Spatial SQL

Querying Spatial Objects Based on Geographic Information

An Open Source Approach

Chieko MAENE
Maps & State Documents Librarian
Government and Geographic Information and Data Services
Northwestern University
c-maene@northwestern.edu

A Spatial Database for the Rest of Us

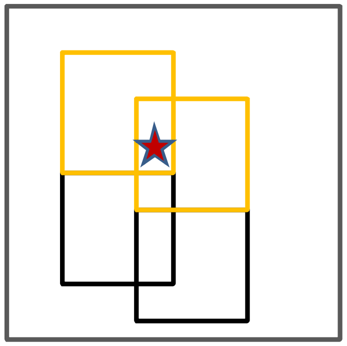
A spatial database allows users to make standard SQL queries and also geographical queries to access and analyze data. Currently , there are several spatial databases, or geo-databases, available on the market, including Oracle Spatial, IBM DB2 Spatial Extender , ArcSDE (ESRI) over proprietary RDBMS, PostgreSQL with PostGIS (Refractions Research) and MySQL with spatial extensions. Microsoft also recently announced that the next release of SQL Server will include spatial data support. This section focuses on an open source product, PostGIS, and its spatial functions.

PostGIS is an extension which adds spatial database capabilities to an open source object-relational database, PostgreSQL. PostGIS complies with the Open Geospatial Consortium (OGC, formerly known as OpenGIS) simple features for SQL specification, and has become the standard spatial database backend for all the other open source GIS tools, such as Mapserver as the project presented on the left shows.

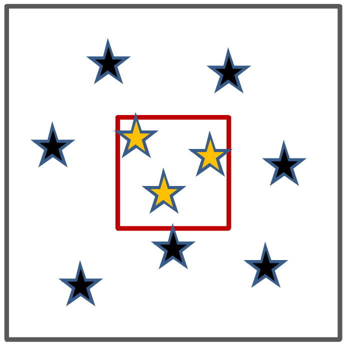
Although PostGIS has implemented a variety of OGC functions to manage, manipulate and create spatial data, new users may find only a few functions useful. For this session, only the most basic spatial relationship functions are examined with examples of the use in a library setting.

References:
PostGIS
http://postgis.refractions.net/
OGC Simple Feature Specification for SQL
http://www.opengeospatial.org/standards/sfs
Product Survey on Geo-databases
http://www.gim-international.com/files/productsurvey_v_pdfdocument_14.pdf

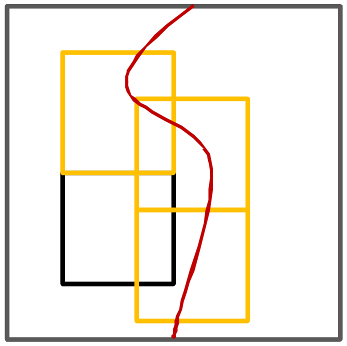
CONTAINS( )



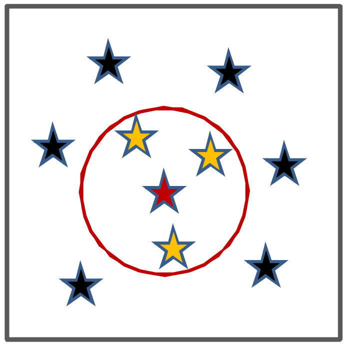
WITHIN( )



INTERSECTS( )



DISTANCE( )



CONTAINS( )

SELECT * FROM table **WHERE** **CONTAINS** (the_geom, object-a)

Returns 1 (TRUE) if the object spatially contains another object. For instance, the example above selects all objects from a table (table) that contains another object (object-a). Don't' confuse CONTAINS() with WITHIN() operator.. CONTAINS(A,B) is interpreted as "A contains B" which equals to "B is within A".

This type of query is useful in identifying spatial objects that contain a particular spot, such as a house or a construction site. For example, one may need maps and aerial photographs that contain a particular house. With electronically indexed map and air photo sheet information and a house address, one will be able to query the map and photo sheets that are available to study a specific location.

SELECT * FROM map_airphoto_sheets_table WHERE CONTAINS(the_geom, house_address)

WITHIN( )

SELECT * FROM table **WHERE** **WITHIN** (the_geom, object-a)

Returns 1 (TRUE) if the object is "spatially within" another object. For instance, the example above selects all objects that are spatially within, or are completely contained by, another object (object-a).

This type of query is useful in finding spatial objects whose centroid falls in a particular area (or areas) such as a city of a neighborhood. For example, one may need to find photographs of historic buildings in a particular neighborhood. By recording an address of such buildings and then converting the addresses to geographic coordinates in GIS and also storing neighborhood boundary information in a database, one will be able to query the buildings by naming neighborhoods.

SELECT * FROM historic_buildings WHERE WITHIN(the_geom, neighborhood)

INTERSECTS( )

SELECT * FROM table **WHERE** **INTERSECTS** (the_geom, object-a)

Returns 1 (TRUE) if the object spatially intersects another object. For instance, the example above selects all objects that intersect (i.e. touches or overlays) another object (object-a).

This type of query is useful in selecting spatial objects that contain a linear feature object, such as a river. For example, one may need to find maps or aerial photographs covering a specific river. In such a case, use the river as another object (object-a) to select maps and aerial photograph sheets that the river touches or intersects.

SELECT * FROM map_airphoto_sheets_table WHERE INTERSECTS(the_geom, the river)

DISTANCE( )

SELECT * FROM table **WHERE** **DISTANCE** (the_geom, object-a) < 2000

Returns the Cartesian distance between two objects in projected units. For instance, the example above selects all objects from a table (table) which are within 2,000 projected units of an object (object-a).

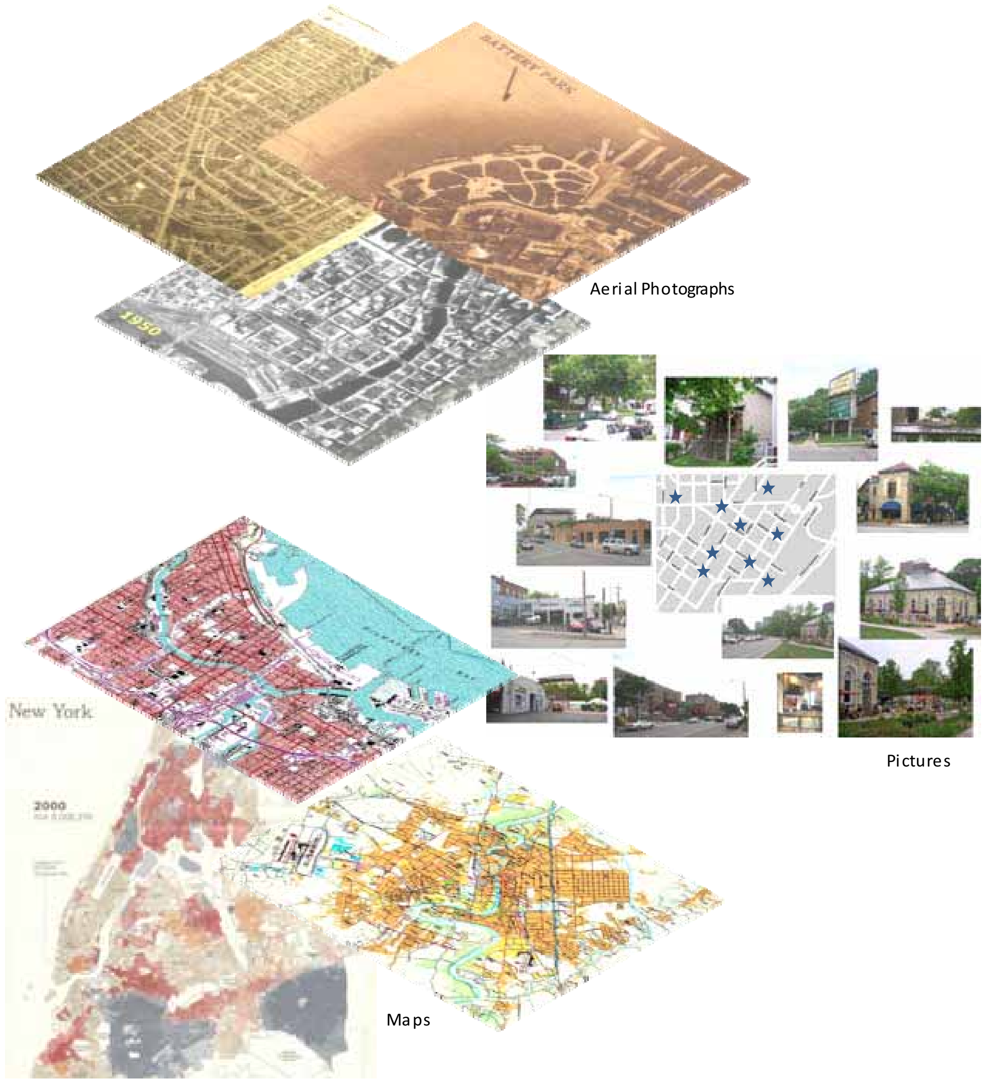
This type of query is useful in finding spatial objects near a specific location. For example, one may need to identify cultural institutions such as museums and libraries near one's house. If geographic locations of such institutions are recorded in a database, finding them by distance criteria is easy. Also, if descriptions of each institution are available, setting up more complex criteria to narrow the search result will be possible.

SELECT * FROM institutions_table WHERE DISTANCE(the_geom, the_location) < 1234 (distance in a projected unit)

Many items in libraries are geographic objects, or objects with spatial information such as place name or street address. Examples include historic photographs, aerial photographs, topographic and fire insurance maps. However, the ways in which most libraries locate information are somehow limited to analog tools, such as catalog records, index cards and index maps. Wouldn't it be better if we could find these objects based on spatial information?

The popularity of mapping sites such as Google Maps indicates that Internet users are comfortable with finding geographic objects by simply typing a street address.

Libraries can implement the same idea. This project introduces the concept of "spatial queries" using open source software, PostGIS and GeocoderUS. A combination of these tools enables users to find photographs and maps by querying an object-relational database with spatial information such as a street address or intersection. The query returns objects that have a proximity or spatial relationship to the input information. The project result is a "spatially enabled" SQL data server which supports such queries as, "find and show me photographs near this location" or "find and show me maps that contain this location."

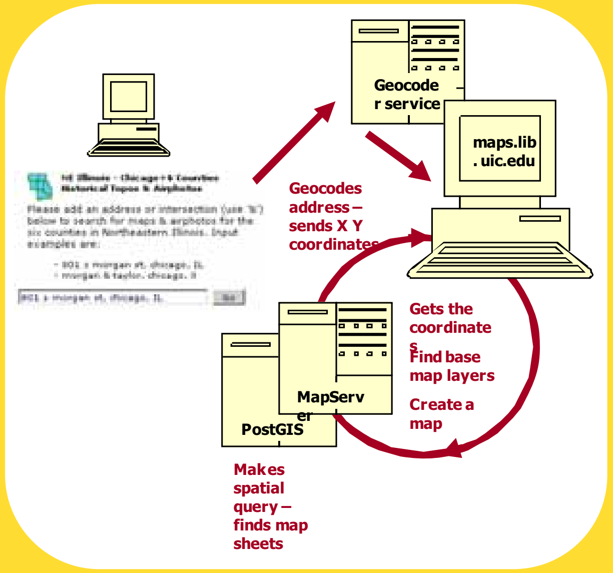


IMPLEMENTATION EXAMPLE – MAPS & AIRPHOTOS FINDER

http://maps.lib.uic.edu

1. IDEA

The initial idea for this project came from our topographic maps and aerial photograph users who needed to find the right sheets that cover their study area typically described with a street address or a street intersection. Their information search would be easy if all the map series and aerial photograph sets used the same indexing system. In reality this is not the case. Maps and air photos are organized in various different ways due to their various sizes and formats and also their historical nature, and thus it makes pulling the information difficult.



NE Illinois – Chicago + 6 Counties Historical Topos & Airphotos

Please add an address or intersection (use '&') below to search for maps & airphotos for the six counties in Northeastern Illinois. Input examples are:

- 801 s morgan st, chicago, IL
- morgan & taylor, chicago, il

801 s morgan st, chicago, IL [Go]

2. SOLUTION

The solution to this is GIS. Once the indexing information is stored in an electronic database, GIS will enable users to find the spatial objects they need by making a simple query like "find all photographs that fall in (or near) this location." Next, we would make the query system and spatially indexed objects accessible via the Internet so that users will benefit from GIS without having GIS software on their computers.

The main components used in the project are (diagram on the left):

1) Web server, Microsoft IIS,
2) Data server, PostgreSQL with PostGIS extension,
3) Online mapping software, UMN MapServer,
4) Address-to-coordinates conversion software, GeocoderUS plus Census TIGER database in BerkeleyDB,

all of which are open source except for a web server (IIS – comes with Windows XP.)

Note: the project could be downsized to just two main components (web & data servers) to create a simple online spatial query system without mapping component.

3. IMPLEMENTATION

The first step to create a spatial query system is creating spatial objects (map and aerial photo indexes) in GIS and then saving them in a database. For the project, metadata (i.e. series name, sheet name, index number, published year, publisher, etc.) as well as geographic coordinates of all the map and aerial photograph sheets were stored in the GIS shapefile format Then the data were imported to a spatial data management database system, PostgreSQL. With a combination of a spatial indexing tool, PostGIS, this database can accept both spatial (i.e. data with geographic coordinates) and non-spatial objects (i.e. data tables) and also query and manipulate the objects using the SQL interface.

Once a database is created, objects can be accessed, retrieved and presented by a series of queries and requests over the web using various web server-side scripting languages, such as Perl, and PHP. Examples of spatial queries are presented on the right.

A map server such as UMN Mapserver used in this project, or a mapping service such as Google Maps will be needed to graphically present the retrieved objects in space, or on a map.

Acknowledgment: Maps and aerial photographs finder project (maps.lib.uic.edu) was supported by and implemented at the University of Illinois at Chicago Library.