not* appear in the test assertion. After all, the values that appear in the test assertion are not the only ones being tested: if the other ones are incorrect, then one may be making test assertions that one shouldn't be making and conversely.

Anyway, if the implementation pieces are field value implementations, then which is to be marked wrong? The U criterion demands an answer; the V criterion doesn't care since it is only concerned with whether or not a test assertion is true. Furthermore, if for some reason the pieces are records or record trees and even if one can decide which piece is at fault, then is it really fair to denigrate every field value implementation in the piece in the event that only one such byte group is involved in the test assertion?

In short, the V criterion is concerned with determining the truth or falsity of assertions *about* a database component (such being integrity constraints) whereas the U criterion is concerned with determining the correctness or incorrectness of the symbol implementations themselves. As it turns out, the former is easier to do than the latter. The U criterion is the result of paying too much attention to symbols and their implementations. The V criterion is the more successful because it works *directly* with the integrity constraints themselves, not with database symbols and implementations.

5. Sampling To Estimate Database Integrity

Statisticians can use the material of the preceding sections to help design appropriate measures of database integrity. Once this has been done, they can begin to decide how to estimate these measures. The first step is to decide what is to be the sampling unit.

As with implementation, anything true to the purpose at hand is fair in the selection of a sampling unit. The best choice for a sampling unit is one which is feasible, one which

yields valid estimates for the parameter, and one which yields the optimal combination of efficient sampling and small variance.

One possible choice for the sample universe is the set of all test assertions over all basic constraints. This is inadequate however since two elements from the same jurisdiction can generate the same test assertion. So consider instead a sample universe which is the union of all the jurisdictions of the basic constraints. Observe though that this is not well-defined since, given a tuple, one does not necessarily know which test assertion to generate from it. Some form of constraint identification is necessary. To `tag` a basic constraint jurisdiction is to map that jurisdiction to a set of ordered pairs such that each jurisdiction tuple is mapped to a pair whose first component is an identifier for the constraint and whose second component is the tuple. The result is that the union of all tagged basic constraint jurisdictions is a legitimate candidate for the sample universe.

In this case, V can be re-written to explicitly indicate its relationship to this particular sample universe. If $UJ = \{(i, \underline{e}) : 1 \leqslant i \leqslant m \text{ and } \underline{e} \in J(C_i)\}$, then $V(C_1 \text{ and } \cdots \text{ and } C_m) =$

$$\frac{\sum_{(i,\,\underline{e}) \in UJ} w(T_i[\underline{e} / \underline{v}]) \cdot V(T_i[\underline{e} / \underline{v}])}{\sum_{(i,\,\underline{e}) \in UJ} w(T_i[\underline{e} / \underline{v}])}$$

As can be seen for this choice of sample universe, the V criterion value for the meta-constraint is a ratio which can be estimated by using standard ratio estimation techniques. If in addition, all test assertions have equal weight, then this V criterion value is a mean which can be estimated by standard mean estimation techniques.

-- root tagged jurisdictions --

In other situations, the basic constraints may be such that some tagged basic constraint jurisdictions can serve as roots for others. For example, consider the situation where the constraints form a list such that the jurisdiction of each constraint after the first is that portion of the jurisdiction of its predecessor which has true predecessor test assertions. In this case, the first tagged jurisdiction is considered to be the root of the others since each subsequent tagged jurisdiction J'_k for $k \geqslant 2$ can be determined from the previous one in the sequence J'_{k-1} by applying a known algorithm to each element of J'_{k-1}, namely, $(k, \underline{e})$ is in J'_k if and only if $(k-1, \underline{e})$ is in J'_{k-1} and $\underline{e}$ generates a true test assertion for the $(k-1)$th constraint.

In a more general setting, tagged jurisdiction A is a `root` of tagged jurisdiction B if a reach function has been defined r from A to B. The function r is a `reach function` from tagged jurisdiction A to tagged jurisdiction B if and only if (i) there is a known algorithm for computing its values, (ii) for each w in A, r(w) is a (possibly empty) subset of B, and (iii) $\{ r(w) : w \in A \}$ is a partition of B. A tagged jurisdiction element z is `reachable` from w if there is a reach function r such that $z \in r(w)$. So, tagged jurisdiction A is a root of tagged jurisdiction B if and only if (i) for each element w of A, one can determine whether or not elements of B are reachable from w and if so, which ones and (ii) each element of B is reachable from exactly one element of A. Every root tagged jurisdiction is considered to reach itself by virtue of the $w \rightarrow \{w\}$ reach function.

Now, in the event that no root tagged jurisdiction is the root of another one distinct from itself and no tagged jurisdiction has more than one root, it may well be convenient to let the sample universe be the union of the root tagged jurisdictions. In order to write the V criterion in terms of measurements on these sampling units, it is necessary to define two measurement functions, say, x and y. The x value on a sampling unit is the sum of the test assertion weights of the tagged jurisdiction elements reachable from that unit and the y value is the weighted sum of V's values on the associated test assertions. The corresponding V criterion value is the ratio of the sum of the universe y values to the sum of the universe x values. When sampling to estimate this V criterion value, one is cluster sampling to estimate a ratio.

Some symbols may aid the reader's understanding of the previous paragraph. Suppose that the first k tagged basic constraint jurisdictions are the roots of all the others. Then the formula for the V criterion may be rewritten to explicitly indicate its relationship to these sampling units and to the x and y measurement functions:

$V(C_1 \text{ and } \cdots \text{ and } C_m) =$

$$\frac{\sum_{1}^{k} \sum_{(k,\,\underline{d}) \in J'(C_k)} \sum_{(i,\,\underline{e}) \in R((k,\,\underline{d}))} \tilde{w}_i(\underline{e}) \cdot \tilde{V}_i(\underline{e})}{\sum_{1}^{k} \sum_{(k,\,\underline{d}) \in J'(C_k)} \sum_{(i,\,\underline{e}) \in R((k,\,\underline{d}))} \tilde{w}_i(\underline{e})}$$

$$= \frac{\sum_{1}^{k} \sum_{(k,\underline{d}) \in J'(C_k)} y((k,\underline{d}))}{\sum_{1}^{k} \sum_{(k,\underline{d}) \in J'(C_k)} x((k,\underline{d}))}$$

where $J'(C_k)$ is $\{k\} \times J(C_k)$, $R((k,\underline{d}))$ is the set of tagged jurisdiction elements reachable from $(k,\underline{d})$, $\tilde{w}_i = w \circ T_i[\,\cdot\,/\underline{v}]$ and similarly for $\tilde{V}_i$. Note that x and y are operating on tagged tuples.

It should be noted that in the preceding two situations, the sampling units may very well not be in one-to-one correspondence with any decomposition of the symbols (or implementations) in the symbol (or implementation) database. For example, observe that the jurisdiction of AGE is not in one-to-one correspondence with the set of segment trees, with the set of segments, with the set of segment field values, with the set of segment field values for AGE_OF_MOTHER, AGE_OF_FATHER, and AGE_OF_CHILD, or with the set of relationship linkings. If, in fact, the tagged jurisdiction of a constraint like AGE is the best choice for a sample universe (as indeed it was for the author in a Bell System application), then those who are thinking only in terms of symbolic sampling units will be at a loss to make the best choice.

-- symbolic sampling units --

However, while there are situations when no symbolic (or implementation) sampling unit is acceptable, there are also situations where symbolic sampling units are the most convenient. What must be done in these cases is to define at least implicitly a function ψ which maps elements of the union of tagged jurisdictions onto an exhaustive set of symbol pieces (such as the set of all segment trees or the set of all segments in the symbol database). One of the better ways to define ψ is to map a tagged jurisdiction tuple to that symbol piece which contains within it at least some of the values in the tuple. Then to define V, define two measurement functions, say x and y, on the sample universe as follows: the value that x takes on a sampling unit is the total test assertion weight of the tagged jurisdiction elements that ψ maps to that unit and the value that y takes on that unit is the weighted sum of V's values on the test assertions associated with those tagged jurisdiction elements. Here again as well, when sampling to estimate such a V criterion value, one is using cluster sampling to estimate a ratio.

As before, in symbols, if S is the set of symbol pieces, then the V criterion can be re-written as a function of the sampling unit as follows: $V(C_1 \text{ and } \cdots \text{ and } C_m) =$

$$\frac{\sum_{s \in S} \sum_{(i,\underline{e}) \in \psi^{-1}(s)} w(T_i[\underline{e}/\underline{v}]) \cdot V(T_i[\underline{e}/\underline{v}])}{\sum_{s \in S} \sum_{(i,\underline{e}) \in \psi^{-1}(s)} w(T_i[\underline{e}/\underline{v}])}$$

Often, one is not only interested in the V criterion value for the meta-constraint; the V criterion values for the individual basic constraints may also be of interest. Such a group of V criterion values form a trivial V criterion hierarchy. Non-trivial V criterion hierarchies also exist. In one of the author's Bell System applications, it was necessary to estimate a four level V criterion hierarchy by using one level of subsampling and two levels of stratification. The details of this application of sampling to estimating database integrity are left to subsequent papers.

By way of summary, an awareness of the assertion, symbol, and implementation components of a database serves to focus appropriate attention on each and to explain the nature of various kinds of integrity constraints. Symbolic logic can be used to reveal the structure of integrity constraints as well as to create a vocabulary for talking about them. Furthermore, the structure of constraints expressed in logic can be used to formally suggest meaningful measures of database integrity. In order to use sampling to estimate these measures of database integrity, a suitable sampling unit must be identified. A theory for the selection of appropriate sampling units is made possible by an understanding of both the nature of databases and the role that symbolic logic can play in formalizing the expression of integrity constraints.

Acknowledgement

I would like to thank B. Gopinath for many valuable conversations. The V criterion was originally developed by Gopinath in a non-logic, non-π-measure setting.

References

[1] W. G. Cochran, *Sampling Techniques -- Third Edition.* New York: John Wiley, 1977.

[2] C. J. Date, *An Introduction To Database Systems -- Third Edition.* Reading, Mass.: Addison-Wesley, 1982.

[3] E. Mendelson, *Introduction To Mathematical Logic.* Princeton: Van Nostrand, 1964.

A Security Model for the Statistical Database Problem

Dorothy E. Denning[1]

Abstract. A security model for addressing the statistical inference problem in online query processing systems is described. The set of all statistics computed over groups of records having common characteristics are structured as a lattice of logical tables. The lattice model provides a mathematical basis for studying the inference problem, and a framework for evaluating and comparing different controls. Although the lattice model has been used by census agencies to protect statistical tables published offline, it has only recently been used to develop inference controls for general purpose query processing systems.

1. Introduction

With the rapid proliferation of online database systems containing valuable or confidential information, computer scientists became concerned with the problem of protecting this data from unauthorized disclosure or modification. This concern led to the development of access controls, which ensure that all retrieval, update, insert, and delete operations are performed only by authorized individuals.

Although access controls solve much of the database security problem, they do not completely solve it. One area they do not address is the inference problem or statistical database problem. The problem here is to provide certain users with statistical access to sensitive data, while ensuring that the sensitive data cannot be inferred from released statistics.

Over the years, census agencies in the United States, Canada, and Sweden have developed many techniques for ensuring the confidentiality of statistical data published offline in tabular form [1, 2, 3, 4, 5, 6, 7]. These techniques have been developed within the framework of a publication hierarchy [8], which is a lattice structure of statistical tables. The table structures are analyzed prior to publication, and table entries (cells) are suppressed or perturbed where needed to protect sensitive data.

In the 1970's, the computer science community began to study the statistical database problem in general purpose online query processing systems, where arbitrary sets of records can be formulated in high-level languages. Because many online databases are continually evolving, efficient controls that can be applied during query processing time are desirable.

The early results of our research were discouraging. A user with the ability to formulate queries for statistics over essentially arbitrary groups could circumvent many controls with just a few queries. To quote a statement made by this author at the 1977 Symposium on the Interface, "With few exceptions, most of the proposed controls are either easy to circumvent or impractical to implement" [9].

Recent research done in collaboration with Jan Schlörer at the University of Ulm, W. Germany has become much more encouraging. An important insight

1. Author's address until May 18: Computer Sciences Dept., Purdue Univ., W. Lafayette, IN 47907; After May 18: SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025. Research supported in part by NSF Grant MCS80-15484.

came when we began to transform our relational view of the data into a lattice of statistical tables. Although we did not consciously plan to use the census model, this is precisely where we landed. In the process, we have come to a better understanding of the problem, and are beginning to find a collection of controls that can provide a high level of security at reasonable cost. Not surprisingly, many of these controls are adaptations of techniques developed by census agencies.

The purpose of this paper is to describe the lattice model, and to show why it is a useful security model for the inference problem in online query processing systems. If a database is described by some other data model, such as the relational model, then this model can be mapped onto the lattice for the purpose of developing security controls. We will show how this can be done for the relational model and for one statistical database model.

2. The Lattice Model

For security purposes, a statistical database can be viewed as a lattice of logical tables defined by the attributes of the database. There are M **primary** or **atomic attributes** $A_1, \ldots, A_M$. Each primary attribute A_i gives rise to a **hierarchical structure** of attributes $A_i^1 < \cdots < A_i^{k_i}$, where $A_i^1 = A_i$ and, for $k_i > 1$, $A_i^2, \ldots, A_i^{k_i}$ are **secondary** or **clustering attributes**. The relation $A_i^j < A_i^{j+1}$ means that A_i^{j+1} is a clustering or aggregation of values in the domain of A_i^j; conversely, A_i^j is a refinement or disaggregation of A_i^{j+1}. Since A_i^1 is atomic, it cannot be further refined. If a primary attribute has no secondary attributes, the superscript "1" may be omitted. In the degenerate case where none of the attributes has secondary attributes, the model corresponds to that described in [10]; if in addition each attribute is binary, the model is equivalent to the bit string model of Kam and Ullman [11] (their model can also be interpreted as attribute hierarchies formed by bisection).

Example 1. Cities are aggregated by state, giving the attribute hierarchy:

$$A = A^1 = City \; < \; A^2 = State.$$

Each value in the domain of *State* names an aggregate of values in the domain *City* (e.g., 'California' names a group of cities that includes 'San Francisco', 'Berkeley', 'Carmel', etc.).

Example 2. In a university, departments are grouped by school, giving the hierarchy

$$A = A^1 = Department \; < \; A^2 = School.$$

Example 3. Ages may be aggregated at several levels:

$A^1 = \{1, 2, 3, 4, \ldots, 100, >100\}$.
$A^2 = \{[1\text{-}10], [11\text{-}20], \ldots, [91\text{-}100], [101\text{-}]\}$.
$A^3 = \{[1\text{-}20], [21\text{-}40], [41\text{-}60], [61\text{-}80], [81\text{-}100], [101\text{-}]\}$.
$A^4 = \{[1\text{-}40], [41\text{-}80], [81\text{-}100], [101\text{-}]\}$.

All primary and secondary attributes can be used to select subsets of records in the database. The database may have additional quantitative (summary) attributes that are used in statistical calculations but cannot be used in subset selection.

There is a functional dependency from an attribute A^j in a hierarchy to A^{j+1} ($j = 1,\ldots, k-1$). For example, *City* in Example 1 determines *State*; *Department* in Example 2 determines *School*; each age group in Example 3 determines the next age group. In some cases, this dependency may not be immediately apparent if the same name is used to denote different entities. For example, the name "Rochester" refers to a city in New York, Michigan, and Minnesota (and probably several other states as well). We will assume this apparent conflict is somehow resolved by using unique names to denote different entities (e.g., qualifying city names with state names).

Although an attribute A^{j-1} is a refinement of a higher-level attribute A^j (for $j > 1$), some values in the domain of A^j may not be disaggregated in A^{j-1}. For example, the age group [81-100] in A^4 of Example 3 appears in A^3. The other age groups in A^4, however, are disaggregated in A^3; for example, the group [1-40] is disaggregated into the two groups [1-20] and [21-40]. In general, some values in a domain may be carried through several levels in a hierarchy; e.g., the age group [101-] in Example 3.

Given $m \geq 0$ primary or secondary attributes $A_1^{j_1}, \ldots, A_m^{j_m}$, an **elementary m-set** over these attributes is an m-set specified by a **conjunctive formula**:

$$E = (A_1^{j_1} = a_1) \;\&\; \cdots \;\&\; (A_m^{j_m} = a_m)\,, \tag{1}$$

where a_i is a value in the domain of $A_i^{j_i}$. The set of all possible elementary m-sets over the attributes defines an m-dimensional logical table, or **m-table** for short. The size of the table is given by $s_m = \prod_{i=1}^{m} |A_i^{j_i}|$, where $|A_i^{j_i}|$ is the size of domain $A_i^{j_i}$. Each m-table partitions the complete database into s_m elementary sets. For $m = 0$, there is a single elementary set, denoted *ALL*, which is an aggregate of all records in the database.

Given attributes $A_1, \ldots, A_m$, there is an m-table for each combination of attributes in the hierarchies for $A_1, \ldots, A_m$. Since each attribute A_i has k_i atomic and clustering attributes in its hierarchy, the total number of m-tables over attributes $A_1, \ldots, A_m$ is $\prod_{i=1}^{m} k_i$.

The set of all tables forms a **lattice** with partial ordering relation "<", where $T_2 < T_1$ means that each elementary set in T_1 is a union of elementary sets in T_2. The total number of tables in the lattice is given by $\prod_{i=1}^{M}(k_i + 1)$. For the special case where there are no secondary attributes, this is 2^M. The top table in the lattice, denoted T_{ALL}, corresponds to the 0-set containing all records in the database. The bottom table partitions the database according to all atomic attributes $A_1^1, \ldots, A_M^1$. None of the tables in the lattice need exist as physical structures of the databases. The lattice is a logical structure.

Let T_1 be defined by the attributes $A_1^{j_1}, \ldots, A_m^{j_m}$, and let T_2 be a table directly beneath T_1 in the lattice. T_2 refines T_1 in one of two ways:

1. T_2 is an $m+1$-table over the attributes of T_1 plus an additional primary attribute A_{m+1}^1; that is, T_2 is defined by the attributes $A_1^{j_1}, \ldots, A_m^{j_m}, A_{m+1}^1$. Then each elementary set $E = (A_1^{j_1} = a_1) \;\&\; \cdots \;\&\; (A_m^{j_m} = a_m)$ in T_1 corresponds to a union of elementary sets in T_2 as follows:

$$E = \bigcup_{a_{m+1} \in A_{m+1}^1} (A_1^{j_1} = a_1) \;\&\; \cdots \;\&\; (A_m^{j_m} = a_m) \;\&\; (A_{m+1}^1 = a_{m+1})\,. \tag{2}$$

The number of possibilities for T_2 by this method is $M - m$.

2. T_2 takes one of the secondary attributes of T_1, say $A_i^{j_i}$, and refines it by taking it down one level in the hierarchy; that is, T_2 is defined by $A_1^{j_1}, \ldots, A_i^{j_i-1}, \ldots, A_m^{j_m}$. Then each elementary set $E = (A_1^{j_1} = a_1)\ \&\ \cdots\ \&\ (A_m^{j_m} = a_m)$ in T_1 is given by the union

$$E = \bigcup_{\substack{a_i' \in A_i^{j_i-1} \\ a_i' \to a_i}} (A_1^{j_1} = a_1)\ \&\ \cdots\ \&\ (A_i^{j_i-1} = a_i')\ \&\ \cdots\ \&\ (A_m^{j_m} = a_m)\ , \qquad (3)$$

where $a_i' \to a_i$ means that a_i' determines a_i. Thus, the union is taken over all values a_i' in $A_i^{j_i-1}$ that disaggregate a_i in $A_i^{j_i}$. The number of possibilities for T_2 by this method is n, where $n \leq m$ is the number of secondary attributes among $A_1^{j_1}, \ldots, A_m^{j_m}$.

Thus the total number of direct descendents of T_1 is $(M - m) + n \leq M$.

Figures 1 and 2 illustrate two simple lattice structures. Figure 1 corresponds to the lattice in [8], where attribute A is the hierarchy

A^1 = *City* (within *County*)
A^2 = *County*
A^3 = *State* ,

and attribute B is the hierarchy

B^1 = 6-digit industry code
B^2 = 4-digit industry code
B^3 = 3-digit industry code
B^4 = 2-digit industry code .

Figure 3 shows a more detailed view of a lattice over two attributes A and B, where B is hierarchically structured: $B = B^1 < B^2$. The entries inside the tables are counts of the number of records belonging to each set. Note that each table partitions the 102 records of the database into disjoint sets.

Statistics, such as the counts in Figure 3, are computed over subsets of records having common attribute values. A set of records is specified by a **characteristic formula** F, which, informally, is any logical formula over the values of the primary or secondary attributes using the logical operators OR (+), AND (&), and NOT(~), plus the relational operators. If a formula is expressed solely in terms of logical AND and equality, then it is a conjunctive formula of the form (1) specifying an elementary set. The set of records whose values match a characteristic formula F is called the **query set** of F.

We will concentrate mainly on additive statistics [10]. Letting q denote a statistical function, q is **additive** if and only if

$$q(F_1 + F_2) = q(F_1) + q(F_2) \qquad (4)$$

when F_1 and F_2 are disjoint query sets. Counts, sums, and higher order moments [12] are additive.

Additive statistics have the important property that the statistics in the tables of the lattice are linearly related. In particular, if $T_2 < T_1$, then the statistics in T_1 are marginal sums of those in T_2. This means that the statistics in T_1 can be computed from those in any descendent of T_1 in the lattice. To make this precise, let T_1 be an m-table over the attributes $A_1^{j_1}, \ldots, A_m^{j_m}$, and let T_2 be a table directly below T_1 in the lattice. Let $E = (A_1^{j_1} = a_1)\ \&\ \cdots\ \&\ (A_m^{j_m} = a_m)$ be an elementary set in T_1. Then $q(E)$ is

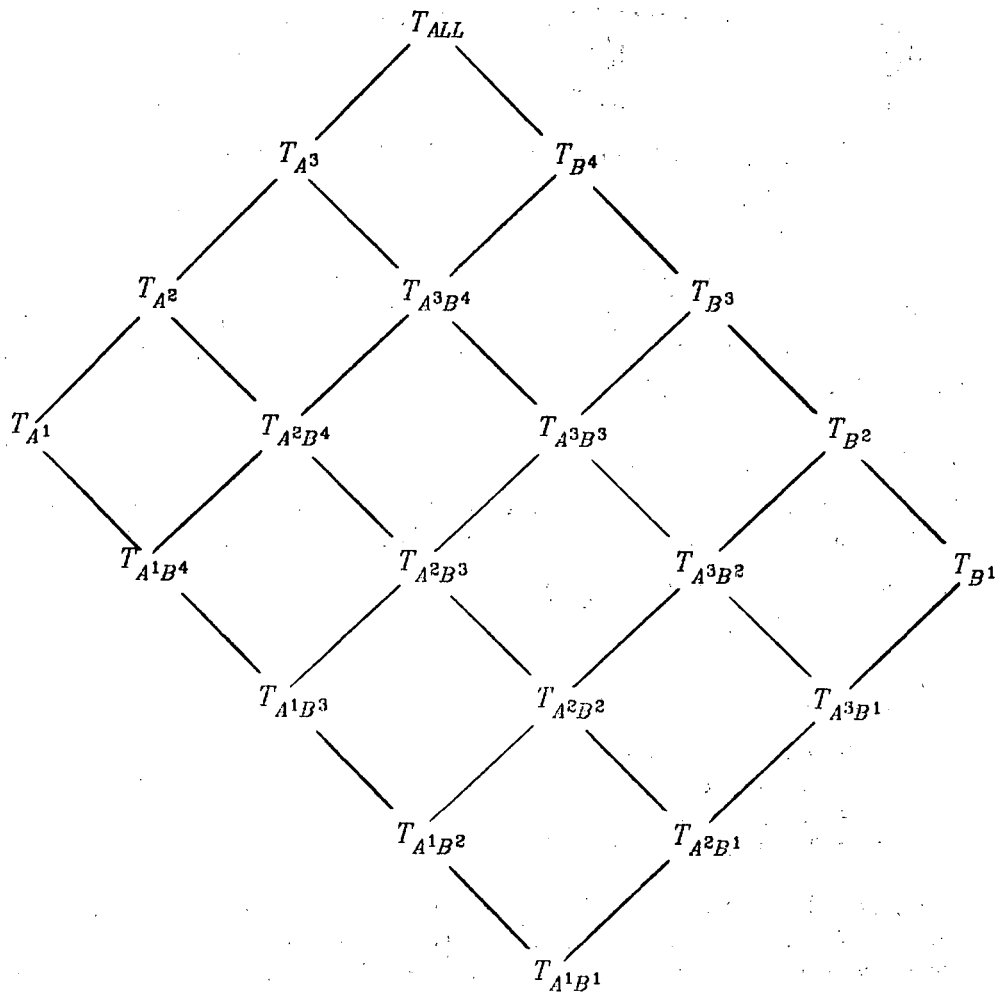


Figure 1. Lattice of tables over attributes $A = A^1 < A^2 < A^3$ and $B = B^1 < B^2 < B^3 < B^4$.

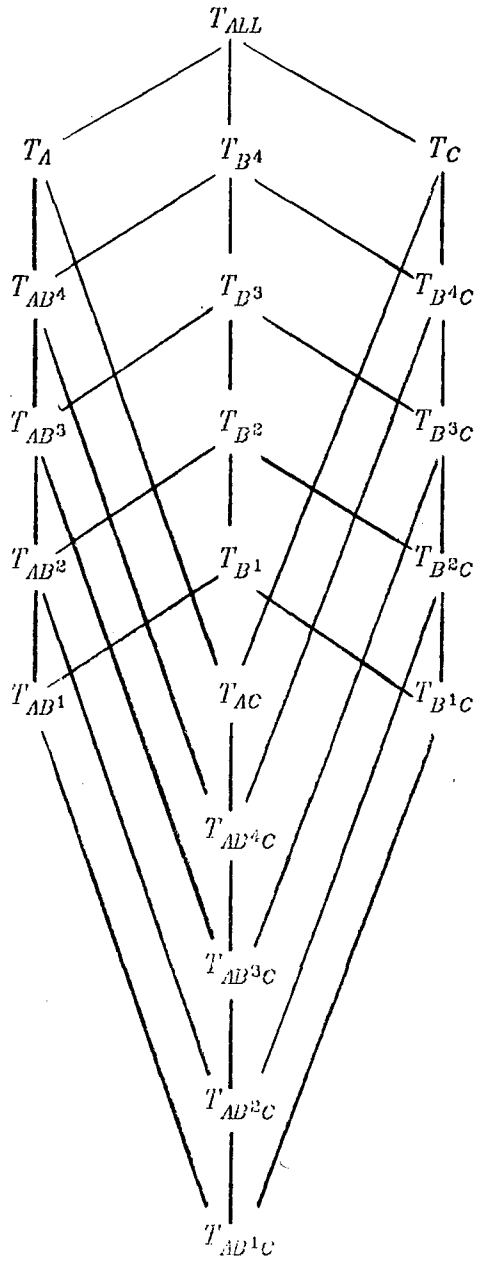


Figure 2. Lattice of tables over attributes A, B, and C; B is hierarchically structured as $B = B^1 < B^2 < B^3 < B^4$.

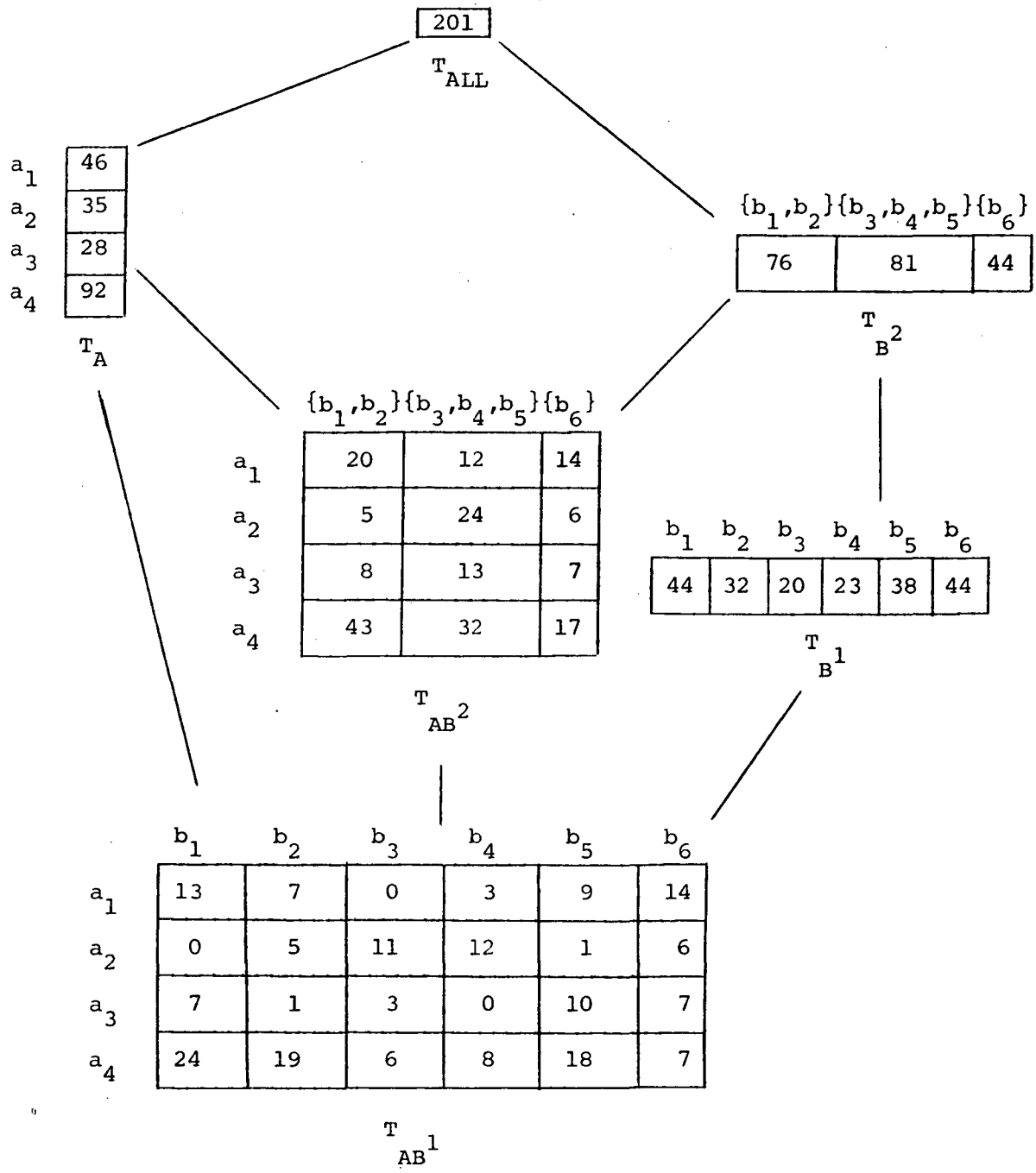


Figure 3. Lattice of counts over attributes A and $B = B^1 < B^2$.

computed from T_2 in one of two ways, depending on whether T_2 refines T_1 by adding an extra attribute (dimension) A^1_{m+1} or by disaggregating one of the attributes A_j:

1. If T_2 is an $m+1$-table over the attributes $A_1^{j_1}, \ldots, A_m^{j_m}, A^1_{m+1}$, then

$$q(E) = \sum_{a_{m+1} \in A^1_{m+1}} q((A_1^{j_1} = a_1) \;\&\; \cdots \;\&\; (A_m^{j_m} = a_m) \;\&\; (A^1_{m+1} = a_{m+1})) . \quad (5)$$

   The counts in tables T_A and T_{B^1} of Figure 3, for example, are the marginal sums of T_{AB^1}; the counts in T_{B^2} are the marginal sums of T_{AB^2}; and the counts in T_{ALL} are marginals of both T_A and T_{B^2}.

2. If T_2 is an m-table defined by $A_1^{j_1}, \ldots, A_i^{j_i-1}, \ldots, A_m^{j_m}$, then

$$q(E) = \sum_{\substack{a_i' \in A_i^{j_i-1} \\ a_i' \to a_i}} q((A_1^{j_1} = a_1) \;\&\; \cdots \;\&\; (A_i^{j_i-1} = a_i') \;\&\; \cdots \;\&\; (A_m^{j_m} = a_m)) . \quad (6)$$

   The counts in T_{AB^2} of Figure 3 are obtained by adding groups of columns in T_{AB^1}.

Additivity implies that statistics in the bottom table of the lattice, namely the table over all M atomic attributes $A^1_1, \ldots, A^1_M$, is a basis for computing all other tables in the lattice.

Additivity has another important property as well: statistics for nonelementary query sets (i.e., those sets that cannot be defined without using logical OR, NOT, or the relational operators other than equality) can be computed from those in the tables. For example, $count((A \neq a_3) \;\&\; (B = b_2))$ can be computed by adding the statistics in column 2 of table T_{AB^1} of Figure 3, omitting the statistic in row 3. This property implies that an m-table of statistics over attributes $A_1^{j_1}, \ldots, A_m^{j_m}$ serves as a basis for computing all statistics of a given type for query sets over these attributes. Moreover, the bottom table of the lattice serves as a basis for computing all possible statistics of a given type.

Additivity has important implications for both security and freedom of information: we can restrict the syntax of queries to conjunctive formulas without causing any information loss. Indeed, we could conceivably withhold all statistics in the database except for those in the bottom table of the lattice without causing any loss. Of course, this does not solve the security problem because the low-level tables, in particular the bottom table, usually contain sensitive data that must be withheld to ensure privacy. But, as discussed later, restricting the syntax is a useful technique for thwarting many attacks.

Eqs. (5) and (6) show how statistics of a given type (e.g., all counts or all sums over a given attribute) are related in the lattice. Statistics of different types may also be related. For example, let T_1 be an m-table over attributes $A_1^{j_1}, \ldots, A_m^{j_m}$, and suppose T_1 contains sums over attribute A_{m+1}. If A_{m+1} can be used to select subsets of records, then the sums in T_1 are linearly related to counts in T_1's descendent over $A_1^{j_1}, \ldots, A_m^{j_m}, A^1_{m+1}$ as follows:

$$sum((A_1^{j_1} = a_1) \;\&\; \cdots \;\&\; (A_m^{j_m} = a_m)) = \quad (7)$$

$$\sum_{a_{m+1} \in A^1_{m+1}} [count((A_1^{j_1} = a_1) \;\&\; \cdots \;\&\; (A_m^{j_m} = a_m) \;\&\; (A^1_{m+1} = a_{m+1})) * a_{m+1}] .$$

Nonadditive statistics are related to additive statistics through counts and other

higher-order statistics. For example, $mean(F) = sum(F)/count(F)$ for any formula F. (See also [13].)

Users often want complete tables of statistics (e.g., total salaries of employees in a 2-table broken down by sex and age) or cross sections of tables. Such queries can be expressed by allowing "*" for an attribute value in the formula, where $(A_i^{j_i} = *)$ generates the set of all values in the domain of $A_i^{j_i}$ [14, 5]. Looking at Figure 3, the query $q((A = *) \& (B^1 = *))$, for example, returns the entire 2-table of statistics T_{AB^1} in Figure 3; the query $q((A = *) \& (B^1 = b_3))$ returns the statistics in column 3 of T_{AB^1}; and $q(B^1 = b_3)$ returns the column sum, which is a cell in the 1-table T_{B^1}.

The table structures of the lattice model are closely tied to the publication format of statistical applications, including census data, where the marginal sums in the higher-level (lower-dimensional) tables are usually displayed with the lower-level tables from which they are derived. Figure 4 illustrates a single-table presentation of the statistics in all six tables of Figure 3.

3. The Security Problem

The problem is to prevent the inference of sensitive statistics. A statistic is sensitive if confidential data could be deduced from the statistic alone. A statistic computed from confidential information for a group of size 1 is usually considered to be sensitive. Thus, the count for $(A = a_3) \& (B = b_2)$ in table T_{AB^1} of Figure 3 is sensitive. A statistic computed from a group of size 2 may also be classified as sensitive because a user with supplementary knowledge about one of the values can deduce the other from the statistic. The exact criterion for sensitivity is determined by the policies of the system. One criterion used by the U.S. Census Bureau for economic data is the "n-respondent, k%-dominance" criterion, which defines a sensitive statistic to be one where n or fewer records comprise more than k% of the total [8, 1]; n and k are parameters of the database, usually kept secret. The **disclosure risk**, or **identification risk**, of a table is given by the number (or percent) of sensitive cells in the table.

Personal disclosure (compromise) occurs when the user can infer a previously unknown sensitive statistic about an identifiable individual [15, 16]. Disclosure may be either exact or approximate, positive or negative [17, 18, 15, 5, 6]. Releasing counts for query sets of size 0 always leads to negative disclosure because one can deduce that a particular individual does not have the associated properties.

Clearly, all sensitive statistics must be restricted (i.e., not permitted). In addition, we must restrict nonsensitive statistics that could lead to disclosure of sensitive ones. Such disclosures arise mainly from the linear relationships in the lattice structure, as defined by Eqs. (5)-(7). Given all of the statistics in one of these equations but one (which is sensitive), the missing statistic is easily computed. For example, using Eq. (5), the sensitive count for $(A = a_3) \& (B = b_2)$ in T_{AB^1} of Figure 3 can be computed by subtracting the other entries in column 2 of the table from the column sum in T_{B^1}.

In general, a sensitive cell in a table T cannot be inferred from tables above T in the lattice; it is usually necessary to obtain other cells in T (or at least statistics computed over the attributes defining T). There are exceptions, for example, when there are "magical zeros" [5, 10], or when the counts disaggregated by attribute A_{m+1} in an $m+1$-table are determined by the sum over A_{m+1} in the parent m-table as defined by Eq. (7) [19]. For example, if the domain of $A_{m+1} = \{11, 17\}$, then the only integral solution to the sum

	b_1	b_2	$\{b_1,b_2\}$	b_3	b_4	b_5	$\{b_3,b_4,b_5\}$	b_6	Total
a_1	13	7	20	0	3	9	12	14	46
a_2	0	5	5	11	12	1	24	6	35
a_3	7	1	8	3	0	10	13	7	28
a_4	24	19	43	6	8	18	32	7	92
Total	44	32	76	20	23	38	81	44	201

Figure 4. Publication format for counts of Figure 3.

$$50 = n_1 \cdot 11 + n_2 \cdot 17$$

is $n_1 = 3$, $n_2 = 1$. Because this method of attack requires exact answers, perturbation techniques (see Section 4.3) are a good countermeasure.

To prove that information in an $m+1$-table cannot be inferred exactly from parent tables, one can show that the database is m-transformable [20,21]. Unfortunately, this is not usually practical.

The lattice model is a powerful and effective tool for modeling the security problem. It provides a simple structure for relating different statistics through linear equations (or more complex equations for nonadditive statistics). This is important for understanding the rules of inference used by an adversary to compute a sensitive statistic from nonsensitive ones, and for proving that a particular countermeasure makes such inferences impossible or unlikely. With this framework, it becomes immediately obvious why certain controls are ineffective or undesirable. For example, neither a **query set size control** [22] nor an **overlap control** [23] foil "trackers" [24, 25, 26, 16, 27] and other inference techniques that exploit the linear relationships [18]. Moreover, an overlap control is readily seen to be undesirable, as it rules out releasing many statistics for aggregates, which are vital to most statistical applications.

4. Security Mechanisms

The lattice model provides a framework for evaluating and comparing different controls in terms of their security and information loss. Security is measured by the relative number of sensitive statistics that can be inferred by circumventing the control, by the difficulty (computational complexity) of doing so, and by the probability of success. Information loss is measured by the number of nonsensitive statistics or tables of statistics that are unnecessarily restricted by the control, and by the amount of noise injected in permitted statistics.

There are two general techniques for enforcing security: restriction and perturbation. Restriction techniques aim to prevent inference of sensitive statistics by withholding additional nonsensitive ones. In [28], we survey various strategies, classifying them according to whether they restrict at the table level or cell level in the lattice. The controls are also classified according to whether they are a priori (precompute which statistics to release), audit based (decide whether to release a statistic by consulting a log of previously released statistics), or memoryless (use heuristics at query processing time).

4.1. Table-Level Restriction Techniques

Table-level controls restrict complete m-tables of statistics, including all statistics for query sets defined by the associated attributes. Security is measured in part by the number of tables that are falsely permitted; i.e., are permitted despite having sensitive cells. Information loss is measured by the number of tables that are falsely restricted; i.e., are restricted even though there are no sensitive cells.

We have used the lattice model to describe and evaluate several memoryless table restriction criterion [10]. Two attractive criterion are relative table size and explicit risk estimation; both are heuristics. The **relative table size** (s_m / N) criterion restricts an m-table of counts when its size s_m relative to the number N of records in the database exceeds a threshold $1/k$. The control is applied to a query $q(F)$ by taking the product of the domain sizes for the attributes named in F, thereby obtaining the size of the corresponding logical table in the lattice. For $k = 10$ and the database of Figure 3, all counts

over attributes A and B^1 would be restricted since the relative size of table T_{AB^1} is 24/201 = .12 > .10; all other counts would be permitted. Thus, the sensitive cells in the restricted table T_{AB^1} are protected. The results of experiments reported in [10, 29] show that relative table size can predict the disclosure risk of a table. **Explicit risk estimation** uses frequency distributions of the data to obtain even closer estimates of table risk, but at the price of increased computation [10, 29]. Table level controls can also be used with statistics other than counts, e.g., sums; higher thresholds are generally needed, however, because higher-order statistics contain more information [10].

Clustering attributes can enhance security while reducing information loss. For example, if the 2-table T_{AB^1} is restricted, statistics over attributes A and B can still be released through the clustering attribute B^2, because table T_{AB^2} does not contain sensitive cells. Schlörer [30] studies the security aspects of clustering, showing that clustering attributes must form hierarchies as in the lattice model. The inclusion of clustering attributes to enhance security is also called "grouping" or "rolling up" [3, 4, 5].

4.2. Cell-Level Restriction Techniques

Cell-level controls aim to restrict only the sensitive cells of an m-table, and just enough nonsensitive statistics over the associated attributes to prevent inference. Security is measured by the number of sensitive cells that can be inferred; information loss by the number of nonsensitive cells that are restricted. Although cell level controls can be applied to queries for single cells (or cell unions), they are better suited to queries for complete tables or cross sections of tables. This is because they must examine more than one cell of a table to determine whether a particular cell can be securely released.

An example of a cell-level control is **cell suppression**, which is an a priori control used by census agencies to protect data published in tabular form. The linear relationships among all cells of a table and the marginal sums in the parent tables are analyzed to determine whether sensitive cells can be deduced (exactly or approximately) from those that are released; additional cells, called complementary suppressions, are suppressed until this is no longer possible [8, 1, 5, 6, 7]. To prevent inferences using Eqs. (5)-(7), the suppressed cells must fall into (possibly overlapping) hypercubes of size 2^m, where m is the size of the table; that is, 2 cells in each row (column, etc.) are suppressed. In addition, if any attribute is aggregated into clusters at a higher level in the lattice, then the hypercubes must be contained within clusters. Figure 5(a) illustrates how cell suppression could be applied to table T_{AB^1} of Figure 3. Three cubes are suppressed, where the cells of each cube are marked X, Y, and Z respectively. (cubes Y and Z overlap on one cell). Figure 5(b) shows an alternative suppression that is not secure because hypercube Z is not contained within a cluster. The sensitive cell in row 2, column 5, for example, can be deduced by subtracting the cells in columns 3 and 4 from the marginal in T_{AB^2} for the row 2 cluster $\{b_3, b_4, b_5\}$.

Because cell suppression can be expensive, it is used as an a priori control rather than on a per query basis. A memoryless heuristic based on the principle of restricting hypercubes of statistics has been proposed as a less expensive alternative for online query processing systems with conjunctive queries [10].

By providing statistics for aggregates, clustering attributes can reduce the information loss caused by suppressing nonsensitive cells. For example, the query $count([(A = a_3) \;\&\; (B^1 = b_1)] + [(A = a_3) \;\&\; (B^1 = b_2)])$ would not be allowed because the formula is the disjunction of restricted cells. This statistic

	$\{b_1$	$b_2\}$	$\{b_3$	b_4	$b_5\}$	$\{b_6\}$
a_1	13	7	Y	Y	9	14
a_2	×	×	11	Z	Z	6
a_3	×	×	Y	Y/Z	Z	7
a_4	24	19	6	8	18	7

a) Secure suppression

	$\{b_1$	$b_2\}$	$\{b_3$	b_4	$b_5\}$	$\{b_6\}$
a_1	13	7	Y	Y	9	14
a_2	×	×	11	12	Z	Z
a_3	×	×	Y	Y	Z	Z
a_4	24	19	6	8	18	7

b) Insecure suppression (hypercube Z crosses partitions).

Figure 5. Cell suppression applied to T_{AB}^{1} of Figure 3.

can be obtained, however, through the clustering attribute B^2 with the query $count((A = a_3) \ \& \ (B^2 = \{b_1, b_2\}))$.

4.3. Perturbation Techniques

Perturbation techniques add noise to statistics. These techniques are usually used with some form of restriction technique, applied at either the table or cell level. Perturbation techniques are judged not only by their security and information loss, but by their bias, which should be zero or at least negligible, and by their consistency. Inconsistencies arise when, for example, repetitions of the same query yield different results, or when the statistics in a row (column, etc.) of a table do not add up to their marginal sum. Unfortunately, the goals of consistency and statistical quality of perturbed statistics can be conflicting [31, 4, 6, 32, 33], so that perfect consistency is probably unrealizable.

We also survey perturbation techniques in [28], classifying them according to whether are record (input) based or output based. Given a query $q(F)$, record based techniques perturb the input to the statistical function for q. An example of a record based technique is **random sample queries** [34, 18], which uses random samples of the records in a query set to compute a statistic. One of the difficulties encountered with this strategy is maintaining consistency. If a statistic can be requested in many different ways, and each query returns a different response, then a better estimate of the true statistic can be obtained by averaging the responses. To see how this might be done, suppose sampling is applied to queries for nonsensitive statistics over attributes A and B^1 of T_{AB^1} in Figure 3 (queries for sensitive statistics would be suppressed). Then the query $count((A = a_3) \ \& \ (B^1 = b_1))$ for the nonsensitive statistic in row 3, column 1 would return a count computed from a sample of the records in the query set, where the variance of the perturbed count is great enough that the "1" in row 3, column 2 cannot be accurately inferred (using the marginal in T_{AB^2}). But additional estimates of the perturbed count might be obtained by formulating the query in different ways, for example:

$$count((A = a_3) \ \& \ (B^1 < b_2))$$
$$count((A > a_2) \ \& \ (A < a_4) \ \& \ (B^1 < b_2))$$
$$count([(A = a_3) \ \& \ (B^1 = b_1)] + [(A = a_3) \ \& \ (B^1 < b_2)])$$

By taking the average of many such estimates, a better estimate of the true count can be obtained.

To protect against averaging attacks, equivalent queries should always return the same response. One way of doing this is by restricting the syntax of queries to conjunctive queries, which allows easy reduction to a normal form. The reduced normal form would then functionally determine which records are selected for the sample. A less restrictive, but somewhat more costly, way is to make the sample dependent on the composition of the query set. This could be done by first making a preliminary pass over the query set to compute a checksum over the record id's; the checksum would then functionally determine which records are selected for the sample [34, 18].

Unfortunately, guaranteeing a constant response for equivalent queries is not sufficient. If table T_{AB^1} has d descendents in the lattice, then estimates of the cell in row 3, column 1 can be obtained from these descendents. Let $E = (A = a_3) \ \& \ (B^1 = b_1)$, and suppose the syntax of queries is unrestricted (with the checksum scheme used to ensure constancy for statistics computed over the same query set). There is an exponential number of ways of estimating $count(E)$ from each descendent of T_{AB^1} in the lattice. For example, suppose T_{AB^1C} is a descendent of T_{AB^1}, where C is the ordered domain $C = \{c_1, \ldots, c_t\}$.

Because there are $2^{t-1}-1$ ways of partitioning the domain of C into two nonempty subsets, there are $2^{t-1}-1$ ways of expressing $count(E)$ using just two queries; e.g.,

$$count(E \ \& \ (C = c_1)) + count(E \ \& \ (C > c_1))$$
$$count(E \ \& \ (C \leq c_2)) + count(E \ \& \ (C > c_2)) .$$

Because different query sets are used in each estimate, the checksum scheme does not detect equivalence among the pairs of queries. If $2^{t-1}-1 > 15|E|(1-p)/p$, where $|E|$ is the size of the query set and p is the expected fraction of records retained in the sample, then the set of all such estimates gives enough information to estimate $count(E)$ to within one record with a 95% confidence interval when $|E|$ is around 30 or so [34]. For $|E| = 30$ and $p = .75$, for example, a domain size t of just 8 gives enough estimates. If the syntax of queries is restricted to conjunctive formulas, then only one estimate of $count(E)$ can be obtained, namely $\sum_{i=1}^{t} count(E \ \& \ (C = c_i))$, so an averaging attack is not likely to succeed.

The checksum scheme has another potentially serious security flaw. Suppose it is known that a formula F uniquely identifies some individual in the database, but it is not known whether that individual has attribute $A = a_1$. Because the size of the query set $F \ \& \ (A = a_1)$ is 0 or 1, the database would withhold the statistic $count(F \ \& \ (A = a_1))$. It is, however, possible to deduce whether this count is 0 or 1 from two queries, $q(F + P)$ and $q(F \ \& \ (A = a_1) + P)$, where P is a tracker [24, 25, 26, 16]; that is, a formula disjoint from F that pads F with enough extra records that the queries are answerable [35]. If both queries return exactly the same answer, one can infer that both have the same query sets (otherwise the samples, and therefore responses, would be different with high probability), whence the individual has the attribute $A = a_1$ with high probability. This security flaw in the checksum scheme is not unique to random sample queries; it arises with any perturbation scheme (e.g., random rounding), where the perturbation is functionally determined by the composition of the query set.

In practice, it may be easy to prevent users from obtaining estimates of cells in T_{AB^1} using descendents of T_{AB^1} in the lattice. Because the descendents are further down in the lattice, they will contain more sensitive cells than T_{AB^1}. Therefore, lattice based restriction techniques can restrict statistics in these tables, and perturbation techniques can introduce larger errors so that statistics in lower-level tables cannot be used to estimate those higher up.

Output based techniques perturb a result $q(F)$ after it has been correctly computed, typically by **systematic or random rounding** [36, 4, 37, 38, 33], or by **controlled rounding** [39, 2, 40, 41, 42]. Systematic and random rounding can introduce inconsistencies. Controlled rounding forces consistencies for additive statistics by making the marginal sums of rounded statistics equal their rounded sum. Since this can be expensive, it is presently used only as an a priori control for offline publication of tables. Random rounding, which has the advantage of being unbiased, is vulnerable to averaging attacks in the same way as random sample queries. The solution is also the same: restrict the syntax to conjunctive formulas, and let the reduced normal form functionally determine whether to round up or down.

Two observations are worth noting. First, the attacks formulated in terms of "key-specified queries" [18], including the linear system attacks on sums [23, 43] and median attacks [44, 45, 46], may not be as serious as originally thought. Indeed, it seems unlikely that any of these attacks could succeed in a

system with security controls developed in the lattice model, though further research is needed to substantiate this. To see why, consider the following linear attack, which determines the value x_7 in 5 queries, where x_i is the value of some numeric attribute in record i, and q_j denotes the statistical sum returned in the jth query:

Obtain these statistics:

$$q_1 = x_1 + x_2 + x_3$$
$$q_2 = x_4 + x_5 + x_6$$
$$q_3 = x_1 + x_4 + x_7$$
$$q_4 = x_2 + x_5 + x_7$$
$$q_5 = x_3 + x_6 + x_7$$

Compute $(q_3 + q_4 + q_5 - q_1 - q_2)/3 = x_7$.

Now, to perform this attack, a user must be able to formulate characteristic formulas for the query sets in the attack (users with statistics-only access should not be permitted to request statistics for groups of named individuals). This means that the user must have enough supplementary knowledge about the individuals in the database that precisely controlled groups of individuals can be identified through characteristic formulas; for many applications, this information will not be available. But even if it is, the formulas defining these groups must use enough attributes to isolate single individuals, whence the query sets will correspond to sets (or set unions) defined over tables containing sensitive cells. Table-level and cell-level controls can prevent their release, or add enough noise that estimates of sensitive cells cannot be obtained.

The second observation is that there is a case for restricting the syntax of queries to conjunctive formulas. There is a paradoxical tradeoff between the power of the query language and the amount of obtainable information [33]. If the syntax of the query language is restricted to conjunctive formulas, then the only statistics released are those corresponding to table cells in the lattice; statistics for query sets defined by logical OR and NOT are not released. But restricting the syntax reduces considerably the number of possible attacks, so that the database can release more table cells, and more accurate statistics for these cells. Because additive statistics for arbitrary formulas can be computed from the table cells, more information may be effectively released than with a free syntax, where controls must be tighter.

This tradeoff arises with random sample queries and random rounding, where restricting the syntax to conjunctive formulas allows easy reduction to a normal form, and reduces exponentially the number of ways of expressing a particular query set. The tradeoff also arises with memoryless cell restriction techniques, where using conjunctive queries can permit the release of partial tables through a heuristic based on withholding hypercubes [10]. With a free syntax, complete tables of statistics must be withheld to ensure security. We also observe that the Swedish National Bureau of Statistics has adopted a partially restricted syntax [5, 6].

5. Data Models

We began our research on the statistical inference problem with a **relational** [47] view of the data. We modeled a statistical database as a single relation (also called universal relation [48]) of N records (tuples), where each record contains values for the M atomic attributes $A_1, \ldots, A_M$, plus any quantitative attributes used in statistical calculations, but not for specifying

query sets.

We chose this model over the publication hierarchy because we wanted to study the problem as it arises in general purpose online query processing systems, such as relational database systems, where arbitrary query sets can be formulated in high-level languages, and where some users may be allowed direct access to the data. Because many online databases are dynamic, we wanted to find efficient inference controls that could be applied at query processing time. This ruled out many techniques used by census agencies, including cell suppression, which are applied a priori to the one-time publication of tables.

The results of our research led us to the lattice model. This is not incompatible with our earlier objective of studying relational databases. A relation over M primary attributes is easily mapped onto a lattice structure for the purpose of studying security. Clustering attributes can be added as an aid for releasing more information without jeopardizing security. Because the tables of the lattice are logical rather than physical structures, the lattice serves only as a tool for estimating disclosure risks and developing security controls.

Recently, efforts have been made to develop a data model for statistical databases that models the semantic concepts of statistical applications, namely clustering attributes and table structures. An example is SUBJECT [49,50], which represents the attributes and tables of the lattice with a graph. A graph has two types of nodes: "clustering" nodes (labeled "C") for attributes (primary or secondary), and "cross product" nodes (labeled "×") for tables over multiple attributes. In addition, special cluster nodes, called "subject nodes", group attributes, tables, and other subject nodes. Figure 6 shows a SUBJECT graph for the database depicted by the lattice of Figure 3. Attributes A and B are represented by clustering nodes; the 2-dimensional tables over A and B by a cross-product node. The SUBJECT graph has the advantage of showing the hierarchical dependency relationships between clusters and values in an attribute hierarchy.

A SUBJECT graph can be mapped into the lattice model for the purpose of developing security controls. Consider the SUBJECT graph in Figure 7, which corresponds to that in Figure 4 of [49] minus the quantitative attributes (variables). The lattice for this graph is equivalent to that in Figure 2, where the attributes are interpreted as follows:

A = *States* (atomic attribute)
B = *Industrial Classes* (attribute hierarchy with 4 levels)
C = *Employment Size* (atomic attribute)

The database contains one quantitative attribute, *Reporting Units*, which is broken down by all three attributes. The raw data for this attribute is associated with table T_{AB^4C} in the lattice, and aggregations of the data are associated with the higher-level tables. The database also has two quantitative attributes, *Number of Employees* and *Taxable Level*, which are broken down by attributes A and B only. The raw data for these attributes is associated with table T_{AB^4}, and aggregations of the data are associated with the ancestors of T_{AB^4} in the lattice.

6. Conclusions

The lattice model provides a mathematical basis for studying the inference problem and its solution. It has provided a framework for estimating disclosure risk, and for evaluating and comparing different controls. It has suggested ways of adapting techniques used by census agencies to online query processing

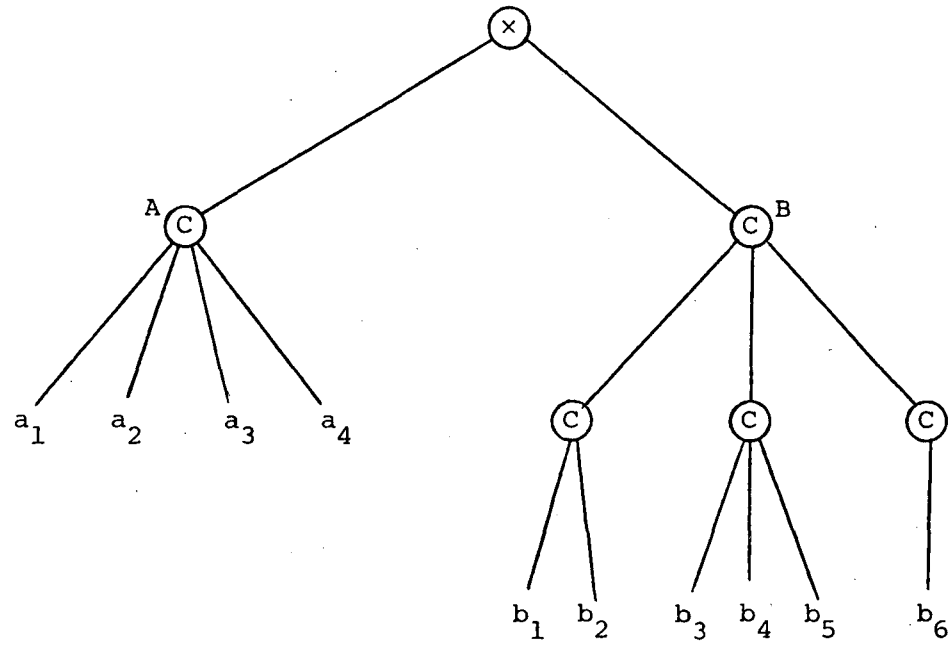


Figure 6. SUBJECT graph for lattice of Figure 3.

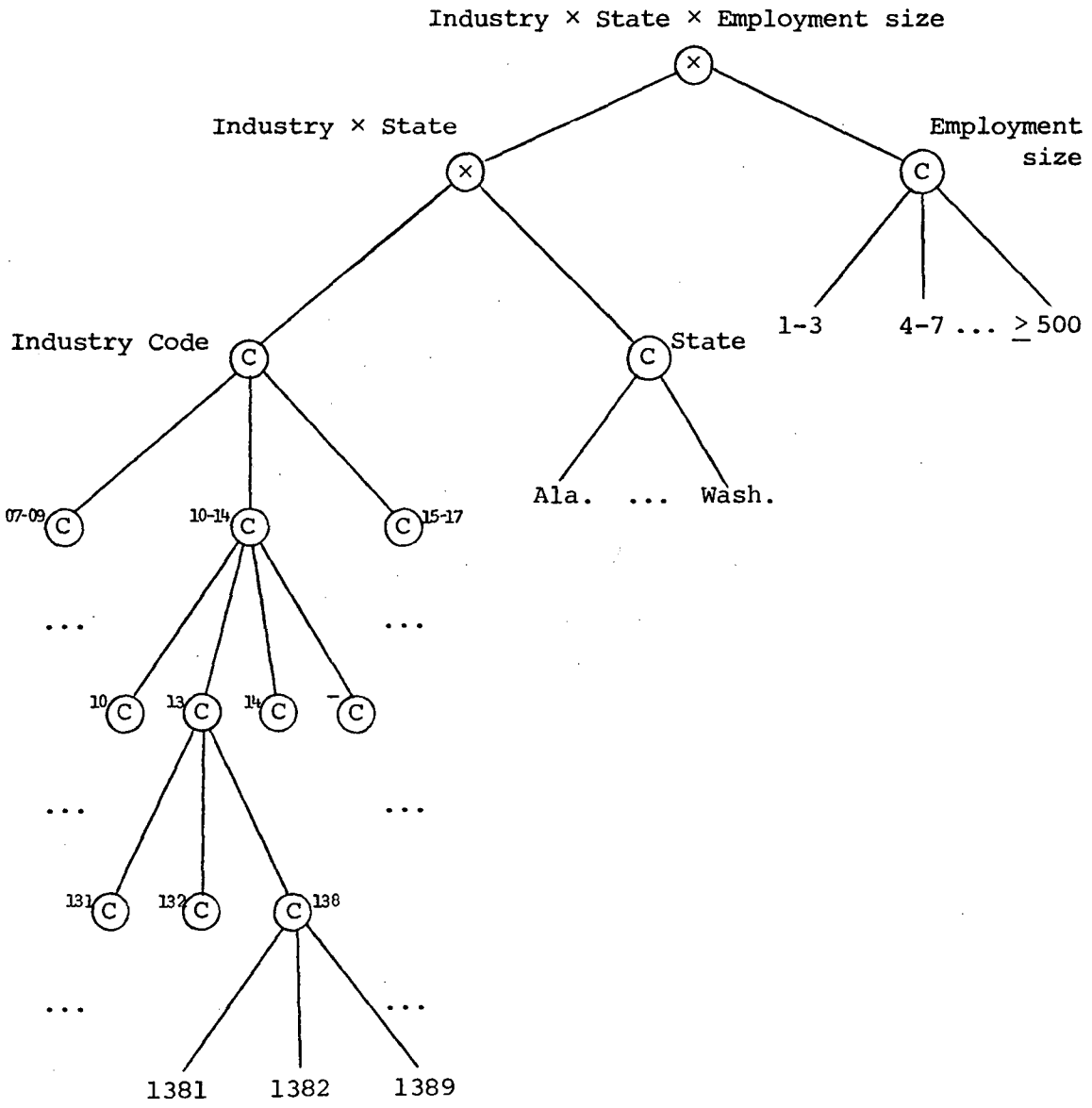


Figure 7. SUBJECT graph of industrial data.

systems, and has suggested new controls.

Although considerable progress has been made towards solving the statistical database problem in general purpose systems, further research is needed to determine more precisely the security and information loss of particular controls. Experimental studies along the lines initiated by Jan Schlörer at the University of Ulm are needed to confirm or refute the effectiveness of these controls on actual databases. Guidelines are needed for selecting the best controls for a particular application, for setting table restriction thresholds, and for selecting perturbation amounts. Additional research is also needed to understand the security problems associated with database updates [51, 52].

Acknowledgements

Many thanks to my colleagues at SRI for a stimulating discussion about the lattice model, especially Leslie Lamport who prodded me to think more about using checksums to guarantee constant perturbation for equivalent queries. Special thanks to Jan Schlörer whose insights and results have strongly influenced this paper. Jan Schlörer, Peter Denning, and Gordon Sande provided helpful comments on an earlier version of this paper.

References

1. Cox, L. H., "Suppression Methodology and Statistical Disclosure Control," *J. Amer. Stat. Assoc.* **75**(370) pp. 377-385 (June 1980).
2. Cox, L. H. and Ernst, L. R., "Controlled Rounding," U.S. Bureau of the Census, Washington, D.C. (Jan. 1981).
3. Fellegi, I. P., "On the Question of Statistical Confidentiality," *J. Amer. Stat. Assoc.* **67**(337) pp. 7-18 (Mar. 1972).
4. Fellegi, I. P. and Phillips, J. L., "Statistical Confidentiality: Some Theory and Applications to Data Dissemination," *Annals Econ. Soc'l Measurement* **3**(2) pp. 399-409 (April 1974).
5. Olsson, L., "Protection of Output and Stored Data in Statistical Databases," ADB-Information, 4, Statistika Centralbyran, Stockholm, Sweden (1975).
6. Rapaport, E. and Sundgren, B., "Output Protection in Statistical Databases," S/SYS-E04, Nat. Central Bur. Stat., Stockholm, Sweden (1975). (Invited paper, Warsaw Meeting Int. Stat. Inst., Oct. 1975)
7. Sande, G., "Towards Automated Disclosure Analysis for Establishment Based Statistics," Statistics Canada (1977).
8. Cox, L. H., "Suppression Methodology and Statistical Disclosure Control," Confidentiality in Surveys, Report No. 26, Dept. of Statistics, Univ. of Stockholm, Stockholm, Sweden (Jan. 1978).
9. Denning, D. E., "Complexity Results Relating to Statistical Confidentiality," *Computer Science and Statistics: 12th Annual Symp. on the Interface*, (May 1979).
10. Denning, D. E., Schlörer, J., and Wehrle, E., "Memoryless Inference Controls for Statistical Databases," Computer Sciences Dept., Purdue Univ. (1982).
11. Kam, J. B. and Ullman, J. D., "A Model of Statistical Databases and their Security," *ACM Trans. on Database Syst.* **2**(1) pp. 1-10 (Mar. 1977).

32. Reiss, S. P., Post, M., and Dalenius, T., "Non-Reversible Privacy Transformations," Technical Report PRT 2/4, Dept. of Computer Science, Brown Univ., Providence, R.I. (Nov. 1981).

33. Schlörer, J., "Query Based Output Perturbations to Protect Statistical Databases," Klinische Dokumentation, Universität Ulm, Ulm, W. Germany (October 1982).

34. Denning, D. E., "Secure Statistical Databases Under Random Sample Queries," *ACM Trans. on Database Syst.* **5**(3) pp. 291-315 (Sept. 1980).

35. Denning, D. E., "The Many-Time Pad: Theme and Variations," *Proc. 1983 Symp. on Security and Privacy*, (April 1983).

36. Achugbue, J. O. and Chin, F. Y., "The Effectiveness of Output Modification by Rounding for Protection of Statistical Databases," *INFOR* **17**(3) pp. 209-218 (Mar. 1979).

37. Nargundkar, M. S. and Saveland, W., "Random Rounding to Prevent Statistical Disclosure," *Proc. Amer. Stat. Assoc., Soc. Stat. Sec.*, pp. 382-385 (1972).

38. Schlörer, J., "Confidentiality and Security in Statistical Data Banks," pp. 101-123 in *Data Documentation: Some Principles and Applications in Science and Industry; Proc. Workshop on Data Documentation*, ed. W. Guas and R. Henzler,Verlag Dokumentation, Munich, Germany (1977).

39. Causey, B., "Approaches to Statistical Disclosure," in *Proc. Amer. Stat. Assoc., Soc. stat. Sec.*, , Washington, D. C. (1979).

40. Dalenius, T., "A Simple Procedure for Controlled Rounding," *Statistisk Tidskrift*, (3) pp. 202-208 (1981).

41. Fellegi, I. P., "Controlled Random Rounding," *Survey Methodology* 1(2) pp. 123-133 (1975). Statistics Canada

42. Newman, D., "Techniques for Ensuring the Confidentiality of Census Information in Great Britain," Office of Population Censuses and Surveys, Great Britain (Sept. 1975). Presented at the 2nd Meeting of the Int'l Assoc. of Survey Statisticians, Warsaw

43. Schwartz, M. D., Denning, D. E., and Denning, P. J., "Linear Queries in Statistical Databases," *ACM Trans. on Database Syst.* **4**(1) pp. 476-482 (Mar. 1979).

44. DeMillo, R. A., Dobkin, D., and Lipton, R. J., "Even Databases That Lie Can Be Compromised," *IEEE Trans. on Software Eng.* **SE-4**(1) pp. 73-75 (Jan. 1978).

45. DeMillo, R. and Dobkin, D. P., "Combinatorial Inference," pp. 27-35 in *Foundations of Secure Computation*, Academic Press, New York (1978).

46. Reiss, S. P., "Medians and Database Security," pp. 57-92 in *Foundations of Secure Computation*, ed. R. A. DeMillo et al.,Academic Press, New York (1978).

47. Codd, E. F., "A Relational Model for Large Shared Data Banks," *Comm. ACM* **13**(6) pp. 377-387 (1970).

48. Ullman, J. D., *Principles of Database Systems*, Computer Science Press (1980).

49. Shoshani, A., "Statistical Databases: Characteristics, Problems, and Some Solutions," *Proc. Eighth Int'l Conf. on Very Large Data Bases*, pp. 208-222 (Sept. 1982).

50. Chan, P. and Shoshani, A., "A Directory Driven System for Organizing and Accessing Large Statistical Databases," *Proc. Int'l Conf. on Very Large Data Bases*, pp. 553-563 (1981).

51. Chin, F. Y. and Ozsoyoglu, G., "Security in Partitioned Dynamic Statistical Databases," pp. 594-601 in *Proc. IEEE COMPSAC Conf.*, (1979).

52. Ozsoyoglu, G. and Ozsoyoglu, M., "Update Handling Techniques in Statistical Databases," in *Proc. First LBL Workshop on Statistical Database Management*, Lawrence Berkeley Lab, Berkeley, CA (Dec. 1981).

STATISTICAL DATABASES: THEIR MODEL, QUERY LANGUAGE AND SECURITY

Zbigniew MICHALEWICZ
Department of Information Science, Victoria University of Wellington
Wellington, Private Bag, New Zealand

Abstract.
In this paper we deal with the problem of security of statistical databases, i.e. file systems. We propose a model of a statistical database in which to investigate the properties of statistical databases and we describe a query language connected with such a database. We discuss the problem of dependencies between attributes and we consider the case when database contains incomplete information. In the case of incomplete information the problems arrive with the interpretation of the query language, mainly for statistical terms. The need for a precise semantic is here evident.

1. INTRODUCTION

There are a number of fundamental distinctions between data bases and file systems. In particular a data base has more structural complexity, different access methods and contains intrarecord relationships. The term "data base" usually refers to both file systems and data bases, for example a statistical database is a file system containing records for some number of individuals. A user generally has access to an external database, which is a view of part of the conceptual database. Such a view is often simply collection of records, e.g. one relation scheme, and may be seen as a file system. For example, with a database containing information about flights and passengers, a clerk may need to know about flight numbers but not about the assignment of pilots to flights. The dispatcher may need to know about flights and aircraft, but does not need to know about personnel salaries (cf. [51]).

This paper is connected with the problem of inference control. The problem of inference control in statistical databases has been of growing concern in recent years; several studies have been reported (cf. [1],[3-7],[11], [13-14],[16],[18-23],[30],[37-43],[45-50],[55]) involving different models and different allowable statistical queries. Inference controls protect statistical databases by preventing questioners from deducing confidential information by posing carefully designed sequences of statistical queries and correlating the responses. In general all protection policies impose some restrictions on the database system. A "good" protection policy should be effective (it should provide security to a reasonable extent), feasible (there should exist a way to enforce restrictions) and efficient and at the same time maintain the richness of the information revealed to users of the database (cf. [4]).

There are two main restrictions connected with the previous studies. Almost all previous studies considered static databases (except [4],[42-43],[55]) in order to simplify the problem. On the other hand the inference control in statistical databases should also be investigated for dynamic databases. The second restriction is more serious: all researchers (as far as the author is aware) studied the case of a statistical database which contain "complete" information, it is, for every object and every attribute (property) there exist a unique value which correspondes to them. Note that the situation when data are incomplete is quite common mainly in statistical databases which should contain a lot of information about large groups of population.

Now we propose and investigate a model for statistical database. For this purpose we use the model proposed by Lipski [32]; in the next section we summarize the basic notions used in this model.

2. A MODEL OF A STATISTICAL DATABASE

We give below a mathematical model of a statistical database. The model will then be used in the rest of the paper.

Def.1.

A statistical database is a quadruple

$S = \langle X, A, (Q_a)_{a\in A}, U\rangle$, where

X is a finite set of objects,

A is a finite set of attributes,

Q_a is a nonempty set called the domain of attribute a,

U is a function $U : Q \longrightarrow \mathbb{P}(X)$,

(where Q denotes the "disjoint union" of attribute domains: $Q = \{\langle a,q\rangle : a\in A \text{ and } q\in Q_a\}$, and $\mathbb{P}(X)$ denotes the set of all subsets of X), such that for every $a\in A$:

(1) $\bigcup\{U(a,q) : q\in Q_a\} = X.$

The function U associates with any $a\in A$ and $q\in Q$ a set of objects $U(a,q)\subseteq X$.

Let us consider the following example. Suppose that a statistical database contains the following information: the age of the object x is 30, the age of the object y is 28, 29 or 30 and the age of object z is 30 or 31. Now we want to list all objects which age is 30 (it is, we want to list all objects which age is 30 in reality). Note that the answer may be $\{x\}$, $\{x,y\}$, $\{x,z\}$ or $\{x,y,z\}$. Of course, the information contained in the system is not sufficient to exactly determine this set. However, for any query we may consider the following two bounds of interpretation of the query:

(2) the set of objects for which we can conclude from the information available in the system, that they must satisfy condition expressed by the query,

(3) the set of objects for which we cannot rule out the possibility of satisfying condition expressed by the query.

In other words, (2) and (3) are the best possible bounds of the interpretation of the query, logically derivable from the system. In our example the answer under the first interpretation is $\{x\}$ and under the second interpretation is $\{x,y,z\}$.

Now we turn to the function U. Intentionally, U(a,q) is the set of objects for which attribute a possibly takes value q. From the function U we can determine the set, denoted by u(a,q), of all the objects for which the value of attribute a is known to be q:

(4) $u(a,q) = X\setminus\bigcup\{U(a,s) : s\in Q_a \text{ and } s \neq q\}.$

The intuition connected with the above rule is that we know that a takes value q exactly when we know, that it is not possible for it to take any other value $s\in Q_a\setminus\{q\}$. From (1) and (4) we may easily obtain the following two intuitively evident facts:

(5) $u(a,q) \subseteq U(a,q)$

(6) $u(a,q)\cap U(a,s) = \emptyset$ for all $q \neq s$, $q,s\in Q_a$.

According to the interpretation of function U, we may determine, for every $x\in X$ and every $a\in A$, the set

(7) $\beta_a(x) = \{q\in Q_a : x\in U(a,q)\}$

of all possible values attribute a can take for object x. Conversely, U can be obtained from functions β_a ($a\in A$) by the formula:

(8) $U(a,q) = \{x\in X : q\in\beta_a(x)\}$.

We shall call $(\beta_a)_{a\in A}$ the classification associated with system $S = \langle X, A, (Q_a)_{a\in A}, U\rangle$.

3. A QUERY LANGUAGE

Below we describe the main characteristic of a query language.

By a query we shall mean a term. A term can be descriptive, numerical or statistical. A descriptive term is built up from certain elementary parts called descriptors and symbols for Boolean operations $\mathbb{0}$, $\mathbb{1}$, $\sim$, $\cdot$, $+$. The set T of descriptive terms is defined to be the least set T_1 with the following two properties:

(i) $\mathbb{0}$, $\mathbb{1}$ and all descriptors are in T_1,

(ii) $\sim t$, $(t\cdot s)$, $(t+s)$ are in T_1 whenever $t,s\in T_1$.

Every descriptor is of the form $\langle a,B\rangle$, where a is an attribute and B is the subset of attribute domain Q_a. Descriptor $\langle a,B\rangle$ denotes the set of objects for which the value of attribute a is in B. We treat descriptors as indecomposable elements without any internal structure.

For every descriptive term t, a numerical term #t can be constructed (cf. [29], [33]). It denotes the number of objects with the property expressed by t.

The set of numerical terms $T_{\#}$ is defined as

$$T_{\#} = \{\#t : t \in T\}.$$

A statistical term is given by the triple $\langle Y,f,a\rangle$, where Y is subset of X, f indicates a "statistical" function which associates a real number with any finite collection (with repetition allowed) of reals, and a is an attribute (not always arbitrary). A statistical term $\langle Y,f,a\rangle$ is intended to denote either the specific value from the set $\{\beta_a(x) : x\in Y\}$ with repetitions allowed, when f is maximum, minimum or medium, or a value computed using the values $\beta_a(x)$, $x\in Y$.

Note that the set of attributes can be partitioned into two groups, $A = D \cup W$, where $a\in W$ if $Q_a \subseteq R$ and $a\in D$ otherwise (R denote the set of real numbers). It means that we may arithmetically add the values $q_1,q_2\in Q_a$ only in the case when $a\in W$. In many cases in modern statistical databases (cf. [3],[13],[15],[17], [30]) the set of attributes A is divided onto two subsets $A = C \cup V$, where C is the set of categories and V - the set of data. These sets need not be disjoint. Categories are used to distinguish the subsets of objects having common characteristic (the values of attributes), and data keep their numerical values. Of course $V \subseteq W$, ie. for all $a\in V$: $Q_a \subseteq R$. Since categories describe characteristics of objects, the descriptors, which are used to build the descriptive terms are of the form $\langle a,B\rangle$, where $a\in C$ (and consequently $B \subseteq Q_a$). On the other hand the requirement for statistical terms $\langle Y,f,a\rangle$ is $a\in V$.

Now we give a definition related to descriptive terms.

Def.2.

(i) A descriptive term is primitive if it is of the form

$$\prod_{j\in J}\langle a_j,B_j\rangle$$

where $a_j \neq a_k$ for $j\neq k$ and $\emptyset \neq B_j \subsetneq Q_{a_j}$ for all $j\in J$

(ii) A descriptive term is in additive normal form if it is of the form

$$\sum_{k\in K} t_k$$

where all t_k's are primitive

(iii) A descriptive term is coprimitive if it is of the form

$$\sum_{j\in J}\langle a_j,B_j\rangle$$

where $a_j \neq a_k$ for $j\neq k$ and $\emptyset \neq B_j \subsetneq Q_{a_j}$ for all $j\in J$

(iv) A descriptive term is in multiplicative normal form if it is of the form

$$\prod_{k\in K} t_k$$

where all t_k's are coprimitive

(v) A descriptive (primitive) term is simple if it is of the form

$$\prod_{a\in C}\langle a,\{q_a\}\rangle$$

(vi) A descriptive term is in standard form if it is of the form

$$\sum_{k\in K} t_k$$

where all t_k's are simple.

4. STATISTICAL DATABASE - COMPLETE INFORMATION

We say that a statistical database contains complete information if and only if $u = U$. Note that this condition is equivalent to "$\beta_a(x)$ consists of a single element of Q_a, for all $a\in A$ and $x\in X$". In a complete system we know exactly the unique value attribute a takes for object x for any $a\in A$ and $x\in X$. Although the semantic connected with complete system is intuitively evident and is "the only natural one", nevertheless we give below a formal definition of the interpretation of (ie. the response to) a query.

Def.3.

The value of a query q in system S is denoted by $\|q\|_S$ (or $\|q\|$) and defined inductively as follows:

(i) $\|\mathbb{0}\| = \emptyset$, $\|\mathbb{1}\| = X$,

(ii) $\|\langle a,B\rangle\| = \bigcup_{q\in B} U(a,q)$,

(iii) $\|\sim t\| = X \setminus \|t\|$,

(iv) $\|t \cdot s\| = \|t\| \cap \|s\|$,

(v) $\|t + s\| = \|t\| \cup \|s\|$,

(vi) $\|\#t\| = \mathrm{card}(\|t\|)$,

(vii) $\|\langle Y,f,a\rangle\| = f(\beta_a(x_1),\ldots,\beta_a(x_m))$, where $Y = \{x_1,\ldots,x_m\}$.

Note that the set Y in (vii) may be given as a value $\|t\|$ for some descriptive term t; than Y is the set of objects satisfying property expressed by t. In that case such a statistical term we denote by $\langle t,f,a\rangle$.

Example 1. A statistical database containing information on employees.

Objects	Categories			Data	
	Sex	Dept.	Position	#Child.	Salary
1.Adams	M	Math	Prof	0	24000
2.Baker	M	Biol	Prof	4	24000
3.Cook	M	Biol	Stu	1	5000
4.Dodd	F	Psy	Asist	1	17000
5.Engel	F	Math	Stu	1	16000
6.Flood	M	Math	Prof	2	22000
7.Grady	F	Psy	Stu	3	9000
8.Hayes	M	Math	Prof	3	21000
9.Iron	M	Biol	Stu	3	10000
10.Jones	M	Math	Asist	0	16000
11.Knapp	F	Biol	Asist	2	17000
12.Lord	F	Biol	Asist	2	15000

In the above statistical database we have 12 objects (X = {Adams, Baker,...,Lord}), which are described by 5 attributes (A = {Sex,Dept., Position, #Child., Salary}). The categories describe characteristics of the objects (sex, position, department), where $Q_{Sex} = \{F,M\}$, $Q_{Dept.} = \{Math, Biol, Psy\}$, and $Q_{Position} = \{Prof, Asist, Stu\}$; whereas data (number of children and salary) specify numerical values for these objects. Below we give three simple queries and responses for them.

$q_1 = \langle Sex,\{M\}\rangle \cdot \langle Dept,\{Math\}\rangle + \langle Posit,\{Stu\}\rangle$,
$\|q_1\| = \{Adams, Cook, Engel, Flood, Grady, Hayes, Iron, Jones\}$,

$q_2 = \#(\langle Sex,\{M\}\rangle \cdot \langle Dept,\{Math\}\rangle + \langle Posit,\{Stu\}\rangle)$,
$\|q_2\| = 8$,

$q_3 = \langle\langle Dept,\{Math\}\rangle,MAX,Salary\rangle$,
$\|q_3\| = 24000$.

Now we give a definition and a theorem which will be important in the following text.

Def.4.

Two descriptive terms $t,s \in T$ are equivalent ($t \approx s$) if for all systems S: $\|t\|_S = \|s\|_S$ (the phrase "for all systems" refers to all systems with fixed A and $(Q_a)_{a \in A}$).

Theorem 1.

(a) For each descriptive term t there is a term s in normal additive form such that $t \approx s$.

(b) For each descriptive term t there is a term s in standard form such that $t \approx s$.

(c) For each descriptive term t there is a term s in multiplicative form such that $t \approx s$.

For the proof the reader is refered to [32].

We say that a compromise occurs when a user deduces, from responses to one or more queries, confidential information of which he was previously unaware (in such case we say that a database is compromisable). A database is secure, if it is not compromisable.

In general, a user is not allowed to know some values in the database. Thus interpretation of a query for a user is usually different from interpretation for the administrator of the system. During the last years there has been a great effort to find a secure interpretation, under which a database is secure. There have been two main approaches to this provision of security. First, the administrator of the system may restrict the class of admissible queries. This restriction may be done in a syntactic way, for example an admissible query may use only a primitive terms (Def.2(i)), cf. [3], [18], [30], or a user must not use particular attributes. Alternatively, the restrictions may be done in a semantic way, for example the answer to a query may depend on the number of objects involved in this query, cf. [16]. The second method of providing database security is by giving an "inprecise" answer to a query (cf.

[1], [14], [18], [20]).

One of the most promising is the concept of defining "statistical information". This correspondes to the definition of sets of population (cf.[4]). We may apply this idea by using our model. Suppose that in our model there are p category attributes and that the i-th category attribute can take n_i values $v_{i1},\ldots,v_{in_i}$. We may form $N = \prod_{i=1}^{p} n_i$ elementary conjunctions, each of them being a conjunction of a different combination of category values. Each elementary conjunction correspondes to some simple term (see Def.2(v)).

Example 3. Consider the database shown in Example 1. It contains 12 records and the set of elementary conjunctions (simple terms) having $\|t\| \neq \emptyset$ is $\{t_1,\ldots,t_8\}$, where

t_1=⟨Sex,{M}⟩·⟨Dept,{Math}⟩·⟨Posit,{Prof}⟩
t_2=⟨Sex,{M}⟩·⟨Dept,{Math}⟩·⟨Posit,{Asist}⟩
t_3=⟨Sex,{M}⟩·⟨Dept,{Biol}⟩·⟨Posit,{Prof}⟩
t_4=⟨Sex,{M}⟩·⟨Dept,{Biol}⟩·⟨Posit,{Stu}⟩
t_5=⟨Sex,{F}⟩·⟨Dept,{Math}⟩·⟨Posit,{Stu}⟩
t_6=⟨Sex,{F}⟩·⟨Dept,{Biol}⟩·⟨Posit,{Asist}⟩
t_7=⟨Sex,{F}⟩·⟨Dept,{Psy}⟩·⟨Posit,{Asist}⟩
t_8=⟨Sex,{F}⟩·⟨Dept,{Psy}⟩·⟨Posit,{Stu}⟩

Under the assumption, that the cardinalities of domains for attributes Sex, Dept. and Position are 2, 3 and 3 respectively, the total number of elementary conjunctions is 18.

Def.5.
We say that $Y \subseteq X$ is describable in S if there is a descriptive term t such that Y is the set of all objects satisfying the property expressed by t ($Y = \|t\|$).

If we restrict the set of allowable queries to queries specified by descriptive terms, a user is allowed to ask for properties of any describable set (not every possible combination of records can be requested). However, such a restriction alone is still ineffective (cf.[16]). So the task of the administrator of the database is to partition records into disjoint groups, which define atomic populations. Note that every atomic population should be one of the describable sets, so the set of atomic populations may be given as a set of descriptive terms.

Def.6.
We say that the set $\{Y_1,\ldots,Y_r\}$ of describable sets form the set of atomic populations, if for arbitrary $i \neq j$: $Y_i \cap Y_j = \emptyset$ and $\bigcup_{i=1}^{r} Y_i = X$.

Example 4. Taking into account the previous example the administrator may define the following atomic populations as a set of descriptive terms:

s_1 = ⟨Dept,{Math}⟩·⟨Sex,{F}⟩
s_2 = ⟨Dept,{Math}⟩·⟨Sex,{M}⟩
s_3 = ⟨Dept,{Biol}⟩·⟨Posit,{Asist}⟩
s_4 = ⟨Dept,{Biol}⟩·⟨Posit,{Stu,Prof}⟩
s_5 = ⟨Dept,{Psy}⟩

Def.7.
Let $\{Y_1,\ldots,Y_r\}$ be the set of atomic populations. A set Y will form a population over $\{Y_1,\ldots,Y_r\}$, if it is a union of a certain number of atomic populations from $\{Y_1,\ldots,Y_r\}$.

Note that the set of atomic populations implies the set of populations. A user is only allowed to learn about populations. For example, a user may learn about the maximum salary of all objects, which satisfy the following condition:

⟨Dept,{Math}⟩·⟨Sex,{M}⟩ + ⟨Dept,{Biol}⟩·
·⟨Posit,{Asist}⟩ = $s_2 + s_3$ (see Example 4).

A formal definition of the interpretation for a user is:

Def.8.
Let $\{Y_1,\ldots,Y_r\}$ be the set of atomic populations, fixed for a given database S.

(a) The value of a numerical term #t for a user is: $\|\#t\| = \mathrm{card}(Y)$, where Y is the least population over $\{Y_1,\ldots,Y_r\}$ such that $\|t\| \subseteq Y$.

(b) The value of a statistical term ⟨t,f,a⟩ for a user is $\|\langle t,f,a\rangle\| = \|\langle Y,f,a\rangle\|$, where Y is the least population over $\{Y_1,\ldots,Y_r\}$ such that $\|t\| \subseteq Y$.

This protection policy may be easily implemented. Each query, numerical or statistical term is specified by a descriptive term, which has an equivalent

standard form (Theorem 1). Each atomic population may be described by a descriptive term in the standard form. An algorithm to transfer an arbitrary descriptive term t into an equivalent standard form is given [32].

To maintain the richness of the information revealed to users, the administrator may fix different atomic populations for different attributes a∈V. For every statistical term $\langle t,f,a\rangle$ we must take into account the set of atomic populations which correspondes to attribute a. However the administrator of the database must be aware that there may be some relationships between the attributes from the set V. More precisely:

Def.9.

Let $S = \langle X, A, (Q_a)_{a\in A}, U\rangle$ be a statistical database ($A=C\cup V$). Then $\underset{a}{\sim}$ is a binary relation defined by $x \underset{a}{\sim} y$ if $\beta_a(x) = \beta_a(y)$.

It is easily seen that $\underset{a}{\sim}$ is equivalence relation. Let a,b ∈ V be two attributes. Attribute b is dependent on a ($a \longrightarrow b$) if $\underset{a}{\sim} \subseteq \underset{b}{\sim}$; a and b are equivalent ($a \approx b$) if $\underset{a}{\sim} = \underset{b}{\sim}$.

For example, there may be some relationship between attributes "Tax" and "Salary" (ie. $\beta_{Sal}(x) < \beta_{Sal}(y)$ implies $\beta_{Tax}(x) < \beta_{Tax}(y)$; moreover, the opposite implication is also true). In that case the attributes "Tax" and "Salary" are equivalent.

Let us denote $\underset{B}{\sim} = \bigodot_{a_i\in B} \underset{a_i}{\sim}$ ($B \subseteq V$), where $\odot$ is a product of partitions (equivalence relations) $\underset{a_i}{\sim}$ defined in a usual way. Obviously, $\underset{B}{\sim}$ is equivalence relation.

Now we give two definitions (cf.[44]):

Def.10.

The set V of attributes in S is called independent in S if for every $V_1 \subseteq V$: $\underset{V_1}{\sim} \supsetneq \underset{V}{\sim}$.

If there is a subset $V_1 \subseteq V$ such that $\underset{V_1}{\sim} = \underset{V}{\sim}$ then the set of attributes V will be called dependent in S.

Def.11.

The smallest set $V_1 \subseteq V$ such that V_1 is independent in $S = \langle X, A, (Q_a)_{a\in A}, U\rangle$ ($A=C\cup V$) will be called reduct of V and the corresponding database $S_1 = \langle X, A_1, (Q_a)_{a\in A_1}, U_1\rangle$ ($A_1=C\cup V_1$) - reduced database (U_1 is the restriction of the function U to the set A_1).

Under the above definition it is clear that the administrator should fix atomic populations for reduced database, ie. for all attributes which belong to the reduct of V. The set of atomic populations for the attributes from the reduct of V imply the other sets of atomic populations for other attributes.

If $a \longrightarrow b$ for some a,b∈V and $\{Y_1,\ldots,Y_r\}$ is the set of atomic populations for attribute a, then for every atomic population Z for attribute b: $Z = \bigcup_{i\in J} Y_i$, where $J \subseteq \{1,\ldots,r\}$.

Therefore equivalent attributes should have the same sets of atomic populations. For example, having $Tax \approx Salary$, the set of atomic populations for attributes Tax and Salary should be the same.

5. STATISTICAL DATABASES - INCOMPLETE INFORMATION

As we point out in the previous section, the semantic connected with the query language for formulating queries to a statistical database with complete information is intuitively evident and is "the only natural one". It is no longer so when the information is incomplete. To give a precise notation of interpretation of a query language, we introduce the following definition (cf.[32]):

Def.12.

Let $S_1 = \langle X, A, (Q_a)_{a\in A}, U_1\rangle$ and $S_2 = \langle X, A, (Q_a)_{a\in A}, U_2\rangle$ be two statistical databases. We say that S_1 is an extention of S_2 (and denote $S_2 \leq S_1$) if $U_1(a,q) \subseteq U_2(a,q)$ for all a∈A and $q\in Q_a$.

It is obvious that $\leq$ is a partial order on the set of all systems with fixed X and $(Q_a)_{a\in A}$. This partial order has the least element (ie. where $U(a,q) = X$ for all a∈A and $q\in Q_a$). Such a database contains

no information at all, except for the mere fact what attributes refer to the objects. Using the above definition we may define a complete statistical database as a maximal element in the order $\lesssim$, ie. if it has no extentions $S \lesssim S_1$ except for $S = S_1$. A statistical database is called incomplete if it is not complete.

Now we are ready to turn to interpretation of a query language in incomplete statistical databases. Taking into account the considerations in Section 2, we give two bounds of interpretation of a query. Note that objects in incomplete database S are in reality described by a completion of S (we do not know by which one). So we may conclude that x has in reality property t only if x has property t in every completion of S. This leads in natural way to the following definition:

Def.13.

(i) The lower value of a descriptive term t in S is $\|t\|_{*S} = \bigcap_{\substack{S \lesssim S_1 \\ S_1 \text{ complete}}} \|t\|_{S_1}$

(ii) The upper value of a descriptive term t in S is $\|t\|^*_S = \bigcup_{\substack{S \lesssim S_1 \\ S_1 \text{ complete}}} \|t\|_{S_1}$

In other words, the lower value of a descriptive term t in S is a subset Y of X such that $x \in Y$ iff for every completion S_1 of S: $x \in \|t\|_{S_1}$. The upper value of descriptive term t in S is a subset Y of X such that $x \in Y$ iff for some completion S_1 of S: $x \in \|t\|_{S_1}$. Sometimes, when S is clear from the context, we write simply $\|t\|_*$ or $\|t\|^*$ instead of $\|t\|_{*S}$ or $\|t\|^*_S$.

The following theorem gives the basic properties of $\|\cdot\|_*$ and $\|\cdot\|^*$.

Theorem 2.

In any statistical database $S = \langle X, A, (\Omega_a)_{a\in A}, U\rangle$ we have

(i) $\|\mathcal{O}\|_* = \|\mathcal{O}\|^* = \emptyset$

(ii) $\|\mathcal{I}\|_* = \|\mathcal{I}\|^* = X$

(iii) $\|\langle a,B\rangle\|_* = X \setminus \bigcup_{q\in Q_a \setminus B} U(a,q)$

(iv) $\|\langle a,B\rangle\|^* = \bigcup_{q\in B} U(a,q)$

(v) $\|\sim t\|_* = X \setminus \|t\|^*$

(vi) $\|\sim t\|^* = X \setminus \|t\|_*$

(vii) $\|t + s\|_* \supseteq \|t\|_* \cup \|s\|_*$

(viii) $\|t + s\|^* = \|t\|^* \cup \|s\|^*$

(ix) $\|t\cdot s\|_* = \|t\|_* \cap \|s\|_*$

(x) $\|t\cdot s\|^* = \|t\|^* \cap \|s\|^*$

For the proof the reader is refered to [32].

Note that the inclusion (vii) cannot be replaced by equality, since $\|t + s\|_*$ may contain objects know to have property t or property s (and it is not known which one).

However, two basic kinds of queries to a statistical database are numerical and statistical terms. We may extend easily the interpretation of query language to capture numerical terms:

Def.14.

(i) The lower value of a numerical term #t in S is $\|\#t\|_* = \text{card}\,\|t\|_*$

(ii) The upper value of a numerical term #t in S is $\|\#t\|^* = \text{card}\,\|t\|^*$

In [29] the authors describe some propreties of $\|\cdot\|_*$ and $\|\cdot\|^*$ for numerical terms; for more details the reader is refered to this work.

More serious problems arrive with statistical terms. Recall that the set of attributes is usually divided into two (not necessarily disjoint) sets named categories and data. Both categories and data may contain incomplete information. The need for a precise semantic is hehe evident. For instance, what should be the response to the query "give sum of salaries of all objects which age is 30"? Should we take into account only the objects which are known to be to be 30 and have the unique value for attribute "Salary"? Can we rule out all objects for which there is a possibility to be 30? What about objects for which age is 30 and the value of attribute "Salary" is unknown or, at least, given in some range? Below we propose the inter-

pretation for the statistical term:

Def.15.

(i) The lower value of a statistical term $\langle t,f,a\rangle$ in S is a range $\|\langle t,f,a\rangle\|_* = [p,r]$, such that

$p = f(\inf(\beta_a(x_1)),\ldots,\inf(\beta_a(x_k)))$

$r = f(\sup(\beta_a(x_1)),\ldots,\sup(\beta_a(x_k)))$

where $\|t\|_* = \{x_1,\ldots,x_k\}$,

(ii) The upper value of a statistical term $\langle t,f,a\rangle$ in S is a range $\|\langle t,f,a\rangle\|^* = [p,r]$, such that p and r have the same meaning as in (i) and $\|t\|^* = \{x_1,\ldots,x_k\}$.

Note that there exist several other possible interpretations of a statistical term. For example the query $\langle t,sum,salary\rangle$ may be interpreted as a sum of salaries of all objects which are known to satisfy the property expressed by t and, in the same time, have a unique value for attribute "Salary", increased by a sum of an average salaries of all objects which are known to satisfy the property t and have not a unique value for attribute "Salary" (similarly for all objects for which the possibility of satisfying property t cannot be ruled out).
We choose a range as an interpretation of a statistical term, since it provides the user with bounds for the true values. The true value (perhaps unknown in the database) can be trusted to lie between these bounds.

Example 5. Let us consider the same set of objects and attributes as in Example 1.
We assume that

$\beta_{Sex}(Cook) = \beta_{Sex}(Engel) = \{F,M\}$,

$\beta_{Dept}(Baker) = \{Biol, Psy\}$,

$\beta_{Dept}(Hayes) = \{Math, Biol\}$,

$\beta_{Salary}(Baker) = \{22000,23000,24000,25000\}$,

$\beta_{Salary}(Iron) = \{8000,9000,10000,11000\}$,

and the rest of values remain the same as in Example 1.
Then for the query:

$t = \langle Dept,\{Biol\}\rangle$, we have

$\|t\|^* = \{Baker,Cook, Hayes,Iron,Knapp,Lord\}$,

whereas

$\|t\|_* = \{Cook,Iron,Knapp,Lord\}$.

For a numerical term $\#q = \#\langle Dept,\{Biol\}\rangle$, we have

$\|\#q\|^* = 6$ and $\|\#q\|_* = 4$,

and for statistical term $s = \langle Dept,\{Biol\} ,SUM,Sal\rangle$ we have

$\|s\|^* = [88000,94000]$,

$\|s\|_* = [45000,48000]$.

The output of ranges to protect confidential information released as statistics about groups of individuals was proposed previously (cf. [43], [49]). However the meaning of those ranges is quite different to these proposed in this paper. The reason is that all previous studies deal with complete information and the range (as the answer for the query) contains the true value. In the case of incomplete information the range has the following meaning: "in every completion of system S, the true value (answer for the query) is in the given range". The one point is common: the administrator of a database should fix the interpretation of a query language such that users may obtain valuable (meaningful) information and in the same time cannot infer about confidential value(s). Below we give one of the several possible interpretation of a query language for a user.

Def.16.

(i) The value of a numerical term #t for a user is the range $\|\#t\| = [\|\#t\|_*, \|\#t\|^*]$,

(ii) The value of a statistical term $\langle t,f,a\rangle$ for a user is the range $\|\langle t,f,a\rangle\| = [p,r]$, where

$\|\langle t,f,a\rangle\|_* = [p_1,r_1]$,

$\|\langle t,f,a\rangle\|^* = [p_2,r_2]$, and

$p = \min\{p_1,r_1,p_2,r_2\}$, $r = \max\{p_1,r_1,p_2,r_2\}$.

Example 6. The value for the statistical term $s = \langle Dept,\{Biol\} ,SUM,Salary\rangle$ for the user is $\|s\| = [45000,94000]$, where the database contains the same values as in Example 5.

The important point connected with above definition is that users are allowed to obtain in some sense "true" values, that is for every completion of the system S, the true value (true answer for

a query) is in the given range. On the other hand, having the range $[p,r]$ as the answer for the query $\langle t,f,a\rangle$, a user is uncertain whether interesting for him (her) object x is an element of the set $\|t\|_*$ and whether $\beta_a(x)$ is one-element set. For more details connected with inference controls in statistical databases with incomplete information, the reader is refered to [40].

In a statistical databases with complete information there is a serious threat of inferring confidential value, when an object is inserted or deleted (cf. [4],[43]), simply by querying just before and after a change. In our proposed model there are two different kinds of updates. The first is when the information about the objects increases, while the objects themselves remain invariant. The second possibility of update is to insert (delete) some object(s) (the modification of a value for existing object we may treat as a sequence of of delete-insert operations). To improve security in our system we may assume that every inserted object x has "large" sets $\beta_a(x)$ for all attributes $a\in A$ (even if the values of this object for some attributes are known to the administrator) and then, gradually, the administrator increase information about this object. There is no more need to process changes in pairs (cf. [4]). We may consider similar approach when the object x is to be deleted. Before this operation we may enlarge the sets $\beta_a(x)$ (for $a\in A$) and then delete object x "safely". Thus statistical databases with incomplete information handle with updates much easier then when information is complete.

6. CONCLUSIONS

As we have mentioned in Introduction, the model of a statistical database with incomplete information was taken from [32]. In that paper the author distinguish two different ways of interpreting a query - the external one and the internal one. The external interpretation refers the queries directly to the real world modelled (in incomplete way) by the system, whereas under the internal interpretation the queries refer to the systems's information about this world rather than to the world itself. In this paper we consider only external interpretation as a tool of preventing a user to infer confidential value(s).

This approach was based on the partial information on the value of an attribute. Note that in the other possible approach to incomplete information based on null values, we are restricted to the two extremal cases when either everithing is known about the value of an attribute, or nothing is known about the value of an attribute.

We may consider also more general description of a statistical term $\langle t,f,W\rangle$, where W is a subset of the set of data V; for example

$$\|\langle t,f,\{a,b\}\rangle\| = \sqrt{\beta_a(x_1)\cdot\beta_b(x_1)+\ldots+\beta_a(x_k)\cdot\beta_b(x_k)}$$

in the case of complete information (where $\{x_1,\ldots,x_k\}$ is the set of objects satisfying property expressed by t). The theory presented in this paper can be extended to the language involving such statistical terms.

Another extention is to allow "binary descriptors" of the form $\langle a,R,b\rangle$, $a,b\in A$, $R\subseteq Q_a\times Q_b$, where $\|\langle a,R,b\rangle\| = \{x\in X : \beta_a(x)\times\beta_b(x) \in R\}$, in the case of complete information. Examples of such descriptors are $\langle$Expenses $\geqslant$ Salary$\rangle$, etc.

Presented model is appropriate also for modelling supplementary knowledge of a user. We may think that $S_1 \lesssim S$ for a given database S. In particular, when a user has not any supplementary knowledge, S_1 correspondes to the least element in the partial order $\lesssim$ on the set of all systems with fixed X and $(Q_a)_{a\in A}$. This knowledge is only about what attributes refer to the objects and what domain particular attribute has, and is essential to formulate queries. However, not every information is representable in a such model.

REFERENCES:

[1] Beck, L.L. "A Security Mechanism for Statistical Databases", ACMToDS, 5(3), 1980, pp.316-338,

[2] Champine, G.A. "Distributed Computer Systems", North-Holland Pub. Co. 1980,

[3] Chin, F. "Security in Statistical Databases for Queries with Small Counts", ACMToDS 3(1), 1978, pp.92-104,

[4] Chin, F. and Ozsoyoglu, G. "Statistical Database Design", ACMToDS 6(1), 1981, pp.113-139,

[5] Chin, F. and Kossowski, P. "Efficient Inference Control for Range SUM Queries on Statistical Data Bases", 1981,

[6] Chin, F. and Kossowski, P. "The Theory of Secure Policies for Inference Control by Auditing in Statistical Databases", 1981 (ext. abstract),

[7] Chin, F. and Kossowski, P. "Formal Theory of Secure Policies for Statistical Queries", (ext. abstract), 1981,

[8] Codd, E.F. "A Relational Model of Data for Large Shared Data Banks", Comm. ACM, 13(6), 1970, pp.377-387,

[9] Codd, E.F. "Extending the Database Relational Model to Capture More Meaning", ACMToDS, 4(4), 1970, pp.397-434,

[10] Date, C.J. "An Introduction to Database Systems", Addison-Wesley Pub., 1978,

[11] Davida, G.I., Linton, D.J., Szelag, C.R. and Wells, D.L. "Database Security", IEEE Trans. of S.E., SE-4(6), 1978, pp.531-533,

[12] Denning, D.E. "A Lattice Model of Secure Information Flow", Comm. of ACM, 19(5), 1976, pp.236-243,

[13] Denning, D.E. "Are Statistical Databases Secure?", Proc.AFIPS, NCC 47(1978), pp.525-530,

[14] Denning, D.E. "Secure Statistical Databases with Random Sample Queries", ACMToDS, 5(3), 1980, pp.291-315,

[15] Denning, D.E. and Denning, P.J. "Data Security", Comp. Surveys, 11(3), 1979, pp.227-249,

[16] Denning, D.E., Denning, P.J. and Schwartz, M.D. "The Tracker: A Threat to Statistical Database Security", ACMToDS, 4(1), 1979, pp.97-106,

[17] Denning, D.E. and Schlörer, J. "A Fast Procedure for Finding a Tracker in a Statistical Database Security", ACMToDS, 5(1), 1980, pp.88-102,

[18] Denning, D.E., Schlörer, J. and Wehrle, E. "Memoryless Inference Controls for Statistical Databases", Proc. 1982 Symp. on Security and Privacy (Berkeley), IEEE Com.Soc.

[19] DeMillo, R.A., Dobkin, D., Jones, A.K. and Lipton, R.J. "Foundations of Secure Computation", Academic Press, Inc., 1978,

[20] DeMillo, R.A., Dobkin, D. and Lipton, R.J. "Even Data Bases that Lie can be Compromised", IEEE Trans. of S.I., SE-4(1), 1978, pp.73-75,

[21] Dobkin, D., Jones, A.K. and Lipton, R.J., "Secure Databases: Protection against User Inference", ACMToDS, 4(1), 1979, pp.97-106,

[22] Dobkin, D., Lipton, R.J. and Reiss, S.P. "Aspects of the Database Security Problem", Proc. Conf. on Theoret. Comp. Sci., Aug. 15-17, 1977, Waterloo, Canada,

[23] Haq, M. "Insuring Individuals Privacy from Statistical Data Users", Proc.AFIPS, NCC, 1975, pp.941-946,

[24] Hartson, H.R. and Hsiao, D.K. "A Semantic Model for Data Base Protection Languages", Systems for Large Data Bases, P.C. Lockemann and E.J. Neuhold,(eds), North-Holland Pub. Co., 1976, pp.27-42.

[25] Hoffman, L.J. "Modern Methods for Computer Security and Privacy", Prentice-Hall, Englewood Cliffs, N.J. 1977,

[26] Hoffman, L.J. and Miller, W.F. "Getting a Personal Dossier from a Statistical Data Bank", Datamation, 15(5), 1970, pp.74-75,

[27] Hsiao, D.K. and Baum, R.K. "Information Secure Systems", Adv. in Comp., 14, 1976, pp.231-272,

[28] Hsiao, D.K., Kerr, D.S. and Madnick, S.E. "Privacy and Security of Data Communications and Data Bases", Proc. Int. Conf. Very Large Data Bases, 1978, pp.55-67,

[29] Jaegermann, M. and Lipski, W. "Numerical queries in incomplete information data bases", ICS PAS Reports 388, Warsaw 1980,

[30] Kam, J.B. and Ullman, J.D. "A Model of Statistical Databases and Their Security", ACMToDS, 2(1), 1977, pp.1-10,

[31] Landwehr, C.E. "Formal Models for Computer Security", Comp. Surveys, 13(3), 1981, pp.247-278,

[32] Lipski, W. "On Semantic Issues Connected with Incomplete Information Data Bases", ACMToDS, 4(3), 1979, pp.262-296,

[33] Lipski, W. and Marek, W. "Information Systems: On Queries Involving Cardinalities", Inf. Syst. 4(3-E), pp.241-246,

[34] Lipski, W. and Marek, W. "On Information Storage and Retrieval Systems", Math. Foundations of Comp. Sci., A. Mazurkiewicz and Z. Pawlak (eds), Banach Center Pubs., 2, Polish Scientific Pubs., Warsaw, 1977, pp.215-259,

[35] Marek, W. and Pawlak, Z. "Information Storage and Retrieval Systems: Mathematical Foundations", Theoret. Comp. Sci., 1, 1976, pp.331-354,

[36] Martin, J. "Security, Accuracy and Privacy in Computer Systems", Englewood Cliffs, Prentice-Hall, 1973,

[37] Michalewicz, Z. "Compromisability of a Statistical Database", Inf. Syst., 6(4), 1981, pp.301-304,

[38] Michalewicz, Z. "Security of a Statistical Database", Ph.D. dissertation, Polish Scientific Publishers, Warsaw, 1982, (in Polish),

[39] Michalewicz, Z. "A Coin-Weighing Problem and its Connection with the Security of a Statistical Database", ICS PAS Report, No.426, Warsaw, 1980,

[40] Michalewicz, Z. "Inference Controls in Statistical Databases with Incomplete Information" Inf. Syst., 8(3), 1983,

[41] Michalewicz, Z. "Improving Security in a Statistical Database", Proc. ACSC 6, Sydney, 10-12 Feb. 1983,

[42] Ozsoyoglu, G. and Chin, F. "Enhancing the Security of Statistical Databases with a Question Answering System and a Kernel Design", IEEE Trans. of S.I.

[43] Ozsoyoglu, G. and Ozsoyoglu, M. "Update Handling Techniques in Statistical Databases", 1981,

[44] Pawlak, Z. "Information Systems, ICS PAS Report 338, Warsaw 1979,

[45] Schlörer, J. "Identification and Retrieval of Personal Records from a Statistical Databank", Methods Inf. Med., 14(1), 1975, pp.7-13,

[46] Schlörer, J. "Confidentiality of Statistical Records: A Threat-Monitoring Scheme for On Line Dialogue", Methods Inf. Med., 15(1), 1976, pp.36-42,

[47] Schlörer, J. "Disclosure from Statistical Databases: Quantitative Aspects of Trackers", ACMToDS, 5(4), 1980, pp.467-492,

[48] Schlörer, J. "Security of Statistical Databases: Multidimensional Transformation", ACMToDS, 6(1), 1981, pp.95-112,

[49] Schlörer, J. "Security of Statistical Databases: Ranges and Trackers", 1981,

[50] Schwartz, M.D., Denning, D.E. and Denning, P.J. "Linear Queries in Statistical Databases", ACMToDS, 4(2), 1979, pp.156-167,

[51] Ullman, J.D. "Principles of Database Systems", Pitman, London 1980,

[52] Van Leeuwen, J. "On Compromising Statistical Data Bases with a Few Known Elements", Inf. Proc. Let., 8(3), 1979, pp.149-153,

[53] Wong, E. "A Statistical Approach to Incomplete Information in Database Systems", ACMToDS, 7(3), 1982, pp.470-488,

[54] Yao, A.C. "A Note on a Conjecture of Kam and Ullman Concerning Statistical Databases", Inf. Proc. Let., 9(1), 1979, pp.48-50,

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

9. Benchmarks and Performance Evaluation

PERFORMANCE PREDICTION METHODS FOR EVALUATING PDE ALGORITHMS ON MIMD MACHINES*

Simon K. Fok, John R. Wilson, Harry G. Heard, Joseph A. Parker

Technology Development of California, Inc.
3990 Freedom Circle, Santa Clara, California 95054

Abstract

In this paper, a variety of current performance measures for SIMD machines are extended to MIMD machines. The concept of cost-effectiveness is elaborated and used to obtain an a posteriori estimate of the optimal number of processors to be used for a particular problem. Using both probability modeling and simulation, performance prediction methods are developed to evaluate the performance of the Large Eddy Code [7] of NASA-Ames on a tightly-coupled system such as the CRAY X-MP.

1. INTRODUCTION

Currently, many performance measures have been developed for SIMD machines; straightforward extensions of these to MIMD machines are not satisfactory, as key features of MIMD machines are not adequately reflected. Considerable care has been taken in generalizing these SIMD concepts of performance measures to MIMD machines. To obtain an a posteriori estimate for the optimal number of processors to be used on a multiprocessor system, the concept of cost-effectiveness is applicable. However, to predict performance of an algorithm on a MIMD machine accurately, either simulation or probability and queuing models have to be used to deal with important problems such as synchronization, memory contention, and interprocessor communication. In this paper, we will examine algorithm performance in a tightly-coupled system such as the CRAY X-MP and use performance prediction methods based on analytical modeling and simulation to evaluate some fluid flow codes.

*Funds for the support of this study have been allocated by the NASA Ames Research Center, Moffett Field, California under Contract No. NAS 2-11065.

2. PERFORMANCE MEASURES FOR MIMD MACHINES

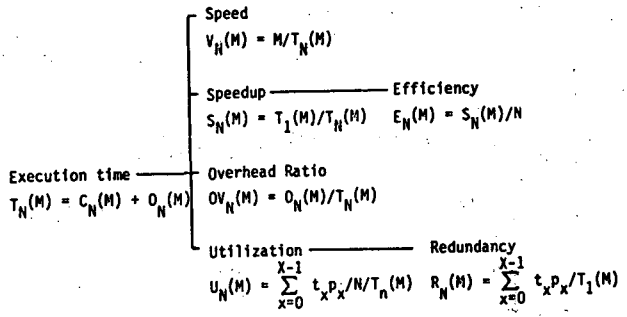
2.1 Measure Extension

Performance measures have become an essential tool in the development of new computer architectures and new computational methods. The basic notions of execution time serve as the foundation for the derivation of other useful measures such as the speedup, the efficiency, and the utilization for SIMD machines.

The objective of measure extension is to clearly separate architectural and algorithmic elements of the measures where possible, so that the suitability of each element to the task at hand can be clearly evaluated.

Following Siegel [1], the fundamental measure of performance is the execution time $T_N(M)$ involved in performing the algorithm for a problem of size M on a system having N Processing Elements (PE's). Other useful measures can be derived from T_N easily, see Figure 2.1. $T_N(M)$ can be expressed as the sum of two components:

$C_N(M)$, the time spent by PE's performing computations which are actually part of the task being performed; and $O_N(M)$, which is the "overhead" time spent "managing" the parallelism. To extend this measure, it is possible to decompose $C_N(M)$ and $O_N(M)$ into various timings which contribute to each component.



Note:
X - # of sequential operation steps
t_x - time to perform step x
p_x - # of PE active for step x

Fig. 2.1 SIMD Algorithm Performance Measures

We have chosen to extend the execution time measure by first observing that the computational element of the execution time has, in general, two contributing elements: $C_N(M) = SEQ_N(M) + P_N(M)$ where $SEQ_N(M)$ is the time required by the system to compute serial portions of the algorithm, and $P_N(M)$ is the time spent in computing in parallel according to the algorithm. All algorithms (depending on the granularity with which one views them) are mixtures of inherently serial and parallel sequences, and this division merely formalizes what must be done in estimating algorithm performance in any case. This division into serial and parallel components is fundamentally algorithmic in orientation, although it could also find origin, perhaps, in the execution strategies of particular architectures. While possible this is rather unlikely, and so the utility of this first division is of primary value in algorithm assessment.

The "overhead" represents another candidate for extension. In general, we feel that a useful decomposition of this element would be: $O_N(M) = SYN_N(M) + D_N(M) + R_N(M)$ where $SYN_N(M)$ is the delay attributable to PE synchronization requirements; $D_N(M)$ is the delay associated with the internal transfer of data within the machine, fundamentally interprocessor communication delays; and, $R_N(M)$ is the residual delay, basically attributable to operating system actions. These elements have architectural and/or algorithmic origins which can be separated in each case of interest.

Finally, in order to estimate the optimal number of PEs to be used in performing the algorithm for a problem of size M on a system having N PEs, the speedup $S_N(M)$ must necessarily be balanced by the 'cost' of the system of PEs. This concept is used in the development of the effectiveness measure in the next section.

2.2 The Effectiveness Measure

The two most commonly used measures in the performance evaluation of SIMD/MIMD machines are speedup $S_N(M)$ and efficiency $E_N(M)$. Unfortunately for a typical problem of fixed size M, S_N increases asymptotically to a finite limit S_∞ as $N \to \infty$, while E_N decreases to 0 as $N \to \infty$, this results in the dilemma of choosing between better speedup or better efficiency. Hence, the determination of an 'optimal' number of processors N is impossible until a balance is struck between speedup and efficiency.

To attain this balance, the notion of performance must be approached from the cost-effectiveness side. In other words, the speedup must be balanced with the basic cost of using N processors. The 'space-time' cost of an algorithm is defined as:

$$CO_N(M) = N\, T_N(M) \ .$$

The effectiveness of an algorithm is then given by:

$$F_N(M) = \frac{S_N(M)}{CO_N(M)} \ .$$

This can be simplified to a more familiar form:

$$F_N(M) = \frac{S_N}{NT_N} = \frac{S_N}{N\frac{T_N}{T_1}T_1} = \frac{S_N E_N}{T_1} \leq 1 \ .$$

Therefore, for a given algorithm (T_1 is fixed), an effective algorithm is one which maximizes the product of speedup and efficiency (Kuck [2]).

Example

This example illustrates the usefulness of the concept of effectiveness based on a simulation run of the fluid code SIMPLE at LLNL (Axelrod et al [3]) using their MPSIM simulator. Their results are tabulated in the first four columns of the following table:

N	Mflops	S_N	E_N	S_N*E_N
1	9.04	1.00	1.00	1.00
2	16.00	1.77	.89	1.99
4	26.45	2.93	.73	2.14
6	32.17	3.56	.59	2.10
8	34.70	3.84	.48	1.84

The measure of effectiveness is calculated in the fifth column. Axelrod et al arrived intuitively at the conclusion that the optimal number of processors should be 4. Using the effectiveness concept, however, this can be easily seen. The effectiveness is maximum around N = 4; moreover, one is better off using 2 processors than 8 in this case. This is clearly illustrated by Figure 2.2.

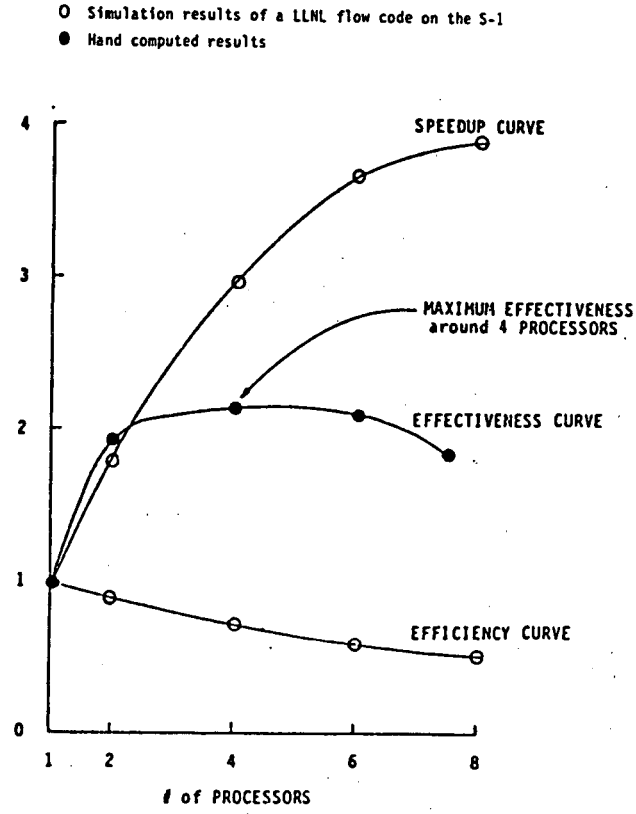


Fig. 2.2 Application of the Effectiveness Measure

3. PERFORMANCE PREDICTION METHODS

In Section 2, the disjoint components of both the computational time $C_N(M)$ and overhead $O_N(M)$ were extended for algorithmic evaluations on MIMD machines. An approach for determining the optimal number of processors to be used for an algorithm of fixed size based on the cost-effectiveness idea was developed and shown to be useful in giving an a posteriori estimate. However, in order to predict performance of an algorithm on a specific MIMD machine, each component of the extended MIMD versions of $C_N(M)$ and $O_N(M)$ has to be computed. There are basically two approaches: (1) Hand compute each

component based on some probability or queuing models and (2) Simulation.

Simulation, though more accurate than hand computation, may be very time consuming, especially when many sets of parameters are required to assess global performance behavior. Furthermore, tailoring the algorithm for simulation, requiring code segmentation for parallel calculations, is both non-trivial and sensitive; slightly different segmentation can produce drastically different results. On the other hand, hand computation, although simplistic, provides a quick way to gain valuable insights into the performance of an algorithm within an MIMD machine. Finally, in order to obtain reliable performance prediction, perhaps the best approach is to cross-correlate the results obtained from hand calculations and simulations.

3.1 Performance Prediction Via Modeling

In order to isolate each component of $O_N(M)$ and $C_N(M)$ and obtain approximate results based on some modeling techniques, the types of algorithms and MIMD architectures of interest have to be put in focus. First, the numerical method to be evaluated, the Large Eddy Code (NASA AMES), is primarily of the synchronized iterative type. Second, a type of MIMD architecture of interest is the CRAY X-MP, and it will be modeled as a tightly-coupled system, see Figure 3.1

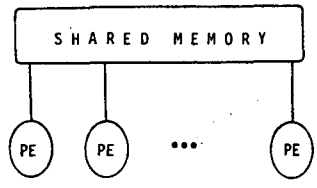


Fig. 3.1 Tightly-Coupled (Shared Memory) System

From Section 2.1, the overhead $O_N(M) = SYN_N(M) + D_N(M) + R_N(M)$.

First, the residual delay $R_N(M)$, due basically to operating system actions will be neglected. The effects of synchronization will be drawn from order statistics. Finally, since the MIMD machine under consideration is tightly-coupled, the dominant factor due to delay associated with the internal transfer of data within the machine will primarily arise from memory contentions and not from interprocessor communication delays.

Based on the above simplification, the strategy for computing the performance measure utilization for a tightly-coupled system is clearly depicted in Figure 3.2.

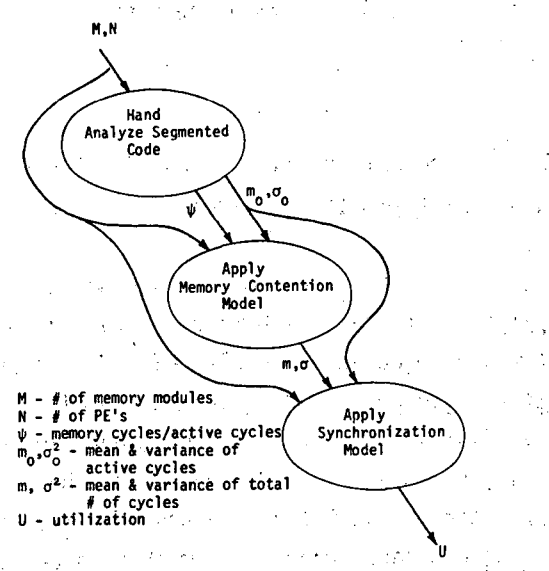


Fig. 3.2 Strategy for Computing Utilization

The set of input parameters necessary for computing the utilization for a shared memory system (tightly-coupled system) is defined in Figure 3.2. It is nontrivial in some cases to obtain numerical values for some of the input parameters.

M, N are architectural design dependent so the values are directly available. ψ represents

the instruction mix of the raw code. This value is very difficult to obtain, because it will involve counting memory access cycles and active cycles line by line in the parallel portion of the code. A more viable approach is to estimate roughly this ratio and compute utilization based on a range of ψ centered at this estimated value.

m_o, σ_o^2 can be computed by performing a flop count for each allocated task of a processor in the multiprocessor system. An easier way is to run the sections of the code corresponding to each allocated task and compute m_o, σ_o^2 directly based on the timings obtained. Note that if all the allocated tasks are the same, then σ_o is zero; however, because of memory contention, the variance of total cycles involved, σ , is non-zero, as seen in Equation 3.6.

3.1.1 Memory Contention Models in Synchronous Multiprocessor Systems

Consider a synchronous multiprocessor system with N processors and M memories with a basic time unit of 1 memory cycle. The N processors are assumed to be independent and their requests are distributed uniformly among the memories. At the beginning of a memory cycle, the processors present their requests. If more than one simultaneous request is made to a particular memory, access conflict occurs. Only one request of a conflicting set is accepted while the others are rejected. All the accepted requests are served simultaneously during the cycle, while processors with rejected requests are blocked during that cycle. These blocked processors will resubmit their requests to the same memories in the next cycle.

The following parameters are crucial to the characterization for modeling memory contention:

ψ = programmed request rate of the processor (or the density of the memory requesting part of a T-cycle trace),

α = dynamic request rate of the processor (or the density of the memory requesting part of the T'-cycle trace),

P_A = probability that a request is accepted.

Since those cycles in which the processor is doing internal computation with no generated memory request are the same in the two traces

$$T(1-\psi) = T'(1-\alpha)$$

(3.1) or $$f = \frac{T}{T'} = \frac{1-\alpha}{1-\psi}.$$

Here f is known as the performance degradation factor. Also, the expected number of rejections (blocked cycles) plus the one accept cycle per request is $\frac{1}{P_A}$, therefore

(3.2) $$\alpha = \frac{\psi/P_A}{1 - \psi + \psi/P_A} = \frac{1}{1 + P_A(1/\psi - 1)}$$

The bandwidth $BW = N\alpha P_A = M\left[1 - (1 - \alpha/M)^N\right]$ so

(3.3) $$P_A = \frac{BW}{N\alpha}.$$

Equations (3.2) and (3.3) constitute a pair of simultaneous equations where P_A and α can be solved for in terms of M, N, ψ.

Yen et al [4] developed a new model which gave better values for BW since the above approach would cause an overestimation of the bandwidth. Their results can be summarized as follows:

$$BW = Nf\psi = M\left\{1 - \left(1 - \frac{f\psi}{M}\right)^N \cdot \left[1 - \frac{1 - \left(1 - \frac{1-f}{M}\right)^N}{M}\right]^M\right\} \tag{3.4}$$

(3.4) can be solved for f by iteration using Newton's method; a reasonable initial guess for f is 1.

P_A and α can then be solved for in terms M, N, ψ via (3.3) and (3.1) respectively.

3.1.2 Law of Total Probability

Let m_o and σ_o^2 be the mean and variance of the random number of active cycles n_o in a task. (Active cycles are execution cycles and memory access cycles which are not rejected.) We need to evaluate the mean (m) and the variance (σ^2) of the total number of cycles (n) in a process. This is done via the law of total probability, i.e.,

$$\begin{aligned} m &= E_o[E[n|n_o]] , \\ \sigma^2 &= E[n^2] - m^2 = E_o[E[n^2|n_o]] - m^2 . \end{aligned} \tag{3.5}$$

where E and E_o designate the expected values taken over n and n_o respectively. Consequently,

$$E[n|n_o] = (1 - \psi)n_o + \psi n_o/P_A ,$$

$$\begin{aligned} E[n^2|n_o] &= E[(n-E[n])^2] + E^2[n|n_o], \\ &= \psi n_o \frac{1 - P_A}{P_A^2} + \\ &\qquad ((1 - \psi) + \psi/P_A)^2 n_o^2 . \end{aligned}$$

Then applying (3.5),

$$\begin{aligned} m &= m_o + m_o\psi(1 - P_A)/P_A \\ \sigma^2 &= \sigma_o^2 + m_o\psi(1 - P_A)/P_A^2 \\ &\quad + \sigma_o^2(1 - P_A)/P_A \\ &\quad \cdot\psi(2 + \psi(1 - P_A)/P_A) \end{aligned} \tag{3.6}$$

In (3.6) the effect of memory conflicts appear as additional terms in the value of mean and variance of a process time. In the next section, the factor of synchronization will be accounted for through use of order statistics; the utilization will then be computed.

3.1.3 Synchronization and Utilization

Dubois and Briggs [5] model the factor of synchronization through order statistics. Let $O_{j:N}$ be the mean of the jth order statistics among N samples drawn from the processing time distribution with mean 0 and variance 1. Then if m_I is the mean iteration time,

$$m_I = m + O_{N:N}\,\sigma . \tag{3.7}$$

Here $O_{N:N}$ depends only on the distribution of the normalized processing time.

The utilization is $U = \frac{m_o}{m_I} = \frac{m_o}{m + O_{N:N}\sigma} = \frac{1}{1 + \Delta}$

$$\Delta = \psi \frac{1 - P_A}{P_A} + O_{N:N}$$

with $$\cdot \sqrt{C_o^2 \left((1 - \psi) + \frac{\psi}{P_A} \right)^2 + \frac{\psi}{m_o} \frac{1 - P_A}{P_A^2}}$$

where $C_o = \frac{\sigma_o}{m_o}$ and P_A comes from Yen's model, Equation (3.4).

In the following sections, this model will be further developed and compared with the results from simulation runs of the Large Eddy Code.

3.1.4 The Large Eddy Code

A study and analysis of mapping the Large Eddy Code [7] from a sequential machine to an MIMD machine can be found in Greenberg & Stevens [6]. A high level structure of the code representing the division between serial and parallel portions of the code is displayed below:

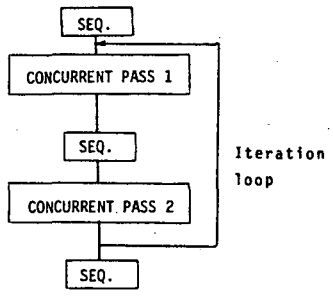


Fig. 3.3 High Level Structure of the Large Eddy Code

The fundamental approach for arriving at the algorithmic concurrency is based on decomposing mathematically the existing 3-D spatial operator into a sequence of orthogonal one-dimensional operators. As a result, when applying one of the operators to the 3-D cube, all lines of data may be updated concurrently since they are totally independent. The two concurrent portions shown in Figure 3.3 represent allocation among N PE's of the 3-D cube divided into N slabs in each of Y and Z directional groups. In particular, if a 2 processor MIMD architecture was used, each processor would take two of the Y and Z slabs. In such a case, each task assigned to a PE is almost identical, i.e., $\sigma_o^2 = 0$. However, based on the analytic model developed the variance of the total number of cycles M is non-zero; in fact, the mean and variance from Equations 3.6 are:

$$m = m_o \left(1 + \psi \frac{1 - P_A}{P_A} \right) ,$$

$$\sigma^2 = m_o \cdot \psi \frac{1 - P_A}{P_A^2} .$$

The utilization is $U = \frac{1}{1 + \Delta}$ where Δ is now expressed as

$$\Delta = \psi \frac{1 - P_A}{P_A} + O_{N:N} \sqrt{\frac{\psi}{m_o} \frac{1 - P_A}{P_A^2}} .$$

However, as noted by Dubois and Briggs [5], for $m_o > 10^4$ cycles, the dependence of Δ on m_o is negligible.

$$\Delta = \psi \frac{1 - P_A}{P_A} ,$$

where P_A depends on a given memory contention model.

In the following, we will apply the two models developed to the Large Eddy Code using the hardware configuration of the Cray X-MP. First, the instruction mix ψ of the Large Eddy Code is based on the simple assumption that in the iteration loop, there are two memory fetches to one floating point operation, i.e., $\psi = \frac{2}{3}$.

Second, for the Cray X-MP, which is a dual processor machine N = 2, while the number of memory modules M is either 16 or 32. Applying the first model based on the rate adjusted probabilistic approach and Yen's model of memory contention, the following table can be derived.

TABLE 3.1. UTILITIZATIONS AND LOCAL SPEEDUP RATIOS

Model A - Rate Adjusted Probabilistic Approach

M	P_A	Δ	U	S_N
16	.981	.013	.987	1.974
32	.990	.0067	.993	1.986

Model B - Yen's Model

M	P_A	Δ	U	S_N
16	.9975	.0154	.984	1.968
32	.990	.0067	.993	1.986

In Section 4.1, these results will be correlated with the simulation results of the Large Eddy Code which we now discuss.

3.1.5 Simulation Results of the Large Eddy Code

Following is a summary of the simulated performance of the Large Eddy Code on the CRAY X-MP:

TABLE 3.2. AVERAGED SPEEDUP RATIOS

Number of Iterations	Grid or Mesh Size	S_N
5	16^3	1.68
100	16^3	1.977
5	32^3	1.74
100	32^3	1.99

Basic data for these results are obtained by running the Large Eddy Code on the CRAY 1S at NASA Ames [6]. Timings thereby obtained are then extrapolated for a CRAY 1S with two processors, assuming that the parallel portions of code are divided evenly among the two processors. The speedup obtained is then assumed to be the same for the CRAY X-MP. There are four cases with different values for grid size and number of iterations. Note that for a problem with fixed mesh size, as the number of iterations increases, the speedup ratio increases also. This can be easily explained. For a problem with fixed mesh size, the timing for the sequential part is constant, so SEQ is constant, while the timing for the parallel part increases as the number of iterations increase, so P_N increases. But from Equation 4.3, $dS_N/dP_N > 0$, so the speedup ratio S_N increases also.

4. CORRELATIONS BETWEEN PERFORMANCE PREDICTIONS VIA MODELING AND SIMULATION

In order to compare results from modeling and simulation, the relationship between the measures speedup and utilization must be clearly drawn.

There is a basic difference between the speedup measure and the utilization measure. Normally, the speedup measure is calculated based on the overall execution time of the code which will include the sequential, as well as parallel portion of the code. On the other hand, by definition, utilization is a measure based solely on the parallel portion of the code. Therefore, speedup measures the global behavior of the code, while utilization is directed specifically towards the parallel portion. However, if one restricts the definition of speedup solely to the parallel section of the code,

(4.1) $$S_N = \frac{N.m_o}{m_I} = N.U_N ,$$

(4.2) and $$E_N = U_N .$$

Therefore, the utilization measure is nothing more than the efficiency measure restricted to the parallel portion of the code. In particular, in an ideal situation where $m_o = m_I$, $S_N = N$ and $U_N = E_N = 1$ as expected.

4.1 Comparisons Between the Analytical Model and Simulation

The results obtained for the Large Eddy Code via analytical modeling and simulation are presented in tables 3.1 and 3.2 respectively. As pointed out in the beginning of this section, the speedup ratios of the analytical model measure only the concurrent portion of the code while speedup ratios from simulation runs assess the global performance of the code. However, as the number of iterations increases (to $\simeq 100$) so that the timings for the concurrent portion become dominant, the speedup ratio obtained by simulation is between 1.977 and 1.99, which is almost identical to that obtained via analytical modeling, see table 3.1. It is interesting to note, however, that even if the parallel portion of the code is 100% efficient, the speedup ratio, according to the simulation methodology used in [6], can never reach 1.0! More interestingly, for a problem of fixed size, using more processors increases the gap from the ideal speedup. This puzzling phenomenon will be discussed in the next section.

4.2 Proper Interpretation and Extension of the Speedup Measure

For simplicity, assume the overhead $O_N(M)$ is zero, so the speedup can be written as:

(4.3) $$S_N(M) = \frac{T_1(M)}{SEQ_N(M) + P_N(M)} ,$$

where as before, SEQ_N represents timing of the sequential part of the code and P_N the parallel part.

To be specific, consider a problem of fixed size. In many cases, the sequential time is independent of N, therefore, for ideal speedup of the N PE's

(4.4) $$S_N = \frac{SEQ + P_1}{SEQ + P_1/N} < N \text{ for } N > 1.$$

In fact if N increases, $N - S_N$ increases. This result does not seem reasonable, because ideal speedup has been attained by the N PE's of the system; if there is any justice at all, S_N should equal N. Moreover, why should performance of a multiprocessor system be penalized for using more PE's just because a portion of the code involves constant sequential execution time? Therefore, in order to reflect correctly the performance of the N PE's in this particular case, SEQ should be eliminate from the evaluation of the speedup S_N. Otherwise, the degree of concurrency attained by the multiprocessors system will be camouflaged!

In other words, when comparing the performance of two MIMD architectures using a fixed algorithm, the 'concurrent' speedup measure is much better suited than the conventional speedup measure, because the conventional one typically disperses the focus on the concurrent portion of the algorithm.

5. CONCLUSIONS

(1) Correlations were made between the simulation results and that of analytic modeling in Section 4. The analytic models calculate the utilization of the multiprocessors per iteration loop and hence assess only the local performance

of the code, whereas simulation models the global performance. Some simulation calculations are based on straightforward extrapolations and are not concerned with memory contention and processor synchronization. Improved tools would come from simulations incorporating analytical models, such as the LLNL MPSIM.

(2) Extensions of performance measures from SIMD to MIMD machines are fundamental in architectural evaluations. The proper extension is important, because the concept of optimization rests entirely on a chosen norm/measure. Therefore, only correct choices can lead to correct optimal performance predictions. We have identified and extended the measures essential to MIMD machine evaluation in Sections 2 and 4.

(3) The development of a methodology for the optimal and orderly mapping of numerical algorithms onto a MIMD machine is entirely lacking today. The present strategy is very primitive and usually done at a very low level of the software -- dividing up a DO-loop evenly among the PE's is typical, as in the case of the Large Eddy Code. The concept of optimization has never been thoroughly investigated in such instances. The creation of taxonomies for both fluid flow codes and MIMD machines aids in bringing the essential elements of both software and hardware into proper focus, so that the domain and range of the mapping can be clearly identified. However, further work in the area of multi-task scheduling is still urgently required to maximize overlapping of the software tasking into complex MIMD machines to obtain optimal utilization.

6. ACKNOWLEDGEMENT

The authors are indebted to Ken Stevens, who read the manuscript critically. The support of the NASA Ames Research Center in this research is gratefully acknowledged.

References

1. L. J. Siegel, H. J. Siegel, and P. H. Swain, "Performance Measures for Evaluating Algorithms for SIMD Machines", IEEE Trans. Software Engineering, Vol. SE-8, No. 4, July 1982.
2. A. H. Sameh and D. J. Kuck, "Parallel Direct Linear System Solvers", Parallel Computers - Parallel Mathematics, 1977, pp. 25-30.
3. T. S. Axelrod, P. F. Dubois, and P. Eltgroth, "A Simulator for MIMD Performance Prediction - Application to the S-1 MkIIa Multiprocessor", Lawrence Livermore Laboratory, Livermore, California, UCRL-88765, 1983.
4. D. W. Yen, J. H. Patel, and E. S. Davidson, "Memory Interference in Synchronous Multiprocessor Systems", IEEE Trans. Comput., Vol. C-31, No. 11, November 1982.
5. M. Dubois and F. A. Briggs, "Performance of Synchronized Iterative Processes in Multiprocessor Systems", IEEE Trans. Software Engineering, Vol. SE-8, No. 4, July 1982.
6. M. G. Greenberg and K. G. Stevens, Jr., "Efficiencies of MIMD Architectures in Computation Fluid Dynamics", NASA Ames Research Center Report, December 1982.
7. R. S. Rogallo, "Numerical Experiments in Homogeneous Turbulence", NASA Technical Memorandum 81315, 1981.

USING STATISTICAL SOFTWARE WITH A DATABASE MANAGEMENT DATA THEORY

Robert J. Muller
Oracle Corporation

ABSTRACT

This paper analyzes several statistical computing environments in an effort to show the relationship between formal database management theory and the computational environment as an integral part of data analysis.

The paper uses Entity-Relationship theory as a data theory to compare the flexibility, representation, and problem-solving difficulties of the analyst in the Minitab, BMDP, SPSS, SAS, OSIRIS, SIR, and the Consistent/System computing environments. It shows that none make full use of database management capabilities; it also shows that even those capabilities are not capable of dealing with all problems confronting the statistically oriented data analyst.

INTRODUCTION*

A data analyst uses a data theory to represent information within a software environment, producing analysis. This simple model--analyst, theory, environment--is treated in more detail elsewhere [1].

As in all socially constructed realities, there is a tendency for the conceptual to seem real. Thus I say that a particular software environment implements a particular data theory. The reader should bear in mind, however, that the data theory may not resemble the physical architecture of the computer in any respect. The software which comprises the environment is the only "real" thing which the analyst sees, and even that software is a relatively high-level abstraction.

The Data Theory

Database management theory is a useful tool for representing data. The purpose of any data theory is to allow an analyst to interpret observations to give them meaning relative to the real world under study--that is, to make observations into data. A true data theory is a set of axioms and definitions that specify the basic structures of observations [2].

There are many variants of database management theory [3, 4, 5]. I have chosen to use one such theory, entity-relationship (ER) theory, because it is compatible with most of the other theories and represents the meaning of data well without incurring the complexity of more specialized theories [6, 7]. This is not to say that all meaning is captured; for example, data relating to procedural entities such as scripts, recipes, or other procedural constructs cannot be represented.

Briefly, the entity-relationship (ER) data theory conists of structures, operations, and constraints. The structures include entities, attributes of entities, relationships, and attributes of relationships. The operations include query and nonquery operations. Query operations retrieve data by specification of a predicate on the structures. Nonquery operations manipulate data, also by specification of a predicate; these include insertion, deletion, and structure definition operations. Finally, constraints restrict the possible states of the database under the available structures and operations, again by specification of a predicate. Major types of constraints include key, domain, dependency, and existence constraints.

---

*The interpretations and opinions presented in this paper are those of the author, and Oracle Corporation bears no responsibility for them. A longer version of this paper is available from the author and in reference [1].

A database is a series of tables that represent sets of entities associated with properties. These tables are connected to each other by relationships, which are tables relating entities to entities but which also contain properties of the relationship. Constraints on these tables--entity and relationship tables--restrict the states of the database under the available set of operations.

The implications of this data theory are numerous, as are the potential problems; too numerous to deal with in a short paper. I have dealt with some of these issues elsewhere [1], as have many authors in the database management literature. In this paper, I will take for granted the basic data theory presented above, using it as a basis for comparing the several software environments.

Flexibility

Flexibility is the manner in which the software environment is able to allow novel uses of the environment. A flexible system is one which an analyst can use to solve a wide variety of problems, at least some of which had not occurred to the system designers. Modern software system designers may take flexibility into account in at least two ways--command configuration and extension facilities.

Command configuration is the relationship between commands and problem solutions. By command, I mean a single procedure or operation that takes some input and produces some output. There are at least two aspects to command configuration: modularity and intercommunication. Modularity is the degree to which an environment breaks up functional elements. Intercommunication is the degree to which the output of one command may act as input to another command. Modularity varies from highly modular to monolithic. For example, a system that has one command for each calculation involved in solving a problem is a highly modular system; a system that solves the entire problem in one large command, performing all subsidiary calculations as part of the command, is a monolithic system. Intercommunication varies from strong to weak. For example, an extremely strong system would be able to transmit results from each command to each other command, at least insofar as the receiving command could sensibly use the communicated data. A weak system would be unable to transmit results.

Extension facilities are software tools that allow analysts to extend the system. That is, these tools allow analysts to produce new commands or novel combinations of old commands. There are at least three methods for providing this sort of facility: an environmental procedural capability, an external procedural capability, and a macro facility [8].

An environmental procedural capability allows the analyst to compose a new command that is used like the other commands of the environment. Usually, this involves writing a program in the language in which the system is written, integrating this new command with the others by means of standardized subcommands and protocols for input and output of data. This sort of facility allows the analyst the full flexibility of the programming language to create new commands.

There are really two factors involved in judging the difficulty of integrating a new command into an environment. Given proper access to the source code and sufficient programming sophistication, new environmental commands can be added to anything. Most social scientific analysts do not have, and do not want to have, such programming sophistication. Effectively, an environmental procedural capability is a part of the software environment that allows more-or-less competent--not just sophisticated--programmers to add commands to the environment.

An external procedural capability is an interface between the software environment and some external environment in which a command is available. Typically, this sort of capability consists of either the ability to run a previsouly written subroutine in the general computer environment from the software environment in which the analyst works or the ability to send data from the system to another software environment (another statistical package, for example) by means of some protocol, such as a system file or

dataset readable by the external environment. The facility always provides some kind of standardized input/output protocol for communicating with the external command.

A macro facility is, in a special sense, an abbreviation facility. There is an enormous variety of macro facilities. One common type is pure abbreviation; a macro consists of a sequence of instructions issued whenever the macro name is issued. More sophisticated types have procedural control arguments such as if-then structures, looping capabilities, and the ability to pass parameterized arguments to the macro--qualities making the macro facility a high-level programming language.

Flexibility is determined by a quite complex combination of these tools in a given software environment. A monolithic system, for example, has less need for strong intercommunication than a modular system. Still, provision for intercommunication can make even a monolithic system more flexible, since many analytic procedures use very similar sorts of input and output. It is certainly much easier to create novel applications in a modular system, however; such a system is designed to be flexible in combining commands. And a modular system combined with a macro facility can be extremely flexible, while a monolithic system can't take much advantage of macros; such a system would depend mainly on environmental procedural extensions.

Representation

The data model is a combination of structures, operations, and constraints; representation is the ability of the software environment to represent the data model desired by the analyst. For the purposes of comparison, a somewhat more general approach must be taken. I will describe the general capabilities of each system in each category of the data model. After this general exposition, I will compare the capabilities to the entity-relationship data model. I will also mention the extensional capabilities of the environment, the extent to which the analyst can extend the data model in novel ways.

Problem-Solving Difficulties

There are many different aspects to this problem area; it defies easy categorization. I will limit myself to several very important aspects of the software environment that have an impact on problem-solving difficulties: processing mode, command style, procedurality, error messages and processing, editing capabilities, and documentation and help facilities.

Processing mode is the way in which the environment proceeds. An environment can be interactive, batch, or both (but not both at the same time). An interactive system is one in which the analyst communicates directly with the computer before and after each command. A batch system is one in which the analyst submits a batch of commands, complete in itself, and gets back output after the computer executes the commands. An environment may have both capabilities; it may be able to run interactively, but it may also be able to execute batches of commands with no intervention from the anlayst. Interactive processing is extremely useful for exploratory analysis; batch processing is useful for jobs that require large amounts of computation with little intervention. Highly modular systems tend to be interactive; monolithic systems then to be batch.

There are two command styles, command-driven environments and prompting environments. A command-driven environment is one in which the analyst issues commands containing all the options necesary for the command to proceed. The choice of which commands are appropriate at any given time is left totally to the analyst. A prompting environment is more structured; the environment prompts the analyst with structured prompts that guide the choice of the analyst either with respect to options or to commands. Prompts may be brief, conversational, or menu (several commands or options displayed at once, allowing the analyst to quickly specify choices). An environment can have both styles.

The relationship between command style and problem-solving difficulty has to do with the way the analyst works. A computer-naive analyst, or an analyst with little software or analysis

experience, may strongly prefer to be prompted for commands or options. But a sophisticated analyst, who knows computers and the software environment, is likely to be annoyed by constant prompting and structuring. In addition, highly structured systems may be difficult to adapt to novel analyses [9].

Procedurality is the extent to which operations in the system are procedural. In particular, the presence or absence of control structures, combinations of statements that conditionally execute commands, determines the extent of procedurality. There are two aspects to the impact of procedurality in a software environment, both relating to the difficulty of proceeding. First, for simple problems it is faster to specify a result rather than to tell the computer just how to achieve that result in terms of control operations on the data [10]. But, second, for many complicated problems, if there is no straightforward way of specifying the result, the procedural solution may be less complex. There is some experimental evidence that procedural environments are better for complex problems than nonprocedural environments [11], but this evidence isn't very conclusive and doesn't really apply to the sort of environments which I discuss. Complexity, in this case, depends on the operations available to the system and on the extensibility of the system.

Error messages and handling are the ways in which the computer responds to the analyst when a mistake, either by the analyst or by the command, occurs. Some systems provide extensive error messages, though most err toward brevity. Although extensive error messages can be useful, most errors are typographical. To print an extensive message detailing all the ways in which the command as given is wrong is likely to be annoying to the person who just typed the wrong letter and realized the mistake immediately [9]. On the other hand, just printing error numbers or meaningless system messages is annoying and difficult to interpret.

Even more important than error messages is error handling, what the environment does on the occurrence of an error. Some systems have virtually no capabilities, essentially destroying the environment when an error occurs. Other systems have extensive error facilities giving the analyst a great deal of control over what happens when an error occurs. Other systems assign an interpretation to the problem and continue processing.

Most systems have the ability to change data known to be invalid. Editors come in many colors, minimal and fancy. Two basic types are editors that work by specifying the location of the entity to be changed and editors that specify the logical characteristics of the entity with a predicate.

The main resource for an analyst confronted with a complex software environment is the documentation for that system, either in printed form or as an on-line help facility. Documentation of either sort comes in at least four levels: (1) the primer or introduction to the system; (2) the middle-level summary of commands; (3) the guide to particular applications; and (4) the complete reference to the system. Most on-line help facilities are limited to middle-level summaries telling the analyst the syntax and usage of commands.

All of these aspects of the software environment have some impact on problem-solving difficulties. The overall environment is a subtle and complex conjunction of these and other components of the software system. Each environment is different; each must be evaluated for a particular analyst's needs and experience.

Table 1 summarizes the three problem areas and the specific aspects of the software environment relative to each.

I selected the several systems I discuss below from those available on the basis of the popularity of the system followed by my own access to the system. I have used all of the systems aside from OSIRIS and SIR, both of which I describe from the documentation. I make no sampling claims; this is basically a series of case studies rather than a sampling of available systems. As well, I haven't the space to go into much detail about each system; the reader is urged to use the system before forming strong conclusions about the system's suitability for any purpose.

Table 1: Aspects of the Software Environment

Flexibility

- Modularity
- Intercommunication
- Extensibility

Representation

- Structures
- Operations
- Constraints
- Extensibility

Problem-Solving Difficulties

- Processing Mode
- Command Style
- Procedurality
- Error Messages and Processing
- Editing Facilities
- Documentation and Help Facilities

In order to better compare these software environments, I will solve two problems as examples for each environment. I assume a database concerning criminal victimization with a hierarchical structure of three tables, household information, personal information, and incident information. The first problem, a problem in tabulation, is designed to show how the environment deals with conceptually simple but structurally complex problems. The problem: tabulate the number of criminal incidents by family income and type of crime. This tabulation would show how the affluence of the victim affects the various rates for different types of crime. I limit the example to counts; percentages or rates in the general population would complicate the issue, since that would involve estimating the relevant population or using the weights assigned by the sampling strategy.

The second problem illustrates the flexibility and power of the environment. The problem is to calculate a special sort of matching similarity measure for incidents, then to use the measure in a straightforward hierarchical clustering porcedure, then to interpret the results using median polishing [1].

This example also illustrates the tension between database and analysis environments. The similarity operation should probably be an internal procedure. To attempt to implement this complex a procedure in regular database operations is (1) unlikely to work or (2) likely to be computationally inefficient or dangerous. Some systems designers (Janus, for example) would discourage the attempt; others (SAS, for example) would encourage it. The latter see their systems as all-purpose systems; the former as instruments designed for relatively special purposes as components of a more general system.

Table 2 shows how the software environments considered rate in each of the various categories considered. Combined with the above discussion, Table 2 should give the reader a clear idea of the nature of these environments. The following sections examine each environment to see how the two example problems might be solved and how well the environment is capable of representing the Entity-Relationship data theory.

MINITAB

Minitab cannot be used to implement the ER data model. The folylowing command, given properly structured input, could do the required tabulation.

```
TABLE COL1 COL2
```

The clustering problem could not be done in Minitab.

BMDP

BMDP cannot be used to implement the ER data model. The following job does the tabulation.

```
//RJMX JOB RJM,
// PROFILE=(DEFER,MEMORY=2560),
// TIME=20
// EXEC BIMEDT,PROG=BMDP4F,PRINT=PRINT,
// TIME=20
//TRANSF DD *
<here would be the appropriate
```

Table 2
Comparison of Software Environments

Aspect of System	Minitab	BMDP	SPSS	SAS	OSIRIS IV	SIR	Consistent System
Flexibility							
Modularity	more or less modular	monolithic	monolithic	monolithic	monolithic	monolithic	modular
Intercommunication	some	some	none	extensive	some	none	extensive
Extensibility	simple macro	environmental procedure	none	environmental, external procedures; macro	environmental, external procedures	environmental procedures	environmental, external procedures, macros
ER Representation							
Structures	rectangular data matrix with limited information	rectangular data matrix	rectangular data matrix	rectangular data matrix	rectangular data matrix, linked tables limited to hierarchical structure	linked tables	multidimensional data matrix; linked tables
Operations	limited	limited	limited	moderate	moderate	moderate	extensive
Constraints	none	value	value	value	key, conditional dependency, existence	key, value	value, conditional dependency
Extensibility	none	none	none	none	none	none	none
Problem-Solving Difficulties							
Processing Mode	interactive, batch	batch	batch	batch, interactive	batch	batch, interactive	interactive
Command Style	command driven	command driven	command driven	command driven	command driven	command driven	command driven
Procedurality	nonprocedural	nonprocedural	nonprocedural	procedural, nonprocedural	nonprocedural	procedural	nonprocedural

Table 2 (continued).

Aspect of System	Minitab	BMDP	SPSS	SAS	OSIRIS IV	SIR	Consistent System
Error Messages and Handling	brief, informative errors; returns control to analyst most of the time	extensive, uninformative errors; stops running on error	extensive, uninformative errors; stops running on error	brief, informative errors; stops current step; sometimes aborts environment	?; returns control to analyst	?	brief, informative errors; returns control to analyst; infrequently aborts environment
Editing	position editor	position, conditional editor	conditional editor	line, full-screen position; conditional editor	conditional editor	conditional editor	positional, conditional editors
Documentation	primer, reference, on-line help	reference	primer, reference	primer, reference, applications guides	reference	reference	primer, reference, applications guides

```
  FORTRAN subroutine>
//GO.FT10F001 DD DSN=NCS,VOL=SER=002035,
// UNIT=T6250,LABEL=(1,SL),
// DISP=(OLD,KEEP)
//GO.SYSIN DD *
/PROBLEM TITLE IS 'INCIDENT, FAMILY
          INCOME VERSUS CRIME TYPE.'
/INPUT VARIABLES ARE 0.
       CASES ARE 13368.
       UNIT IS 10.
/VARIABLE ADD = 2.
          NAMES ARE INCOME, TYPE.
/CATEGORY CODE (1) ARE 1, 2, 3, 4, 5, 6,
          7, 8, 9, 10, 11, 12, 13, 14,
          15, 16, 17, 18, 19, 20, 21,
          22, 23, 24, 25, 26, 27, 28,
          29, 30, 31, 32, 33, 34, 35,
          36.
          CODES (2) ARE 1, 2, 3, 4, 5,
          6, 7, 8, 9, 10, 11, 12, 13.
/TABLE COLUMN IS TYPE.
       ROW IS INCOME.
/END
/*
```

The only way to do the clustering problem would be to design a BMDP environmental procedure complete with similarity judgment, clustering, and median polish, since none of these are present elsewhere in BMDP.

SPSS

SPSS can't be used to represent the ER model. The job to produce the tabulation follws.

```
RUN NAME          INCIDENTS, FAMILY INCOME
                  VERSUS CRIME TYPE
VARIABLE LIST     INCOME, TYPE
INPUT MEDIUM      CARD
N OF CASES        13368
MISSING VALUES    INCOME   (14,15,15)/TYPE
                  (0)
INPUT FORMAT      FIXED (2F2.0)
VALUE LABELS      INCOME  (1)  UNDER $1000
                  (2)   $1000-$1999    (3)
                  $2000-$2999          (4)
                  $3000-$3999          (5)
                  $4000-$4999          (6)
                  $5000-$5999          (7)
                  $6000-$7499          (8)
                  $7500-$9999         (10)
                  $12000-$14999       (11)
                  $15000-$19999       (12)
                  $20000-$24999       (13)
                  $25000  AND  OVER/  TYPE
                  (1)  RAPE  THEFT  (2) AT
                  RAPE THEFT  (3)  ASSAULT
                  WEAPON THEFT (4) ASSAULT
                  NO  WEAP  THEFT  (5) MIN
                  ASS THEFT  (6)  RAPE  NO
                  THEFT  (7)  ATT  RAPE NO
                  THEFT (8)  ASS  WEAP  NO
                  THEFT (9) ASS NO WEAP NO
                  THEFT  (10)  MIN  ASS NO
                  THEFT (11) ATT ASS  WEAP
                  NO  THFT  (12)  AT AS NO
                  WEAP NO THF (13) ROBBERY
                  WEAPON (14)  ROBBERY  NO
                  WEAPON   (15)   ATT  ROB
                  WEAPON (16) ATT  ROB  NO
                  WEAPON (17) PURSE SNATCH
                  (18)  ATT  PURSE  SNATCH
                  (19) POCKET PICKING (20)
                  BURG  FORC  NO  STL  DAM
                  (21)  B FC NO STL NO DAM
                  (22) BURG FORCE (23) BUR
                  NO FORC  (24)  BURG  ATT
                  FORCE  (25)  LARC  <  10
                  (26)  LARC  10-24   (27)
                  LARC   25-49   (28)  LARC
                  50-99 (29) LARC  100-249
                  (30)  LARC  >  250  (31)
                  LARC NA AMOUNT (32)  ATT
                  LARCENY  (33)  CAR THEFT
                  (34) OTHER VEHICLE THEFT
                  (35) ATT CAR THEFT  (36)
                  ATT OTHER VEH THEFT
CROSSTABS         VARIABLES  =  INCOME (1,
                  13) TYPE (1,  36)/TABLES
                  = INCOME BY TYPE

READ INPUT DATA
<data here>
FINISH
```

The clustering example can't be done in SPSS at all.

SAS

SAS can't be used with the ER data theory. The following SAS job will produce the required table of incident frequencies tablulated by type of crime (TYPE) and family income (INCOME).

```
//RMX JOB RJM,
// PROFILE='DEFER,MEMEORY=1000',
// TIME=20
//EXEC SAS,PRINT=PRINT,TIME=20
//IN DD DSN=RM.SAS.LIBRARY,DISP=OLD
PROC SORT DATA=IN.PERSON; /*SORT DATA */
  BY PKEY;   /*PREPARING FOR THE MERGE*/
PROC SORT DATA = IN.INCIDENT;/* SORT */
  BY PKEY;   /*PREPARING FOR THE MERGE*/
DATA A; /*CONSTRUCT INCIDENT LEVEL */
  MERGE IN.PERSON IN.INCIDENT; /*MERGE*/
    BY PKEY; /*ASSOCIATE PERSON VARS
                WITH INCIDENT VARS */
  KEEP IKEY PKEY HKEY TYPE; /* KEEP ONLY
                                THESE */
```

```
PROC SORT; /* PREP FOR MERGE */
  BY HKEY;
PROC SORT DATA=IN.HSEHOLD;/*PREP MERGE*/
  BY HKEY;
DATA B; /*CREATE INCIDENT LEVEL DATA */
  MERGE IN.HOUSEHOLD A; /*MERGE*/
    BY HKEY;
  KEEP TYPE INCOME; /*KEEP THESE */
PROC FREQ; /*PRODUCE THE TABLE*/
  TABLE TYPE*INCOME;
/*
```

It would be possible, if difficult, for a sophisticated programmer to write an internal procedure to do the clustering problem.

OSIRIS IV

OSIRIS can represent more, but not all, of the structures, operations, and constraints of the ER model. The following example assumes a structured OSIRIS file and produces the appropriate tabulations.

```
&ENTRY
 ENTRY = 1
 UNIT = 3
 G1 + G3
 GNUM = 1
 GNUM = 3
&TABLES DICTIN=<dict file>
     DATAIN=<data file>
 TABLE OF INCIDENTS, FAMILY
    INCOME BY CRIME
 ENTRY = 1
 VAR=V3081 STRATA=V1024
&END
```

OSIRIS can't do the similarities at all.

SIR

SIR does not have the full range of operations and constraints necessary to the ER model. The following retrieval would produce an SPSS file which could be input to the SPSS job above to produce the appropriate table.

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC 2
.     MOVE VAR INCOME
.     PROCESS REC 4
.        MOVE VAR TYPE
.        PERFORM PROCS
.     END PROCESS REC
.  END PROCESS REC
END PROCESS CASES
SPSS SAVE FILE FILENAME = XTABFILE
END RETRIEVAL
```

SIR probably can't perform the retrieval necesssary to enabling SPSS to do the clustering.

CONSISTENT SYSTEM

Janus, the database management system of the Consistent System, has almost the full ER model capabilities. The following series of commands yields the tabulation.

```
create_relation MEMBER_VICTIMIZED_IN :=
  compose (POPULATED_BY, VICTIMIZED_IN);
create_attribute family_income in
   incident := infer (family_income
   thru MEMBER_VICTIMIZED_IN);
change_default_dataset incident;
eval xtab (family_income,
   type_of_crime);
```

The similarity/clustering problem could be most easily done in the CS by altering an already existing program to do the similarities and feeding the output into a clustering program. This requires moderate programming sophistication.

SUMMARY

None of the software environments examined proved completely satisfactory from the analyst's perspective. Some environments were, however, better than others. Some, such as Minitab, could not handle complex data at all. Others, such as BMDP, proved very unwieldy. Three general conclusions can be drawn from the evaluative effort.

First, most software environments oriented toward data analysis are not very friendly to the analyst. Most lack

basic facilities such as help files or extensibility; others are unwieldy and difficult to use, such as the batch systems.

Second, most such software environments cannot deal with complex data structures. Most rely on the rectangular data matrix. Most do not have the capacity to represent the basic structures of the ER model, itself not a particularly strong representational system from the viewpoint of theoretical social science [1].

Third, most environments lack the flexibility necessary to doing data analysis. Even if the system has moderately sophisticated data representation, the system usually can't be easily extended to novel applications.

Data analysis, because it is done by people who don't know much about computers but who have sophisticated scientific problems to solve in creative ways, has some special needs for a software environment. Such an environment must be able to represent data to the satisfaction of the analyst. It must be flexible enough to allow the analyst to do what he or she wants. And last but not least, the environment must be easy to use.

REFERENCES

1. Robert J. Muller, Data Organization: The Integration of Database Management, Data Analysis, and Software Technology Applied to the National Crime Survey, PhD dissertation, Massachusetts Institute of Technology, October 1982.

2. Clyde H. Coombs, A Theory of Data, Wiley, 1964.

3. Christopher J. Date, An Introduction to Database Systems, Third Edition, Addison-Wesley, 1981.

4. Dionysios C. Tsichritzis and Frederick H. Lochovsky, Data Models, Prentice-Hall, 1982.

5. Jeffrey D. Ullman, Principles of Database Systems, Computer Science Press, 1980.

6. Peter Pin-Shan Chen, ``The entity-relationship model--toward a unified view of data,'' Association for Computing Machinery Transactions on Database Systems, Vol. 1, March 1976, pp. 9-36.

7. Peter Pin-Shan Chen, Entity-Relationship Approach to Systems Analysis and Design, North-Holland, 1980.

8. Ree Dawson and John C. Klensin, ``User extension to statistical software,'' Proceedings of the Statistical Computing Section, American Statistical Association, American Statistical Association, 1980, pp. 332-334.

9. John C. Klensin, ``Is all the world a list? or The ontology of programming languages or It is what you say it in or if we build a context-dependent model in a context-dependent programming language, do we understand the resulting context?''.

10. Ben Shneiderman, Software Psychology: Human Factors in Computer and Information Systems, Winthrop Publishers, Inc., 17 Dunster Street, Cambridge, MA, 02138, 1980.

11. Charles Welty and David W. Stemple, ``Human factors comparison of a procedural and a nonprocedural query language,'' Association for Computing Machinery Transactions on Database Systems, Vol. 6, December 1981, pp. 626-649.

AUTHOR INDEX