HIGH LEVEL DOMAIN DEFINITION IN A RELATIONAL DATE BASE SYSTEM

Dennis J. McLeod
IBM Research Laboratory
San Jose CA

20 January 1976

ABSTRACT

A relational data base is herein defined as a collection of normalized relations (relations in first normal form) and a collection of domains. A normalized relation may be viewed as a table, wherein each row of the table corresponds to a tuple of the relation, and the entries in a given column belong to the set of values constituting the domain underlying that column. The domains of a data base have an abstract existence apart from the data base relations.

The data base also includes various types of semantic integrity rules, which specify additional properties of the data in the data base. One such type of semantic integrity rule is the domain definition. A domain definition includes the precise description of the set of values (objects) constituting the domain. In a normalized data base, all domains are sets of atomic data values. A domain definition also includes a specification of the ordering on the values in a domain, for comparability purposes. In addition, a domain definition contains a specification of the action that is to occur if an attempt is made to violate the restriction that every entry in each column of a relation must be from the underlying domain of that column.

A nonprocedural language permitting the high level expression of domain definitions is defined. Language details and examples are presented, and the syntax and informal semantics of the domain definition language are given. This approach to domain definition is analyzed in terms of its impact on other aspects of data base semantic integrity. The relationship with the data base system in general is outlined. An analysis of intradomain and interdomain comparability is included. An introduction to relevant implementation issues is also presented. Emphasis is placed on a general approach to implementation and implementation techniques, rather than on a specific system.

* This research was sponsored, in part, by the IBM Research Laboratory, San Jose CA.

Author's address: MIT Laboratory for Computer Science (formerly Project MAC), 545 Technology Square, Cambridge MA 02139.

1. INTRODUCTION

A data base is more than a collection of values; it should be a model of some application environment. The semantic integrity [9] of a data base is violated when the data base ceases to represent a legitimate configuration of that application environment. In the context of the relational data model, multiple levels of semantic integrity information have been proposed [9], to capture the properties of the environment being modelled.

1.1. The Relational Data Model

Although the ideas discussed in this paper are applicable to data base systems in general, the discussion herein is couched in terms of the relational model of data [4]. As concisely stated by Codd [7]: "In the relational approach there exists an interface at which the totality of formatted data in a data base can be viewed as a collection of nonhierarchic relations of assorted degrees on a given collection of simple domains (domains whose elements are not decomposable as far as the data base management system is concerned)."

For the purposes of this paper, a data base is defined to be a collection of normalized relations and a collection of domains. A domain is a set of atomic data values (objects). In particular, a domain is a subset of one of the two "natural" domains: real number and character string. A normalized relation may be viewed as a table, wherein each row of the table corresponds to a tuple of the relation, and the entries in a column belong to the set of values constituting the underlying domain of that column. (An entry is the value in some particular column for a given row of a relation.)

1.2. Semantic Integrity in the Relational Data Model

There are three levels of semantic integrity information in the context of the relational data model, which may be identified as follows:

1. Domain definition is the description of abstract sets of atomic data values.
2. Relation structure specification is the description of the fundamental structure of the base relations. This includes naming each constituent column of a relation, and stating the underlying domain of that column.
3. Relation constraints are used to define all additional properties of the relations. For example, primary key specification is accomplished by the appropriate use of relation constraints. However, relation constraints go far beyond merely supporting functional dependencies [5,6]; they provide the capability to define a rich variety of types of data properties.

The purpose of this paper is to consider the first aspect of semantic integrity, domain definition, in some detail. Specifically, the precise definition of domains, viewed as sets of atomic data values, is discussed. This

includes a review of the functional requirements for dealing with the problem of domain definition, a discussion of some of the work that has been done in the area, and an outline of a specific scheme. The proposed domain definition scheme may in fact be viewed as an extension of a current research relational data base system, such as the SEQUEL system [1,2].

It seems reasonable to assume that domain semantic integrity errors, i.e., errors which involve the presence of entries in some column of a relation which do not belong to the domain underlying that column, occur frequently enough to justify a facility to handle them. Specific experience with a particular data base application environment has shown that, for an experimental sample of user-data base interactions, a large percentage of errors discovered are domain semantic integrity errors [11].

1.3. Describing Sets of Atomic Data Values

Approaches to semantic integrity for relational data bases have been recently discussed [8,9,12,16]. Eswaran and Chamberlin have discussed the functional requirements of a semantic integrity subsystem, and have considered semantic integrity in the context of SEQUEL and "System R" [3,8]. Stonebraker has discussed semantic integrity in terms of the INGRES system and the language QUEL [12,13,14]. Zloof has considered the problem of semantic integrity with respect to the expression of semantic integrity specifications in Query by Example [15,16].

All of these approaches essentially deal with relation constraints, as defined above and discussed in [9]. That is, facilities are provided that allow the user to state predicates (expressed in SEQUEL, QUEL, or Query by Example) which are to hold on the data base. Although this aspect of semantic integrity is a crucial one, the first aspect of semantic integrity, domain definition, is not sufficiently supported in any of these systems.

The capability that is lacking is the ability to precisely define domains, as abstract sets of atomic data values. All of these systems allow the data type of each column of a relation (not each domain of the data base) to be defined, but the possible types are limited and very representation-oriented. What is needed is the ability to abstractly define domains. It should be possible, for example, to define domains like SOCIAL_SECURITY_NUMBER and GEO_COORDINATE, rather than being limited to such domains as INTEGER and CHARACTER_STRING. The work of Liskov and Zilles [10] concerning abstract data types is related to this notion, in that classes of abstract data objects (values) are being described.

Boyce and Chamberlin [1] have proposed attaching attributes to each column of a relation ("column descriptors"). One of these attributes is the scope of a column, which specifies the set of permissible values for entries in that column, e.g., salary is a positive integer less than 20000. Similarly, Zloof [16] has indicated that provisions should be made for facilitating the specification of entry "formats" ("their type, size, etc."). Unfortunately, no details are provided.

It is clear that a scheme is needed to facilitate the precise description of domains, and to integrate the domain definitions with the structure of the relational data base. Such a scheme should have at least the following properties:

1. facilitate the precise and detailed description of sets of atomic data values, as subsets of one of the natural domains real number and character string,
2. provide for the proper abstraction of defining domains independent of their use as underlying domains of columns in one or more relations,
3. force a domain definition to be a single logical entity, so that domain semantic integrity information is localized (modularity considerations),
4. encourage meaningful specifications of which atomic values (i.e., which columns) are comparable,
5. allow columns to have different associated units (such as feet or inches) so that the data base system may automatically convert from one units measure to another when comparing entries from these columns.

2. A DOMAIN DEFINITION LANGUAGE

A high level, nonprocedural language may be used to express domain definitions. In this language, each domain in a data base is described by a single domain definition. The definition of a domain is "installed" (bound) at the time the domain is created. Note that a domain definition specifies an underlying set of atomic values. Domains are not dynamic as are unary relations; rather, they constitute fixed abstract sets of data values. The definition of a domain may be modified, but this must imply that the abstraction has changed.

Three types of information are required by the semantic integrity subsystem to deal with domain semantic integrity [9]:

1. a specification of the set of atomic data values constituting the domain,
2. information describing when the domain definition is to be verified,
3. a specification of the action that is to occur if the domain definition is not satisfied.

Domain definitions are verified whenever an entry in some column of a relation is created or altered (e.g., by an insert or update row operation). Thus all that need be explicitly expressed in the statement of a domain definition is the precise description of the set of values comprising the domain, and the action that is to occur if an entry in some column of a relation is created or modified so that it does not belong to the underlying domain of that column.

Each domain definition therefore consists of the following four components, represented as clauses in the domain definition language:

1. Domain name

2. Description
The description clause allows the set of atomic data values constituting a domain to be specified. External to a domain definition, a domain is seen as a subset of one of the natural domains (character string and real number). However, within the domain definition, the set of data values constituting the domain is specified. This specification may be via enumeration of the domain values, description of the set of values by specifying the subunits of which they are composed, the placement of restrictions on the set of values by stating predicates that describe a subset of one of the natural domains, or a combination of the above. The special data value "null" (undefined) is present in each domain.

3. Ordering
The ordering clause is used to indicate how domain values are ordered with regard to comparisons with other values in the same domain. This information is important in identifying the semantic properties of a domain. One type of ordering specification is that the values in a domain inherit the (total) ordering of the natural domain of which the domain is a subset. This is a numeric ordering for real numbers and a lexicographic ordering for character strings. Another type of ordering specification is that no ordering exists, in which case only equality comparisons are meaningful. An external program may also be called, which accepts two domain values to be compared, and returns the value that is first in the ordering sequence.

4. Violation-action
The violation-action clause specifies the action that is to occur if an entry in some column of a relation is created or changed in such a way that the entry does not belong to the underlying domain of that column. Types of violation-action include:

a. the change may be refused and an error signalled,
b. a particular value, either constant or calculated from the erroneous value by means of operations (such as substring, concatenate, etc.) may be substituted as the new value of the entry,
c. a call may be made to an enternal program, the erroneous value being passed as the argument to the program, and the program returning the new value of the entry.

System-generated or user-specified messages may be optionally returned to the user or calling program. Note that in cases b and c, it is necessary to reverify the domain definition after the corrected value of the entry has been determined.

The definition of domains is integrated with the specification of the fundamental structure of the data base relations in the following way; each column of a relation has the associated attributes listed below:

1. the name of the column,
2. the name of the underlying domain,
3. a units specification (which may be null, meaning no units),
4. an indicator specifying whether or not "null" (undefined) values may be present in the column ("null" values may be disallowed from any column),
5. a narrative description of the column (optional).

Domains are defined, then, independent of relations. When a relation is created, the above column attributes are specified for each column of the new relation; this constitutes the specification of the fundamental structure of the relation. Most important is the specification of the underlying domain of each column; this indicates the set of atomic data values from which entries in that column are to be selected. Note that units are defined as a property of a column, and that units conversions map atomic values into atomic values.

2.1. Language Details and Examples

Figure 1 contains an example data base. The name of each relation is listed therein, along with the name of each constituent column. The underlying domain of each column is listed below the name of the corresponding column. Relation EMP contains information on the employees of some company, SALES records information on supplies of items for the company, ORDERS records order information, and BUDGET contains the salary budget for each department in the company. Units information could also be specified (e.g., Cost in SALES is in units "dollars" while Salary_budget in BUDGET is in units "thousands_of_dollars"), but is omitted here for simplicity.

Figure 2 contains domain definitions for some of the example data base domains. An indentation-oriented syntax is used in this figure. Examples of values in each domain are listed (in parentheses) to the right of the corresponding domain definition.

Figure 3 contains a specification of the syntax of the domain definition language. In figure 3, syntactic classes are denoted by lower case strings, while keywords are in upper case; actually, the language handles lower case keywords. Optional parts are enclosed in "[]", and alternatives are separated by "|".

In figure 2, the description clause of the NAME domain definition specifies that it consists of (character) strings, each of which is composed of a string followed by a ", ", followed by another string. In this description clause, data values are decomposed into subunits; the first and third are variable subunits, while the second is constant. Subunits may be labeled, so that they may be referenced elsewhere in the domain definition. As stated above, external to a domain definition, the data values constituting a domain are either atomic numbers or atomic strings. The rule is, if a description clause of a domain contains only number subunits (variable or constant), then the values in that domain are numbers, otherwise they are strings. Number and string subunits may be mixed, and if so, number subunits are converted to string form to yield the string values constituting the domain.

The description clause of the domain SEX indicates that it consists of two data values: "female" and "male" (in addition to the ever-present "null"). The domain MONEY is composed of strings which are a "$" followed by a number (to be converted into a string to obtain the domain value). The number subunit labeled "value" must be greater than or equal to zero, as specified by the subunit where restriction.

A subunit where restriction contains a predicate that is to be true for the subunit and involves only that subunit. It is thereby possible to express properties of number subunits involving comparators (such as "=" and ">") and number constants. It is also possible to state that a number is an exponential (exponential notation) or an integer (as for domain DATE). For string subunits, properties may be expressed such as a size (length) specification, a restriction of the characters present in a string (as for domain ITEM), and a lexicographic ordering comparison (such as "=" or ">") with constants.

A global where restriction permits expression of properties involving multiple subunits, as well as those on domain values as a whole (i.e., viewed as atomic values). A global where restriction contains a predicate that may involve a domain value, subunit values, operations, and comparators. String operations are provided which generate substrings, calculate lengths, perform concatenations, etc. Number operations include the usual arithmetic operations and "maximum" and "minimum". For example, in the description of domain MONEY, the global where restriction states that domain values (viewed as strings) must either have two digits to the right of the

decimal point or else no decimal point is present. Here, "right(*, '.'+1)" evaluates to the right substring of the domain value (which is referenced by "*"), starting at the character after the rightmost ".". The operation "present" yields "true" if the first string specified contains an occurrence of each of the following strings, otherwise it yields "false". The global where restriction of domain ITEM illustrates the specification of the number of times some contiguous group of subunits can repeat.

A where restriction may also contain a call of an external boolean program (as for domain ITEM). If this program call is in a global where restriction, the program is invoked with the domain value in question as its argument; the program returns "true" if the value is present in the domain, otherwise it returns "false". If the program call is in a subunit where restriction, the program is invoked with the subunit value in question as its argument; the program returns "true" if the subunit value is legal, otherwise it returns "false".

Boolean combinations of the above are allowed in both subunit and global where restrictions, as are conditionals (as for domain DATE). In addition, an "or" may be used to indicate that the domain contains values that come in more than one form.

The second clause in a domain definition is the ordering clause. This may specify that no ordering exists on values in the domain ("none") and that only equality comparisons are allowed (as for domain SEX). An ordering specification of "atomic" means that values in the domain are ordered by the usual numeric or lexicographic ordering, viewing the domain values as atomic numbers or strings (as for domain QUAN). The ordering clause may also contain an ordered list of labels (subunit names), indicating that domain values are ordered according to the values of the specified subunits. The usual numeric or lexicographic ordering on these subunits is used, and the subunits are taken in sequence: primary ordering, secondary ordering, etc. (as for domains NAME, MONEY, and DATE). Finally, an external program may be called, which is passed the two objects being compared, and which returns the first of the two in the ordering sequence (as for domain ITEM).

The third clause in a domain definition is the violation-action clause. As discussed above, it may specify that an error is to be signalled, indicating that the data base change specified by a user is incorrect and should be rejected. A system-generated or user-specified message may be optionally returned to the user or calling program, as is the case for all types of violation-action. If the violation-action is specified as "error", then an error is signalled and a system-generated message is returned (as for domains NAME and DATE). Domain SEX has a violation-action clause that specifies error signalling with a user-specified error message. The "substitute" violation-action allows a constant value to be substituted as the new value of the entry being created or changed (as for domain MONEY). A calculated value, obtained via string or number operations, can also be substituted (as for domain ITEM). In the specification of this calculation, "*" represents the value that is being checked to determine if it is in the domain. The calculated value is then checked to make sure that it is in fact a valid domain value; if not, then an error is signalled (to avoid infinite recursion). The definition of domain QUAN offers an example of an external program call violation-action.

3. IMPLEMENTATION CONSIDERATIONS

The domain definition language processor translates domain definitions into an internal form useful in semantic integrity verification. The semantic integrity subsystem has the responsibility of determining what verification is to be done whenever a data base change request is issued by a user. It must also assume the responsibility of performing the necessary verification. Whenever a new entry is created in a column (e.g., by an insert row operation) or an existing entry in some row is changed (e.g., by an update row operation), the system must make sure that this new entry belongs to the underlying domain of the column in which it occurs. The information in the description clause of the underlying domain of the column is used for this purpose. If the domain description is violated, the information in the violation-action clause is used. The ordering information is used when comparing two values in the same domain, as discussed in the next section.

A domain definition may be used to obtain the information necessary to construct several internal relations, which are used by the semantic integrity subsystem to facilitate domain definition verification:

1. The domain definition relation has the following columns (with primary key domain name):

   a. domain name,
   b. description type, which is "simple" if the domain has one nonlabeled subunit with no where restriction, otherwise "complex",
   c. global where restriction (in an abbreviated internal form),
   d. violation-action type, which is "error", "substitute", or "call",
   e. violation-action modifier, which if d is "substitute" is the value (constant or calculated) to be substituted, for "call" is the name of the external program to be called, otherwise "null",
   f. error/warning message, which is either a constant (user-specified message), "system-generated", or "null",
   g. ordering type, which is "atomic", "none", "subunit" (for subunit specified ordering), or "call" (for external program call ordering),
   h. ordering program name, which is the name of the external ordering program if g is "call", otherwise "null".

2. The subunit definition relation has the following columns (with primary key domain name, subunit index):

   a. domain name,
   b. subunit index, which is the ordinal number of the subunit in the domain definition,
   c. subunit type, which is either "constant" or "variable",
   d. label, which for constant subunits is "null",
   e. variable subunit class, which is "number", "string", or "oneof", and "null" for constant subunits,
   f. subunit where restriction, "null" if none exists,
   g. ordering index, which is the ordinal number of the subunit in the ordering clause, and "null" if this subunit is not referenced in the ordering clause.

3. The oneof constant relation has the following columns (with primary key all columns in the relation):

   a. domain name,

b. subunit index,
c. oneof constant, which is a constant in the "oneof" list for the subunit identified in b and for the domain named in a.

The above information is used by the semantic integrity subsystem to verify the domain semantic integrity of the data base. The details of this mechanism are not presented here.

Domain definitions may be utilized to automatically determine the appropriate physical storage type to be used to represent values in a domain. For strings, a fixed length character string representation is used when possible, such as when domain values are enumerated (via "oneof"), or when an upper bound is placed on the length of string values in the domain. In other cases, varying length character strings are used.

For numbers, it is necessary in many cases to make a compromise for efficiency. Integers ("number where integer") may be represented by a fixed binary storage scheme (e.g., single word binary), but it must be clear that this is only an approximation to the domain definition. Perhaps warnings are appropriate for integers that are too large. A similar situation exists for numbers in general: a float binary representation may be used for storage.

4. COMPARABILITY

The term comparability is used herein to refer to the general problem of determining when two or more atomic data values may be compared or otherwise manipulated. Three types of comparability may be distinguished:

1. equality-type, in which:
   a. values are compared for equality ("=") or inequality ("="),
   b. numbers are added ("+") or subtracted ("-"),
   c. sets of numbers are manipulated via set operations, such as "maximum" and "minimum",
   d. sets of values are manipulated by "union", "intersection", or "difference",
2. ordering-type, in which values are compared via "<", "<=", ">", or ">=",
3. mixing-type, in which values are manipulated via multiplication ("*"), division ("/"), exponentiation ("**"), or any string operation or user-defined operation (if they are allowed).

Equality-type comparisons are always allowed if the two values being compared (or manipulated) are from the same domain, i.e., if the values are from the same column or from columns with the same underlying domain. If the values are not from the same domain, then they may be compared if and only if the two columns (one value being from each column) have defined (nonnull) units, and a units conversion is defined between these two columns.

Recall that units may be specified for each column in a relation. Units conversions are defined, and automatically employed as appropriate. Another type of conversion could be supported, allowing conversions from one domain (not column) to another domain (e.g., a conversion from DATE to JULIAN_DATE). Then, entries in any columns having underlying domains DATE and JULIAN_DATE (respectively) could be compared via the conversion. This latter type of conversion is considered potentially useful, but somewhat redundant; it is not further considered here.

Ordering-type comparisons are allowed if two values are from the same underlying domain and the ordering of that domain is not "none". The ordering information in the domain definition is used to determine how the values are to be compared. Ordering-type comparisons are also allowed if the two values are from different domains but both columns (one value from each column) have defined (nonnull) units specifications, and a units conversion is defined between these units. In this case, the values are compared as atomic values, using the units conversion. In any other case, ordering-type comparisons are not allowed.

Mixing-type comparisons are always allowed. Presumably, values may be multiplied, divided, and exponentiated with no limitations, except of course for the requirement that the values be numbers. Comparisons are performed treating the values as atomic. Note that if user-defined operations are allowed, they may be placed in this category.

If the user wishes to state an unusual type of query, such as asking for all employees whose name is the same as the name of their department, this may be handled by allowing a comparison to be "forced". Entries in the two columns are then compared using the default numeric or lexicographic ordering, treating the values as atomic numbers or strings (respectively).

5. REMARKS AND DIRECTIONS

In this paper, an approach to domain definition has been discussed, in the context of the relational data model. Some issues to be considered in future papers include the following:

1. It is possible to extend the domain definition language so that previously defined domains may be used as subunits in the definition of a new domain. If this hierarchic approach is used, care must be taken by the system to retain domain definitions until they are no longer referenced in any other domain definition.
2. It may be useful to introduce "domain operations". In this approach, operations are defined for each domain, and manipulation of values in the domain is restricted to the specified operations. This approach is similar to the notion of abstract data types of Liskov and Zilles [10]. It may also be argued that the approach taken in this paper is still too representation-oriented. For example, values in the domain MONEY may be strings or numbers, but this is irrelevant with respect to abstraction. The important properties of the values constituting a domain may be best characterized by specifying the operations that are defined on the values in the domain.
3. A hierarchic approach to the naming of column groups may be adopted, wherein a group of columns may be named and referenced as a unit. Additionally, different column groupings could be used in different data base "views" (e.g., for providing different "formats").
4. It may be advantageous, in some cases, to defer the verification of domain definitions. For example, in the case where a data base is being "bulk loaded" or updates are being "batched", it may be desirable to report all violations of domain definitions at a later time, say to an interactive user or as part of a summary report.
5. The modifiability of domain definitions is very important. It should be possible for the definition of a domain to be changed as the corresponding abstraction changes. If so, then the problem is to verify that all entries in columns having a given

underlying domain satisfy the new definition of that domain.

6. It is possible to call an external program to verify that a value in question belongs to a domain. An external program call may also be used in the ordering and violation-action specifications. The problem is that nothing has been done to guarantee that the external program is correct. However, some reliability is guaranteed by the fact that this external program must use the normal data base system interface. In addition, the domain definition is again verified after the external program has terminated.

ACKNOWLEDGEMENTS

The author would like to thank D. D. Chamberlin of the IBM Research Laboratory, San Jose CA and M. M. Hammer of the MIT Laboratory for Computer Science (formerly Project MAC), Cambridge MA for their enthusiastic support of this work. D. D. Chamberlin, E. F. Codd, K. P. Eswaran, and J. N. Gray of the IBM San Jose Research Laboratory and M. M. Hammer of the MIT Laboratory for Computer Science have made many helpful suggestions and contributions, both with regard to the technical content and the prose.

REFERENCES

1. Boyce, R. F. and D. D. Chamberlin, Using a Structured English Query Language as a Data Definition Facility, IBM Research Report RJ1318, San Jose CA, 10 December 1973.

2. Chamberlin, D. D. and R. F. Boyce, "SEQUEL: A Structured English Query Language", Proceedings of ACM-SIGFIDET Workshop on Data Description, Access, and Control, Ann Arbor MI, 1-3 May 1974.

3. Chamberlin, D. D., J. N. Gray, and I. L. Traiger, "Views, Authorization, and Locking in a Relational Data Base System", Proceedings of National Computer Conference, Anaheim CA, 19-22 May 1975.

4. Codd, E. F., "A Relational Model for Large Shared Data Banks", Communications of the ACM, Volume 13, Number 6, June 1970.

5. Codd, E. F., "Further Normalization of the Data Base Relational Model", Courant Computer Science Symposia 6, New York NY, 24-25 May 1971, in Data Base Systems, Prentice Hall, 1971.

6. Codd, E. F., "Normalized Data Base Structure: A Brief Tutorial", Proceedings of ACM-SIGFIDET Workshop on Data Description, Access, and Control, San Diego CA, 1971.

7. Codd, E. F., "Recent Investigations in Relational Data Base Systems", Information Processing '74, North Holland, 1974.

8. Eswaran, K. P. and D. D. Chamberlin, "Functional Specifications of a Subsystem for Data Base Integrity", Proceedings of International Conference on Very Large Data Bases, Framingham MA, 22-24 September 1975.

9. Hammer, M. M. and D. J. McLeod, "Semantic Integrity in a Relational Data Base System", Proceedings of International Conference on Very Large Data Bases, Framingham MA, 22-24 September 1975.

10. Liskov, B. H. and S. Zilles, "Programming with Abstract Data Types", Proceedings of a Symposium on Very High Level Languages, Santa Monica CA, March 1974.

11. McLeod, D. J. and M. J. Meldman, "RISS: A Generalized Minicomputer Relational Data Base Management System", Proceedings of National Computer Conference, Anaheim CA, 19-22 May 1975.

12. Stonebraker, M. R., High Level Integrity Assurance in Relational Data Base Management Systems, Electronics Research Laboratory Report ERL-M473, University of California, Berkeley CA, 16 August 1974.

13. Stonebraker, M. R. and E. Wong, "INGRES: A Relational Data Base System", Proceedings of National Computer Conference, Anaheim CA, 19-22 May 1975.

14. Stonebraker, M. R., "Implementation of Integrity Constraints and Views by Query Modification", Proceedings of ACM-SIGMOD International Conference on the Management of Data, San Jose CA, 14-16 May 1975.

15. Zloof, M. M., "Query by Example", Proceedings of National Computer Conference, Anaheim CA, 19-22 May 1975.

16. Zloof, M. M., "Query by Example: The Invocation and Definition of Tables and Forms", Proceedings of International Conference on Very Large Data Bases, Framingham MA, 22-24 September 1975.

Figure 1. Example Data Base

Domains:

NAME	QUAN
SEX	ORDER_NUM
MONEY	CUST
DEPT	DATE
ITEM	

Relations:

EMP (Name, Sex, Salary, Manager, Department)
NAME SEX MONEY NAME DEPT

SALES (Item, Department, Quantity_on_hand, Cost)
ITEM DEPT QUAN MONEY

ORDERS (Order_number, Customer, Item, Date_shipped)
ORDER_NUM CUST ITEM DATE

BUDGET (Department, Salary_budget)
DEPT MONEY

Figure 2. Selected Example Data Base Domain Definitions

```
domain NAME                                        ("Smith, John")
     description
          last: string
          ', '
          first: string
     ordering
          last, first
     violation-action
          error

domain SEX                                         ("female")
     description
          oneof 'female', 'male'
     ordering
          none
     violation-action
          error 'sex must be female or male'

domain MONEY                                       ("$100")
     description
          '$'
          value: number where >=0
          where length(right(*, '.'+1))=2
               or not present *, '.'
     ordering
          value
     violation-action
          substitute null 'value in error, null has been assumed'

domain ITEM                                        ("AB-75-326")
     description
          string where not has numerics, '-'
          i1: '-'
          i2: string where not has alphabetics, '-'
          where repititions i1 through i2 >=1 and <=3
     or
          string where call check_item
     ordering
          call compare_item
     violation-action
          substitute left(*, 5)


domain QUAN                                        (17)
     description
          value: number where integer
                    and >=0
     ordering
          atomic
     violation-action
          call fixup_quan

domain DATE                                        ("1/20/76")
     description
          month: number where >=1 and <=12
          '/'
          day: number where integer and >=1 and <=31
          '/197'
          year: number where integer and >=5 and <=9
          where (if (month=4 or =5 or =9 or =11) then day<=30)
               and (if month=2 then day<=29)
               and (if (month=2 and year^=6) then day<=28)
     ordering
          year, month, day
     violation-action
          error
```

Figure 3. Syntax of the Domain Definition Language

```
domain-definition ::= DOMAIN domain-name
                          DESCRIPTION
                               description-clause
                          [ORDERING
                               ordering-clause]
                          [VIOLATION-ACTION
                               violation-action-clause]

domain-name ::= string-constant

description-clause ::= description-subclause
                     | description-clause
                       OR
                       description-subclause

description-subclause ::= description
                                [where-restriction]

description ::= [label:] subunit
              | description
                [label:] subunit

label ::= string-constant

subunit ::= STRING [WHERE string-boolean]
          | NUMBER [WHERE number-boolean]
          | ONEOF string-constant-list
          | ONEOF number-constant-list

string-constant-list ::= string-constant-component
                       | string-constant-list, string-constant-component

string-constant-component ::= string-constant
                            | ALPHABETICS
                            | NUMERICS
                            | SPECIALS

number-constant-l.   .::= number-constant
                         | number-constant-list, number-constant

string-boolean ::= string-boolean-term
                 | string boolean OR string-boolean-term

string-boolean-term ::= string-boolean-factor
                      | string-boolean-term AND string-boolean-factor

string-boolean-factor ::= string-boolean-primary
                        | NOT string-boolean-primary

string-boolean-primary ::= string-predicate
                         | (string-boolean)

string-predicate ::= comparator string-constant
                   | IF string-predicate THEN string-predicate
                                         [ELSE string-predicate]
                   | SIZE comparator number-expression
                   | HAS string-constant-list
                   | CALL program

comparator ::= = | ^= | > | >= | < | <=

number-boolean ::= number-boolean-term
                 | number-boolean OR number-boolean-term

number-boolean-term ::= number-boolean-factor
                      | number-boolean-term AND number-boolean-factor

number-boolean-factor ::= number-boolean-primary
                        | NOT number-boolean-primary
```

Figure 3. (continued)

```
number-boolean-primary ::= number-predicate
                         | (number-boolean)

number-predicate ::= comparator number-constant
                   | IF number-predicate THEN number-predicate
                                          [ELSE number-predicate]
                   | INTEGER
                   | EXPONENTIAL
                   | CALL program

where-restriction ::= boolean

boolean ::= boolean-term
          | boolean OR boolean-term

boolean-term ::= boolean-factor
               | boolean-term AND boolean-factor

boolean-factor ::= boolean-primary
                 | NOT boolean-primary

boolean-primary ::= predicate
                  | (boolean)

predicate ::= expression comparator expression
            | IF predicate THEN predicate
                          [ELSE predicate]
            | PRESENT expression, string-constant-list
            | CALL program

expression ::= [addition-operator] unsigned-expression

unsigned-expression ::= arithmetic-term
                      | unsigned-expression addition-operator arithmetic-term

arithmetic-term ::= arithmetic-factor
                  | arithmetic-term multiply-operator arithmetic-factor

arithmetic-factor ::= subexpression
                    | (expression)

subexpression ::= atomic-expression
                | set-function(expression-list)
                | APPEND(expression, expression)
                | SUBSTRING(expression, expression, expression)
                | LEFT(expression, expression)
                | RIGHT(expression, expression)
                | LOCATION(expression, expression)
                | LENGTH(expression)
                | REPITITIONS label THROUGH label

atomic-expression ::= label
                    | string-constant
                    | number-constant
                    | *

expression-list ::= expression
                  | expression-list, expression

set-function ::= MAXIMUM | MAX | MINIMUM | MIN | string-constant

addition-operator ::= + | -

multiply-operator ::= * | / | **

ordering-clause ::= ordering-list
                  | NONE
                  | ATOMIC
                  | CALL program
```

Figure 3. (continued)

```
ordering-list ::= label
                | ordering-list, label

violation-action-clause ::= violation-action
                          | violation-action-clause
                            violation-action

violation-action ::= ERROR
                   | ERROR message
                   | SUBSTITUTE expression
                   | SUBSTITUTE expression message
                   | CALL program
                   | CALL program message

message ::= string-constant

program ::= string-constant
```

Notes:

The nonterminals string-constant and number-constant are not further defined.

ALPHABETICS refers to the characters "A" through "Z" and "a" through "z", NUMERICS refers to the digits 0 through 9, and SPECIALS refers to all other characters.

SIZE returns the length of a string subunit. HAS s1, ..., sn returns "true" if a subunit has an occurrence of each of the strings s1, ..., sn (otherwise "false"). SIZE and HAS appear only in subunit where restrictions.

SUBSTRING(s,i1,i2) returns the substring of string s starting at character i1 and extending i2 characters. LEFT(s,i) and RIGHT(s,i) return the left and right substring (respectively) of s having length i. SUBSTRING, LEFT, and RIGHT may also be invoked with a second argument which is a string. This means that the substring is to start at the leftmost or rightmost occurrence of the second string argument, e.g., "LEFT(*, '.')" and "LEFT(*, INDEX(*, '.'))" are equivalent. LENGTH(s) returns the length of string s. APPEND(s1,s2) concatenates s1 and s2. LOCATION(s1,s2) returns the index of the first occurrence of s2 in s1 (or 0 if s2 is not a substring of s1). REPETITIONS s1 THROUGH s2 returns the number of repetitions (of the domain value) for subunits labeled s1 through s2.