MongoDB Geospatial Queries

Find Restaurants with Geospatial Queries

MongoDB Documentation

May 5, 2025

Overview

- Introduction to MongoDB's geospatial indexing capabilities.
- Demonstration of geospatial queries using:
  - `$geoWithin`
  - `$geoIntersects`
  - `$nearSphere`
- Use case: Finding restaurants in New York City.

Geospatial Indexes

- MongoDB supports geospatial indexes to efficiently execute spatial queries.
- Types of geospatial indexes:
  - `2d` index: For planar geometry.
  - `2dsphere` index: For spherical geometry (e.g., Earth).
- This tutorial uses `2dsphere` indexes.

GeoJSON Format

- MongoDB uses GeoJSON objects to represent geospatial data.
- Common GeoJSON types:
  - `Point`
  - `LineString`
  - `Polygon`
- Example of a GeoJSON Point:

```
{
    "type": "Point",
    "coordinates": [-73.856077, 40.848447]
}
```

Importing Data

- Download datasets:
  - `restaurants.json`
  - `neighborhoods.json`
- Import into MongoDB:

```
mongoimport <path to restaurants.json> -c=restaurants
mongoimport <path to neighborhoods.json> -c=neighborhoods
```

Creating Geospatial Indexes

- Create `2dsphere` indexes on the collections:

```
db.restaurants.createIndex({ location: "2dsphere" })
db.neighborhoods.createIndex({ geometry: "2dsphere" })
```

Sample Restaurant Document

```
{
  "location": {
    "type": "Point",
    "coordinates": [-73.856077, 40.848447]
  },
  "name": "Morris Park Bake Shop"
}
```

Sample Neighborhood Document

```
{
  "geometry": {
    "type": "Polygon",
    "coordinates": [[
      [-73.99, 40.75],
      [-73.98, 40.76],
      [-73.99, 40.75]
    ]]
  },
  "name": "Hell's Kitchen"
}
```

Finding Current Neighborhood

- Determine the user's neighborhood using `$geoIntersects`:

```
db.neighborhoods.findOne({
  geometry: {
    $geoIntersects: {
      $geometry: {
        type: "Point",
        coordinates: [-73.93414657, 40.82302903]
      }
    }
  }
})
```

Counting Restaurants in Neighborhood

- Find all restaurants within the user's neighborhood:

```
var neighborhood = db.neighborhoods.findOne({
  geometry: {
    $geoIntersects: {
      $geometry: {
        type: "Point",
        coordinates: [-73.93414657, 40.82302903]
      }
    }
  }
})
db.restaurants.find({
  location: {
    $geoWithin: {
      $geometry: neighborhood.geometry
    }
  }
}).count()
```

Finding Nearby Restaurants (Unsorted)

- Use `$geoWithin` with `$centerSphere` to find restaurants within 5 miles:

```
db.restaurants.find({
  location: {
    $geoWithin: {
      $centerSphere: [[-73.93414657, 40.82302903], 5 / 3963.2]
    }
  }
})
```

Note: `$centerSphere`'s second argument accepts the radius in radians, so you must divide it by the radius of the earth in miles.

Finding Nearby Restaurants (Sorted)

- Use `$nearSphere` with `$maxDistance` to find and sort restaurants by proximity:

```
var METERS_PER_MILE = 1609.34
db.restaurants.find({
  location: {
    $nearSphere: {
      $geometry: {
        type: "Point",
        coordinates: [-73.93414657, 40.82302903]
      },
      $maxDistance: 5 * METERS_PER_MILE
    }
  }
})
```

Conclusion

- MongoDB's geospatial features enable efficient spatial queries.
- Utilizing `2dsphere` indexes and GeoJSON formats allows for complex geospatial operations.
- Operators like `$geoWithin`, `$geoIntersects`, and `$nearSphere` provide powerful querying capabilities.

References

- MongoDB Geospatial Tutorial: https://www.mongodb.com/docs/manual/tutorial/geospatial-tutorial/
- GeoJSON Specification: https://geojson.org/