OBSERVATIONS ON THE ATTRIBUTES OF DATABASE SETS[*]

By

Robert W. Taylor
Computer and Information Science
University of Massachusetts
Amherst, MA 01002

COINS Technical Report 75A-1
January 1974

[*]The research reported in this paper was supported in part by the National Science Foundation under grant No. NSF GJ 41829 and in part by a grant from Control Data Corporation.

OBSERVATIONS ON THE ATTRIBUTES OF DATABASE SETS

Robert W. Taylor
Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01002

1. INTRODUCTION

Sets have been one of the most widely debated aspects of the 1971 CODASYL DBTG Report [1] and its schema language successor defined in the CODASYL Data Description Language Journal of Development [2]. Depending on one's viewpoint, sets are the most powerful and simultaneously the most dangerous data structuring feature of the report--powerful because they allow network structures among a diversity of record types, dangerous because they tend to add complexity to the programmer's task. Further, sets tend to bind program to data structure rather tightly in the sense that the allowable database traversals invoked by various forms of the FIND verb depend on the continued existence of these sets--thus limiting a database administrator's ability to change the explicitly represented access paths. (Codd in [3] called this phenomenon "access path dependence.")

To review briefly, a <u>database set type</u> is a declared association between one designated record type, called the owner record type, and one or more additional record types distinct from the owner, called the member record type(s). A <u>database set occurrence</u> is one occurrence of the owner record type together with zero or more occurrences of each member record type(s). A member record occurrence may be associated with at most one occurrence of a given set type.

By declaring a set type in the database schema language, a database administrator implies the existence of an access path from the owner record occurrence through the associated member record occurrences such that a programmer, if permitted, may traverse the access path using Data Manipulation Language statements such as FIND NEXT, FIND PRIOR, FIND Ith and FIND OWNER.

Along with its name, a database set carries a number of additional attributes. Among these are:

1) Ordering - the member record occurrences may be declared to be in a given order, which will then be maintained by the system.

2) Prior Processable/linked to owner - linkages are maintained by the system to allow efficient processing in the "prior" and "owner" as well as the "next" directions.

3) System Owned (singular) - the "owner" is the database management system. This is useful for providing an access path through all occurrences of a given record type.

4) Dynamic - A set can have an occurrence of any record type as a member.

5) System maintained insertion/removal - record types in their capacity as members of a set type may have attributes which govern whether or not a programmer must explicitly INSERT them in a set occurrence, and whether or not they may be REMOVED from a set occurrence.

The rest of this paper will examine these last two database set attributes in some detail. There appear to be certain problems in the definition of DYNAMIC sets, and the paper suggests revisions which, in the author's opinion, do not limit the usefulness of DYNAMIC sets, while at the same time avoiding some of their problems. With respect to system maintained insertion/removal, suggestions are made which would allow the extension of these capabilities in certain cases. Finally, a few remarks are made concerning additional attributes which could aid in set occurrence verification by database utilities.

2. DYNAMIC SETS

A DYNAMIC databse set has a declared owner record type, an ordering, and perhaps privacy lock and ON condition database procedures to be called. It has no declared member record types. The semantics of a DYNAMIC declaration is that an occurrence of any record type may be INSERTed into and REMOVED from the dynamic set associated with a given occurrence of the owner record type.

Such a facility is useful in applications which need some "scratch" sets in which record occurrences may be collected. Because of the CODASYL architecture, in which all database sets are declared at schema definition/schema compile time, the provision of such "scratch" access paths must be the responsibility of the database administrator. A programmer has no ability to add additional "scratch" sets associated with his particular run-unit. But at the same time, it would be an unwarranted burden on a database administrator to have to anticipate and name the scratch sets that might be needed for each application program; hence the facility for a few scratch sets which can have any type of record occurrence as a member.

We turn now to some of the problems arising from the DYNAMIC set attribute. It is interesting to note that the 1971 DBTG Report [1] referred to this attribute as POINTER ARRAY DYNAMIC and proposed a conceptual implementation strategy as shown in Figure 1. This may appear at first to be a feasible implementation strategy. Note that there is no access path from the member record occurrences back

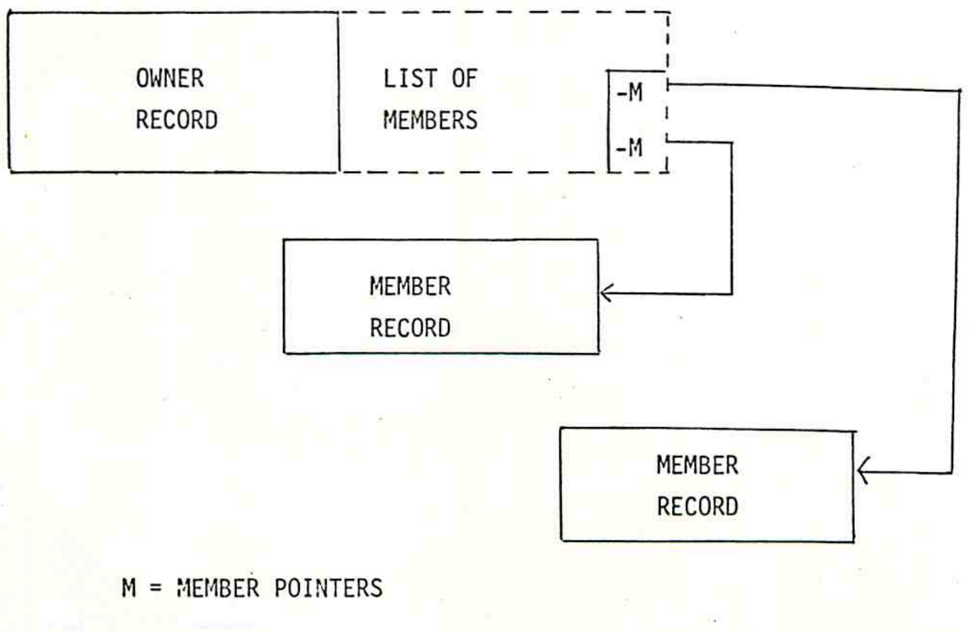

Figure 1. Dynamic Pointer Arrays in the DBTG Report.

to the associated owner record occurrence. One might argue that this is appropriate since, to do otherwise, would mean the allocation of one pointer (database key) field for every DYNAMIC set for every record occurrence in the database*, on the off chance that a given record occurrence might participate in every dynamic set. On the other hand, since the access path from member to owner does not exist, various traversals are precluded. The DBTG report [1] acknowledges this (page 51): "there is no provision for declaring that the owner record of a dynamic set is to be directly accessible from its members." However, the implications of this restriction are severe. For example, the DBTG report states (p. 234) that an INSERT statement must fail if the object record occurrence is already a member of some occurrence of the set into which the object record is being inserted. Under normal circumstances, this is easy to enforce, since there will be a pointer field in the record occurrence which, if non-null, indicates that the record already participates in some set occurrence (which one can be ascertained by doing a FIND OWNER). If dynamic sets are implemented as in Figure 1, no such field exists and further, there is no way to know without exhaustive search whether or not a record occurrence already participates in an occurrence of a dynamic set.

Other anomalies requiring exhaustive search are easy to construct: consider the processing necessary to answer the questions:

a) IF RECORD MEMBER OF dynamic-set

b) FIND NEXT RECORD OF dynamic-set SET
   (when the current record has been located through some independent path.)

Thus even dynamic sets must have a direct path to the associated owner record occurrence, and this has been recognized in [2] by removal of the original provision.**

Thus, with the access path from owner to member being a necessity, we are back to the dilemma of many pointer fields, most of which are null most of the time. There are, of course, several approaches to this problem. One would be to add a system pointer field to each member occurrence, regardless of type. This field would point to a variable length vector of pointers which would grow and shrink dynamically as records came and went from dynamic sets. Such a strategy is illustrated in Figure 2. For example, dynamic set 1 might be declared prior processable with a general strategy of chaining while dynamic set 2 might be implemented using pointer array techniques. Such a strategy is, of course, flexible and could even be used for all MANUAL sets, not just dynamic ones. It has the accompanying space management inefficiencies.

A second strategy, similar to that adopted in one existing implementation, [5], would be to reserve a bit map and k pointer fields in each member record, where k is some number less than or equal to the number of dynamic sets. The bit map would be used to indicate which dynamic sets are participated in with the k pointer fields used to hold the relevant pointers. Membership in more than k dynamic sets simultaneously would be an error. Once again, all MANUAL members could use such a strategy together with dynamic members, and the inefficiencies are somewhat reduced (see Figure 3).

*This is only approximately true, since if there exist m dynamic sets and a given record type is declared owner in n of them, $n \leq m$, then occurences of this record type need only reserve $m - n$ pointers for the other dynamic sets. If the dynamic sets are all singular, then the statement holds.

**However, this author can find no explicit provision in [2] of what access paths must be provided (by whatever storage organization strategy) when a set type

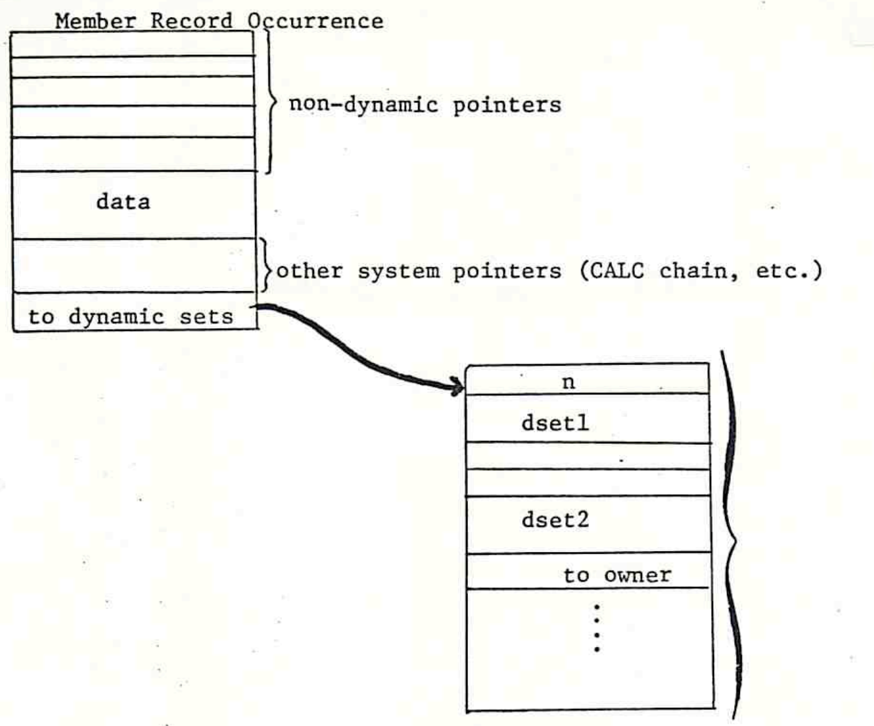

Figure 2: Dynamic vector for participation in dynamic sets.

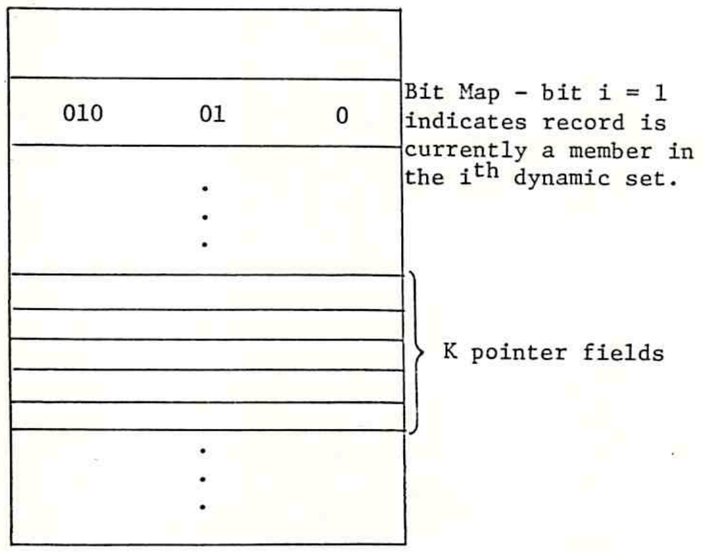
010 01 0
Bit Map - bit i = 1 indicates record is currently a member in the ith dynamic set.
K pointer fields

Figure 3. Bit map and membership in at most K dynamic sets.

A still better solution, in the author's opinion, is to return to the basic idea of a scratch set and how a programmer would like to use it. It seems a case can be made for the dynamic set, defined at <u>program</u> compile time rather than schema compile time, and having only one occurrence per run unit. The set is dynamic in that it behaves much like a temporary area--the set no longer exists when the run unit ceases its interaction with the database management system.

Such an architecture would have the following advantages:

1) It would give a programmer the ability to define as many scratch sets as he found convenient, with no long term space commitment in the database itself.

2) The ability to enforce which record types could and could not be members of which set types would remain--a valuable run-time check on database integrity.

3) There would be no "stray" sets around. This author can find no prohibition against a run unit inserting various record occurrences into dynamic sets and then terminating without removing them. This would certainly lead to database clutter, as well as confuse later run-units trying to re-insert the same record occurrences into different occurrences in the dynamic set. Such clutter would probably occur since there is currently no DML statement to "empty" a set occurrence, so the "emptying" of a set is a matter of some programming.

4) Rather than having a programmer use scratch files outside the reaches of the database management system, the programmer may stay "inside" the system and thus profit from the extended database recovery and other bookkeeping services. In this sense, the special case of dynamic sets proposed here is rather similar to the idea of temporary areas as proposed in both [1] and [2]. The approach here is to be preferred, however, as it is not bound by the restriction ([2] page 3.25) that "records in a temporary area cannot participate either as owner or member records, in sets which contain records that are not in temporary areas." Note also that there need be no copying of data records for the purposes of entering them in scratch sets. This must be done when using temporary areas since the same record occurrence cannot simultaneously exist in two areas.

5) Use of the SORTED INDEXED and SEARCH KEY facilities would allow programmers to specify the dynamic creation of extra indices. This could be particularly useful if a database administrator wished to experiment with extra indices which, if they truly improve performance, will be added to the schema and maintained permanently when the schema is next revised.

6) The ability to define extra scratch sets at program compile time greatly enhances the ability of a macro-oriented preprocessor [6]. For example, if macro statements are added to allow selection of records using Boolean selection expressions or other macros are allowed for set operations (and a relational view of data [3]), scratch sets become extremely useful.

An architecture for supporting DYNAMIC sets of this type is shown in Figure

---

is declared. Presumably at least one path from owner record occurrence to associated member record occurrences and vice-versa. If this is not so, then the revised Data Manipulation Language of [4] cannot be supported.

4. Basically, using a pointer array strategy, the members of the single occurrence of the dynamic set are maintained by the database management system for the

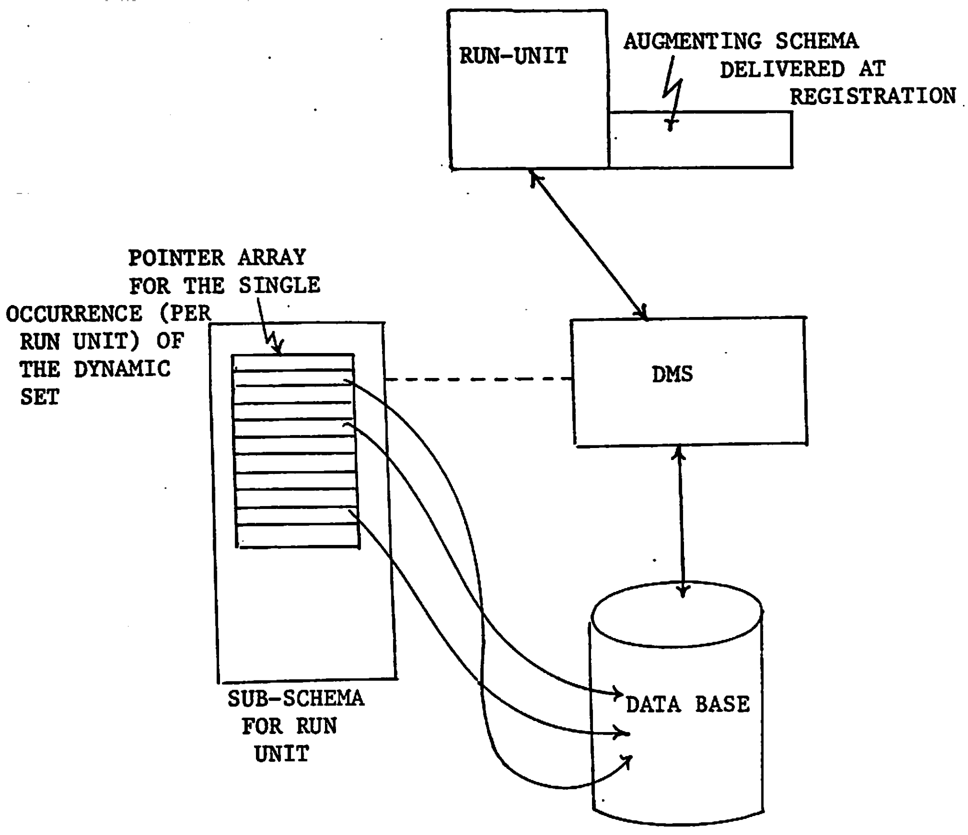
RUN-UNIT
AUGMENTING SCHEMA DELIVERED AT REGISTRATION
POINTER ARRAY FOR THE SINGLE OCCURRENCE (PER RUN UNIT) OF THE DYNAMIC SET
DMS
SUB-SCHEMA FOR RUN UNIT
DATA BASE

Figure 4. Architecture for dynamic sets.

given run unit. Member record types may be INSERTED and REMOVED and there is no "owner pointer" within the member record type. However, the ability to get to the "owner" pointer array is maintained because there is only one set occurrence for the given run unit.

Because this form of dynamic set is programmer rather than database administrator defined, there remains the problem of informing the database management system of the existence of this new set. This could be handled as follows. At execution time, during the "registration" process that run-units undergo in order to tell the database management system of their existence, an "augmented" subschema would be delivered. Contained therein would be the names and other allowed attributes of these additional dynamic singular sets. Thereafter, the behavior will be as if these singular sets were contained in the schema except that the singular occurrence will be deleted at the end of the run-unit execution. Note that because the set is singular, there is no problem in interpreting a FIND NEXT, FIND PRIOR, etc., because there is only one "pointer array"* for this singular

*Of course, it need not be a pointer array, strictly.

set (for this run unit) and that is locatable by the system. With respect to enforcement of the rule against duplicate INSERT of the same record occurrence into a given set occurrence, this can be enforced in logarithmic time by searching the pointer array or more quickly by keeping a bit map in the record occurrence (though this latter method puts an upper bound on scratch sets as well as raising some additional concurrency problems.)

There is, of course, an additional burden placed on the compiler (or pre-compiler) in that it must provide the compiled version of the augmented schema. However, this would seem to be a reasonable price to pay for these added facilities.

We thus conclude that DYNAMIC sets as they currently are constituted in [2] could more usefully exist when defined externally to the schema itself. Rather, dynamic sets should augment the schema. Thus DYNAMIC should be removed from the schema language, but some provision should be made in the documentation to allow for the concept of augmentation. That is, system designers should not be misled into thinking that because DYNAMIC sets are not in the schema language that the system does not support dynamic sets.

3. SYSTEM-MAINTAINED SETS

One of the reasons database management systems exist is to provide common services for programmers. An example would be the set attribute AUTOMATIC combined with MANDATORY. When a record occurrence is stored, it will automatically be included in those sets in which it is an AUTOMATIC member, with set occurrence selection being governed by the associated SET SELECTION clause in the set definition. If it is a MANDATORY member, it cannot be removed from the set, and will automatically be deleted under some variations of the deletion function when the owner is deleted. Such services are properly included in a data management system: they not only relieve programmers of routine bookkeeping details, but also help the database administration staff to insure database integrity--they need not worry so much about making sure that each programmer remembers to do each insertion that he is supposed to.

We ask then, if system help with bookkeeping (i.e., insertion/removal) is desirable, how difficult would it be to extend the system's capabilities for these semantic rules, and can this be done in a way that maintains the "inessentiality" [7] of the set itself.

Consider the case shown in Figure 5. Here we have several sets being used to

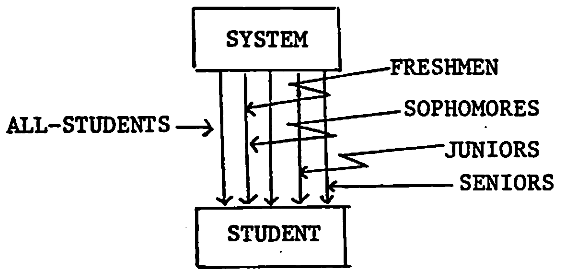

Figure 5. Sets used as state indicators.

partition a set of student records into classes based on accumulated credits (note: not all students start with zero credits because of transfers and the granting of advanced standing). The use of multiple set types from an owner to a member to indicate a "current state" of some variable is not uncommon. Yet as currently constituted, the student record must be declared a MANUAL member in each except that set which collects all students. Consider also the code that a programmer might write for such a situation. This is shown in Figure 6. What would probably happen in a current installation is that the database designer/database administrator will know at design time the criteria under which a record will be a member in such a set. This will be communicated to the programmer via memoranda

```
STORE   STUDENT
IF CREDITS IS GREATER-EQUAL TO
       ZERO AND LESS THAN 30 THEN
           INSERT STUDENT INTO FRESHMEN
ELSE IF CREDITS IS GREATER-EQUAL TO
           30 AND LESS THAN 60 THEN
           INSERT STUDENT INTO SOPHOMORES
                  .
                  .
                  .
```

Figure 6. Typical conditional insertion code.

etc., and the programmer will write the code. How much easier it would be for the criteria of membership to be written in the schema language. The code would then define a database procedure to be executed on STORE, or, if database procedures are not implemented, the schema code could be inserted at compile directly after any STORE statement of the designated record type. The following advantages of this "conditional automatic" membership can be noted:

1) Increased data independence. The programmer need not be aware of these extra access paths. They can be removed from the schema (after all, they are inessential) without impacting the program except, perhaps, for recompilation.

2) A more complete schema definition of why the set exists and the conditions for membership--and in a form that allows enforcement of these conditions.

3) The ability to generate validity checking programs automatically from the schema definition. This will be discussed more fully in section 4.

A similar set of "conditions" should be included in the MEMBER record clause to define the actions to be taken ON DELETE of the owner record of the set, ON MODIFY of the member record, etc.

Thus it appears such a facility would enhance the schema language. There remains the problem of guaranteeing the inessentiality of the set, for reasons of data independence and compatability among data models [8]. From the definition of an inessential set [7], it must be possible to determine the set occurrence in which a newly stored record will be inserted directly from the stored record. From the definition of legal set occurrence, there must exist at most one owner record occurrence having values such that the "conditions" are satisfied. Thus a necessary and sufficient condition is that an item (or concatenation of items) in an owner record which serves as the match condition governing conditional automatic insertion of member records, must use the DUPLICATES ARE NOT ALLOWED FOR database identifier -1, -2, ...* In the current DDLC schema language, this clause

*And, of course, there must be an independent access path to occurrences of is only available when the record type participates as a member type in a set type. Thus we conclude the following:

> If semantic rules governing insertion and deletion of records in sets are to replace the AUTOMATIC/MANUAL and MANDATORY/OPTIONAL attributes, as has been suggested [8], and the idea of inessential sets is to be maintained, then the owner record in the set must not have duplicates on the items participating in the rule. Hence the owner record must be a member in a singular set which enforces these duplication conditions (one duplication condition for each set owned) or else a similar duplication clause and and null control must be added to the record description clause of the schema language.

It is also worth noting that because of this condition, it is likely that records will be members in a variety of singular sets (unless the language is extended more fully in the record section). If the problem is dealt with via singular sets, it becomes paramount that pointer space not be committed in member records for each singular set. The storage structure or device media control language must provide for such an option.

4. SET SEMANTICS

Researchers in data management systems seem to be unanimous in their agreement that at some level of specification the concept of an allowable access path must be included. By access path specification, we mean a specification of the circumstances under which it is meaningful for a user (or the system in the user's behalf) to move from one record occurrence to another (possibly of a different type) within the data base. Thus, for example, the relational model has the notion of conformable domains, and one would be surprised if it were allowable to take the equi-join of age (in years) with melting point (in degrees Kelvin). Similarly, the schema language embodies the notion of access path in the database set, and, for better or worse, the definition of such an access path in current (non-relational) systems tends to be equated with the explicit representation of that access path.

We do not intend to debate that point here, however. Rather, our intent is to discuss the lack of "meaning" currently associated with the existence of a set in the schema language. This is perhaps best done by example. Consider the data structure diagrams shown in Figure 7. One is the traditional parts explosion of a

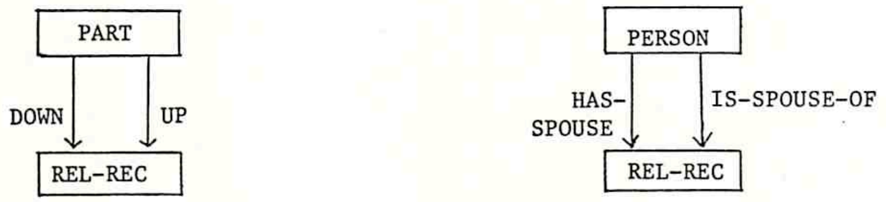

Figure 7. Syntactically equivalent data structures which will have differing instance structures.

---

the owner record type.

part made of sub-parts made of, etc., while the other is self-explanatory. Note that syntactically, these two cases are identical, yet we know that the occurrence structures which conform to these two data structure diagrams will be different--in one case there will be symmetry (if A is married to B, B is married to A) while in the other case, it had better not be true and we should have anti-symmetry.* Such specification of the legal instances is particularly important in the checking of database validity by background procedures.

Figure 8 offers another example. It is probably true that the set of records

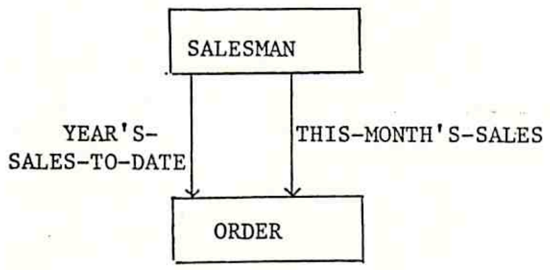

Figure 8. Sets used to represent the "state" of an object.

in an occurrence of the THIS-MONTH'S-SALES set is a subset of the set of records in the corresponding YEAR'S-SALES-TO-DATE set occurrence. Yet there is not way to state such a condition in the schema language. If there were, then the system could check the truth of this assertion.

The overriding point is that, in the definition of a set type, the schema language should provide an opportunity for the definition of the precise conditions which the access path embodies. (Other data structure models, e.g., [9], [10], have realized the importance of such a facility and have included it in their language). When such a facility is included, the opportunity for a number of system provided services arises. Two of these are that:

1) The system has a greater chance of determining complex traversal for a casual user because it can determine what predicate is true of participants on that access path and hence whether that path will yield records which are relevant to the question (see, e.g., [11]).

2) The system can increase its validity checking mechanisms because it has a definitive test for valid versus invalid structures.

Drawing from the DIAM model [9], we can make the following statements about necessary features of the semantic rules that should be added to the language as an attribute of sets:

1) There should be a subsetting semantic rule (as in Figure 8) for those sets which are selecting a subset of some other set in order to speed access. It follows that every member record in the "subset path" is also a member of the "full set" access path for the database to be valid.

2) When the set is connecting dissimilar (or the same) entities (either

*This example continues to hold, even if the necessity for the "relationship record" is removed from the language.

in a 1:n or m:n fashion) there should be a rule stating what condition must hold in order for such an access path to be built. This is equivalent to saying that all sets must be inessential.

3) There should be a rule stating when two paths to a set of record occurrences must "commute." An example is shown in Figure 9, where the set of SALE records reachable via the two paths should be the same. Such a rule provides an additional cross check on database validity.

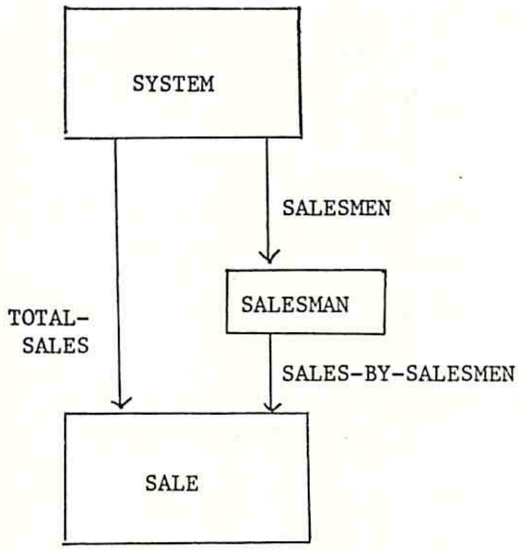

Figure 9: Commutativity of access paths.

5. CONCLUSION

It has been suggested that, in the long term, a single DDL should evolve to serve a variety of users, in particular, the "relational" user and the "navigational" user. While such a goal is certainly desirable, it can be seen from the arguments above that the current DDL, even when various "features" are not used, is not currently suitable to achieve these goals. In particular, the treatment of DYNAMIC sets is, in the author's opinion, ill suited to either class of user. Also, the desire for inessential sets implies enforcement of more powerful duplication control rules than are currently available except through extensive use of singular sets. Finally, the semantic rules which govern the behavior of records in sets are not adequately covered in the DDL. None of these problems are insurmountable, but all must be resolved in a future design.

REFERENCES

[1] CODASYL Data Base Task Group Report, April 1971, available from ACM.

[2] CODASYL Data Description Language Journal of Development, June 1973, available from U.S. Government Printing Office, Sup. of Documents, (#C13.16/2: 113)

[3] Codd, E.F., "A Relational Model of Data for Large, Shared Data Banks," Comm.

ACM, 13:6, June 1970, pp. 377-387.

[4] CODASYL Data Base Language Task Group, COBOL Data Base Facility Proposal, March 1973, available from Canadian Government.

[5] Univac Corporation, DMS 1100 Schema Definition, UP7907.

[6] Taylor, R.W., "Data Administration and the DBTG Report," Proceedings 1974 ACM SIGFIDET Workshop, May 1-3, Ann Arbor, MI, available from ACM.

[7] Codd, E.F., and C.J. Date, "Interactive Support for Non-Programmers, the Relational and Network Approaches," 1974 ACM SIGFIDET Workshop, Ann Arbor, MI, available from ACM.

[8] Nijssen, G.M., "Data Structuring in the DDL and the Relational Data Model," IFIP TC-2 Working Conference on Data Management Systems, North-Holland Publishing Company, Amsterdam.

[9] Senko, M., E.B. Altman, M.M. Astrahan, and P.L. Fehder, "Data Structures and Accessing in Data-base Systems," IBM Systems Journal, 12:1, 1973.

[10] Mathers, J.O. and R. W. Taylor, "The DIAM Model: A Review and Discussion," COINS Technical Report 74A-1, University of Massachusetts.

[11] Astrahan, M.M. and S.P. Ghosh, "A Search Path Selection Algorithm for the Data Independent Accessing Model," Proceedings of the 1974 ACM SIGFIDET Workshop, Ann Arbor, MI, available from ACM.

ACKNOWLEDGEMENT

The research reported in this paper was supported in part by the National Science Foundation under grant No. NSF GJ 41829 and in part by a grant from Control Data Corporation. The author would like to acknowledge helpful discussions with R. Eckhouse, and A. Karshmer of the University of Massachusetts.

BIBLIOGRAPHIC DATA SHEET	1. Report No. NSF-OCA-GJ41829-75A1	2.	3. Recipient's Accession No.
4. Title and Subtitle OBSERVATIONS ON THE ATTRIBUTES OF DATABASE SETS			5. Report Date date of issue January 1975
			6.
7. Author(s) Robert W. Taylor			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Computer & Information Science Dept. Graduate Research Center University of Massachusetts Amherst, MA 01002			10. Project/Task/Work Unit No.
			11. Contract/Grant No. GJ41829
12. Sponsoring Organization Name and Address National Science Foundation Office of Computing Activities Washington, D.C. 20550			13. Type of Report & Period Covered
			14.

15. Supplementary Notes Presented at the IFIP TC-2 Special Working Conference: "A Technical In-Depth Evaluation of the DDL" in Brussels, January 13-17, 1975.

16. Abstracts

Sets have been one of the most widely debated aspects of the 1971 CODASYL DBTG report. This paper discusses various of their shortcomings with special emphasis on "dynamic" sets and sets that are "system maintained" through the use of an "augmented schema" is introduced to overcome some shortcomings of dynamic sets, and extensions to the concept of AUTOMATIC membership are proposed. A discussion of data base validity checks is also included.

17. Key Words and Document Analysis. 17a. Descriptors

CODASYL DBTG Report
data base management systems
data structure design

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement Release unlimited	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 14
	20. Security Class (This Page UNCLASSIFIED	22. Price