# INTEGRATIONS OF STRUCTURED QUERY LANGUAGE WITH GEOGRAPHIC INFORMATION SYSTEM PROCESSING: GeoServer™

Matthew Heric, Geoscientist
and
Kevin D. Potter, Product Manager

Autometric, Incorporated
5301 Shawnee Road
Alexandria, Virginia 22312-2333 USA

## ABSTRACT

Geographic Information System (GIS) technology has progressed rapidly during the last ten years; as computer hardware and software capabilities have improved, GIS has been brought to the forefront of the high-technology arena. Indeed, the relative success of any developing or developed GIS is its ability to manage a variety of functional data.

To continue the expanding evolution of GIS technology, Autometric, Incorporated has researched and developed the GeoServer™. The GeoServer™ uses a revolutionary query processing capability which combines the strengths of Structured Query Language (SQL) database accessing techniques with geographic data processing techniques. As a result, information can be screened or selected using well-focused spatial and subject criteria. Furthermore, these advances allow the GeoServer™ to handle time-sensitive applications such as real-time data fusions, correlations, tracking, historical archiving, and event prediction.

This paper discusses how GIS techniques form the foundation of the GeoServer™ and details how it represents a substantial advance in GIS technologies. Additional information is provided on system architecture and functionality.

## KEY WORDS

Structured Query Language, GeoServer™, and Geographic Information System

## GEOSERVER™ BACKGROUND AND INTRODUCTION

The GeoServer™ provides the ability for an applications programmer (client) to invoke Geographic Information System (GIS) functionality across a network or on the same processor utilizing client/server technology. With this architecture, the GeoServer™ allows the programmer to work within a resident environment and invoke the individual GIS functions when needed.

The geographic information system, known as the Server, is running at all times, awaiting requests from the client which are written to the network. The Server receives the request and invokes the appropriate routines. Once the processing is completed, the Server writes any output parameters and the error status to the network and awaits the next request from the client. The client receives the Server's output and decides on the appropriate action to take.

The query process is independent of particular spatial relationships and object definitions and is based upon the use of function, recursion, and spatial constraint propagation. In general, the principal data structure used by the search procedures responsible for assembling together the locations of a multi-component object is a *semantic network* (Nilsson, 1980). Such a structure represents objects and relationships between objects as a labeled graph.

A query processor that can support both object- and location-based organizations of spatial data requires the ability to overlay multiple thematic layers dynamically and the ability to process spatial relation–ships between multiple objects dynamically during query processing. A flexible GIS cannot rely on the explicit representation and storage of every spatial relation that may be of potential significance to a user. A spatial query processing mechanism for a GIS must therefore be capable of handling implicit spatial information in a flexible and automated manner (Menon and Smith, 1989).

The GeoServer™ database consists of layers of stored information. Basic vector layer types are polygons (soils, parcels), lines (streams, roads), and points (manhole covers, telephone poles). Raster layer types are imagery (SPOT, Landsat, ARC Digital Raster Imagery) and maps ARC Digital Raster Graphics (ADRG). All layers of information in the database support common and user-definable feature attributes.

These information layers can be extracted from the database using the structured query language, called GeoSQL, processes. Specifically, using GeoSQL, data can be routed to a display or hardcopy output device. Layers of information can be accessed using a combination of spatial and attribute constraints as shown in Figure 1. Any application (Figure 2) can retrieve, update, delete, or insert data using the client libraries provided with the GeoServer™.

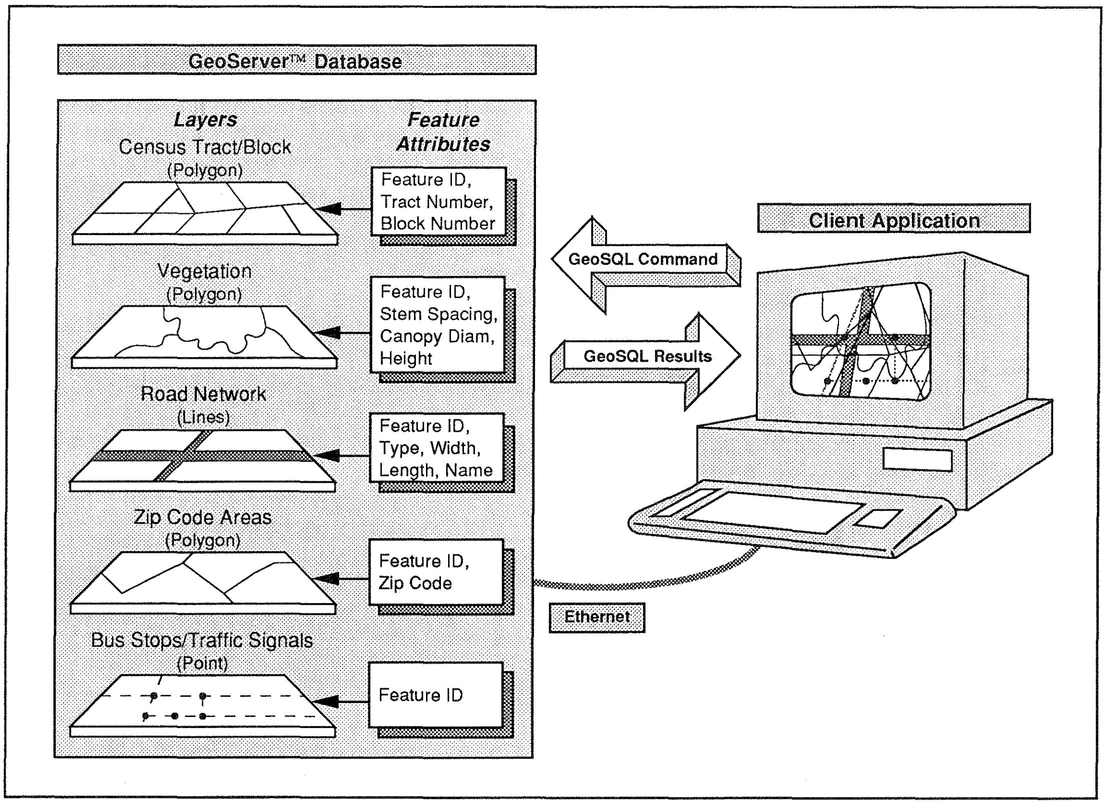

**Figure 1.** The GeoServer™ database consists of layers of stored information. Basic vector layer types are polygons (soils, parcels), lines (streams, roads), and points (manhole covers, telephone poles). Raster layer types are imagery (SPOT, Landsat, ARC Digital Raster Imagery) and maps (ARC Digital Raster Graphics). These layers of information can be accessed using a combination of spatial and attribute constraints.

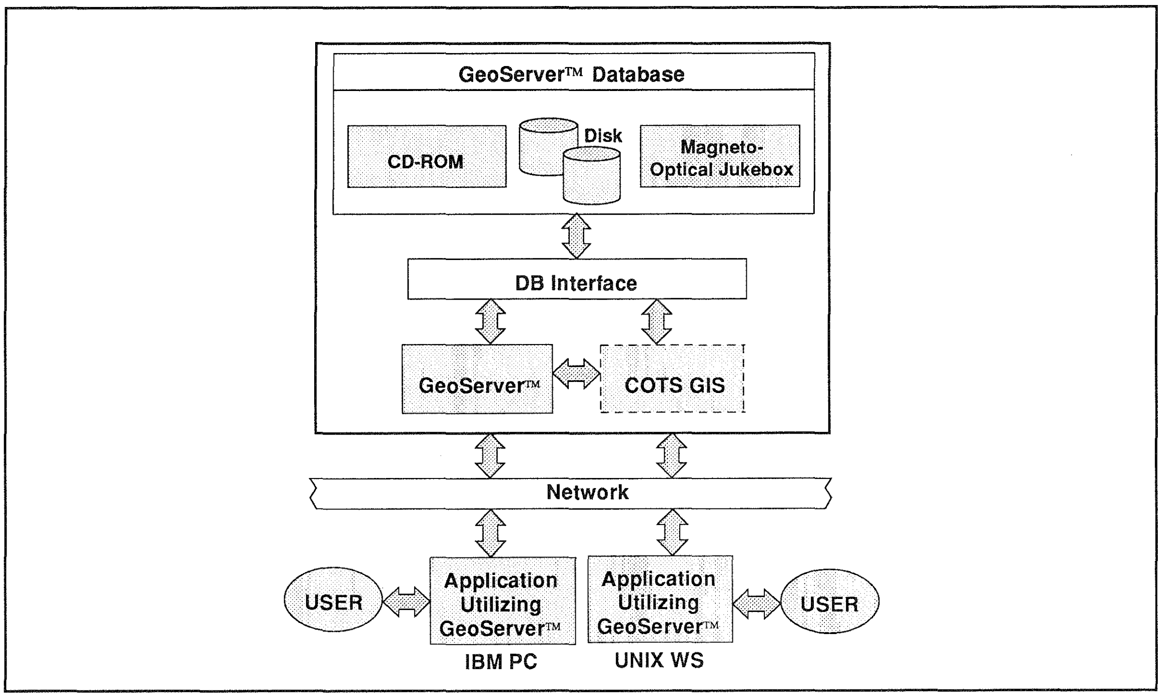

**Figure 2.** Applications can retrieve, update, delete, or insert data using the client libraries provided with the GeoServer™. Specifically, the GeoServer™ is shown here as a functional element in a complete analytical computational environment.

## GeoSQL Command Components

With GeoSQL, four command components may be included as part of the query command. When considered individually each is optional; however, a subset request must contain at least a Source Dataset specification, or a 'WHERE' expression. The following describes the four command components:

| | | |
|---|---|---|
| Destination Dataset | - | identifies the attributes that will be retrieved for features which pass the extraction criteria. The dataset may be conceptualized as a table with rows and columns; a column for each item listed in the Destination Dataset, and a row for each feature passing the extraction criteria contained in the 'WHERE' expression. Multiple attributes may be specified but must be separated by a comma. |
| Source Dataset | - | (optional if 'WHERE' expression is specified) identifies the dataset against which the selection is made. If included, the Source Dataset must follow the control word 'FROM' and precede the 'WHERE' expression. |
| Stored Subset | - | (optional) assigns a unique name to the subset of data that is extracted. If used, the Stored Subset must follow the control word 'INTO' and precede the 'WHERE' expression. |
| Expression | - | (optional if Source Dataset is specified) this component is used to specify the feature selection criteria. If included, it must follow the control word 'WHERE' and contain at least one operand. |

## Valid Command Words

Words which are used in the command to specify a component are defined within the system in the following tables:

Common Data Dictionary;
Source Dataset Tables;
User Tables (Stored Subsets); and
Template Tables.

The Common Data Dictionary (CDD) contains key words that are standard components of a query command (e.g., control words, operators), and words that are representative of a specific areas of interest. When each word in the query command is tested, it is first compared to words in the CDD.

Source Dataset Tables contain attribute information for features in a dataset. If a word is used to identify explicitly the source dataset, it must match a valid data set name (e.g., roads, vegetation, weather, water).

The results of a subset selection may be stored in a User Table. This is a table of information kept in computer memory which is made up of rows and columns; there is one row for every feature retrieved, and each column corresponds to an attribute that has been retrieved. The table name is assigned by specifying the Stored Subset component of a query command.

The Template Table contains definitions of user-definable spatial objects (circle, rectangle, point, line, polygon, ellipse); each has been assigned a unique template name. A template may be used as an operand in the query by using any valid template name.

## Valid Command Expressions

The Expression can vary in terms of the complexity of the Expression itself and in terms of the complexity of the operands.

| | | |
|---|---|---|
| Expression Complexity | - | A 'simple' expression contains only one operand; a 'complex' expression contains an operand-operator-operand sequence. |

e.g., simple—WHERE operand
complex—WHERE operand operator operand

| | | |
|---|---|---|
| Operand Complexity | - | A 'single' operand contains only one component whereas a 'multiple' operand contains more than one component, each separated by a semicolon. A single operand may be a feature specification, template, or the name of a previously derived Stored Subset. A multiple operand must be made up of ONLY feature specifications; a template name or Stored Subset name may not be used as a component of a multiple operand. |

e.g., simple—WHERE operand
multiple—WHERE operand; operand; operand

The semicolon in this context acts as a logical 'OR', where each operand member will be retrieved independent of the others.

An operand may also be qualified with attribute criteria. Only a feature operand may receive attribution, a template or Stored Subset may not. The attribute specification is enclosed in brackets and immediately follows the operand. It contains an attribute name, arithmetic operator, and numeric value or character string (depending on the attribute data type). The attribute specification may include more than one test, as long as the tests are joined by standard boolean operators (e.g., AND, OR, NOT). Numeric values are assumed to be in units which are compatible with the stored values in the database, unless the unit measurement is explicitly indicated. Character strings must be enclosed in single quotes.

e.g., WHERE operand [attribute1 = value]
WHERE operand [attribute2 = 'string']
WHERE operand [attribute1 = value and attribute2 = 'string']

Control Words

The 'SELECT' control word is mandatory and is automatically included in the command prompt. If 'FROM' is used to signal the specification of the Source Dataset component, it must immediately precede this component.

The Source Dataset 'clause' must follow either the word 'SELECT' or the Destination Dataset component if specified. If 'INTO' is used to signal the specification of a Stored Subset component, it must immediately precede this component. The Stored Subset 'clause' must follow the Source Dataset 'clause' if specified, the Destination Dataset 'clause' if specified, or the word 'SELECT'. A WHERE expression begins with the specification of the control word 'WHERE'. The expression must follow all other control words and clauses that may be specified in the query command.

The spatial operators 'INSIDE' and 'OUTSIDE' are used to initiate a spatial test. The test is conducted between a feature operand and a template or between a User Table containing feature IDs and a template. The template is referenced by name and must be the second operand. The criteria for passing the inside and outside tests are as follows:

INSIDE—feature totally or partially inside template; and OUTSIDE—feature totally outside template.

The Template referenced must be two-dimensional in nature when the 'INSIDE' and 'OUTSIDE' operators are used.

The spatial operators 'WITHIN' and 'BEYOND' are used to initiate a proximity test. This test may be between two feature operands, between a feature operand and a template, or between a User Table and a template. If a template is used, it must be the second operand and it must be a point, area, or line construct. The distance value to be used in the proximity test must follow the spatial operator in the 'SELECT' command.

EX: SELECT where roads within 100 mt swamp; and
SELECT where roads beyond 500 mt swamp

## GEOSERVER™ FUNCTIONALITY

Understanding the operative foundation of GeoSQL, the GeoServer™'s routines have been categorized into three function groups:

| | | |
|---|---|---|
| GeoServerCore™ | - | routines to initialize and disconnect from the Server, create spatial Templates, manage virtual memory, and access the database. |
| GeoServerModeler™ | - | routines to access cell map data and to create/manipulate Geosearch Templates. |
| GeoServerDisplay™ | - | routines that utilize the Server's display window to display, erase, modify, or interface with a user. |

## THE GEOSERVER™ DATABASE

The GeoServer™ database consists of an embedded relational DataBase Management System (DBMS) for managing most location and attribute data, system housekeeping tables, and optimized database models for managing vector, cell, and raster information. (More information concerning DBMS is provided by Abel and Smith (1986) and Guting (1988).)

Vector data contain features stored in "vector classes." Each feature is defined by its component point(s), line(s), or area(s). Examples of vector data include the location of all fire hydrants within a city (point data), a road network (line data), and a map of land use (area data). Vector features contain both spatial information (i.e., the location of the coordinates that define the feature) and attribute information (i.e., descriptors that define the feature).

Cell and raster data do not contain individual features. Instead, these data represent a grid of thematic information such as elevations, solid type categories, or a display color code. In the case of cell maps, each cell represents an area of a specified height and width. Each cell is given a value when it is created or manipulated. This value can be the minimum, maximum, or average value for a specific variable (e.g., elevation) within the cell or it can represent nominal data (e.g., land use). In either case, the entire cell is represented by that value. There are two types of cell maps in the GeoServer™: discrete, which can contain a limited number of values, and continuous, which can contain an infinite set of values.

Raster data consist of the raster maps and imagery displayed with GeoServerDisplay™ routines. These data are intended as background graphics to vector data and cannot be analyzed or manipulated.

## UTILIZING THE GEOSERVER™

To begin, the applications programmer must initialize the Server. This is accomplished by invoking the "c_Init" routine. The routine belongs to the GeoServerCore™ function group and has two arguments:

1) the Server's node name; and
2) the Server_Display_Flag.

The node name identifies the machine where the Server resides on the network, and the Server_Display_Flag determines whether the Server's graphics windows are to be utilized. The user must have the GeoServerDisplay™ installed for the Server_Display_Flag to be functional. If the Server_Display_Flag is set to "ON" (value =1), then the Server's graphics windows are initialized and displayed. The programmer can then invoke the GeoServerDisplay™ routines or utilize the Server's menuing system. If, however, the Server_Display_Flag is set to "OFF" (value =0), the Server's display windows are not initialized, and the programmer can neither invoke GeoServerDisplay™ routines nor use the Server's menuing system.

The second routine that must be invoked is "c_SelectArea" which selects the database area. This function has one argument: the name of the database area. If the Server_Display_Flag is "ON" and the GeoServerDisplay™ is installed, the minimum bounding rectangle of the database area will be displayed. In addition, if the programmer inputs a null string as the database area name, a menu of all database areas will appear within the Primary Graphics Window, and the programmer will be able to choose an area from the menu.

## GEOSERVERCORE™ ROUTINES

The GeoServerCore™ routines provide the functionality to initialize the Server, create spatial Templates, manage virtual memory, access the database, and disconnect from the Server.

### Spatial Figures (Templates)

The routines that create spatial figures, also known as Templates, allow the programmer to create geometric shapes of given size and shape. These Templates can be used as spatial filters. The programmer can query the database to determine the items that are "inside," "outside," "within" a given distance, or "beyond" a given distance of a Template.

The Template database will be updated if the programmer exits the system with the routine "c_NormExit." If the programmer exits with the routine "c_AbnormalExit," the Template creations and deletions that were done during the session with the Server will not be updated in the database.

### Virtual Memory

The GeoServerCore™ routines allow the programmer to store relational data in memory in a matrix format know as a virtual table. As with an RDBMS table, in a virtual table the columns represent field names, and the rows present individual records of data. A virtual table can have the identical fields as a table in the relational database or can be created to satisfy other programmer requirements.

The first step when using a virtual table is to allocate the specified table and obtain a reference table identification or ID. After obtaining the ID, the virtual table must be initialized by determining the number of columns, column names, column types (e.g., double precision, long integer, etc.), and column lengths (e.g., 2 or 4 bytes). Once the virtual table has been initialized, data can be added or modified; although the operator must proceed in row-order, and all columns must be populated within the given row before proceeding to the next row. Data can be retrieved from a virtual table as well.

### Relational Database Access

The GeoServerCore™ routines allow the analyst to add/update, retrieve, and delete data from the relational database. The adding and updating of the relational database must be done with a virtual table. With a routine, the programmer specifies a row in the virtual table to be inserted to a database table. For this to occur successfully, each field name and its respective type and length in the virtual table must match identically to a field in the specified relational database table. All fields in the relational database table are given a zero or null value for the newly inserted row.

Updating the relational database is similar to adding. Again, the programmer specifies a row in the virtual table to be used for the update to a database table. As with adding, for updating to occur successfully, each field name and its respective type and length in the virtual table must match identically to a field in the specified relational database table. In addition, the virtual table must have an "ID" field. The value in this field for the specified row will determine which record will be updated (i.e., the one that has the same ID) in the database table. All fields in the relational database table that do not have a corresponding field in the virtual table are left with their original values.

There are three routines that retrieve data from the relational database:

c_DbSelect;
c_Query; and
c_GeoQuery.

These three routines retrieve data based on an input query.

The routine "c_DbSelect" retrieves data for the relational database that satisfy a standard SQL command which begins with the word "SELECT." The data are placed in an initialized virtual table. The files in this table will be those requested in the command. For example, the command:

```
SELECT LATITUDE, LONGITUDE FROM DATA_
BASE_TABLE
```

will produce a virtual table with three columns: ID, LATITUDE, and LONGITUDE (the "ID" field is always added to a virtual table). The number of rows in the table equals the number of records in the relational database table, DATA_BASE_TABLE. The command:

```
SELECT * FROM DATA_BASE_TABLE
```

will produce a virtual table with the same files as the table, DATA_BASE_TABLE. The number of rows in the table equals the number of records in DATA_BASE_TABLE.

The routine "c_Query" allows the programmer to utilize the Server's GeoSQL language.

The routine "c_GeoQuery" works similarly to "c_Query" and uses the GeoSQL to retrieved data from the relational database. The output, however, is a file rather than a virtual table. In this manner, query results can be stored for subsequent processing by an application program.

## GEOSERVERMODELER™ ROUTINES

The GeoServerModeler™ routines provide the functionality to access cell map data including cell map values for a given geographic location and the creation/manipulation of Geosearch Templates.

### Obtaining a Cell Map Value

The GeoServerModeler™ allows the programmer to obtain a value from a cell map given an input latitude and longitude. The steps required consist of opening the cell map, describing the cell map, obtaining the cell value, and closing the cell map.

### Geosearch Templates

A Geosearch Template is a specific type of cell map where cell values range from zero to one. The value of a cell is a relative probability that a target or other point of interest will be at that location.

Two routines operate on Geosearch Templates. One routine extracts the polygons from a Geosearch Template in a cell-to-vector transformation. These polygons, or areas of interest, become intelligence features which are referenced by their minimum bounding ellipse. The second routine creates a new continuous Geosearch Template based on the content of an old template.

## GEOSERVERDISPLAY™ ROUTINES

The GeoServerDisplay™ routines provide the functionality to change the Graphics Window. The purposes of these routines are to display to, erase from, or modify the Graphics Window. In addition, these routines manage the display of Templates and retrieve information from the window.

## CONCLUSION

This paper describes the Autometric GeoServer™. Specifically, by implementing advanced query processing capabilities which utilize the significant strengths of GeoSQL database accessing and geographic data processing techniques, the GeoServer™ represents a significant development in the GIS arena. Accordingly, a variety of functional data can be screened or selected using well-focused subject and spatial criteria; therefore, the GeoServer™ supports a dynamic management of specific spatial and subject relationships. Finally, as a data management system, the GeoServer™ consists of three components: the GeoServerCore™, the GeoServerModeler™, and the GeoServerDisplay™. Each of these are discussed in detail with reference to GeoSQL.

## REFERENCES

Abel, D.J., and J.L. Smith, 1983. A Data Structure and Algorithm Based on a Linear Key for Rectangle Retrieval, *Computer Vision, Graphics, and Image Processing*, Vol. 24: No. 1, pp. 1-13.

Guting, R.H., 1988. Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems, *Proceedings International Conference on Extending Database Technology*, Venice, Italy, Springer-Verlag, pp. 506-527.

Menon, S., and T.R. Smith, 1989. A Declarative Spatial Query Processor for Geographic Information Systems. *Photogrammetric Engineering and Remote Sensing*, Vol. 55: No. 11, pp. 1593-1600.

Nilsson, N., 1980. *Principles of Artificial Intelligence*, Tioga Press, Palo Alto, California.