FROM INFORMATION REQUIREMENTS TO DBTG-DATA STRUCTURES*

by

J.A. Bubenko jr,** S. Berild,

E. Lindencrona-Ohlin, S. Nachmens

Research group CADIS,
Department of Information Processing and Computer Science,
Royal Institute of Technology/University of Stockholm, Sweden.

ABSTRACT

The problem of determining, analysis and description of a particular application's information structure (and relations) and the process of mapping the information structure to a "good" data structure (in this case a DBTG-type structure) is considered. The applicability of a top-down oriented design procedure to a relatively large practical data base design case is demonstrated. A conceptual framework and a notation to be used for determining and definition of information requirements, information structure and information relations is suggested. A systematic and partly formalized approach to map an information structure to a DBTG-type data structure is discussed. The problem of analysis and evaluation of alternative DBTG-type structures is also considered.

1 INTRODUCTION

Researchers in the field of data bases do (see f ex (4), (5), (6)) on the whole agree that three design and description levels can be recognized, namely

(1) the information structure (or "infological") level

(2) the data structure ("datalogical") level and

(3) the storage structure level.

The information structure level corresponds to the end-user level where information is referred to in problem-oriented and implementation independent terms. The data structure level corresponds roughly to a level where one has decided upon the data representation of information (for instance, when using a Codasyl DBTG-type data base management system (1), the definition of record-types, set-types and access principles). The storage structure level is concerned with how records, sets etc are implemented and how these entities are allocated to physical devices.

These levels imply some important design problems and decisions:

- how to map "reality" into an information structure
- how to map an information structure into a data structure
- how to map a data structure into a storage structure.

On each level, an almost infinite number of alternative mappings exist, i.e. reality (the actual part of it) can be represented by many alternative information structures, an information structure by many alternative data structures, etc. Additional problems are introduced by the fact that alternatives are difficult to evaluate. They possess many, often not measurable, properties which cannot be weighted into one objective function. Also the absence of a generally accepted notation and conceptual framework is noticeable. This is especially the case concerning the information structure level. It has the effect that in most practical situations "reality" is mapped directly to a data structure and that data structure alternatives are seldom examined and evaluated.

Schema design can be seen as a problem solving process which starts with analysis of the user's needs and definition of the information structure and which leads to a feasible and "good" data structure. In order to constitute a useful basis for the subsequent part of the schema design process, the information structure must be documented in a precise and formal way and include information such as requirements concerning input/output (including information relations), response times, security etc. The result of the schema design is

- a data structure (schema) described in Schema Data Description Language (Schema DDL)
- estimates of quantifiable properties of the data base, such as storage requirements, response times, times for creation and updating, etc
- estimates of unquantifiable properties of the data base, such as flexibility, simplicity of implementation, security etc.

This paper

- demonstrates the applicability of a top-down oriented design procedure to a relatively large practical data base design case
- suggests a conceptual framework and a notation to be used to determine and define information requirements, information structure and information

* This research was supported by the Swedish Board for Technical Development (STU)

** Address in 1976: IBM Research, Yorktown Heights, N.Y.

relations, and demonstrates its applicability and utility

- discusses procedures for analysis and evaluation of alternative DBTG-type structures.

The theoretical work (see also [2]) was carried out in close association with work on a practical data base schema design project. The objective of the project was to study the feasibility and consequences of applying a DBTG-type DBMS [3] to construct a data base for a Swedish organisation's financial follow-up, budgeting and forecasting information system. The size of data base is approximately 250 Mbytes and the transaction rate is about $2.5.10^6$ per year.

2 THE SCHEMA DESIGN PROCESS - AN OUTLINE

The design process outlined here evolved from the work on the above practical case. The general validity and applicability of the results presented must be evaluated considering the special characteristics of this particular case. These are:

- The input boundary, in terms of existing transaction types, is given.
- There are no strong requirements concerning the functional availability of the DB-system or concerning data privacy and security.
- Retrieval of information from the DB will be made by a predefined set of queries, which do not contain conditional or relational expressions of any noticeable complexity.

The schema design process (figure 1) contains three main tasks:

(1) Determination of information structure and relations.

This task includes

- collection of quantitative information about the reality,
- mapping of the actual part of the reality into an information structure,
- determination of information relations,
- determination of required and desirable properties of the data base.

Close collaboration between the user and the data base designer is essential here. Only the user can state the information requirements and only the designer knows which information (about requirements) is essential for the subsequent stages in the schema design process.

(2) Design of a schema alternative.

This task constitutes the mapping of the information structure into a DBTG-data structure. Requirements, such as ability to respond to certain types of queries within a predescribed response time have to be considered. At the same time one has to try to arrive at as good values as possible for certain properties that are more or less easily measurable, such as need of secondary storage space, ease of implementation and maintenance, flexibility etc.

(3) Analysis and evaluation of the schema alternative.

Relevant data base properties are derived and evaluated against specified properties, according to phase (1). Some quantifiable properties can be analytically derived while other properties can only be estimated.

The above three tasks require information and produce information. Systematic collection and documentation of information constitutes therefore important and necessary subtasks of the design process. In the actual case described, more than 50% of the time was spent in phase 1. Construction of a data base schema is a multi-goal task, which makes it less meaningful to discuss "optimal" data structures. What can be done, is to, in a systematical way, arrive at different schema alternatives, which then have to be partly formally evaluated. This, in turn, makes it necessary that the tasks (2) and (3) are performed in an iterative way with the objective to arrive at a "balanced" schema alternative, where stated requirements are satisfied and where desirable properties are considered.

The similarity of this process with the "Sketch of basic theory of systems analysis" by Langefors (8), should be observed.

The amount of information about the design object that the designer has to handle increases rapidly with the complexity of the data base. This raises the need for a precise and compact notation for describing information needs, relations and properties of the data base. The concepts and notation used is discussed in the next section.

3 INFORMATION STRUCTURE AND RELATIONS

The transactions and queries described verbally or by old computer-produced reports, had to be translated into a formal notation. The "translation" started with examination of:

- About what (objects) entities should the data-base contain information?
- What "kind of information" should be kept about these (objects) entities?
- How are these information elements functionally related?

3.1 Different kinds of objects

An object is considered an abstract or concrete entity, about which information is recorded or derived. An object class is then a set of objects who in some sense are considered sufficiently similar from the application problem point of view.

For unique identification of objects, objects from a set of base object classes were used. A base object class X consists of a finite set of identifiers $\{x_1, x_2, \ldots, x_n\}$ which uniquely identify an administrative (abstract or physical) concept within the organization (cf an attribute in a "candidate key" (9) or "name set" in (4)).

As it turned out, in this application, the base object classes corresponded to the main accounts in the accounting system. For instance departments

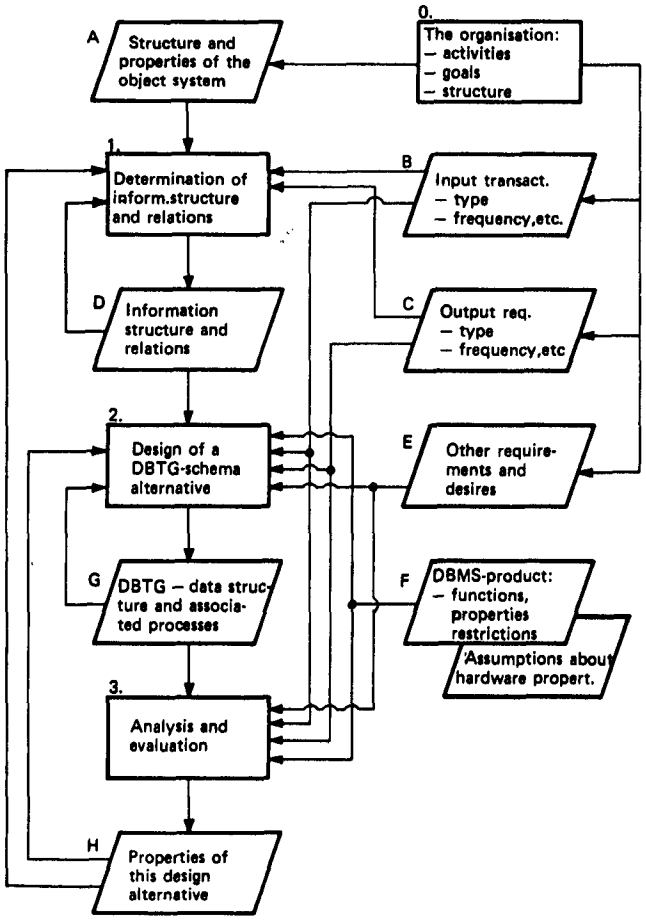


Figure 1: Crude outline of the schema design process

within the organization were identified by department numbers, which then were used for identification of accounts. We regarded the set of department numbers as a base object class.

Eight base object classes (table 1) were recognized. For each of them their cardinality was estimated.

Base object class	Abbreviation	Cardinality
Department number	DEP	35
Division "	DIV	1500
Program "	PRG	75
Project "	PRO	500
Sub-project "	SUB	1000
Activity "	ACT	1500
Technique "	TEC	500
Resource "	RES	100

Table 1

3.2 Information object classes

An information object class is a set of objects sufficiently similar. Information about elements of an information object class may be recorded or derived. An information object class is associated with a set of base object classes and tuples of these uniquely identify an information object. Notation used:

(A) an information object class associated with base object class A. Elements of this class are denoted $(a_i) \in (A)$.

(A,B) elements of this class are denoted $(a_i , b_j) \in (A,B)$.

The relevant information object classes were determined by examination of each specified transaction and query type. For instance, the queries

(1) What amount of money has been spent by a certain department last month?

(2) What amount of money is budgeted for a specific department on a specific program for the next year?

lead to the information object classes (DEP) resp (DEP,PRG). Elements of these classes are denoted (dep) and (dep,prg) respectively.

By examination of each transaction and query type 24 information object classes were identified and their cardinality determined. Some examples are shown in table 2

3.3 Relations between information object classes

Let $\overline{B}$ = (DEP,DIV,PRG,PRO,SUB,ACT,TEC,RES) be the set of defined base object classes. For a particular application problem, "meaningful" subsets B_i, of this $\overline{B}$ can be defined (see table 2).

The set of meaningful subsets is a subset of the power set of $\overline{B}$ and it defines the set of information object classes relevant to the problem.

An information object class B_i is a constituent of an information object class B_j iff $B_j \subset B_i$. The notation used for this property is B_i (<) B_j.

The pragmatic interpretation of constituence should be that information associated with an object class B_i is in some object-system sense "finer" (further partitioned) than information associated with an object class B_j , B_i (<) B_j . For instance information about the money spent by a department-division (DEP,DIV) is regarded as a refinement of information about the money spent by a department (DEP).

Constituent relations hold the transitive property, i.e.

B_i (<) $B_j \wedge B_k$ (<) $B_i \Rightarrow B_k$ (<) B_i .

B_i is an immediate constituent to B_j , B_i (<<) B_j , iff B_i (<) B_j and there does not exist another B_k such that B_k (<) B_j and B_i (<) B_k .

In table 2, for instance

B_3 (<) B_1 , B_3 (<<) B_2 , B_2 (<<) B_1 etc.

The set of constituents of an information object class B_i will be denoted by $<(B_i)$ and the set of its immediate constituents by $<<(B_i)$. Obviously $<<(B_i) \subseteq <(B_i) \subset \overline{B}$. An information object class B_i is called initial if $<(B_i) = \emptyset$.

Inf.obj. class	Base object classes	Card.
B_1	DEP	35
B_2	DEP,DIV	1.500
B_3	DEP,DIV,PRG	11.700
B_4	DEP,DIV,PRG,PRO	12.550
B_5	DEP,DIV,PRG,PRO,SUB	16.900
B_6	DEP,DIV,PRG,PRO,SUB,ACT	14.200
B_7	DEP,DIV,PRG,PRO,SUB,ACT,TEC	73.700
B_8	DEP,DIV,PRG,PRO,SUB,ACT,TEC,RES	377.000
B_9	PRG	75
B_{10}	PRG,PRO	2.015
⋮	⋮	⋮

Table 2

3.4 Information "kinds" and elements

Having defined the information object concept we need a notation to define what "kind of information" about elements in information object classes we want to record or to derive. This can be done on several levels of precision. For instance, on a crude level we may say that we are interested in "budget information". On a finer level "budget information" may be refined to "amount of $ budgeted" and on a still finer level this "kind of information" could be refined to "amount of $ budgeted for the first month of this year".

No general rules can, however, be given how to partition and how to refine information of an organization.

In the actual case, decisions concerning the "kinds of information" to be contained in the data base were relatively simple because

- there were few "crude" kinds of information
- the partitioning of the "crude" kinds of information was uniform for different information object classes
- with few exceptions, all kinds of information (at all levels of refinement) were associated with all information object classes.

There were, in all, essentially four crude kinds of information:

- B (Budget information)
- P (Forecast information)
- U (Follow-up-information)
- R (Economic frame information).

These four crude kinds of information could all be further partioned by Q (quantity) and by S (sum or amount), which was denoted, for instance

B.Q , B.S , P.Q , etc.

A further refinement of the information kinds concerned time, for instance, a budget was partitioned in quarters of a year. The following examples illustrate the notation used to refer to information elements and sets of such elements

B.Q.1(dep,prg)	budgeted quantity first quarter for a particular department on a particular program
B.Q.1(DEP,PRG)	the set of information elements {B.Q.1(dep,prg)} for all (dep,prg) $\in$ (DEP,PRG)
U.S.3(dep,div,prg)	the amount of $ spent in month 3 for a particular (dep,div,prg)-tuple.

3.5 Information relations

In a data base, or in a large set of information elements, there normally exists a certain redundancy. This means that the value of an information element can be derived (given the derivation rule) from a set of other information elements because a functional relation holds between these elements. If the value of an information element cannot be derived from other elements then this element is said to be initial. Functional relations of this kind can be object-local or object-global. An object-local relation defines the rule for derivation of the value of an information element from information elements associated with the same object. An object-global relation may hold between an information element and a set of information elements associated with another information object class.

As an example, consider two information object classes (DEP,DIV) and (DEP,DIV,PRG). An object local relation is then for example:

$$\text{U.S.Y. (dep,div)} = \sum_{i=1}^{12} \text{U.S.i (dep,div)}$$

U.S.Y is the notation for "amount of money spent this year" and U.S.i is the amount of money spent month i.

It may also be true that U.S.Y (dep,div) can be derived from information elements associated with the class (DEP,DIV,PRG), for instance

$$\text{U.S.Y (dep,div)} = \sum_{\text{prg/dep,div}} \text{U.S.Y (dep,div,prg)}$$

which means that the amount of $ spent this year by a (dep,div) can be deduced by summing the amount of $ spent this year by all prg associated with a particular (dep,div)-tuple.

Alternatively

$$\text{U.S.Y (dep,div)} = \sum_{\text{tec/dep,div}} \text{U.S.Y (dep,div,tec)}$$

In this case U.S.Y (dep,div) is a function of a set of information elements U.S.Y (dep,div,prg) and/or a function of a set of information elements U.S.Y (dep,div,tec) .

Object-local resp object-global information relationships can be graphically illustrated as in figure 2.

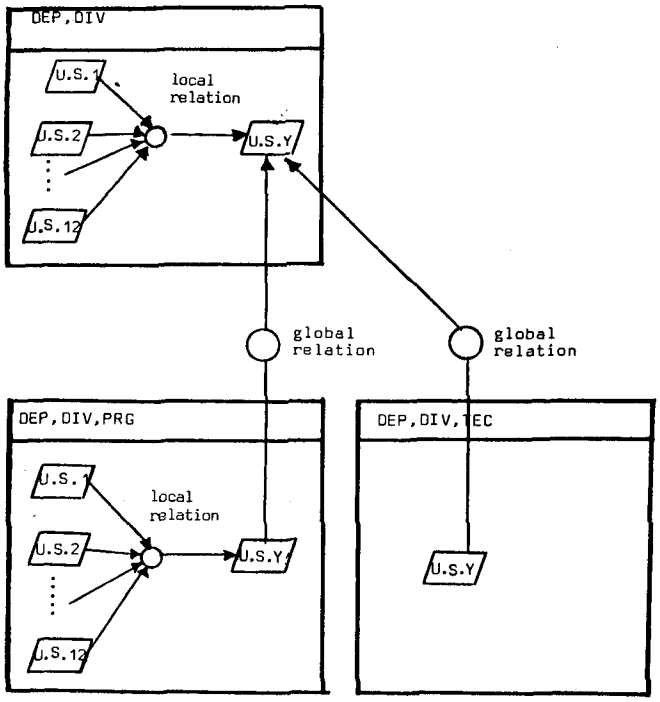


Figure 2: Illustration of object-local, object-global and alternative functional relations between information elements.

In this example it is shown that two (or more) alternative sets of information elements could be used to derive a particular information element.

The notation used for functional precedence relations is:

$I_b (B_i) < I_a (B_j)$, meaning that

$I_a (b_j)$, $\forall b_j \in B_j$ is derivable from a set of information elements

$\{I_b (b_i : b_i \in B_{i/j} \subset B_i , B_i (<) B_j\}$

Concerning functional precedence relations we also observe that

- the transitivity property holds
- from $B_i (<) B_j$ it does not (in general) follow that there exist information kinds I_x such that $I_x (B_i) < I_a (B_j)$ for an I_a associated with B_j .

3.6 The correspondence aspect

For information relations which follow constiuent relations, the correspondence aspect is "one-to-many" (1:M). This means that an element $b_j \in B_j$ is normally associated with one or more elements $b_i \in B_i$, if $B_i (<) B_j$, and that one element $b_i \in B_i$ is associated with one and only one element $b_j \in B_j$. It was observed that the knowledge of some statistics about the number of elements b_i associated with an arbitrary b_j was of importance when designing and analyzing schema alternatives. In this feasibility study, as only approximate calculations were made, the correspondence average seemed to be sufficient. The correspondence average between classes B_i and B_j , $B_i (<<) B_j$ is denoted $\bar{c}_{ij}$ and calculated simply as

$$\bar{c}_{ij} = CRD (B_i) / CRD (B_j)$$

where $CRD (B_i)$ denotes the number of elements in class B_i .

3.7 Description of transactions and queries

The notation used when analyzing and documenting the required transaction and query types were

T (<inf.kind spec.> (<object spec.>)) and

Q (<inf.kind spec.>(<object spec.>) .

The same notation for on-line and "batch" transactions and queries was used. For each transaction kind it was also noted whether the operation concerned updating, insertion or deletion. The <inf.kind.spec.> described the kind of information required and <object spec.> "about which objects".

The queries concerned a single object or a hierarchical set of objects from specified object classes.

Examples:

Q_1 (B.S.1 (DEP = dep, DIV = div)) =

the amount of $ budgeted the first month this year for a specific division under a specific department.

Q_2 (B.S.1 (DEP = dep, DIV = ALL)) =

the amount of $ budgeted the first month this year for all the divisions of one specific department, (sorted by divisions).

In this case, the order of specifying base object classes was relevant because it determined the sorting order.

Q_3 (B.S.1 (DEP = ALL,DIV = ALL)) =

the amount of $ budgeted the first month this year for all departments and all divisions within the departments, sorted on departments, and within each department, on division.

The frequency was estimated for each transaction and query type. For queries, requirements concerning the actuality of the answers/reports were defined as well as requirements concerning response-time. Approximately 30 different transaction types and 80 different query types were specified.

3.8 Information structuring: a summary

The result of task (1) of the schema design process is

a) a description of transactions in terms of
   - their content - information objects and
   - frequency
   - the sort order
   - the type of processing: update, insert or delete.

b) a description of queries in terms of

- their content - information objects and kinds of information
- the sort order
- frequency
- response time requirements
- actuality requirements, i.e. maximum time between input of a transaction and the time when the affected information in the data base is updated.

c) a description of the information structure in terms of

- base objects and base object classes, their cardinality
- information objects and information object classes, cardinality
- correspondence between relevant information objects
- kinds of information
- information elements
- functional precedence relations between information elements.

3.9 User requirements and desires

The schema design process was guided by a set of

- functional requirements
- performance requirements
- statements concerning "desirable properties" of the design object.

Functional requirements concerned ability to accept specified transactions, ability generate replies to specified queries or to requests for reports and ability to specify procedures for retrieval of information about any combination of base object classes (i.e. new information object classes). Performance requirements concerned average and 95-percentile response times for replies to certain specified query types, generation times for reports, updating times and data base creation time. Statements about desirable properties were made concerning

- secondary storage space
- sensitivity to variations in relative frequencies of the specified queries and transactions
- security
- functional availability
- flexibility and incrementality.

These latter properties, as well as the quantifyable performance properties, are further discussed (with reference to the case study) in section 4.6.

4 DESIGN OF SCHEMA ALTERNATIVES

4.1 The approach

Two extreme schema alternatives can be recognized

(1) data are stored on the initial (transaction) level only - responses to queries are produced by deriving information from the initial information level

(2) the information in the data base is organized in such a way that information for the reply to every known query is explicitly stored and updated when a transaction, affecting it, arrives.

Alternative 1 minimizes the secondary storage space required but this is done at the expense of extremely long query-response times and long times for generation of reports. Alternative 2, on the other hand, provides fast response times to on-line queries and requires less time for generation of reports. This is, however, achieved at the expense of considerable resource consumption for data base updating, record insertion/deletion and also for storage of the data.

The first alternative was unacceptable considering the response time requirements. For instance, one of the on-line queries would need access to all initial information objects (377000). The second alternative, on the other hand, satisfied response time requirements but did not satisfy other requirements well enough.

A partly formalized procedure for construction of a feasible schema alternative, close to extreme alternative 2, was developed (section 4.2). It was then analyzed with respect to stated requirements and desirable properties. After several iterations (schema modifications) three final alternatives were suggested, each with the requirements met, but to different degrees (sections 4.3-4.5). The user then had to make the final evaluation and choice of alternative.

4.2 Five steps to a response time oriented extreme schema alternative

The goal was to design a schema which is characterized by easily accessible, explicitly stored replies to every known query.

A query (see also 3.7) concerns an information object class B_i and consists of two parts B_i^1 and B_i^2 such that $B_i = B_i^1 \cup B_i^2$ and $B_i^1 \cap B_i^2 = \emptyset$. One of the two, but not both, may be empty. The first part requires unique access to information objects B_i^1. The second part specifies access to information objects $B_i^1 \cup B_i^2$ which are associated with the unique object of class B_i^1. This query-structure property is used in the design steps below.

1 Define all information object classes needed (see section 3).

2 Separate those information object classes (B_i) for which there is a need for access to unique objects i.e. $B_i^2 = \emptyset$. Consolidate the information concerning each (B_i) into a record type identifiable by the base object classes included in (B_i). For these record types direct access is desirable.

3 Search those object classes (B_i^1) whose objects are the unique identification (owner) of a set of objects (constituents, members) of an information object class (B_i). For each of them define a dummy record-type, identifiable by the base object classes included in B_i^1, and indicate it as directly accessible. Consolidate the information concerning the members (B_i) into a record-type.

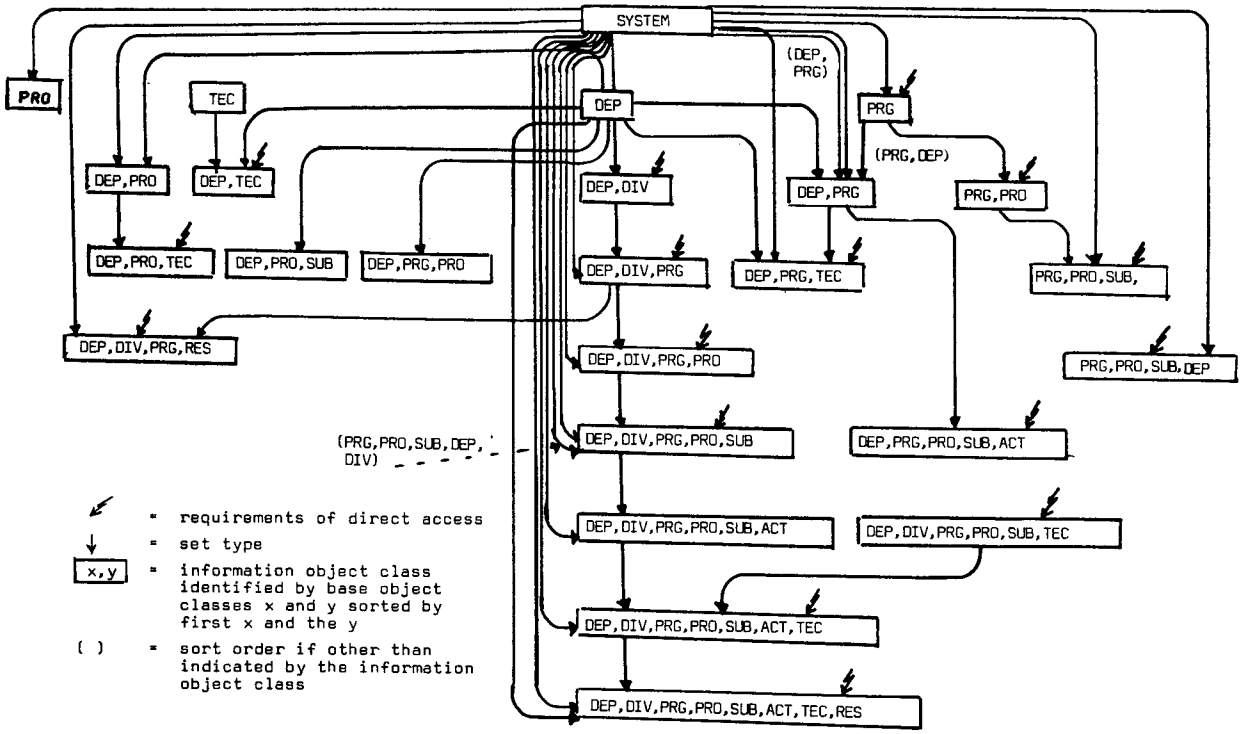


Figure 3: An extreme, response-oriented schema alternative.

4 Establish a set-type from each owner-record-type (B_i^1) found in the previous step to its constituent information object class ((B_i), member) . The members within a set occurrence are sorted according to the conditions (order of B_i^2) given by the query. This means that a record type can be member in more than one set type. It also means that more than one set type can be defined for two record types. Define each member record-type as sequentially accessible.

5 If there are queries, specified as ALL-combinations only i.e. $B_i^1 = \emptyset$, define the corresponding information object classes as record types. Define a set type for each such record type with the universe (SYSTEM) as owner and the actual record type as member. Define each such member record type as sequentially accessible.

Application of these steps resulted in a schema according to figure 3.

4.3 Design of a response time oriented schema alternative

4.3.1 Design principles

The extreme alternative (fig 3) was used as a basis for stepwise refinements to increase storage economy in the first hand. For each refinement the schema was evaluated against stated requirements. Properties and restrictions of the DBMS-product to be used for implementation were also considered.

To evaluate different schema alternatives on the logical level there is a need for various resource measures. The most appropriate would have been to use measures for number of secondary storage accesses, CPU-time, etc. However, these are not possible to calculate at this stage as they depend on several implementation and allocation decisions to be made in a later stage.

Therefore the "measure" dba (= "database access" = = resources consumed for retrieval or storage of a record occurrence) was introduced. The assumption was made that a schema alternative consuming less database accesses per reference-period concerning a stated requirement will, in this respect be more efficient than an alternative using a larger number of dba:s .

An overview of a set of refinement steps which lead to a tree-like schema structure is given below:

1 Reduction of set types

A reduction of the number of set types yields a decrease of insertion/deletion time and the storage space needed for link addresses. If there is a path (a set type) from a unique owner record type to a member record type, where the members are sorted in a given order, and there also exists a path from the same owner record type over two or more set types to the same member record type, where the members are sorted in the same order, then the first set type can be eliminated at the expense of a longer access path.

The effect of eliminating S1 is (see fig. 4)

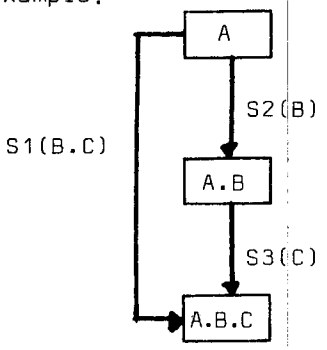
(a) less space for link addresses

(b) fewer dba for insertion and deletion of A,B,C-type records

(c) more dba to retrieve appropriate A,B,C-t records.

The positive effects (a and b) have to b against the negative effect (c) on res

Example:



( ) = sort-order

Figure 4: Reduction of set-types.

2 Reduction of direct access entries

Some queries indicated a need for direct access to records of specific record types. This could be effectuated either through a LOCATION MODE CALC or INDEXED declaration in the record description, or through a SEARCH KEY declaration for the record type if it is declared as a member in a set type. The need for direct access is in the latter case "moved up" one step. Generalization of the use of SEARCH KEY will give a starting point at the highest possible level.

If there is a need for direct access to an A.B.C-record as indicated in fig 5a, this can alternatively be effectuated as shown in fig 5b. (Thick arrows indicate the search paths).

3 Entry through the SYSTEM-record

Introduction of these changes would result in a schema alternative where almost all queries would have to be answered by starting from the SYSTEM (universe) - level and by searching the appropriate records through SEARCH KEY-governed direct access, sort key governed sequential access or a combination of both. As the maximum number of characters allowed in a direct access key in the actual DBM$-product was limited, some direct-access needs could not be met. Therefore it was decided to reconstruct the schema in order to get a consistent way of searching records, namely always from the SYSTEM level. The affected record-types are linked through a new set type as members to an existing record type (owner) which have as many identifiers (base object classes) as possible in common with the member. The owner record type must be, or become, linked, directly or indirectly, to the SYSTEM record (see fig. 6a and 6b).

Two new set-types are added to the schema (S3 and S4). In this way the search can always begin from the SYSTEM record and further through SEARCH KEY down the appropriate paths in the schema to the searched record, (for an A-record over S3, for an A.B.D-record over S3-S1-S4).

4 Eliminate multiple search- or sort-keys

If there are two member record types X and Y, which have the same owner record type but through two different set types and where the base object classes in X is a subset of those in Y

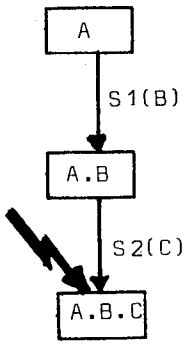


Figure 5a

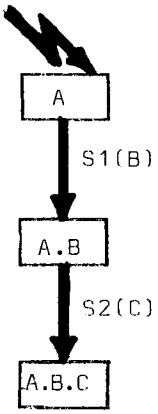


Figure 5b

with the same sort order, then the set type having Y as a member can be replaced by one where X becomes the owner record type. This would reduce the space needed for storage of the identification terms of the Y-record (see step 5), at the cost of extra dba to access Y-records. The set type S3, which has a A.B.C-record type as member, will get its owner record type changed from A to A.B (see fig. 7a and 7b).

5 Reduce set membership

With the refinements according to steps 1-4 above the different record types could be reduced without loss of unique identification of records. Only those identification terms of a member record type not included in the identification of the owner record type must be declared. If a record type is a member in more than one set type then there is a choice between

- using an identification built up by the union of base object classes needed in the different set types
- by duplicating the record-type to get one occurrence for each set type where each occurrence containes only the necessary identification terms.

In our case, the duplication method was carried out in order to reduce the number of identification terms in all record types. However, this is done at the expense of extra dba for update of the "copied" records.

Together with some minor refinements, not mentioned here, these restructuring steps led to the schema alternative shown in figure 8. The figure also shows the correspondence averages between record types and the expected number of occurrences of each record type.

This schema alternative can be characterized as follows

- It is output oriented, i.e. it makes it possible to give relatively fast replies to queries achieved at the cost of considerable resource consumption for updating and data storage.
- It is a pure tree-structure.

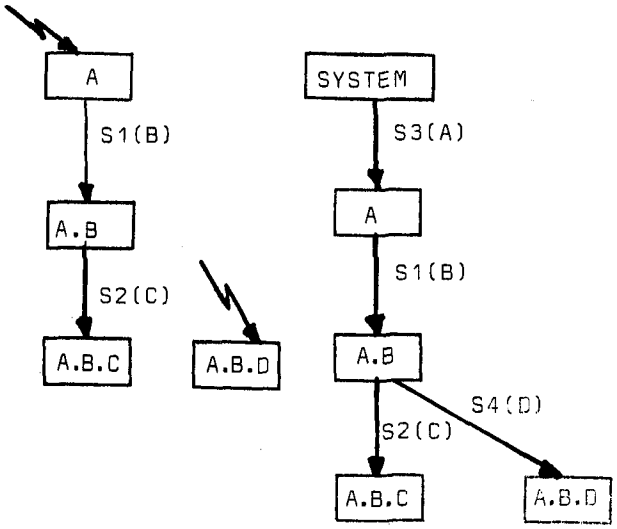


Figure 6a Figure 6b

- Each record type is identified by its set membership and by one base object class only.
- All searches start from the SYSTEM record and follow the branch corresponding to the searched record's base objects (identification terms) in a predefined sort order.
- Replies to all queries are explicitly stored.

4.3.2 Quantitative properties

Four quantitative properties, shortly discussed below, were considered when comparing different schema alternatives.

Search times

Search work can be expressed as the number of data base accesses (dba) needed

1 per query, each time a query of this type is submitted to the data base

2 per query type, per month

3 for all queries per a given time period.

For instance the query Q (DEP=dep, PRJ=ALL, SUB=ALL) used a search path[1] SEK/S1-SOK/S24-SOK/S25 which required (see also figure 8) an average of 176 dba to locate an average of 100 requested record occurrences.

In this way the total number of dba for all known queries/month was estimated to about 800 000.

Update times

Update times were expressed as the number of dba needed to perform a certain update operation. About 95% of the transactions concerned the information object B_8 (see table 2), which required

1) SEK/S_x means: search a unique record occurence in a given set occurrence S_x using SEARCH-KEY.
SOK/S_x means: ordered search for all record occurrences in a set occurrence S_x .

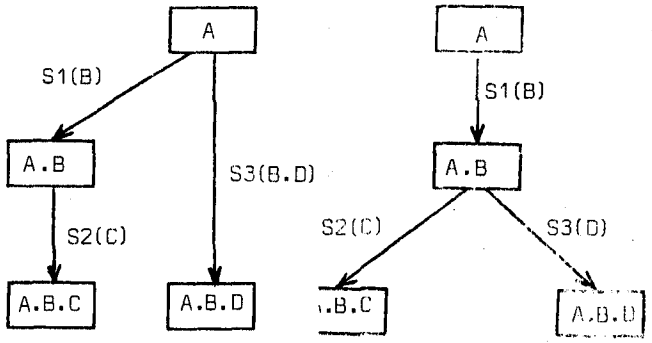


Figure 7a Figure 7b

updating of the involved record occurrences from the lowest level and up to the top level through all existing paths. When estimating the number of dba needed for updating the other transaction types were not considered because of their low frequency.

The estimated transaction rate was about 40.000 per week. As information in one record occurrence, in our case, could be derived from information in record occurrences to which it is an owner, each transaction affected one record occurrence of most record types. Thus, with 29 record types and 40.000 transactions, about 1160000 updates of record occurrences would have to be performed each week. This figure could however be reduced by

- a suitable transaction sort order for each branch in the tree
- transaction level accumulation before database updating
- creation of temporary files, when updating one branch, with accumulations on different levels suitable for updating of other branches.

In this way the number of affected records was estimated to about 380000/week and 1500000/month. To these efforts the work required for various sorting and accumulation operations on transactions must be added.

Insertion of new records

The main part of the database is each year successively established. Establishment of a record occurrence is initiated by the arrival of an input transaction with a combination of identification terms (base object classes) which is non-existent in the data base. At the end of each year the main part of the database is deleted. No deletions are made during the year. Therefore the database can be regarded as being established once a year. The total work for this includes storage of about 857 000 new record occurrences linked into set occurrences.

Need for secondary storage space

When calculating necessary secondary storage space, space for identification and data, link addresses, overhead, overflow, indices, etc must be estimated. The need for secondary storage space was estimated to about 500 million characters.

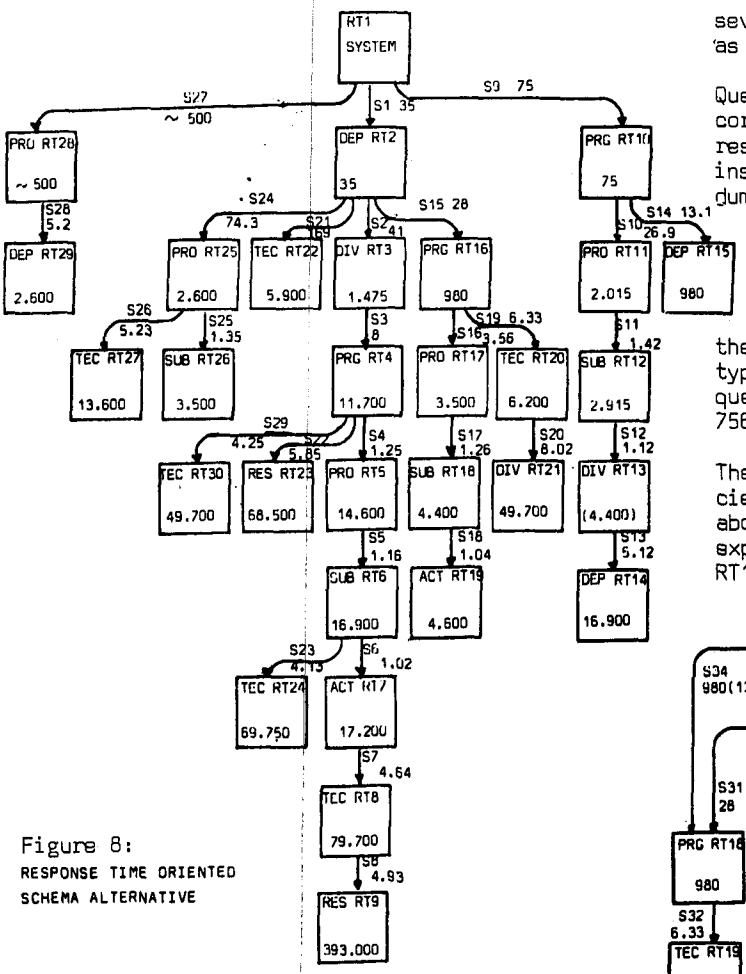


Figure 8:
RESPONSE TIME ORIENTED
SCHEMA ALTERNATIVE

4.4 Design of an updating oriented schema alternative

4.4.1 Design principles

This alternative was designed with the objective to decrease the number of database accesses needed for updating, under the restriction that stated response-time requirements should be satisfied.

Two techniques for this were used:

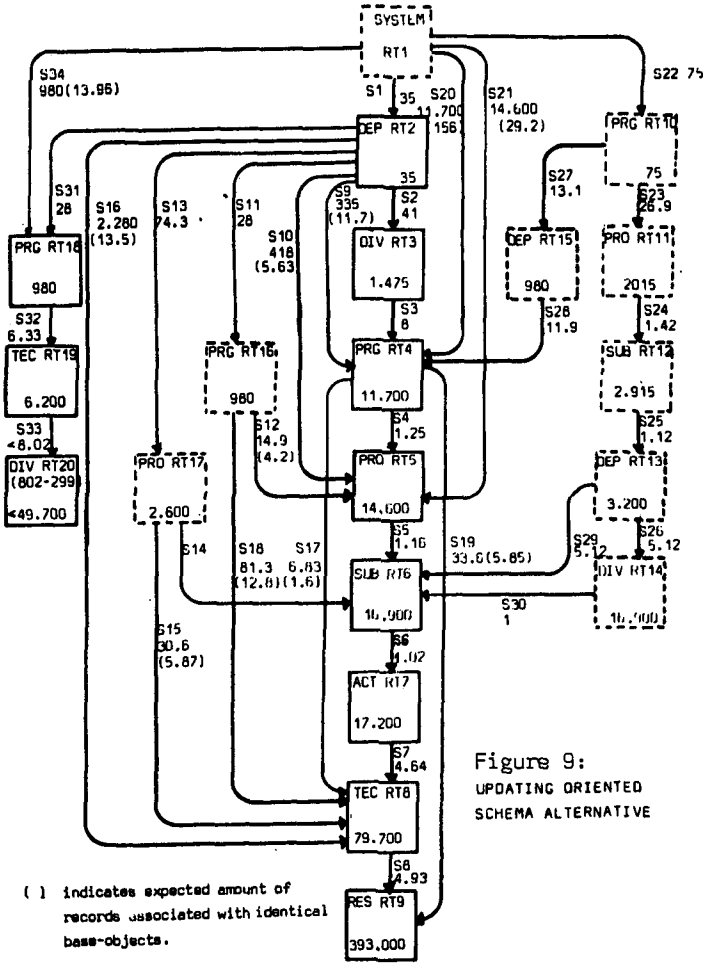
- Avoiding redundant storage of information
- Avoiding accumulations over some levels at update time.

Application of these two techniques (iteratively) resulted in the schema presented in figure 9.

The most frequently used path from the output oriented alternative was taken as basis for explicit storage of information (S1-S8). Most queries concerned this main path. An effort was made to, as far as possible, use this "main branch" for answering also the rest of the queries. Queries with the identification terms in the same sort order as the main branch, but with one or more levels omitted, could be, for instance, answered by new set types linking together the involved levels (f ex set type S10 in figure 8). This required, however, 337 dba:s for a particular query compared to 28 dba:s in the response-time oriented alternative. The method of linking records over several levels must, however, be used with caution as it may introduce extensive search work.

Queries that require searching in an order not corresponding to the main branch, or which are very resource consuming if the main branch is used can instead be effectuated by introducing one or more dummy record types as owner records to set types whose purpose is to provide a search path the record occurrences in the main branch where the required information is to be found. This arrangement is done to reduce the resource requirements for answering the queries through the main branch. For instance, by a dummy record type RT16, the number of database accesses for the query Q(DEP=dep, PRG=ALL, PRJ=ALL) was reduced from 756 dba as described above to 447 dba.

There were a few queries which could not be efficiently handled with the arrangements described above. For this purpose a second branch with explicit storage had to be defined, namely RT18-RT19-RT20 according to figure 9.



( ) indicates expected amount of records associated with identical base-objects.

Figure 9:
UPDATING ORIENTED
SCHEMA ALTERNATIVE

This schema alternative includes less record types than the output oriented alternative (20 against 30) but it contains on the other hand more set types (34 against 29). This alternative was considered more complex. Furthermore, it is more strongly built up according to the requirements and conditions which are known today and was therefore estimated to be more "sensitive" to changes of user information needs.

4.4.2 Quantitative properties

The total number of dba needed for answering all queries was estimated to about 1 000 000 dba/month. This is about 25% more than for the response time oriented alternative. Instead, this alternative gives a considerable saving of dba for updating: about 1 500 000 dba/month were needed. Furthermore, less work for sorting of transactions is required.

As this schema is more "complex" (e.g. some record types are members in more than one set type), insertion of new records consumes more resources. About 620 000 new record occurrences are established each year requiring about 1470000 linkings into set occurrences. The total need of secondary storage was estimated to about 300 M char. For more details see section 4.6.

4.5 Design of a compromise alternative

Considering the properties of the two alternatives, a compromise alternative was discussed. The strategy was to start with the response time oriented alternative, examine record types and set types taking part only in a few query-types and exclude them from the schema if the queries can be answered through other paths. For instance, RT30 and S29 were found to be involved in only one query-type namely Q (DEP=dep, DIV=div, PRG=prg, TEC=ALL). When searching from a unique (DEP,DIV,PRG)-combination over a S29 set occurrence, about 4.25 TEC-records are found i.e. 4.25 dba are required. Answering the query over set occurrences S4-S7 gives instead 11 dba, making a difference of 6 dba. If the estimated query-frequency is low, then this extra search effort is negligeable.

Elimination of records also save updating work. For instance, if RT30 and S29 are excluded the saving in updating work is approximately 170 000 dba/month.

The same type of discussions were carried out for other record/set type combinations and this resulted in a compromise schema alternative according to figure 10.

The number of dba needed for queries is here about the same as for the response time oriented alternative, but the number of dba for updating per month would be about 2 000 000 dba/month less. The need for secondary storage was estimated to about 400 M char.

4.6 Summary of properties considered in the evaluation

In this section some quantitative properties for the two "extreme" schema alternatives are estimated. An informal discussion of some qualitative properties is also included.

Qualitative properties

- the possibility to include new types of queries seemed simpler for the response time oriented alternative, as the procedures for the queries are simpler in this case
- new kinds of information is technically easier to include in the response time oriented alternative

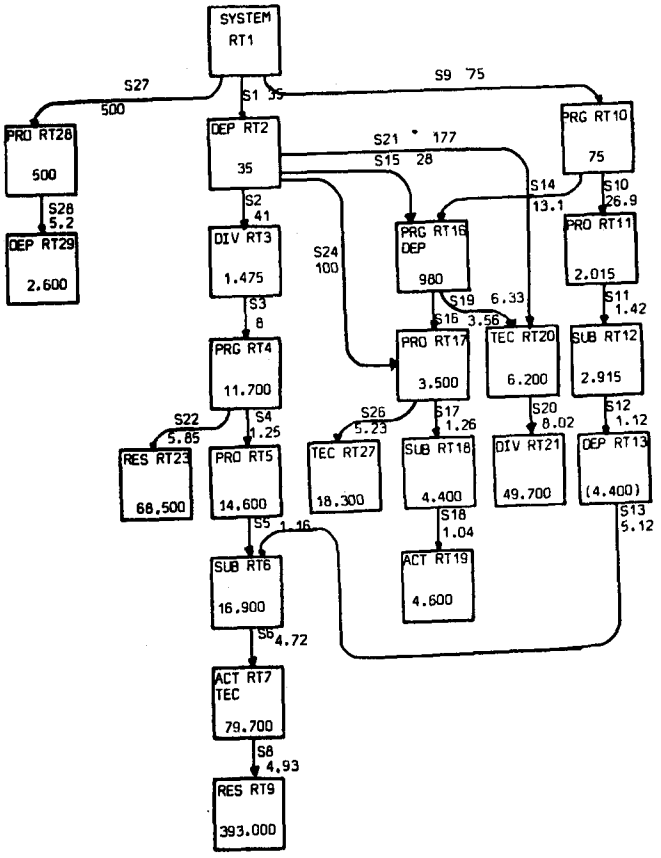


Figure 10:
COMPROMISE SCHEMA ALTERNATIVE

- new information objects will often cause only "local" reorganization in the response time oriented alternative, while introduction of new information objects might, in the updating oriented alternative, cause reorganization of the whole data base
- security: the response time oriented alternative can more easily than the updating oriented alternative be partitioned into separate parts which can be allocated to different physical devices
- implementation: the response time oriented alternative seems to be easier to implement as it is a pure tree structure and allows parallel processing
- the updating oriented alternative has simpler updating operations as most information elements are stored in one place only
- it was estimated that certain performance properties of the response time oriented alternative might be possible to improve considering the possibility to use parallel processing. In this alternative it is also possible to make more efficient choices of location modes and area allocations. Furthermore, a dba is supposed to have less probability to cause a secondary storage access in this alternative.

In summary, concerning the qualitative properties we found the response time oriented alternative - - to have several advantages compared to the network structure - the updating oriented alternative. The

compromise alternative was designed to preserve most of the performance properties of the pure tree structure.

Quantitative properties:

R = response time alternative (sect. 4.3)

U = updating alternative (sect. 4.4)

	R	U
Need for secondary storage space (M ch)		
- data	307	189
- record overhead	18	21
- sum	325	210
- total (150%)	488	313
Time for establishing the data base		
- amount of record occurrences to establish	856 830	620 050
- amount of links to be updated	856 830	1 467 055
- time[1] for establishment/transaction (sec)	0.6	2.4
- total time[1] for secondary storage accesses (hrs)	18	126
- total CPU-time (hrs)	10	18
Time for updating		
- dba/month	3 050 000	1 454 000
- time/month (hrs)	63.54	30.29
- time for sort (hrs)	3-4	1-2
Time for answering queries		
- dba/month for on-line queries	21 310	108 385
- total amount of dba/month	800 000	1 020 000

1) Presumption: 1 dba = 0.075 seconds

5 CONCLUSIONS. AREAS FOR FURTHER RESEARCH

The concepts and notation described in this paper were found useful for communication between the data base designer and the user as they made it possible to precisely and unambigously define required properties and functions of the data base. They were also found useful for definition and documentation of the information structure and information relations in an implementation independent way.

The outlined method for schema design, i.e. design of extreme alternatives followed by successive refinements in order to arrive at a feasible compromise, was practically applicable. The method made it possible to successively check that stated requirements were fulfilled. However, the large number of calculations necessary to analyze the consequences of schema restructurings raised the need of computer based tools.

Three areas within the schema design process can be recognized as presumptive goals for automation.

(1) Analysis of quantitative properties of a schema alternative such as

- secondary storage space,
- number of data base accesses for updating,
- number of data base accesses for answering queries, etc.

This task is trivial but burdening if done manually. A minor change in a schema alternative may affect the processing of all query and transaction types, implying that the quantitative properties of the alternative have to be recalculated.

With this kind of tool, a larger number of schema alternatives could be analyzed. Schema alternatives could also be analyzed in order to find out the consequences of changes in the presumptions, e.g. for changes in the frequencies of different transaction or query types.

(2) Automatic generation of schema alternatives.

Our knowledge in this area is today limited. However, the authors are, due to experiences from the application described in this paper, optimistic about the possibilities to develop formal procedures in the future, at least for restricted design cases.

Using a tool like this in combination with a tool for analysis of quantitative properties, the work of the data base designer could be restricted to definition of information structure and relations, statement of requirements and manual evaluation of generated alternatives. This evaluation must consider also the qualitative properties of the design alternative.

(3) Automatic generation of an "optimal" schema.

To be able to develop procedures for automatic design of an optimal schema for a specific application, we must have more insight in this multi-goal problem. The authors are not very optimistic about the possibilities to develop algorithms for automatic generation of "optimal" schemata within a foreseeable future. However, crude heuristics for design and evaluation of schema alternatives, considering also some qualitative properties can probably be developed.

In summary, the schema design process, as seen by the authors, will within a few years be aided by computerized tools for design of extreme-schema alternatives, refinements of these extreme alternatives, and calculation of consequences concerning quantitative properties. The possibility to design and evaluate a large number of schema alternatives will increase the chance to arrive at a good solution for a stated application problem.

REFERENCES

1. CODASYL Systems Committee, "Report of the CODASYL Data Base Task Group", ACM, April 1971.

2. Bubenko J.A. jr., "Some theoretical and practical observations in a data base case", in Lundeberg, Bubenko, "Systemeering -75", Studentlitteratur, Lund, 1975.

3. DATASAAB SYSTEM, DBMS LANGUAGE SPECIFICATION, Computer Systems D23 and D22 SAAB-SCANIA, Computer Section, S-581 88 Linköping, Sweden.

4. Senko M.E., Altman E.B., Astrahan M.M. and Fehder P.L., "Data Structures and Accessing in Data Base Systems", IBM Systems Journal, No 1, 1973, pp30-93.

5. Langefors B., "Theoretical Aspects of Information Systems for Management", IFIP Congress -74, pp937-945.

6. Sundgren B., "An Infological Approach to Data Bases", Ph.D. Thesis, Univ. of Stockholm, 1973.

7. Kerr D.S. (editor), Proceedings of the International Conference on Very Large Data Bases, ACM, Framingham, Mass., September 1975.

8. Langefors B., "Theoretical Analysis of Information Systems", Studentlitteratur/Auerbach, 1973 (3rd edition).

9. Codd E.F., "Further Normalization of the Data Base Relational Model", in Rustin (ed.), "Data Base Systems", Prentice Hall, 1973, pp33.