INTRODUCTION

In an earlier paper [5], we introduced a new set theoretic expression technique involving the natural quantifiers [2, 3] for relationships in relational data bases [8, 9]. The new technique is an extension of the conventional set theoretic techniques used in first order predicate calculus [7, 10, 11, 14, 16]. The innovation in the new technique is quantification, by any kind of quantifier [2, 3], of related sets of tuples. In contrast, conventional set theoretic techniques permit only quantification of entire relations, using either the existential or universal quantifiers [11, 14]. The natural quantifier set theoretic technique depends on new tuple groupings, namely related sets of tuples, which are precisely defined in [5]. However, in [5] only non recursive relationships were considered. In this paper, we further extend the technique to include recursive relationships.

Natural quantifier set theoretic techniques are important because they can be used as the foundation for natural quantifier data base languages [2, 3]. In like manner, languages such as SQL [6,12,13,15, 17,18], which do not permit natural quantifiers, need to be soundly based on the conventional set theoretic techniques of first order predicate calculus [7]. A description of one prototype natural quantifier data base language, called SQL/NQ, which is an extension of SQL, has been given elsewhere [2]. SQL/NQ is one of many possible languages that can be based on natural quantifier set theoretic techniques. However, published treatments on SQL/NQ deal only with non recursive relationships. In this paper we show how

natural quantifier set theoretical techniques for recursive relationships help us devise recursive relationship facilities for a natural quantifier data base language, such as SQL/NQ.

The reader is assumed to be familiar with the concept of a natural quantifier [2, 3]. A list of common natural quantifiers, and a useful notation, is given in Appendix 1.

Notation

Upper case letters are used for attributes of relations, and upper case bold, or subscripted upper case bold, is used for relation names. A primary key attribute is underscored, and for reader convenience has the same upper case letter as the relation name. Relation schemes are implied but are not named [14]. Subscripted lower case is used for attribute values, and lower case letters are used for tuple identifiers. Thus the relation scheme [T, U, V, W] could give rise to the relation instance **T**($\underline{T}$,U,V,W), which may have a tuple t(t_1, u_1, v_1, w_1). TX denotes a tuple variable for the relation **T**.

1 RELATIONSHIPS

1.1 Relationship definition

A practical definition of a relationship of any kind in a relational data base is as follows:

Definition 1 In a relational data base with a set of relations [**A**, **B**, **C**, ...], there is a relationship between any arbitrary pair of relations (**A**,**B**) if it is possible to generate a relation **R**(A,B, ...), with A and B as minimal attributes, from the set of relations by a process of relational algebraic operations. **R** is called the relationship relation for the relationship.

Notice that attributes A and B are the primary keys of relations **A** and **B**. Each tuple of **R** relates an A entity and a B entity, and thus **R** must define a relationship.

1.2 Classification of relationships

Relationships can be either primitive or composite. In addition, whether or not a relationship is primitive, it must also be either recursive or non recursive.

Assume that **p** and * are used for relational algebraic projection and natural join respectively [1].

Definition 2 The relationship between two relations (**A**,**B**) from a set of relations [**A**,**B**, **C**, ...] is primitive
if **R**(A,B) = $\mathbf{p}_{A,B}(\mathbf{A}*\mathbf{B})$.

Definition 3 A relationship that is not primitive is composite.

Definition 4 Any relationship, whether primitive or composite, between relations **A** and **B** is recursive if, and only if, **A** = **B**. Otherwise the relationship is non recursive.

Primitive relationships are simple in concept, and will occur if the two relations involved have at least one common attribute drawn on the same domain [9]. It is this common attribute that makes possible formation of **R** from a single join. We refer to these common attributes as the relationship (supporting) attributes.

Where the relationship is composite there is no common join attribute, and the relationship is due to a chain of primitive relationships; it thus requires a chain of joins to form **R**. Typically we will have $\mathbf{R}(A,B) = \mathbf{p}_{A,B}(\mathbf{A*H*K*L} \ldots \mathbf{*B})$

The primitive relationships are:

(a) Simple 1:n relationship Here one of the relationship attributes is a primary key and the other is not. If A in **A** is the primary key attribute and A in **B** is not, then for a given **A** tuple there may exist many related **B** tuples.

(b) Co-relationship Here neither of the relationship attributes is a primary key. This relationship is common but much misunderstood and little investigated. It is often confused with a many-to-many relationship. It is implicated in the connection trap and join dependencies [4].

The main composite relationships are:

(a) Composite 1:n relationship There is a composite 1:n relationship between **A** and **Z** if there is a primitive 1:n relationship between **A,B**, between **B,C**, ..., between **X,Y**, and between **Y,Z**, that is, if **A** and **Z** are connected by a chain of 1:n relationships.

(b) Many-to-many or n:m relationship This relationship occurs between **A** and **Z** if there exists a single relation **M**, such that -there is either a primitive or composite 1:n relationship between **A** and **M**, and also a primitive or composite 1:n relationship between **Z** and **M**.

Note that other, more obscure, composite relationships are a possibility, according to Definition 3.

Recursive relationships may be either primitive or composite according to Definition 4. The most common and important recursive relationships are (a) the recursive primitive 1:n relationship, where relation **A** is primitively 1:n related to **A**, and (b)

the recursive version of the many-to-many relationship between **A** and **B**, where **M** is the only other relation in the chain, and, of course, **A** = **B**.

We examine these to types of recursive relationship in a set theoretic context in this paper.

1.3 Related tuples

In an earlier paper [5], we proposed the following method of defining a set of related tuples:

Definition 4 Suppose we have two relations **A**(A,...), **B**(B, ...) related in any relationship for which there is a relationship relation **R**(A,B,...). The set of **B** tuples that are related to any **A** tuple $a(a_n,..)$ in this relationship is the set:

$$[BX \in \mathbf{B}: \exists(>=1)\ RX \in \mathbf{R}\ (RX.B = BX.B\ \&\ RX.A = a_n)] \quad (1)$$

A convenient notation for the set of **B** tuples related to an **A** tuple a for the relationship defined by the relationship relation **R** is [a:**R**:**B**]. Where the **A** tuple involved is the value of a tuple variable AX, the set of related **B** tuples can be denoted by [AX:**R**:**B**]. We use the symbol $\exists$ to denote quantification. Thus $\exists(>=1)$, for at least one, is the existential quantifier. A list of quantifiers is given in the appendix.

2 RECURSIVE 1:n RELATIONSHIPS AND SET THEORETIC EXPRESSIONS.

2.1 Recursively related tuples

Suppose we have the relation **A**(A, B, P), where P is drawn on the same domain as A. There is a recursive primitive 1:n relationship between the tuples of **A**, where the relationship relation is

$$\mathbf{R}(A_s, A_j) = \mathbf{p}_{A_s, A_j}[\mathbf{A}(A_s, B, P) *_{A_s, P} \mathbf{A}(A_j, B, P)] \qquad (2)$$

Here the attributes A_s, A_j, and A are all identical. The subscript s denotes senior and j denotes junior, and are used for purposes of clarifying operations. In any **R** tuple the two A values will typically not be the same, and the tuple will relate a parent or senior A entity to a child or junior A entity.

A recursive 1:n relationship can have many levels. A given A entity a_1 can be related to many other A entities. These other related A entities can be determined from **R**, and this gives level 1 of the relationship. However, each of these other A entities are related to many further A entities, which can can also be determined from **R**. This gives us the second level. Similarly we can have a third level, and so on.

The number of levels depends on the A entity chosen. An example clarifies this. An A enity a_j could be a corporation, and P gives the A entity (corporation a_s) that is the outright owner of that corporation a_j. Thus for a given corporation a_s, there are

many 100% owned subsidiary corporations, and for each of these subsidiaries there are many lower level subsidiaries, and so on, down to the lowest level subsidiaries that have no further subsidiaries.

There is no problem with set theoretic expressions where only one level is involved. These are no different from expressions where there is no recursion, covered in [5]. For example, suppose we want the records for those companies with B value 4, where all immediate subsidiaries have a B value of 6. The set theoretic expression is:

$$[AX \in \mathbf{A}: B = 4 \ \& \ \exists (\forall)\ AY \in [AX:\mathbf{R}:\mathbf{A}]\ (B = 6)] \qquad (3)$$

Here [AX:**R**:**A**] is the set of **A** tuples that are 1:n related to the tuple AX via **R**. Because of the definition of [AX:**R**:**A**] as:

$$[AY \in \mathbf{A}: \exists(>=1)\ RX \in \mathbf{R}\ (RX.A_j = AY.A_s \ \& \ RX.A_s = AX.A] \qquad (4)$$

following expression (1), we are dealing only with the first level of the relationship. Clearly, in order to deal with higher levels of the relationship, we need a way of defining a related set of tuples where 'related' refers to a higher level of the relationship.

2.2 Higher level related sets of tuples

To enable to to clearly discuss multiple levels in a recursive 1:n relationship, let us adopt the following terminology. Sup-

pose the relation **A** dealt with corporations. Then at any level of the relationship for a given parent or senior company there will be many immediate or level-1 subsidiary or junior companies. In addition, for a given parent company there will be many subsidiaries of the level-1 subsidiaries, that is, many level-2 subsidiaries. Similarly, for a given parent company, there will be many level-3 subsidiaries, and so on.

In the general case, a given parent **A** tuple a_1 can be 1:n level-1 related to many (immediately) subsidiary **A** tuples, can be level-2 related to many subsidiary **A** tuples, level-3 related to many **A** tuples, and so on.

Thus expression (4) and [AX:**R**:**A**] refer to the set of **A** tuples that are level-1 related to the tuple contained in AX. We need expressions and notations for the sets of tuples that are level-n related to the tuple contained in AX.

When we are dealing with level-2 related tuples the relationship relation that applies is not **R**, as defined in (2), but the projection on A_s and A_j of

$$\mathbf{R}(A_s, A_{j'}) \; *_{A_{j'}, A_{s'}} \; \mathbf{R}(A_{s'}, A_j)$$

which we can denote as $\mathbf{R}^{2(*)}$, or $\mathbf{R}^{2(*)}(A_s, A_j)$. Similarly, when dealing with level-3 related tuples, the relationship relation that applies will be the projection on A_s and A_j of:

$$\mathbf{R}^{2(*)}(A_s, A_{j'}) \; *_{A_{j'}, A_{s'}} \; \mathbf{R}(A_{s'}, A_j)$$

which can be denoted by $\mathbf{R}^{3(*)}$. In this way $\mathbf{R}^{n(*)}$ is the relationship relation for dealing with level-n related tuples. Note that with this notation $\mathbf{R}^{1(*)}$ is the same as **R**.

We can now define the set of tuples that are level-n related to a given tuple a as:

$$[AY \in \mathbf{A}: \exists(>=1)\ RX \in \mathbf{R}^{n(*)}\ (RX.A_j = AY.A\ \&\ RX.A_s = a.A)] \quad (5)$$

A convenient notation for the set of tuples that are level-n related to the tuple in variable AX, instead of a, is $[AX:\mathbf{R}^{n(*)}:\mathbf{A}]$. And we note that $[AX:\mathbf{R}^{1(*)}:\mathbf{A}]$ for set of tuples that are level-1 related to AX is the same as [AX:**R**:**A**], used in expression (3) above.

2.3 Set theoretic expressions with higher level related sets of tuples

Once we can define a set of related tuples at any level of the relationship, and have suitable symbols for all of the quantifiers [appendix], it is possible to construct exact theoretic expressions involving natural quantification of any set of related tuples in a recursive 1:n relationship. A quantity of a set of level-n related tuples of **A** for which a certain condition holds can be expressed as:

$$\exists(\text{quantity})\ AY\ C\ [AX:\mathbf{R}^{n(*)}:\mathbf{A}]\ (\text{condition})$$

Here $\exists$(quantity) is any quantifier. The above expression is a logical expression, and will be true only if each one of the specified quantity of level-n related tuples satisfies the specified condition. Thus

$$\exists(\forall)\ AY \in [AX:\mathbf{R}^{4(*)}:\mathbf{A}]\ (B = 6) \qquad (6)$$

is true only if all (AY-contained) tuples of **A** that are level-4 related to tuple AX have B = 6. We can incorporate such logical expressions in general set theoretic expressions as explained below.

Consider again the universal quantifier expression (3) earlier. Suppose now that we are dealing with level-3 subsidiaries, instead of immediate (level-1) subsidiaries. We can rewrite expression (3) for this case as:

$$[AX \in \mathbf{A}:\ B = 4\ \&\ \exists(\forall)\ AY \in [AX:\mathbf{R}^{3(*)}:\mathbf{A}]\ (B = 6)] \qquad (7)$$

This expression is clearly concise, when we consider the complexity of what has been specified. Of course, it depends on prior definitions of $\mathbf{R}^{3(*)}$ and related sets of tuples $[AX:\mathbf{R}^{3(*)}:\mathbf{A}]$.

Just how concise this expression is can be better appreciated by comparing with the corresponding SQL expression, based on conventional set theoretic expression techniques:

```
SELECT * FROM A, W WHERE B = 4
AND NOT EXISTS (SELECT * FROM A, Z
                WHERE Z.B NOT = 6
                AND Z.P IN (SELECT A FROM A, Y
                            WHERE Y.P IN (SELECT A FROM A, X
                                          WHERE X.P = Z.A)))    (8)
```

where **W** (parent), **X** (level-1 subsidiary), **Y** (level-2 subsidiary), and **Z** (level-3 subsidiary) are SQL relation labels.

2.4 Natural quantifiers

The expression structure in (7) can be used with the natural quantifiers, as well as the universal or existential quantifiers [10,11]. For example, the expression for those **A** tuples where B = 4 and most, or a majority, of level-3 related **A** tuples have a B value of 6 is:

$$[AX \in \mathbf{A}:\ B = 4\ \&\ \exists(>\forall/2)\ AY \in [AX:\mathbf{R}^{3(*)}:\mathbf{A}]\ (B = 6)] \qquad (9)$$

2.5 General expressions

The general set theoretic expression is given by

[AX ∈ **A**: attribute-condition & quantifier-condition]

where:

quantifier-condition = (quantity) BX ϵ [AX:**R**:**B**] (attribute-condition);

$\exists$(quantity) is any quantifier;

attribute-condition is any logical expression involving the attributes of a relation, or:

attribute-condition is:

(attribute-condition & quantifier-condition)

This permits unlimited expansion of the expressions.

As an example, suppose we want the set theoretic expression for the **A** tuples with B = 4 with a majority of level-2 related **A** tuples with B = 6 and with at least 3 level-1 related **A** tuples with B = 8:

$$[AX \in \mathbf{A}:\ B = 4\ \&\ \exists(>\forall/2)\ AY \in [AX:\mathbf{R}^{2(*)}:\mathbf{A}]\ (B = 6\ \&\ F(>=3)\ AY \in [AY:\mathbf{R}:\mathbf{A}]\ (B = 8))] \quad (10)$$

2.6 Natural quantifier data base languages

One experimental natural quantifier data base language is SQL/NQ, which is an extension to SQL. SQL/N has been described in detail elsewhere for relationships that are non recursive [2,3]. Just as SQL is based on conventional set theoretic expression techniques, SQL/NQ can be viewed as based on natural quantifier set

theoretic expression techniques. Thus SQL/NQ expressions for recursive relationships should be based on the set theoretic expression techniques described above.

To extend SQL/NQ for the case of recursive 1:n relationships, we need an obvious, simple and unambiguous way to refer to related sets of tuples for the different levels of the relationship. One possibility is to adopt the convention that '**R** RELATED **A** TUPLES' is the same as 'LEVEL-1 **R** RELATED **A** TUPLES' and is the same as [AX:$\mathbf{R}^{1(*)}$:**A**]. Continuing, 'LEVEL(2) **R** RELATED **A** TUPLES' is [AX:$\mathbf{R}^{2(*)}$:**A**], and so on.

With this notation we can write expression (9) in SQL/NQ as:

```
SELECT * FROM A WHERE B = 4 AND
          FOR MOST R RELATED A TUPLES (B = 6)      (11)
```

Similarly, expression (10) above can be rewritten in SQL/NQ as:

```
SELECT * FROM A WHERE B = 4 AND
    FOR MOST LEVEL(2) R RELATED A TUPLES
      (B = 6 AND FOR AT LEAST 3 R RELATED A TUPLES (B = 8))  (12)
```

The natural quantifiers here are AT LEAST 3 and FOR MOST.

3 RECURSIVE MANY-TO-MANY RELATIONSHIPS

3.1 Relationship levels

As with the recursive 1:n relationship, there are multiple levels in a recursive many-to-many relationship, although things are more complex.

Consider the relations **A**(A, B, C) and **M**(A_s, A_j, D), where A, A_s and A_j are all drawn on the same domain. For convenience we can regard A as identifying a corporation, where a corporation can own a portion (given by D shares) of another corporation. Thus A_s owns a part (D) of A_j. Note that attributes in **M** give information about the relationship between two corporations; attributes in **A** give information about a corporation. Also let M be the composite A_sA_jD.

There are two 1:n relationships between **A** and **M**. We can denote the relationship relations for the relationships supported by (A, A_s) and (A, A_j) as $\mathbf{R}_s(A_s, M)$ and $\mathbf{R}_j(A_j, M)$ respectively. The many-to-many relationship between **A** and **A** has the relationship relation $\mathbf{R}(A_s, A_j)$, which is the projection of **A*M*A** on A_s and A_j.

The relationship relation $\mathbf{R}_s(A_s, M)$ gives the relationship for one parent (A) and many immediate subsidiaries (M). The relationship relation $\mathbf{R}_j(A_j, M)$ gives the relationship for one subsidiary (A) and many immediate parents (M). The relationship relation $\mathbf{R}(A_s, A_j)$ gives the relationship for either one parent (A_s) and many related subsidiaries (A_j), or one subsidiary (A_j) and many related parents (A_s).

If we apply expression (1) to each of these relationship relations, the expressions for related sets of tuples are:

M tuples related to AX via $\mathbf{R}_s$ is: $[AX:\mathbf{R}_s:\mathbf{M}]$

M tuples related to AX via $\mathbf{R}_j$ is: $[AX:\mathbf{R}_j:\mathbf{A}]$

A subsidiary tuples related to AX via $\mathbf{R}(A_s,A_j)$ is :

$[AX:\mathbf{R}(A_s,A_j):\mathbf{A}]$

A parent tuples related to AX via $\mathbf{R}(A_j,A_s)$ is:

$[AX:\mathbf{R}(A_j,A_s):\mathbf{A}]$

The relation **R** gives us the immediate (level-1)subsidiaries for a given parent, or the immediate (level-1) parents for a given subsidiary. However, for a given parent there can be many related subsidiaries of immediate subsidiaries (level-2 subsidiaries), and so on. And for a given subsidiary, there can be many related parents of immediate parents (level-2 parents), and so on.

To deal with level-2 of the relationship, the relationship relation needed is the projection on A_s and A_j of the join of

$$\mathbf{R}(A_s,A_j') \; *_{A_j',A_s'} \; \mathbf{R}(A_s',A_j)$$

which we can denote by $\mathbf{R}^{2(*)}$. Similarly, the relationship relation for level-3 of the relationship is $\mathbf{R}^{3(*)}$, formed by a join of $\mathbf{R}^{2(*)}$ and **R**, and so on.

With level-n of the relationship, and following expression (1), we can denote related sets of tuples as follows:

level-n subsidiary **A** tuples related to AX via $\mathbf{R}^{n(*)}(A_s,A_j)$ are:

$$[AX:\mathbf{R}^{n(*)}(A_s,A_j):\mathbf{A}]$$

level-n parent **A** tuples related to AX via $\mathbf{R}^{n(*)}(A_j,A_s)$ are:

$$[AX:\mathbf{R}^{n(*)}(A_j,A_j):\mathbf{A}]$$

3.2 Set theoretic expressions with a recursive many-to-many relationship

The related sets of tuples defined above enable us to write set theoretic expressions for any level or levels of a recursive many-to-many relationship. A few examples should show this, beginning with level 1.

Suppose we want the companies with B value 4 that own more than 10 million shares (D = 10) in all of their immediate subsidiaries:

$$[AX \in \mathbf{A}:\ B = 4\ \&\ \exists(V)\ AY \in [AX:\mathbf{R}_s:\mathbf{M}]\ (D = 10)] \qquad (13)$$

Suppose we want the each company with B value 4 such that each of the immediate parents owns more than 10 million shares in that company:

$$[AX\ C\ \mathbf{A}:\ B = 4\ \&\ \exists(V)\ AY \in [AX:\mathbf{R}_j:\mathbf{A}]\ (D = 10)] \qquad (14)$$

Suppose we want each company with B value 4 where the majority of immediate subsidiaries each have a C value of 6:

$$[AX \in \mathbf{A}: B = 4 \ \& \ \exists(>V/2)\ AY \in [AX:\mathbf{R}(A_s,A_j):\mathbf{A}]\ (C = 6)] \quad (15)$$

Suppose we want each company with a B value of 4 where at least 4 of the immediate parent companies each have a C value of 6:

$$[AX \in \mathbf{A}: B= 4 \ \& \ \exists(>=4)\ AY \in [AX:\mathbf{R}(A_j,A_s):\mathbf{A}]\ (C = 6) \quad (16)$$

Suppose we want each company with a B value of 4 where the majority of immediate subsidiaries each have a C value of 6 and more than 10 million shares in at least two of their immediate subsidiaries:

$$[AX \in \mathbf{A}: B = 4 \ \& \ \exists(>V/2)\ AY \in [AX:\mathbf{R}(A_s,A_j):\mathbf{A}]$$
$$(C = 6 \ \& \ \exists(>=2)\ AZ \in [AY:\mathbf{R}_s:\mathbf{M}]\ (D = 10))] \quad (17)$$

Finally, suppose we want each company with a B value of 4, where the majority of level 3 subsidiaries each have a C value of 6:

$$[AX \in \mathbf{A}: B = 4 \ \& \ \exists(>V/2)\ AY \in [AX:\mathbf{R}^{3(*)}(A_s,A_j):\mathbf{A}]\ (C = 6)] \quad (18)$$

3.3 Natural quantifier data base languages with recursive many-to-many relationships

The above set theoretic technique can guide us as to ways of incorporting recursive many-to-many relationship facilities in natural quantifier data base languages, such as SQL/NQ. As with

recursive 1:n relationship facilities, we need an obvious, simple and unambiguous method of referring to the sets of related tuples. The following is merely a suggestion

[AX:$\mathbf{R}_s$:**M**]	$\mathbf{R}_s$ RELATED **M** TUPLES
[AX:$\mathbf{R}_j$:**M**]	$\mathbf{R}_j$ RELATED **M** TUPLES
[AX:$\mathbf{R}^{n(*)}(A_s,A_j)$:**A**]	LEVEL(n) **R** RELATED **A** JUNIOR TUPLES
[AX:$\mathbf{R}^{n(*)}(A_j,A_s)$:**A**]	LEVEL(n) **R** RELATED **A** SENIOR TUPLES

In practice more informative names might be chosen for the relationship relations $\mathbf{R}_s$, $\mathbf{R}_j$, and **R**. In addition LEVEL(1) could be omitted.

Using these naming conventions, expressions 13-18 could be expressed in SQL/NQ as follows:

SELECT * FROM **A** WHERE B = 4 & FOR ALL $\mathbf{R}_s$ RELATED **M** TUPLES (D = 10)

SELECT * FROM **A** WHERE B = 4 & FOR ALL $\mathbf{R}_j$ RELATED **M** TUPLES (D = 10)

SELECT * FROM **A** WHERE B = 4
& FOR MOST **R** RELATED **A** JUNIOR TUPLES (C = 6)

SELECT * FROM **A** WHERE B = 4
& FOR MOST **R** RELATED **A** SENIOR TUPLES (C = 6)

```
SELECT * FROM A WHERE B = 4
       & FOR MOST R A JUNIOR TUPLES (C = 6
         & FOR AT LEAST 2 R_S RELATED M TUPLES (D = 10))

SELECT * FROM A WHERE B = 4
      & FOR MOST LEVEL(3) R RELATED A JUNIOR TUPLES (C = 6)
```

CONCLUSIONS

We have shown that is it possible to extend conventional set theoretic techniques to permit the use of natural quantifiers. With conventional set theoretic techniques applied to relational data bases, the only quantifiers allowed are the existential and universal quantifiers; these quantifiers are exclusively used for specifying the quantity of tuples, in an entire relation of tuples, that satisfy specified conditions. In an earlier extension to conventional set theoretic expression techniques, facilities for quantification of related sets of tuples for any non recursive relationship were developed. In this paper, a further extension has been developed, which premits natural quantification, with any quantifier, of sets of related tuples where the relationship is recursive - either recursive 1:n, or recursive many-to-many. We have also proposed facilities for handling recursive relationships in a natural quantifier data base language, SQL/NQ. This language

can thus be regarded as fundamentally based on the natural quantifier extension to conventional set theoretic techniques.

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 Data Base Sys., 4(3), 1979, 317-314.

2. Bradley, J. SQL/N and attribute/relation associations implicit in functional dependencies, Int. J. Computer & Information Science, 12(20), 1983, 65-86.

3. Bradley, J., A group-select operation for relational algebra and implications for data base machines, IEEE Trans. on Software Sys. 14(1), 1988, 126-29.

4. Bradley, J. Polygonal join dependencies, closed co-relationship chains, and the connection trap in relational data bases, Computer Journal, in press.

5. Bradley, J., Natural quantifier set theoretic expression techniques, to be published.

6. 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.

7. Codd, E. F. Relational completeness of data base sublanguages. In "Database Systems", Courant Computer Science Symposia, Vol. 6, Prentice-Hall, Englewood Cliffs, New Jersey, 1971.

8. Codd, E. F. Extending the relational data base model to capture more meaning, ACM Trans. on Database Sys., 4(4), 1979, 397-434.

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

10. Gallaire, H., Minker, J. and Nicholas, J. M. Logic and data

bases: A deductive approach, ACM Comput. Surveys, 16(2), 1984, 183-185.

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

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

13. Luk., W.S. and Kloster, S. ELFS: English language from SQL, ACM Trans. on Database Sys., 11(4) 1986, 447-442.

14. Maier, D. Theory of Relational Databases, Computer Science Press, Potomac, Maryland, 1983.

15. 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.

16. Rybinski, H. On first-order-logic databases, ACM Trans. on Database Sys., 12(3), 1987, 325-349.

17. Stonebraker, M., Anton, J., and Hanson, E. Extending a database system with procedures, ACM Trans. on Database Sys., 12(3) 1987, 350-376.

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