EXQUEL: A SEMANTIC EXTENSION TO QUEL

by

E. Wong

Memorandum No. UCB/ERL M82/44

17 May 1982

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

EXQUEL: A SEMANTIC EXTENSION TO QUEL

E. Wong

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, California 94720

ABSTRACT

One of the most natural ways of extending the semantics of a relational query language is to enhance the data types used in the domains. This is especially true of QUEL, which makes extensive use of domain-level operators. In this paper we exploit this approach in the specific cases of geometric and lexical data.

Research sponsored by the National Science Foundation Grant ECS-8007683.

1. INTRODUCTION

Since it was first proposed [CODD 72], "relational completeness" has become the standard by which the semantic power of a relational 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 [HELD 75] derives 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. The purpose of this paper is to elucidate this framework, and, using it, to extend QUEL to handle geometric and lexical data.

Enhancing the semantics of a language through the use of enriched data types is hardly new. The use of "abstract data types" [ROWE 80] 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.

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

EXQUEL: A SEMANTIC EXTENSION TO QUEL

E. Wong

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, California 94720

ABSTRACT

One of the most natural ways of extending the semantics of a relational query language is to enhance the data types used in the domains. This is especially true of QUEL, which makes extensive use of domain-level operators. In this paper we exploit this approach in the specific cases of geometric and lexical data.

Research sponsored by the National Science Foundation Grant ECS-8007683.

new many relational operator not present in the "relational algebra" as originally defined in [CODD.72]. 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

As 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 $\{\ell_1,\ell_2,\ldots,\ell_N\}$ such that $start(\ell_{k+1}) = end(\ell_k)$, $k = 1,\ldots,N-1$

polygon = path such that $start(\ell_1) = end(\ell_N)$

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 they 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 database. 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 (point 1,point 2)
  = shortest ({path: start(path) = point 1 and end(path) = point 2})
```

As in the case of existing aggregations, the most interesting thing about 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 condition, e.g.,

```
shortest ({path: start(path) = pointl and end(path) = point2
          and path pass-thru point3})
shortest ({path: start(path) = pointl and end(path) = point2
          and polygon3 enclose path})
shortest ({path: start(path) = pointl 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
largest  - 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. AN ERSATZ IMPLEMENTATION OF GEOMETRIC DATA TYPES

One way of extending QUEL to include new data types is to represent objects of new types in terms of existing ones. This may not be the best way in terms of efficiency, but has the advantage of requiring the least amount of changes to INGRES.

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.

Such a representation of geometric data objects is suggestive of "views." For example, consider a relation

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

in extended INGRES where cname and nation are of type "character string" and location is of type "point." It can be represented as a pair of ordinary INGRES relations:

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

Now, consider the following query in extended QUEL with an obvious interpretation:

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

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 a minimum change 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 view "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 an ordered set of sentences. Collectively, these will be referred to as the lexical data types.

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

type-preserving

concatenate : (word-sequence,word-sequence) ⟶ word-sequence
combine : (nested sequence,nest 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:

```
text = {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 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 be 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" [STON 82]). It may allow the prefix (sno,cno,wno) to be eliminated or drastically compressed.

7. CONCLUSION

The purpose of this paper is demonstrate that domain-level operations can be incorporated in QUEL using the same vehicle that supports numerical operations. In so doing, we can achieve a great deal of semantic enhancement with a minimum of modification to QUEL. This approach is illustrated in the specific cases of geometric and lexical data. The possibility of implementing such extensions by extending the existing "views" facility of INGRES is also briefly explored.

REFERENCES

[CODD 72] E. F. Codd. "Relational Completeness of Data Base Sub-languages." In Data Base Systems, Courant Computer Science Symposia Series, vol. 6, Prentice Hall, 1972.

[HELD 75] G. D. Held, M. R. Stonbraker and E. Wong. "INGRES - A Relational Data Base System." Proc. NCC 44, 1975.

[ROWE 80] L. A. Rowe. "Data Abstraction from a Programming Language Viewpoint." Proceedings ACM Workshop on Data Abstraction, Databases and Conceptual Modelling, 1980.

[STON 82] M. R. Stonebraker and J. Kalash. "TIMBER: A Sophisticated Relation Browser." Proc. 8th VLDB Conference, 1982.