ADAMS: an Object-Oriented System for Epidemiological Data Manipulation

Leonardo Meo-Evoli*, Enrico Nardelli+, Domenico M. Pisanelli#, Fabrizio L. Ricci*

* ISRDS, Consiglio Nazionale delle Ricerche, Italy.
+ IASI, Consiglio Nazionale delle Ricerche, Italy.
# ITBM, Consiglio Nazionale delle Ricerche, Italy.

Abstract

In epidemiology it is extremely useful to compare the observed trends of various phenomena with the expected trends in order to find out abnormal morbid phenomena: in statistical databases (SDBs) this means performing table manipulation on aggregated data (macro data). The table manipulation are often implemented in different ways, since different aggregate functions are used to generate different kinds of macro data (data type) from disaggregated data. Therefore, logically similar manipulations at macro data level need to be defined separately and ad-hoc by the user for each data type. We propose to model macro data using an object-oriented approach with an instance-inheritance mechanism, which allows the user to manage an SDB without having to explicitly deal with the different data types (automatic data type management). In the paper we describe the static and dynamic properties of our object oriented-model; the metaschema of statistical database system embodying such concepts (ADAMS: Aggregated DAta Management System) and the advantages of our system are discussed.

Keywords: human-machine interface, statistical database, object oriented approach, epidemiological data manipulation

Introduction

Throughout the world epidemiological data is recognized as being important for an effective health care policy. In fact these data, when correlated to geographical patterns, may evidence local deviations from standard values and give rise to adequate action.
Such a "unique source of readily-available health status indicators" [11] is also of priceless value for the prevention of diseases, a fact which has been acknowledged by the World Health Organization (WHO) and which was ratified in the twenty-ninth World Health Assembly. WHO recommends member nations to properly identify and correctly tabulate the causes of death (reported by each country on appropriate forms) in order to identify the main trends and to effectively stress prevention. "The most effective public health objective is to prevent the precipitating cause from operating." [28].
Epidemiological data are therefore relevant instruments in supporting decision-making processes in health care management and planning. In this paper we present ADAMS, an object oriented system for epidemiological data manipulation whose aim is to help in the phase of exploiting statistical tables describing a particular phenomenum.

ADAMS (Aggregated DAta Management System) allows a given table (e.g.: local values) to be examined by generating the reference table (e.g.: national values) with the same structure and therefore it is a valid tool in extracting information from statistical tables. It can contribute to the prevention and the detection of "sentinel event" [23].
We focus on the study of databases (SDBs) which contain aggregated data (macro data) and which are able to support the work of statisticians, assuming that the disaggregated data from which they have been generated are no longer available (for reasons of safety and efficiency).
Manipulating SDBs generally involves changing their descriptive data (for instance, eliminating an attribute). When the descriptive data of an SDB are modified, the macro data must be modified in accordance with a well-defined algorithm. This algorithm is strictly dependent on the aggregate function which generated the macro data from disaggregated data. Users often find difficulty in understanding the semantics of this algorithm [10]. Our solution is to incorporate these semantics in the data model; in this way it is possible to free the user from having to understand or express the algorithm that is required to perfom the query.
The object oriented model we propose is centered around the concept of "data type". This model is standard enough and essentially follows the lines defined by Smalltalk-80 [7]. We have added the definition of an instance inheritance mechanism. It makes possible to share information at the instance level in a controlled way, since it assigns only to root instances of instance inheritance hierarchies the responsibility for dynamic evolution.
The paper is set out as follows: in the next section we shall discuss issues of data type management; in the following we

This work is partially supported by Consiglio Nazionale delle Ricerche under *MULTIDATA* Project and by the Commission of European Communities (EUROSTAT) under *DOSES* program

ACM-SAC '93/2/93/IN, USA
© 1993 ACM 0-89791-568-2/93/0002/0652...$1.50

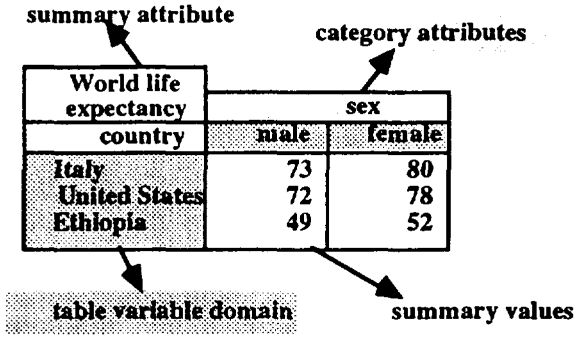
present our object-oriented model; then the metaschema of the statistical database system (ADAMS) embodying such concepts is shown and finally the advantages of our system are discussed.

Automatic Data Type Management

Macrodata, also called *Statistical Tables* (STs), have a very complex data structure. The elements that characterize a ST may be grouped into two classes [17]:

- Quantitative Data: i.e. the single *summary attribute* representing the phenomenon described by the ST; its instances, *summary values*, are the numeric values inside the ST; its type, *data type*, depends on the statistical aggregate function used to generate the summary values (e.g. 'average').
- Descriptive Data: i.e. the set of *category attributes* (also called *variables*) uniquely classifying summary values; for each category attribute a finite set of values, the *table variable domain* (generally strings of alphanumeric characters), is defined; the cartesian product of these domains represents the "set of points" in the *table space* for which a summary value can be defined.

An example of ST is shown in figure 1.



World life expectancy	sex	
country	male	female
Italy	73	80
United States	72	78
Ethiopia	49	52

Figure 1
A simple example of statistical table
(data type is "average"; source is [27]).

Users of SDBs generally perform two kinds of table processing:

- Table Management, that is manipulation of the category attributes and of the variable domains (e.g. summarization, which eliminates one of the category attributes);
- Data Analysis, that is performing calculations on summary values contained inside STs (e.g. carrying out the χ^2 test).

The former builds up new STs from STs already existing in the database and requires appropriate operations to allow table space manipulations. The latter performs statistical analysis on summary values and requires the use of statistical packages and programming languages, according to the analysis employed.
Here we consider only table management issues and focus on problems related to the operations to be performed on summary values when varying descriptive data of STs (for precise and formal definition of these operations see [22]). Table management is extremely useful in order to compare observed trends of phenomena with their expected trends or with trends which are typical of other geographical areas in order to find out abnormal morbid phenomena (the so-called "sentinel events"[23]).
In, for example, the case of summarization: this operator eliminates a category attribute. Let us take the case of a user studying indicators on hospital recovered patients . The starting table reports patients per "ward" (the table variable domain is: "anesthesiology", "intensive care unit", "cardiology", "pediatry") and "type of hospital" (the table variable domain is: "public", "private"). If we want to know the distribution of patients per "ward" a summarization has to be performed on "type of hospital". Let us take as a starting point the statistical table T_1 ("Number of patients recovered in hospital in Italy in 1990 per ward and type of hospital"), whose data type is an absolute value. The summarization described above gives the table T_2 by eliminating the attribute "type of hospital"

number of patients

	type of hospital	
ward	public	private
anesthesiology	8 424	382
intensive care unit	82 897	2 089
cardiology	175 133	33 162
pediatry	464 494	4 958

Table T1
Number of hospital recovered patients in Italy in 1990
per ward and type of hospital (source is [9])

number of patients

ward	
anesthesiology	8 806
intensive care unit	84 986
cardiology	208 295
pediatry	469 452

Table T2
Number of hospital recovered patients
in Italy in 1990 per ward (source is [9])

In this case the values of T_2 must be computed by summing the corresponding values of T_1 for every element of the table variable domain associated with "kind of hospital", applying to T_1 the computation function:

$$t_i^2 = \sum_j t_{i,j}^1$$

where:

t_i^2	is the generic instance of the summary attribute of T_2
i	is the generic element of the table variable "ward"
$t_{i,j}^1$	is the generic instance of the summary attribute of T_1
j	is the generic element of the table variable "type of hospital".

If the summarization on "kind of hospital" is applied to table T_3 (where the data type is an arithmetic mean), the result is table T_4. We note that this manipulation produces the same result as the previous one at the descriptive data level (i.e., to delete the attribute "type of hospital").

Mean length of hospital stay

	type of hospital	
ward	public	private
anesthesiology	7.1	2.0
intensive care unit	6.3	7.2
cardiology	8.8	14.9
pediatry	5.3	8.1

Table T3
Mean length of hospital stay (in days) in Italy in 1990 per ward and type of hospital (source is [9])

Mean length of hospital stay

ward	
anesthesiology	6.9
intensive care unit	6.3
cardiology	9.8
pediatry	5.3

Table T4
Mean length of hospital stay (in days) in Italy in 1990 per ward (source is [9])

In this case, however, the values of T_4 cannot be computed as in the previous case: the summary values in T_1 must be involved in the computation in order to obtain the correct values in T_4. This means that the following computation function has to be used:

$$t_i^4 = \frac{\sum_j t_{i,j}^3 \, t_{i,j}^1}{\sum_j t_{i,j}^1}$$

where:

t_i^4	is the generic instance of the summary attribute of T_4
i	is the generic element of the table variable "ward"
$t_{i,j}^3$	is the generic instance of the summary attribute of T_3
$t_{i,j}^1$	is the generic instance of the summary attribute of T_1

From this example we note two critical points in the table management.
The first is that summary values processing is strictly dependent on the data type of the ST being manipulated: the same kind of table space manipulation can require several different algorithms (called *resolution algorithms*) which depend on the data type.
A second major point is that generally it is not possible to know 'a priori' which data types must be defined in a SDB. In fact, statistical activity typically involves the identification of new aggregate functions to generate indicators which are valid for specific areas of investigations. Since the set of data types a SDB is required to manage can always be enlarged when new needs arise, a SDB has to have the possibility of being extended to deal with new data types.
In any case, when manipulations on descriptive data have to be performed, algorithms for computing new summary values must be known at run time. Such algorithms are neither simple nor commonplace. Therefore, the simple solution of getting the user to specify them (as proposed by several authors [19], [12], [26], [14] and [6]) poses some difficulties (e.g. users have to know a programming language, similar algorithms need to be specified again and again with slight variations). In fact, users often find it difficult to understand the semantics of resolution algorithms; furthermore, once the user understands the resolution algorithm it is complex and/or lengthy to express it in these query languages [10].
The approach we propose is to transfer into SDBs the knowledge needed to achieve automatic table manipulation and to associate this knowledge with of STs data type (*automatic data type management*). Such a solution presents the advantage of a more flexible and compact definition of data types, since the various manipulation operations are not defined for a specific ST but for a class of STs with the same data type [18].
Various proposals have been made which follow approaches similar to ours. The STRAND query language [10] is inconvenient because one has to define 'a priori' the data type management procedures for each single ST. The G-relations [25] suggests the possibility of implementing data type management procedures, but only for the summarization operation. In [8] operations that work on STs use 'hidden information', but it is not clear from the paper how such information is used; moreover, these operations are not performed if the data type of the input ST to the manipulation is different from 'absolute value'. The limitation [16] is that the approach is valid only for the manipulation of the simplest data type to be processed (i.e. absolute value). In general all the proposals found in the literature are rather generic, failing to define precisely how it is possible to obtain a true automatic data type management.

Our proposal is based on the use of an object-oriented approach, which has a number of significant advantages from the database point of view (e.g. encapsulation, inheritance, overriding: see, for example, [2]). In particular, with our approach it is possible for the user to specify the kind of manipulation to be executed on a given ST independently from its data type; in fact the system takes care of calling the correct algorithm, depending on the specific data type of the ST. In order to ensure to the user a table management which is independent from the data type (*logically independent statistical table manipulation*), the database has to know, for each data type and for each particular operation, the resolution algorithm for calculating the output summary values. The various algorithms for calculating the summary values as a function of data type are presented and discussed in [4].

The Concepts of Our Object-Oriented Approach

There is no general agreement on what exactly an object-oriented model is and different authors use the basic concepts (like class, instance, inheritance,...) in slightly but significantly different ways.

Our modeling of STs is based on the concepts of class, subclass, superclass, instance, method, message, class variable, instance variable and (simple) inheritance as they are defined in Smalltalk-80 [7]. The rationale for our choice is the fact that Smalltalk-80 is probably the best known object-oriented system and one of the basic references for everyone working in this field. At the same time, it offers a very clean and homogeneous set of definitions for all the most important and used concepts of the object oriented field.

A class defines the static structure and the dynamic behaviour of its instances. The static structure is described by specifying the instance variables of each instance. The dynamic behavior is described by specifying the actions to be executed by each class instance in response to requests to do actions (messages) from other instances. Instance variables may be manipulated only by the instance which owns them. Class variables can be also defined, which are variables owned and managed directly by the class itself.

A class may have instances, that is objects which have the structure and the behaviour specified in their class definition. The behavior of an instance (and of all the instances belonging to the same class) is completely defined by the set of messages it responds to. Instances are dynamically created and destroyed.

A subclass of a class inherits its definitions of instance structure and behaviour and possibly specializes them (class inheritance). In this framework, we consider only simple inheritance, that is inheritance from at most one class, called a superclass.

For the purposes of ST modeling, we introduce an instance inheritance relationship between instances. This means that two instances which belong to two classes with a class inheritance relationship (i.e. one class is directly a subclass of the other) may also have an instance inheritance relationship between them. In other words, just as every class has a superclass from which it inherits, every instance may have a superinstance from which it inherits. The superinstance of instance A, if it exists, is necessarily an instance of the superclass of A's class. Also for the instance inheritance relationship we shall consider only simple inheritance, i.e. an instance can inherit from at most one superinstance. In such a way a number of instance inheritance hierarchies may be built up, which always run parallel to class inheritance hierarchy. The instance which is the root of an instance inheritance hierarchy is called exemplar. In the example of figure 2, there are two instance inheritance hierarchies, where A1 and A2 are the two exemplars.

The instance inheritance mechanism we have introduced is more restrictive with respect to classical object oriented approaches, since it imposes tighter constraints on the dynamic evolution of instance variable values. Namely, when two instances are in a direct instance inheritance relationship, they are bound to have the same values for instance variables which are common to both classes.

Suppose, for example, that A is an instance of class CA, B is an instance of class CB, CB is a subclass of CA, and an instance inheritance relationship between A and B has been defined. Then, since CB is a subclass of CA, its instances have the same structures defined in CA (and possibly additional structures defined in CB). Moreover, instance B, besides having the same structures as instance A, is constrained to assume the same values.

Figure 2 shows a fragment of a class hierarchy and of an instance inheritance hierarchy illustrating this constraint: the dependency between instance variables is shown graphically.

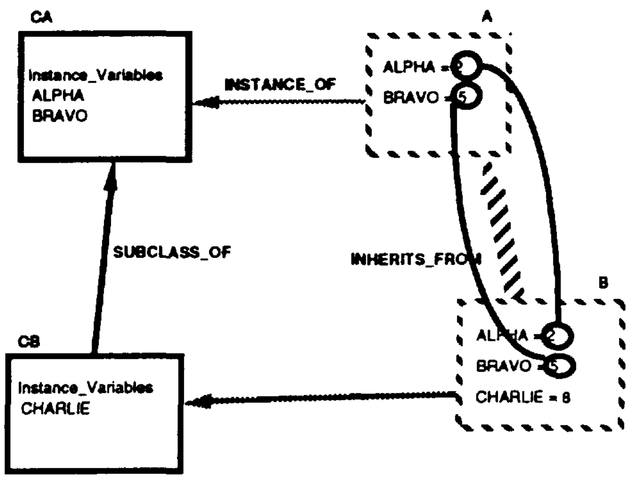


Figure 2
A fragment of a class hierarchy
and of an instance inheritance hierarchy

When an instance receives a message requiring that one of its variables is modified, such a message is handled in different ways, depending both on the initialization of the variable and on whether or not the instance has inherited that variable from a superinstance.

When the variable to be changed is an inherited one, the receiver instance, in fact, handles the message by delegating the task to its superinstance, and possibly to the superinstance's superinstance, up to the exemplar for that specific INHERITS_FROM hierarchy. It is the exemplar that now manages the way in which changes really happen. If the instance which originally received the message has a non-initialized variable then the receiver instance and all its subinstances have the variable changed in the same way. Otherwise, there will be a superinstance of the receiver instance (called here top-instance, possibly the exemplar itself), so that the variable is initialized and its superinstance is not. In this case, the top instance and all its subinstances will receive the same change.

If the variable is not inherited, the receiver instance itself will instead activate the appropriate method for answering the

message. After having executed this method, the instance advises instances of which it is a superinstance so that they can update the changed variables to the same value.
In such a way, for each superinstance/subinstance chain of N instances there always exists an index K ($1 \leq K \leq N$) so that each instance from K to N has the variable assigned to the same value, while each instance from 1 (the exemplar) to K-1 has the non assigned (non initialized) variable.
In this way, a message may be sent to an instance for updating one of its inherited variables. The message climbs up to its top instance to be executed and to produce the change of the specific variable in the top instance and in all its subinstances.
Such a mechanism makes it possible to share information at the instance level in a controlled way, since it only assigns to exemplars the responsibility for the dynamic evolution of the variables they defined. The reader interested in the information sharing mechanism in the object oriented approach using exemplars should also consult [13], [15], [24], [1].

ADAMS' Metaschema

A prototype of the ADAMS system has been tested at Consiglio Nazionale delle Ricerche, Istituto di Studi sulla Ricerca e la Documentazione Scientifica on a Macintosh IIx using the MacApp environment and the Object Pascal language [5]. We shall discuss the ADAMS system with an example: the study of indicators on hospital recovered patients.
The model we propose has four class levels. Let us now examine in a more detailed way the structure of each of the classes (see also figure 5).
Level 1. TABLE_SCHEMA defines the characteristics of STs which are independent of data type. All the STs which have the same summary attribute, the same category attributes, and the same category attribute domains are defined by the creation of an instance of the class TABLE_SCHEMA. That is, TABLE_SCHEMA defines the following instance variables:

- SUM_ATTR: this variable defines the semantics of data contained in a ST; it therefore contains the name of the examined phenomenon;
- CAT_ATTR: this variable defines the schema used for classifying data contained in a ST; it therefore contains the category attribute names, the definition domain of each of the category attributes, the real values taken within the definition domain by each of the category attributes (table variable domains). It is therefore structured as a set of triples <category_attribute_name, category_attribute_domain, table_variable_domain>. Note that category attributes are usually set-valued. At this level, clearly only the names and the domains of the category attributes will be known.

Regarding instance methods, TABLE_SCHEMA contains only those methods used for manipulating the defined set of category attribute names and domains.
The specification of category attribute domains is made by choosing one of the following predefined domains: integer, real, boolean, char, string, date & time, integer-interval.
In our example, a category of STs relative to employment by sex, economic branch, and age is defined by creating the instance OCC of the class TABLE_SCHEMA as follows:
SUM_ATTR: hospital recovered patients
CAT_ATTR: { (ward, String, -), (type of hospital, String, -)}
Level 2 TABLE_SPACE does not add instance variables, but specifies the table variable domains for the variable CAT_ATTR; that is, it specifies the table space of STs. In particular the method 'New' for this class explicitly requires the assignment of these values. The instance at this level is put into the INHERITS_FROM relation with the instance of TABLE_SCHEMA which represents the whole category of STs.
In the example there is an instance of TABLE_SPACE which is an instance inheritance from exemplar OCC:
CAT_ATTR: { (ward, String, {anesthesiology, intensive care unit, cardiology, pediatry}), (type of hospital, String, {public, private}}.
Level 3 TABLE_STRUCTURE adds structures for the physical manipulation of STs. In particular it defines the following class variable:

- DT: this variable specifies the data type of the ST. It is not an instance of this class but only of the subclasses of TABLE_STRUCTURE, which use it for recording data type value.

Moreover, class TABLE_STRUCTURE adds the following instance variable:

- FUN_TAB: this variable provides a way of accessing the object which contains the correspondence between points of the table space and the corresponding value of the summary attribute.

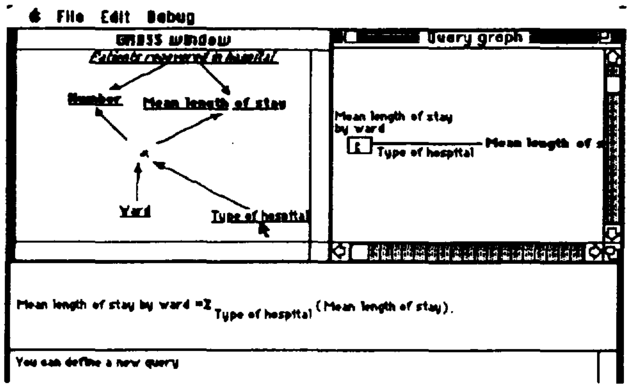
Regarding instance methods, TABLE_STRUCTURE adds the specification of methods for computing, given the values of category attributes, a virtual index to pass to the object identified by FUN_TAB for accessing the physical structure which contains summary attribute values (This physical structure is stored by multidimensional matrix). As far as instance methods are concerned, TABLE_STRUCTURE adds methods which define the remaining basic manipulation operations for aggregation of STs [22]; for example Summarization. These methods are not completely specified, since at this level it may not be possible to thoroughly define them (they are data type dependent). The aim of TABLE_STRUCTURE is to provide, via variables and methods defined at this level for the logical representation of FUN_TAB, a support for separation between the conceptual (i.e. table structure oriented) and logical (i.e. data type oriented) levels of representation of FUN_TAB. Therefore, every instance of TABLE_SPACE has one and only one subinstance, which is an instance of TABLE_STRUCTURE, containing the methods used for table manipulations, common to all STs with different data types.
Our example defines an instance of TABLE_STRUCTURE (OCC_1_T); it is put into an instance inheritance relationship with instance OCC_1.
Level 4 Subclasses of TABLE_STRUCTURE are the classes which specify the behavior for the different data types. All these subclasses inherit class variable DT and assign it a value. Moreover, they define additional structures for executing table manipulation operations on STs; some of these additional structures depend on the data type of the ST [4].
Each subclass of TABLE_STRUCTURE re-specifies methods introduced by TABLE_STRUCTURE for the basic aggregation operations for STs.
New subclasses of TABLE_STRUCTURE may be added by the Database Administrator in order to take into account specific

kinds of statistical data or for the purpose of modelling particular statistical functions.
In our example, a specific ST is defined as an instance of a suitably defined subclass of TABLE_STRUCTURE which is associated with a specific data type. Such an instance is then put into an instance inheritance relationship with OCC_1_T. Namely, we defined two instances (O_1, O_2) relative to the homogeneous set OCC_1. The instance O_1 refers to the class AVERAGE_TYPE, the instance O_2 refers to the class ABSOLUTE_TYPE.
The resolution algorithm for performing manipulation operations is encapsulated in the object AVERAGE_TYPE. Such manipulation needs a ST with data type absolute value and therefore the instance O_2 is the reference table of the instance O_1.

Discussion

The main tasks performed by an SDB user are:
- to express table manipulation as the trasformation of the descriptive part of ST;
- to set up queries, that is, to build complex queries from elementary ones;
- to browse through schema and queries;
- to define subsets of tables of interest;
- to display and format the result.

The user does not have to insert, update or delete STs; this is the task of the Database Administrator.
With an example of a working session we illustrate how the ADAMS system allows the user to perform browsing and querying. The database schema is represented for the user by means of the GRASS model (Graphical Approach to Statistical Summaries) [21]. This formalism is used to represent STs graphically by means of a direct acyclic graph.
ADAMS supports different manipulation paradigms following a conversational style and tailored to different profiles of users [5]. They are classified adopting the "user cube" approach [3].



Mean length of stay by ward =$\Sigma_{\text{Type of hospital}}$ (Mean length of stay).

You can define a new query

Figure 3
ADAMS' multi-window graphical interface

Those with a good technical knowledge of applicative domain employ a keyword language (STAQUEL*), but there also the possibility of building incremental quieries by means of VISTA. VISTA is based on an operation graph where each operand is a query element. In this way a non expert user is facilitated in building complex queries. The multi-window graphical interface is depicted in figure 3.
In our approach, which ensures the user a Table Management independent from the data type (i.e., logically independent statistical table manipulation), the database has to know, for each data type and for each particular operation, the resolution algorithm for calculating the output summary values. It is possible because there is the class AVERAGE_TYPE, where the method summarization is specified. We note that such classes may define the following instance variable: REF_TAB refers to a reference ST, that is to an instance of other subclasses of TABLE_STRUCTURE, whose ST is necessary for the execution of table manipulations (in the example the object O_2).
Note also the query displayed in text format by STAQUEL* language and the GRASS view of the Logical Schema rationalized after the automatic layout command has been given.

Using simple visual interaction with the icons representing objects of the ADAMS system, the user can activate the relevant methods, such as the 'GET_STRUCTURE' applied to a query type object resulting in the display of its structure.
The query languages which result from this object oriented system enable table management to be carried out without having to take into account of summary values and data types. This means that table management refers only to the elements which form the schema of the database.
In the following, we compare two statistical query languages: STAQUEL* and Summary_Table_by_Example [20]. Our aim is to show, by means of a typical user query, that languages based on this object oriented system are easier to use for the end user than the traditional query languages where the user must specify the resolution algorithm.
Let us take the case of a user studying the distribution of patients recovered in hospital (tables T1 and T2). The user performs the query (expressed by Summary_Table_by _Example) of figure 4 to eliminate the category attribute "type of hospital"
If the user employs the query language STAQUEL*, the query is the following one:

$$\Sigma_{\text{type\_of\_hospital}} \text{(patients\_recovered)}.$$

The query expressed by STAQUEL* is both simpler and more compact than one expressed by Summary_Table_by_Example, as well as closer to the statistician's way of operating.
The above example shows that, with the traditional query languages, the usual simple queries for the table manipulation (the summarization) is expressed in an extremely complex way, while an important characteristic for a user friendly interface is the simplicity of use. Our approach easily permits the expression of usual statistical queries.
It can be seen that the user does not have to:
1) express the formula for the calculation of the new summary value;
2) know the existence of other data, even though they are necessary to calculate the new summary values.

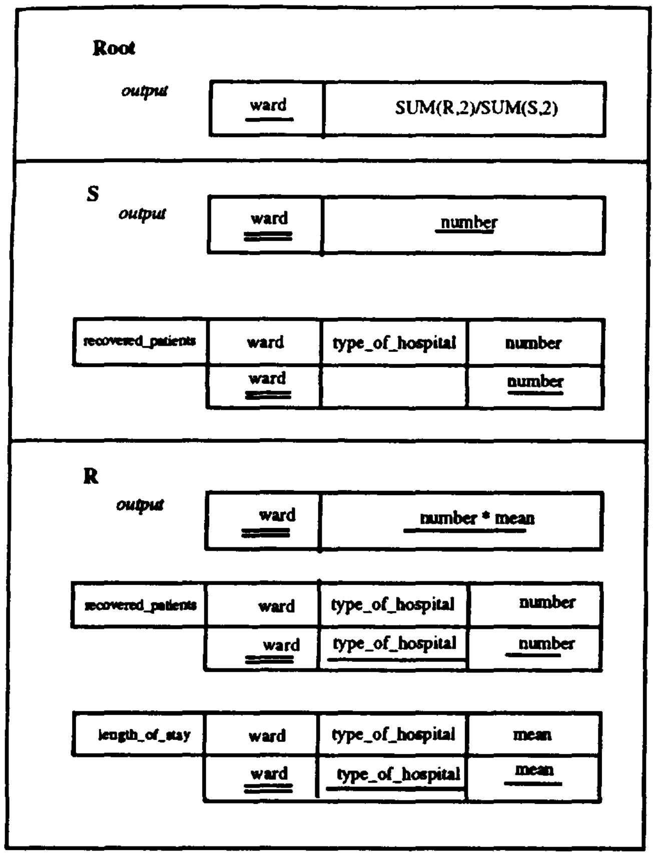


Figure 4
An example of application of the statistical query language Summary_Table_by_Example [20]

In fact, the user manipulates, by STAQUEL*, the STs at metadata level; for example, he/she performs the summarizaton operation to specify only the category attribute (an element of the intensional aspect of metadata). But the system has to know, for each data type and for each affected operator, the resolution algorithm for calculating the output summary values. An apparent limitation of the automatic data type management approach is that it only considers the data type defined 'a priori' by the DBA. However, the limitation is not as strong as it might seem in that:

1) the data type generally defined in the SDB and of which the resolution algoritms are kown, cover almost all the data types made available to the users;
2) in planning the SDB, the DBA already knows beforehand the resolution algoritms which have to be provided in order to enable the stored STs to be manipulated.

Conclusions

ADAMS allows to define and manage ST. The system uses context-driven editors and represents operations and metadata by means of the icon-graphical paradigm. Three alternative interaction modalities are provided. An editor using visual languages is available to the novice/casual users (the user defines.mathematical links between statistical tables). Interaction may also be performed using the direct manipulation approach (the user specifies directly on the GRASS* graph her/his manipulation), whereas, for expert users, a key word language is implemented. Independently from the approach adopted by user for querying SDB, the system displays all the three different query representations. Therefore the user is able to verify the system's interpretation of his query.
The object-oriented model for ST representation allows the user to integrate with the "closed world" of databases the features of the "open world" of statistics. Even if this model is not directly utilizable by the user, it allows him to express his table management queries without having to worry about the algorithms to compute the summary values.
This model represents a starting point to capture statistical knowledge in such a way as to simplify user interaction with the system because his attention is directed towards the semantics of the statistical operation and not towards the procedures for implementing it.
In addition, it allows to express table management operations at table space level. If, for instance, summarization is concerned, one has only to indicate the category attribute to be eliminated.
This implies that it is possible studying table management operators properties and relative algebra (completeness, reachability,...) independently from the data type. These properties are the formal groundwork for defining an interface based on logical independence that is aimed at simplifying man-machine interaction [18].

References

[1] J.Almarode, "Rule-based delegation of prototypes", Proc. OOPSLA'89, Oct.1989.
[2] F.Bancilhon, "Object-oriented database systems", Proc. VII ACM SIGACT/SIGMOD/SIGART symposium on PODS, 1988.
[3] W.W.Cotterman, K.Kumar, "User Cube: A Taxonomy of End Users", Communications of the ACM, 32,11, 1989.
[4] G. Falcitelli, L. Meo Evoli, E. Nardelli, F.L.Ricci, "The Mefisto* model: an object oriented representation for statistical data management", Proceed. of the Data Analysis and Learning Symbolic and Numeric Knowledge, 1989 .
[5] F. Ferri, P. Grifoni, L. Meo-Evoli, F.L. Ricci, "ADAMS: an aggregate data management system with multiple interaction techniques", Database and expert systems applications, Proceed. of the DEXA 91, Springer-Verlag, 1991.
[6] S.P.Ghosh, "Statistical relational tables for statistical database management", IBM Res. Lab., San Jose, CA, Tech.Rep. RJ 4394, 1984.
[7] A.Goldberg, D.Robson, "Smalltalk 80: The language and its implementation", Addison-Wesley, 1983.
[8] H.Ikeda, Y.Kobayashi, "Additional facilities of a conventional DBMS to support interactive statistical analysis", Proceed. of the I° Intern. Workshop on Statistical Database Management, Menlo Park, California, December 1981.

[9] Istituto Internazionale per gli Sudi e l'Informazione Sanitaria, "Statistiche Sanitarie", 1992 (in Italian).

[10] R.R.Johnson, "Modelling summary data", Proceed. of the International Conference on Management of Data, ACM-SIGMOD, Ann Arbor, Michigan, April-May 1981.

[12] A.Klug, "Equivalence of relational algebra and relational calculus query language having aggregate functions", Journal of the ACM, Vol.29, N.3, July 1982.

[11] J.C.Kleinman, "The Continued Vitality of Vital Statistics", editorial, American Journal of Public Health, 72 (2), 1982.

[13] W.R.LaLonde, D.A.Thomas, J.R.Pugh, "An exemplar based Smalltalk", Proc. of 1986 Conf. on Object Oriented Programming Systems, Languages and Applications, 1986.

[14] L. Lakhal, R. Cicchetti, S. Miranda, "RTL: a relational and table language for statistical databases", Proceed. of the 2° symposium on mathemetical fundamentasls of database sytems, Lecture Notes in Computer Science, 364, Springer-Verlag, 1989.

[15] H.Lieberman, "Using prototypical objects to implement shared behaviour in object oriented systems", Proc. of OOPSLA 86, 1986.

[16] F.M.Malvestuto, "Answering queries in categorical data base", Proceed. of A.C.M. PODS Conference, 1987.

[17] L. Meo-Evoli, M.Rafanelli, F.L.Ricci, "The relational model and the statistical tables", 7th Statistical Software Newsletter, vol. 18, N. 3, December 1990.

[18] L. Meo-Evoli, F.L. Ricci, A.Shoshani, "On the semantic completeness of macro-data operators for statistical aggregation", Proceed. of the VII° Intern. Working Conference on Scientific and Statistical Database Management, 1992.

[19] G.Ozsoyoglu, Z.M.Ozsoyoglu, V. Matos, "Extending relational algebra and relational calculus with set-valued attributes and aggregate functions" ACM Trans. Database Systems, 12, 4, 1987.

[20] G.Ozsoyoglu, V.Matos, Z.M.Ozsoyoglu, "Query processing techniques in the Summary-Table-by-Example database query language", ACM trans. Database Systems, 14, 4, 1989.

[21] M.Rafanelli, F.L.Ricci, "Proposal of a logical model for statistical data base", Proceed. of the II° Internat. Workshop on Statistical Database Management, 1983.

[22] M.Rafanelli, F.L.Ricci, "Mefisto: a functional model for statistical entities", IEEE Trans. on Knowledge and Data Engineering, October 1993 (in press).

[23] D.D.Rutstein, R.J.Mullan, T.M.Frazier, W.E.Halperin, J.M.Melius, J.P.Sestito, "Sentinel Health Events (Occupational): A Basis for Physician Recognition and Public Health Surveillance", American Journal of Public Health, 1983.

[24] L.A.Stein, "Delegation is inheritance", Proc. of 1987 Conf. on Object Oriented Programming Systems, Languages and Applications, 1987.

[25] S.Y.W.Su, "SAM*: a semantic association model for corporate and scientific-statistical databases", Information Sciences, Vol.29, N.2 and 3, May and June 1983.

[26] A.U.Tansel, M.E.Arkun, G.Ozsoyoglu, "Time-by-Example query language for historical databases", IEEE Transactions on Software Engineering, Vol.SE-15, N.4, April 1989.

[27] J.A.Toolley, L.A.Carle, U.S. News & World Report, March 1989.

[28] World Health Organization, International Classification of Diseases. Manual of the International Statistical Classification of Diseases, Injuries, and Causes of Death, Voll.1 and 2, 9th Revision, Geneva, 1977.

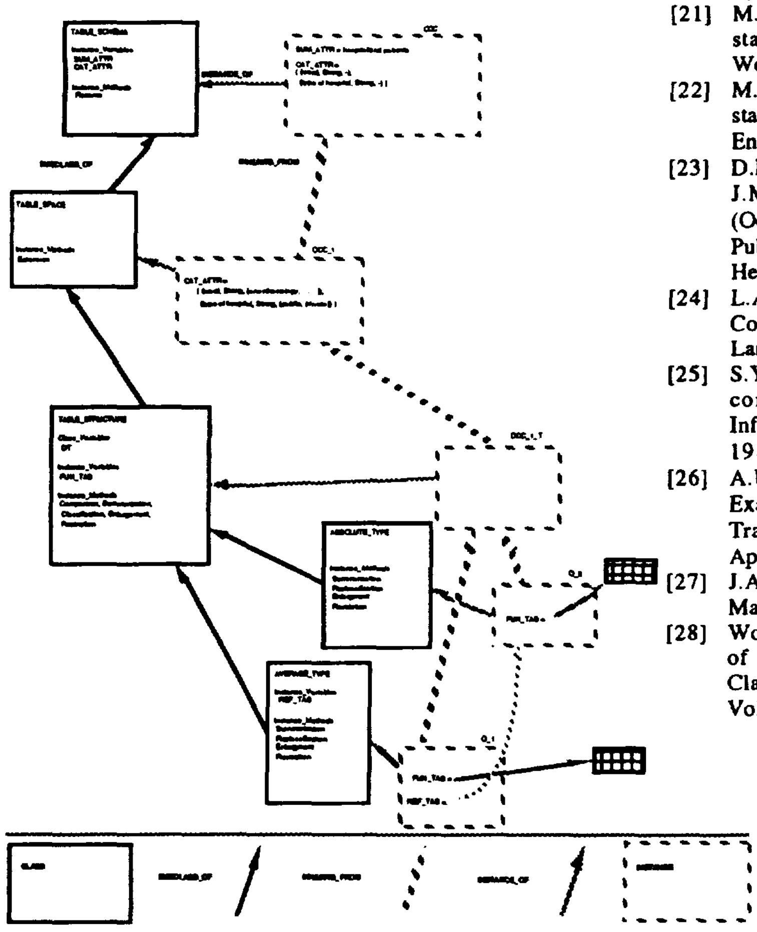


Figure 5
ADAMS' Metaschema

Address requests for reprints and extended version of this paper to F.L. Ricci, ISRDS-CNR, V. C. de Lollis 12, 00185 Rome, Italy (fax: +396 4463836; e-mail: isrd@vm.cnuce.cnr.it)