Communications of the IIMA

Volume 9 | Issue 3 Article 8

2009

Geocoding Data Analysis and Processing in Relational Databases

Jiangping Wang
Webster University

Geocoding Data Analysis and Processing in Relational Databases

Jiangping Wang
Webster University
USA
wang@webster.edu

ABSTRACT

Geocoding data are widely used in applications to support geographical locations and compute spatial relationships, such as distances. It is a common practice to capture geospatial data in relational database management systems (RDBMS), where most business data are stored and processed. However, RDBMS's lack of ability in geocoding analysis and processing has prevented applications from handling spatial data effectively. Relational databases need to support geocoding data analysis and processing, such as geocoding data sorting, searching, geo-positioning, and distance computing. More advanced requirements may include the analysis of an area based on geo-location and area coverage of a geo-location. This paper provides analysis of some of the requirements on geocoding data processing in RDBMS from business application perspectives. The challenges in design and implementation will be discussed. The requirements will be further transformed to modeling, analysis, and processing on geocoding support in relational databases. When database volume increases, performance issues need to be addressed in order to provide efficient data queries and data processing. Database objects, such as auxiliary tables, views, and indexes, can be designed to provide performance improvement.

INTRODUCTION

Geocoding data are geospatial coordinates that are used to identify location points on the earth. Geocoding data, especially physical location latitude and longitude data, become part of business data when businesses collect information for their decision making purpose. Business applications rely on data that are commonly stored and processed by relational database management systems (RDBMS). As part of business data, geocode, namely latitudes and longitudes, are incorporated more and more into applications' data processing to represent business locations and to compute distances from each others.

As part of the data structure, geocoding coordinates can be stored in relational databases same as other business data fields to participate in data processing, such as updating, sorting, computing, as well as complex querying. RDBMS is a database system that maintains and manipulates data repository based on relational model (Codd, 1970). Through database objects, such as tables and views, relational databases organize data records into rows and columns. Therefore latitude and longitude data can be easily fit into any data attribute structure. Relational database is designed to maintain data integrities by using a variety of database constraints, such as primary keys and foreign keys. Data integrities, such as entity integrity and referential integrity, are enforced to comply with any business rules in database (Date, 2004).

Geocoding data, and the way to process them, have their unique characteristics. Latitude and

longitude data deserve special care from inside relational databases to reflect their computation intensive nature and to keep up with the database performance. Business applications can be degraded if the response time takes too long under low performance. There exists a variety of challenges in dealing with spatial data in relational databases, ranging from data formatting, sorting, transforming, computing, as well as storing. This paper discusses some of the issues in the process of using geocode as part of business data. The discussion starts from data analysis based on the business application requirements. Challenges to fulfill design requirements in relational databases will be addressed, followed by computational algorithms and implementation designs. Performance issues will be discussed under special relational design to minimize the computation process in responding to application queries.

REQUIREMENT ANALYSIS

Geocoding information is part of business data. Locations, such as customer location, business location, shipping address, contact address, are collected and stored by almost all applications that deal with business data. Accurate location information, such as street address, city, ZIP code, and state, are crucial in locating and contacting the given party. Geocoding data, latitudes and longitudes, are frequently part of location data to represent geographical location of the concerned data entity. Nowadays, geocoding process can be easily used to obtain geocoding coordinates. Geocode-enabled data can be easily incorporated into geographic information system (GIS) applications. Maps representing specified locations are also available through geocoding data. Latitude and longitude data can be provided by client directly or be retrieved based on location addresses collected. For example, public available web services, such as Google and Yahoo, can be invoked by feeding raw physical address to obtain geocoding data. The geocoding data can then be used, for example, to retrieve location map and calculate distances under certain conditions.

Geocoding data have to be collected and stored in the way that they can be easily queried to support data manipulations. Latitude and longitude coordinates can be represented in different formats, such as degree-minute-second format and decimal degree format. In order to be used in a relational database and minimize dynamic conversion, coordinates need to be stored in a single format. Unified format storage facilitates data ordering and sorting, which is one of the most frequently performed query operations. One requirement, for example, is to order query results by direction, such as from east to west.

Storing data is only the first step of using geocoding data. The spatial distance among locations can be further required to provide proximities. On the earth's surface, the spatial distance, or geo-distance, can be defined as the sphere distance with shortest path, which is the curve of great-circle distance (Wikipedia, 2009a). The spatial distance can be used to measure areas from any location point, such as a physical address or a metro area. The business rule can be the inquiry of retrieving all businesses within certain distance for a given metro area, or all businesses within certain distance for a given address. Database records will be queried to meet these distance criteria. To obtain geo-distance, calculations will be involved by providing geocoding data to the selected computation algorithms. Built-in functions are available in most relational database systems to support basic math calculations. However, specific user-defined functions have to be implemented to provide geocode related computations.

Business requirements can be further extended to question the area that is covered by a route that is drawn, or defined, by two geocoding locations. The query may inquire all data entries located along the line from geocoding point A to geocoding point B within certain distance. This requires the calculation of the distance from a given point to the line, the arc line on the surface of the earth, defined by the two locations based on the great-circle distance. The route can be defined by a real route, such as the one following interstate highway exits. The requirements can ask for all locations along a given interstate highway within certain distance from all highway exits. To fulfill the computation in this scenario, not only specially designed functions have to be involved, but the way to deal with the calculation intensity will play a critical role.

CHALLENGES

Any relational database system can host geocoding data, such as latitude and longitude. Two columns can be created in a database table to capture the two attributes. In order to keep its original precision, the coordinates can be stored as a variable character data type. Optionally, numerical data types, such as decimals, can be used, however the precision may be lost during transferring and processing. Variable character data type also provides an ideal domain to encompass different geocoding representation formats, such as degree-minute-second (DMS) format (e.g. 38°42’46”, -90°37’47”) and decimal degree (DD) format (e.g. 38.712652, -90.629659). No matter what format is used in database storage, data conversion and transformation will be necessary for processing and manipulation. The data type of storage has to be transformed to the data type for other calculations, where suitable data format is selected to maintain computing precision and calculation efficiency.

Relational database systems are not optimized to store and retrieve data related to objects in space as spatial databases do. Therefore, there are no built-in functions for supporting the basic spatial data measurements and calculations. For example, functions that support finding the distance between points or querying data related to the center of a circle have to be implemented to meet business requirements. A database function has to be designed and implemented to accept geocoding coordinates for two locations and return the distance for the selected algorithm. One more function is required to calculate distance from a geocoding point to a route line defined by location A and location B. The function will have to take three latitude/longitude points where two of them define the route and one is used to calculate the distance.

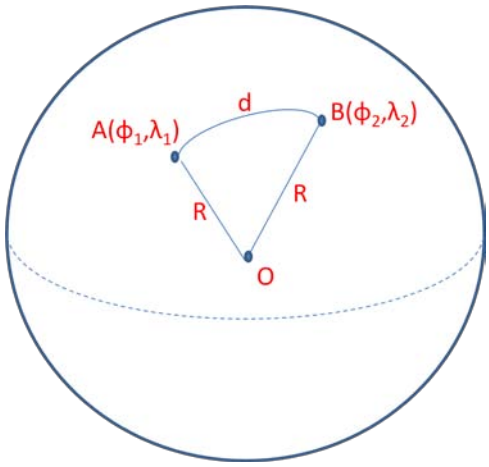
The above mentioned basic calculations can be modeled by mathematical formulae from geographical algorithms. Considerations in selecting algorithms include data involved in calculation, accuracy, complexity, data presentation, as well as efficiency. When dealing with a large amount of application data, however, performance becomes the real challenge. If the calculation of one measurement will be iterated thousands, or even millions, of times, the performance improvement will be the top priority of the process. In database environment, timely response to user queries is vital, which is among one of the key factors for the application to be successful.

ALGORITHMS

The very basic computation for geocoding latitude/longitude data is the calculation of distance for any two geocoding locations on the earth. There are a number of sphere formulae, such as great-circle distance equation, that can be used. The great-circle distance is the shortest distance between two geo-locations on the surface of a sphere. The Haversine formula (Wikipedia, 2009a; Wikipedia, 2009b; Sinnott, 1984) is preferred for common cases to minimize rounding errors. It assumes a spherical earth and ignores ellipsoidal effects. For the calculation of most business geocoding data, its accuracy is satisfactory.

In order to compute the distance from point A to point B, as demonstrated in Figure 1, the following algorithm can be implemented.

Figure 1: Distance from point A to point B on a sphere.



Given: two points A (φ_1, λ_1) and B (φ_2, λ_2), sphere radius R, and distance (between A and B) d,

$$haversin\left(\frac{d}{R}\right) = haversin(\Delta\varphi) + \cos\varphi_1 \cos\varphi_2 haversin(\Delta\lambda)$$

Where: φ_1 is the latitude of A, λ_1 is the longitude of A, φ_2 is the latitude of B, λ_2 is the longitude of B, $\Delta\varphi = \varphi_2 - \varphi_1$, $\Delta\lambda = \lambda_2 - \lambda_1$, and $haversin(\theta) = \sin^2\left(\frac{\theta}{2}\right)$. Then, d can be obtained by:

$$d = 2R \times \arcsin\left(\sqrt{haversin(\Delta\varphi) + \cos\varphi_1 \cos\varphi_2 haversin(\Delta\lambda)}\right)$$

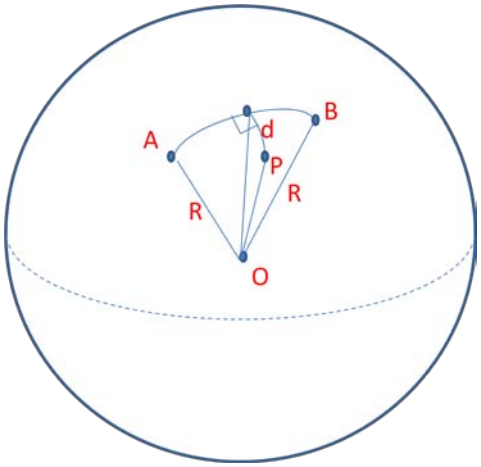
And then

$$d = 2R \times \arcsin\left(\sqrt{sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos\varphi_1 \cos\varphi_2 sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Although above formula is only an approximation for calculation on earth, its accuracy is sufficient for common cases for the involved application.

Another basic calculation for the distance (d) from a geo-point (P) to a great-circle route (A-B) can be obtained using the similar algorithms (McGovern, 2004; Ersts, Horning, & Polin, 2009) as depicted in Figure 2.

Figure 2: Distance from point P to a line segment of A and B.



DATABASE IMPLEMENTATIONS

A database user-defined function is a routine that accepts parameters, performs the calculation defined by algorithms, and returns a result. User-defined functions are stored in the database and can be invoked in the same way as any built-in functions. A user-defined database function *fn_latlondistance* has been implemented in SQL Server relational database, which takes two geo-location points and returns the distance based on the algorithm discussed in the previous section.

The following is the function interface:

```
CREATE FUNCTION fn_latlondistance
   (@lat1 float, @lon1 float, @lat2 float, @lon2 float)
RETURNS float AS
BEGIN
   --function implementation;
END
```

Similarly, function *fn_latlonpointsegmentdistance* has been implemented, which takes three geo-location points and returns the distance from the first point to the line defined by the last two point coordinates:

```
CREATE FUNCTION fn_latlonpointsegmentdistance
   (@px_lon decimal(18,14), @py_lat decimal(18,14),
    @x1_lon decimal(18,14), @y1_lat decimal(18,14),
    @x2_lon decimal(18,14), @y2_lat decimal(18,14))
RETURNS decimal(18,14) AS
```

```
BEGIN
   --function implementation;
END
```

Functions can be used in SQL queries in different ways. The simplest use of distance functions in query can be demonstrated in the following query situations. For a metro area search, the SQL query can be performed under the following WHERE clause to calculate all records that the distance is within 25 miles of St. Louis metro area:

```
WHERE {other criteria} AND (dbo.fn_latlondistance(latitude,
longitude, 38.6211623464225, -90.19775390625) <= 25);
```

To obtain the proximity towards a certain geocoding location, calculations on all records that the distance is within 10 miles of the ZIP code 63119 can be invoked with the following WHERE clause:

```
WHERE {other criteria} AND (dbo.fn_latlondistance(latitude,
longitude, 38.588120, -90.351265) <= 10);
```

The above examples show that the center points are specified by end user for a geo-location, such as St. Louis metro area (*38.6211623464225, -90.19775390625*) or ZIP code 63119 (*38.588120, -90.351265*), whereas each record's latitude and longitude stored in database table columns, which varies from record to record, are used as arguments in calling the functions. The function return participate further comparison to fulfill query requirements.

Similarly, the function for distance from a point to a line segment can be used to query records based on the proximity to the route defined by two pairs of geocoding coordinates. The great-circle line, or shortest path, in the following query criteria is defined from St. Louis, Missouri to Pittsburgh, Pennsylvania. The query will cover the band defined by the two location points and a distance of 5 miles from the line.

```
WHERE {other criteria} AND
(dbo.fn_latlonpointsegmentdistance(longitude, latitude, -90.198954,
38.627610, -79.997459, 40.438310) <= 5);
```

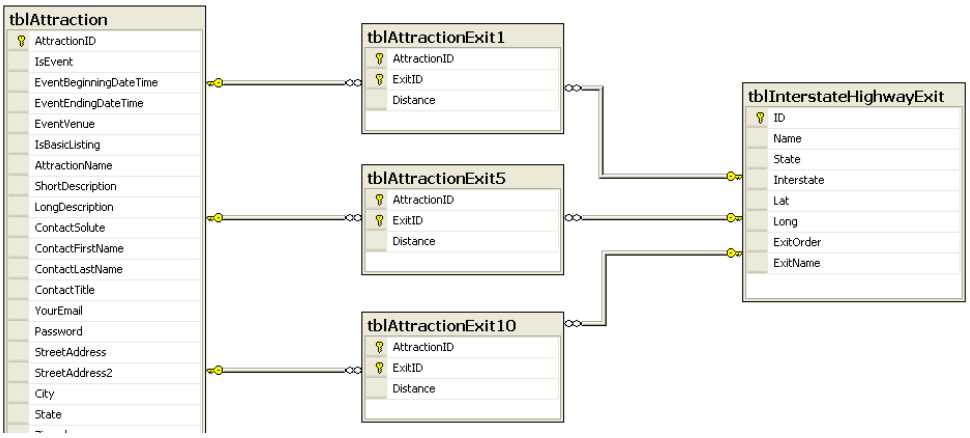
PERFORMANCE ISSUES

In querying database records, multiple records in the related tables will be searched to test against the query criterion, which is indicated by record attribute names (*latitude, longitude*). Therefore, the basic function calculations will have to iterate through each record performing computation and qualify the record. In large volume database, this intensive computation will cost a large amount of processing time, which would slow down the query performance. The situation becomes even worse when queries are performed against interstate highway exits. For instance, one of the query requirements is to locate all businesses within 5 miles along highway Interstate 270 exits in St. Louis metro area (total of 27 exits). If the query was for Interstate 70,

there would be 665 exits involved. It would take too long to get the results back to respond to user's query.

To address performance issues, database can be redesigned to include data preprocessing. Geo-distance calculations are performed when data are uploaded and the calculation results are stored in newly created tables. There are three targeted distance ranges that have to be supported by the system, 1, 5, and 10 miles from each highway exit. Therefore, three auxiliary database tables are created for each of the distance ranges to host business identifiers and highway exit identifiers according to the preprocessing results. Taking advantage of database relationships and referential integrities, the auxiliary tables are linked to raw interstate exit table, as well as the geocode location table. The search can quickly find out which geocoding coordinates are related to which exits, as well as in which distance range, as depicted in Figure 3.

Figure 3: Relationships between business locations and interstate exits.



A database view can be implemented to retrieve data from the three linked tables. In a database, a view is a logical table which is not physically used to store data. A view, through the implementation of the underlying query, retrieves data from defined database tables. The view to support interstate exits is implemented with union of three linked tables, as listed below, and a sample subset of data from the view is show in Figure 4. This implementation of the auxiliary tables and the view has greatly improved the query performance for interstate highway exits distance queries by removing intensive computation, especially for the situations where large number of highway exits are involved in calculations.

```
CREATE VIEW dbo.vw_AttractionExit
AS
   SELECT *, '1 mile' AS MaxDistance FROM tblAttractionexit1
   UNION
   SELECT *, '5 miles' AS MaxDistance FROM tblAttractionexit5
   UNION
   SELECT *, '10 miles' AS MaxDistance FROM tblAttractionexit10;
```

Figure 4: Subset of geocode exit view.

AttractionID	ExitID	Distance	MaxDistance
7737	1	8.60853	10 miles
17116	1	4.51011	5 miles
78116	1	7.94997	10 miles
99069	1	9.40287	10 miles
17116	2	0.851226	1 mile
75335	2	7.65077	10 miles
76578	2	7.18918	10 miles
76579	2	7.18918	10 miles
77484	2	8.97202	10 miles
97866	2	7.57819	10 miles
76318	2	9.48099	10 miles
17116	3	8.6876	10 miles
17188	3	8.69788	10 miles
17189	3	2.77053	5 miles
17191	3	3.41524	5 miles
17192	3	2.457	5 miles

In addition to auxiliary tables, indexes can be created on the tables and the views to improve the query performance. Depending on query needs, indexes can be created on one or more database columns to speed up data searching and efficient access to records.

LIMITATIONS

Business requirements have been met by the above design and implementation to support geocoding data processing. There are still some issues that need to be addressed to further facilitate the ever growing business requirements. Currently the system supports only geocoding data generated by addresses in the United States. The globalization of the application can certainly reveal limitations in calculation accuracy and storage data type. For example, variable character data type for latitude and longitude fields may need change to nvarchar (national variable character) to support Unicode character set. Another issue is the proximity route definition. Currently the route from a point to another point is defined either by a great-circle path or by multiple geo-points, such as points for interstate exits, on the route. More complex algorithms have to be implemented to provide accurate distance along the actual natural route. Further study and experiments on these issues are to be conducted.

CONCLUSION

Relational database systems are designed to support relational data models through the enforcement of both entity and referential integrities. Relational databases have become the dominant choice for storing and processing business information. As part of the business information, location addresses, as well as geocoding coordinates, are to be processed by relational databases. Therefore, spatial data processing support in relational databases is an important area contributing to the success of database driven applications. The approach in

geocoding data analysis and processing discussed in this paper is an attempt towards this direction. The design and implementation have been tested against real world business requirements and will be improved to suit further growing business needs.

REFERENCES

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), 377–387.

Date, C. J. (2004). *An Introduction to Database Systems*, Chapter 9 Integrity, Pearson Education, Inc.

Ersts, P. J., Horning, N., & Polin, M. (2009). Perpendicular Distance Calculator (version 1.2.2) Documentation. American Museum of Natural History, Center for Biodiversity and Conservation. Retrieved March 29, 2009 from http://biodiversityinformatics.amnh.org/open_source/pdc/documentation.php

McGovern, A. (2004). Geographic Distance and Azimuth Calculations. Retrieved March 29, 2009 from http://www.codeguru.com/Cpp/Cpp/algorithms/article.php/c5115/

Sinnott, R. W. (1984). *Virtues of the Haversine, Sky and Telescope,* 68(2), 159.

Wikipedia (2009a). Great-Circle Distance. Retrieved March 29, 2009 from http://en.wikipedia.org/wiki/Great-circle_distance

Wikipedia (2009b). Haversine Formula. Retrieved March 29, 2009 from http://en.wikipedia.org/wiki/Haversine_formula

This Page Left Intentionally Blank