Geo (proximity) Search with MySQL

Alexander Rubin
Senior Consultant, MySQL AB
Why Geo Search?

- Stores: find locations near you
- Social networks: find friends close to you
- Online maps: find points of interest near your position
- Online newspapers/yellow pages: find show times next to you home.
POI Search Example

Refine by: Distance | Cuisine | User Rating

Results 1-10 of about 8,690 for pizza near 5001 Great America Pkwy, Santa Clara, CA 95054
Categories: Pizza Restaurants, Pizza Restaurants, Round Table

A. Round Table Pizza Santa Clara
   4300 Great America Pkwy #0000, Santa Clara, CA
   (408) 970-9000 - 1 review - 0.7 mi S

B. Tomatina
   3127 Mission College Blvd, Santa Clara, CA
   (408) 654-9000 - 5 reviews - 1.1 mi S
   Category: Pizza

C. Little Caesars Pizza
   4767 Lafayette St, Santa Clara, CA
   (408) 496-1393 - 0.9 mi E

D. Giovanni's Pizzaria
   1127 N Lawrence Expwy, Sunnyvale, CA
   (408) 734-4221 - 3 reviews - 1.1 mi W
   Category: Pizza

E. Pizza Depot
   919 E Duane Ave, Sunnyvale, CA
   (408) 245-7760 - 5 reviews - 1.7 mi SW

F. Round Table Pizza San Jose
   3730 N 1st St, San Jose, CA
   (408) 321-9922 - 1.8 mi E
Common Task

- **Task:** Find 10 nearby hotels and sort by distance

- **What do we have:**
  - Given point on Earth: **Latitude, Longitude**
  - Hotels table:  
    | Hotel Name | Latitude | Longitude |

- **Question:** *How to calculate distance between us and hotel?*
Latitudes and Longitudes

(30° N. Latitude, 90° W. Longitude)
Distance between 2 points
The Haversine Formula

For two points on a sphere (of radius $R$) with latitudes $\phi_1$ and $\phi_2$, latitude separation $\Delta \phi = \phi_1 - \phi_2$, and longitude separation $\Delta \lambda$ the distance $d$ between the two points:

$$\text{haversin}\left(\frac{d}{R}\right) = \text{haversin}(\Delta \phi) + \cos(\phi_1) \cos(\phi_2) \text{haversin}(\Delta \lambda)$$

$$\text{haversin}(\theta) = \frac{\text{versin}(\theta)}{2} = \sin^2\left(\frac{\theta}{2}\right)$$

$$\text{versin}(\theta) = 1 - \cos(\theta) = 2 \sin^2\left(\frac{\theta}{2}\right)$$
The Haversine Formula in MySQL

\[ R = \text{earth's radius} \]

\[ \Delta \text{lat} = \text{lat}_2 - \text{lat}_1; \quad \Delta \text{long} = \text{long}_2 - \text{long}_1 \]

\[ a = \sin^2(\Delta \text{lat}/2) + \cos(\text{lat}_1) \times \cos(\text{lat}_2) \times \sin^2(\Delta \text{long}/2) \]

\[ c = 2 \times \text{atan2}(\sqrt{a}, \sqrt{1-a}); \quad d = R \times c \]

\[ 3956 \times 2 \times \text{ASIN} \left( \sqrt{ \\text{POWER} \left( \sin \left( \frac{\text{orig.lat} - \text{dest.lat}}{180} \times \pi / 2 \right) \right), 2 \right) + \cos(\text{orig.lat} \times \pi / 180) \times \cos(\text{dest.lat} \times \pi / 180) \times \text{POWER} \left( \sin \left( \frac{\text{orig.lon} - \text{dest.lon}}{180} \times \pi / 2 \right) \right), 2 \right) \] as distance

*angles need to be in radians*
MySQL Query: Find Nearby Hotels

set @orig_lat=121.9763; set @orig_lon=37.40445;
set @dist=10;

SELECT *, 3956 * 2 * ASIN(SQRT(
POWER(SIN((@orig_lat - abs(dest.lat)) * pi()/180 / 2),
2) + COS(@orig_lat * pi()/180 ) * COS(abs(dest.lat) *
pi()/180) * POWER(SIN((@orig_lon - dest.lon) *
pi()/180 / 2), 2) )) as distance
FROM hotels dest
having distance < @dist
ORDER BY distance limit 10\G

Lat can be negative!
### Find Nearby Hotels: Results

<table>
<thead>
<tr>
<th>hotel_name</th>
<th>lat</th>
<th>lon</th>
<th>dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hotel Astori..</td>
<td>122.41</td>
<td>37.79</td>
<td>0.0054</td>
</tr>
<tr>
<td>Juliana Hote..</td>
<td>122.41</td>
<td>37.79</td>
<td>0.0069</td>
</tr>
<tr>
<td>Orchard Gard..</td>
<td>122.41</td>
<td>37.79</td>
<td>0.0345</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10 rows in set *(4.10 sec)*

*• 4 seconds - very slow for web query!*
MySQL Explain query

```
Mysql> Explain ...  
select_type: SIMPLE  
    table: dest  
        type: ALL  
possible_keys: NULL  
            key: NULL  
            key_len: NULL  
            ref: NULL  
            rows: 1787219  
        Extra: Using filesort  
1 row in set (0.00 sec)
```
How to speed up the query

• We only need hotels in 10 miles radius – no need to scan the whole table
How to calculate needed coordinates

• 1° of latitude ~= 69 miles
• 1° of longitude ~= cos(latitude)*69

• To calculate lon and lat for the rectangle:
  
  set lon1 = mylon -
  dist/abs(cos(radians(mylat))*69);
  set lon2 = mylon
  +dist/abs(cos(radians(mylat))*69);
  set lat1 = mylat - (dist/69);
  set lat2 = mylat + (dist/69);
Modify the query

```
SELECT destination.*,
3956 * 2 * ASIN(SQRT( POWER(SIN((orig.lat - dest.lat) *
    pi()/180 / 2), 2) +
    COS(orig.lat * pi()/180) * COS(dest.lat * pi()/180) *
    POWER(SIN((orig.lon -dest.lon) * pi()/180 / 2), 2) )) as distance
FROM users destination, users origin
WHERE origin.id=userid
and destination.longitude
between lon1 and lon2
and destination.latitude
between lat1 and lat2
```
Speed comparison

• Test data: Users and coordinates
  – (id, username, lat, lon)

• Original query (full table scan):
  – 8 seconds

• Optimized query (stored procedure):
  – 0.06 to 1.2 seconds (depending upon the number of records in the given radius)
**Stored procedure**

```sql
CREATE PROCEDURE geodist (IN userid int, IN dist int) 
BEGIN

declare mylon double; declare mylat double;
declare lon1 float; declare lon2 float;
declare lat1 float; declare lat2 float;

-- get the original lon and lat for the userid
select longitude, latitude into mylon, mylat from users
where id=userid limit 1;

-- calculate lon and lat for the rectangle:
set lon1 = mylon-dist/abs(cos(radians(mylat))*69);
set lon2 = mylon+dist/abs(cos(radians(mylat))*69);
set lat1 = mylat-(dist/69); set lat2 = mylat+(dist/69);
```

Stored Procedure, Contd

-- run the query:
SELECT destination.*,
3956 * 2 * ASIN(SQRT(POWER(SIN((orig.lat - dest.lat) * pi()/180 / 2), 2) +
COS(orig.lat * pi()/180) * COS(dest.lat * pi()/180) *
POWER(SIN((orig.lon - dest.lon) * pi()/180 / 2), 2) ) ) as
distance FROM users destination, users origin
WHERE origin.id=userid
and destination.longitude between lon1 and lon2
and destination.latitude between lat1 and lat2
having distance < dist ORDER BY Distance limit 10;
END $$
Stored Procedure: Explain Plan

Mysql>CALL geodist(946842, 10)\G

table: origin
  type: const
  key: PRIMARY
  key_len: 4
  ref: const
  rows: 1, Extra: Using filesort

table: destination
  type: range
  key: lat_lon
  key_len: 18
  ref: NULL
  rows: 25877, Extra: Using where
Geo Search with Sphinx

- Sphinx search (www.sphinxsearch.com) since 0.9.8 can perform geo distance searches.
- It is possible to setup an "anchor point" in the api code and then use the "geodist" function and specify the radius.
- Sphinx Search returns in 0.55 seconds for test data regardless of the radius and zip.

```
$ php test.php -i zipdist -s @geodist,asc Query '' retrieved 1000 matches in 0.552 sec.
```
Speed comparison of all solutions

- Original MySQL query: 8
- Stored Procedure: large range: 1.2
- Stored Procedure: small range: 0.06
- Sphinx Search: 0.55

The graph compares the performance of different database solutions.
Different Type of Coordinates

• Decimal Degrees (what we used)
  – 37.3248 LAT, 121.9163 LON

• Degrees-minutes-seconds (used in most GPSes)
  – 37°19′29″N LAT, 121°54′59″E LON

• Most GPSes can be configured to use Decimal Degrees

• Other
Converting between coordinates

- Degrees-Minutes-Seconds to Decimal Degrees:
  - degrees + (minutes/60) + (seconds/3600)

CREATE FUNCTION `convert_from_dms`
  (degrees INT, minutes int, seconds int)
RETURNS double DETERMINISTIC
BEGIN
  RETURN degrees + (minutes/60) + (seconds/3600);
END $$

mysql>select convert_from_dms (46, 20, 10) as DMS\G
dms: 46.33611111
Geo Search with Full Text search

• Sometimes we need BOTH geo search and full text search
• Example 1: find 10 nearest POIs, with “school” in the name
• Example 2: find nearest streets, name contains “OAK”
• Create FullText index and index on LAT, LON
  – Alter table geonames add fulltext key (name);
  – MySQL will choose which index to use (boolean mode)
Geo Search with Full Text search: example

- Grab POI data from www.geonames.org, upload it to MySQL, add full text index

MySQL> SELECT destination.*,
3956 * 2 * ASIN(SQRT(POWER(SIN((orig.lat - dest.lat) * pi()/180 / 2), 2) +
COS(orig.lat * pi()/180) *
COS(dest.lat * pi()/180) *
POWER(SIN((orig.lon - dest.lon) * pi()/180 / 2), 2) ) as distance
FROM geonames destination
WHERE match(name)
against (‘OAK’ in boolean mode)
having distance < dist ORDER BY Distance
limit 10;
Geo Search with Full Text search: Explain

mysql> explain SELECT destination.*,
3956 * 2 * ASIN(SQRT(POWER(SIN(...

table: destination
type: fulltext
possible_keys: name_fulltext
key: name_fulltext
key_len: 0
ref:
rows: 1
Extra: Using where; Using filesort
CREATE TABLE `zipcode_spatial` ( 
`id` int(10) unsigned NOT NULL AUTO_INCREMENT, 
`zipcode` char(7) NOT NULL, ... 
`lon` int(11) DEFAULT NULL, 
`lat` int(11) DEFAULT NULL, 
`loc` point NOT NULL, 
PRIMARY KEY (`id`), 
KEY `zipcode` (`zipcode`), 
SPATIAL KEY `loc` (`loc`) 
) ENGINE=MyISAM;
Zipcode with Spatial Extension

```sql
mysql> select zipcode, lat, lon, AsText(loc) from zipcode_spatial
where city_name = 'Santa Clara'
and state = 'CA' limit 1

****** 1. row***********
zipcode: 95050
  lat: 373519
  lon: 1219520
AsText(loc): POINT(1219520 373519)
```
Spatial Search: Distance

Spatial Extension: no built-in distance function

CREATE FUNCTION `distance`
(a POINT, b POINT)
RETURNS double DETERMINISTIC
BEGIN
RETURN round(glenghth(linestringfromwkb
(linestring(asbinary(a),
asbinary(b)))));
END $$

(forge.mysql.com/tools/tool.php?id=41)
Spatial Search Example

```
SELECT DISTINCT
  dest.zipcode,
  distance(orig.loc, dest.loc) as sdistance
FROM
  zipcode.spatial orig,
  zipcode.spatial dest
WHERE
  orig.zipcode = '27712'
HAVING sdistance < 10
ORDER BY
  sdistance limit 10;
```