

# 在 MySQL 数据库中由经纬度计算最近的点

Yuhong Zhong

Symphony

## 1 问题背景

给定点集  $S = \{(x, y) \mid x \in [-90, 90], y \in [-180, 180]\}$ , 其中每一个点  $(x, y) \in S$  都表示三维球面上的一个点, 并且  $x$  表示纬度,  $y$  表示经度。假设  $S$  存储在 MySQL 数据库 sites 中, 并且 sites 的列为 latitude, longitude, 以及 route\_id, 其中 route\_id  $\in \{0, 1, 2, \dots\}$ , 且  $S$  中每个元素都有不同的 route\_id。现有点  $a = (x_0, y_0)$  ( $x_0 \in [-90, 90], y_0 \in [-180, 180]$ ), 往求解在球面距离下  $S$  中距离  $a$  最短的点  $b \in R$ 。

## 2 距离的计算

对于三维球面坐标系下已知经纬度时在球面上的距离的计算, 我们有如下定理。

**定理 1** (三维球体表面的距离). 设两个点  $s_1, s_2$  在半径为  $R$  的三维球体表面, 并且  $s_1$  的纬度为  $\phi_1$ ,  $s_2$  的纬度为  $\phi_2$ ,  $s_1, s_2$  的纬度差为  $\Delta\phi$ , 经度差为  $\Delta\lambda$ , 则  $s_1, s_2$  间的距离  $d$  为:

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

其中,  $\text{haversin}(\theta) = \frac{\text{versin}(\theta)}{2} = \sin^2\left(\frac{\theta}{2}\right)$ ,  $\text{versin}(\theta) = 1 - \cos(\theta) = 2 \sin^2\left(\frac{\theta}{2}\right)$ 。

在 MySQL 数据库中, 若第一个点的纬度为 orig.latitude, 经度为 orig.longitude, 第二个点的纬度为 dest.latitude, 经度为 dest.longitude, 则可用如下命令计算两个点之间在球表面的距离:

```
3956 * 2 * ASIN(SQRT(
POWER(SIN((orig.latitude - dest.latitude) * PI() / 180 / 2), 2)
+ COS(orig.latitude * PI() / 180) * COS(dest.latitude * PI() / 180)
* POWER(SIN((orig.longitude - dest.longitude) * PI() / 180 / 2), 2)))
AS distance
```

## 3 计算最近点的 MySQL 语句

首先, 根据上一节中距离的计算公式, 我们可以直接得到一个直接的命令来求解最近的点:

```

SELECT *, 3956 * 2 * ASIN(SQRT(
POWER(SIN((@orig_latitude - dest.latitude) * PI() / 180 / 2), 2)
+ COS(@orig_latitude * PI() / 180) * COS(dest.latitude * PI() / 180)
* POWER(SIN((@orig_longitude - dest.longitude) * PI() / 180 / 2), 2)))
AS distance
FROM sites dest
ORDER BY distance LIMIT 1;

```

其中, @orig\_latitude, @orig\_longitude 的值分别为点  $a$  的纬度与经度值。

以上命令在数据量较小时有较好的表现, 但是当数据量增多时, 则查询速度明显较慢。因此, 在数据量较大时, 我们需要采取一些方法提高计算的性能。

一个可行的改进即为限定搜索的区域, 即划定一个边长为  $2d$  km 的区域, 仅在此区域内计算距离进行搜索。首先, 需要计算这个正方形区域主对角线两个顶点的经纬度。我们知道, 1 latitude 对应的距离约为 111 km, 1 longitude 对应的距离约为  $111 \times \cos(\text{latitude})$  km。因此, 我们有计算正方形两个顶点的经纬度的 MySQL 命令:

```

SET @lon1 = @orig_longitude - @d / ABS(COS(RADIANS(@orig_latitude)) * 111);
SET @lon2 = @orig_longitude + @d / ABS(COS(RADIANS(@orig_latitude)) * 111);
SET @lat1 = @orig_latitude - (@d / 111);
SET @lat2 = @orig_latitude + (@d / 111);

```

由此, 可将查询最近点的 MySQL 命令改写为:

```

SELECT *, 3956 * 2 * ASIN(SQRT(
POWER(SIN((@orig_latitude - dest.latitude) * PI() / 180 / 2), 2)
+ COS(@orig_latitude * PI() / 180) * COS(dest.latitude * PI() / 180)
* POWER(SIN((@orig_longitude - dest.longitude) * PI() / 180 / 2), 2)))
AS distance
FROM sites dest
WHERE dest.longitude BETWEEN @lon1 AND @lon2
AND dest.latitude BETWEEN @lat1 AND @lat2
ORDER BY distance LIMIT 1;

```

由实验可知, 该查询语句能够大大提高查询速度。最后, 还可将上述过程利用 MySQL 中的存储过程 (Storage Procedure) 加以实现, 以进一步提高查询的速度。该存储过程的定义命令如下:

```

DELIMITER $$
CREATE PROCEDURE find_nearest_point(IN lat DOUBLE, IN lon DOUBLE)
BEGIN
DECLARE lon1 DOUBLE; DECLARE lon2 DOUBLE;

```

```

DECLARE lat1 DOUBLE; DECLARE lat2 DOUBLE;
DECLARE dist DOUBLE;
SET dist = 100;
SET lon1 = lon - dist / ABS(COS(RADIANS(lat)) * 111);
SET lon2 = lon + dist / ABS(COS(RADIANS(lat)) * 111);
SET lat1 = lat - (dist / 111);
SET lat2 = lat + (dist / 111);
SELECT *, 3956 * 2 * ASIN(SQRT(
POWER(SIN((lat - dest.latitude) * PI() / 180 / 2), 2)
+ COS(lat * PI() / 180) * COS(dest.latitude * PI() / 180)
* POWER(SIN((lon - dest.longitude) * PI() / 180 / 2), 2)))
AS distance
FROM sites dest
WHERE dest.longitude BETWEEN lon1 AND lon2
AND dest.lantitude BETWEEN lat1 AND lat2
ORDER BY distance LIMIT 1;
END $$

```

由此，我们只需通过命令 `CALL find_nearest_point(@orig_latitude, @orig_longitude)`；即可较为高效的查询距离  $a$  最近的点  $b \in S$ 。

## 参考文献

- [1] Alexander Rubin: Geo (proximity) Search with MySQL

## 附录

附录中将给出在“跟着电影去旅行”应用中创建寻找最近点的 MySQL 存储过程的命令，以及在 Django 中调用该存储过程的命令。

创建存储过程的 MySQL 命令为：

```

USE symphony;
DELIMITER $$
CREATE PROCEDURE find_nearest_point(IN lat DOUBLE, IN lon DOUBLE)
BEGIN
DECLARE lon1 DOUBLE; DECLARE lon2 DOUBLE;
DECLARE lat1 DOUBLE; DECLARE lat2 DOUBLE;
DECLARE dist DOUBLE;

```

```

SET dist = 100;
SET lon1 = lon - dist / ABS(COS(RADIANS(lat)) * 111);
SET lon2 = lon + dist / ABS(COS(RADIANS(lat)) * 111);
SET lat1 = lat - (dist / 111);
SET lat2 = lat + (dist / 111);
SELECT *, 3956 * 2 * ASIN(SQRT(
POWER(SIN((lat - dest.lat) * PI() / 180 / 2), 2)
+ COS(lat * PI() / 180) * COS(dest.lat * PI() / 180)
* POWER(SIN((lon - dest.lng) * PI() / 180 / 2), 2)))
AS distance
FROM main_page_site dest
WHERE dest.lng BETWEEN lon1 AND lon2
AND dest.lat BETWEEN lat1 AND lat2
ORDER BY distance LIMIT 1;
END $$

```

在 Django 中调用存储过程的 Python 命令为:

```

from django.db import connection
cur = connection.cursor()
cur.callproc('find_nearest_point', [str(lat), str(lng)])
results = cur.fetchall()
cur.close()
result_id = results[0][0]

```