this is my data in the restaurants collection:

{"address": {"building": "1007", "coord": [-73.856077, 40.848447], "street": "Morris Park Ave", "zipcode": "10462"}, "borough": "Bronx", "cuisine": "Bakery", "grades": [{"date": {"$date": 1393804800000}, "grade": "A", "score": 2}, {"date": {"$date": 1378857600000}, "grade": "A", "score": 6}, {"date": {"$date": 1358985600000}, "grade": "A", "score": 10}, {"date": {"$date": 1322006400000}, "grade": "A", "score": 9}, {"date": {"$date": 1299715200000}, "grade": "B", "score": 14}], "name": "Morris Park Bake Shop", "restaurant\_id": "30075445"}

{"address": {"building": "469", "coord": [-73.961704, 40.662942], "street": "Flatbush Avenue", "zipcode": "11225"}, "borough": "Brooklyn", "cuisine": "Hamburgers", "grades": [{"date": {"$date": 1419897600000}, "grade": "A", "score": 8}, {"date": {"$date": 1404172800000}, "grade": "B", "score": 23}, {"date": {"$date": 1367280000000}, "grade": "A", "score": 12}, {"date": {"$date": 1336435200000}, "grade": "A", "score": 12}], "name": "Wendy'S", "restaurant\_id": "30112340"}

{"address": {"building": "351", "coord": [-73.98513559999999, 40.7676919], "street": "West 57 Street", "zipcode": "10019"}, "borough": "Manhattan", "cuisine": "Irish", "grades": [{"date": {"$date": 1409961600000}, "grade": "A", "score": 2}, {"date": {"$date": 1374451200000}, "grade": "A", "score": 11}, {"date": {"$date": 1343692800000}, "grade": "A", "score": 12}, {"date": {"$date": 1325116800000}, "grade": "A", "score": 12}], "name": "Dj Reynolds Pub And Restaurant", "restaurant\_id": "30191841"}

{"address": {"building": "2780", "coord": [-73.98241999999999, 40.579505], "street": "Stillwell Avenue", "zipcode": "11224"}, "borough": "Brooklyn", "cuisine": "American ", "grades": [{"date": {"$date": 1402358400000}, "grade": "A", "score": 5}, {"date": {"$date": 1370390400000}, "grade": "A", "score": 7}, {"date": {"$date": 1334275200000}, "grade": "A", "score": 12}, {"date": {"$date": 1318377600000}, "grade": "A", "score": 12}], "name": "Riviera Caterer", "restaurant\_id": "40356018"}

{"address": {"building": "97-22", "coord": [-73.8601152, 40.7311739], "street": "63 Road", "zipcode": "11374"}, "borough": "Queens", "cuisine": "Jewish/Kosher", "grades": [{"date": {"$date": 1416787200000}, "grade": "Z", "score": 20}, {"date": {"$date": 1358380800000}, "grade": "A", "score": 13}, {"date": {"$date": 1343865600000}, "grade": "A", "score": 13}, {"date": {"$date": 1323907200000}, "grade": "B", "score": 25}], "name": "Tov Kosher Kitchen", "restaurant\_id": "40356068"}

{"address": {"building": "8825", "coord": [-73.8803827, 40.7643124], "street": "Astoria Boulevard", "zipcode": "11369"}, "borough": "Queens", "cuisine": "American ", "grades": [{"date": {"$date": 1416009600000}, "grade": "Z", "score": 38}, {"date": {"$date": 1398988800000}, "grade": "A", "score": 10}, {"date": {"$date": 1362182400000}, "grade": "A", "score": 7}, {"date": {"$date": 1328832000000}, "grade": "A", "score": 13}], "name": "Brunos On The Boulevard", "restaurant\_id": "40356151"}

help me write some mongoDB queries

It appears that this is a collection of restaurant documents in the JSON format. Each document contains information about a specific restaurant, such as its address, cuisine type, name, and grades from health inspections. The $date field in the grades sub-document appears to be a Unix timestamp representing the date of the health inspection.

Alaqian Zafar

Write mongoDB Queries, print code only:

14.Find the restaurants which do not prepare any cuisine of 'American ' and

achieved a grade point 'A' and not in the borough of Brooklyn, sorted by cuisine

in descending order.

15.Find the restaurant Id, name, borough and cuisine for those restaurants which

contain 'Wil' as first three letters for its name.

16.Find the restaurant Id, name, borough and cuisine for those restaurants which

contain 'ces' as last three letters for its name.

17.Find the restaurant Id, name, borough and cuisine for those restaurants which

contain 'Reg' as three letters somewhere in its name.

18.Find the restaurants which belong to the borough Bronx and prepared either

American or Chinese dish

19.Find the restaurant Id, name, borough and cuisine for those restaurants which

belong to the boroughs of Staten Island or Queens or Bronx or Brooklyn.

20.Find the restaurant Id, name, borough and cuisine for those restaurants which

are not belonging to the borough Staten Island or Queens or Bronx or Brooklyn

21.Find the restaurant Id, name, borough and cuisine for those restaurants which

achieved a score below 10.

22.Find the restaurant Id, name, borough and cuisine for those restaurants which

prepared dish except 'American' and 'Chinese' or restaurant's name begins with

letter 'Wil'.

23.Find the restaurant Id, name, and grades for those restaurants which achieved a

grade of "A" and scored 11 on an ISODate "2014-08-11T00:00:00Z" among

many of survey dates.

24.Find the restaurant Id, name and grades for those restaurants where the 2nd

element of grades array contains a grade of "A" and score 9 on an ISODate

"2014-08-11T00:00:00Z".

25.Find the restaurant Id, name, address and geographical location for those

restaurants where 2nd element of coord array contains a value which is more

than 42 and up to 52.

Copy code

db.restaurants.find({

cuisine: {$ne: "American"},

borough: {$ne: "Brooklyn"},

grades: {$elemMatch: {grade: "A"}}

}).sort({cuisine: -1});

Copy code

db.restaurants.find({

name: {$regex: /^Wil/}

}, {

restaurant\_id: 1,

name: 1,

borough: 1,

cuisine: 1

});

Copy code

db.restaurants.find({

name: {$regex: /ces$/}

}, {

restaurant\_id: 1,

name: 1,

borough: 1,

cuisine: 1

});

Copy code

db.restaurants.find({

name: {$regex: /Reg/}

}, {

restaurant\_id: 1,

name: 1,

borough: 1,

cuisine: 1

});

Copy code

db.restaurants.find({

borough: "Bronx",

cuisine: {$in: ["American", "Chinese"]}

});

Copy code

db.restaurants.find({

borough: {$in: ["Staten Island", "Queens", "Bronx", "Brooklyn"]}

}, {

restaurant\_id: 1,

name: 1,

borough: 1,

cuisine: 1

});

Copy code

db.restaurants.find({

borough: {$nin: ["Staten Island", "Queens", "Bronx", "Brooklyn"]}

}, {

restaurant\_id: 1,

name: 1,

borough: 1,

cuisine: 1

});

Copy code

db.restaurants.find({

grades: {$elemMatch: {score: {$lt: 10}}}

}, {

restaurant\_id: 1,

name: 1,

borough: 1,

cuisine: 1

});

Copy code

db.restaurants.find({

$and: [

{cuisine: {$ne: "American"}},

{cuisine: {$ne: "Chinese"}},

{name: {$regex: /^Wil/}}

]

}, {

restaurant\_id: 1,

name: 1,

borough: 1,

cuisine: 1

});

Copy code

db.restaurants.find({

grades: {

$elemMatch: {

grade: "A",

score: 11,

date: {$eq: ISODate("2014-08-11T00:00:00Z")}

}

}

}, {

restaurant\_id: 1,

name: 1,

grades: 1

});

Copy code

db.restaurants.find({

grades: {

$elemMatch: {

$and: [

{grade: "A"},

{score

Alaqian Zafar

repeat no. 24 and 25

Copy code

db.restaurants.find({

grades: {

$elemMatch: {

$and: [

{grade: "A"},

{score: 9},

{date: {$eq: ISODate("2014-08-11T00:00:00Z")}}

]

}

}

}, {

restaurant\_id: 1,

name: 1,

grades: 1

});

Copy code

db.restaurants.find({

"address.coord.1": {$gt: 42, $lte: 52}

}, {

restaurant\_id: 1,

name: 1,

address: 1,

"address.coord": 1

});

Alaqian Zafar

Here is my data:

{"date": "-300", "description": "Pilgrims travel to the healing temples of Asclepieion to be cured of their ills. After a ritual purification the followers bring offerings or sacrifices.", "lang": "en", "category1": "By place", "category2": "Greece", "granularity": "year"}

{"date": "-300", "description": "Pyrrhus, the King of Epirus, is taken as a hostage to Egypt after the Battle of Ipsus and makes a diplomatic marriage with the princess Antigone, daughter of Ptolemy and Berenice.", "lang": "en", "category1": "By place", "category2": "Egypt", "granularity": "year"}

{"date": "-300", "description": "Ptolemy concludes an alliance with King Lysimachus of Thrace and gives him his daughter Arsinoe II in marriage.", "lang": "en", "category1": "By place", "category2": "Egypt", "granularity": "year"}

{"date": "-300", "description": "Seleucus founds the city of Antioch, some 20 miles up the Orontes River, naming it after his father.", "lang": "en", "category1": "By place", "category2": "Seleucid Empire", "granularity": "year"}

{"date": "-300", "description": "After the death of his wife Apama, Seleucus marries Stratonice, daughter of Demetrius Poliorcetes.", "lang": "en", "category1": "By place", "category2": "Seleucid Empire", "granularity": "year"}

15 points: (historical-events.json): Count the number of events per year.

Note, you will need to create a new date field from the string provided.

Assume numbers are years if the entry is not a valid date.

The following query can be used to count the number of events per year in the historical-events collection:

Copy code

db.historical-events.aggregate([

{

$addFields: {

year: {

$cond: {

if: {$dateToString: {format: "%Y", date: "$date"}},

then: {$dateToString: {format: "%Y", date: "$date"}},

else: "$date"

}

}

}

},

{$group: {\_id: "$year", count: {$sum: 1}}},

{$sort: {\_id: 1}}

]);

The first stage of the pipeline uses the $addFields operator to create a new year field by attempting to parse the value of the date field as a date using the $dateToString operator. If the date field is not a valid date, the year field will be set to the original value of the date field.

The second stage of the pipeline uses the $group operator to group the documents by the year field and count the number of documents in each group.

The final stage of the pipeline uses the $sort operator to sort the documents by the year field in ascending order. This ensures that the resulting documents are ordered by year.

This query will return a document for each year, with the \_id field set to the year and the count field set to the number of events that occurred in that year.

Alaqian Zafar

is there another way to do this

Yes, there are several other ways to achieve the same result. For example, instead of using the $dateToString operator to parse the date field, you could use the $dateFromString operator to convert the date field to a Date type. This would allow you to use the $year operator to extract the year from the date field directly.

Here is an example of a query that uses the $dateFromString and $year operators to count the number of events per year:

Copy code

db.historical-events.aggregate([

{

$addFields: {

year: {

$cond: {

if: {$dateFromString: {dateString: "$date"}},

then: {$year: {$dateFromString: {dateString: "$date"}}},

else: "$date"

}

}

}

},

{$group: {\_id: "$year", count: {$sum: 1}}},

{$sort: {\_id: 1}}

]);

Alternatively, you could use the $type operator in the $cond expression to check the type of the date field, and only attempt to extract the year if the date field is a string.

Here is an example of a query that uses the $type and $year operators to count the number of events per year:

Copy code

db.historical-events.aggregate([

{

$addFields: {

year: {

$cond: {

if: {$eq: ["$date", {$type: "$date"}]},

then: {$year: "$date"},

else: "$date"

}

}

}

},

{$group: {\_id: "$year", count: {$sum: 1}}},

{$sort: {\_id: 1}}

]);

These are just a few examples of how you could count the number of events per year in the historical-events collection. There may be other ways to achieve the same result.

Alaqian Zafar

would this code produce the same result?

db.historical-events.aggregate([

{ $addFields: { date: { $toDate: "$year" } } },

{ $group: { \_id: { $year: "$date" }, count: { $sum: 1 } } }

]).forEach(printjson)

No, this code will not produce the same result. The $toDate operator converts a string or number to a Date type, but it does not automatically extract the year from the date. As a result, the $group stage of the pipeline will group the documents by the full date rather than just the year.

To extract the year from the date field and group the documents by year, you would need to use the $year operator in the $group stage of the pipeline, as in the following example:

Copy code

db.historical-events.aggregate([

{ $addFields: { date: { $toDate: "$year" } } },

{ $group: { \_id: { $year: "$date" }, count: { $sum: 1 } } },

{ $sort: { \_id: 1 } }

]).forEach(printjson)

This code will first use the $addFields operator to convert the date field to a Date type using the $toDate operator. Then, it will use the $group operator to group the documents by the year extracted from the date field using the $year operator. Finally, it will use the $sort operator to sort the documents by the \_id field in ascending order, so that the resulting documents are ordered by year.

This query will return a document for each year, with the \_id field set to the year and the count field set to the number of events that occurred in that year.

Alaqian Zafar

I have data in 2 files:

meteorites.json:

{"fall":"Fell","geolocation":{"type":"Point","coordinates":[6.08333,50.775]},"id":"1","mass":"21","name":"Aachen","nametype":"Valid","recclass":"L5","reclat":"50.775000","reclong":"6.083330","year":"1880-01-01T00:00:00.000"}

worldcities.csv:

"city","city\_ascii","lat","lng","country","iso2","iso3","admin\_name","capital","population","id"

"Malishevë","Malisheve","42.4822","20.7458","Kosovo","XK","XKS","Malishevë","admin","","1901597212"

"Prizren","Prizren","42.2139","20.7397","Kosovo","XK","XKS","Prizren","admin","","1901360309"

Use the MongoDB geospatial facilities to find the nearest city to each meteorite “fallen” (not found) since the year 1950, inclusive. Distance is between coordinates, straight line.

Use the $near operator and select this closest entry per city

'worldcities' is a CSV file. You will need to import into MongoDB AND clean-up the double quotes.

To find the nearest city to each meteorite that has "fallen" (not found) since the year 1950, inclusive, using the MongoDB geospatial facilities, you can use the following steps:

Import the worldcities.csv file into a MongoDB collection, using the mongoimport tool. This will convert the CSV file into a series of documents that can be queried using MongoDB.

Copy code

mongoimport --type csv --headerline worldcities.csv -d meteorites -c worldcities

Clean up the double quotes in the worldcities collection by using the $replaceAll operator in an update query. This will replace any double quotes in the values of the fields with a single quote character.

Copy code

db.worldcities.updateMany({}, {$replaceAll: {"$quote": "'"}})

Create a geospatial index on the geolocation.coordinates field in the worldcities collection. This will enable MongoDB to efficiently search for cities near a given set of coordinates.

Copy code

db.worldcities.createIndex({'geolocation.coordinates': '2dsphere'})

4

Alaqian Zafar

repeat after 4.

After creating the geospatial index, you can use the $near operator in a query to find the nearest city to each meteorite that has fallen since the year 1950, inclusive. Here is an example of how this could be done:

Copy code

db.meteorites.find({

year: {$gte: ISODate("1950-01-01T00:00:00.000")},

fall: "Fell"

}).forEach(function(meteorite) {

var nearestCity = db.worldcities.findOne({

geolocation: {

$near: {

$geometry: meteorite.geolocation,

$maxDistance: 1000

}

}

});

printjson(nearestCity);

});

This query will iterate over all the meteorites in the meteorites collection that have fallen since 1950, and for each meteorite, it will use the $near operator to find the nearest city within a distance of 1000 meters. The result will be a series of documents representing the closest city to each meteorite.

Note that this query will only return the closest city to each meteorite, not all cities within a certain distance of the meteorite. To return all cities within a certain distance, you can use the $geoWithin operator instead of the $near operator. For example:

Copy code

db.meteorites.find({

year: {$gte: ISODate("1950-01-01T00:00:00.000")},

fall: "Fell"

}).forEach(function(meteorite) {

var nearbyCities = db.worldcities.find({

geolocation: {

$geoWithin: {

$geometry: meteorite.geolocation,

$maxDistance: 1000

}

}

});

printjson(nearbyCities);

});

This query will return all cities within 1000 meters of each meteorite that has fallen since 1950. The result will be a series of documents representing the cities near each meteorite.

Alaqian Zafar

i just want the nearest city

To find the nearest city to each meteorite that has fallen since the year 1950, inclusive, using the MongoDB geospatial facilities, you can use the following query:

Copy code

db.meteorites.find({

year: {$gte: ISODate("1950-01-01T00:00:00.000")},

fall: "Fell"

}).forEach(function(meteorite) {

var nearestCity = db.worldcities.findOne({

geolocation: {

$near: {

$geometry: meteorite.geolocation,

$maxDistance: 1000

}

}

});

printjson(nearestCity);

});

This query will iterate over all the meteorites in the meteorites collection that have fallen since 1950, and for each meteorite, it will use the $near operator to find the nearest city within a distance of 1000 meters. The result will be a series of documents representing the closest city to each meteorite.

Note that this query will only return the closest city to each meteorite, not all cities within a certain distance of the meteorite. If you want to return all cities within a certain distance of each meteorite, you can use the $geoWithin operator instead of the $near operator, as in the following example:

Copy code

db.meteorites.find({

year: {$gte: ISODate("1950-01-01T00:00:00.000")},

fall: "Fell"

}).forEach(function(meteorite) {

var nearbyCities = db.worldcities.find({

geolocation: {

$geoWithin: {

$geometry: meteorite.geolocation,

$maxDistance: 1000

}

}

});

printjson(nearbyCities);

});

This query will return all cities within 1000 meters of each meteorite that has fallen since 1950. The result will be a series of documents representing the cities near each meteorite.