Jerish Balakrishnan  
  
Date:19th December 2024

**Problem Statement:**

I have a database with 4+ millions users data, including their latitude and the longitude.  
  
I have to create an API where i can find users who are located near to a specified radius

e.g radius = 2 km

**What we have tried [18th Dec 2024]:**

We have created an API where we used Haversine formula to find the users who are located near to a specified radius.  
  
The Haversine formula is a mathematical function used in SQL to calculate the distance between two points on a sphere based on their latitude and longitude

As Earth is considered a spherical shape we have used this formula.  
  
But the complexity arises as we are doing this complex operation for 4+ million records everytime when we try to find users, that puts a huge load on the database server (considering its not cached).

Given, the formula used is correct how can we optimise this to make the API more efficient?  
  
**[19th Dec 2024]:**

**What is the need, that pushes us to search 4+ million users records?**

“The Haversine formula” to compute the distance for every single user and get the users who are located within the specified radius

**Approach 1:**

Avoid Haversine formula and use ST\_Distance\_Sphere MYSQL function, as we have created the SPATIAL INDEX on the location column.

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(255) NOT NULL UNIQUE,

location POINT NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

SPATIAL INDEX(location)

);

[ChatGPT]  
  
The Haversine formula is correct for calculating distances, but it can become slow when applied to millions of records in a large database. Spatial indexing is a method to efficiently store and query geographical data, and MySQL provides spatial indexing features that can drastically speed up these types of queries.

ST\_Distance\_Sphere function to calculate the spherical distance between the provided point and the users’ location. MySQL’s geospatial functions are optimized for such calculations.

SELECT id, name, email,

ST\_X(location) AS latitude,

ST\_Y(location) AS longitude,

ST\_Distance\_Sphere(location, POINT(:myLon, :myLat)) AS distance

FROM users

WHERE ST\_Distance\_Sphere(location, POINT(:myLon, :myLat)) <= :radius

ORDER BY distance ASC

LIMIT 10;

Variables:  
  
:myLon -> Current user’s longitude

:myLat -> Current user’s latitude

:radius -> targetted radius

This query looks much better than the old one as we avoided the costly Haversine formula, but still we are checking all the 4+ millions users on the database... that’s still bad.

How can we avoid that? Let’s see approach 2...

**Approach 2:**

Create a Bounding Boxes (bbox) to Narrow Down the Search Space

Now as i have specified we know the radius and the current user’s latitude and longitude.  
  
What if i use a PHP function to create a targetted latitude and longitude pairs (min & max)

Then filter and narrow down the results and apply any complex MYSQL functions only on this space?

So that we don’t need to apply for all the 4+ million records.

PHP function:

function getBoundingBox($latitude, $longitude, $radiusInKm) {

$earthRadius = 6378; // Earth radius in kilometers

$latDelta = $radiusInKm / $earthRadius;

$lonDelta = $radiusInKm / ($earthRadius \* cos(deg2rad($latitude)));

$minLat = $latitude - rad2deg($latDelta);

$maxLat = $latitude + rad2deg($latDelta);

$minLon = $longitude - rad2deg($lonDelta);

$maxLon = $longitude + rad2deg($lonDelta);

return [$minLat, $maxLat, $minLon, $maxLon];

}

How to use it:

list($minLat, $maxLat, $minLon, $maxLon) = getBoundingBox($yourLat, $yourLon, $radiusInKm);

SELECT id, name, email,

ST\_X(location) AS latitude,

ST\_Y(location) AS longitude,

ST\_Distance\_Sphere(location, POINT(:myLon, :myLat)) AS distance

FROM users

WHERE ST\_X(location) BETWEEN :minLon AND :maxLon

AND ST\_Y(location) BETWEEN :minLat AND :maxLat

HAVING distance <= :radius

ORDER BY distance ASC

LIMIT 10;

Variables:  
  
:myLon -> Current user’s longitude

:myLat -> Current user’s latitude

:radius -> targetted radius

**Approach 3:**

How can we make it even better?

Create two more columns, latitude and longitude in addition to the location column with the POINT data type.  
  
Now the latitude and longitude column avoids the need of MYSQL functions like ST\_X(), ST\_Y()

which will make it more efficient.  
  
CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(255) NOT NULL UNIQUE,

location POINT NOT NULL,

latitude DOUBLE NOT NULL, -- Latitude column

longitude DOUBLE NOT NULL, -- Longitude column

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

SPATIAL INDEX(location),

INDEX lat\_lon\_index (latitude, longitude) -- Index on the latitude and longitude columns

);

SELECT id, name, email,

latitude,

longitude,

ST\_Distance\_Sphere(location, POINT(:myLon, :myLat)) AS distance

FROM users

WHERE longitude BETWEEN :minLon AND :maxLon

AND latitude BETWEEN :minLat AND :maxLat

HAVING distance <= :radius

ORDER BY distance ASC

LIMIT 10;

Now this is way more efficient...

**Approach 4:**

Now we can cache frequently used results so that we don’t need to query the server again, instead we can serve it from the cache.

e.g Redis

Cache the results for specific latitude/longitude pairs and radius combinations.

When a similar query is received, check the cache first before querying the database.

Use cache expiration strategies to keep the data fresh.