17

GISQL - A Query Language Intepreter for Geographical Information Systems

G. Costagliola, G. Tortora, M. Tucci

Dipartimento di Informatica ed Applicazioni
University of Salerno - I84081 Baronissi (SA) - ITALY
Tel. +39 - 89 - 965 333 Fax +39 - 89 - 965 382
e-mail: jentor@udsab.dia.unisa.it

M. Busillo

ITALDATA S.p.A.
Pianodardine - Avellino - ITALY

Abstract

This paper presents the definition and manipulation language GISQL for Geographical Information Systems together with its interpreter. The main characteristic of the GISQL query language is to provide the user with an homogeneous interface for querying both spatial and alphanumeric information. The homogeneous interface consists of an extended version of the well-known SQL query language. Although the information is actually stored in two different databases, the user is presented with the view of a single virtual relational database, and the GISQL interpreter is in charge of managing the actual connections to the two phisical databases. Then, a GISQL query can contain any combination of spatial and alphanumeric conditions in a single command. In order to provide the user with the view of a unique relational database, the GISQL interface makes use of the concept of virtual tables. A virtual table contains both columns to hold the values of alphanumeric attributes and columns to hold the values of graphical attributes. The alphanumeric values are directly extract from the SQL database while the graphical values are derived from the graphical database.

Keywords

Spatial data management, modeling of visual information, geographic information systems

1 INTRODUCTION

Geographical Information Systems (GIS) are based upon the integration of a DBMS with (interactive) graphical technologies for the manipulation of maps. The growing interests and applications for GIS's is raising new and complex problems in the management, storage, analysis, and visualization of geographic systems in a graphical environment. The basic functionalities of such systems can be summarized as follows:

- geographic input data coding and processing;

S. Spaccapietra et al. (eds.), *Visual Database Systems 3*

- DBMS organization;
- querying;
- analysis and manipulation of data;
- visualization.

The information that a GIS has to manage regards both usual features of data, traditionally represented by alphanumeric attributes, and geometrical features related with spatial shape and location. Thus, the data management in a GIS has to take into account also topological relations based on spatial attributes (e.g., connection, adjacency, overlapping, etc.). Moreover, in most cases alphanumeric and spatial information derive from different sources and are organized in disjoint and independent databases. Therefore, alphanumeric and spatial data are often not strongly integrated and different query languages are used in the same system. In fact, conventional query languages are not suited for both kind of data, or at least need more functionalities and new executing mechanisms.

This work aims to the integration of the different nature of data considered in a GIS, through the use of a unique definition and manipulation language, called GISQL, for alphanumeric and spatial data. A prototype version of an interpreter for GISQL has been implemented based on the Siemens product SICAD (SIemens Computer Aided Design) in collaboration with SNI-ASW (Siemens Nixdorf Information Systems - ASW). Our approach is based on the extension of SQL to the management of spatial data. This is mainly motivated by the fact that SQL is widely accepted and used as a standard by both application designers and users. Although this approach has already been attempted in several works, e.g., (Egenhofer 1994, Herring 1987, Ingram 1987, ISO 1990], our proposal expressly addresses the following topics:

a) Definition of topological relations among geographic entities. A given entity is considered as geographic if it is described at least by one geographically referred attribute. The use of geographic attributes requires the analysis of spatial properties, deriving from the shape and location of entities, that can involve more than a single entity. As an example, a spatial relation can be considered between a river and a house that are closer than one kilometer, or it can be interesting to consider all the cities touched by a road. In general, a *topological relation* states a link among entities based on their geographic attributes. Although a number of topological relations can be defined, we aim at reducing them in terms of elementary relations between points, lines and regions, such as adjacency, inclusion, overlapping, etc.

b) Uniform vision of alphanumeric and graphical information. Our goal is to obtain a logical organization of data based on the relational model. This corresponds to allow each table to contain any number of columns representing geographic attributes. Of course, we also have to consider new data types for the values of these attributes. We have chosen a minimal set of topological data types, including *points*, *polylines* and *regions*, to match all the description requirements for graphical entities in a two-dimensional space. Each of these types can be seen as structured. For example, a list of scalar values defines the vertex of a region, its color and filling pattern.

c) Integration of an extended DDL (Data Definition Language) and DML (Data Manipulation Language) with topological data. The extended language allows for a uniform management of alphanumeric and geographic data with the same SQL syntax. Users are not forced to learn new commands: the same commands apply to the new data types at the logical level. The existence of two distinct databases at implementation level is made transparent to the user.

d) Design and implementation of a prototype interpreter. An SQL query can be seen as a single selection and projection on a unique large table obtained as a Cartesian product of all the tables involved in the query. Of course this brute force approach is not reasonable in practice, because of its space and time inefficiency. The goal of an SQL executor is the decomposition of the query in a sequence of simple and efficient operations of projection and joining with the

generation of smaller temporary tables. The same approach has been chosen to design an interpreter for GISQL. Although alphanumeric and geographic information is physically split in two different databases, one can formulate queries with arbitrary combinations of textual and topological conditions, and the interpreter is in charge of decomposing and executing them separately. It also creates virtual temporary tables used to combine the results obtained in a sequence of subqueries solutions.

The remainder of the paper is organized as follows. Section 2 presents a concise overview of some related works on spatial query languages based on extensions of SQL. A description of GISQL is given in section 3, where spatial data types, spatial operators and functions are defined and an extension of the SQL syntax and semantics is discussed. Section 4 describes the architecture of the GISQL interpreter, together with some implementation notes. Some remarks on current and future developments conclude the paper.

2 RELATED WORKS ON SPATIAL QUERY LANGUAGES BASED ON SQL

The need for spatial query language is common to all the application domains for Geographical Information Systems. In this field, the main obstacle to the diffusion of a standard language for spatial databases is the fact that languages for traditional databases already accepted as standards, like SQL, QUEL and Query-by-Example, are still lacking of capability to support spatial data manipulation. In the GIS domain, it is necessary to develop and extend spatial query languages to support spatial applications. Many efforts have been made to achieve a query language that satisfies all the needs of spatial applications and that is not too far from the currently accepted standards (ISO 1990). In particular, the wide diffusion of SQL-based applications motivates the significant commercial and research interests toward the extension of the SQL language. However, the nature of spatial data is intrinsically different from conventional one, and it is not very easy to manage them without modifying the syntax of a language conceived and suited for alphanumeric data. As an example, the tabular presentation of the results of a query typical of standard SQL is not adequate for spatial applications, which usually require graphical outputs (Egenhofer 1988).

Among the proposals of query languages extended for GIS applications and based on SQL, two recent ones are shortly illustrated in the following, to explain the characteristics and requirements of such languages. An extended analysis can be found in (Egenhofer 1992).

One of the main features of Spatial SQL (Egenhofer, 1994) is how it combines the description of a query and the graphical presentation of its solution. The approach is based on an extension of SQL to formulate queries and on the GPL (Graphical Presentation Language) tools to present graphical results. Spatial SQL supports spatial data abstraction to reflect the human conception of topological space. It provides operations, methods and relations that apply to spatial objects produced by direct manipulation devices, and introduces new domains for spatial attributes of zero, one, two and three dimensions, respectively. Spatial operations include some unary functions like *dimension*, *length*, *volume* and the binary operations *distance* and *direction*. Spatial binary relations apply to pairs of spatial attributes and return a Boolean value based on the intersections of their borders and internal parts. Typical spatial relations are *disjoint*, *meet*, *overlap*, *inside* and *covered*.

GPL provides some tools for the manipulation and graphical presentation of spatial data resulting from a query. A *visualization environment* contains the information for visually represent output data when a query is processed.

SQL3 is a project both ANSI and ISO, (ISO 1990), are working on to define a new SQL standard, including enhancements to manage spatial data for GIS applications. Many of the proposed extensions consists of object-oriented features such as inheritance, encapsulation and modularity. SQL3 allows to define subtables/supertables and abstract data types (ADTs). A subtable inherits all the columns of a parent table and can have additional columns. Users will

benefit of adopting standardized sets of ADTs for different application areas. The set of basic ADTs proposed for GIS applications include:

- POINT : a position in a n-dimensional space
- NODE : a point with topological connections
- LINK : a polyline connecting two nodes and consisting of a sequence of points
- CHAIN : an ordered set of links
- POLYGON : a region enclosed in a sequence of links
- AREA : a set of polygons.

The proposed spatial functions include spatial operators which return both scalar and spatial values. The introduction of *list*, *set*, *record*, etc., data types makes SQL3 able to support functions returning structured values. Some of the spatial operators defined in SQL3 follow:

x-coordinate:	POINT → REAL
y-coordinate:	POINT → REAL
inside:	(POINT, AREA) → BOOLEAN
links_attached_to:	NODE → REAL
bounds:	LINK → POLYGON
union:	(POLYGON, POLYGON) → POLYGON
inside:	(AREA, AREA) → BOOLEAN.

Although there are many other proposals for a query language for geographic information systems based on SQL, their use and diffusion is still very small and in most cases limited to the academic environment. Most of the currently available software products for GIS applications keep alphanumeric and spatial data split into different databases that are accessed by different tools. The lacking of a general method for solving queries independently of the specific applications is one of the main obstacles to the success of such languages. In the present work we aim both at defining a new spatial query language based on extending SQL, called GISQL, and at implementing a prototype interpreter for the GISQL language independent of any particular application. To achieve this result, we have performed an abstraction step from the twofold nature of a GIS, namely from the two separated alphanumeric and spatial databases. The GISQL language considers a unique virtual database and the designed interpreter deals with this higher level abstraction. As an experimental application, we have implemented this interpreter within the SICAD environment. The GISQL language and its interpreter are described in the following sections.

3 THE GISQL LANGUAGE

The main characteristic of the GISQL query language is to provide the user with an homogeneous interface for querying both spatial and alphanumeric information. The homogeneous interface consists of an extended version of the well-known SQL query language. Although the information is actually stored in two different databases, the user is presented with the view of a single virtual relational database, and the GISQL interpreter is in charge to manage the actual connections to the two physical databases. Then, a GISQL query can then contain any combination of spatial and alphanumeric conditions in a single command.

In order to provide the user with the view of a unique relational database, the GISQL interface makes use of the concept of *virtual tables*: all the geographic information (both spatial and alphanumeric) is virtually organized in relational tables. This virtual representation is all the user needs to know about the database structure in order to use the GISQL query language. A virtual table contains both columns to hold the values of alphanumeric attributes and columns to

hold the values of graphical attributes. The alphanumeric values will be directly extract from the SQL database while the graphical values must be derived from the graphical database.

Corresponding to each virtual table, there is a *physical table* consisting of the alphanumeric attributes plus a new attribute named *location* holding a pointer to a data structure (BLOB - Binary Large OBject) representing a graphical object in the graphical database (Figure 1).

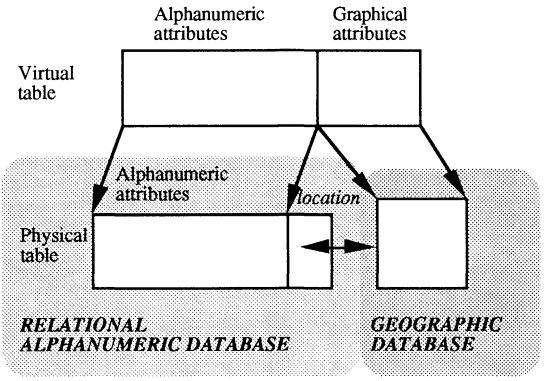

Figure 1 The correspondence between virtual and physical tables.

An entry in a graphical column of a virtual table contains the field values of geographic structured data types in the same way as an entry mm/dd/yy corresponds to the field values of the structured data type DATE. Analogously, we provide operators to extract field values from the structured data and functions to manipulate them. These operators and functions will actually operate on the BLOBs of the graphical database through SICAD procedures.

Spatial data types

The identification of a set of spatial data types and a set of operators and functions is one of the basic processes in the definition of a query languages for geographic databases.

In this section we introduce the structured spatial data types POINT, POLYLINE and REGION. Although more complex models for spatial data have been adopted in several applications, all the most frequently used spatial operators can be reduced to these three elementary types. We have chosen a minimal set of data types needed to represent geographic information in a two-dimensional space because more specific and structured types can make the syntax of a query language dependent on the particular model for spatial data. This choice is also suitable for future extensions to allow user defined data types.

For each structured type, a set of fields is defined:

POINT == (X, Y, COLOR)
where
X and Y are the x- and y- coordinates of the point,
COLOR is an optional field indicating the color of the point.

POLYLINE == (LCOORD, COLOR, MODE)
where
LCOORD is a list of coordinates,
COLOR is an optional field indicating the color of the polyline
MODE is an optional field indicating the polyline type (i.e., continue, dashed).

REGION == (LCOORD, X, Y, COLOR)
where
LCOORD is a list of coordinates,
X is the x-coordinate of a reference point inside the region,
Y is the y-coordinate of a reference point inside the region and
COLOR is an optional field indicating the filling color for the region.
The reference point is used to determine which region is referenced in a plane. As an example consider Figure 2: in (a) the referenced region is A, in (b) the referenced region is B.

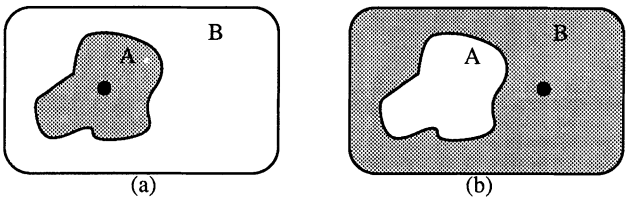

Figure 2 The use of a reference point to indicate a region.

In each type, a field corresponds to an attribute of the objects of that type.

Spatial Operators and Functions
To analyze the geometric characteristics of the objects and the topological relations among them, two types of operators need to be defined: *boolean* and *field operators*; and a set of *spatial scalar functions*.

A boolean operator verifies whether a topological relation holds between two elements, while a field operator and a spatial scalar function return the value of a graphical attribute of a particular object of type POINT, POLYLINE or REGION.

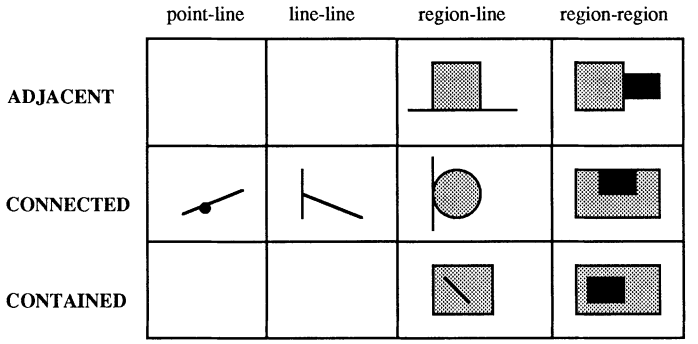

Figure 3 The spatial operators of GISQL.

The boolean operators introduced in GISQL and the possible combinations with object types are summarized in Figure 3. They are:

-ADJACENT(<*elem1*>, <*elem2*>)
true, if *elem1* is adjacent to *elem2* or vice versa

-CONNECTED(<*elem1*>, <*elem2*>)
true, if *elem1* is connected to *elem2* or vice versa

-CONTAINED(<*elem1*>, <*elem2*>)
true, if *elem1* is contained in *elem2* or vice versa.

A field operator is defined for each of the fields of the types POINT, POLYLINE and REGION defined above:

- **color_of**(<*elem_name*>)
It returns the color of the graphical element of type POINT, POLYLINE or REGION identified by *elem_name.*

- **xcoord**(<*elem_name*>)
-**ycoord**(<*elem_name*>)
They return the x- and y- coordinate, respectively, of the graphical element identified by *elem_name.* The graphical element type may be both POINT and REGION. In this last case the operators are applied to the reference point of *elem_name.*

-**lcoord**(<*elem_name*>)
It returns the list of vertices of the graphical element identified by *elem_name.* The type of may be both POLYLINE and REGION.

-**mode_of**(<*elem_name*>)
It returns the kind of line used to draw a POLYLINE.

-**set_color**(<column_name, color>)
It modifies the color of the graphical element identified by column_name

-**set_xcoord**(<column_name, value>)
-**set_ycoord**(<column_name, value>)
They modify the values of the x-coordinate and y-coordinate, respectively, of the graphical element identified by column_name

-**set_lcoord**(<column_name, list_value>)
It modifies the list of vertices of the graphical element identified by column_name

-**set_mode**(<column_name, mode>)
It modifies the kind of pen used to draw the graphical element identified by column_name.

The field operators **color_of**, **ycoord**, **xcoord**, **lcoord** and **modeof** may be used as arguments of the SELECT SQL statement to project particular attributes of a graphical object or to set conditions on the attributes of graphical data in the WHERE clause. The remaining **set_*** field operators may be used as arguments of the UPDATE statement to modify the tuples of a table.

To manipulate graphical information more easily, a number of spatial scalar functions have been defined. These, applied to graphical objects, return scalar values each indicating some properties of the objects themselves. Some examples include the length of a line, the area of a region, the distance between two objects. The following spatial scalar function have been added to GISQL:

DISTANCE(*region1*, *region2*)
DISTANCE(*region*, *polyline*)
DISTANCE(*region*, *point*)

DISTANCE(*polyline1*, *polyline2*)
DISTANCE(*polyline*, *point*)
DISTANCE(*point1*, *point2*)
AREA(*region*)
PERIMETER(*region*)
LENGTH(*polyline*).

The syntax specifications of the most characteristic parts of the GISQL language have been given by means of grammar rules.

Examples of use of the GISQL language:
Due to the introduction of the new data types, operators and functions, it is possible to define SQL-like commands which mix both alphanumeric and spatial information such that the syntax and semantics of SQL is not modified. Further, while the user is left with the view of a relational database, in fact he/she operates with two different non-homogeneous databases.

In the following, we provide a sample set of GISQL commands to show the potentiality of the GISQL language:

Commands from the Description Definition Language of GISQL

CREATE TABLE **ROAD**
(*name* char[15],
type char[5],
graph polyline)
This creates a table named **ROAD** with three columns:
-*name* a string of 15 characters to store the name of the road
-*type* a string of 5 characters to store the type of the road
-*graph* a polyline to store the graphical representation of the road.

INSERT INTO **ROAD** (*name*, *type*, *graph*)
VALUES ("Appia", "SS", ((10,4)(100, 75)(86, 50), BLUE, CONTINUOS)
This inserts into the table **ROAD** the tuple representing the "SS" type road named Appia. Its graphical representation is a BLUE, CONTINUOS polyline with coordinate vertices (10, 4)(100, 75)(86, 50).

UPDATE **ROAD**
SET *graph* = (**set_color**(*graph*, GREEN))
WHERE **ROAD**.*name* = "Appia"
This updates the table **ROAD** by modifying the color of the road named Appia from BLUE to GREEN.

DELETE FROM **ROAD**
WHERE **mode_of**(*graph*) = CONTINUOS
This deletes all the tuples from the table **ROAD** which are represented by CONTINUOS lines.

SELECT *name*, *graph*
FROM **ROAD**
WHERE **mode_of**(*graph*) = CONTINUOS.
This returns all the names of the roads drawn with a continuous line together with the corresponding road visualizations.

Commands from the Description Manipulation Language of GISQL

If a new table **STATE** is defined with attributes *NAME* of type string of characters and *GRAPH* of type region, the following query:

SELECT **ROAD.***name*, **ROAD.***graph*
FROM **ROAD, STATE**
WHERE CONTAINED(**ROAD.***graph*, **STATE.***graph*)
AND **STATE.***name* = "Pennsylvania"
AND LENGTH(**ROAD.***graph*) = 100

returns the names and the corresponding visualizations of all the roads in Pennsylvania which are 100 miles long.

SELECT min(LENGTH(*graph*))
FROM **ROAD**
WHERE **ROAD.***type* = "SS"

This selects the minimum length among all the "SS" type roads.

SELECT *name*, max(LENGTH(*graph*))
FROM **ROAD**
WHERE *name* <> "Appia"
GROUP BY **color_of**(*graph*)

This returns all the roads with name different from "Appia" with max length grouped by the color of the roads.

SELECT *name*, LENGTH(*graph*)
FROM **ROAD**
WHERE *name* <> "Appia"
GROUP BY **color_of**(*graph*)
HAVING sum(LENGTH(*graph*)) < 200

This returns all the roads with name different from "Appia" with length grouped by the color of the roads and with length less than 200 miles.

SELECT *name*, *type*
FROM **ROADS**
WHERE *type* <> "SS"
ORDER BY **color_of**(*graph*), *type*

This returns all the names and types of the roads which are not of "SS" type. The names and types are ordered with respect to the color and the type of the roads.

4 THE GISQL INTERPRETER

The GISQL interpreter translates a query into a set of subqueries each of which is specific either to the alphanumeric relational database or to the spatial database (Figure 4).

These queries are then solved by an appropriate module. The SQL module of Figure 4 solves alphanumeric subqueries by calling a standard SQL interpreter. The spatial module solves spatial subqueries by executing access functions to the geographic database. The set of these functions form a library that constitutes the interface with the physical spatial data.

The interpretation process consists of six phases, as shown in Figure 5. Several transformations are performed on an input query in order to separate the accesses to the alphanumeric and spatial databases. It is decomposed in elementary subqueries each containing only predicates of alphanumeric or spatial type.

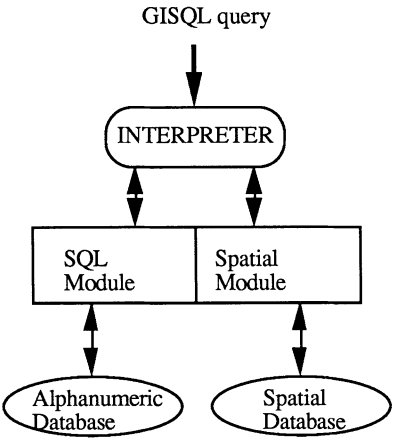

Figure 4 The schema for the GISQL System.

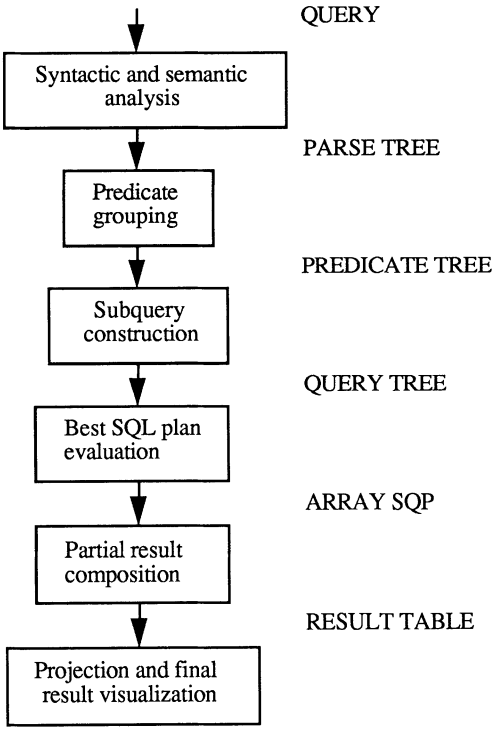

Figure 5 The interpretation process.

In the parsing phase, the query is syntactically and semantically analyzed and, after a preliminary optimization, a parse tree is created to represent the query in all the following phases. Then, the interpreter groups the predicates, in the subtree representing the *where* clause, that involve the same relation to avoid multiple readings of the same archive and possibly reducing the number of the subqueries generated in the next step. The rules to group predicates produce two types of subqueries: alphanumeric and spatial subqueries. The first type ones are solved by the SQL module; the second type ones are processed by the spatial module.

In general, the subqueries can be executed in more than one sequence, but different orders, called Sub-Query Plans (SQP), have corresponding different costs in terms of database accesses. In the optimization phase, some heuristic rules are used to select the optimal SQP. The cost estimation for an SQP is based on considerations regarding partial result propagation.

An example of heuristic rule is: "execute the alphanumeric subqueries before the graphical ones". This rule reduces the accesses to the geographical database which are usually more expensive than those to the alphanumerical database.

The last step concerns the presentation and visualization of the resulting table as specified in the *select* clause of the query. The actual visualization is a complex task involving context and zooming features (Egenhofer 1988) that are out of the scope of this work, since they depend on the particular GIS application.

5 FURTHER WORK

One of the most powerful mechanisms of SQL is the ability to specify nested queries (subqueries) within a *where* clause. This feature allows one to retrieve information by a unique complex query with any number of nesting levels. To efficiently solve nested subqueries, it is necessary to transform them into unnested ones by linearization mechanisms (Kim 1992). Our prototype interpreter for GISQL includes a solution to this problem which takes into account optimization issues and does not overload the interpreter. In fact, a linearization is accomplished by transforming the parse tree of a query before the optimization phase.

The GISQL interpreter has been implemented by using the tool YACC (Aho 1985) and the experimental prototype is under testing for possible engineering and commercial distribution. Further developments of both the language and the interpreter will probably be suggested by this testing phase. The most likely extensions presently foreseen regard user defined complex spatial data types and output presentation tools for spatial data.

6 REFERENCES

Egenhofer, M.A. (1994), SPATIAL SQL: A Query and Presentation Language, *IEEE Trans. on Knowledge and Data Engineering*, **6**, 1, 86-95.

Egenhofer, M.A. (1992), Why not SQL!, *Int. Journ. Geographical Information Systems*, **6**, 2, 71-85.

Egenhofer, M.A., Frank, A. (1988), Towards a Spatial Query Language: User Interface Consideration, *in Proceedings of 14th International Conference on Very Large Databases*, Los Angeles, 124-133.

Herring, J. (1987), TIGRIS: Topologically Integrated Geographic Information Systems, *in Proc. AUTO-CARTO 8*, Baltimore, 282-291.

Ingram, K., Phillips, W. (1987), Geographic Information Processing Using a SQL-based Query Language, *in Proc. AUTO-CARTO 8*, Baltimore, 326-335.

ISO/IEC JTC1/SC21/WG3 DBL-SEL3b (1990) Database Language SQL2, also including SQL3 extensions *ISO working draft.*.

Kim, W. (1992), On Optimizing an SQL-like Nested Query, *ACM Transactions on Database Systems*, 7, 3, 443-469.

Ooi, B.C., Sacks-Davis, R., McDonnell, K.(1989), Extending a DBMS for Geographic Applications, *Proceedings IEE 5th Conference on Data Engineering*, Los Angeles, 590-597.

Aho, A.V., Sethi, R. and Ullman, J.D. (1985), *Compilers,principles, techniques and tools*, Addison Wesley.

7 BIOGRAPHIES

Gennaro Costagliola

Gennaro Costagliola received the Laurea degree in computer science from the University of Salerno, Italy, 1987 and the M.S. degree in computer science from the University of Pittsburgh, Pittsburgh, PA in 1991. Since 1993 he is a Research Associate and Teaching Assistant with the Department of Computer Science at the University of Salerno.

His research interests include visual languages, visual programming, image database indexing, picture matching on parallel architectures.

He is a member of the Association for Computing Machinery and the IEEE Computer Society.

Genoveffa Tortora

Genoveffa Tortora received the Laurea degree in computer science from the University of Salerno, Italy, 1978.

From 1978 to 1988 she was a Research Associate and Teaching Assistant of computer science at the University of Salerno. From 1989 to 1990 she was an Associate Professor, and she is currently a Professor of Computer Science at the University of Salerno.

She published more than 30 papers in international journals, books, and conference proceedings, in the area of programming Languages and software engineering.

She chaired the Fourth International Conference on Software Engineering and Knowledge Engineering that was held in Capri in June 1992 whose proceedings were published by the IEEE CS Press.

From 1991 she is Dean of the Graduate School in Computer Science at the University of Salerno.

She is a Senior Member of the IEEE Computer Society.

From 1990 she is on the board of the Ph. Doctor School in Applied Mathematics and Computer Science of the University of Naples.

Maurizio Tucci

Maurizio Tucci received the Laurea degree in computer science from the University of Salerno, Italy, 1988.

Since 1992 he is a Research Associate and Teaching Assistant with the Department of Computer Science at the University of Salerno.

His research interests include syntactical and semantic aspects of visual languages, parsing technologies for visual languages, visual geographical databases, and tool integration for software engineering.Craig Standing is a lecturer at the Department of Information Systems at Edith Cowan University and is currently working towards a Ph.D. at The University of Western Australia. He obtained his B.A.(Hons) in Geography at the University of Lancaster, UK., and his M.Sc. in Computation at the University of Manchester Institute of Science and Technology. His research interests include: visual programming languages, and systems development methodologies.