Introduction

In this paper we propose a fundamental alternative to conventional methodology for constructing relational data base manipulation languages [7, 9, 12, 13, 14, 16, 17]. Conventional relational languages, such as DSL Alpha and SQL, are based on the twin foundations of set theory and mathematical logic [9, 10, 15]. The set theoretic foundation is undoubtedly very necessary. However, with respect to the mathematical logic are worth a critical examination, since these restrictions have been transferred to relational languages.

The restrictions that concern us have to do with the methodology of quantification. In mathematical logic, where sets are involved, quantifiers are conventionally applied to entire sets of elements [10, 15]. They are never applied to a subset of a set that is related to an element in another set. The reason appears to be the lack of the concept of a relationship between sets in mathematical logic. Clearly, if a set subset is to be related to another set element, then a definition of that relationship will be needed for qunatification.

The quantification of related subsets is of interest for designers how relational languages. In natural languages, quantification of related subsets, using any of the very large number of quantifiers that occur in natural language, is by far the most common use of quantifiers. This likely means that if we can develop a sound foundation for the quantification of related subsets, we can use it for the development of relational languages whose expressions have a structure, where quantification is involved, that mirrors the structure of natural language expressions, yet be amenable to unambiguous interpretation and translation. We believe that we have been able to show how this can be done.

We begin with the concept of a relationship, and show how a formal definition leads to the definition of many different kinds of relationship. We show that there is a relationship relation [8] for each kind of relationship. We then show how the key to quantification of related subsets is the use of such relationship relations.

1.1 Notation

We use upper case letters for attributes or attribute concatenations, except where otherwise stated. Upper case bold is used for relation names. Relation schemes are implied throughout, but are not named [15]. We use subscripted lower case for attribute values within a tuple. Thus the relation scheme [T, U, V, W] could give rise to a relation (instance) **T**(T, U, V, W), which may have a tuple $t(t_1, u_1, v_1, w_1)$. The primary key attribute (or any candidate key attribute) is underscored, and for reader convenience will always have the same upper case letter as the relation name.

1.2 The relationship concept with relational data bases

A formal definition of a relationship between relations has been proposed in [5], as follows:

Definition 1

> Consider a data base with a set of relations [**A**, **B**, **C**, ...]. There is a relationship between any arbitrary pair of relations (**A**, **B**) of this data base iff, by a process of relational algebraic operations, involving only relations from in [**A**, **B**, **C**, ...], it is possible to generate a relation **R**(A, B, ...), with A, B as minimal attributes. **R** is not necessarily a relation of the data base, and may be empty. **R** is called the relationship relation for the relationship.

In other words, there is a relationship between relations **A** and **B** if we can form a relationship relation from **A** and **B**, and possibly other relations of the data base, where the relationship relation contains attributes that are primary keys in **A** and **B**.

1.3 Relationship classification components

The major classification of relationships is into primitive and composite relationships. Suppose that we use the symbols ¶ and * for relational algebra projection and join respectively [1, 9].

Definition 2 If two relations **A** and **B** from a data base are such that a relationship relation **R**(A, B) can be generated from $\Pi_{A,B}(\mathbf{A} * \mathbf{B})$, then the relationship defined by **R** is primitive.

In other words, if we can perform a join of **A** and **B** alone, using any join attributes, and project out a relation **R** involving the primary key attributes from **A** and **B**, then we have a primitive relationship.

Definition 3. A relationship that is not primitive is composite.

This means that a composite relationship between relations **A** and **B** exists only if there is a chain of primitive relationships involving other relations of the data base, with **A** and **B** at opposite ends of the chain. The proof is given in [5]. For example, if **A** is primitively related to **P**, and **P** is primitively related to **Q** and **Q** is primitively related to **B** then there is a composite relationship between **A** and **B**, and also between **A** and **Q**, and between **P** and **B**.

There is a large number of primitive and composite relationship types. In every case it is possible to define the relationship by means of an algebraic expression that generates the relationship relation. Most relationships types are very uncommon, and [5] gives a reasonably detailed coverage. In order to illustrate the natural quantification of relationships in general, for the purpose of this paper we shall restrict our discussion to the more common relationship types. These are:

A. PRIMITIVE RELATIONSHIPS

(a) Simple 1:n relationship. This occurs when **A** and **B** contain an attribute drawn on a common domain, and that attribute is a primary or candidate key in one of the two relations. If this primary key attribute is in **A**, then the relationship is such that for one **A** tuple, there will in general be zero or more related **B** tuples. Simple 1:n relationships are very common.

(b) Co-relationship This occurs when **A** and **B** contain an attribute drawn on a common domain that is a primary or candidate key in neither relation. This type of relationship is very common. It usually has vague semantics associated with it, and has been given little recognition as a significant relationship in the literature. Because of the semantic vagueness asociated with the relationship, to fully characterize it, we need to define the level of (semantic) significance attached to any particular co-relationship. These have been defined elswhere [5, 6]. It can also be shown that the ubiquitous connection trap is related to levels of significance in co-relationships [6].

B. COMPOSITE RELATIONSHIPS

(a) Composite 1:n relationship This occurs if **A** and **B** are at opposite ends of a chain of simple (primitive) 1:n relationships. Thus if for one **A** tuple there are many **P** tuples, and for one **P** tuple many **B** tuples, then for one **A** tuple there are many **B** tuples, and there is a composite 1:n relationship between **A** and **B**.

(b) Matrix or n:m relationship This occurs if there is at least one intermediate relationship **M** in the chain between **A** and **B**, such that there is either a simple or composite 1:n relationship between **A** and **M**, and between **B** and **M**. When the 1:n relationships are both simple (or primitive), then we have the common many-to-many (n:m) relationship, where for one **A** tuple there will be zero of more related **B** tuples, and for one **B** tuple, zero or more related **A** tuples. In the literature, co-relationship and matrix relationships are sometimes confused. A little thought will show that theyare quite different.

For each of the above types of relationship there can be a cyclic version, that is, where **A** and **B** are the same relation. Cyclic relationships are covered in detail in [5], and will not be considered further in this paper. Nevertheless it can

be easily shown that the quantification principles demonstrated in this paper for the major non cyclic relationships also apply to cyclic relationships.

2. RELATIONSHIPS AND QUANTIFICATION

The universal quantifier and the existential quantifier are used in mathematical logic for specifying the quantity of a set of elements for which a logical condition holds true. A relation is a set of tuples, and so in early relational languages, such as E. F. Codd's DSL Alpha, entire relations, and only relations, were permitted to be quantified by these two quantifiers [9, 15]. Thus if we have the relation P(P, Q, S), the the expression $\exists PX \in P\ (S = 4)$ is a valid logical expression that states that there exists at least one tuple (PX) that is a member of the set of tuples P, and which has an S value equal to 4. Here the entire relation P is being quantified. Similarly, the expression $\forall PX \in P(S = 4)$ is a valid logical expression that for all (PX) tuples in P the S value is 4. The quantity specified by the existential quantifier is thus at least one member of the specified set, and the quantity specified by the universal quantifier $\forall$ is all members, without exception, of the specified set.

There are other quantifiers as well as the basic quantifiers of mathematical logic. These are the natural quantifiers of natural language. A list of the common natural quantifiers is given in the appendix. In mathematical logic these natural quantifiers are not necessary, since it is always possible to replace them by basic quantifier expressions, albeit at the expense of simplicity [10].

We have stated that in older relational languages, quantifiers are used for specifying the quantity of tuples for an entire relation, that is, quantification of the entire relation. With the exception of the existential quantifier, this has given rise to profound difficulties for the development of languages for manipulation of relational data bases, and to date the only quantifier allowed in the widely used SQL language, which is based on mathematical logic, is the existential quantifier.

The problem with the use of quantifiers is a disparity between their

use in mathematical logic, and their use in natural language. As an example of this consider a simple (primitive) 1:n relationship between relations **P**(P, Q, S) and **C**(C, P, W, V). We might imagine that tuples of relation **P** described aircraft carriers, with each carrier identified by P, and that tuples of **C** describe aircraft, with an aircraft identifed by C, and the carrier on which it is based described by P. Since P is a primary key in **P** and not a primary key in **C**, the relationship is clearly primitive and simple 1:n.

Suppose now that we want the carriers with S value 4, on which at least one aircraft has W value 7. We specify the set of **P** tuples:

$$[PX \in \mathbf{P} : S = 4 \ \wedge \ \exists CX \in \mathbf{C} ((W = 7) \ \wedge \ (P = PX.P))$$

There is no difficulty with this, for the mathematical logic formulation mirrors that used in natural language. We want each PX tuple from **P** such that S is 4, and where there exists at least one **C** tuple with matching P value (that is, at least one related **C** tuple). The above expression should be studied carefully. What we want is each carrier tuple for which at least one of the related aircraft tuples has W value 7. This just happens to be equivalent to wanting each carrier tuple for which at least one of the entire relation of aircraft tuples has a matching carrier value and a W value of 7. This type of equivalence is fortunate, for it enables relational languages to be constructed that permit the use of the existential quantifier (or equivalent structure) in a manner akin to its use in natural language, and thus permitting fairly easy use.

Unfortunately, this type of equivalence does not apply to the other quantifiers, and not even the universal quantifier. For example, suppose that we wanted each carrier with S value 4 where all of the aircraft on the carrier had W value 7. We specify the set of **P** tuples:

$$[PX \in \mathbf{P} : S = 4 \wedge \forall CX \in \mathbf{C}(((W = 7 \wedge (P = PX.P)) \ \vee \ (P \neq PX.P))]$$

This expression does not have a structure akin to any we would use in English. The structure is due entirely to the requirement that the universal quantifier

be used to specify a quantity from the entire relation **C** [3, 14, 18]. Specifically, the expression means that we want each carrier tuple where if we take all of the **C** tuples, then they either do not have a matching P value, or have a matching P value and a W value of 7. Logically this formulation is correct, but could never be acceptable in a commercial relational language. It is simply too far removed from everyday use of the universal quantifier. In everyday language we would quantify the related tuples of **C**. We would want each carrier where all of the related aircraft had a W value of 7.

The difficulties that arise with the convention of quantifying entire relations can be circumvented in the case of the universal quantifier by using a negated existential quantifier, since the statement that all the members of a set have property X means that there is not at least one without property X. It is the negated existential quanyifier technique that is used with commercial relational languages such as SQL. Although acceptable, the need for double negatives is an inconvenience that becomes unacceptable when there is nesting of quantified expressions [7, 12, 13, 14, 17].

Most relational languages permit the existential quantifier and the negated existential quantifier equivalent of the universal quantifier, and it is the entire relation that is quantified. Unfortunately, they do not permit any of the large number of natural quantifiers, such as <u>for most</u> (a majority of), or <u>for all but one</u>, or <u>for one and for all</u> [3, 18]. This is likely due to the difficulty, and often impossibility of quantifying the entire relation in a meaningful way using a natural quantifier. As an example, using relations **P** and **C**, suppose that we wanted each carrier with an S value of 4 where the majority of aircraft on the carrier had a W value of 7. Notice the phrase "the majority of aircraft on the carrier" is semantically equivalent to "the majority of related aircraft". The author can find no construction, even in plain English, that would permit the application of the quantifier <u>for most</u> to the entire relation of aircraft tuples. But, of course, all of the natural quantifiers can be easily applied to

a set of tuples that are related to some tuple.

We can summarize the argument in favor of relational languages that permit quantification of related tuples, as follows:

A. BASIC QUANTIFIERS

(a) Existential quantifier Can be used to quantify either entire relation or set of related tuples with ease.

(b) Universal quantifier. Can be used to quantify entire set only with contrivance in most applications. Can be used to quantify a set of related tuples. Is replaced by negated existential quantifier in relational languages that require quantification of entire set.

B. NATURAL QUANTIFIERS

Are in very many cases used only for quantifying a set of related tuples.

From the above it should be clear that quantification of a set of related tuples would permit expressions that mirror the English language expression, and would permit the use of all known quantifiers in a simple manner. The problem is defining "related" tuples.

2.1 Related tuples

Suppose that we have two relations **A**(A, F, G, ...) and **B**(B, T, X, ...) that are related in either a primitive or composite relationship resulting in a relationship relation **R**(A, B, ...).

Definition 5 Given an **A** tuple $a(a_n, \ldots)$, the set of related **B** tuples with the relationship defined by **R**(A, B, ...) is

$$[\ BX \subset \mathbf{B} : \quad \exists RX \in \mathbf{R}\ (RX.B = BX.B \ \wedge \ RX.A = a_n)]$$

In other words, if we have any **A** tuple $a(a_n, \ldots)$, then the set of related **B** tuples are those whose B values match the B values in those **R**(A, B) tuples with A value equal to a_n. This is illustrated in Figure 1.

In the case of the simple (primitive) 1:n relationship, if for one **A** tuple there are many related **B** tuples, then given an **A** tuple $a(a_n, \ldots)$ the related **B** tuples are those with an A value also equal to a_n. This follows from the above

definition, since for each tuple of **R**(A, B) with A = a_n there will be a corresponding **B** tuple with matching B value.

Because for any A tuple, no matter what the type of relationship, there is a clearly definable set of (via **R**) related **B** tuples, that means that it will be possible to use the related set of **B** tuples in the specification of conditions for selection of an **A** tuple. And in particular, we could specify a quantity of the set of related tuples that had to satisfy a condition in order for the **A** tuple to be selected. In other words, it will be possible to quantify the set of related tuples, which means that it will be possible to use the natural quantifiers for this purpose.

2.2 Quantification of related tuples

In order to specify an **A** tuple for selection provided a quantity of related **B** tuples satisfy specific conditions, there has to be a way to uniquely specify the relationship involved. This relationship specification has to permit unique determination of the set of related **B** tuples for the **A** tuple in question, since between **A** and **B** there could be more than one relationship. For example, if we have relations **A**(A, E, F) and **B**(A, B, K, F) where attributes drawn on the same domain have the same symbol, we will have two distinct relationships between **A** and **B**. There is a simple 1:n relationship supported by A, with relationship relation $\mathbf{R_1}$. $\mathbf{R_1}$ can be specified in a conceptual data base definition, as:

$$\mathbf{R_1}(A, B) = \pi_{A,B}(\mathbf{A} *_A \mathbf{B})$$

There is also a co-relationship supported by F, with relationship relation $\mathbf{R_2}$ $\mathbf{R_2}$ can also be specified in a conceptual data base definition, as:

$$\mathbf{R_2}(A, B) = \pi_{A,B}(\mathbf{A} *_F \mathbf{B})$$

Suppose now that we wanted each **A** tuple with E value 6, where most (that is, a majority) of related **B** tuples have K value 13. Without a specification of which relationship is involved, the request is ambiguous. But suppose that it was the relationship with relationship relation $\mathbf{R_1}$. Then we could specify the request

as

$$[AX \in \mathbf{A}: AX.E = 6 \wedge \exists (> V/2)\ \mathbf{R}_1\ [\text{related}]\ BX \in \mathbf{B}\ (K = 13)]$$

Here both AX and BX are tuple variables, and the word related is included only to help readability. The expressions specifies each **A** tuple for which E is 6, and for which a majority of $\mathbf{R}_1$ related **B** tuples have K value 13. If we were dealing instead with the relationship with relationship relation $\mathbf{R}_2$, the request could be specified similarly as:

$$[AX \in \mathbf{A} : AX.E = 6 \wedge \exists (> V/2)\ \mathbf{R}_2\ [\text{related}]\ BX\ C\ \mathbf{B}\ (K = 13)]$$

These set theoretic expressions, clearly depart from the convention of quantifying only entire relations. Nevertheless, we are departing merely from a convention that was acceptable in mathematical logic. Expressions of the type shown above are also soundly based in logic, and are simply an alternative that may be useful for constructing relational languages that are more natural for use by persons not trained in mathematics.

2.2 The natural quantifier language SQL/NQ

Set theoretic expressions of the unconventional type shown above are the basis for the natural quantifier language SQL/NQ. SQL/NQ is upward compatible with SQL, and the idea is that a user should be able to use either conventional SQL, or natural quantifier logic [2, 3, 4]. The two requests above would be specified in SQL/NQ as:

(a) Relationship relation $\mathbf{R}_1$

```
SELECT *  FROM A
WHERE E = 6 AND
      FOR MOST R1 [RELATED] B [TUPLES] (K = 13)
```

(b) Relationship relation $\mathbf{R}_2$

```
SELECT * FROM A
WHERE E = 6 AND
      FOR MOST R2 [RELATED] B [TUPLES] (K = 13)
```

Quantities inside [ ] can be omitted and merely help readability. FOR MOST is a natural quantifier, and quantifies a set of related tuples, not an entire relation.

For comparison purposes, the conventional SQL expression for the relationship supported by A (relationship relation $\mathbf{R_1}$) would be:

```
SELECT * FROM A
WHERE E = 6 AND
    (SELECT COUNT(*) FROM B
     WHERE K = 13  AND
           A.A = B.A)
     >
    (SELECT COUNT(*) FROM B
     WHERE K ≠ 13 AND
           A.A = B.A)
```

The reader is left to decipher the logic behind this. It is clearly quite unnatural, but neverthless correct and in conformance with the conventions of mathematical logic. On the other hand the SQL/NQ expressions conform more to the conventions of the use of quantifiers in natural language. Readers are left to construct the SQL expression for the case of the relationship supported by F as an exercise.

Very briefly, a reasonable sysntax for SQL/NQ expressions involving quantifiers with one level of nesting is:

```
SELECT attributes FROM relation
WHERE simple-conditions AND/OR
     xreference-conditions
```

Here an xreference-condition refers to a quantified set of related tuples that must obey some simple conditions:

```
xreference-condition := quantifier relationship-relation [RELATED] related-relation
                        [TUPLES] simple-conditions
```

If we wish to allow for nesting of quantified sets of related tuples, instead of the above production for xreference-condition, we could have:

xreference-condition:= quantifier relationship-relation [RELATED] related-relation [TUPLES]simple-conditions [AND/OR xreference-conditions]

2.4 Expressibility power

The expressibility power of a language is still a subjective measure but is a loose measure of how concise and understandable are expressions in that language compared with English. SQL/NQ rates high on such a measure, in comparison with SQL. This becomes especially apparent when composite relationships are involved. As an example, consider the relations:

A(A, F, G) **B**(B, D, C)

P(P, A, T) **Q**(Q, B, H)

M(M, P, Q, W)

Between **A** and **M**, there is one composite 1:n relationship, and another between **B** and **M**. This means that there is a matrix relationship between **A** and **B**. For this matrix relationship there will be a relationship relation **R**(A, B, M) that could be specified in the conceptual data base definition as:

$$\mathbf{R}(A, B, M) = \P_{A,B,M} (\mathbf{A} *_{A} \mathbf{P} *_{P} \mathbf{M} *_{Q} \mathbf{Q} *_{B} \mathbf{B})$$

Suppose now that we have the retrieval request involving only **A** and **B** and the relationship between them whose relationship relation is **R**:

> Retrieve each **A** tuple for which the G value is 6, and where for the majority of related **B** tuples the C value is 13.

With SQL/NQ the expression is trivial:

```
SELECT * FROM A
WHERE G = 6 AND
      FOR MOST R RELATED B TUPLES (C = 13)
```

Here the economy of expression is obtained because of the use of a previously specified relationship (based on **R**) and the use of a quantified set of related

tuples. In the corresponding SQL expression, we have neither of these conveniences:

```
SELECT * FROM A, XA
WHERE G = 6 AND
    (SELECT  COUNT(*) FROM B
     WHERE C = 13 AND
     B IN (SELECT B FROM Q
           WHERE Q IN (SELECT Q FROM M
                       WHERE P = (SELECT P FROM P
                                  WHERE A = (SELECT A FROM A
                                             WHERE A = XA.A)))))
     >
    (SELECT COUNT(*) FROM B
     WHERE C = 13 AND
     B IN (SELECT B FROM Q
           WHERE Q IN (SELECT Q FROM M
                       WHERE P = (SELECT P FROM P
                                  WHERE A = (SELECT A FROM A
                                             WHERE A = XA.A)))))
```

with the result that we get a complex expression that requires skill in SQL to construct.

2.5 The mode of association concept and the relationship relation

There is an alternative method of interpreting a relationship relation that emphasises the semantics involved, rather than the technical basis for the relationship (such as the attributes that support it, and so on). This involves the mode of association concept. A mode of association is simply a set of propositions in logic **R**(A, B). An element of the set could be $r(a_2, b_3)$, where a_2 and b_3 identify entities that are the subject and object of the proposition. For example, if **A** and **B** are relations related in a simple 1:n relationship, with A identifying a carrier and B identifying aircraft,then $r(a_2, b_3)$ is the symbolic form for the

proposition:

Carrier a_2 carries aircraft b_3.

It is as a set of such propositions that the relationship relation **R** could be presented to users.

Conclusions

The convention that that only the basic existential and universal quantifier be used to quantify entire relations has resulted in very retrictive relational manipulation languages, such as SQL. However, to permit quantification of related sets of tuples, which is the equivalent of the technique used in natural language, it is necessary to use precise definitions of relationships in data base definitions or schemas. We propose that this be done using relationship relations. This will enable every kind of relationship possible in a relational data base to be used in the quantification of related sets of tuples. We have shown in this paper how this can be done with non cyclic relationships, and given an example of a language, called SQL/NQ, which is upward compatible with SQL, in which these techniques are incorporated. An advantage of quantifying related sets of tuples instead of merely entire relations is that the the complete range of natural quantifiers can then be used in the language. As we have demonstated, the use of both quantified sets ofrelated tuples, and natural quantification, will permit retrieval expressions that both much more concise that those required with SQL, and also much closer in structure to those of natural language.

Appendix 1. Common natural quantifiers

1. FOR n, FOR THE n, FOR EXACTLY n	$\exists(n)$
2. FOR AT LEAST n, FOR n OR MORE	$\exists(\geq n)$
3. FOR AT MOST n, FOR n OR LESS	$\exists(\leq n)$
4. FOR AT LEAST 1, FOR ONE OR MORE, FOR SOME	$\exists(\exists), \exists, \exists(\geq 1)$
5. FOR BETWEEN n AND m	$\exists(>n \wedge <m)$
6. FOR ALL, FOR EACH, FOR ALL IF ANY, FOR EACH IF ANY	$\exists(\forall), \forall$
7. FOR ALL BUT n	$\exists(\forall - n)$
8. FOR ONE AND ALL	$\exists(\geq 1 \wedge \forall), \exists(\exists \wedge \forall)$
9. FOR NO	$\exists(0)$
10. FOR SOME BUT NOT ALL	$\exists(\geq 1 \wedge \neg\forall), \exists(\exists \wedge \neg\forall)$
11. FOR SOME BUT NOT n	$\exists(\geq 1 \wedge \neg n), \exists(\exists \wedge \neg n)$
12. FOR SOME BUT NOT MORE THAN n	$\exists(\geq 1 \wedge \leq n), \exists(\exists \wedge \leq n)$
13. FOR SOME BUT LESS THAN n	$\exists(\geq 1 \wedge < n), \exists(\exists \wedge < n)$
14. FOR MOST, FOR A MAJORITY OF	$\exists(> \forall/2)$
15. FOR A MINORITY OF	$\exists(< \forall/2)$
16. FOR x PERCENT OF, FOR EXACTLY x PERCENT OF	$\exists(x/100)$
17. FOR AT MOST x PERCENT OF FOR x PERCENT OR LESS OF	$\exists(\leq x/100)$
18. FOR AT LEAST x PERCENT OF FOR x PERCENT OR MORE OF	$\exists(\geq x/100)$
19. FOR BETWEEN x AND y PERCENT OF	$\exists(> x/100 \wedge < y/100)$

REFERENCES

1. Aho, A. V., Beeri, C., and Ullman, J. D., The theory of joins in relational data bases. ACM Trans on Database Syst., 4(3), 1979, 317-314.

2. Bradley, J., An extended owner-coupled set data model and predicate calculus for data base management, ACM Trans. on Database Syst., 3(4), 1978, 385-416.

3. Bradley. J. SQL/N and attribute/relation associations implicit in functional dependencies, Int. J. Computer & Information Science, 12(20), 1983.

4. Bradley, J. A group-select operator for relational algebra and implications for database machines, IEEE Trans. on Software Syst., to appear.

5. Bradley, J. A fundamental classification of relationships in relational data bases, Research Report No. 87/265/13, Univ. of Calgary, Alberta, Canada, 1987, 30 pages.

6. Bradley, J. Co-relationships, levels of significance, and the source of the connection trap in relational data bases, Research Report No. 86/250/24, 1986.

7. Chamberlin, D. D., et al. SEQUEL 2: A unified approach to data definition, manipulation and control, IBM J. Res. & Dev., 20(6), 1976, 560-575.

8. Chen, P. P. The entity-relationship model: Towards a unified view of data, ACM Trans. on Database Syst., 1(1), 1976, 9-36.

9. Codd, E. F. Relational database: A practical design for productivity, CACM, 25(2), 1982, 109-117.

10. Hilbert, D. and Ackerman, W. Principles of Mathematical Logic, Chelsea Publishing Co., New York, 1950.

11. Kaplan, S. J. Designing a portable natural language query system, ACM Trans. on Database Syst., 9(1), 1984, 1-19.

12. Kim, W. On optimizing an SQL nested query, ACM Trans. on Database Syst., 7(3), 1982, 443-469.

13. Kim, W. Gajski, D., Kuck, D. J. A parallel pipelined relational query processor, ACM Trans. on Database Syst., 9(2), 214-242.

14. Luk, W. S., and Kloster, S., ELFS: English language from SQL, ACM Trans on Database Syst., 11(4), 1986, 447-472.

15. Maier, D. The Theory of Relational Databases, Computer Science Press, Potomac, Md., 1983.

16. Reiter, R. A sound and sometimes complete query evaluation algorithm for relational data bases with null values, J. of ACM, 33(2), 1986, 349-370.

17. Ullman, J. D., Implementation of logical query languages for data bases, ACM Trans on Database Syst., 10(3), 1985, 289-321.

18. Welty, C., and Stemple, D. W. Human factors comparison of procedural and non procedural query languages, ACM Trans on Database Syst., 6(4), 1981, 626-649.