Profile Functions And Bag Theory

William Kent
Database Technology Department
Hewlett-Packard Laboratories
Palo Alto, California

[August 18, 1992 5:02 pm]
profile.fmk

1 Introduction

To duplicate or not to duplicate, that is the question — which we shall avoid. We explore a possible theoretical foundation for relational databases with duplicates, comparable to the foundation that set theory provides for traditional relational theory. We deliberately sidestep the controversy over the legitimacy of duplicates [Co], and merely take the pragmatic view that such a theoretical foundation would be useful to rationalize the many real implementations of relational database which admit duplicates.

To put it simply, we are thinking about duplicates because they're there. If you're looking for an argument about duplicates, this is not the place.

Our approach is based on the use of numeric-valued profile functions in place of logical predicates, indicating "how many times" something is true rather than simply whether or not it is true. In [K1,K2] we described adaptations of relational operators to such a foundation. In the present paper, we explore ramifications for the underlying logical foundations, shifting from a logic-oriented set theory to an arithmetic-oriented bag theory.

2 Profile Functions And Bags

A *profile function* (or *profile*, for short), takes a single argument and returns a non-negative integer or the infinity object ∞. The non-negative integers together with ∞ are *counters*. The argument of a profile may be any arbitrary object, including an aggregate such as a bag or a list.

A *bag* is an object defined by an infix *Membership* function $\in$ such that $x \in b$ is interpreted as the number of occurrences of x in b. Every profile p has a corresponding bag b such that $p(x)=x \in b$: the value of p(x) is the number of times that x occurs in b. We say that b is the *extension* of p, and p is the *characteristic function* of b:

$p(x) = x \in b$

b = Extension(p)

p = Characteristic(b).

It suffices for our present purposes to define a *relation* as a bag of tuples (lists) each containing the same number of elements.

This all maps nicely into set theory and logic in the absence of duplicates. A set is a bag b subject to the constraint that $x \in b < 2$ for all x. A profile p such that $p(x) < 2$ for all x behaves like a logical predicate, if we accept that 1 and 0 correspond to True and False. The extension of a predicate is a set.

We can achieve some economy and elegance by letting a bag itself *be* a profile function, such that

Extension(b) = b

Characteristic(b) = b

$b(x) = x \in b$.

If some profile functions are updatable objects, as in the Iris object-oriented database system [Fi], then the converse doesn't always hold. Not all profiles are bags. Many profile functions might have the same bag as their extension, and update to a profile function amounts to changing its extension to be a different bag.

Under these conditions, the characteristic of a bag is not uniquely determined; however, in the absence of update all the characteristics of a bag behave the same, and we will proceed as though each bag had one "canonical" characteristic function.

3 Bag Operators

We are looking for bag-theoretic counterparts to the operators of set theory and logic.

Logical operators are closed on {True, False}. By extension, a *bag operator* is closed on counter objects, i.e., it returns a counter when its arguments are counters. For example, addition, suitably extended to cover ∞, is a bag operator which corresponds to concatenation of bags:

$$\text{Extension}(p_1+p_2) = \text{Extension}(p_1) \cup \text{Extension}(p_2).$$

(We write $\cup$ for bag union, i.e., concatenation.) Thus, if b_1 and b_2 are bags with characteristics p_1 and p_2, their bag union $b_3=b_1 \cup b_2$ is characterized by $p_3=p_1+p_2$. Conversely, the extension of p_1+p_2 is the concatenation of the extensions of p_1 and p_2.

We refer to such operators as bag operators because in general, if

$$p_0(x) = p_1(x) \text{ bag-op } p_2(x),$$

we can take this to define the contents of the extension of p_0 given the contents of the extensions of p_1 and p_2. That is, given the number of occurrences of some object in the extensions of p_1 and p_2, this expression defines the number of occurrences of that object in the extension of p_0.

Calling them bag operators is even more natural when we take the bags themselves to be profile functions, so that we have

$$b_3(x) = b_1(x) \text{ bag-op } b_2(x).$$

We define another interesting bag operator, the unary prefix operator δ, which returns 0 when its argument is 0 and 1 otherwise. It has the effect of converting a profile to a predicate, since it converts the corresponding extension to a set. That is, δp(x) is always 0 or 1, and if q(x) ::= δp(x), then the extension of q is a set.

We label this function "δ" to suggest "distinct".

4 Pseudo-logical Operator Classes

In looking for bag-theoretic counterparts to logical operators, it is of paramount importance that we preserve a subset that is isomorphic to traditional set theory. That is, we want operators that continue to behave in familiar ways when the bags happen to be sets and the profiles happen to be predicates.

A *pseudo-logical operator* is a bag operator which is closed on {0,1}. Multiplication and maximum are examples; if both arguments are 0 or 1, then their results are 0 or 1. δ is another example, being the identity function on 0 and 1.

We define *classes* of pseudo-logical operators according to the logical operator whose behavior they mimic when the arguments are only 0 or 1. For example, the *And-class* of bag operators contains all binary bag operators which return 1 when both arguments are 1, and 0 when the arguments are 0,0 or 1,0 or 0,1:

the AND class:

x	0 0 1 1 0 2 2 3 …
y	0 1 0 1 2 0 3 2 …
And	0 0 0 1 ? ? ? ? …
*	0 0 0 1 0 0 6 6 …
Min	0 0 0 1 0 0 2 2 …
δ*	0 0 0 1 0 0 1 1 …
and others ...	

Multiplication and minimum are both And-class operators. We could think of δ* ("distinct multiplication") as another one; it returns 0 if either argument is 0, and 1 otherwise.

Maximum is an *Or-class* operator (try it on 0 and 1). We could also think of δ+ as an Or-class operator; it returns 0 if both arguments are 0, and 1 otherwise.

the OR class:

x	0	0	1	1	0	2	2	3	…
y	0	1	0	1	2	0	3	2	…
Or	0	1	1	1	?	?	?	?	…
Max	0	1	1	1	2	2	3	3	…
δ+	0	1	1	1	1	1	1	1	…
and others ...									

It's important to notice that addition is not an Or-class operator, since 1+1=2. This is consistent with our interpretation of + as bag concatenation, which does not eliminate duplicates from the result. However, if bag concatation were mis-labelled as "union", and one recalled that union corresponds to Or, then one might implement Or as bag-concatenation (masquerading as union), giving Or the additive behavior.

We introduce a Negation-class unary operator ¬, which returns 1 if its argument is 0 and 0 otherwise. We observe that ¬¬x=x does *not* hold for x>1.

the NEGATION class:

x	0	1	2	3	…
Not	1	0	?	?	…
¬	1	0	0	0	…
and others ...					

We introduce a Difference-class operator Θ, defined as

$$x \Theta y ::= \max(0, x - y),$$

i.e., non-negative subtraction. It corresponds naturally to set or bag difference. We may observe that ¬y=1Θy.

With that definition, we might note that (x+y)Θ(x*y) is an Or-class operator.

We can observe that ≤ is an Implication-class operator, if we interpret it as returning 0 when false and 1 when true. That is, x≤y is 0 when x is 0 and y is 1, and it is 1 when x,y are 1,1 or 0,1 or 0,0. Thus, if x≤y, then y must be true (non-zero) if x is true (non-zero).

A weaker form of implication is one that would be satisfied so long as y is non-zero whenever x is non-zero, even if x>y. This could be formulated as δx≤y.

5 Interesting Operators

Our goal is to choose one operator definition from each class to be the bag-theoretic counterpart of such important operators as And, Or, Not, etc. Unfortunately, there does not seem to be an obviously superior set of candidates. It will take further investigation to determine an appropriate set of operators.

But there are positive results. To begin with, we recognize that all such operators can be well-defined in terms of arithmetic. Thus the behavior of query languages offering such operators can be rigorously defined. It may be appropriate for such languages to offer a variety of operators, e.g., And_1, And_2, Or_1, Or_2... (or bearing more mnemonic names),

with the user choosing the one whose behavior matches the semantics appropriate to his application. Furthermore, so long as each operator is chosen from its corresponding operator class, the query language will be well-behaved in the absence of duplicates.

Beyond that, we can begin to investigate relevant criteria for choosing such operator definitions. What would it take to make certain definitions interesting?

Intuition:

My intuition suggests:

p & q ::= min(p,q)
p | q ::= p + q

but the latter isn't even an Or-class operation.

Extend set/predicate correspondences to bags/profiles:

$p \cup q ::= p \mid q$
$p \cap q ::= p \,\&\, q$
$p \subseteq q ::= p \rightarrow q$

Symmetry:

Another intuitive feel is that pairs in the following rows ought to be preserved. Thus | and & should either be defined as + and *, + and –, or min and max. Defining one as min and one as + seems like an unnatural asymmetry.

$\cup$ $\cap$
| &
+ *
+ –
max min

Logical behavior:

It would certainly be desirable to preserve logical axioms and tautologies:

$p|\neg p = 1$ (excluded middle),
$p|p = p$,
$p\&p = p$,
$\neg\neg p = p$ (double negation),
$p|q = \neg(\neg p\&\neg q)$ (DeMorgan),
$p\rightarrow q = \neg q|p$,
etc.

It may be demonstrable that those can't all be satisfied for p>1.

Current implementations:

Being somewhat crass, correspondence to current query languages and their implementations would also be of interest, suggesting & as cross-product, | as bag concatenation:

p & q ::= p * q
p | q ::= p + q

Efficiency:

Still being crass, interest in one definition or another might even be contingent on efficiency of implementation and execution. With current technology, that weighs against elimination of duplicates, in favor of treating & as * (via cross-product) and | as + (via concatenation, i.e., bag union).

On the whole, there seems to be no single set of definitions for the logical and set-theoretic operators which satisfies all relevant criteria. Perhaps the most serious concern is that the logical behaviors described above cannot be preserved except for operators that always force their results to 1 or 0, e.g., an And operation that returns 0 if either operand is 0 and 1 otherwise. Such operators destroy the character of bags: unions and intersections contain no duplicates.

Negation continues to be a particularly sticky point, as it has been all along in database query languages. For database, since our underlying metaphor is a count of tuples in a relation, the only numeric domain that seems relevant is the non-negative integers. It's difficult to find a useful negation operator on this domain for which $\neg\neg p=p$ when $p>1$.

If our underlying model was based on counting the net of assertions and retractions, then it would be appropriate to allow negative values, and negation might be well behaved. But we don't know what to make of a negative count of occurrences of tuples in a relation.

Fuzzy set theory is more tractable, since its numeric domain is the interval [0,1], over which the expression $1-x$ is closed. However, once again, we don't know what to make of fractional occurrences of tuples in relations.

If the logical properties no longer hold, then certain query transformations may no longer be valid, and certain inferencing capabilities may not exist. On the other hand, putting query semantics on an arithmetic foundation opens up new proof approaches. The validity of a query transform can be established by demonstrating that the queries before and after transformation would return the same numeric value. An implication $p \rightarrow q$ can be established by demonstrating $p \leq q$. And so on.

6 Alternatives: Sets Of Ordered Pairs

An alternative treatment of duplicates within the framework of traditional set theory is via sets of ordered pairs. If x occurs k times, this is represented by the pair <x,k>.

There are some drawbacks to this approach. To begin with, we cannot admit arbitrary sets of this form. There has to be a sort of "functional dependency" such that a given x appears in only one such tuple. We wouldn't know how to interpret a set of the form {<x,i>,<x,j>}.

Then, how should absent elements be dealt with? Should there be no corresponding tuple in the set, or should there be a tuple of the form {<x,0>}? To avoid all such sets being uncountably large, we might require i>0 in such sets, omitting members of the form <x,0>; however, this will soon lead to other difficulties.

More seriously, this approach still does not let us exploit the traditional set theoretic operators, since

$\{<x,2>\} \cap \{<x,3>\} = \emptyset$,
$\{<x,2>\} \cap \{<x,2>\} = \{<x,2>\}$,
$\{<x,2>\} \cup \{<x,3>\} = \{<x,2>, <x,3>\}$,
$\{<x,2>\} \cup \{<x,2>\} = \{<x,2>\}$.

These do not correspond to what we might want for bag behavior, such as

$\{<x,2>\} \sqcap \{<x,3>\} = \{<x,2>\}$,
$\{<x,2>\} \sqcap \{<x,2>\} = \{<x,2>\}$,
$\{<x,2>\} \sqcup \{<x,3>\} = \{<x,5>\}$,
$\{<x,2>\} \sqcup \{<x,2>\} = \{<x,4>\}$.

We still have to define functions for the arithmetic behavior of "bag intersection" $\sqcap$ and "bag union" $\sqcup$ (concatenation) such that

$\{<x,i>\} \cap \{<x,j>\} = \{<x,f(i,j)>\}$,
$\{<x,i>\} \cup \{<x,j>\} = \{<x,g(i,j)>\}$,

Furthermore, we have a dilemma with absent members. If we exclude <x,0> pairs, we have no way to capture them in a union or intersection expression of the form f(i,j); they just aren't present to occur as arguments. That is, how do we express

$\{<c,2>\} \cap \{\ \}$

in terms of f(i,j)? Where is j?

On the other hand, if we include the <x,0> pairs, then all sets become uncountably large.

Cardinality of a bag doesn't correspond to the cardinality of such a set. It must be re-defined as a summation of the i's in all the <x,i>.

And finally, a minor point. This treatment doesn't reduce naturally to sets as a special case. If the bag is constrained to i<2, and we choose to omit absent members, the bag reduces to a set of the form {<x,1>} which, though isomorphic, is not quite the same as a set of the form {x}.

Ordered pairs can also be used in a different way to simulate bags. Instead of simulating a bag containing k occurrences of x by an element of the form <x,k>, we could have k elements of the form <x,1>, <x,2>, .., <x,k>. This simulates the correct number of actual occurrences of x; the i's in the <x,i> pairs serve as "discriminators", making apparent duplicates be distinct elements of the set. Many similar problems still remain, e.g., we still don't get the right results from set-theoretic union.

Thus, on the whole, representing duplicates as <x,i> pairs does not seem to be a fruitful approach.

7 Conclusions

Our quest for a bag theory that accommodates duplicates has met with mixed results.

There is no obviously good definition for the behavior of logical operators in the presence of duplicates. It may not be possible to satisfy all the decision criteria. It is useful, though, to have identified what the decision criteria are.

We could just walk away from the problem, and let the implementations define the semantics of those databases and query languages which admit duplicates. But it would be sounder to try to converge on some meaningful semantic model.

We do have a sound basis for database query semantics, founded on arithmetic interpretations of operators. The semantics are clear and results are predictable. Whether they are satisfying depends on what arithmetic behaviors we assign to familiar operator keywords such as And and Or, and how we feel about such behaviors. We won't be able to satisfy everybody, and might even have to admit variants of the basic keywords.

Thus there is a coherent theory, but it doesn't have quite the elegance of set theory. We don't provide fixed definitions of the logical operators, and we do not have the same deductive behavior as predicate logic when duplicates are present.

On the other hand, the bag operators are defined by arithmetic, which should provide a sound foundation for reasoning about their behavior. This may suggest new arithmetic approaches to predicting the results of queries, and to the validation of query transforms. Furthermore, the approaches suggested here are "safe", in that the classical behaviors of logic and set theory are preserved in the absence of duplicates.

8 References

[Co] Codd, E.F., "Fatal Flaws in SQL", Datamation, Aug. 15, 1988.

[Fi] Fishman, D.H. et al, "Iris: An Object-Oriented Database Management System", ACM Transactions on Office Information Systems, Volume 5 Number 1, January 1987.

[K1] Kent, W., “A Generalized Select Function”, HPL-SAL-89-17, Hewlett-Packard Laboratories, Jan 3 1989.

[K2] Kent, W., “Shadow Relations and Function Families”, HPL-SAL-89-18, Hewlett-Packard Laboratories, Jan 10 1989.