Chapter 9

# SEMANTIC ENHANCEMENT THROUGH EXTENDED RELATIONAL VIEWS

Eugene Wong

Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California 94720, USA

In this paper two ideas on relational databases are presented and combined; First, it is shown that the concept of "views" can be usefully extended to include virtual operations in addition to virtual relations, and that both can be easily supported through query modification. Second, it is suggested that a simple yet powerful way of semntically extending a relational query language (such as QUEL) is by augmenting the supported operations at the domain level. These ideas are then developed and illustrated in two important applications: geometric data and lexical processing.

## 1. INTRODUCTION

This paper results from a confluence of two streams of ideas concerning relational databases: extending "views" to include virtual operators as well as virtual relations, and enhancing the relational model with application specific semantics such as those found in geometric and lexical data by adding new data types for the domains.

One of the strengths of the relational data model is the elegant and efficient way in which diverse user views can be supported. What makes this possible is the "closure" property of relational operators Briefly stated the situation is as follows: Consider a set of view relations V=(V1,V2,...,Vm) defined by a transformation f on a set of base relations B=(B1,B2,...,Bn). i.e.

$$V=f(B)$$

Then, a mapping of the view relations can be written as

$$g(V)=g*f(B)$$

---

* The work reported in this paper was supported by the National Science Foundation Grant ECS-8007684

when g*f is the composition of g and f defined by

$$(g*f)(x)=g(f(x))$$

It follows that any function of the view relations is a function of the base relations by virtue of the closure property of functions under composition. The practical signficance of this fact is that an operation g on a view relation V can be evaluated be evaluating g*f on the base relations and this process is known as "query modification" [6].

Powerful as it is, the concept of "relational views" thus defined is not as general as it can usefully be. A "base-dependent" operation on view relations. i.e.

$$h(B,V)=h(B,f(B))=g(B)$$

can also be evaluated by "query modification".

As an example, consider a base relation

emp (ename, dept, salary)

and a view relation

salesmen = emp(dept = "sales")

where () denotes restriction. Now, consider a query: "find the names of all salesmen who earn more than the companywide average salary." As a query on the view relation "salesmen" it can be expressed in QUEL syntax as [HELD]

```
range of s is salesmen
retrieve (s.ename) where s.salary > {average}
```

where both s and {average} need to be interpreted. Query modification then yields the QUEL query:

```
range of e is emp
retrieve (e.name) where e.dept = "sales"
                  and e salary > avg(e.salary)
```

A natural extension to the concept of "views" that would accommodate such situations is to include "view" operators" as well as "view relations." While in terms of the base relations the view operators are ordinary relational operators, perceived in terms of the view-relations alone, they transcend the boundaries of the class of relational operators. For a single user-view, the effect is a semantic enhancement, and one that can be achieved with a minimal augmentation to existing systems.

Since it was first proposed [1] "relational completeness" has become the standard by which the semantic power of a relation query language is measured. As such a measure it is imperfect, for it fails to

consider explicitly the operations that are defined at the domain level. For example, QUEL [2] derived a great deal of its semantic power from operations on numerical domains. These include arithmetical operators such as multiplication and addition, aggregational operators such as sum and average, and comparison operators such as = and <. The way QUEL embeds domain-level operators provides a general framework for semantic extension. For each new domain type that is introduced, its domain level operators are automatically propagated throughout QUEL.

Enhancing the semantics of a language through the use of enriched data types is hardly new. The use of "abstract data types" [4] is precisely such an approach. Yet, what we are proposing is different. Rather than adding a general facility for handling abstract data types. We accept the structure of QUEL as it is, and seek to extend its semantics through its existing machinery. In so doing, we are motivated by the consideration that QUEL is already in widespread use, and whatever now exists should be left undisturbed.

The objective of this paper is to combine "semantic enhancement" and "extended views" by making the domain-level operators of new data types virtual operators supported by "views". In the process, a general machinery for semantic enhancement will be made available with only minor augmentations to the existing facilities in relational systems for supporting "views".

The organization of this paper is as follows: first, we consider how domain level operators are embedded in QUEL. Next, geometric data types and operators appropriate to them are introduced. Representation of geometric operators as "view" operators is then illustrated. As a second example of semantic enhancement through domain level operations, data types appropriate to "text" are introduced. The power of the semantic enhancement so achieved is then illustrated by expressing some sophisticated lexical processing tasks in the enhanced version of QUEL.

## 2. DOMAIN LEVEL OPERATIONS IN QUEL

As in any relational query language worthy of the name, the primitives of QUEL are relation-at-a-time operations. How, then, can domain-level operations be incorporated in such a language? This is done in QUEL by introducing the construct "computing a new column" which adds a new unary relational operator not present in the "relational algebra" as originally defined in [1]. Since new columns are computed using arithmetical and aggregational operators, these are then propagated through the language via concatenation of relational operators.

Comparison operators are embedded in the relational algebra via the selection condition of the operator "restriction." In QUEL the selection condition is known as the "qualification" clause of a query.

Since columns (i.e. attributes) of relations are explicitly manipulated in QUEL, a compact notation for them is needed. In QUEL

columns are denoted with the use of range variables. For example, suppose that employee (eno, ename, byr, dept, salary) is a relation. The declaration

```
range of e is employee
```

defines e as a variable that ranges over employee and e.salary denotes the salary column.

New columns can be constructed in QUEL in two ways:

a) Through arithmetical operators -- For example,

```
e.salary/(1982-e.byr)
```

defines a new column on employee.

b) Through aggregational operations -- For example,

```
avg(e.salary by e.dept)
```

defines a new column that gives for each employee the average salary of her department.

In addition, aggregational operations can be qualified by a selection condition, e.g.,

```
avg(e.salary by e.dept where e.byr < 1930)
```

A newly constructed column can be used like any other column. Specifically, it can take part in arithmetical operations, in aggregations, and in qualifications. These, in turn, can be used to produce new columns. Thus, nesting can occur in a number of ways as in the following examples:

```
retrieve (e.dept, rate = avg(e.salary/(1982-e.byr) by e.dept))

retrieve (e.dept, var = avg((e.salary-avg(e.salary by e.dept))**2
                            by e.dept))

retrieve (number = countu(e.dept where
                  avg(e.salary by e.dept ) > 25000))
```

In summary, domain level operations are absorbed into QUEL in two ways. First, arithmetical and aggregational operators are embedded through the construction of new columns. Second, comparison operators take part in qualifications. Nesting to any level is allowed.

## 3. A GEOMETRIC EXTENSION TO QUEL

Database applications that involve geometric operations are both common and important [3,7]. As our first example of semantic extension, we consider the case of geometric data. For the primitive objects of the geometric data types, we propose the following:

a) atomic objects : point, line (finite oriented line segment)

b) composites: point-group (finite collection of points) line-group(finite collection of lines)

c) constrained composites:

path = ordered line group {L1,L2,...,LN} such that
start(Lk+1) = end (Lk),k=1,.... N-1

polygon = path such that start(L1) = end(LN)

We distinguish an object from its representation in terms of other objects. For example, a line is uniquely represented by an ordered pair of points, but a line and its endpoints are different geometric objects. For clarity, we shall use the term "type" to denote one of the six possibilities: point, line, point-group, line-group, path and polygon. Collectively, they will be known as the "geometric data types."

Geometric objects are rich in the operations that accept. Indeed, in the modern era the very term "geometry" has come to mean the study of transformations. The following is a list of some familiar operations, but the list is not intended to be complete in any way:

Unary and Type-Preserving
rigid body motions - translation and rotation
isometries - rigid body motions plus reflection
isomorphic operators - isometries plus scaling

Unary, Not Type-Preserving
connect: point-group-> path
close: point-group -> polygon
vertices: path or polygon -> point-group

Binary
intersection: (path, path) -> point-group
common part: (path, path) -> line-group
border: (polygon, polygon) -> line-group
overlap: (polygon, polygon) -> polygon

Metric
length (line-group)
count (point-group)
area (polygon)

Comparison
equality
congruence
similarity
set inclusion
intersect
enclose
pass-thru

An aggregation is an operator on a set. The existing aggregation operators in QUEL act on sets of values from the databases. It is useful to generalize these operations to act on sets defined mathematically rather than by data. For example, "shortest" is an operator on a set of paths, and the operator "connect" can be expressed as:

```
connect (point1, point2)
= shortest ({path: start(path) = point1 and end(path) = point2})
```

As in the case of existing aggregations, an interesting property of set operators is that they are subject to qualifications. For example, to find the shortest "route" often means finding not the shortest among all paths connecting two points but the shortest among those that satisfy some additional conditions. e.g.,

```
shortest ({path: start(path) = point1 and end(path) = point2
  and path pass-thru point3})

shortest ({path: start(path) = point1 and end(path) = point2
  and polygon3 enclose path})

shortest ({path: start(path) = point1 and end(path) = point2
  and path3 not intersect path})
```

It is not hard to make up examples of set operations in addition to "shortest." The following examples come readily to mind:

```
longest - on set of paths
smallest - on set of polygons
longest - on set of polygons
closest to "a" - on set of points
straightest - on set of paths
```

The set on which these operate can be defined using qualification as in the case of aggregations in the existing QUEL.

## 4. GEOMETRIC OPERATIONS IN VIEW OPERATORS

Suppose that we begin by viewing all objects of geometric data types as entities, and seek to represent all information concerning them as properties of the entities and interrelationships among them. For example, each point is an entity and its x and y coordinates are two of its properties. An oriented line segment is also an entity, with beginning-point and end-point as two functions. One consequence of viewing points and lines this way is that it immediately suggests that points and lines can be represented as ordinary INGRES relations:

```
point (pid, xcoord, ycoord)
line (lid, pt1-pid, pt2-pid)
```

where all domains are of numerical type.

Paths and polygons can be expressed uniquely in terms of ordered sets of points. One way of representing them is to combine them in a single relation

```
pointgroup (pgid, type, order, pid)
```

where "type" can be either "polygon" or "path" and "order" indicates the sequence of the points in the pointgroup.

Now, consider a relation that has a geometric domain, e.g.,

```
city (cname, nation, location)
```

where cname and nation are of type "character string" and location is of type "point." Such a relation can be represented as a view on base relations that contain only ordinary domains (i.e., numbers or character strings), e.g.,

```
cityb (cname, nation, location-pid)
point (pid, xcoord, ycoord)
```

Geometric operations on the domain "location" are then expressed as operations on the base relations. For example, consider the query:

```
range of c is city
range of c1 is city
retrieve (c.name) where c1.name = "Chicago"
and distance(c.location, c1.location) < 500
```

If distance is interpreted as euclidean distance, then this query can be mapped into a query in ordinary QUEL as follows:

```
range of c is cityb
range of c1 is cityb
range of p is point
range of p1 is point
retrieve (c.name) where c1.name="Chicago"
and p.pid=c.location and p1.Did=c1.location
and sqrt((p.xcoord-p1.xcoord)**2+(p.ycoord-p1.ycoord)**2) < 500
```

In principle, all geometric operators that we have introduced can be re-expressed in terms of the coordinates. However, the operators on numerical domains that INGRES currently supports are mainly arithmetical and will have to be extended. The difficulty in doing this is largely a matter of obtaining good performance.

The use of "query-modification" to support extended data types is very much in the spirit of views, but the current "views" facility of INGRES (or of any other relational system) is inadequate to support it. The big difference is that instead of view-relations, we now have view-domains and view-operators on such domains. The class of operators that can be supported through query-modification is of great interest, since they can be implemented with only minimum changes to the existing INGRES.

## 5. EXTENDING QUEL TO SUPPORT TEXT

Some of the ideas in database management have their origin in automatic text searching. For example, secondary indexing and query language were both used in information retrieval long before database management existed as a technical discipline. Thus, it is ironic to note that the existing database management systems all handle text badly. Fundamentally, the problem is that text as data has a semantic depth far exceeding anything that is recognized in existing systems.

We take "words" as the semantic atoms of text. The semantic components of text are lexical in nature, and they form a natural hierarchy as follows:

words
word-sequences
clause
nested sequences
sentences
text

A word-sequence is an ordered set of words. A clause is a word-sequence that ends in a punctuation. A nested sequence is an ordered set of clauses, and a sentence is a nested sequence that satisfies a special constraint. A text is a ordered set of sentences. Collectively, these will be referred to as the lexical data types.

The following list contains some examples of natural operations on lexical objects:

type-preserving
concatenate : (word-sequence, word-sequence) -> word-sequence
combine : (nested-sequence, nested-sequence) -> nested-sequence
root : word ->word
synonym : word -> word

non-type-preserving
punctuate : (word-sequence, "symbol") -> clause

metric
length (word)
count (word-sequence)

comparison
contain : word-sequence vs. word-sequence
match : word-sequence vs. word
root-equal : word vs. word
equal : word vs. word

## 6. LEXICAL PROCESSING USING QUEL

The hierarchical nature of the lexical data types makes them easy to support in the framework of the existing QUEL. Stripped of the information used only for display and formatting purposes, a text consists of word occurrences and punctuation symbols structured as follows:

```
test = {sentence}
sentence = {clause}
clause = {word} symbol
```

Define sno as the numerical order of a sentence within a given text, cno as the numerical order of a clause within a given sentence, and wno as the numerical order of a word within in a clause. Each sentence in a text is uniquely identified by its sno, each clause by (sno, cno), and each word occurrence by (sno, cno, wno). A text, then, can be represented by two INGRES relations:

```
text (sno, cno, wno, word)
punctuation (sno, cno, symbol)
```

The operations in the existing QUEL can be used not only for searches but also for lexical operations that are statistical in nature. For example, the following QUEL statement can be used to produce a histogram of words:

```
range of t is text
retrieve into hist (t.word. freq = count (t.wno by t.word))
```

To find sentences in which the word "data" occurs more than once, we can write

```
retrieve (t.sno) where count (t.wno by t.sno where t.word = "data") >1
```

Although these examples can he done in INGRES now, storage efficiency may be poor because the domain "word" in the relation "test" will be required to have a fixed length. Changing INGRES to support variable-width domains will be necessary to support lexical data types efficiently. Another possible source of efficiency gain is the ability to maintain ordering in a relation (the "ordered relation" [5]). It may allow the prefix (sno, cno, wno) to be eliminated or drastically compressed.

## 7. CONCLUSION

In this paper two ideas are presented: first, the concept of "relational views" can be usefully extented to operations in addition to relations; second, an easy yet powerful way of extending the semantics of a relational query language is to augment the domain-level operators. These two ideas are then combined and illustrated in two important cases: geometric data and lexical processing.

## REFERENCES

[1] E. F. Codd. "Relational Completeness of Data Base Sub-languages." In Data base Systems, Courant Computer Science Symposia Series, vol. 6, Prentice Hall. 1972.

[2] G. D. Held, M. R. Stonebraker and E. Wong. "INGRES - A Relational Data Base Systems." Proc. NCC 1975.

[3] G. Nagy and S Wagle. "Geographic Data Processing." Computing Surveys, vol. 11, 1979.

[4] L. A. Rowe. "Data Abstraction from a Programming Language Viewpoint." Proceedings ACM Workshop on Data Abstraction, Databases and Conceptual Modelling, 1980.

[5] M. R. Stonebraker and J. Kalash. "Timber: A Sophisticated Relation Browser." Proc. 8th VLDB Conference, 1982.

[6] M. R. Stonebraker. "Implementation of Integrity Constraints and Views by Query Modification." Proc. ACM-SIGMOD International Conference on Management of Data, 1975.

[7] M. Tamminen. "Efficient Spatial Access to a Data Base." Proc. ACM-SIGMOD International Conference on Management of Data, 1982.