THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

AUTOMATICALLY GENERATED UPDATABLE FORM SCREENS AS A DATABASE INTERFACE LANGUAGE

David Volk Beard and Toby J. Teorey

CRL-TR-13-84

FEBRUARY 1984

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

[1]Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

UMR0285 ~~enm~~

TABLE OF CONTENTS

FIGURE

Automatically Generated Updatable Form Screens as a Database Interface Language

David Volk Beard
Toby J. Teorey

February 1984

Computing Research Laboratory
Dept. of Elec. and Comp. Eng.
The University of Michigan
Ann Arbor, MI., 48109

Abstract

Current general purpose database interface languages, while being acceptable for computer professionals, are difficult for many other users to learn and use. They require understanding the structure of the database, as well as comprehending the use of variables. Such interface languages are especially difficult to use when formulating multi-relational queries. The need for such queries has become more pronounced in the last several years due to the trend toward third normal form and entity relationship databases.

A new database interface language called Metaform has been developed which automatically generates multirelational form screen interfaces for use by non-computer professionals. A form screen is a hierarchical subschema of the database schema, with a particular entity type at the apex of the hierarchy. This system - when given an entity type - will produce a corresponding form schema. Such a form schema displayed as a form screen on a CRT provides a user with a view of the database focused on a single entity type. A user can simultaneously query and update several relations in the database without having to know about its actual schema. Formal definitions of form schema and forms are given. Results from human factors experiments are shown to illustrate Metaform's user interface potential.

CR Categories and Subject Descriptors: H.1.2 [User/ Machine Systems], H.2.3 [Database Management Languages];Query Languages, H.4.1 [Office Automation]

1.1 Introduction

A rapidly growing way in which users interact with a database is through a form-screen application program which produces a multirelational form screen interface for the end user. This allows the user to have a view or perspective of the data which is easy to understand, and the user does not have to know about the actual database schema. It also means that the query and update languages used in conjunction with the form screen can be very simple and still be adequate for the user performing a task.

Unfortunately, such application programs are not easily created. A considerable amount of code is needed to implement them. Other difficulties include the usual amount of bugs, maintenance costs, and modification difficulties.

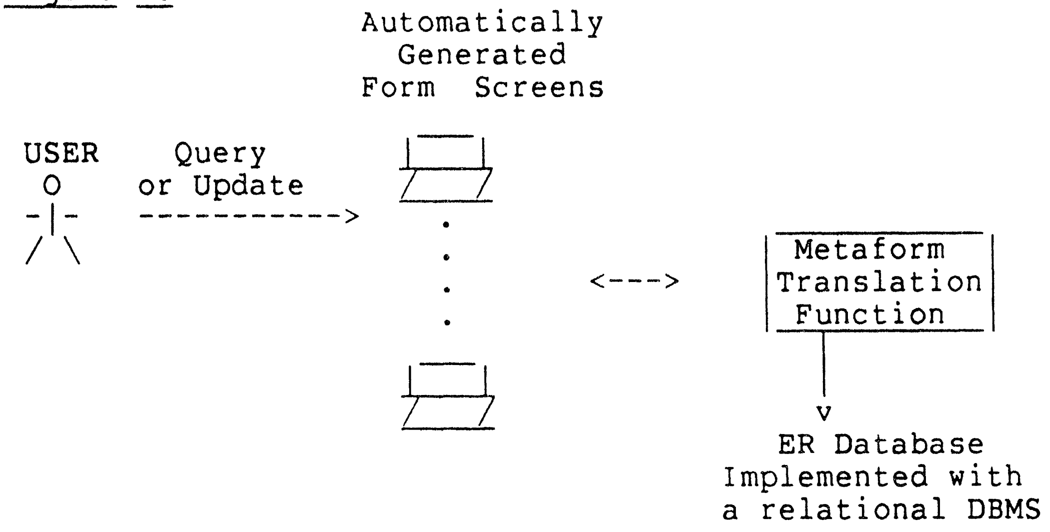
Despite these drawbacks, a great deal of application software is developed - at considerable expense - ignoring the availability of more powerful, more general-purpose vendor-supplied interface languages such as Sequel ([1]). The most likely reason is that such customized interface (systems) - even written as expensive applications programs - have been found to be more productive and cost efficient interfaces for many users.

Conventional general-purpose database-interface languages are unacceptable in many cases because they require the user to know about(and therefore be affected by changes to) the database schema. With such languages, a user needing multirelational interaction with a database has to think in terms of separate relations abstractly connected.

<u>Metaform</u>

A database interface system called Metaform has been developed which automatically generates multirelational form screen interfaces for use by non-computer professionals interacting with an entity-relationship database embedded in the relational model. This system - when given an entity type - will produce a corresponding form schema Such a form schema displayed as a form screen on a CRT provides a user with a view of the database focused on a single entity type. By using several operations such as "insert", "delete", "select", etc., a user doing a task related to that entity can interact with the information.

<u>Figure 1.</u>


Automatically Generated Form Screens
USER
Query or Update
Metaform Translation Function
ER Database Implemented with a relational DBMS

Most database interface languages are very concerned with the mathematical querying power of the interface. But the abstract power of a particular aspect of a language is of no value if the user cannot use such a feature. In the process of creating a powerful query language, the complexity of multirelational interaction greatly increases.

The production of an updatable multirelational view of the database has been a major goal of the Metaform system. The effect of an update operation on such a view must be clear to the user. By using forms as a data schema, we have forced the user to look at the data from the perspective of a single entity. Thus all the update operations, which are similar to verbs, have a subject. Ambiguity is reduced. Forms are also advantageous because they are so ubiquitous in every day life. Potential users have experience with forms and other hierarchical data. This aides their learning the system.

Besides being very widely used as a data design language, the entity relationship(ER) model ([2]) has an important property which is essential for producing multirelational updatable views: the ER model insists that the existence of an entity be separate from whether or not it is related to another entity. Thus, one may delete a relationship without having to delete an entity.

The ER model is not semantically equivalent to the relational model; there are ER database schemata which cannot be expressed as relation schemata, and vice versa. Therefore, despite some advantages of the ER model in this context, the relational model is formally developed and a significant number of relational database management systems(DBMS) currently exist. Embedding the semantics of the

ER model into the relational model allows us to use the advantages of both and we have done so.

Our objective in this paper is to present the Metaform system, concentrating on how the language design effects the user interface. Section 2 gives an overview of relevant database theory and interface languages. Definitions of the relational model and entity relationship model are then shown. Application form screen progress and practice is examined in section 3. Section 4 presents the Metaform language, an overview of its operation, and formal definitions of forms and form schemata. Details of how form screens are automatically generated, and how queries and updates are performed are then shown. An analysis of the language is presented in section 5, and the correctness of the update operators is examined. We then give an overview of the results of a human factors evaluation of Metaform. Learning times, error rates, and query times are given. Finally, we draw several conclusions as to the applicability of this language and indicate several areas of future work.

2.0 Formal Database Models and Interface Languages

2.1 The Relational Model

Relational databases ([3]) are known to offer certain advantages over other database management methods, including a simplified user interface, and a concise formal definition which we briefly review here. An illustrative example follows in section 2.3.

DEF A database schema is a 4-tuple dbs= <A, D, Dom, R > where:

1) A is a set of attributes.
2) D is the domain of the database.
3) Dom: A --> D so that $\mathrm{Dom}(A_i)$ is the domain of $A_i \in A$.
4) R is a finite set of relation schemata $R \subseteq A^+$. (The "+" is the Kleene plus, see [4]).

DEF A relation r_i on a relation schema $R_i = \langle A_0, \ldots, A_{n-1} \rangle$ is a finite set such that $r_i \subseteq \mathrm{Dom}(A_0) \times \ldots \times \mathrm{Dom}(A_{n-1})$. (The "X" is the cross-product set operation.)

DEF r is a set of relations called a database on dbs. There is exactly one relation in r on each of the relation schemata in R.

By using ordered sets or "strings" of attributes to denote relational schemata, we allow ourselves greater simplicity when developing query languages because no attribute renaming is needed. However, because of possible duplicates, to uniquely identify an attribute one technically has to use its location in a relation schema rather than its name. Since we do not have any duplicate attributes in our example relation schemata we ignore this.

Often it is necessary to imposes constraints on the information stored in the database. The primary method for this in the relational model is the functional dependency.

DEF Let r_i be a relation on schema R_i and let B_1 and B_2 be subsets of R_i. We say that $B_1 \dashrightarrow B_2$ (the set of attributes B_1 functionally determines the set of attributes B_2) IFF for all $t_1 \in r_i$ and $t_2 \in r_i$, $t_1(B_1) = t_2(B_1)$ implies that $t_1(B_2) = t_2(B_2)$.

DEF a subset of attributes X from R_i is said to be a key of R_i IFF X functionally determines R_i and no proper subset of X functionally determines R_i.

A detailed overview of the relational model with a somewhat different formalism may be found in [5]. In our notation, we use identical subscripts and superscripts to indicate the correspondence between an instance and a type.

2.2 Entity Relationship(ER) Model

While the relational model is the model of preference for formal analysis of an information system, many database designers use another approach which is more useful for the practical side of analysing a user's data needs.

The ER model divides information into three classes. Firstly, there are entities which are "things" in the real world such as cars, parts, or employees. Secondly, relationships are used to relate one or more entities. For example, if a database contains the entities "supplier" and "part", then a relationship might be "supplies". We only deal with binary relationships. A binary relationship has the property of being either one-to-one (1-1), one-to-many (1-n), many-to-one (m-1), or many-to-many (m-n). These are defined as follows:

a) 1-n: Two entity schemata are in 1-n relationship when each entity in the second schemata is related to at most one entity in the first entity schemata.
b) m-1: Two entity schemata are in m-1 relationship when each entity in the first schemata is related to at most one entity in the second.
c) 1-1: Two entity schemata are in 1-1 relationship when each entity in either schemata is related to at most one entity in the other.
d) m-n: This relationship imposes no constraints.

Finally, the ER model has attributes which are properties of either entities or relationships. These properties might include color, weight, or marital status.

In addition to the 1-1, 1-n, m-1, and m-n constraints imposed by the data model an additional constraint insists that a relationship exists only among existing entities.

The ER model may be thought of as a graph with the nodes denoting entity schemata, and the arcs representing relationships between the entity schemata. Since form schemata can be defined to be subgraphs of the ER graph, from the prospective of an entity schema, they are very similar, and operations on forms can be acceptably translated into ER operations.

Though the ER model may be superior for actually specifying real world data requirements, relational database management systems(DBMS) commonly exist for the implementation of information systems. For this reason, we have chosen to embed the ER model within the formalism of the relational model.

The relational model as originally defined by Codd does not recognize the concept of "entity". In [6], the model was expanded to encompass the ER model and a reader interested in a more detailed treatment of this subject might examine either that paper or [7] which contains the formal details of the Metaform system.

To embed the ER model in the relational model one may use relation schemata to represent entity schemata and relationship schemata. The model must then be expanded to allow a rigorous determination of which schemata are entity relation schemata. Relationship schemata are formed by concatenation of the keys of the two corresponding entity schemata plus possibly several other attributes.

A key is defined for each entity schema in the database schema. The key of the relationship schema is the concatenation of the keys of the two corresponding entity schemata. The relationship constraints are imposed using functional dependencies among the keys of the entity schemata in a relationship schema. A detailed analysis of the Entity Relationship model and its applications may be found in [8].

2.3 Example Database and Schema in ER Form

The following is an ER parts database and schema defined using the above relational formalism. It consists of parts which have a name, color, and number; suppliers which have a name, base city, and number; and projects which have a name, base city and number. These entity types are related by binary relationships.

This example follows the above definitions. A more conventional presentation using tables is given later in Figure 7.

The database schema PARTS = <A, D, Dom, R,> where:
Attributes A = {pnum, pname, color, snum, sname, psqty, scity, pjnum, pjcity, ppjqty }.

In naming the above attributes "p" stands for "part", "s" for "supplier", and "pj" for "project". Relationships formed between two entities are named by concatenating the entities symbols so that "ppj" stands for the "part-project" relationship and "ps" for "part-supplier".

```
The function Dom is defined as follows:
Dom(pnum)    = { 101, 102, 103, 104 }
Dom(pname)   = { nut, bolt, widget, screw }
Dom(color)   = { red, blue, green }
Dom(snum)    = { 9, 8, 7, 6 }
Dom(sname)   = { acme, wacky, wolly, best }
Dom(pjnum)  = { 1001, 1002, 1003, 1004 }
Dom(pjname) = { alpha, beta, gamma, delta }
Dom(scity)  =
  Dom(pjcity) = {  detroit, ann-arbor, new-york }
Dom(psqty)  =
 Dom(ppjqty) = whole numbers  ("qty" stands for quantity)
```

The domain of the database D = Dom(pnum) U Dom(pname) U Dom(color) U Dom(snum) U Dom(sname) U Dom(pjnum) U Dom(pjname) U Dom(scity) U Dom(pjcity) U Dom(psqty) U Dom(ppjqty).

The set of relational schemata R={ P,S,PS,PJ,PPJ } where:

```
 P   = <pnum, pname, color>  PJ  = <pjnum, pjname, pjcity>
 S   = <snum, sname, scity>  PPJ = <pnum, pjnum, ppjqty>
 PS  = <pnum, snum,  psqty>
```

P (parts), S (supplier), and Pj (project) are the entity schemata. Their keys respectively are <pnum>, <snum>, and <pjnum>. The relationship schema PPJ (parts-project) and PS (part-supplier) have the concatenated keys of the corresponding entities. These are respectively <pnum, pjnum> and <pnum, snum>.

PS is a m-1 relationship and PPJ is a m-n relationship.

A relational database called "parts" on the database schema PARTS is defined as follows. Let parts = {p, s, ps, pj, ppj} a set of relations where:

```
p = { <101, nut,    red>,  pj ={ <1001, alpha, detroit>,
      <102, widget, blue>,       <1002, beta,  detroit>,
      <103, bolt,   red>,        <1003, gamma, ann-arbor>}
      <104, screw   blue>,
      <105, nail,   green>}
```

```
s = { <9, acme,  detroit>,     ppj={ <101, 1001, 100>,
      <8, wacky, ann-arbor>,         <101, 1002, 111>,
      <7, best,  new-york> }         <102, 1001, 92>,
                                     <102, 1003, 132>,
                                     <103, 1002, 45>}

ps= { <101, 9, 10>,
      <102, 9, 12>,
      <103, 8, 12>,
      <104, 7, 23>,
      <105, 7, 14> }
```

2.4 Database Query languages

General purpose database interface languages have been developed in conjunction with relational database management systems in order to allow end users to be able to have access to the stored information. These languages are normally based on either the relational calculus or the equivalent relational algebra, where the user is required to know something about the structure of the database schema. We examine two relational-calculus-based query languages developed by researchers at IBM. A user interface study comparing these systems is discussed in [9].

Sequel ([1]) is a commercially-available line oriented tuple relational calculus (variables denote tuples) query language which is relationally complete. It was developed by IBM. It is very similar to the tuple relational calculus with English key words used to increase comprehension. For experienced database users with a good math background, this is a very efficient query language. Very complex queries may be formulated by an experienced user in a short amount of time.

While novice database users with weak math backgrounds can be taught to create simple queries using Sequel, a long learning time is needed. Additionally, complex queries seem to be beyond the scope of many users. Researchers recognized that several problems encountered with line-oriented query languages such as Sequel could not be solved without a drastic change in approach ([Zloof76]). In particular, users with a limited math background seemed to have trouble correctly using variables.

Query-by-example (QBE) ([Zloof76]) was developed in an attempt to allow a "visual editor" approach to database interface without having the understanding needed to use variables. Variables have been found to be a difficult concept for users to comprehend. To use the system, a template for a relation is displayed on a CRT screen. The user fills in blank spaces with information, constructing a query, and the DBMS then selects the tuples which correspond to the query. It is called Query-By-Example because instead of variables, as used in Sequel, the user gives a generic example of what information is desired.

However, QBE is really just a different and possibly more efficient method for formulating Sequel type queries. It is still a relational-calculus-based query language. The user still has to know about the structure of the database schema, while multirelational information must still be thought of as several relations abstractly connected.

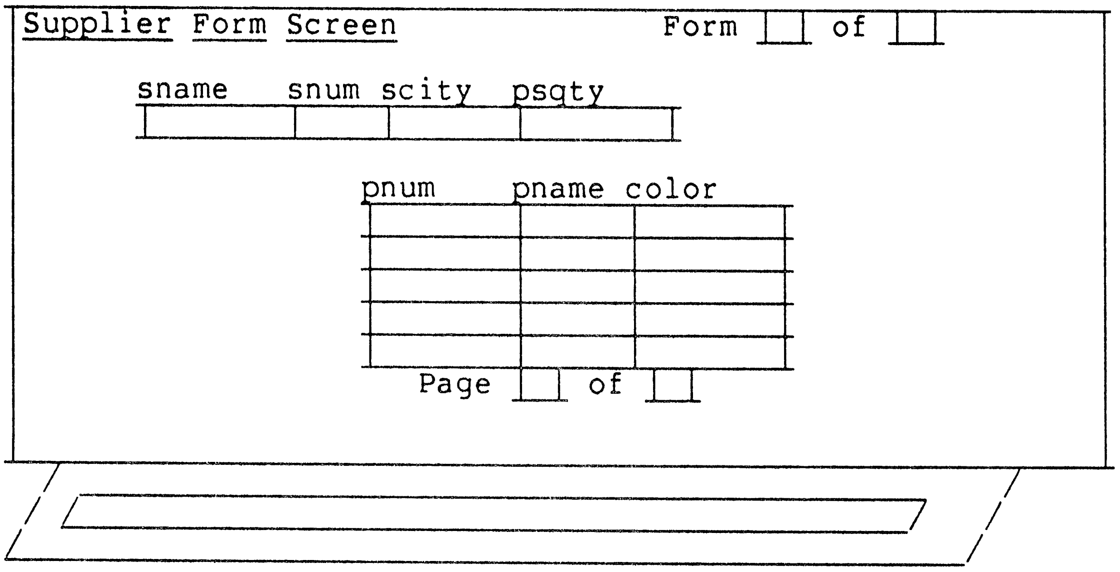
The trend in the last several years toward ER or third normal form relational databases has greatly increased the number of multirelational interactions of the typical user. Thus many database administrators (DBA) have had to develop other methods of interaction to databases.

3.0 Form Screens

3.1 Overview

Because of the interface problems with the general purpose language tools mentioned above, application programs with prescribed screens are often written. The purpose of these programs is to provide a special purpose form screen interface that is easy to learn and use. The user is shown a form screen on his terminal which displays all the information from the database needed to complete the task. The user does not need to know much about the database schema, and so is shielded from many changes to it.

Figure 2. An example "supplier" form screen.



3.2 Advantages of Forms

What aspects of forms make them so useful and prevalent in computer information systems? A possible reason is that they are so ubiquitous in non-computer situations that many first-time computer users are able to recognize and understand forms quickly from their non-computer experience. Another underlying reason might be due to the nature of human information interaction. The vast majority of questions seem to involve simple combinations of related information.

A form schema may be considered to be a subgraph of an ER diagram. This means that from the perspective of a particular entity type, the ER diagram and a form schema are semantically very similar.

Still another insight into the applicability of forms to human information interaction is available by comparing the form (and possibly the whole ER database) to an English sentence. Relationships are very similar to verbs. Entities correspond to nouns. Attributes can be either adverbs or adjectives. Sentences have a subject, and usually have only one verb and possibly several relative clauses; they typically deal with closely related information.

Querying the database is a very similar activity. The query has a subject, verb, and several relative clauses. In the vast majority of cases, only simple queries involving closely related information are ever composed, despite the theoretical possibilities of the relational calculus. Forms allow a view or "chunking" of the database information which is simple, has a subject, and combines closely related information.

3.3 Disadvantages of Conventional Form Screens

Clearly, from the prospective of a typical end user, specialized form screens are - in most cases - superior to general purpose interface tools, but they have several major flaws. First, they are very expensive to develop initially and even more expensive to maintain. Second, such programs are hard to test. Errors in specification, design, or implementation are common and costly. Third, application form screens are specialized interfaces which are inflexible and difficult to change. They take too much time to develop, so one cannot quickly develop a prototype application nor modify an existing application form screen.

3.4 Specifications for a New Interface Language

What is needed is a combination of the comprehensibility of form screens without having to create costly programs - an interface language which automatically produces form screens for human computer interaction.

The following are the set of goals we have attempted to meet in the creation of the Metaform system.

Language:

1) No programming activity needed to create an interface for a user.
2) Automatic adaptation to changes in the database schema.
3) Formal syntax and semantics.

Human Factors:

1) Very quickly learnable by a wide variety of users including secretaries, vocational trainees, systems analysts, etc.

2) Initial learning time to be able to perform very simple queries: computer professionals - less than 5 minutes, secretaries - less than 15 minutes.
3) The interface language should be layered ([9]). This means that a minimal subset of the language is taught first, with other aspects of the language added later in several stages or layers. Ideally, The "learning inertia" (unwillingness to learn something new) of a new layer is much less than the perceived benefits of the new layer.
3) Average learning time for the above group - less than 30 minutes, with 80% correctness.

4.0 Metaform

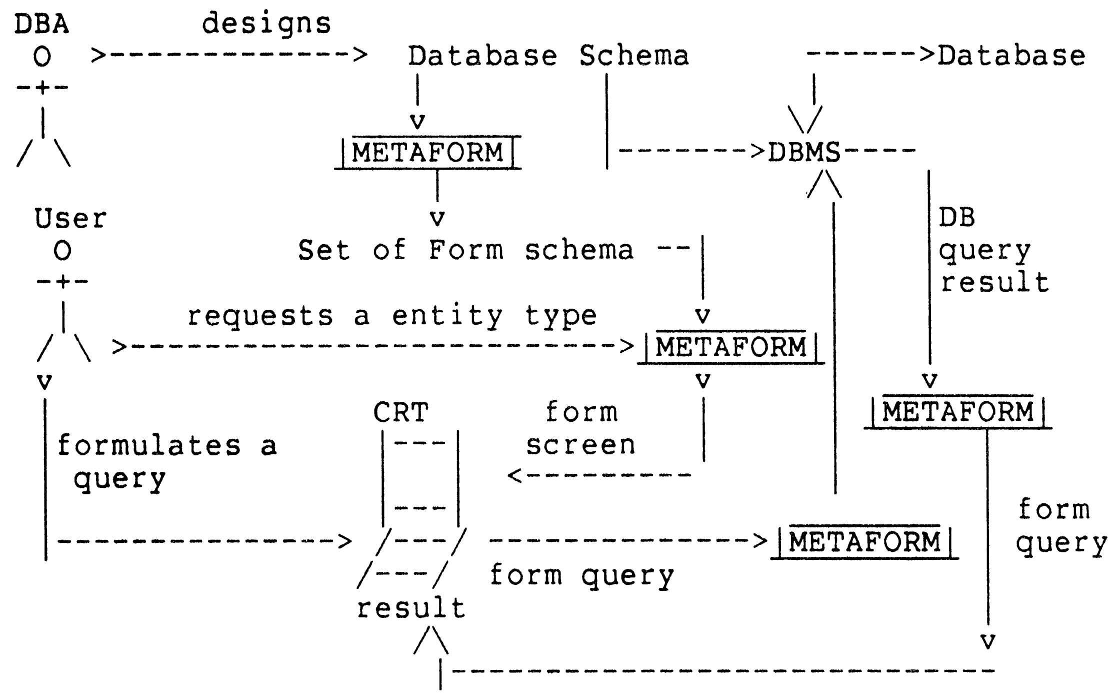
Metaform is a database interface language that provides updatable multirelational form screen interfaces to the system's end users. A simple but reasonably powerful query and update language is provided to be used in conjunction with the form screens. No specific knowledge of the database schema is needed.

Since Metaform is a general purpose tool, generating an individual user interface does not require any new computer code to be written. Therefore there is virtually no development cost for an individual form screen.

Metaform works in practice because the vast majority of user queries seem to be localized. Only a small part of the graph representing the ER schema is generally considered for a query. A reach-N subgraph of an ER schema graph consists of the entity type in question, and all the entity types which are each related to the entity type in question by at most N relationships. In most cases a query about an entity type can be contained in a reach-one or reach-two subgraph. This assessment is difficult to demonstrate empirically, but a human factors study ([11]) indicates that the complexity of a query (based on time) increases dramatically as the number of relations in a query increase. A great deal of time was needed for even experts to understand an English query involving more than four or five relations.

Figure 3 shows a flow diagram of the Metaform system. A database administrator (DBA) analyzes the real world situation and produces the database schema. Metaform then automatically generates a form screen (schema) for each entity type in the ER schema. These screens may be modified by the DBA in various ways if needed.

Figure 3. An Overview of Metaform.



A session may begin when the user requests a form screen which corresponds to an entity type. Metaform displays this on a CRT. A form query is formulated on the form screen by the user, and the system is requested to perform this operation. The form query is translated into a relational database query by Metaform and passed on to the DBMS. The DBMS processes this relational calculus operation using the data stored in the database, and obtains a relational query result. Metaform translates this relational query result into a set of forms, and displays the result for the user. Updates are processed in a similar manner.

4.1 Form and Form Schema Definitions

While forms and form schemata are very familiar to the information processing community, some readers will prefer the more rigorous description. The following are the definitions of a form and form schema. A more detailed presentation of this is given in [7].

Let dbs = $<A, D, Dom, R>$ be a database schema with database r. F is the set of form schemata defined as follows:

1) $F_0 \in A^+$, $F_0 = <A_0, \ldots, A_{n-1}>$ is a form schema i.e. $F_0 \in F$.
2) if $\{F_0, \ldots, F_{m-1}\} \subseteq F$ and $<A_0, \ldots, A_{n-1}> \in A^+$, then ,$<A_0, \ldots, A_{n-1}, F_0, \ldots, F_{m-1}> \in F$.
3) Nothing is in F except by 1 and 2.

Domain of $F_0 \in F$.

We extend the definition of the function Dom (from section 2.1) for $F_0 \in F$. Dom(F_0) is defined as follows:

1) if $F_0 = <A_0, \ldots, A_{n-1}> \in A^+$, then $\mathrm{Dom}(F_0) = \mathrm{Dom}(A_0) \times \ldots \times \mathrm{Dom}(A_{n-1})$.
2) if $\{F_0, \ldots, F_{m-1}\} \subseteq F$ and $<A_0, \ldots, A_{n-1}> \in A^+$, then $\mathrm{Dom}(<A_0, \ldots, A_{n-1}, F_0, \ldots, F_{m-1}>) = \mathrm{Dom}(A_0) \times \ldots \times \mathrm{Dom}(A_{n-1} \times \mathrm{Power}(\mathrm{Dom}(F_0)) \times \ldots \times \mathrm{Power}(\mathrm{Dom} (F_{m-1}))$ where 'Power' denotes the power set operation.

Forms

Let f be the set of forms on F. $f_0 \in f$ is a form on schema $F_0 \in F$ IFF $f_0 \in Dom(F_0)$ and f_0 is finite.

4.2 Automatically Generated Form Screens

A form schema is automatically produced for each entity type in the database schema. This schema includes the entity type and its attributes; all directly related (reach-one) entity types and their attributes; and the corresponding relationships and their attributes.

From the perspective of an entity in a database, it is related to either one other entity of an entity type(1-1 or M-1) in the database schema which we call a single relationship, or to many entities of an entity type(1-M or M-N) which we call a repeating relationship.

A form schema for an entity type in a database schema is an unnormalized (not in first normal form) relation schema. The base entity type (and its attributes) form the apex of the form schema. Other related entity types form repeating groups.

When an entity type is related to another by a repeating relationship, that other entity type (and the corresponding relationship) forms a repeating group in the form. When an entity type is singly related to another entity type, that entity type and the corresponding relationship become attributes of the base entity type in the form schema, and along with the base entity type they are placed in the apex of the form schema.

Using the parts database in section II and the form schemata definition given above, we can formulate an example part form schema. Since there can be several projects using a particular part, the project entity type and the attribute ppjqty from the relationship between part and project together form a repeating group. Because there is at most one supplier for a given part, the form schema allows only for only one supplier.

Figure 4. Example part form schema.

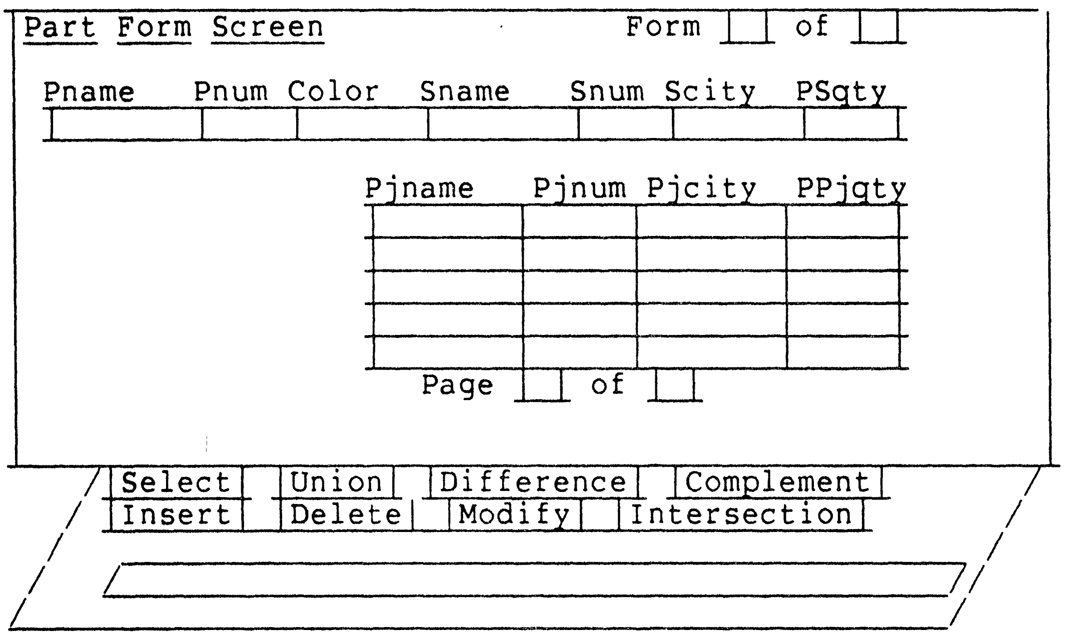
```
    p rel           s rel          ps rel
<pnum,pname,color,snum,sname,scity,pnum,snum,psqty
                       pj rel           ppj  rel
                 <pjnum,pjname,pjcity,pnum,pjnum,ppjqty>>
```

Because of the constraint that a relationship cannot exist unless the two corresponding entities exist, we find that the duplication of keys - from the entity types, and from the corresponding relationship schema - can be eliminated.

Forms which are generated in this manner are subgraphs of the actual entity relationship database schema. These are very simple subgraphs which - from the perspective of a particular entity type - behave the same as the ER database schema under updates.

We show the final part form screen as it would be displayed on a CRT.

Figure 5. The Part Form Screen.



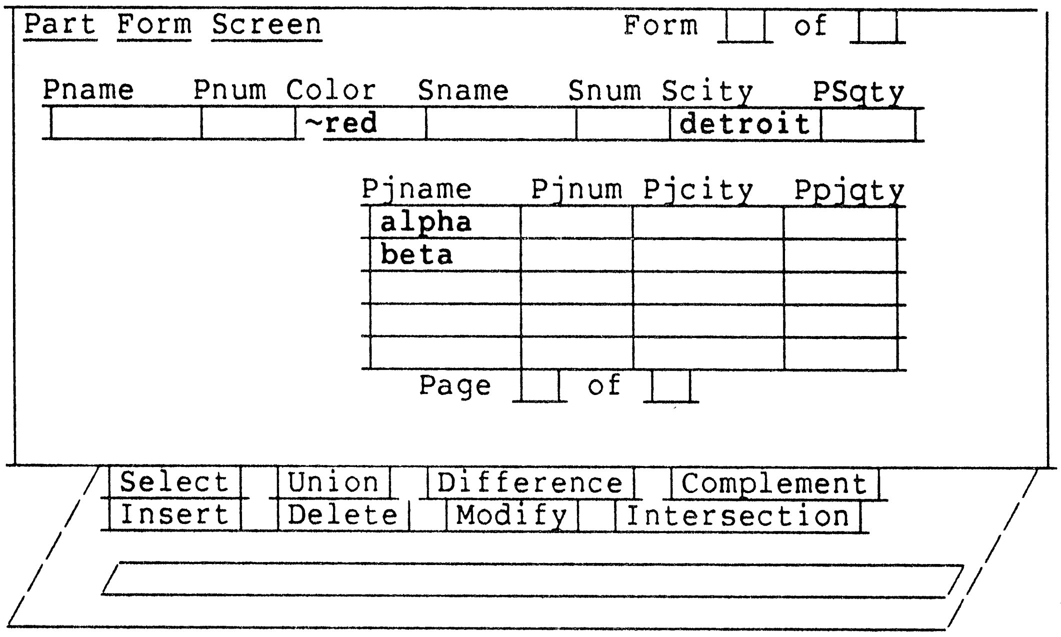
4.3 The Metaform Query Language

In Figure 6 the part form screen is displayed along with a form query which corresponds to the English sentence: "Display all the parts which are not 'red', are supplied by suppliers based in 'detroit', and are used by EITHER the 'alpha' OR the 'beta' projects".

The user moves the cursor around the form screen and fills in the appropriate boxes with constants. When the user presses the "select" function key, the Metaform system translates this form query into a relational-calculus query and the DBMS retrieves the result which is translated by Metaform into a set of forms which are displayed to the user.

Figure 7 shows the current database(in table form) and Figure 8 displays the result set of part forms which would be shown to the user after making this query.

Figure 6. An Example Form Query.



Part Form Screen Form of

Pname	Pnum	Color	Sname	Snum	Scity	PSqty
		~red			detroit	

Pjname	Pjnum	Pjcity	Ppjqty
alpha			
beta			

Page of

Figure 7. The Example Database in Table Form.

P<pnum, pname, color>

pnum	pname	color
101	nut	red
102	widget	blue
103	bolt	red
104	screw	blue
105	nail	green

S<snum, sname, scity>

snum	sname	scity
9	acme	detroit
8	wacky	ann-arbor
7	best	new-york

PS<pnum, snum, psqty>

pnum	snum	psqty
101	9	10
102	9	12
103	8	12
104	7	23
105	7	14

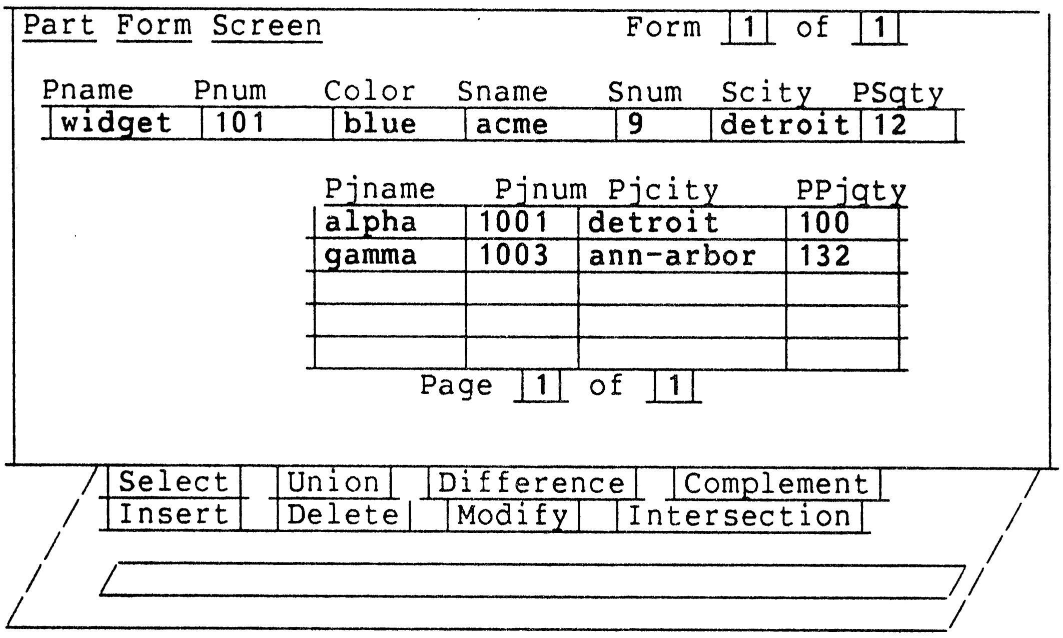
PJ<pjnum, pjname, pjcity>

pjnum	pjname	pjcity
1001	alpha	detroit
1002	beta	detroit
1003	gamma	ann-arbor

PPJ<pnum, pjnum, ppjqty>

pnum	pjnum	ppjqty
101	1001	100
101	1002	111
102	1001	92
102	1003	132
103	1002	45

Figure 8. The Part Form Screen Query Result.



Part Form Screen Form 1 of 1

Pname	Pnum	Color	Sname	Snum	Scity	PSqty
widget	101	blue	acme	9	detroit	12

Pjname	Pjnum	Pjcity	PPjqty
alpha	1001	detroit	100
gamma	1003	ann-arbor	132

Page 1 of 1

As an aid in teaching, the form query language is divided into four levels or layers ([9]).

1) The first level allows queries which involve one constant placed in one data field.
2) The second level allows queries which have several data fields filled in with constants. A constant in a field corresponds to a simple term in the relational calculus. An "and" condition occurs between different "terms". An "or" condition occurs between the rows ("and-ed" groups of terms) of a repeating group.
3) The third level uses inequalities such as "<", ">", "~=", etc. to produce terms.
4) The forth and final level allows several result sets of forms to be combined into a single result set

using set operators. A stack of previously obtained results is kept. The set operators are applied in a postfix manner to produce new result sets.

The above query language is not relationally complete, but it is very easy to learn and use, and seems to deal with the vast majority of user query situations.

4.4 The Metaform Insert Operator

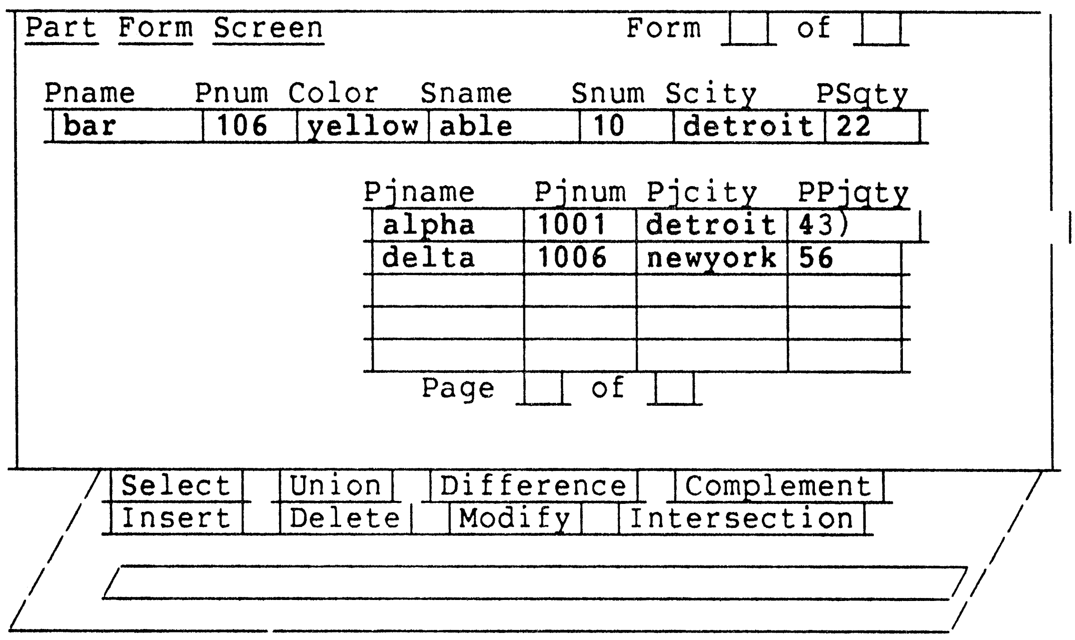
To insert a form into the database, the user fills in the fields of a form screen with constants and presses the "insert" function key. The key of any entity type displayed on the form screen must then be specified.

If the entity specified for the form already exists in the database, the insert request is rejected. Otherwise, we may assume that it does not exist and because of constraints on the database schema, no relationship contains that entity.

If any of the single or repeating related entities already exist in the database and conflict with those being proposed by the user, the insert request is rejected and the user is given an appropriate error message. Otherwise, the DBMS is requested to insert the apex entity, all the single or repeating entities, and all the relationships.

In Figure 9 we show an insert request, and in Figure 10 we show the database given in Figure 7 after the insert request is applied. A new part, supplier, and project has to be added. it is not necessary to add the alpha project since it already exists in the database.

Figure 9. An Example Form Insert.


Part Form Screen   Form of

Pname	Pnum	Color	Sname	Snum	Scity	PSqty
bar	106	yellow	able	10	detroit	22

Pjname	Pjnum	Pjcity	PPjqty
alpha	1001	detroit	43)
delta	1006	newyork	56

Page of

Figure10. Example Database after Insert.

P<pnum, pname, color>

pnum	pname	color
101	nut	red
102	widget	blue
103	bolt	red
104	screw	blue
105	nail	green
106	bar	yellow

S<snum, sname, scity>

snum	sname	scity
9	acme	detroit
8	wacky	ann-arbor
7	best	new-york
10	able	detroit

PS<pnum, snum, psqyt>

pnum	snum	psqyt
101	9	10
102	9	12
103	8	12
104	7	23
105	7	14
106	10	22

PJ<pjnum, pjname, pjcity>

pjnum	pjname	pjcity
1001	alpha	detroit
1002	beta	detroit
1003	gamma	ann-arbor
1006	delta	new-york

PPJ<pnum, pjnum, ppjqty>

pnum	pjnum	ppjqty
101	1001	100
101	1002	111
102	1001	92
102	1003	132
103	1002	45
106	1001	43
106	1006	56

4.5 The Metaform Delete Operator

Metaform deletes a form from a database by deleting the corresponding entity and deleting any relationship that deals with that entity. No other entities are deleted. Since the form and corresponding entity are uniquely identified by the key of the apex entity, only the attributes in the key need be specified.

It should be emphasized that an ER database is necessary for the delete operation to function in an acceptable manner. A user expects the deletion of an entity to delete its existence, and delete any relationship it might have had with another entity. No other entity can be deleted. A conventional relational database does not insist that the existence of an entity be separated from its relation with another entity.

Figure 11 shows an example delete request and Figure 12 shows the database in Figure 10 after the update has been performed. The delete request in Figure 11 may at first appear to be the inverse of the insert request in Figure 9, but the entities added in the insert example are not removed in the delete example.

Figure 11. An Example Form Delete.

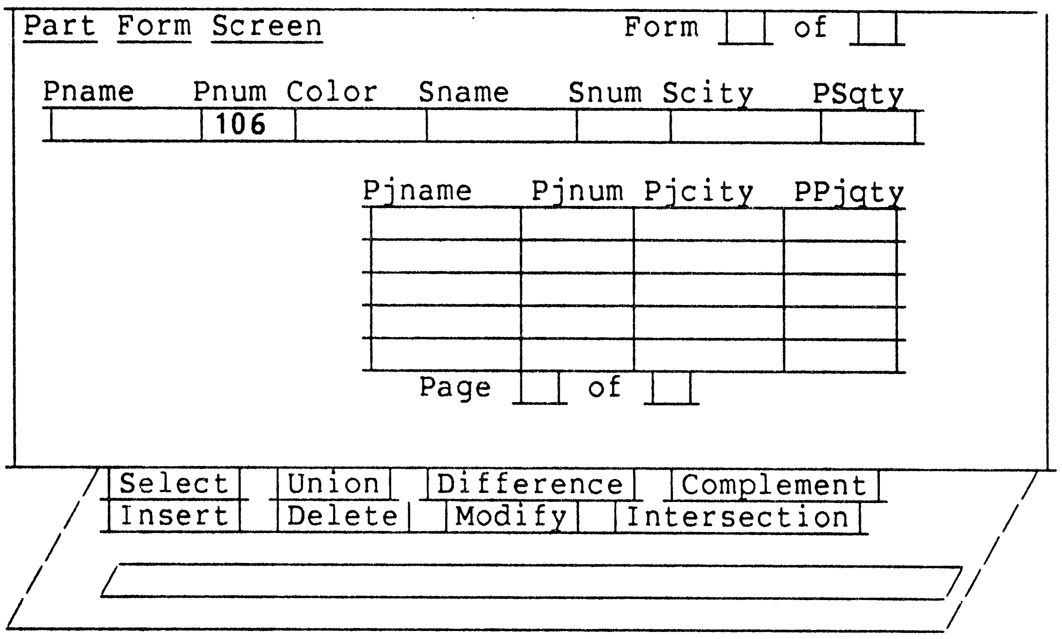


Figure 12 Example Database after Delete.

P<pnum, pname, color>

pnum	pname	color
101	nut	red
102	widget	blue
103	bolt	red
104	screw	blue
105	nail	green

S<snum, sname, scity>

snum	sname	scity
9	acme	detroit
8	wacky	ann-arbor
7	best	new-york
10	able	detroit

PS<pnum, snum, psqyt>

pnum	snum	psqyt
101	9	10
102	9	12
103	8	12
104	7	23
105	7	14

PJ<pjnum, pjname, pjcity>

pjnum	pjname	pjcity
1001	alpha	detroit
1002	beta	detroit
1003	gamma	ann-arbor
1006	delta	new-york

PPJ<pnum, pjnum, ppjqty>

pnum	pjnum	ppjqty
101	1001	100
101	1002	111
102	1001	92
102	1003	132
103	1002	45

4.6 The Metaform Modify Operation

As currently defined, the modify operation is a delete operation followed by a insert operation.

5.0 Analysis of Metaform

5.1 Correctness of Update Operators

When one tries to develop a new database interface language which allows a user to modify the information in a view which is not in the actual schema, there is always the possibility that this language allows the user to express something which cannot be expressed in the language of the actual database update operations. In short, the semantics of the new update language cannot be embedded into the original database update language.

In order to insure correctness of the Metaform update operators, we have to be able to unambiguously express both the Insert and the Delete operators in terms of sets of insertion and deletion operations on the actual database.

The definitions of the Metaform insert and delete operations given in section 4 provide a unique set of insert or delete relational database operations for each of the Metaform insert and delete operations.

There is another side to correctness, however, which has generally been ignored in the literature. A beginning database interface language user has preconceived notions as to the meanings of the various operations on the database view. It is essential that:

1) The user expects the form update operations to have exactly one meaning.

2) These operations have the user's expected meaning.

One advantage of using form screens is that such screens have a subject or focus, so that an operation such as "delete"(a verb) has the subject of the form as a object. "delete" becomes, in the user's mind, "delete a part".

Little or no human factors testing of database update operations has been recorded in the literature. Considering that a large part of database interaction involves updating the database, this is a potential area for future research.

5.2 Human Factors

A user-interface study was conducted to determine whether the Metaform language could be used by a wide variety of end users ([12]). The experiment involved teaching a group of subjects how to use the form-screen interface and then testing their knowledge. Times for learning and doing query tasks and the number of errors made in doing these tasks were collected. The query tasks involved translating queries into corresponding English sentences and translating English sentences into Metaform queries.

In order to avoid a biased database, we used one from [13]. It was translated into ER form and four form screens were extracted. Both queries from [13] and others were used.

Subjects ranged from engineers, physicists, and math students, to secretaries and elementary school teachers who had no math background. These subjects learned how to use the Metaform system(levels 1 through 4) in an average of 20 minutes, with a correctness of about 85%.

6.0 Conclusions

The Metaform system allows a new approach to the design of database interface languages. It presents automatically generated form screens as updatable views of multirelational information stored in an ER database embedded in the relational model.

Users are more likely to interact with a database using an information system language which is easily learned. For these reasons the Metaform system was designed to be learned in a minimal amount of time.

Clearly there are user/task combinations which are not appropriate for the Metaform system. For example, multi-relational queries (those involving several relations) which do not have a single entity as a subject or focus are troublesome; such queries are contrary to the assumptions used in developing Metaform.

Additionally, more difficult queries involving variables and/or universal quantification cannot currently be dealt with by Metaform.

However, expert relational calculus users would probably still find the Metaform interface superior in many circumstances. A programmer might be able to develop a superior form screen, but Metaform can generate its screens automatically, in almost no time, and with little cost. Thus is it ideal for many small database systems which currently are not cost efficient. Additional uses might involve being able to prototype a database design quickly in order to generate user feedback to a database administrator (DBA).

Even if a more customized form screen were needed for a particular user situation, screens generated by Metaform could serve as a starting place. Plans are underway to develop extensions to the system so that a DBA could customize form screens to include more entity types, or to have a modified layout.

In developing Metaform, we have concentrated our efforts on a particular hypothetical user: one who deals with closely related information, and who needs to learn quickly how to query and update the database. In the final analysis, the overall usefulness of Metaform depends on the ubiquity of our hypothetical user. Our experience, and the large number of form screen database application programs in current use tends to indicate the widespread need for such a system, but careful field studies on this question would be helpful in validating our assumptions.

Acknowledgments

We wish to thank Prof. Bernard Galler for taking the time to proof read this manuscript. His comments on both content and writing style have been immeasurable. We also wish to thank Prof. Marilyn Mantei for her help in undertaking the human factors research mentioned in this paper.

References

[1] Chamberlin, D. D., et al. "SEQUEL 2: A unified approach to data definition, manipulation, and control," IBM Journal of Research and Development, vol. 20, pp. 560-575, Nov. 1976.

[2] Chen, P. The Entity-Relationship Model - Toward a Unified View of Data, ACM TODS, vol. 1 no. 1, 1976, pp. 9-36.

[3] Codd, E. F. A Relational Model for Large Shared Data Banks, CACM, vol. 13, 1970, pp. 377-387.

[4] Hopcroft, J. E. and Ullman, J. D. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, Mass., 1979 418 pp.

[5] Maier, D. The Theory of Relational Databases. Computer Science Press, Rockville, MD., 1982.

[6] Codd, E. F. Extending the Database Relational Model to Capture More Meaning, IBM research report, San Jose, California, RJ2472 (32359) 1/31/79, 1979

[7] Beard, D.V., "The Formal Aspects of Metaform", Working Paper, 1984.

[8] Tsichritzis, D. C., Lochovsky, F. H. Data Models, Prentice-Hall, Englewood Cliffs, N.J., 1982, 381 pp.

[9] Reisner, P. Human Factors Studies of Database Query Languages: A Survey and Assessment, Computing Surveys, vol. 13 no. 1, 1981, pp. 13-31.

[10] Zloof, M. M., "Query-by-Example," AFIPS Conference Proceedings, National Computer Conference 44, pp. 431-438, 1975.

[11] Beard, D. V., "The Effect of the Number of Relations on Database Query Formation", Working paper, April, 1981.

[12] Beard, D. V., "A Pilot User interface Study Evaluating A form-Screen Database-Interface Language", Working Paper, March, 1983.

[13] Reisner, P. Use of Psychological Experimentation as an Aid to Development of a Query Language. IEEE Transactions on Software Engineering, SE-3, 1977, pp. 218-229.