HELLENIC REPUBLIC
National and Kapodistrian
University of Athens
EST. 1837

DEPARTMENT OF
INFORMATICS &
TELECOMMUNICATIONS

Programming Project II
NoSQL 311-Chicago-Incidents

Fotis Memis - cs2200010
Anna-Aikaterini Kavvada - 1115201500050

January 16, 2021

Professor: Alexis Delis
Class: M149 - Fall 2020

1 Introduction

In this project we have designed and implemented a NoSQL database solution to manage *"311 Incidents"* data openly published by the city of Chicago, IL. Along with it we have also implemented a rest API, providing access to the aforementioned database to the users, namely the residents of Chicago. The database termed $db_311ciispopulatedbydataavailableat:$ $https://www.kaggle.com/chicago/chicago-311-service-requests.$

2 Project Structure

- ***ChicagoIncidents_311_NonRelational***
  - ***ci_311***
    - ***views.py****: containing all the methods implementing the queries according to the rest api rules.*
  - ***migration***
    - ***indexes.bson****: all the indexes we used over the collections*
    - ***queries.bson****: all the native queries*
    - ***userGeneration.py****: script for citizen generation and insertion in collection*
    - ***migrationPreprocess.py***
  - ***README***

Github Repository Link

3 Data Base Schema, Indices and Data Migration

3.1 Schema

The database solution we provide consists of 2 collections, incidents and citizens respectively. The incident collection holds all the documents about the recorded incidents in the city of Chicago, along with an embedded field (names) the citizen upvotes that each incident has received. The citizen collection contains the name, phone, address and an embedded field containing the id's of the incidents that the citizen might have upvoted.

3.2 Indices

We concluded using the following incidents over the aforementioned collection to otpimize the run time of the queries we implement. First in the incident collection we added 3 indexes; ***a)*** *in creationDate, optimizing the run time for all queries depending on the creationDate field.* ***b)*** *requestType, enhansing the queries depending on actions over this particular field. Furthermore, we have a query running when we are about to insert a new incident in order to secure check that the new incident about to be inserted belongs to one of the existing request types. Without the index this query takes a respective amount of time, while now it is instant.* ***c)****requestServiceNumber this index is not actually used in any of the queries, but it is useful if we are going to search a particular incident. We consider this number as the id of the incident to the outside world, whereas the _id field is just for the programmer and the system.*

3.3 Data Migration

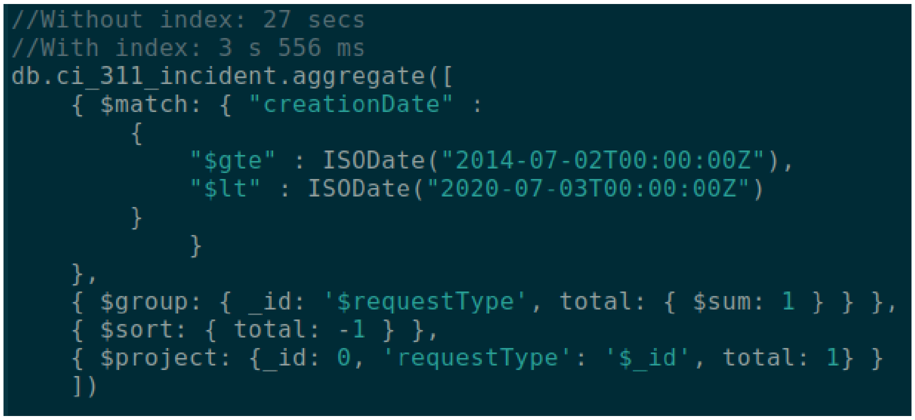
For the data migration we have use the official mongo GUI available. Before that, we run a mild preprocess (migrationPreprocess.py) to rename the CSVs' columns we use for input data so that they are uniform between the shared fields. n order to insert the citizens in a collection and add the embedded field voters in the incident collection, we created the script userGeneration.py using the database client from the libary pymongo to interact with the database and. To generate the users we use the open source library faker.

4 Rest API

For the Rest API we used the Django+Rest framework along with djongo, which integrates the django ORM when mongoDB is used as back-end. However, we ended up with the pymongo library since we saw it better fitting for this particular project, since one of the main subjects of this project is writing mongo native queries for the mongoDB we use as a back-end.

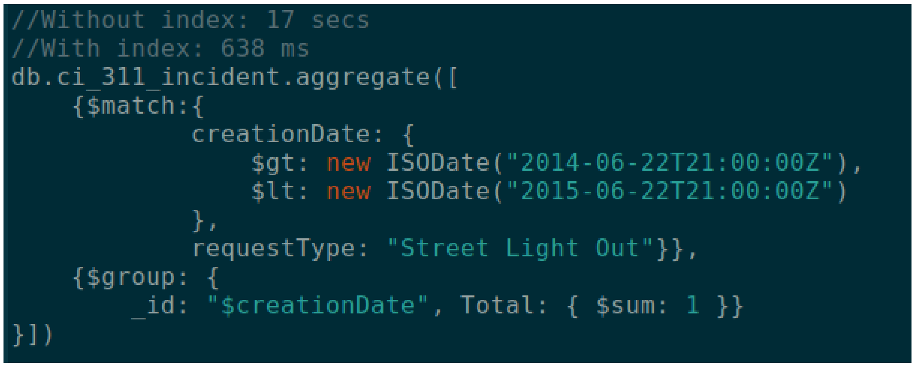
5 Sample snapshots of the query results - Metrics

1. *Find the total requests per type that were created within a specified time range and sort them in a descending order.*



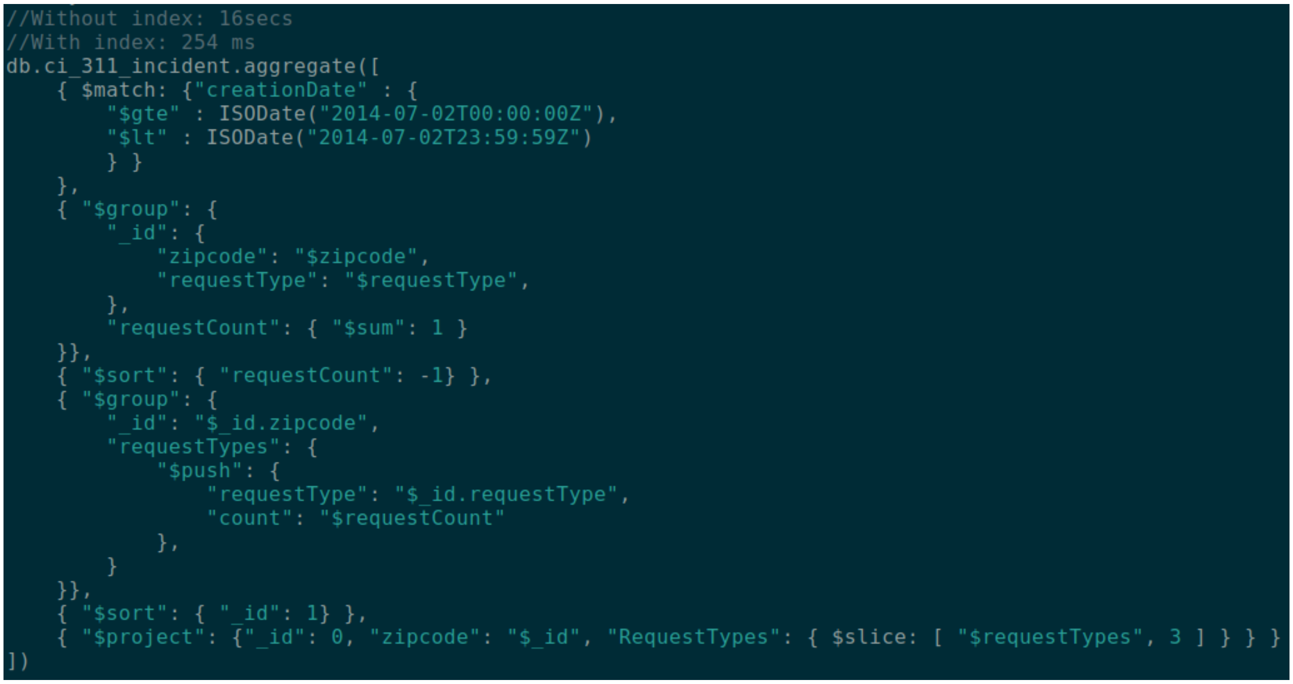
```
//Without index: 27 secs
//With index: 3 s 556 ms
db.ci_311_incident.aggregate([
    { $match: { "creationDate" :
        {
            "$gte" : ISODate("2014-07-02T00:00:00Z"),
            "$lt" : ISODate("2020-07-03T00:00:00Z")
        }
            }
    },
    { $group: { _id: '$requestType', total: { $sum: 1 } } },
    { $sort: { total: -1 } },
    { $project: {_id: 0, 'requestType': '$_id', total: 1} }
    ])
```

2. *Find the number of total requests per day for a specific request type and time range.*



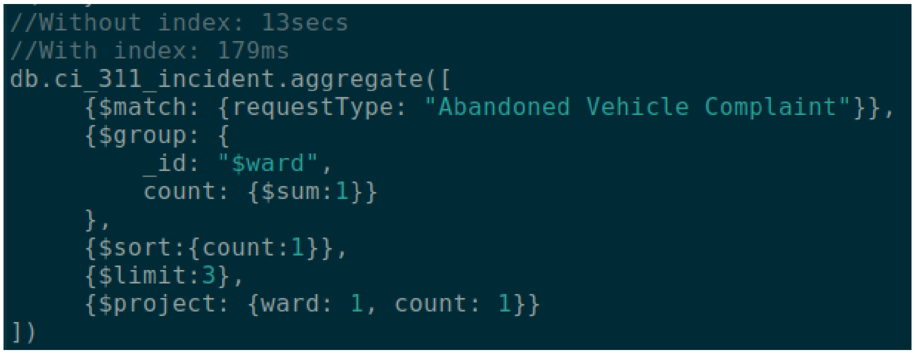
```
//Without index: 17 secs
//With index: 638 ms
db.ci_311_incident.aggregate([
    {$match:{
            creationDate: {
                $gt: new ISODate("2014-06-22T21:00:00Z"),
                $lt: new ISODate("2015-06-22T21:00:00Z")
            },
            requestType: "Street Light Out"}},
    {$group: {
        _id: "$creationDate", Total: { $sum: 1 }}
}])
```

3. *Find the three most common service requests per zipcode for a specific day.*



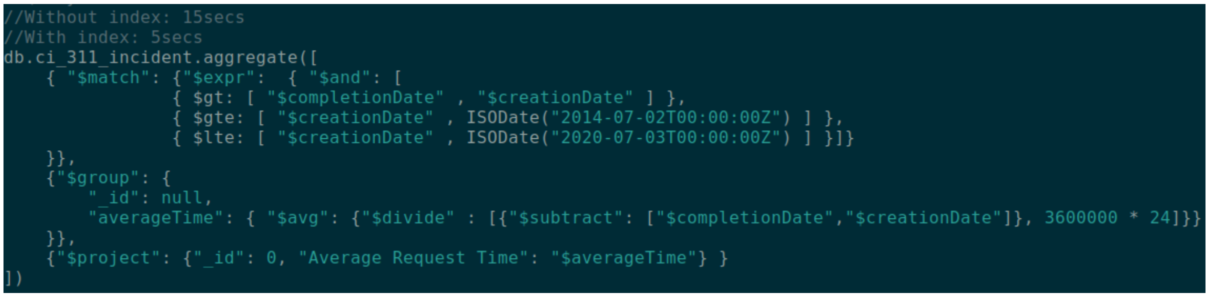
```
//Without index: 16secs
//With index: 254 ms
db.ci_311_incident.aggregate([
    { $match: {"creationDate" : {
        "$gte" : ISODate("2014-07-02T00:00:00Z"),
        "$lt" : ISODate("2014-07-02T23:59:59Z")
        } }
    },
    { "$group": {
        "_id": {
            "zipcode": "$zipcode",
            "requestType": "$requestType",
        },
        "requestCount": { "$sum": 1 }
    }},
    { "$sort": { "requestCount": -1} },
    { "$group": {
        "_id": "$_id.zipcode",
        "requestTypes": {
            "$push": {
                "requestType": "$_id.requestType",
                "count": "$requestCount"
            },
        }
    }},
    { "$sort": { "_id": 1} },
    { "$project": {"_id": 0, "zipcode": "$_id", "RequestTypes": { $slice: [ "$requestTypes", 3 ] } } }
])
```

4. *Find the three least common wards with regards to a given service request type.*



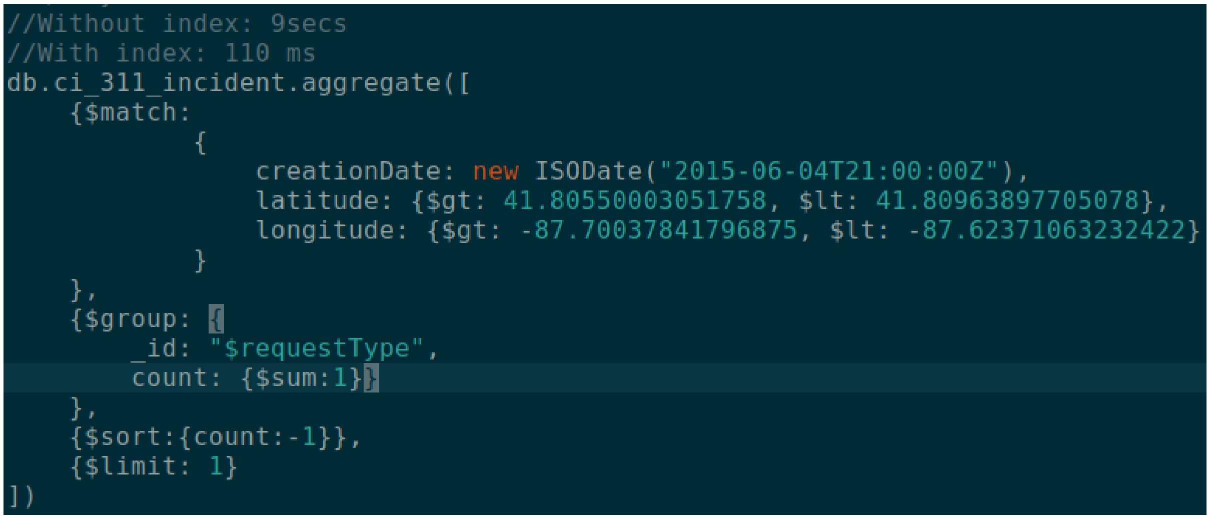
```
//Without index: 13secs
//With index: 179ms
db.ci_311_incident.aggregate([
     {$match: {requestType: "Abandoned Vehicle Complaint"}},
     {$group: {
          _id: "$ward",
          count: {$sum:1}}
     },
     {$sort:{count:1}},
     {$limit:3},
     {$project: {ward: 1, count: 1}}
])
```

5. *Find the average completion time per service request for a specific date range.*



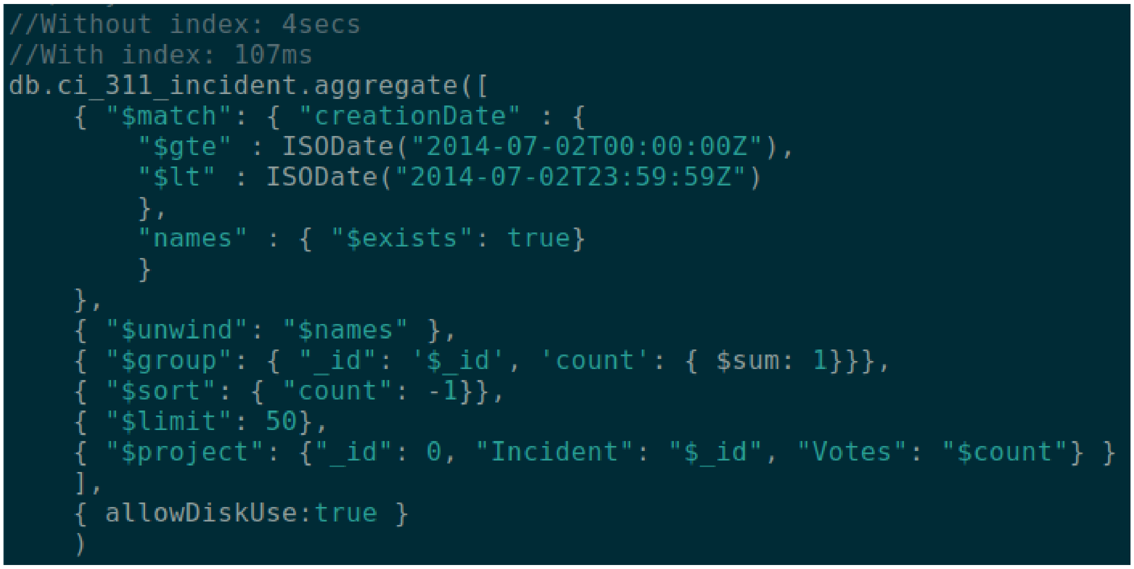
```
//Without index: 15secs
//With index: 5secs
db.ci_311_incident.aggregate([
    { "$match": {"$expr":  { "$and": [
                { $gt: [ "$completionDate" , "$creationDate" ] },
                { $gte: [ "$creationDate" , ISODate("2014-07-02T00:00:00Z") ] },
                { $lte: [ "$creationDate" , ISODate("2020-07-03T00:00:00Z") ] }]}
    }},
    {"$group": {
        "_id": null,
        "averageTime": { "$avg": {"$divide" : [{"$subtract": ["$completionDate","$creationDate"]}, 3600000 * 24]}}
    }},
    {"$project": {"_id": 0, "Average Request Time": "$averageTime"} }
])
```

6. *Find the most common service request in a specified bounding box for a specific day. You are encouraged to use GeoJSON objects and Geospatial Query Operators*



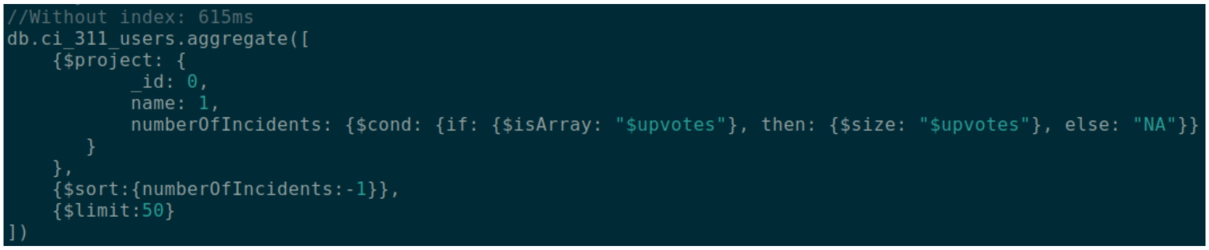
```
//Without index: 9secs
//With index: 110 ms
db.ci_311_incident.aggregate([
    {$match:
            {
                creationDate: new ISODate("2015-06-04T21:00:00Z"),
                latitude: {$gt: 41.80550003051758, $lt: 41.80963897705078},
                longitude: {$gt: -87.70037841796875, $lt: -87.62371063232422}
            }
    },
    {$group: {
        _id: "$requestType",
        count: {$sum:1}}
    },
    {$sort:{count:-1}},
    {$limit: 1}
])
```

7. *Find the fifty most upvoted service requests for a specific day.*



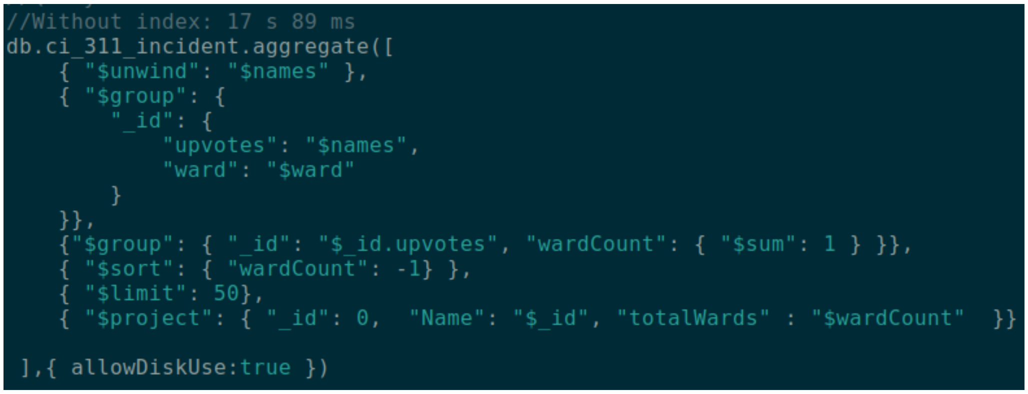
```
//Without index: 4secs
//With index: 107ms
db.ci_311_incident.aggregate([
    { "$match": { "creationDate" : {
        "$gte" : ISODate("2014-07-02T00:00:00Z"),
        "$lt" : ISODate("2014-07-02T23:59:59Z")
        },
        "names" : { "$exists": true}
        }
    },
    { "$unwind": "$names" },
    { "$group": { "_id": '$_id', 'count': { $sum: 1}}},
    { "$sort": { "count": -1}},
    { "$limit": 50},
    { "$project": {"_id": 0, "Incident": "$_id", "Votes": "$count"} }
    ],
    { allowDiskUse:true }
    )
```

8. *Find the fifty most active citizens, with regard to the total number of upvotes.*



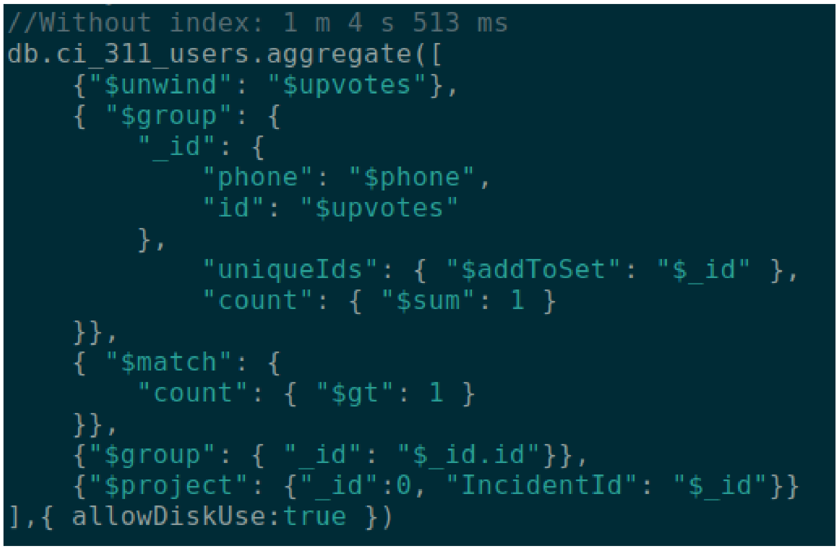
```
//Without index: 615ms
db.ci_311_users.aggregate([
    {$project: {
           _id: 0,
           name: 1,
           numberOfIncidents: {$cond: {if: {$isArray: "$upvotes"}, then: {$size: "$upvotes"}, else: "NA"}}
       }
    },
    {$sort:{numberOfIncidents:-1}},
    {$limit:50}
])
```

9. *Find the top fifty citizens, with regard to the total number of wards for which they have upvoted an incidents.*



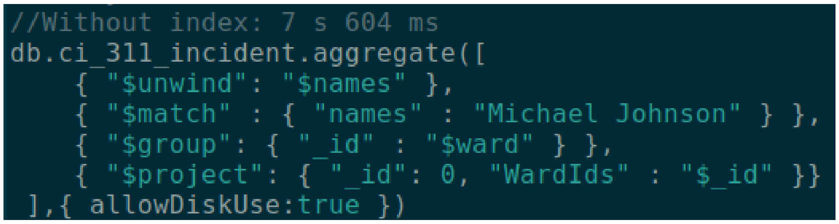
```
//Without index: 17 s 89 ms
db.ci_311_incident.aggregate([
    { "$unwind": "$names" },
    { "$group": {
        "_id": {
            "upvotes": "$names",
            "ward": "$ward"
        }
    }},
    {"$group": { "_id": "$_id.upvotes", "wardCount": { "$sum": 1 } }},
    { "$sort": { "wardCount": -1} },
    { "$limit": 50},
    { "$project": { "_id": 0,  "Name": "$_id", "totalWards" : "$wardCount"  }}

 ],{ allowDiskUse:true })
```

10. *Find all incident ids for which the same telephone number has been used for more than one names.*



```
//Without index: 1 m 4 s 513 ms
db.ci_311_users.aggregate([
    {"$unwind": "$upvotes"},
    { "$group": {
        "_id": {
            "phone": "$phone",
            "id": "$upvotes"
        },
            "uniqueIds": { "$addToSet": "$_id" },
            "count": { "$sum": 1 }
    }},
    { "$match": {
        "count": { "$gt": 1 }
    }},
    {"$group": { "_id": "$_id.id"}},
    {"$project": {"_id":0, "IncidentId": "$_id"}}
],{ allowDiskUse:true })
```

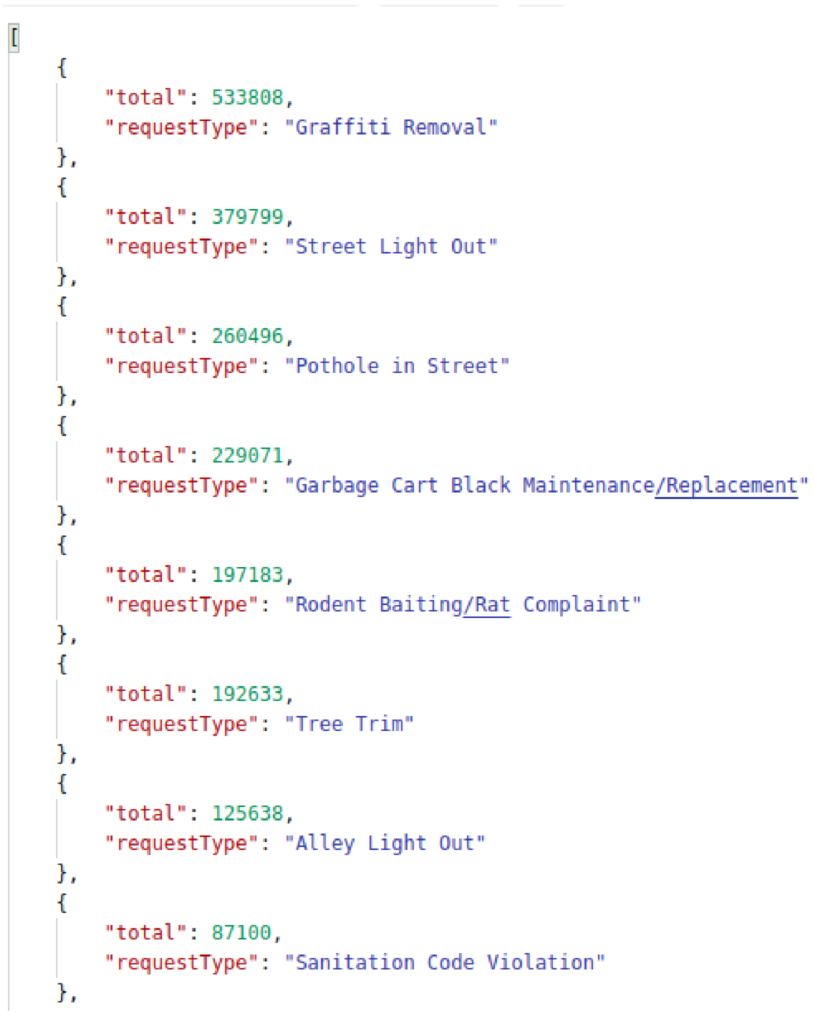
11. *Find all the wards in which a given name has casted a vote for an incident taking place in it.*



```
//Without index: 7 s 604 ms
db.ci_311_incident.aggregate([
    { "$unwind": "$names" },
    { "$match" : { "names" : "Michael Johnson" } },
    { "$group": { "_id" : "$ward" } },
    { "$project": { "_id": 0, "WardIds" : "$_id" }}
 ],{ allowDiskUse:true })
```

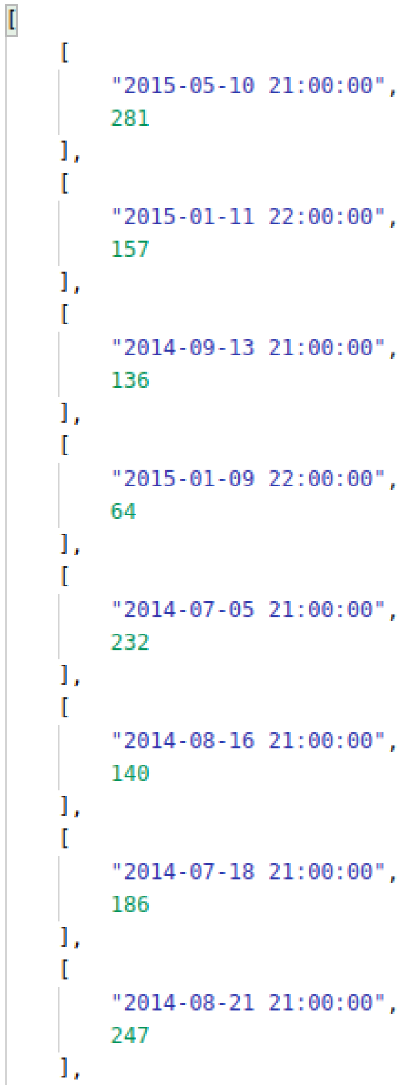
5.1 Queries' Output

1. *Find the total requests per type that were created within a specified time range and sort them in a descending order.*



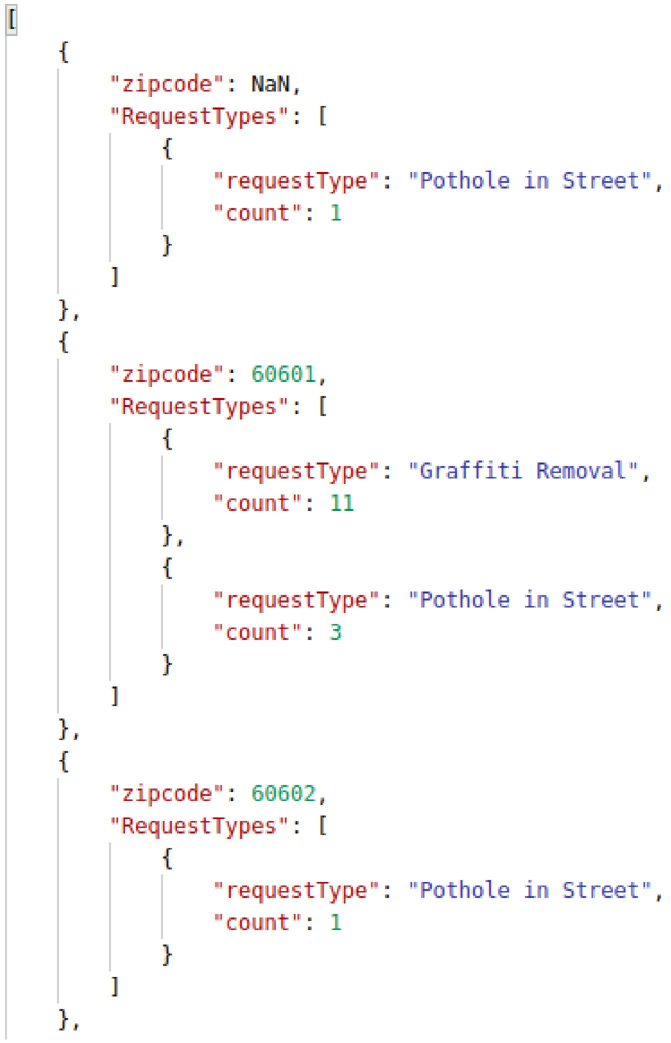
```
[
    {
        "total": 533808,
        "requestType": "Graffiti Removal"
    },
    {
        "total": 379799,
        "requestType": "Street Light Out"
    },
    {
        "total": 260496,
        "requestType": "Pothole in Street"
    },
    {
        "total": 229071,
        "requestType": "Garbage Cart Black Maintenance/Replacement"
    },
    {
        "total": 197183,
        "requestType": "Rodent Baiting/Rat Complaint"
    },
    {
        "total": 192633,
        "requestType": "Tree Trim"
    },
    {
        "total": 125638,
        "requestType": "Alley Light Out"
    },
    {
        "total": 87100,
        "requestType": "Sanitation Code Violation"
    },
```

2. *Find the number of total requests per day for a specific request type and time range.*



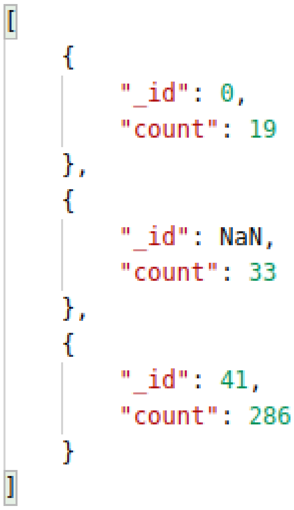
```
[
    [
        "2015-05-10 21:00:00",
        281
    ],
    [
        "2015-01-11 22:00:00",
        157
    ],
    [
        "2014-09-13 21:00:00",
        136
    ],
    [
        "2015-01-09 22:00:00",
        64
    ],
    [
        "2014-07-05 21:00:00",
        232
    ],
    [
        "2014-08-16 21:00:00",
        140
    ],
    [
        "2014-07-18 21:00:00",
        186
    ],
    [
        "2014-08-21 21:00:00",
        247
    ],
```

3. *Find the three most common service requests per zipcode for a specific day.*



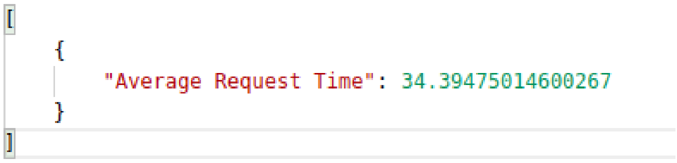
```
[
    {
        "zipcode": NaN,
        "RequestTypes": [
            {
                "requestType": "Pothole in Street",
                "count": 1
            }
        ]
    },
    {
        "zipcode": 60601,
        "RequestTypes": [
            {
                "requestType": "Graffiti Removal",
                "count": 11
            },
            {
                "requestType": "Pothole in Street",
                "count": 3
            }
        ]
    },
    {
        "zipcode": 60602,
        "RequestTypes": [
            {
                "requestType": "Pothole in Street",
                "count": 1
            }
        ]
    },
```

4. *Find the three least common wards with regards to a given service request type.*



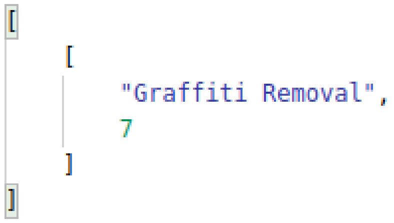
```
[
    {
        "_id": 0,
        "count": 19
    },
    {
        "_id": NaN,
        "count": 33
    },
    {
        "_id": 41,
        "count": 286
    }
]
```

5. *Find the average completion time per service request for a specific date range.*



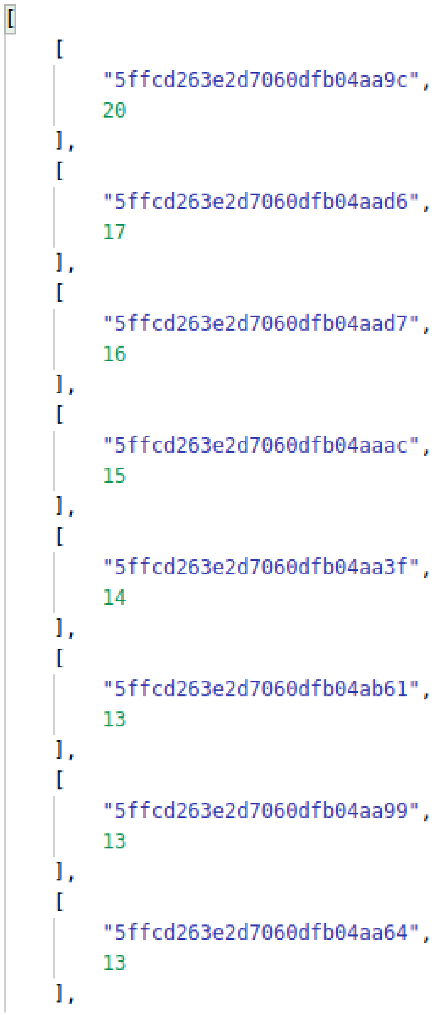
```
[
    {
        "Average Request Time": 34.39475014600267
    }
]
```

6. *Find the most common service request in a specified bounding box for a specific day. You are encouraged to use GeoJSON objects and Geospatial Query Operators*



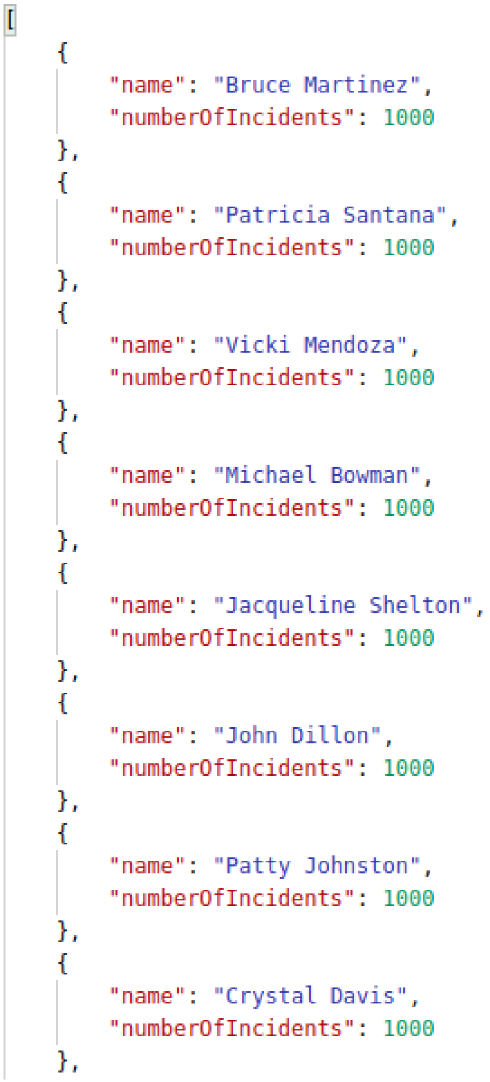
```
[
    [
        "Graffiti Removal",
        7
    ]
]
```

7. *Find the fifty most upvoted service requests for a specific day.*



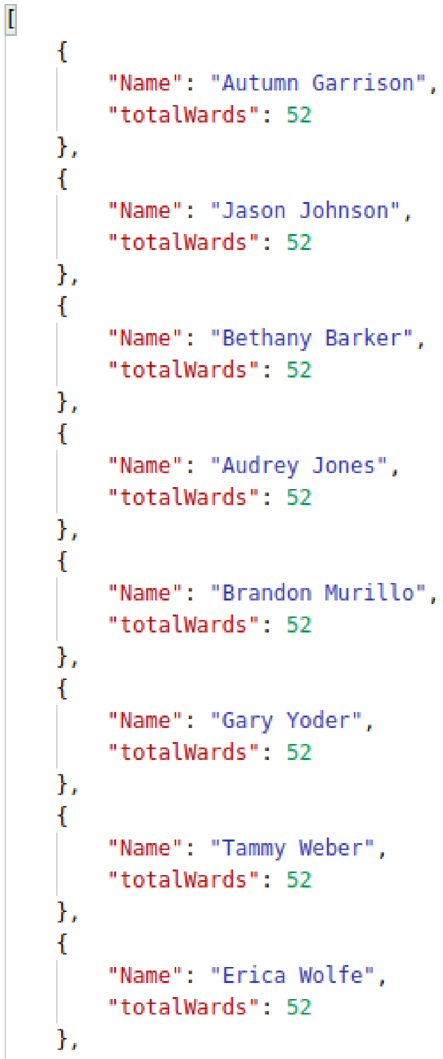
```
[
    [
        "5ffcd263e2d7060dfb04aa9c",
        20
    ],
    [
        "5ffcd263e2d7060dfb04aad6",
        17
    ],
    [
        "5ffcd263e2d7060dfb04aad7",
        16
    ],
    [
        "5ffcd263e2d7060dfb04aaac",
        15
    ],
    [
        "5ffcd263e2d7060dfb04aa3f",
        14
    ],
    [
        "5ffcd263e2d7060dfb04ab61",
        13
    ],
    [
        "5ffcd263e2d7060dfb04aa99",
        13
    ],
    [
        "5ffcd263e2d7060dfb04aa64",
        13
    ],
```

8. *Find the fifty most active citizens, with regard to the total number of upvotes.*



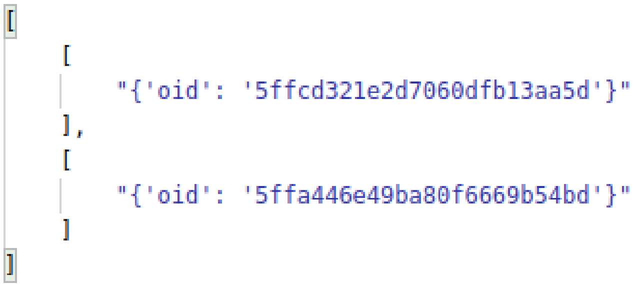
```
[
    {
        "name": "Bruce Martinez",
        "numberOfIncidents": 1000
    },
    {
        "name": "Patricia Santana",
        "numberOfIncidents": 1000
    },
    {
        "name": "Vicki Mendoza",
        "numberOfIncidents": 1000
    },
    {
        "name": "Michael Bowman",
        "numberOfIncidents": 1000
    },
    {
        "name": "Jacqueline Shelton",
        "numberOfIncidents": 1000
    },
    {
        "name": "John Dillon",
        "numberOfIncidents": 1000
    },
    {
        "name": "Patty Johnston",
        "numberOfIncidents": 1000
    },
    {
        "name": "Crystal Davis",
        "numberOfIncidents": 1000
    },
```

9. *Find the top fifty citizens, with regard to the total number of wards for which they have upvoted an incidents.*



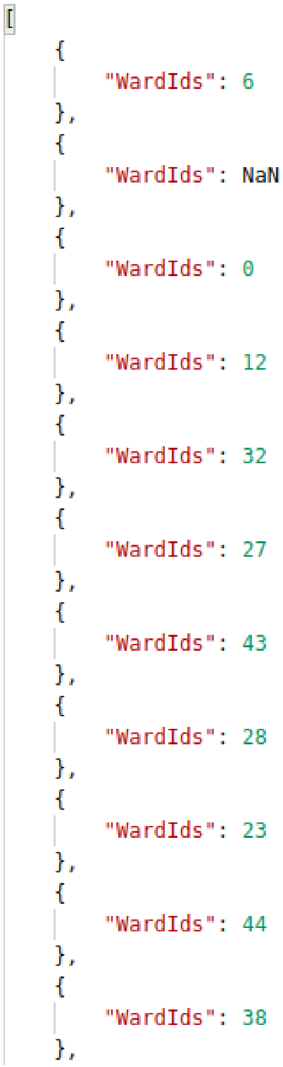
```
[
    {
        "Name": "Autumn Garrison",
        "totalWards": 52
    },
    {
        "Name": "Jason Johnson",
        "totalWards": 52
    },
    {
        "Name": "Bethany Barker",
        "totalWards": 52
    },
    {
        "Name": "Audrey Jones",
        "totalWards": 52
    },
    {
        "Name": "Brandon Murillo",
        "totalWards": 52
    },
    {
        "Name": "Gary Yoder",
        "totalWards": 52
    },
    {
        "Name": "Tammy Weber",
        "totalWards": 52
    },
    {
        "Name": "Erica Wolfe",
        "totalWards": 52
    },
```

10. *Find all incident ids for which the same telephone number has been used for more than one names.*



```
[
    [
        "{'oid': '5ffcd321e2d7060dfb13aa5d'}"
    ],
    [
        "{'oid': '5ffa446e49ba80f6669b54bd'}"
    ]
]
```

11. *Find all the wards in which a given name has casted a vote for an incident taking place in it.*



```
[
    {
        "WardIds": 6
    },
    {
        "WardIds": NaN
    },
    {
        "WardIds": 0
    },
    {
        "WardIds": 12
    },
    {
        "WardIds": 32
    },
    {
        "WardIds": 27
    },
    {
        "WardIds": 43
    },
    {
        "WardIds": 28
    },
    {
        "WardIds": 23
    },
    {
        "WardIds": 44
    },
    {
        "WardIds": 38
    },
```