RELATIONSHIP RELATIONS FOR RELATIONAL DATA BASES AND NATURAL QUANTIFIER SET THEORETIC TECHNIQUES

In this paper we propose a set theoretic expression technique that enhances conventional set theoretic expression methods [10, 15]. Set theoretic expression techniques are fundamental in relational data base management because all important relational data base manipulation languages, such as SQL [7, 12, 14, 15, 18], are based on them [9, 15]. An exception is languages for the entity-relationship variation on the relational approach [8].

Unfortunately, conventional set theoretic expression techniques have restrictions with regard to relationships and quantifiers [10, 15, 18]. These restrictions have resulted in an unnecessary degree of inflexibility [18] in current data base manipulation languages. In this paper we show how to extend existing set theoretic expression techniques so as to remove these restrictions. The result is a set theoretic foundation on which to base languages that permit application of the wide range of quantifiers found in natural language [11, 18] to any relationship that can occur in a relational data base.

We begin with a general definition of a relationship that leads naturally to the possibility of many different kinds of relationship. We then define relationship relations for unambiguous

characterization of relationships. Finally we define sets of related tuples in terms of relationship relations and introduce a convenient notation for such sets. These components allow us to develop a set theoretic expression technique that permits the use of natural quantifiers with all types of relationships.

Notation

Upper case letters are used for attributes of relations, and 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 [15]. 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 must define a relationship.

1.2 Classification of relationships

Relationships can be either primitive or composite. 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 $\mathbf{R}(A,B) = \mathbf{p}_{A,B}(\mathbf{A}*\mathbf{B})$.

Definition 3 A relationship that is not primitive is composite.

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 [5,6] and join dependencies [5].

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. However, in practice, it is composite 1:n and many-to-many relationships that matter, so that when we refer in this paper to any relationship, for convenience it may be assumed that if the relationship should be composite, it will be one of these two, and not an obscure relationship.

2 RELATIONSHIPS AND QUANTIFICATION

2.1 Quantifiers

A quantifier is used to specify a quantity of objects. In mathematical logic a quantifier is used to specify a quantity of members of a set for which some stated property is true [10]. The quantifiers used in logic are restricted to the existential quantifier and the universal quantifier. The quantity specified by the existential quantifier is <u>at least one</u> of the members of the set; the quantity specified by the universal quantifier is <u>all</u> of the members of the set.

In addition to the universal and existential quantifiers there is a large number of other quantifiers, or natural quantifiers, that are used in everyday language with quite precise meanings [3]. Examples are: <u>for all but one</u>, <u>for one and all</u>, <u>for</u>

at least 2, and for a majority of. A list of the more common natural quantifiers and suitable symbols is given in the appendix. We shall use the symbolism ∃(quantity) for all quantifiers. In this symbolism ∃(∀) is the univeral quantifier, ∃(>=1) is the existential quantifier, and ∃(>∀/2) is the quantifier for a majority of.

2.2 Quantifiers and data base manipulation languages

Most relational data base languages, such as SQL [7, 15], are based on set theoretic expression techniques as applied to relations [15]. In set theoretic expressions, as in logic, the universal and existential quantifiers are used to specify a quantity of members of a set that has a specific property.

A relation is a set of tuples, and so set theoretic expressions applied to relations use the existential and universal quantifiers to specify the quantity of tuples of an entire relation that has a specific property. Conventionally, no other quantifiers have been used in set theoretic expressions [10]. This requirement that (a) only the universal and existential quantifiers be used, and (b) that they quantify an entire relation, has given rise to profound difficulties in the development of relational data base languages [18]. It is widely accepted that such data base languages be soundly based on set theoretic techniques [9, 15].

The most common problem has to do with the quantification of those tuples in relation **B** that are related to a tuple in relation **A**. To illustrate the difficulty, suppose a primitive 1:n relationship between relations **A**(A,D) and **B**(A,B,Q).

Suppose we want to express those **A** tuples where D is 4 and all related B tuples have a Q value of 6. We need the universal quantifier, but in conventional set theoretic expressions it must quantify the entire relation **B**, and not the set of tuples related to an **A** tuple:

$$[AX \in \mathbf{A}: D = 4 \ \& \ \nexists(\forall)BX \in \mathbf{B}((Q = 6 \ \& \ A = AX.A) \vee (A = AX.A))] \quad (1)$$

This results in an expression structure that is not like any that would be used in English. For this reason the universal quantifier has not been implemented in popular relational languages such as SQL. In the above expression, AX and BX are tuple variables.

Fortunately, the requirement that the quantifier quantify the entire relation in set theoretic expressions is not such an impediment to comprehension in the case of the existential quantifier. Suppose we want to express those **A** tuples where D is 4 and at least one related **B** tuple has a Q value of 6. The set theoretic expression:

$$[AX \in \mathbf{A}: D = 4 \ \& \ \nexists(>=1) \ BX \in \mathbf{B} \ (Q = 6 \ \& \ A = AX.A)] \quad (2)$$

has similarities to that used in English. The corresponding SQL expression thus does not prevent major difficulties to the average user.

The problem with the awkward universal quantifier expressions may be mitigated to some extent by using negated existential quantifier expressions. Thus expression (1) above can be replaced by the negated ($\neg$) exixtential quantifier expression:

[AX ∈ **A**: D = 4 & ¬∃(>=1) BX ∈ **B** (Q ¬= 6 & A = AX.A)] (3)

This explains why in SQL we have only the existential quantifier EXISTS (and equivalent IN [set] formulations). In SQL we must use negated existential quantifier (or equivalents) to construct universal quantifier expressions.

In SQL the negated existential quantifier expression (3) above could be expressed as either:

```
SELECT * FROM A WHERE D = 4 AND
  NOT EXISTS (SELECT * FROM B WHERE Q NOT = 6 AND B.A = A.A)    (4)
```

or

```
SELECT * FROM A WHERE D = 4 AND
  A NOT IN (SELECT A FROM B WHERE Q NOT = 6)          (5)
```

This method of quantification appears to be barely adequate. First the need for double negatives is a source of error when retrieval conditions are complex. Second, there is no way to fit in the natural quantifiers as long as the convention of quantifying only entire relations is adhered to.

The conventional set theoretic rule of quantifying only entire relations has no parallel in natural language. Natural language is flexible and permits the natural quantification of both entire sets of objects and the sets of related objects. It would

follow that a more flexible data base language would permit natural quantification of both entire relations and sets of related tuples.

It is clear that a wide variety of data base languages that permit natural quantification of related sets of tuples should be possible [2, 3] - witness the large number of natural languages. However, all these languages should be soundly based on set theoretic expressions. It follows that there is a need for a set theoretic technique for handling quantification of related sets of tuples. It is just such a set theoretic technique that is proposed in this paper. The core of the problem is defining and referencing related sets of tuples.

2.3 Related tuples

We propose 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 thee 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:

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

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 is the value of a tuple variable AX, the set of related tuples can be denoted by [AX:**R**:**B**].

3 NATURAL QUANTIFIER SET THEORETIC EXPRESSIONS

3.1 Set theoretic expressions with related sets of tuples

Once we can define a set of related tuples, and have suitable symbols for all of the quantifiers, it is possible to construct exact theoretic expressions involving natural quantification of related tuples. A quantity of a set of related tuples of **B** for which a certain condition holds can be expressed as:

$$\exists\text{(quantity) BX} \in [\text{AX:}\mathbf{R}\text{:}\mathbf{B}] \text{ (condition)}$$

This is a logical expression, and will be true only if each one of the specified quantity of related tuples satisfies the specified condition. Thus

$$\exists(\forall) \text{ BX} \in [\text{AX:}\mathbf{R}\text{:}\mathbf{B}] \text{ (Q = 6)}$$

is true only if all tuples of **B** that are related to tuple AX have Q = 6. We can incorporate such logical expressions in general set theoretic expressions as follows.

Let us begin by using this technique for the universal quantifier expression (1) earlier. We can rewrite expression (1) as:

$$[\text{AX} \in \mathbf{A}: \quad \text{D} = 4 \ \& \ \exists(\forall) \text{ BX} \in [\text{AX:}\mathbf{R}\text{:}\mathbf{B}] \text{ (Q = 6)}] \qquad (6)$$

This expression is clearly simpler than expression (1), and has a structure similar to that used in natural language. Of course, it depends on prior definitions of **R** and related sets of tuples, so that the complete expression of the retrieval is:

$$[AX \in \mathbf{A}: \; D = 4 \; \& \; \exists(\forall) \; BX \in [AX:\mathbf{R}:\mathbf{B}] \; (Q = 6)]$$

$$\mathbf{R} = \mathbf{p}_{A,B}(\mathbf{A}*\mathbf{H}*\mathbf{K} \; \ldots *\mathbf{B})$$

$$[AX:\mathbf{R}:\mathbf{B}] = [BX \in \mathbf{B}: \; \exists(>=1) \; RX \in \mathbf{R} \; (RX.B = BX.B \; \& \; RX.A = AX)] \qquad (7)$$

The set theoretic expression in (7) is thus a higher level and more powerful expression than that in (1).

Notice that there is no restriction on the type of relationship used. Expression (7) is valid for the 1:n relationship assumed with expression (1). But it will also apply no matter what relationship is involved. Expression (1) requires a primitive relationship for validity.

3.2 Natural quantifiers

The expression structure in (6) can be used with the other quantifiers. For example, the expression for those **A** tuples where D = 4 and most, or a majority, of related **B** tuples have a Q value of 6 is:

$$[AX \in \mathbf{A}: \; D = 4 \; \& \; \exists(>\forall/2) \; BX \in [AX:\mathbf{R}:\mathbf{B}] \; (Q = 6)] \qquad (8)$$

As an another example, the expression for those **A** tuples with D = 4 for which all but one related **B** tuple has a Q value of 6 is either:

[AX ∈ **A**: D = 4 & ∃(∀ - 1) BX ∈ [AX:**R**:**B**] (Q = 6)] (9)

or

[AX ∈ **A**: D = 4 & ∃(1) BX ∈ [AX:**R**:**B**] (Q ¬ = 6)] (10)

The general set theoretic expression for **A** tuples with specific attribute values and a quantity of related **B** tuples with specific attribute values is:

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

Here:

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

∃(quantifier) is any quantifier;

A-attribute-condition is any logical expression involving the attributes of **B**;

B-attribute-condition is any logical expression involving the attributes of **B**;

In a similar manner general set theoretic expressions involving any set of related relations may be formed. Unlimited expansion of the expressions is facilitated by replacing attribute-condition anywhere by

(attribute-condition & quantifier-condition)

As an example, suppose we have relations **A** and **B** with relationship relation $\mathbf{R}_{AB}$ and relations **B** and **C** with relationship relation $\mathbf{R}_{BC}$. The set theoretic expression for the **A** tuples with D = 4 with a majority of related **B** tuples with Q = 6 and with at least 3 related **C** tuples with E = 8 would be:

$$[AX \in \mathbf{A}:\ D = 4\ \&\ \exists(>V/2)\ BX \in [AX:\mathbf{R}_{AB}:\mathbf{B}]\ (Q = 6\ \&\ \exists(>=3)\ CX \in [BX:\mathbf{R}_{BC}:\mathbf{C}]\ (E = 8))] \quad (10)$$

2.6 Natural quantifier data base languages

One experimental natural quantifier data base language that can be based on the above set theoretic expression technique is SQL/NQ, which is an extension to SQL. SQL/N has been described in detail elsewhere [2,3]. The natural quantifer set theoretic expression basis for SQL/NQ can be seen from the SQL/NQ expression for expression (8) above:

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

The words is square brackets may be omitted. That there is convenience in this method of expression for non mathematical users can be seen by comparing with the SQL expression for expression (8):

```
SELECT * FROM A WHERE 4 AND
      (SELECT COUNT(*) FROM B WHERE Q = 6 AND A.A = B.A) >
      (SELECT COUNT(*) FROM B WHERE Q NOT = 6 AND A.A = B.A)
```

Note that, although the above expression gives the correct SQL formulation for dealing with the natural quantifier for most, current SQL implementations do not provide for its reduction [12, 13, 16, 17].

It should be possible to construct other natural quantifier data base languages that are soundly based on natural quantifier set theoretic expression techniques. In addition, such set theoretic techniques should help with proposals to enrich and extend SQL. Currently, SQL/NQ is the only natural quantifier extension to SQL known to the author.

2.7 Reduction of natural quantifier set theoretic expressions

Research on reduction techniques has revealed that reduction [12, 13] of SQL/NQ expressions involving 1:n relationships to relational algebra expressions is greatly facilitated by the use of a group-select operation [4]. The group select operation involves the use of a quantifier and selects each group of tuples from which a specific quantity of tuples obeys a specified condition.

A general reduction technique for natural quantifier set theoretic expressions involving any relationship has been developed. The technique uses the group select operation, and will be

presented elsewhere.

The group select operation can be regarded as introducing a natural quantifier extension to relational algebra. Likewise the set theoretic techniques presented in this paper can be regarded as introducing a natural quantifier extension to conventional set theory.

4 CONCLUSIONS

Conventional set theoretic expressions as applied to relations are limited by the convention that quantifiers quantify only entire relations and by the use of only existential and universal quantifiers. Data base languages, such as SQL , that are based on conventional set theoretic expressions consequently also suffer these limitations and present unnecessary difficulties to non mathematical users. What is worse, universal quantifier expressions cannot be used in SQL because the corresponding set theoretic expressions with entire relations have a structure that is foreign to persons not expert in mathematical logic.

The solution is the development of a set theoretic expression technique that permits quantification, by any quantifier, of both sets of related tuples, for any relationship, as well as entire relations. We have carried out such a development. Key components are relationship relations, definitions of sets of related tuples and the set of natural quantifiers. Such a set theoretic expression technique can be used as a theoretical foundation for natural quantifier data base languages and natural quantifier extensions to SQL.

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., 14(1), 1988, 126-29.

5. Bradley, J. Polygonal join dependencies, closed co-relationship chains and the connection trap in in relational data bases. Research Report No 87/1273/21, University of Calgary, Calgary, Alberta, 1987.

6. Bradley, J. Co-relationships, levels of significance, and the source of the connection trap in relational data bases, Computer Journal, in press.

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.