Towards an Extended SQL for Treating Spatial Objects
M. Egenhofer
in: Y.C. Lee (ed.), Trends and Concerns of Spatial Sciences,

TOWARDS AND EXTENDED SQL FOR TREATING SPATIAL OBJECTS

Max Egenhofer
University of Maine
Department of Civil Engineering
103 Boardman Hall
Orono, ME 04469
U.S.A.
MAX@MECAN1.bitnet

Abstract

Current conventional query languages for databases such as SQL or Query-By-Example deal exclusively with alpha-numeric type data and do not provide support for specific queries on spatial data. There is a considerable trend towards combining non-spatial data with space-related data or even founding information systems such as GIS/LIS and CAD/CAM on geometry. Besides a support from the databases, this incorporation of spatial data requires that the user's interface must provide suitable tools for interacting with spatial data.

In this paper, we will propose in detail how SQL, a conventional query language, should be extended in order to be able to deal with spatial data. First, what the components of a query language are and what tools a user should have to query databases is analysed. The structure of SQL as relational calculus is investigated and compared with the structure of an object-oriented model, a more advanced and sophisticated data model than the relational model. Subsequently, the specific properties of spatial data are presented, and it is proposed how to incorporate them into the current SQL syntax without adding major changes.

1. Analysis of User Queries

In this chapter we will analyse what a query language should treat and what its parts are. We will investigate what the user should be able to address from a query language and
what information the database needs in order to access data. We will look at data retrieval and presentation in a more general and more advanced matter than the relational model does. The object-oriented approach concentrates on objects and relations between objects without any dealing with more details of the objects. This is different from the way the relational data model structures data. In the following, we will see that the object-oriented approach allows the user to deal with data with less or even without any knowledge of the internal structure of the objects.

1.1 A Query To A Database

In an object-oriented view, a query to a database consists of

- addressing the object types with which the query deals,
- specifying the constraints on the objects (this includes constraints between objects),
- specifying the fields for extract by the query,
- specifying the type of the output display and the display device,
- specifying the output format.

These steps somehow reflect the way humans extract information from traditional filing systems. The query corresponds to a request to someone else to search the existing files for some data. The person doing the search first has to choose the files where the related data is archived, then he or she picks the referenced data and combines them with others using cross-references or parts of the data found. Finally, he or she takes the requested information from the selected data and brings them in a form such that they are most useful for the purpose of the requestor.

2. Analysis of SQL Statements

SQL is a structured query language for relational databases [Chamberlin et al., 1976] and was introduced as the query language for System R [Astrahan et al., 1976]. Due to the underlying relational data model [Codd, 1970], SQL deals only with relations and
combinations of relations, and it can be even seen as a purely relational algebra and a relational calculus with 'syntactic sugar' added [Korth and Silberschatz, 1986]. Any query in SQL results in a relation (including the empty relation). For an intensive discussion on the relational data model, see Ullman [1980].

SQL is not only a 'query' language (where query is understood in its proper sense), but also a manipulation language allowing the user to both inquire about the current state of a database and to change this state by adding, deleting, or modifying its contents. In this section, we will concentrate on the basic 'query' aspects. The needs for an appropriate data manipulation language will be discussed in another paper.

An SQL-query can be separated into 3 parts:

- The SELECT clause corresponds to the projection operation of the relational algebra and is used to specify the attributes to be included in the output list from the items chosen.

- The FROM clause designates a list of the relations with which the query is concerned.

- The WHERE clause imposes conditions between a relation and a value or between two relations.

2.1 SELECT Clause

The SELECT clause specifies how the resulting table will look. On the one hand, SELECT is a filter passing only the values of the specified attributes to the next higher level; on the other hand it determines the sequence of the attributes in the table.

SELECT has two operational modes dependent on the level of the query. A 'SELECT x, y, z' on the main level means 'make a table with the attributes x, y, z' and implicitly write the table to the output device. The same statement on a lower level, however, would only influence the composition of the table to be passed without any direct influence on the output.
2.2 FROM Clause

The FROM clause specifies the tables (relation names) which must be retrieved for combining and filtering. The relation names in a FROM clause correspond to the former SELECT clause and the following WHERE clause(s).

In general, all attributes stated in a SELECT must be part of the relations mentioned in the following FROM. The same applies to the WHERE clause in which only attributes which were mentioned in the former FROM clause may be used.

Regulations which are defined in subqueries are local, i.e., they are only 'visible' in the subquery itself and in its directly following subqueries. Thus, 'local' relations can be 'global' for all following, nested subqueries.

2.3 WHERE Clause

The WHERE clause imposes conditions on relations. A condition is a combination of two predicates with an operator where the predicates are either a specified value, the value of a field in a relation, or a value evaluated by some function.

Several conditions can be combined with the logical operators AND and OR. NOT allows the negation of a condition or of a combination of conditions.

The operators for the conditions include the relational operators =, >, >=, <, <=, <>. Inclusion in a set can be checked by IN, ranges are stated by BETWEEN .. AND .... LIKE is a weaker EQUAL comparing similar strings and substrings.

Note that the predicates in the WHERE clauses always act on attributes, not on relations. This requires that the user has detailed knowledge about the structure of the relation(s) he deals with. So, in what he is doing is comparing relations by comparing their detailed parts. For example, the query on all persons smaller than 6'3" must be expressed by a predicate on the height of persons, namely

WHERE height < 6'3"

and cannot be expressed by
WHERE person smaller than 6'3".
These restrictions will be essential when defining operators for spatial objects.

2.4 Subqueries

Subqueries are parts of queries which form a complete query by themselves; however, the result of subqueries is not processed to some output, but contributes to a condition in a higher-level query part. SELECT can be used for subqueries contributing to the conditions in WHERE clauses. The very first SELECT-statement of a query is overloaded as it implies that both a relation is returned as a result and the result is printed on the screen. Formatting parameters can be added to the SELECT clauses allowing the user to structure the output. They include:

- DISTINCT for elimination of duplicates.
- ORDER BY allows the user to define the output to appear in sorted order.
- GROUP BY allows the user to compute functions of groups of tuples. The functions to compute include AVERAGE, MINIMUM, MAXIMUM, TOTAL, and COUNT. It is possible to state conditions that apply to groups rather than to relations by HAVING clauses.

2.5 SQL — A Relational Query Language

How close this SELECT-FROM-WHERE pattern is to the operations on a relation can be easily seen by looking at a table and how data can be extracted from it. A table is a representation of a relation with the colons being the structure of each item (occurrence) and each row representing one item.

<table>
<thead>
<tr>
<th>HOUSE:</th>
<th>Number</th>
<th>Type</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>residence</td>
<td>white</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>residence</td>
<td>yellow</td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>office</td>
<td>gray</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Table HOUSE with the colons number, type, and color.
The FROM clause determines the name of the table to choose from. The WHERE clause specifies the conditions on the rows, i.e., it determines which items (rows) to select. Finally, the SELECT clause determines which colons to take from the result. For example:

```
SELECT number, type
FROM house
WHERE color = 'yellow'
```

determines to take the table HOUSE (FROM house), thence all the rows satisfying the condition (WHERE color = 'yellow').

<table>
<thead>
<tr>
<th>number</th>
<th>type</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>residence</td>
<td>yellow</td>
</tr>
</tbody>
</table>

Figure 2: The relation HOUSE with all items satisfying the condition color = 'yellow'.

The resulting relation must be reduced to a table containing only the colons number and type (SELECT number, type).

<table>
<thead>
<tr>
<th>number</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>residence</td>
</tr>
</tbody>
</table>

Figure 3: The relation HOUSE with all items satisfying the condition color = 'yellow' and the desired fields extracted.

The 5 functional operations of relational algebra (Cartesian produce, projection, selection, set union, and difference) are incorporated in SQL as follows [Korth and Silberschatz, 1986].

- The Cartesian product is represented in the FROM clause.
- Projection is performed in the SELECT clause.
• Algebraic selection is represented by the WHERE clause.

• Set union and set difference appear explicit in the syntax of SQL.

2.6 SQL Vs. Object-Oriented

Comparing the SQL-syntax with the object-oriented query design, we see two major differences:

1. The sequence in processing a query is somehow inverted starting with the last two issues (extracting and presentation), then addressing the objects, and finally formulating the conditions.

<table>
<thead>
<tr>
<th>OBJECT-ORIENTED</th>
<th>RELATIONAL CALCULUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>addressing object types</td>
<td>SELECT</td>
</tr>
<tr>
<td>constraints</td>
<td>FROM</td>
</tr>
<tr>
<td>extraction</td>
<td>WHERE</td>
</tr>
<tr>
<td>presentation</td>
<td></td>
</tr>
</tbody>
</table>

2. The SELECT clause is used for three different purposes: the extraction of the desired object parts, the destination of the output, and the specification of the representation.

This overloading is an obstacle when extending SQL for two major issues: (1) treating objects which can have representations other than just alphanumerical printings, and (2) using the results for different purposes than creating a completely new output.

3. Spatial Data

In an object-oriented data model, data is structured such that similar objects sharing common operations form a class. Objects are the occurrences (instantiations) of data describing something that has some individuality. In addition to non-spatial data, spatial objects are characterized by their spatial location and extension.
From the mathematical theory of simplicial complexes we learned that each spatial object has a dimension. This dimension corresponds to the space in which this object is the elementary building block.

The presentation of spatial data can be different from the presentation of non-spatial data as spatial data can be graphically presented reflecting their spatial location and extension. Besides some statistical graphics, non-spatial data is presented exclusively as alphanumerical output. Moreover, combinations of graphical and alphanumerical output are needed in order to satisfy queries which require labeling of objects in a graphical presentation.

3.1 Properties of Spatial Data

Properties of objects are described as relations either between an object and a value or between objects. Such binary relations are of the Boolean type either being true or false. Very common are relations such as equal or greater than. Spatial data own additional properties which are specific for them and cannot be shared with non-spatial data. These relations are independent of the dimension of the object and thus can be applied for any space. We distinguish two types of relationships: topological relations and metrical relations. Topological relations are invariant under continuous transformation, while metrical relations depend on the metrical space. In the metrical space, we distinguish two relations between objects which are disjoint from each other: distance and direction.

In order to suitably combine and manipulate spatial data from an SQL interface, these spatial properties must be addressable by the user.

4. Towards Extending the SQK Syntax

In its current form, SQL is not well-suited for dealing with spatial data. The major deficiencies are:

- The specific relations of spatial data are not supported.
TRENDS AND CONCERNS OF SPATIAL SCIENCES

- The output facilities are supposed to be purely alphanumerical.
- No interactive interaction between a result presented and a query is incorporated.
- The results of a query always create a new output. No manipulation with the current output is possible.

It is obvious that these issues can only be incorporated by extending the syntax of SQL. The goal of changes in the syntax, however, is to add as few as possible new keywords, and not to change the overall structure of SQL.

4.1 Proposals

There have been several proposals on how to extend SQL such that it includes the treatment of spatial data; however, most of them lacked a consistent concept and were incomplete. We will give a short summary of the most significant proposals.

In Sikeler [1985] it is proposed to extend the syntax of SQL for two issues: (1) the treatment of spatial relations is included and (2) a picture list managing the graphical output is added. The spatial relations are strictly separated from the relations between non-spatial data by adding another clause (‘WITH LOCATION’) in which additional relations such as NEXT_TO are treated. The picture list is added to the ‘attribute list’ of the SELECT statement. The picture list is supposed to be the parallel instruction to the projection of the non-spatial data parts. In order to distinguish which parts to print and which parts to draw, the keyword GRAPHIC is introduced. Results of subqueries are passed to the next higher level to be fed into spatial relations by adding the postfix .LOC to the spatial objects indicating that the geometry (location) of the relation should be passed.

A typical query in this extended form would look like:

```sql
SELECT e.name, GRAPHIC (p)
FROM e, p
WITH LOCATION p NEXT_TO
    SELECT s.LOC
FROM s
WHERE s.name = xyz
WHERE e.nr = p.name
```
Another extension of SQL is tailored to raster image processing [Roussopolos and Leifker, 1985]. PSQL (Pictorial SQL) combines direct spatial and indirect alphanumeric search. Each spatial object is extended by an attribute 'loc' for its location. This attribute is referenced in the SELECT clause for graphical output and in a specific clause for treating spatial relations.

The syntax is extended by two clauses; AT specifying the area to treat, and ON specifying a predefined output format (picture list). The picture list allows juxtaposition (synthesis) of dissimilar information of the same area from different sources.

The additional operations on raster cells are COVER, COVERED_BY, and OVERLAP. Additional functions are offered for calculating the area of an areal object and extreme points of lines or areas, such as NORTHERN and NORTHEASTERN.

A typical query in PSQL is:
```
SELECT city, LOC
FROM cities
ON us-map
  AT LOC COVERED_BY (4 +/- 2, 6 +/- 3)
WHERE population > 45000
```

4.2 Extended SQL

In the following section, we will address extensions to the SQL syntax which are essential and obvious. We will show where they should be added and how the new syntax will look.

In order to provide solutions for the deficiencies we addressed earlier, we propose the following solutions.

4.2.1 Extending the FROM Clauses

The specific properties of spatial data must be supported by incorporating the topological and metrical relations into the SQL syntax. The spatial relationships correspond to the relationships between non-spatial data. Thus, the FROM clause is the
adequate place for adding the spatial relations. We propose to add the following, minimal set of relations between spatial objects:

1. topological relations
   1. disjoint
   2. contains
   3. inside
   4. meets
   5. overlap
   6. intersect
   7. common_boundaries
   8. equal

2. metrical relations
   1. direction: the relations for cyclic intervals (which correspond to the topological relations between spatial objects)
   2. distance: the conventional relations such as equal, less, greater.

We do not include 'fuzzy' relations, i.e., relations which cannot be strictly defined. Fuzzy relations are expressions such as CLOSE or FAR. First attempts on defining fuzzy relations between spatial data are promising [Robinson et al., 1985].

4.2.2 The Parameters of the Spatial Relations

Topological relations are exclusively between two spatial objects and there are no constant values which can be used as parameters in these relations. For example, it is not possible to formulate

WHERE parcel intersects 5.

At first glance, it seems feasible to define the spatial relations on objects directly. For example, the INTERSECT relation between two parcels could be expressed by the WHERE clause

WHERE parcel_1 INTERSECT parcel_2.

This approach is different from the conventional, relational formulations of conditions in SQL. In general, conditions are exclusively expressed on predicates. In order to achieve consistency with the relational model, we either have to express the spatial relationship on a field of a relation (attribute of an object) or on a value which is
evaluated by a function. For the first variant, a field (attribute) ‘geometry’ must be added to each space-related relation. The INTERSECT relation then looks like

$$\text{WHERE parcel}_1.\text{GEOMETRY INTERSECT parcel}_2.\text{GEOMETRY}.$$  

This implies that GEOMETRY is a reserved word and must not be used for any other purpose explicitly. Using fields somehow implies that the values for establishing the spatial relation are directly and explicitly stored; however, such an assumption cannot be made. The geometry for an area, for example, is derived from its bounding edges.

The alternative is to express the spatial conditions by using a function evaluating the geometry of an object. The function GEOMETRY determines the geometry of an object such that it can be processed by the spatial relations.

The INTERSECT relation then looks like

$$\text{WHERE GEOMETRY (parcel}_1) \text{ INTERSECT GEOMETRY (parcel}_2).$$

5. Conclusion

In this paper, we investigated how to extend SQL in order to include the specific treatment for spatial data. We analysed the components of a query language in an object-oriented view and compared them with the structure of SQL, a purely relational calculus. We found out that SQL somehow inverts the single steps of a query by first addressing what to extract from the objects and then to state which objects to take and how to combine them. Moreover, SQL overloads the SELECT clause by using it for both the extraction of data and the direction of the alphanumerical output. As a result of this, any output is supposed to be printed on the current output device.

Spatial data own properties which are different from non-spatial data. Topological and metrical relationships between spatial data have to be added to the conventional conditions between non-spatial data. We proposed to extend the FROM clause in the SQL syntax by adding keywords for the spatial relations. Details of the extension showed again the difference between an object-oriented approach, where operations on objects can be formulated, and a relational calculus requiring operations on fields of relations.
Acknowledgement

This project was funded by a grant from Digital Equipment Corporation.

References


