The subject of database management systems is receiving attention at an accelerating rate. The work of the COBOL Data Base Task Group (DBTG) has achieved major acceptance. This acceptance can be measured in terms of the number of computer systems for which DTTG software is now available, and the number of customers who have successfully installed application systems based upon this software. While the specifications are yet imperfect and development committees will be refining them for years, it appears to this author that a major step in the evolution of computerized information systems has been accomplished. Integrated database systems are here to stay.

Given DTTG as a fact, my theme is to suggest where database management systems might go from here. These remarks should not be considered as a survey of the possibilities, but rather as a description of three areas where the author has personal knowledge and which seem to have promise of a strong affect upon the future database systems. The first such area is development of a three schema technology to replace the two schemas of DTTG. This technology is being developed by ANSI/X3/SPARC/Study Group Database Systems. It is being driven by the thrust for data independence. The second area is concerned with the very active debate between the supporters of data-structure-set models which underly the DTTG systems, and the relational model. The third area relates to the development of hardware to assist in the repetitive operations characteristic of database systems. The database features of the new Honeywell Series 60/Level 64 computer is the specific case to be discussed.

The ANSI/X3/SPARC/Study Group Database System 1 was activated in November 1972 to examine the question of whether database systems were ready for standardization. To-date, this group has concentrated upon the question, "What is the basic architecture of the database systems necessary to support the user requirements of the late 1970's and the 1980's?" The first requirement was to provide a series of alternate data manipulation interfaces. These would include both procedural and non-procedural approaches. The second requirement, and the one I wish to describe in detail, deals with the requirement for data independence.

What is data independence? For some of you this may be a new term. For others, let me give you my understanding. Data independence is the end result of a mechanism which permits two seemingly contradictory conditions to coexist. The first is that the data structure, as declared and stored in the database, is allowed to continue evolving, to support the enterprise's own evolution and to support the evolution of the applications systems running against the database. This evolution is permitted while simultaneously maintaining the operability of all the existing application programs which were written, tested, and put into production sometime in the past. All that was done when the database upon which they were running was quite different in its structure and optimization. The alternative, which we consider to be undesirable and actually not feasible, is that every production program must be reviewed for each projected database change. Each program affected would have to be modified, recompiled and retested. Then the database restructuring could be carried out. This alternative is so cumbersome, expensive and error-prone that the net result has been database stagnation with information systems effectively halted on some plateau in their development.

The DTTG specifications, as now published by the CODASYL Data Description Language Committee, provides a partial solution to this problem. Each application program knows the database through its subschema data description. If the schema itself is modified in a way that the existing subschemata remain a proper subset, then the matter is one of modifying the mapping statements within the subschemata to make them again conform. This is illustrated by Figure 1 which shows a number of subschemata on the right side and two schemata on the left. The lines between represent the mapping statements that tie the subschemata back to their source within the data declarations of the schema. The addition of new field and set types to existing record types, and the addition of new record types are handled quite easily as there are no existing maps which reference them. However, when records are split or combined, or when record, field or set implementation technique is changed, then the existing maps are placed under heavy stress and many must be changed. The SPARC Study Group is recommending that the DTTG schema be replaced with two new schemas. These are called the Internal Database Schema and the Conceptual Database Schema. In addition, SPARC defines an External Database Schema which is equivalent to the DTTG subschema.

The conceptual schema is considered to hold the best
available definition of the basic information of the enterprise. It would include record, field, and data-structure-set declarations representing real world entities and their properties, whether or not they have become a matter of immediate interest to the actual computer processing of data.

The creation and maintenance of the conceptual database schema is considered to be a business operations' responsibility rather than an information systems operations' responsibility and administratively, would report to or be part of the enterprise's operations organization. A person, called the Enterprise Administrator, is assigned the responsibility for its creation and maintenance.

The internal database schema is the province of a person called the Database Administrator. He is charged with optimizing the performance, response time and media space requirement tradeoffs of the database system. He can change these optimization decisions from time to time without the requirement to modify, recompile or retest existing application programs.

Figure 2 illustrates the three kinds of schemas in the SPARC approach of things and the mapping between them. It should be contrasted with the DBTG approach illustrated in Figure 1.

The figure shows the internal database schemata on the left, the conceptual database schema in the center, and the multiple external databases schemata on the right. Now only the internal schema is changed to recognize new access or performance requirements.

The only mapping changes are in the one set of mappings between the internal and conceptual schemata. This advantage is really greater than is immediately obvious. One reason is that all of the performance, space, and response optimization aspects are concentrated in the internal schema. This permits the conceptual schema itself to take on a more stable and enlightened posture. This division of conceptual structure from storage structure is already observable in actions of the CODASYL Data Description Language Committee. Their publication of the Data Description Language Journal of Development has already discarded certain storage description elements of the schema which had been specified by the Data Base Task Group Report.

The list below suggests some of the optimization areas in which the Database Administrator could practice his skills:

- record implementation technique
- field implementation technique
- set implementation technique
- pointer implementation technique
- index implementation technique
- record placement statements
- file placement statements
- redundant data storage statements

For each conceptual record in the conceptual schema there would be a corresponding internal schema declaration which answers the following questions: (1) Is the record to be stored at all? (2) If it is to be stored, will it appear as a single linear record? (3) Will it appear as several separate linear records each with its own fields and storage rules. (4) Or, should it appear as a node in an inverted file structure? (5) If it is to be a linear record, should it be represented as a free standing database record, or (6) as a repeating group within a higher level database record? The separation of the conceptual record description from the internal record description has given the database administrator six choices, whereas the DBTG schema seems to limit him to one. Furthermore, it offers the opportunity to switch between them without obsoleting the external application programs or the maps that tie the external record descriptions back to the conceptual record descriptions. There are similar optimization declarations within the internal database schema with regard to implementation of fields, data-structure-sets, and indices. These
can be changed without the collapse of programs interfacing at the external database manipulation level.

In support of these changes in the internal database schema, there must be database utility programs which can transform the database from its old internal format to its new internal format efficiently. Both batch and incremental restructuring utilities must be available.

The previous discussion treated with the evolution of the stored data, as described by the “internal database schema,” to maintain efficient database operations. This was the province of the database administrator. Paralleling these changes, there also must be facilities by which the enterprise can change its own view of itself as represented by the conceptual database schema. These changes may result from structural changes in the enterprise, from changing views of the same structure or simply because the enterprise had been viewed narrowly and new information processing requirements have forced an expansion of viewpoint. The enterprise administrator is the person responsible for such changes. Many of these changes can be made without the loss of integrity to existing external database schemata. The anticipated result of most revisions to the conceptual schema is the phenomenon called “attribute migration” or “entity splitting.” I can best explain this phenomenon through the use of an example that we discussed at a recent meeting. In this example, the conceptual data structure goes through a series of changes as the business information is better understood and new processing requirements are determined. Figure 3 contains two data structure diagrams. The one to the left illustrates the conceptual data structure declared by the Enterprise Administrator.

This data structure diagram illustrates two conceptual record types, “company” and “personnel”, with a data-structure-set “a” representing the fact that each person is associated with exactly one company and that each company has a set of personnel.

The right side of the diagram is supposed to suggest that there are a number of external database schemata as declared by the Application System Administrators. Schema #1, the only one fully visible, is a very simple, “flat file”, view of the database. An application program accessing the database through this external schema would find a file of “pay-rec” records.

The conceptual database schema declaration of the “personnel” record type is illustrated in Figure 4. It should be noted that its field descriptions contain no declarations as to the type, length or mode of either the storage format of the fields or the format in which they should be delivered to a program which is accessing an instance of such a record. Note the I-D-S like 98 level entry which declares that the “personnel” record type is a MEMBER of the “a” data-structure-set type. It includes the declarations “SELECT UNIQUE OWNER,” “DUPLICATES NOT ALLOWED” which relate to the integrity of the database. There are no set ordering declarations, as ordering is a matter of interest to a particular procedural program and might be specified differently in each external schema. If the set were declared to be implemented in the internal database schema, then its ordering rules would be specified for that purpose in an internal set declaration of the internal schema.

Note that this description is expressed in I-D-S/COBOLese modified to convey the new concepts. All of the reserved words are written in capital letters, and the variable names are in lower case letters. Please accept this for illustration purposes only. The SPARC study group is not charged with inventing languages and will deny any responsibility for the syntax I invented for this example.

The text of the external database schema #1 would look something like that illustrated in Figure 5. The “FD” is supposed to represent an external file declaration which would hold all the personnel records for the company with that company’s code equal to “12498.” Sequence is not declared in this example but would be necessary unless some arbitrary sequence were acceptable.

```
FD personnel-file, WHERE company-code EQUALS 12498
  01 pay-rec, SOURCE IS personnel
    02 name X(16)
    02 office-phone 9(10)
    02 year-to-date-earnings PIC(99,999.99).
```

Figure 5—External database schema #1

From the collection of the Computer History Museum (www.computerhistory.org)
The 01 entry record declaration source statement, "SOURCE IS personnel", is the most interesting aspect of this declaration. This is the mapping statement that relates the "pay-rec" record type in the external schema back to a "personnel" record type in the conceptual schema. Presumably, if the names were identical, then the source statement could have been left out and the default would be to assume the mapping to a conceptual record type with the same name. In our example, the 02 entries declaring the fields have no source statement and do make use of the default assumption in that they match with similarly named fields in the conceptual record type "personnel."

Given the initial state of the conceptual schema as set forth in Figures 3 and 4, let us assume that the following changes were made to the conceptual database declarations, while remembering that the existing external database schemata must maintain their effectiveness. There were two changes. First, it was recognized that the detailed earnings of each employee were of interest from time to time. Therefore, a new conceptual record type is declared for the "payment" record. It has a field called "earnings". The second change was to recognize that the personnel of the company were persons in their own right. In fact, it was discovered at the merger of several companies that some of the persons held two jobs and personnel of the company were persons in their own right. "earnings" field actually to be stored in the "personnel" record? Or is it "virtual" and recomputed when needed? Clearly these are choices which the Database Administrator should have open to him without that choice having any logical effect upon the existing application programs.

If you think of these changes in terms of the internal database schema, a number of questions arise. Did the Database Administrator actually implement these new conceptual record types with the corresponding internal record types. Assuming that he did so, then the further questions can be asked. Is the person's "name" to be redundantly stored in the "personnel" record? Is it to be extracted from the "person" record whenever it is needed? This would be controlled by an internal database schema declaration. Similarly, is the "year-to-date-earnings" field actually to be stored in the "personnel" record or is it "virtual" and recomputed when needed? Clearly these are choices which the Database Administrator should have open to him without that choice having any logical effect upon the existing application programs.

The "ACTUAL" and "VIRTUAL" attributes in the DBTG schema illustrate the current state of the art. With the SPARC study group's separation of the storage aspects from the conceptual schema, the internal schemata would show the fields only if they were to be actually stored.

Another efficiency question is whether the "payment" record type should be implemented as internal database records in a data-structure-set, or as a repeating group (array) within the "personnel" internal database record. The OCCURS clause in the DBTG schema is the means by which this storage issue is declared. Under the SPARC

01 personnel.
02 name, SOURCE IS OWNER OF "b" SET.
02 year-to-date-earnings, RESULT IS SUM OF earnings of "c" SET, WHERE year OF payment EQUALS year OF current-date.
02 employee-number, PRIMARY KEY.
02 office-phone.
98 MEMBER "a" SET, SELECT UNIQUE OWNER.
98 MEMBER "b" SET, SELECT UNIQUE OWNER.
98 OWNER "c" SET.

Figure 7—Modified conceptual database schema
approach, the conceptual database schema does not recognize the concept of repeating group as it is considered a special and limited case of a record type within a data-structure-set. However, both the internal and external schemas would allow for their existence as they are useful in the storage and access of data and the programming of applications.

Let me take this evolution of the conceptual database structure forward two more steps. The first of these steps came about when the Enterprise Administrator decided to factor the address of residence out of the person record. For simplicity, let us assume that an address uniquely represented a place of dwelling. In other words, all persons with the same address were assumed to have the same residence. Figure 8 illustrates the addition of the "place" conceptual record type and the data-structure-set type "d" which associates occurrences of a "person" conceptual record type with a "place" conceptual record occurrence. The "address" in the "person" record will now migrate from the "person" record to the "place" record and the change is described with a source statement in the "person" record declaration.

Stretching the example the last step, it is now recognized that people move from place to place and that it is desirable to know current address as well as past addresses. Figure 9 illustrates the change in the nature of relationship between the "person" and the "place" records. Where there had been a n:1 relationship, there is now a n:m relationship. The new "address" conceptual record type serves to define this relationship. A result statement,

02 address RESULT IS address OF THE FIRST MEMBER OF "f" SET

would be associated with the "address" field description in the "person" record description of the modified conceptual data base schema. In addition, the source statement,

02 address SOURCE IS OWNER OF "e" SET

would be associated with the address field description in the address record description.

These two statements jointly define the derivation function for the "address" field of the "person" conceptual record. Please accept the language as having been just invented to support the example. It is the intent of these statements to find the first "address" record and from that "address" record to find its owner "place" record via set "e" to obtain the specific address value. Some very interesting questions can be asked concerning the meaning of modifying a person's address. Does this mean to create a new "address" record relating the person to a new place and further to create a new "place" record for the designated address if one does not currently exist? Or, does it mean to change the address of the place and all the other persons who currently or once lived there? Clearly, I believe this means the former. However, modifying the "address" field of a "place" record would mean that the dwelling has somehow had its address changed to correct an error, or perhaps all the houses on that street had been renumbered in some unification plan. This happened to me once when an entire section of town was renumbered.

I would like to shift the subject now to another very important area. This is an area where there is considerable technical debate at the present time. A formal debate was held at the ACM SIGFIDET meeting a year ago. This is the debate as to whether the Relational Model or the Data-Structure-Set Model is the best one upon which to build future database management systems. Codd of IBM
Research is the author and a very articulate sponsor of the relational model. Date of IBM (Hursley) has also written extensively on the subject. It would be presumptuous for me to try to explain the position of the supporters of the Relational Model in the limited space available. However, it is fair to say that the relational model has received tremendous support within the academic circles in the United States. One of the reasons is its use of set theoretic operations. This makes it very attractive to mathematically trained people. Another factor is its promise of usefulness in handling enquiries as a non-procedural language.

The competitive model in the current debate is the data-structure-set network model. This is the model that underlies both DBTG systems and Honeywell's Integrated Data Store. This is the model that I developed and have supported for a number of years. This model is so well established that it was described, in the ACM Communications news article reporting the debate, as being the "establishment" point of view. The most critical question for all of us is, "What is the significance of the debate?" Is it "full of sound and fury signifying nothing" or does it portend a radical change ahead? Let me oversimplify Codd's position as follows: "The data-structure-set model is based upon an access mechanism and has no place in a high level language approach to database systems. It will ultimately be replaced and disappear." I take a different approach to the subject. I consider data-structure-sets as representing natural relationships which exist in the real world. These are the relationships between teachers and students, between companies and customers, or as in the earlier example between persons and places. For information processing purposes, these relationships can be recorded by carrying around redundant data (the relational model) or by one of the many set implementation techniques. Possibly the redundant data fields used in the relational model should just be characterized as the 10th set implementation technique and closely related to the "phantom" technique.

This might be the ideal set implementation technique if and when hardware support for associative retrieval is achieved and becomes competitive with directly addressed secondary storage.

I consider the relational model and the data-structure-set model essentially compatible and subject to transformation from one form to the other. In the months since the debate last May, I have been attempting to clarify this transformability between the models. During this time, a third model, the Data Independence Access Model (DIAM) has also been studied for comparison. It was developed at IBM Research by Senko and his associates. It has been found to offer some interesting perspective upon the debate. The thing that is most interesting is that the relational model and the data independence access models appear to be completely at odds with each other. At best, they are at the opposite ends of a spectrum. By comparison, the data-structure-set model seems to be an effective hybrid between these two extremes.

Can this amalgam be explained in a few words? Let me try. The relational model, at first glance, appears to be concerned with only fields and records. Of course, it is the way in which one uses the fields that makes the system work.

Figure 10 illustrates the use of the relational model to describe certain aspects of information about a person. Each row is called a "tuple" and is similar to a record. Each column is a "role" on a "domain" and is similar to a field.

The DIAM model seems to work only with single field records and the 1:1 directed relationships that exist between these single field records. Fields are only used for the unique or primary key identification of a record. The same information about people used in Figure 10 for the relational model is recast in Figure 11 for the DIAM model. The ovals represent entities and their unique identification. The arrows between the ovals represent directed 1:1 relationships between a pair of such entities.

The data-structure-set model, as implemented by DBTG
and I-D-S, offers the database designer the choice of whether to represent a property of an entity as a field or as a data-structure-set according to which approach seems more natural to him. Figure 12 recasts the information used in the last two figures into the data-structure-set model. In this case, the blocks represent records and their associated data fields. The multi-headed arrows represent the 1:n relationships between a record in an “owner” role and the records in the “member” role of a data-base-set.

In the example I have chosen to model “year of birth” and “sex” attributes as fields and the “parent:child” relationship as a data-structure-set.

In the prior example describing the conceptual database schema, the splitting of entities and the resultant migration of attributes was the prominent action described. What actually happened was that the Enterprise Administrator progressively shifted his viewpoint from a mostly relational model viewpoint toward a more DIAM-like viewpoint. However, during this entire time, the conceptual schema was always a mixture of fields, records and data-structure-sets. It was always a data-structure-set model type of compromise.

The third area which I want to discuss deals with the use of hardware and firmware to assist in handling time consuming operations which appear frequently in database systems. While progress in this direction is limited, a trend is suggested. The virtual memory hardware pioneered by the Ferranti Atlas and the Honeywell MULTICS systems provided hardware support for addressing a paged database. The IBM 370 has followed this course. While the address space of these machines is too limited in size for most databases, they do represent a significant step forward. Time should take care of the size problem.

A second example of hardware and firmware support was announced last year for the Honeywell Series 60 Level 64. This support focuses upon the data independence issue discussed earlier. The most time consuming aspects of supporting data independence is the need to continually transform data from the stored format (internal schema) to the format in which the program expects it (external schema). This requires a field by field examination and transfer. The H60/64 has provided a rudimentary set of field instructions which are controlled by data descriptors. These are the “MOVE A TO B” and “COMPARE A WITH B” instructions. These instructions operate through a pair of field descriptors. Each descriptor identifies the offset of the field within a record or within the program’s work space or stack frame and identifies a base register which establishes the record’s, program’s or stack frame’s location within the processes address space. It also states the field’s length and data type. (BCD, ASCII, EBCDIC, packed decimals, ...). Thus during a move with type control, all required conversions are made and the result is truncated or filled with blanks or zeros according to the description of the receiving field.

The H60/64 also has a “hashing” or randomizing command in the instruction set. This instruction is designed to operate with fields of various lengths and with multiple fields which are not contiguous. The instruction is based upon a folding, shifting, negating algorithm which is particularly sensitive to the fact that identifiers, especially numeric, stored as character strings do not make use of the high order bit patterns. The algorithm is patterned after the I-D-S randomizing routine which has more than ten years of field exposure on three different machine architectures and which has almost universal acceptance by I-D-S users on those machines.

The selection of these three commands to be supported by hardware was based upon instrumentation of actual I-D-S applications on the H6000 line. The instrumentation determined which software routines would yield the greatest return if reimplemented in hardware.

In summary, the art of database management is evolving and has made significant progress. The DBTG type systems pioneered by the Integrated Data Store have become available from a number of computer manufacturers. The evolution of these DBTG systems to the ANSI/SPARC tripartite data descriptions is under way and should continue. The current debate on data-structure-set vs. relational models of data will find its greatest contribution in the development of a greater understanding of the nature of data. This is a little like the continued study of particle physics; it is a seemingly unending struggle for fine and finer resolution. The introduction of database commands into the instruction sets of our computer is a yet unproven aspect. Unproven in that their motivation is one of efficiency. If the operations which are assisted by this hardware are executed with sufficient frequency, then they pay off. If the frequency diminishes, then the hardware investment is lost. It would now seem that money invested in “scatter/gather” instructions and formatted track discs to support unit record processing at the physical I/O transfer level may have been wasted as their utilization is very low in the new database systems. However, if some advances are false, still others are true and pay their way. It appears to this author that there is still a good yield to be gained in providing new hardware assistance in support of well defined database functions.
REFERENCES

3. CODASYL Data Base Task Group Report, April 1971 (various sources).