

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NRL Report 7905	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle)  DATA SECURITY IMPLICATIONS OF AN EXTENDED SUBSCHEMA CONCEPT		5. TYPE OF REPORT & PERIOD COVERED An interim report on a continuing NRL problem.
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s)  Frank A. Manola Stanley H. Wilson		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Research Laboratory Washington, D.C. 20375		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NRL Problem B02-24 Project RF-21-222-401-4361
11. CONTROLLING OFFICE NAME AND ADDRESS Department of the Navy Office of Naval Research Arlington, Virginia 22217		12. REPORT DATE  July 15, 1975
		13. NUMBER OF PAGES 17
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report)  Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)  Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
Access control	Data structures	Logical structure
CODASYL	Data Base Task Group	Schema
Data base design	Data base management system	Strongly typed languages
Data transformations	Data base management system architecture	Subschema
Data security	Extensible languages	
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)		
<p>Modern data base system architectures, such as those based on the CODASYL Data Base Task Group (DBTG) specifications, stress the idea that the users of the system should interact with a logical description of that portion of the data base with which they are concerned, called a subschema, which is derived from a logical description of the data base as a whole, called a schema. One of the benefits of this architecture is its ability to provide enhanced data security, since the mapping from schema to subschema may conceal information from the user. Use of this architecture to enhance data security was studied with respect to schema, subschema,</p>		

20. ABSTRACT (Cont.)

and design and implementation of application programs. Instances of such use include both that in the existing DBTG specifications and some proposed extensions to that architecture.

## CONTENTS

INTRODUCTION .....	1
DATA SECURITY .....	2
A MODEL OF THE SCHEMA/SUBSCHEMA ARCHITECTURE .....	3
DATA SECURITY IN THE SCHEMA/SUBSCHEMA ARCHITECTURE .	4
A SIMPLE DATA SECURITY PROBLEM .....	5
CODASYL DATA SECURITY FEATURES .....	8
USE OF DERIVED DATA .....	9
LIMITATIONS ON THE CODASYL SPECIFICATIONS REQUIRING EXTENSIONS .....	10
EXTENSIONS TO THE CODASYL ARCHITECTURE .....	11
IMPLEMENTATION .....	12
CONCLUSION .....	13
ACKNOWLEDGMENTS .....	13



## DATA SECURITY IMPLICATIONS OF AN EXTENDED SUBSCHEMA CONCEPT

### INTRODUCTION

One of the most important concepts in the CODASYL data base language specifications (1,2) is that while a logical description of the entire data base, called the *schema*, is important for use in the administration function of controlling and optimizing the entire data base, the user of a data base should have his own logical description of that portion of the data base with which he must be concerned, called the *subschema*. The user can interact with this subschema via program or terminal. The schema/subschema architecture has important data security ramifications, since it provides the possibility of designing the subschema so as to conceal from the user information in the data base to which he is not allowed access. Such a design, in effect, presents the user with a virtual data base which reflects not only that user's data requirements, but also his data security constraints.

The data base which the user sees in his subschema is the result of two transformations: first, the transformation from the storage on physical devices to the logical data base, defined by the schema; second, the transformation between the logical data base and the user's view, defined by the subschema. Because we are interested in logical data security and because, of the two, only the subschema transformation is a logical transformation, it is the only one that will be addressed in this report. In the CODASYL specifications, only very limited transformations between schemas and subschemas are permitted—omitting schema data from the subschema, or renaming schema data. As a result of this and certain dependencies between the schema and subschema, the concealment which can be accomplished via the subschema has been somewhat limited. The amount that can be accomplished, however, has been generally underestimated; and, consequently, the data security possibilities have also been underestimated.

More recent work (3-5), has suggested the possibility of more complex transformations between schema and subschema and has also suggested that such transformations may allow for enhanced data security. This has been reinforced by a recent paper by Minsky (6), who not only suggests that data security requires the ability to form "abstractions" and to define appropriate operators on them, but further contends that data security cannot be enforced without the ability to control the internal actions of the user's program.

This report attempts to discuss briefly the problem of data security in a schema/subschema architecture from a broad perspective. While arguing for facilities for more complex schema-to-subschema transformations and the ability to define abstractions and appropriate operators, we contend that these facilities should be used to provide subschemas which simultaneously reflect user data needs and data security constraints and which keep the security responsibility within the data base management system (DBMS) software—rather

---

Note: Manuscript submitted May 1, 1975.

than to attempt to control the internal actions of the user program. This is important if the widest range of programming languages is to be used in connection with data base management systems. Some facilities within the existing CODASYL specifications that allow subschema designs along these lines are also discussed. A general familiarity with the CODASYL specifications is assumed.

## DATA SECURITY

Ideally, to maintain security we must prevent a user's extracting information from, or inserting information into, the data base in an unauthorized manner. We may term this the *information security problem*. However, data base systems do not deal with information, but rather with data. Data base users gain information by extracting data from a data base system, as well as from other sources. Thus, the *data security problem*, which is the problem of keeping a user from misusing the data base, is only a subset of the entire information security problem, which includes such things as preventing a user's deducing information to which he is not authorized from data in the data base to which he is authorized. This subject, addressed by work such as Ref. 7, is outside the scope of this report. Similarly, we will not consider the *authentication problem* (the verification that a user is who he says he is and the association of that user with a defined set of access rights) or the *operating system integrity problem* (the verification that the operating system performs properly). Rather, we will assume that a user is always associated with his proper subschema, that access to the data base is always through authorized channels, and that physical data, program modules, work spaces, and such, whose protection is essential for the reliable operation of the data base system security measures, can be protected. Thus, we will interpret the data security problem in this context as being the control of the data to which the user ultimately gains access by means of any programs he may invoke with respect to the data base (e.g., DBMS, operating system, application program, or query interpreter).

The type of data security with which we are particularly concerned is logical data security, i.e., the protection of the data with which the user interacts from unauthorized access. This means that we are specifically concerned with the protection of data items, records, and more abstract objects, as opposed to disk tracks or blocks of main memory. Likewise, we are not concerned with physical attributes of the data in determining the security criteria (e.g., the physical address of the data on the disk), but we are concerned with logical attributes. Naturally, these physical units must be protected too, but we are concerned with statements of security criteria expressed with respect to the logical data units which the users handle and the legal operations on those data units.

The logical attributes we are concerned with may be based on content or on logical structure. In the CODASYL architecture, the qualifications, "PERSONNEL records with the DEPARTMENT data item equal to PURCHASING," is an example of a content-based attribute, and "PERSONNEL records in the FEMALE set relationship" is an example of a structure-based attribute.

Before describing the security mechanism, we will present a model of the system architecture within which the mechanism will operate.

## A MODEL OF THE SCHEMA/SUBSCHEMA ARCHITECTURE

That portion of the schema/subschema architecture of concern in this report is shown conceptually in Fig. 1. The user interprets the real world in terms of a set of abstract objects  $u_i$  and conceptual operations on those objects. The data in the data base presumably

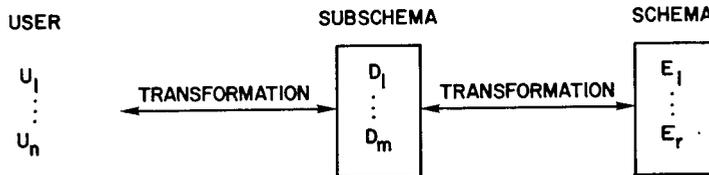


Fig. 1 — Schema and subschema architecture

is useful to some population of users; hence, the user interacts with the data base to extract information which he needs or to store information which he has about the real world for his own use and that of others. This cannot be done directly because the data base system does not operate in terms of the user's conceptual objects and operations, but rather in terms of data objects and operations. Thus, to set up and use a data base, the user must perform a transformation from his conceptual objects and operations to the objects and operations of what he perceives the data base to be. The relevant attributes of each conceptual object must first be described in terms of legal attributes of data objects; then, in order to model some action or sequence of actions in the real world, the user must transform these actions to a sequence of operations allowed by the data base and submit these operations to the data base (for example, a program or sequence of query language statements). The result (the composition of the operations invoked) presumably corresponds to some real-world result.

The user's perception of the data base is defined by his subschema, and he interacts with the data base by invoking a sequence of operations with respect to the data objects  $d_i$  in his subschema. Data to be transmitted to the data base must be placed in a subschema data object, and data transmitted from the data base will be materialized in one of these data objects. The data base itself is defined by the schema, which provides a set of data objects  $e_i$  and a set of operations on them. We assume that a central authority, the data administrator, defines both the schema and the subschemas: the schema to reflect the data requirements of all data base users, and the subschema to reflect the data requirements of a particular data base user or set of data base users.

Just as a transformation must be performed by the user between his conceptual objects and operations and the subschema objects and operations, so the data base system implementation must perform a transformation between the subschema objects and operations and the schema objects and operations. This transformation may be defined in many ways. In the CODASYL specifications, the transformations allowed are so simple that they can be specified implicitly by examination of the differences between the schema

and subschema. However, more complex transformations would require some combination of definitions in the schema, in the subschema, or in a separate definition. Because the purpose of the subschema is to define the user's perception of the data base, these transformations, if designed and implemented properly, will permit only objects defined in the subschema to be transmitted to and from it. Thus, the transformation between his conceptual data and the subschema data is the only one with which the user should be directly concerned.

Viewing the overall transformation from schema data to the user's conceptual data, we see that there are choices of (a) how much transformation must be performed and (b) where the bulk of the required transformation is to be performed. For example, the complexity of the user's transformation from real world to subschema naturally depends on the degree to which the objects and operations in the subschema match his conceptual objects and operations. If there is little correspondence, then the transformation will be extensive, which we may interpret as requiring a large program or sequence of query language statements and extensive computations based on the resulting data, since much of the burden of performing the transformation is placed on the user. If there is substantial correspondence, then the mere retrieval or storage of the data may be sufficient to satisfy the user's requirements, since no transformation may be required. For example, the user may require the average age of all male persons in the United States. If one of the subschema variables  $d_i$  happens to contain that data directly, a simple retrieval of that value will satisfy the request. If, on the other hand, the subschema contains only the value of age for each person, and one such record exists for each person in the United States, then the user will have to retrieve all the records for male persons, extract the age, and compute the average.

Similarly, the difficulty of the transformation between the subschema data and the schema data depends on the degree to which the objects in the subschema match the objects in the schema. In this case, the fact that the schema must reflect the data requirements of *all* users, rather than only a single user, is an additional complication. Continuing with the example above, suppose that we choose to materialize for the user a subschema variable containing the average age of all males in the United States. Again, we are faced with a choice. If one of the schema variables  $e_i$  happens to contain that data directly, then again the transformation is trivial. On the other hand, if only this information is available, it may be inadequate for other users who require information on specific people and on females as well as males. So, as an alternative, we may choose to define individual *person* records in the schema and perform the complex transformation required to materialize the average male age for that particular user in the schema/subschema transformation. Alternatively, we may define both and accept a possible penalty for redundant storage.

## DATA SECURITY IN THE SCHEMA/SUBSCHEMA ARCHITECTURE

In the context of the previous discussion, the application of logical data security constraints to the data base may be viewed as the application of a transformation from what

is defined in the schema to what the user is permitted to access. As with the transformation to produce the subschema, the degree to which the schema data matches the data the user is permitted to access determines the complexity required in the security transformation. At the same time, the degree to which the subschema data matches the data to which the user is permitted access on the basis of his security constraints determines the amount of the security transformation implemented simply by means of the transformation which produces the subschema data. This implies that if we can carefully design the operations and data objects of the subschema so as to match not only the user's data requirements but also his data security constraints, we can simultaneously satisfy the demands of security and those of data independence and simplicity.

Our ability to perform such a design is not, however, unlimited. It depends on the degree to which we can specify data objects in the schema and subschema, operations on the subschema objects, and the schema-to-subschema transformations. No system allows complete generality in this regard, and the limitations of particular systems determine the degree to which such designs can be accomplished. Generally, the user's perception of the data base is limited to subsets, or simple transformations, of standard data base objects—such as data items and records. The consequence of a design not tailored to the user is that the user assumes more of the responsibility for the transformation from the data base to his conceptual objects and operations. When this occurs, usually the user must be given access to more data in a manner which is more difficult to control than if the data base system itself were responsible for the transformation.

This suggests that the design principle for a subschema should be that its objects and operations be as close to the conceptual objects and operations of the user as possible, and that the security constraints should be imposed in the transformation to the subschema so that the user's view of the data includes these constraints. This minimizes the total amount of transformation which must be performed and minimizes the amount of unnecessary data which must be given to the user, because the data base system performs most of the necessary transformation. This principle provides two additional advantages. The transformation is implemented below the subschema level; hence, the security function is more centralized and may be easier to protect. Also, the user is not restricted to a particular programming language that must be used when accessing the data base. Additional advantages are that the user becomes independent of the exact method used in implementing the security transformation, which may need to be changed as the data base itself changes; and, as additional declarative facilities for directly specifying security constraints become available, it may be possible for the system to optimize the data base structure, taking into account each user's security constraints without impacting the users.

## A SIMPLE DATA SECURITY PROBLEM

To study the use of schema/subschema transformations in a data security problem using these principles, let us examine one of the examples from Minsky [6]. A file of personal data records, with attributes  $a_1, \dots, a_k$ , is to be used in performing a statistical study. The user is provided by the data base with a set of statistical procedures  $p_1, \dots, p_n$ . The security constraints are that the user is to be restricted to selecting a relevant set of records using attribute  $a_1$  (which he is free to see) and to performing the statistical study. He is not to be allowed to see attributes  $a_2$  through  $a_k$ .

Minsky points out that if the user performs his task by selecting a set of records using  $a_1$  in a program and then feeding the records to the statistical routines, there is no way to enforce the security constraints in common programming languages. This is because once a record is retrieved, there is no way to force the user to restrict himself to feeding the record to one of the procedures; he may instead print out the entire record. As a result, it is argued that control of the internal actions of a user program with respect to output from the data base must be provided.

Minsky suggests that “strongly typed” languages could be used for this purpose. In such a language, every variable has a type which defines its set of legal values and operations [8, 9]. Transfers of information between two variables of the same type are not affected, but transfer of information between two variables of different types requires explicit conversion. The compiler must be able to determine the type (and, thus, the legal set of expressions and assignments) of a variable at compile time. In this particular example, the feature would be used to define the types of the variables containing outputs from the DBMS in a manner that would allow them to be used as inputs to the statistical procedures, but would not allow them to be viewed by the user (i.e., they cannot be stored in other variables or printed out). This allows the compile-time enforcement of the necessary security constraints—provided, of course, that users are restricted to using only compilers with these features and that the output of the compilers can be protected from modification. It is argued that such features are a requirement for languages which interact with data bases, because in some cases this is the only way in which the necessary security can be maintained, and in others it is the only way security can be maintained with reasonable efficiency.

Interpreting this example in light of our model of data base transformations, we find that the above interpretation assumes that the only place where the user interface may be defined is at (a) in Fig. 2. This means that, while the DBMS is responsible for producing

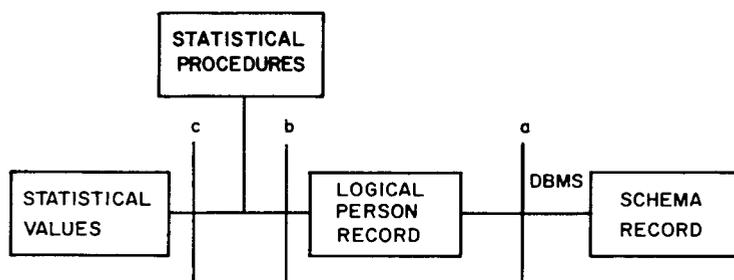


Fig. 2 — Effects of defining user interface at various points in the data transformation process

the logical record, the user is responsible for the subsequent transformation from logical record to statistical values. Even though some of the programming support is provided to him, making the user responsible for this transformation requires giving him access to information which is inconsistent with the security requirements—namely, data about individual persons.

Naturally, given this interpretation, there is a requirement to control internal actions of the user's program and, thus, for features such as strong typing. However, Fig. 2 also shows that there are at least two other places, (b) and (c), where the user interface can be defined to solve the same problem, but in such a way as to deny the user inappropriate access to the logical records and the transformations necessary to derive statistical values from them. This technique then eliminates the need to control internal operations in the user's program.

At interface (b), the user is given access to special statistical operations (e.g., AVERAGE) which take as inputs the attributes to be averaged and a selection expression on  $a_1$ . The operations act as the composition of a record retrieval and the passage of the record to one of the statistical routines. Thus, while the user sees the definition of the logical record, he is allowed to invoke only statistical operations on those records; thus, he cannot gain access to inappropriate data.

At interface (c), the user is given access to a logical record which is a transformation of the logical person record and contains only the statistical values which would be output from the statistical routines. Thus, the user would see only certain attributes (e.g., Average Age, Average Salary) and an attribute containing the selection expression based on attribute  $a_1$  of the original logical record. Examples of such records are shown in tabular form in Fig. 3.

Selection Expression	Average Age	STD Deviation Age	Sum Salary	...
$a_1$ GRT 10	41		138751	
5 LEQ $a_1$ LEQ 50	37		248988	
⋮				

Fig. 3 — Examples of statistical records in the user subschema

The motivation for using this approach is that the user is authorized to see only statistical values and in order to ensure enforcement of this constraint, the DBMS must retain responsibility for materializing the values. (Also, statistical routines must be written in such a way that they do not give an output for very small sample sizes; otherwise they would be susceptible to attack using techniques such as those described in Ref. 10. This is true no matter where the interface is defined and will not be discussed further.) The run-time burden is the same in any case, since the same statistical procedures must be invoked at either of the interfaces. If the interface is at (a) in Fig. 2, it is virtually impossible to perform all the compile-time and run-time checking necessary to use existing programming languages. On the other hand, if the interface is defined at (c), there are no compile-time checks necessary. The DBMS at run time would ensure that only legal operations and objects were used in user requests, and these run-time checks are straightforward. This technique, however, does require extensions to the facilities described in the CODASYL specifications. These will be discussed later.

In general, the problem these solutions address is that the transformation from schema data to the user's conceptual requirements often must be done by means of several smaller transformations; and there must be some way of ensuring that the output of one transformation can only become the input to another, if the user is restricted to the output of the entire process only. This problem is illustrated in Fig. 4. The use of compile-time checks to

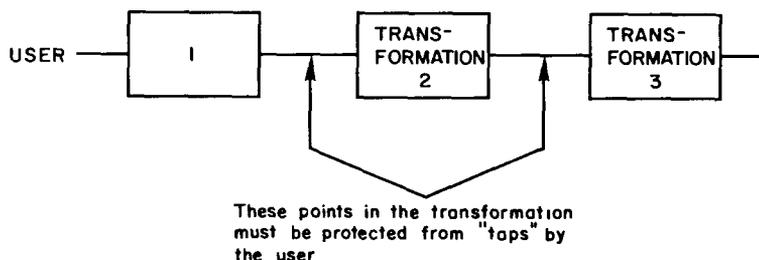


Fig. 4 — Overall transformation from schema to user

ensure the integrity of the interfaces is one solution; we believe that a better one at present is to require that the DBMS perform as much of the transformation as possible prior to presenting data to the user. While the DBMS itself may be required to perform the overall transformation in several steps, it is better able to protect and conceal the existence of the interfaces between the transformations, thus denying the user access to them. This solution also allows the user to employ existing programming languages in accessing the data base; this is an economy which must be balanced against the cost of the run-time checking involved if a realistic evaluation of competitive data security techniques is to be made.

## CODASYL DATA SECURITY FEATURES

Within the three languages specified in the CODASYL specifications, there are essentially four facilities which are important in the provision of data security:

1. Definition of password protection for data items, records, sets, areas, and schemas and subschemas themselves, using the PRIVACY clauses.
2. Invoking of data administrator-supplied procedures for more complex security checking, using the PRIVACY and ON clauses.
3. Concealment of data in the schema from the user by not including that data in the user's subschema, using the schema/subschema architecture.
4. Definition of derived data in the schema which can be an arbitrarily complex transformation of other data, provided by the SOURCE and RESULT clauses. The data may either be materialized at run time or stored redundantly in the data base.

Although these facilities work together in providing data security, the ability to apply the design principles we have been discussing depends mainly on the ability to invoke the necessary transformations below the subschema level in the architecture so as to materialize the data in accordance with the user's requirements. This in turn depends on the ability to conceal these transformations by means of the subschema, to define schema/subschema transformations, and to define derived data.

## USE OF DERIVED DATA

The ability to define derived data (using the SOURCE and RESULT clauses) and the ability to control the implementation of that data (using the ACTUAL and VIRTUAL specifications) allows the data administrator, in effect, to define some types of schema/subschema transformations in the schema. This allows the data administrator to handle those cases where the user may not be allowed access to the data as defined in the schema but may be allowed access to the output of some transformation of that data. This is accomplished by forcing the user to access only the output of the procedures which implement the transformation. The statistical routines discussed in a previous example are examples of such procedures. Situations in which a user should be restricted to only certain statistics of data, rather than the data itself, are fairly common. Ordinarily, in an environment where security is important, the routines that produce the statistics from the raw data would be either written by trusted personnel or carefully screened by such personnel to ensure their conformity to their specifications and security constraints. The problem is to force the user to use these procedures. This may be done in the CODASYL specifications as follows.

Suppose that we wish to restrict the user's access to information about employees to the mean and standard deviation of their salaries by department. The data base could then be defined as shown in Fig. 5.

The user's subschema definition would contain only the DEPARTMENT record. Here, the MEAN-SALARY and STD-DEV-SALARY data items would be declared using the RESULT clause, which indicates that the value of the data item is that produced by the computation of a named procedure (in this case, one of the trusted procedures described above, which accesses the EMPLOYEE records and computes the value). The data administrator can at the same time decide whether to maintain redundant data (by declaring the items ACTUAL and speeding retrieval) or to materialize the data only as required (by declaring them VIRTUAL and saving storage space). In either case, consistency is maintained by the DBMS, since it knows the relationship of the data items from the declaration. Even though this is a rather simple example, the fact that a procedure may be invoked means that arbitrarily complex transformations on data may be performed before the data are delivered to the user. This example also illustrates the importance of anticipating security requirements and incorporating them into the logical data base design. (This, of course, is important regardless of the security architecture used.)

The SOURCE clause is very similar to the RESULT clause and specifies that the value of a particular data item is to be maintained the same as the value of some other data item. Like a RESULT item, a SOURCE item may be declared either ACTUAL or VIRTUAL. The SOURCE item is useful in constructing what are in effect virtual records whose data

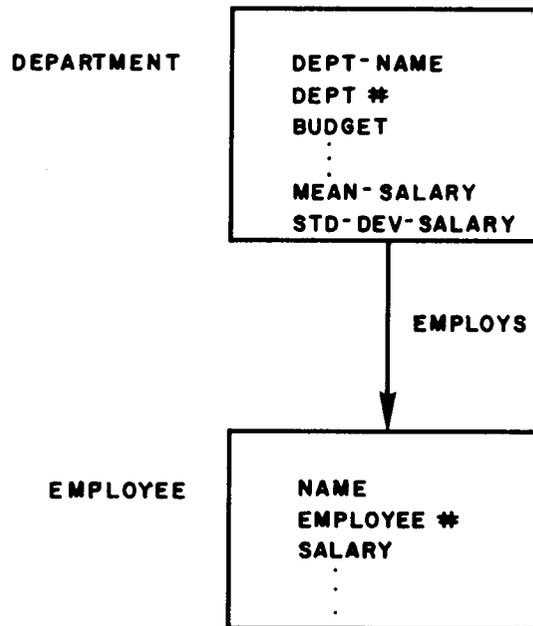


Fig. 5 — DEPARTMENT and EMPLOYEE records in a schema. Only the DEPARTMENT record is included in the subschema of a user who should not have access to individual employee salary information.

items are taken from other schema records, parts of which are prohibited to certain users. These users can be given free access to the virtual records, which effectively denies them access to those prohibited parts of the schema records, as well as knowledge of the existence of the schema records. Facilities are available to maintain consistency if updating of the virtual records is permitted.

#### LIMITATIONS ON THE CODASYL SPECIFICATIONS REQUIRING EXTENSIONS

While the facilities described in the CODASYL specifications certainly provide the possibility for enhanced data security through careful data base design, currently there are limitations in the specifications which can make certain types of control awkward to apply. While the CODASYL architecture allows the data administration to define views of the data base involving arbitrarily complex transformations from the schema objects and operations, in many instances this cannot be done using the subschema mechanism. Instead, an application program must be written to materialize the view. To the extent that these programs can be written in a secure way and the programs and their interfaces to the DBMS adequately protected, this is a satisfactory approach. However, the ability to do this with commonly available programming languages is often restricted. For example, Conway [11] has described a number of techniques that may be used with Fortran, Cobol, and PL/1 that, in effect, allow the user to escape from the language syntax restrictions and execute in an uncontrolled manner. The measures which must be taken to have compilers of these languages perform checking in a manner that guarantees reasonable security may be very

costly or involve unacceptable or, at least, undesirable restrictions on the facilities of the language which can be used. It is our belief that, in such cases, it may be less expensive and more secure to perform sufficient transformation below the subschema level, coupled with such run-time checks as are necessary, to ensure that the user is presented with a subschema which he can use safely even via assembly language. While the limitations which we will discuss are contained in the CODASYL specifications, they are not inherent in the general CODASYL approach and, thus, may be corrected by extensions to the specifications. These limitations are briefly described below.

1. Only the simplest transformations are allowed between the schema and subschema, and even those allowed in the schema (via the SOURCE and RESULT clauses) do not allow, for example, other than data items to be derived in a straightforward way. This has two effects.

a. Even simple transformations, such as those in the examples, must be declared in the schema. This means that the schema becomes cluttered with application-specific data declarations and that the schema becomes sensitive to changes in user requirements, since the schema would have to be modified in such cases. This is particularly important in the case of transformations for security purposes, because these requirements may often change more frequently than user data requirements. While a schema change need not necessarily trigger data base reorganization or impact other subschemas, it is not something that is desirable on a frequent basis.

b. The data administrator is limited in defining an abstract view of the data base tailored to user requirements via the subschema. Thus, the user may be forced in the subschema to see records and data items, rather than application objects. This means that the user must supply programming to effect this transformation with the resulting possibility that he may require access to more data than he really needs.

2. Operations specific to the subschema cannot be defined. This is a serious deficiency in a situation where one would like to define abstractions, as described above, and tailor operations to work on them. However, this is often useful even when ordinary records are involved. If, for example, a user could be presented with a subschema in which the STORE operation was not defined, attempts by that user to use the STORE operation could be more easily checked.

3. Dependencies in the CODASYL specifications (specifically, between the schema declarations and the subschema operations, as in Ref. 12) mean that, in certain cases, information cannot be concealed from the user, since he needs to know it in order to write appropriate operations. For example, if a record is a member of only one set, unless it is declared as an entry point in the data structure, the set must be known to the user in order that he may use the name in the operations (even though the set name required to access the record is implicitly known from the declaration).

## EXTENSIONS TO THE CODASYL ARCHITECTURE

The extensions to the CODASYL architecture to remove such limitations fall into the following general categories.

1. The ability for the data administrator, by declaration, to control more precisely the use of various operators defined for the subschema. This involves two things:

a. Added syntax to allow more precise declaration of security constraints for a subschema in terms of legal operations on data defined in the subschema.

b. The ability to control the responsibility for checking specified conditions. Thus, the data administrator should be able to inform the system that the output from a particular compiler is to be considered trustworthy with certain types of constraints, that the DBMS is to do the checking of the output, or that he will supply a procedure to perform the checking. This involves providing a mechanism for identifying the output of each language translator and protecting it from unauthorized modification.

2. The ability to define more complex schema/subschema transformations, so that data objects (including not only derived data items, records, etc., but also more abstract objects such as reports, warehouses, etc.) tailored to the user's application may be defined, and so that special operations appropriate to these data objects may be defined.

For example, a user concerned with shipping and receiving might be given access to a data object WAREHOUSE and such operations as STORE PART  $x$  IN WAREHOUSE  $y$ , rather than to the more data-base-oriented objects such as records and data items which might be used to define the data at the schema level. Such a subschema definition would not only make run-time security checking easier, it would also help to enforce desirable access patterns for concurrency control and would make the user independent of the data structures used at the schema level to support the warehouse data.

## IMPLEMENTATION

The present subschema languages are essentially extensions to existing host languages. In order to provide for the more flexible types of subschema descriptions which we are proposing, we must have essentially an extensible language facility available. However, the compiler modifications needed are not extensive. While a preprocessor approach or the use of any extensible language facilities available in the compiler might be considered, they assume that the compiler is essentially "trustworthy" and that the compiled code can be adequately protected from modification. If the language is not trustworthy, as for example Fortran, more of the responsibility must be borne by the DBMS itself. In this case, all that would be required of the compiler or preprocessor is that calls be generated to an appropriate entry point in a system-provided transformation particular to the subschema for the new operations. Thus, a MEAN operator in a program would either be recognized as legal, and a call to the entry point for MEAN in the system transformation generated, or the operator would simply be recognized as nonlanguage syntax and a call generated to a single entry point for the transformation with the operator as one of the parameters. The system-provided transformation would then be responsible for ensuring that the particular user involved entered the system transformation only at one of the proper entry points defined for him and, at such an entry point, that the parameters passed in the call (either operator or operands) were legal for that user's subschema. In this case, all the compiler would have to know is how to generate the call to the system entry point. The extent to

which the compiler is expected to detect syntax errors depends on how extensively the compiler is modified to do so. However, the real responsibility for security checking resides in the system transformation, not in the compiler. Even if the user knew how to access the system entry point (e.g., in assembly language), sufficient checking would be performed by the transformation to ensure that no illegal actions could be taken by that user. The user is free to use whatever language he wishes with respect to such an interface. Naturally, to the extent that the data administrator wishes to delegate security responsibility to a compiler, the user would have to be restricted to that compiler, and adequate measures would have to be taken to protect the compiled code and to identify modules compiled by the compiler when they call the transformation.

## CONCLUSION

It is our belief that only when security checking is considered to the exclusion of all other considerations is the necessity obvious for compile-time checking and languages with features such as strong typing. While such facilities are clearly useful, the existence of associated security costs (the cost to protect the compiled code, to ensure that users only use specified compilers, and to check the source of any code executed) and other nonsecurity costs (the requirement to convert existing applications and packages to new languages, and to retrain programming personnel) makes the evaluation of the tradeoffs fairly complicated. Further, present security requirements, we feel, will not wait until such languages are in common use. As a result, we feel that design techniques such as those suggested here, which simultaneously provide enhanced security, user convenience, and data base system control, need to be employed to the greatest extent possible.

Work on extended schema/subschema transformations is currently under way by the CODASYL Data Description Language Committee (DDL) Subschema Task Group (SSTG). The DDL itself has created a subgroup to investigate additional data security facilities. This work, in addition to current research efforts under way at several universities and corporate research centers connected with increasing security facilities, provides hope that additional capabilities for controlling data security will be placed in the hands of data administrators.

## ACKNOWLEDGMENTS

The authors gratefully acknowledge the help of John Shore, Naval Research Laboratory; John Berg, National Bureau of Standards; and Tax Metaxides, Bell Laboratories; during the preparation of this report. The authors also acknowledge the contributions to the ideas in this report from various internal documents from the CODASYL Data Description Language Committee's Working Group on Environment, and the ANSI/X3/SPARC Data Base Systems Study Group.

REFERENCES

1. *CODASYL Data Base Task Group, April 1971 Report*, Association for Computing Machinery, New York.
2. *CODASYL Data Description Language Journal of Development, June 1973*, NBS Handbook 113, U. S. Department of Commerce, National Bureau of Standards, Washington, D.C. (SD Catalog No. C13.6/2:113).
3. R. F. Boyce, and D. D. Chamberlin, "Using a Structured English Query Language as a Data Definition Facility," *IBM Research Report RJ 1318*, IBM Research Laboratory, San Jose, Calif., Dec. 10, 1973.
4. I. Palmer, "Levels of Database Description," *Information Processing 74* (Proceedings IFIP Congress 1974), North-Holland Publishing Company, Amsterdam, 1974.
5. C. J. Date, and P. Hopewell, "File Definition and Logical Data Independence," in *Proceedings of the 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control*, Association for Computing Machinery, New York, Nov. 1971.
6. N. Minsky, "On Interaction with Data Bases," in *Proceedings of the 1974 ACM-SIGMOD Workshop on Data Description, Access and Control*, Association for Computing Machinery, New York, May 1974.
7. R. I. Baum, and D. K. Hsiao, "A Data Secure Computer Architecture (Part I)," Report OSU-CISRC-TR-73-10, Computer and Information Science Research Center, The Ohio State University, Columbus, Ohio, July 1974.
8. D. Parnas, "Use of the Concept of Type Classes to Simplify CS-4," private communication, Darmstadt, Technische Hochschule, Darmstadt, W. Germany, 1975.
9. B. Liskov, and S. Zilles, "Programming with Abstract Data Types," *ACM SIGPLAN Notices* 9, No. 4 (Proceedings of an ACM SIGPLAN Symposium on Very High Level Languages), Apr. 1974.
10. L. J. Hoffman, and W. F. Miller, "Getting a Personal Dossier from a Statistical Data Bank," *Datamation* 16, No. 5, 74-75 (May 1970).
11. R. W. Conway, W. L. Maxwell, and H. L. Morgan, "On the Implementation of Security Measures in Information Systems," *Communications of the ACM*, 15, No. 4, 211-220 (Apr. 1972).
12. R. W. Engles, "An Analysis of the April 1971 Data Base Task Group Report," *Proceedings of the 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control*, Association for Computing Machinery, New York, Nov. 1971.