PS3

November 7, 2022

CS 61 Lab 3: MongoDB, Javascript

Student: Amittai Siavava

Date: 2022-11-07

1 Part 1 (30 Points)

Use the zipcodes dataset.

1.0.1 1. What is the primary schema represented in zipcodes.json?

```
{
  "_id"   : String,
  "city"  : String,
  "loc"   : [ Number, Number ],
  "pop"   : Number,
  "state" : String
}
```

[ ]:

1.0.2 2. Get the zipcodes.json file loaded into a db on your machine (local) or Atlas (cloud).

[ ]:
```
# Import a bunch of stuff we need.
from pymongo import MongoClient, InsertOne, DeleteOne, ReplaceOne, UpdateOne
import json
from IPython.display import display, Markdown, Latex

display(Markdown('##### Connecting to MongoDB'))

# Connect to the database.
client = MongoClient('localhost', 27017)
db = client['test']
collection = db['test']
```

```
# Clear collection if it has any data.
collection.delete_many({})

# Insert each line in file into the collection.
record_count = 0
display(Markdown('##### Inserting data into MongoDB'))
with open("zipcodes.json") as f:
  for line in f:
    if line.strip(): # ignore blank lines
      record_count += 1
      record = json.loads(line)
      collection.insert_one(record)

# Query the collection
display(Markdown(f"""\n
##### Report on insertions.

  | Metric | Value |
  | :--- | ---: |
  | Records read from file | {record_count} |
  | Records in database    | {collection.count_documents({})} |

##### Sample record:
  ```json
    {collection.find_one({})}
  ```
"""))
```

Connecting to MongoDB

Inserting data into MongoDB

Report on insertions.

Metric	Value
Records read from file	29353
Records in database	29353

Sample record:


```
{'_id': '01001', 'city': 'AGAWAM', 'loc': [-72.622739, 42.070206], 'pop': 15338, 'state': 'M
```

1.0.3 3. Count the number of zip codes in the collection.

```
[ ]: display(Markdown(f"""
#### Strategy

Since zipcodes are the `_id` field, we can use `count_documents`
to get the number of total documents in the collection.

The `_id` field is a unique index, so it will match the number of documents.

#### Full Command
```js
  collection.count_documents({{}})
```

Results

Total number of zip codes in the collection: {collection.count_documents({})}
"""))
```

Strategy Since zipcodes are the `_id` field, we can use `count_documents` to get the number of total documents in the collection.

The `_id` field is a unique index, so it will match the number of documents.

Full Command

```
collection.count_documents({})
```

Result Total number of zip codes in the collection: 29353

1.0.4 4. Count the total number of zip codes in the New England states (CT, RI, MA, VT, NH, and ME).

```
[ ]: display(Markdown(f"""
#### Strategy

We can use the `$in` operator to match any of the values in the list
`["CT", "RI", "MA", "VT", "NH", "ME"]`.

We can then use `count_documents` to get the number of documents that match.

#### Full Command
```js
  collection.count_documents({{"state": {{"$in": ["CT", "RI", "MA", "VT", "NH",␣
 ↪"ME"]}}}})
```
```

```
#### Results

Total number of zip codes in `CT, RI, MA, VT, NH, ME`: \
{collection.count_documents({"state": {"$in": ["CT", "RI", "MA", "VT", "NH",␣
 ↪"ME"]}})}
"""))
```

Strategy We can use the `$in` operator to match any of the values in the list `["CT", "RI", "MA", "VT", "NH", "ME"]`.

We can then use `count_documents` to get the number of documents that match.

Full Command

```
collection.count_documents({"state": {"$in": ["CT", "RI", "MA", "VT", "NH", "ME"]}})
```

Result Total number of zip codes in `CT, RI, MA, VT, NH, ME`: 1677

1.0.5 5. Determine the total population of Rhode Island.

[ ]:
```
display(Markdown(f"""
#### Strategy

We can use the `collection.aggregate` function, with:

1. `$match` to get documents in `RI`
2. `$group` to group the matched documents.
3. `$sum` to sum the `pop` field while grouping.

#### Full Command:
```js
  collection.aggregate([
    {{"$match": {{"state": "RI"}}}},
    {{"$group": {{"_id": "RI", "totalPop": {{"$sum": "$pop"}}}}}}]
  ).next()["totalPop"]
```

Results

The total population in RI is: \
{collection.aggregate([{"$match": {"state": "RI"}}, {"$group": {"_id": "RI",␣
 ↪"totalPop": {"$sum": "$pop"}}}]).next()["totalPop"]}
"""))
```

Strategy We can use the `collection.aggregate` function, with:

1. `$match` to get documents in `RI`

2. `$group` to group the matched documents.
3. `$sum` to sum the `pop` field while grouping.

Full command:

```
collection.aggregate([
  {"$match": {"state": "RI"}},
  {"$group": {"_id": "RI", "totalPop": {"$sum": "$pop"}}}]
).next()["totalPop"]
```

Results The total population in RI is: 1003218

1.0.6 6. Determine which zip code has the smallest population.

[ ]:
```
display(Markdown(f"""

#### Strategy

We can use the `collection.aggregate` function, with:

1. `$sort` with `"pop": 1` to sort by population in ascending order.
2. `.next()` to get the first document.
3. `["_id"]` to get the zip code (`_id`) field.

#### Full Command:
```js
  collection.aggregate([{"$sort": {"pop": 1}}}])
            .next()["_id"]
```


Results

The zip code with the smallest population is: \
{collection.aggregate([{ "$sort": { "pop": 1}}]).next()["_id"] }

"""))
```

Strategy We can use the `collection.aggregate` function, with:

1. `$sort` with `"pop": 1` to sort by population in ascending order.
2. `.next()` to get the first document.
3. `["_id"]` to get the zip code (`_id`) field.

Full Command:

```
collection.aggregate([{"$sort": {"pop": 1}}])
          .next()["_id"]
```

Results The zip code with the smallest population is: 02163

1.0.7 7. Determine the southernmost zip code in the database.

[ ]:
```
display(Markdown(f"""
#### Strategy

The southernmost zip code will have the smallest longitude.
The longitude is the second index in the `loc` array.
We can use the `collection.aggregate` function, with:

1. `$sort` with `"loc.1": 1` to sort by longitude in ascending order.
2. `.next()` to get the first document.
3. `["_id"]` to get the zip code (`_id`) field.

#### Full command:
```js
  collection.aggregate([{"$sort": {"loc.1": 1}}}])
            .next()["_id"]
```

Results

The southernmost zip code is: \
{collection.aggregate([{ "$sort": { "loc.1": 1}}]).next()["_id"] }

"""))
```

Strategy The southernmost zip code will have the smallest longitude. The longitude is the second index in the `loc` array. We can use the `collection.aggregate` function, with:

1. `$sort` with `"loc.1": 1` to sort by longitude in ascending order.
2. `.next()` to get the first document.
3. `["_id"]` to get the zip code (`_id`) field.

Full command:

```
collection.aggregate([{"$sort": {"loc.1": 1}}])
          .next()["_id"]
```

Results The southernmost zip code is: 96772

1.0.8 8. Determine the average population of states beginning with the letter 'M'.

[ ]:
```
display(Markdown(f"""
#### Strategy

We can use the `collection.aggregate` function, with:
```

```
1. `$group` to group the documents by `state`, summing `pop`.
2. `$match` with `$regex: "^M"` to match states that start with `M`.
3. `$group` with `_id: null` to group all aggregations and average the total␣
 ↪populations.


#### Full Command:
```js
  collection.aggregate([
    {{"$group": {{"_id": "$state", "totalPop": {{"$sum": "$pop"}}}}}},
    {{"$match": {{"_id": {{"$regex": "^M"}}}}}},
    {{"$group": {{"_id": "null", "avgPop": {{"$avg": "$totalPop"}}}}}}
  ]).next()["avgPop"]
```

Results

The average population of states beginning with `M` is: \
{collection.aggregate([
  { "$group": { "_id": "$state", "totalPop": { "$sum": "$pop"}}},
  { "$match": { "_id": { "$regex": "^M"}}},
  { "$group": { "_id": "null", "avgPop": { "$avg": "$totalPop"}}}
]).next()["avgPop"]}

"""))
```

Strategy We can use the `collection.aggregate` function, with:

1. `$group` to group the documents by `state`, summing `pop`.
2. `$match` with `$regex: "^M"` to match states that start with `M`.
3. `$group` with `_id: null` to group all aggregations and average the total populations.

Full Command:

```
  collection.aggregate([
    {"$group": {"_id": "$state", "totalPop": {"$sum": "$pop"}}},
    {"$match": {"_id": {"$regex": "^M"}}},
    {"$group": {"_id": "null", "avgPop": {"$avg": "$totalPop"}}}
  ]).next()["avgPop"]
```

Results The average population of states beginning with `M` is: 4271942.875

1.0.9 9. Which zip codes have more than $50,000$ population?

[ ]:
```
display(Markdown(f"""
#### Strategy

We can use `collection.find` with `"pop": $gt 50000` to get all documents
with a population greater than 50,000.

We can select only the `_id` field with `"_id": 1`.

#### Full Command:
```js
  collection.find({{"pop": {{"$gt": 50000}}}}, {{"_id": 1}})
```


Results

The zip codes with a population greater than 50,000 are: \
{list(
  collection.find({"pop": {"$gt": 50000}}, {"_id": 1})
)}"""))
```

Strategy We can use `collection.find` with `"pop": $gt 50000` to get all documents with a population greater than 50,000.

We can select only the `_id` field with `"_id": 1`.

Full Command:

```
collection.find({"pop": {"$gt": 50000}}, {"_id": 1})
```

Results The zip codes with a population greater than 50,000 are: [{'_id': '01201'}, {'_id': '01701'}, {'_id': '02146'}, {'_id': '02148'}, {'_id': '02154'}, {'_id': '02155'}, {'_id': '02401'}, {'_id': '02895'}, {'_id': '06010'}, {'_id': '06040'}, {'_id': '06450'}, {'_id': '06511'}, {'_id': '06516'}, {'_id': '06902'}, {'_id': '07002'}, {'_id': '07047'}, {'_id': '07055'}, {'_id': '07087'}, {'_id': '07111'}, {'_id': '07305'}, {'_id': '07306'}, {'_id': '08360'}, {'_id': '08753'}, {'_id': '10002'}, {'_id': '10003'}, {'_id': '10009'}, {'_id': '10016'}, {'_id': '10021'}, {'_id': '10023'}, {'_id': '10024'}, {'_id': '10025'}, {'_id': '10027'}, {'_id': '10029'}, {'_id': '10031'}, {'_id': '10032'}, {'_id': '10033'}, {'_id': '10128'}, {'_id': '10314'}, {'_id': '10452'}, {'_id': '10453'}, {'_id': '10456'}, {'_id': '10457'}, {'_id': '10458'}, {'_id': '10462'}, {'_id': '10463'}, {'_id': '10466'}, {'_id': '10467'}, {'_id': '10468'}, {'_id': '10469'}, {'_id': '10472'}, {'_id': '10473'}, {'_id': '10701'}, {'_id': '10940'}, {'_id': '11203'}, {'_id': '11204'}, {'_id': '11206'}, {'_id': '11207'}, {'_id': '11208'}, {'_id': '11209'}, {'_id': '11210'}, {'_id': '11211'}, {'_id': '11212'}, {'_id': '11213'}, {'_id': '11214'}, {'_id': '11215'}, {'_id': '11216'}, {'_id': '11218'}, {'_id': '11219'}, {'_id': '11220'}, {'_id': '11221'}, {'_id': '11223'}, {'_id': '11224'}, {'_id': '11225'}, {'_id': '11226'}, {'_id': '11229'}, {'_id': '11230'}, {'_id': '11233'}, {'_id': '11234'}, {'_id': '11235'}, {'_id': '11236'}, {'_id': '11354'}, {'_id': '11355'}, {'_id': '11368'}, {'_id': '11372'}, {'_id': '11373'}, {'_id': '11375'}, {'_id': '11377'}, {'_id': '11385'}, {'_id': '11432'}, {'_id':

‘11434’}, {’_id’: ‘11550’}, {’_id’: ‘11691’}, {’_id’: ‘11717’}, {’_id’: ‘11746’}, {’_id’: ‘12180’}, {’_id’: ‘13440’}, {’_id’: ‘14120’}, {’_id’: ‘14150’}, {’_id’: ‘14221’}, {’_id’: ‘15601’}, {’_id’: ‘16001’}, {’_id’: ‘17013’}, {’_id’: ‘17042’}, {’_id’: ‘17055’}, {’_id’: ‘17603’}, {’_id’: ‘17701’}, {’_id’: ‘18042’}, {’_id’: ‘18103’}, {’_id’: ‘18702’}, {’_id’: ‘19020’}, {’_id’: ‘19104’}, {’_id’: ‘19111’}, {’_id’: ‘19120’}, {’_id’: ‘19124’}, {’_id’: ‘19134’}, {’_id’: ‘19140’}, {’_id’: ‘19143’}, {’_id’: ‘19145’}, {’_id’: ‘19464’}, {’_id’: ‘19711’}, {’_id’: ‘20002’}, {’_id’: ‘20011’}, {’_id’: ‘20019’}, {’_id’: ‘20020’}, {’_id’: ‘20906’}, {’_id’: ‘21061’}, {’_id’: ‘21122’}, {’_id’: ‘21206’}, {’_id’: ‘21207’}, {’_id’: ‘21215’}, {’_id’: ‘21217’}, {’_id’: ‘21218’}, {’_id’: ‘21222’}, {’_id’: ‘21224’}, {’_id’: ‘21229’}, {’_id’: ‘21234’}, {’_id’: ‘21801’}, {’_id’: ‘22003’}, {’_id’: ‘22110’}, {’_id’: ‘22901’}, {’_id’: ‘23320’}, {’_id’: ‘23452’}, {’_id’: ‘23454’}, {’_id’: ‘23456’}, {’_id’: ‘23462’}, {’_id’: ‘23464’}, {’_id’: ‘23602’}, {’_id’: ‘26505’}, {’_id’: ‘27292’}, {’_id’: ‘28540’}, {’_id’: ‘28655’}, {’_id’: ‘28677’}, {’_id’: ‘29501’}, {’_id’: ‘29801’}, {’_id’: ‘30032’}, {’_id’: ‘30060’}, {’_id’: ‘30062’}, {’_id’: ‘30223’}, {’_id’: ‘30236’}, {’_id’: ‘30318’}, {’_id’: ‘30906’}, {’_id’: ‘31907’}, {’_id’: ‘32210’}, {’_id’: ‘32211’}, {’_id’: ‘32216’}, {’_id’: ‘33012’}, {’_id’: ‘33023’}, {’_id’: ‘33024’}, {’_id’: ‘33064’}, {’_id’: ‘33142’}, {’_id’: ‘33157’}, {’_id’: ‘33165’}, {’_id’: ‘33311’}, {’_id’: ‘37211’}, {’_id’: ‘38109’}, {’_id’: ‘38128’}, {’_id’: ‘43055’}, {’_id’: ‘43130’}, {’_id’: ‘43302’}, {’_id’: ‘43701’}, {’_id’: ‘44035’}, {’_id’: ‘44060’}, {’_id’: ‘44102’}, {’_id’: ‘44105’}, {’_id’: ‘44107’}, {’_id’: ‘44130’}, {’_id’: ‘45601’}, {’_id’: ‘46227’}, {’_id’: ‘46360’}, {’_id’: ‘46383’}, {’_id’: ‘47130’}, {’_id’: ‘47374’}, {’_id’: ‘47905’}, {’_id’: ‘47906’}, {’_id’: ‘48043’}, {’_id’: ‘48044’}, {’_id’: ‘48060’}, {’_id’: ‘48066’}, {’_id’: ‘48093’}, {’_id’: ‘48161’}, {’_id’: ‘48180’}, {’_id’: ‘48185’}, {’_id’: ‘48192’}, {’_id’: ‘48203’}, {’_id’: ‘48205’}, {’_id’: ‘48213’}, {’_id’: ‘48219’}, {’_id’: ‘48224’}, {’_id’: ‘48227’}, {’_id’: ‘48228’}, {’_id’: ‘48235’}, {’_id’: ‘48238’}, {’_id’: ‘48601’}, {’_id’: ‘48823’}, {’_id’: ‘49017’}, {’_id’: ‘49504’}, {’_id’: ‘49505’}, {’_id’: ‘49509’}, {’_id’: ‘49684’}, {’_id’: ‘50010’}, {’_id’: ‘53209’}, {’_id’: ‘54401’}, {’_id’: ‘54901’}, {’_id’: ‘55337’}, {’_id’: ‘60004’}, {’_id’: ‘60016’}, {’_id’: ‘60056’}, {’_id’: ‘60067’}, {’_id’: ‘60085’}, {’_id’: ‘60103’}, {’_id’: ‘60148’}, {’_id’: ‘60187’}, {’_id’: ‘60402’}, {’_id’: ‘60411’}, {’_id’: ‘60435’}, {’_id’: ‘60441’}, {’_id’: ‘60453’}, {’_id’: ‘60505’}, {’_id’: ‘60608’}, {’_id’: ‘60609’}, {’_id’: ‘60614’}, {’_id’: ‘60617’}, {’_id’: ‘60618’}, {’_id’: ‘60619’}, {’_id’: ‘60620’}, {’_id’: ‘60621’}, {’_id’: ‘60622’}, {’_id’: ‘60623’}, {’_id’: ‘60624’}, {’_id’: ‘60625’}, {’_id’: ‘60626’}, {’_id’: ‘60628’}, {’_id’: ‘60629’}, {’_id’: ‘60632’}, {’_id’: ‘60634’}, {’_id’: ‘60636’}, {’_id’: ‘60637’}, {’_id’: ‘60638’}, {’_id’: ‘60639’}, {’_id’: ‘60640’}, {’_id’: ‘60641’}, {’_id’: ‘60643’}, {’_id’: ‘60644’}, {’_id’: ‘60647’}, {’_id’: ‘60649’}, {’_id’: ‘60650’}, {’_id’: ‘60651’}, {’_id’: ‘60657’}, {’_id’: ‘62301’}, {’_id’: ‘63031’}, {’_id’: ‘63136’}, {’_id’: ‘66502’}, {’_id’: ‘70065’}, {’_id’: ‘70072’}, {’_id’: ‘70117’}, {’_id’: ‘70560’}, {’_id’: ‘72401’}, {’_id’: ‘75007’}, {’_id’: ‘75051’}, {’_id’: ‘75150’}, {’_id’: ‘75211’}, {’_id’: ‘75216’}, {’_id’: ‘75217’}, {’_id’: ‘75228’}, {’_id’: ‘77036’}, {’_id’: ‘77901’}, {’_id’: ‘78207’}, {’_id’: ‘78228’}, {’_id’: ‘78501’}, {’_id’: ‘78520’}, {’_id’: ‘78521’}, {’_id’: ‘78539’}, {’_id’: ‘78572’}, {’_id’: ‘79907’}, {’_id’: ‘79924’}, {’_id’: ‘79936’}, {’_id’: ‘80123’}, {’_id’: ‘80221’}, {’_id’: ‘80631’}, {’_id’: ‘84118’}, {’_id’: ‘84120’}, {’_id’: ‘85023’}, {’_id’: ‘85032’}, {’_id’: ‘85204’}, {’_id’: ‘85224’}, {’_id’: ‘85364’}, {’_id’: ‘85705’}, {’_id’: ‘85706’}, {’_id’: ‘85710’}, {’_id’: ‘87105’}, {’_id’: ‘87501’}, {’_id’: ‘88001’}, {’_id’: ‘88201’}, {’_id’: ‘89115’}, {’_id’: ‘89121’}, {’_id’: ‘90001’}, {’_id’: ‘90003’}, {’_id’: ‘90004’}, {’_id’: ‘90006’}, {’_id’: ‘90011’}, {’_id’: ‘90019’}, {’_id’: ‘90022’}, {’_id’: ‘90026’}, {’_id’: ‘90027’}, {’_id’: ‘90033’}, {’_id’: ‘90034’}, {’_id’: ‘90037’}, {’_id’: ‘90042’}, {’_id’: ‘90044’}, {’_id’: ‘90063’}, {’_id’: ‘90066’}, {’_id’: ‘90201’}, {’_id’: ‘90250’}, {’_id’: ‘90255’}, {’_id’: ‘90262’}, {’_id’: ‘90274’}, {’_id’: ‘90280’}, {’_id’: ‘90631’}, {’_id’: ‘90640’}, {’_id’: ‘90650’}, {’_id’: ‘90660’}, {’_id’: ‘90701’}, {’_id’: ‘90706’}, {’_id’: ‘90731’}, {’_id’: ‘90745’}, {’_id’: ‘90805’}, {’_id’: ‘90813’}, {’_id’: ‘91331’}, {’_id’: ‘91335’}, {’_id’: ‘91342’}, {’_id’: ‘91402’}, {’_id’: ‘91605’}, {’_id’: ‘91702’}, {’_id’:

‘91706’}, {’_id’: ‘91710’}, {’_id’: ‘91720’}, {’_id’: ‘91732’}, {’_id’: ‘91744’}, {’_id’: ‘91745’}, {’_id’: ‘91754’}, {’_id’: ‘91766’}, {’_id’: ‘91770’}, {’_id’: ‘91786’}, {’_id’: ‘91801’}, {’_id’: ‘91910’}, {’_id’: ‘91911’}, {’_id’: ‘91950’}, {’_id’: ‘91977’}, {’_id’: ‘92020’}, {’_id’: ‘92021’}, {’_id’: ‘92054’}, {’_id’: ‘92071’}, {’_id’: ‘92083’}, {’_id’: ‘92105’}, {’_id’: ‘92114’}, {’_id’: ‘92115’}, {’_id’: ‘92126’}, {’_id’: ‘92154’}, {’_id’: ‘92324’}, {’_id’: ‘92335’}, {’_id’: ‘92345’}, {’_id’: ‘92376’}, {’_id’: ‘92392’}, {’_id’: ‘92404’}, {’_id’: ‘92503’}, {’_id’: ‘92509’}, {’_id’: ‘92553’}, {’_id’: ‘92627’}, {’_id’: ‘92630’}, {’_id’: ‘92646’}, {’_id’: ‘92647’}, {’_id’: ‘92680’}, {’_id’: ‘92683’}, {’_id’: ‘92701’}, {’_id’: ‘92703’}, {’_id’: ‘92704’}, {’_id’: ‘92707’}, {’_id’: ‘92708’}, {’_id’: ‘92714’}, {’_id’: ‘92804’}, {’_id’: ‘92805’}, {’_id’: ‘93030’}, {’_id’: ‘93033’}, {’_id’: ‘93065’}, {’_id’: ‘93257’}, {’_id’: ‘93277’}, {’_id’: ‘93307’}, {’_id’: ‘93309’}, {’_id’: ‘93454’}, {’_id’: ‘93550’}, {’_id’: ‘93612’}, {’_id’: ‘93727’}, {’_id’: ‘94015’}, {’_id’: ‘94080’}, {’_id’: ‘94086’}, {’_id’: ‘94110’}, {’_id’: ‘94112’}, {’_id’: ‘94122’}, {’_id’: ‘94501’}, {’_id’: ‘94509’}, {’_id’: ‘94533’}, {’_id’: ‘94536’}, {’_id’: ‘94544’}, {’_id’: ‘94550’}, {’_id’: ‘94558’}, {’_id’: ‘94565’}, {’_id’: ‘94587’}, {’_id’: ‘95035’}, {’_id’: ‘95076’}, {’_id’: ‘95122’}, {’_id’: ‘95123’}, {’_id’: ‘95127’}, {’_id’: ‘95207’}, {’_id’: ‘95336’}, {’_id’: ‘95340’}, {’_id’: ‘95350’}, {’_id’: ‘95351’}, {’_id’: ‘95380’}, {’_id’: ‘95608’}, {’_id’: ‘95616’}, {’_id’: ‘95687’}, {’_id’: ‘95823’}, {’_id’: ‘95926’}, {’_id’: ‘96734’}, {’_id’: ‘96744’}, {’_id’: ‘96797’}, {’_id’: ‘96818’}, {’_id’: ‘96819’}, {’_id’: ‘98031’}]

1.0.10 10. Which city has the most zip codes?

HINTS

1. Include the state in case the city name isn’t unique across the states.
2. Be sure to handle the possibility of a tie.

[ ]:
```
display(Markdown(f"""
#### Strategy

We can use `collection.aggregate` with:

1. `$group` to group the documents by `city` and `state`, counting aggregated␣
 ↪documents.
2. `$sort` with `"totalCount": -1` to sort by total count in descending order.

#### Full Command:
```js
  collection.aggregate([
    {{"$group": {{"_id": {{"city": "$city", "state": "$state"}}, "totalCount":␣
 ↪{{"$sum": 1}}}}}},
    {{"$sort": {{"totalCount": -1}}}}
  ]).next()["_id"]
```

Results
```

```
The city with the largest population is: \
{collection.aggregate([
  { "$group": { "_id": { "city": "$city", "state": "$state"}, "totalCount": {␣
 ↪"$sum": 1}}},
  { "$sort": { "totalCount": -1}}
]).next()["_id"]}

"""))
```

Strategy We can use `collection.aggregate` with:

1. `$group` to group the documents by `city` and `state`, counting aggregated documents.
2. `$sort` with `"totalCount": -1` to sort by total count in descending order.

Full Command:

```
collection.aggregate([
  {"$group": {"_id": {"city": "$city", "state": "$state"}, "totalCount": {"$sum": 1}}},
  {"$sort": {"totalCount": -1}}
]).next()["_id"]
```

Results The city with the largest population is: {'city': 'HOUSTON', 'state': 'TX'}

[ ]: