A Comparison of Abstract Data Type and Constraint Database Approaches to GIS Query Languages

Peter Revesz
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, Nebraska 68588, USA
revesz@cse.unl.edu

ABSTRACT
Designing query languages for geographic information systems using the traditional approach of constantly adding new data types and operations on the new data types has reached a limit beyond which the query language is no longer easy to understand or convenient to use. We advocate instead the design of query languages based on constraint databases. We show that many formerly difficult-looking queries, such as the shortest path query, can be expressed using simple SQL-like queries of constraint databases.

Categories and Subject Descriptors
H.2.8 [Database Management]: Database Applications—Spatial databases and GIS

Keywords
abstract data types, constraint databases, query language

1. INTRODUCTION
Most current work on designing query languages for GISs tries to extend relational database query languages with new operators [5, 17, 18]. At first sight the additional operators look attractive, but the problem is that they are inflexible. The additional operators often provide simple solutions to toy problems presented in many research articles and textbooks. However, when a more complex real-life problem arises, then the problem is often inconvenient or impossible to express.

The typical solution to the inherent inflexibility is to introduce more operators. Unfortunately, carrying this process too far results in rather difficult-to-understand and unwieldy query languages. For example, the International Committee for Information Technology Standards described a 166 pages long standard for spatial data operators (ISO # 19107) and 49 pages for a limited set of moving objects (ISO #19141).

A fundamental problem is that current GIS users are not given direct access to the defined spatial and spatio-temporal data types as a set of points \( \{ (x, y) \} \) in the real plane \( \mathbb{R}^2 \) or \( \{ (x, y, t) \} \) in real space \( \mathbb{R}^3 \). In contrast, constraint query languages (Revesz [13], Rigaux et al. [15] Chap. 4, and Gütting and Schneider [8] Chap. 6), although implemented only in prototype systems, such as MLPQ [14], allow the users access to the \( x \) and \( y \) coordinates of the objects. As we describe below, this enables expressing shortest path and other complex queries in a much simpler, SQL-like way without the need for additional abstract data types and operators.

2. GEOGRAPHIC DATA MODELS
We review some geographic data models from [2, 3, 8, 10, 13, 15] without trying to be comprehensive.

Vector Data Model [1, 4, 16], used in the ARC/INFO system [11, 12], can describe streets and towns as follows.

<table>
<thead>
<tr>
<th>Street</th>
<th>Type</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bear</td>
<td>polyline</td>
<td>(2, 11), (17, 16)</td>
</tr>
<tr>
<td>Hare</td>
<td>polyline</td>
<td>(4.5, 20), (9, 11)</td>
</tr>
<tr>
<td>Maple</td>
<td>polyline</td>
<td>(7, 5), (20, 5)</td>
</tr>
<tr>
<td>Oak</td>
<td>polyline</td>
<td>(11, 2), (11, 12)</td>
</tr>
<tr>
<td>Vine</td>
<td>polyline</td>
<td>(5, 2), (5, 14)</td>
</tr>
<tr>
<td>Willow</td>
<td>polyline</td>
<td>(4, 19), (20, 19)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Town</th>
<th>Type</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lincoln</td>
<td>polygon</td>
<td>[(2, 18), (6, 18), (8, 16), (12, 18), (14, 18), (14, 8), (8, 8), (6, 14), (2, 14)]</td>
</tr>
</tbody>
</table>

Worboys’ Data Model [19] can describe a park [13].

<table>
<thead>
<tr>
<th>Park</th>
<th>Type</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fount</td>
<td>Polygon</td>
<td>[10, 4, 10, 4, 10, 4, 1980, 1986]</td>
</tr>
<tr>
<td>Road</td>
<td>Polygon</td>
<td>[5, 10, 9, 6, 9, 6, 1995, 1996]</td>
</tr>
<tr>
<td>Tulip</td>
<td>Polygon</td>
<td>[2, 3, 2, 7, 6, 3, 1975, 1990]</td>
</tr>
<tr>
<td>Park</td>
<td>Polygon</td>
<td>[1, 2, 1, 11, 12, 11, 1974, 1996]</td>
</tr>
<tr>
<td>Pond</td>
<td>Polygon</td>
<td>[4, 9, 7, 6, 6, 5, 1991, 9, 1996]</td>
</tr>
</tbody>
</table>

Topological Data Models, for example [6, 7], describe relations of spatial objects, such as a distance graph [13]:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM GIS ’09, November 4-6, 2009. Seattle, WA USA
Copyright 2009 ACM 978-1-60558-649-6/09/11 ...$10.00.
3. QUERIES

Example 3.1 Find areas where three park objects intersect.

SQL and ADT: Add the operators:

- \text{inter}_\Delta (\text{polyline}, \text{rectangle}) \to \text{polylines} & \quad & \text{This operator returns the intersection of a polyline ADT of the vector data model by a factor of } c_1 \text{ units in the } x \text{ and } c_2 \text{ units in the } y \text{ direction.}

- \text{shift}((\text{polyline}, b_1, b_2)) \to \text{polyline} & \quad & \text{This operator shifts a polyline ADT of the vector data model by } b_1 \text{ units in the } x \text{ and } b_2 \text{ units in the } y \text{ direction.}

Using these operators with the proper values of } c_1, c_2, b_1, \text{ and } b_2, \text{ the query can be now correctly expressed as follows:}

\begin{align*}
\text{SELECT} & \quad \text{inter}_\Delta((\text{shift}(\text{scale}(\text{S}, c_1, c_2), b_1, b_2), \text{T}), \text{P1}) \\
\text{FROM} & \quad \text{Street AS S, Town AS T}
\end{align*}

Example 3.2 Find intersections of the streets and the town.

SQL and ADT: Since the streets and the town maps have different units and origins, we need the following operators:

- \text{inter}_\Delta (\text{polyline}, \text{region}) \to \text{polylines} & \quad & \text{This operator returns the intersection of a vector data model polyline and region ADT.}

- \text{scale}((\text{polyline}, c_1, c_2)) \to \text{polyline} & \quad & \text{This operator scales a polyline ADT of the vector data model by a factor of } c_1 \text{ units in the } x \text{ and } c_2 \text{ units in the } y \text{ direction.}

Using these operators with the proper values of } c_1, c_2, b_1, \text{ and } b_2, \text{ the query can be now correctly expressed as follows:}

\begin{align*}
\text{SELECT} & \quad \text{inter}_\Delta((\text{scale}(\text{S}.X, c_1, c_2), \text{S}.Y), \text{P1}) \\
\text{FROM} & \quad \text{Street AS S, Town AS T}
\end{align*}

Example 3.3 Projected on a table, the shadow of a book is enlarged by 10 centimeters on all sides when the book is raised up towards the ceiling. Find the enlarged shadow.

SQL and ADT: We introduce the following operator.

- \text{buffer}(\text{rectangle}, \text{rational}) \to \text{rectangle} & \quad & \text{This operator takes in a rectangle and a rational number } d \text{ and finds a rectangle that contains the points } (x, y) \text{ such that } |x' - x| \leq d \text{ or } |y' - y| \leq d \text{ from some point } (x', y').

Using this operator with the proper values of } x_1, y_1, x_2, y_2, \text{ and } h, \text{ the query can be now expressed as follows:}

\begin{align*}
\text{SELECT} & \quad \text{buffer}(x_1, y_1, x_2, y_2, h) \\
\text{FROM} & \quad \text{Book}
\end{align*}
Example 3.4 Find (the length of) the shortest path from Lincoln to Chicago using the Edge relation.

SQL and ADT: We introduce the following operator.

\[
\text{shortest} \text{distance}(\text{graph}, \text{vertex}, \text{vertex}) \rightarrow \text{number}: \text{This operator takes in a distance graph with positive distances and two vertices and returns the shortest distance from the first vertex to the second vertex.}
\]

\[
\text{shortest} \text{path}(\text{graph}, \text{vertex}, \text{vertex}) \rightarrow \text{number}: \text{This operator takes in a distance graph with positive distances and two vertices and returns the shortest path from the first vertex to the second vertex.}
\]

The respective queries can be expressed similarly to the shadow query replacing buffer with the above operators.

SQL and Constraint Database: First we calculate the possible distances from Lincoln to the other cities.

\[
\begin{align*}
\text{CREATE VIEW Possible} & : \text{Possible}(\text{City}, \text{Mile}) \\
\text{SELECT} & : \text{City2, Mile2} \\
\text{FROM} & : \text{Travel} \\
\text{WHERE} & : \text{City1} = \text{"Lincoln" AND Mile1} = 0 \\
\text{RECURSIVE} & : \text{City2, Mile2} \\
\text{FROM} & : \text{Possible, Travel} \\
\text{WHERE} & : \text{City} = \text{City1 AND Mile} = \text{Mile1}
\end{align*}
\]

We can find now the length of the shortest path from Lincoln to any other city as follows.

\[
\begin{align*}
\text{CREATE VIEW Distance} & : \text{Distance}(\text{City}, \text{Mile}) \\
\text{SELECT} & : \text{City, Min(Mile)} \\
\text{FROM} & : \text{Possible} \\
\text{GROUP BY} & : \text{City}
\end{align*}
\]

(If we only wanted the distance to Chicago we could add a WHERE clause which selects the city to be Chicago.) We can use the above Distance relation and the Edge relation to find the actual shortest path. Note that an edge from City1 to City2 is on (one of) the shortest path(s) if City1 is Mile1 from Lincoln, City2 is Mile2 from Lincoln, and the length of the edge is exactly Mile2-Mile1 miles long. We can express this observation and find all the edges that are on any shortest path from Lincoln to any other city as follows.

\[
\begin{align*}
\text{CREATE VIEW Shortest} & \text{DAG}(\text{City1, City2, Mile}) \\
\text{SELECT} & : E.\text{City1}, E.\text{City2}, E.\text{Mile} \\
\text{FROM} & : \text{Distance D1, Distance D2, Edge E} \\
\text{WHERE} & : D1.\text{City} = E.\text{City1} \text{ AND D2.\text{City} = E.\text{City2} AND D2.\text{Mile} > D1.\text{Mile} \text{ AND D2.\text{Mile} - D1.\text{Mile} = E.\text{Mile}}
\end{align*}
\]

Let us view the Shortest_DAG relation as a directed acyclic graph (DAG) where the edges are directed from City1 to City2 because a shortest path always goes from City1 to City2. Clearly, Shortest_DAG is acyclic because no shortest path can contain a cycle. There can be several incoming edges to a City2 only if all of the incoming edges are on equally short paths. Hence if City1, City2, Mile) and (City1’, City2, Mile’), then Mile = Mile’. The DAG can be made a tree by selecting for each City2 the alphabetically smallest parent node City1 in the DAG.

Example 3.5 Find whether all the streets are connected.

SQL and ADT: We introduce the following operator.

\[
\text{connected(street map)} \rightarrow \text{Boolean}: \text{This operator takes in a street map and tests whether each pair of streets is connected by a path.}
\]

The query can be expressed similarly to the shadow query replacing buffer with the above operators.

SQL and Constraint Database: Consider the Street relation and assume that it is represented as a constraint relation. If the streets are connected, then we can go from any arbitrary street point, for example (5, 2), to any street. At first we find each pair of streets that intersect, making it possible to cross from one to the other:

\[
\begin{align*}
\text{CREATE VIEW Cross} & : \text{Cross(From, To)} \\
\text{SELECT} & : S1.\text{Name}, S2.\text{Name} \\
\text{FROM} & : \text{Street AS S1, Street AS S2} \\
\text{WHERE} & : S1.\text{X} = S2.\text{X} \text{ AND S1.\text{Y} = S2.\text{Y}}
\end{align*}
\]

Second, we find all possible ways to turn as follows:

\[
\begin{align*}
\text{CREATE VIEW Reach} & : \text{Reach(To)} \\
\text{SELECT} & : \text{Name} \\
\text{FROM} & : \text{Street} \\
\text{WHERE} & : \text{X} = 5 \text{ AND Y} = 2, \\
\text{RECURSIVE} & : \text{C.To} \\
\text{SELECT} & : \text{Cross AS R, Cross AS C} \\
\text{WHERE} & : \text{R.To} = \text{C.From}
\end{align*}
\]

Next, we find the number of different reachable streets.

\[
\begin{align*}
\text{CREATE VIEW} & : \text{Cross(From, To)} \\
\text{SELECT} & : \text{S1.Name, S2.Name} \\
\text{FROM} & : \text{Street AS S1, Street AS S2} \\
\text{WHERE} & : \text{S1.X} = \text{S2.X AND S1.Y} = \text{S2.Y}
\end{align*}
\]

Example 3.6 While making a furniture delivery, a truck cannot use wet streets with a slope greater than 27°. The furniture store is at (5, 2) on Vine street, just opposite the
hospital. The delivery location is at (19, 19) on Willow street. Suppose the entire town map is located on a hillside with the elevation $z = 0.5(x + y)$. If it rains now, can the truck make the delivery today?

**SQL and ADT:** There is no standard ADT for this query.

**SQL and Constraint Database:** First, find the slope of the streets. Consider any street with points $(x_1, y_1)$ and $(x_2, y_2)$ on it. Between these two points, the length of the street projected onto the $(x, y)$ plane is:

$$length = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

The rise in elevation is:

$$rise = 0.5(x_2 + y_2) - 0.5(x_1 + y_1) = 0.5((x_2 - x_1) + (y_2 - y_1))$$

Hence we have the constraint: We also have:

$$\frac{rise}{length} \leq \tan(27^0) \approx 0.5$$

or

$$rise \leq 0.5 length$$

Since the truck has to go both up and down the street, we can assume that the rise is positive (otherwise interchange points $(x_1, y_1)$ and $(x_2, y_2)$). Hence we can square both sides:

$$rise^2 \leq 0.25 length^2$$

Substituting, we get:

$$(x_2 - x_1)(y_2 - y_1) \leq 0$$

We can use the above observation to find the safe streets by:

```
CREATE VIEW Safe(Name, X, Y)
SELECT S1.Name, S1.X, S1.Y FROM Street AS S1, Street AS S2, WHERE S1.Name = S2.Name AND (S2.X - S1.X) (S2.Y - S1.Y) \leq 0
```

Finally, we can proceed as in Example 3.5 but using relation Safe instead of Street.

## 4. CONCLUSION

Our argument that "fewer is better" regarding ADTs and operators may be as popular among members of the International Committee for Information Technology Standards as the notion of "basic English is enough" among the editors of the Oxford English Dictionary, which lists over 500,000 English words. Our aim was not to please experts who love complexity but to appeal to users who seek simplicity. While dictionaries in libraries grow larger, language in life is simplified. Yet expressiveness is preserved because simpler grammar is compensated by stricter word order and smaller vocabulary by words that can be both nouns and verbs.

As New English simplifies Old English, constraint query languages simplify GIS query languages. Yet expressiveness is again preserved because fewer ADTs and operators are compensated by allowing attributes like $x, y$ and $t$ to refer to both finite and infinite sets of points in space and time.

In human languages, flexibility springs from simplicity. Likewise, for GIS query languages to be widely used by millions, simplicity is not an option but a must.

## 5. REFERENCES