

Exercises with a document database

We use MongoDB as example for a document store to demonstrate map reduce jobs for aggregating data.

Get a MongoDB installation

See [INSTALL.md](#) for Details and Troubleshooting

Exercises with MongoDB

Rewrite the example from [MongoDB Website](#). Work with the mongo shell within the provided image or get an Account on MongoDB Atlas.

Step 1: Adding Data

1. Start `mongo`

```
docker exec -it mongodb /bin/bash -c "mongosh"
```

2. Show available databases

```
show dbs
```

```
test> show dbs
admin    40.00 KiB
config   60.00 KiB
local    40.00 KiB
```

3. Create database onlineshop

```
use onlineshop
```

4. Insert the following data sets

```
db.orders.insertMany([
  { _id: 1, user_id: 1, date_ordered: new Date("2020-03-01"), total_sum: 25,
    items: [ { sku: "oranges", qty: 5, price: 2.5 }, { sku: "apples", qty: 5, price:
    2.5 } ], status: "A" },
  { _id: 2, user_id: 1, date_ordered: new Date("2020-03-08"), total_sum: 70,
    items: [ { sku: "oranges", qty: 8, price: 2.5 }, { sku: "chocolates", qty: 5,
    price: 10 } ], status: "A" },
  { _id: 3, user_id: 2, date_ordered: new Date("2020-03-08"), total_sum: 50,
    items: [ { sku: "oranges", qty: 10, price: 2.5 }, { sku: "pears", qty: 10, price:
    2.5 } ], status: "A" },
  { _id: 4, user_id: 2, date_ordered: new Date("2020-03-18"), total_sum: 25,
    items: [ { sku: "oranges", qty: 10, price: 2.5 } ], status: "A" },
  { _id: 5, user_id: 2, date_ordered: new Date("2020-03-19"), total_sum: 50,
    items: [ { sku: "chocolates", qty: 5, price: 10 } ], status: "A" },
  { _id: 6, user_id: 3, date_ordered: new Date("2020-03-19"), total_sum: 35,
    items: [ { sku: "carrots", qty: 10, price: 1.0 }, { sku: "apples", qty: 10, price:
```

```

2.5 } ], status: "A" },
    { _id: 7, user_id: 3, date_ordered: new Date("2020-03-20"), total_sum: 25,
items: [ { sku: "oranges", qty: 10, price: 2.5 } ], status: "A" },
    { _id: 8, user_id: 4, date_ordered: new Date("2020-03-20"), total_sum: 75,
items: [ { sku: "chocolates", qty: 5, price: 10 }, { sku: "apples", qty: 10,
price: 2.5 } ], status: "A" },
    { _id: 9, user_id: 4, date_ordered: new Date("2020-03-20"), total_sum: 55,
items: [ { sku: "carrots", qty: 5, price: 1.0 }, { sku: "apples", qty: 10, price:
2.5 }, { sku: "oranges", qty: 10, price: 2.5 } ], status: "A" },
    { _id: 10, user_id: 4, date_ordered: new Date("2020-03-23"), total_sum: 25,
items: [ { sku: "oranges", qty: 10, price: 2.5 } ], status: "A" }
]);

```

5. Find all rows of collection orders.

`db.orders.find()`

```

onlineshop> db.orders.find()
[
  {
    _id: 1,
    user_id: 1,
    date_ordered: ISODate("2020-03-01T00:00:00.000Z"),
    total_sum: 25,
    items: [
      { sku: 'oranges', qty: 5, price: 2.5 },
      { sku: 'apples', qty: 5, price: 2.5 }
    ],
    status: 'A'
  },
  {
    _id: 2,
    user_id: 1,
    date_ordered: ISODate("2020-03-08T00:00:00.000Z"),
    total_sum: 70,
    items: [
      { sku: 'oranges', qty: 8, price: 2.5 },
      { sku: 'chocolates', qty: 5, price: 10 }
    ],
    status: 'A'
  },
  {
    _id: 3,
    user_id: 2,
    date_ordered: ISODate("2020-03-08T00:00:00.000Z"),
    total_sum: 50,
    items: [
      { sku: 'oranges', qty: 10, price: 2.5 },
      { sku: 'pears', qty: 10, price: 2.5 }
    ],

```

Step 2: Working with map_reduce

Map-Reduce is no longer supported for versions above 5.0 or in MongoDB Atlas Cloud.

Read the Doku to understand a principle introduced by Google for GFS and the open source hadoop Filesystem, that is based on the idea of GFS.

See [Map Reduce Documention](#) to see, how that works.

When using sharded collection as the input for a map-reduce operation, mongos will automatically dispatch the map-reduce job to each shard in parallel. It will also automatically wait for all jobs on all shards to finish.

Step 3: Use MongoDB Atlas Aggregation to build a pipeline

Create an Atlas Account for free [here](#), if you'd like to work in the cloud.

If you use the shell see [Doku](#)

1. Write a query using aggregation to build the sum of total_sum.

```
[
  {
    $group:
    {
      _id: null,
      total_sum: {
        $sum: "$total_sum",
      },
    },
  },
]
```

2. Write a query using aggregation to build the sum of prices*qty for every item of a user.

```
[
  {
    $unwind: "$items",
  },
  {
    $project: {
      _id: 1,
      total_price: {
        $multiply: ["$items.price", "$items.qty"],
      },
    },
  },
  {
    $group: {
      _id: "$_id",
    },
  },
]
```

```

    total_sum: {
      $sum: "$total_price",
    },
  },
},
{
  $project: {
    _id: 0,
    id: "$_id",
    total_sum: 1,
  },
},
{
  $sort:
  {
    id: 1,
  },
},
]

```

The screenshot shows the MongoDB Atlas interface for the 'onlineshop.orders' collection. The aggregation pipeline is defined as follows:

```

[
  {
    $unwind: '$items'
  },
  {
    $project: {
      _id: 1,
      total_price: {
        $multiply: ['$items.price', '$items.qty'],
      },
    },
  },
  {
    $group: {
      _id: "$_id",
      total_sum: {
        $sum: "$total_price",
      },
    },
  },
  {
    $project: {
      _id: 0,
      id: "$_id",
      total_sum: 1,
    },
  },
  {
    $sort: {
      id: 1,
    },
  },
]

```

The right sidebar displays the 'PIPELINE OUTPUT' with a sample of 10 documents:

total_sum	id
25	1
70	2
50	3
25	4
50	5
50	6

3. See [Map Reduce Doku](#) for an example aggregation. Scroll down for examples.
4. For more details see [Aggregation Quick Reference](#)
5. Or use the Aggregation Tab in MongoDB Atlas

Step 4: Deleting rows and the Database

1. Delete all rows in the collection orders

```
db.orders.remove({})
```

```
Atlas atlas-oxmaid-shard-0 [primary] onlineshop> db.orders.remove({})
{ acknowledged: true, deletedCount: 10 }
```

2. Drop the database onlineshop

```
db.dropDatabase()
```

```
Atlas atlas-oxmaid-shard-0 [primary] onlineshop> db.dropDatabase()
{ ok: 1, dropped: 'onlineshop' }
```

Step 5 Exercise for additional points

Build your own example to demonstrate a complex aggregation with a stage `geonear`, `search` (based on Lucene, that is only available on Atlas Cloud)

Add a few location entries for geonear aggregation:

```
db.places.insertMany( [
  {
    name: "Central Park",
    location: { type: "Point", coordinates: [ -73.97, 40.77 ] },
    category: "Parks"
  },
  {
    name: "Sara D. Roosevelt Park",
    location: { type: "Point", coordinates: [ -73.9928, 40.7193 ] },
    category: "Parks"
  },
  {
    name: "Polo Grounds",
    location: { type: "Point", coordinates: [ -73.9375, 40.8303 ] },
    category: "Stadiums"
  }
] )
```

Add a 2dsphere index on the location field: `db.places.createIndex({ location: "2dsphere" })`

Create an aggregation which returns all parks within 10km of the given coordinates. There is also a distance field added which returns the distance to the target.

```
[
  {
    $geoNear:
      /**
       * near: The point to search near.
       * distanceField: The calculated distance.
       * maxDistance: The maximum distance, in meters, documents can be before
       being excluded from results.
       * query: Limits results that match the query
       * includeLocs: Optional. Labels and includes the point used to match the
       document.
       * num: Optional. The maximum number of documents to return.
       * spherical: Defaults to false. Specifies whether to use spherical
       geometry.
       */
      {
        near: {
```

```

    type: "Point",
    coordinates: [-73.99279, 40.719296],
  },
  distanceField: "dist.calculated",
  maxDistance: 10000,
  query: {
    category: "Parks",
  },
  includeLocs: "dist.location",
  spherical: true,
},
],
]

```

[+ CREATE NEW](#)
[EXPORT TO LANGUAGE](#)

PREVIEW
STAGES
TEXT

```

1 [
2 {
3   $geoNear:
4     /**
5      * near: The point to search near.
6      * distanceField: The calculated distance.
7      * maxDistance: The maximum distance, in meters, documents
8      * query: Limits results that match the query
9      * includeLocs: Optional. Labels and includes the point use
10     * num: Optional. The maximum number of documents to return
11     * spherical: Defaults to false. Specifies whether to use s
12     */
13     {
14       near: {
15         type: "Point",
16         coordinates: [-73.99279, 40.719296],
17       },
18       distanceField: "dist.calculated",
19       maxDistance: 10000,
20       query: {
21         category: "Parks",
22       },
23       includeLocs: "dist.location",
24       spherical: true,
25     },
26   },
27 ]

```

PIPELINE OUTPUT

Sample of 2 documents

OUTPUT OPTIONS

```

category: "Parks"
dist: Object
  calculated: 0.9539931676365992
  location: Object
    _id: ObjectId('643a69212f32049bef2730fd')
    name: "Sara D. Roosevelt Park"
    location: Object

_id: ObjectId('643a69212f32049bef2730fc')
name: "Central Park"
location: Object
  category: "Parks"
  dist: Object
    calculated: 5962.597839230235
    location: Object

```

Work with MongoDB Atlas Data API

Create an Atlas Account for free [here](#) Create an API Key.

Create a database onlineshop, if not already done before. Create a collection product with pname, price and status ('published', 'revision') Create a collection user with frist_name, last_name, email, password, date_registered and a subcollection for phone numbers (mobile, private, fax)

Add some data manually.

Products:

```

db.products.insert([
  { _id: 0, pname: "test1", price: 100, status: { published: false, revision: 1 } },
  { _id: 1, pname: "test2", price: 150, status: { published: false, revision: 1 } },
  { _id: 2, pname: "test3", price: 220.0, status: { published: false, revision: 1 } },
])

```

```
{_id: 3, pname: "test4", price: 330.32, status: { published: true, revision:
3}},
{_id: 4, pname: "test5", price: 40, status: { published: true, revision: 1}},
{_id: 5, pname: "test6", price: 19.99, status: { published: true, revision:
1}},
{_id: 6, pname: "test7", price: 1.99, status: { published: true, revision: 1}}
])
```

Users:

```
db.users.insert([
  {_id: 0, first_name: "fname1", last_name: "lname1", email:
"fname1.@lname1.com", password: "franz", date_registered: "2023-04-15",
phone_numbers: [ { _id: 0, phone_number: "06998805551" } ] },
  {_id: 1, first_name: "fname2", last_name: "lname2", email:
"fname2.@lname2.com", password: "frenz", date_registered: "2023-04-15",
phone_numbers: [ { _id: 1, phone_number: "06998805551" } ] },
  {_id: 2, first_name: "fname3", last_name: "lname3", email:
"fname3.@lname3.com", password: "frunz", date_registered: "2023-04-15",
phone_numbers: [ { _id: 2, phone_number: "06998805551" } ] }
])
```

Test and work with API:

- [Data API Basics](#)
- [Advanced Atlas Client](#)
- [Standard Data API Resources](#)

In Linux bash:

Write curl query for

1. findOne

```
curl --location --request POST 'https://eu-central-1.aws.data.mongodb-
api.com/app/data-hloeg/endpoint/data/v1/action/findOne' \
--header 'Content-Type: application/json' \
--header 'Access-Control-Request-Headers: *' \
--header 'api-key:
G76S5a1SDokn3f8s2S8uCdCL4Uezfs67NnvDgN35Hw42M8x2ej75pyGU9RbJNKg5' \
--data-raw '{
  "collection": "products",
  "database": "onlineshop",
  "dataSource": "Cluster0",
  "filter": { "_id": 1 }
}'
```

```
$ curl --location --request POST 'https://eu-central-1.aws.data.mongodb-api.com/app/data-hloeg/endpoint/data/v1/action/findOne' \
--header 'Content-Type: application/json' \
--header 'Access-Control-Request-Headers: *' \
--header 'api-key: G76S5a1SDokn3f8s2S8uCdCL4Uezfs67NnvdgN35Hw42M8x2ej75pyGU9RbJNKg5' \
--data-raw '{
  "collection": "products",
  "database": "onlineshop",
  "dataSource": "Cluster0",
  "filter": { "_id": 1 }
}'
{"document":{"_id":1,"pname":"test2","price":150,"status":{"published":false,"revision":1}}}
```

2. findMany

```
curl --location --request POST 'https://eu-central-1.aws.data.mongodb-
api.com/app/data-hloeg/endpoint/data/v1/action/find' \
--header 'Content-Type: application/json' \
--header 'Access-Control-Request-Headers: *' \
--header 'api-key:
G76S5a1SDokn3f8s2S8uCdCL4Uezfs67NnvdgN35Hw42M8x2ej75pyGU9RbJNKg5' \
--data-raw '{
  "collection": "products",
  "database": "onlineshop",
  "dataSource": "Cluster0"
}'
```

```
$ curl --location --request POST 'https://eu-central-1.aws.data.mongodb-api.com/app/data-hloeg/endpoint/data/v1/action/find' \
--header 'Content-Type: application/json' \
--header 'Access-Control-Request-Headers: *' \
--header 'api-key: G76S5a1SDokn3f8s2S8uCdCL4Uezfs67NnvdgN35Hw42M8x2ej75pyGU9RbJNKg5' \
--data-raw '{
  "collection": "products",
  "database": "onlineshop",
  "dataSource": "Cluster0"
}'
{"documents":[{"_id":0,"pname":"test1","price":100,"status":{"published":false,"revision":1}},{"_id":1,"pname":"test2","price":150,"status":{"published":false,"revision":1}},{"_id":3,"pname":"test4","price":330.32,"status":{"published":true,"revision":3}},{"_id":4,"pname":"test5","price":19.99,"status":{"published":true,"revision":1}},{"_id":6,"pname":"test7","price":1.99,"status":{"published":true,"revision":1}}]}
```

3. insertOne

```
curl --request POST \
  'https://eu-central-1.aws.data.mongodb-api.com/app/data-
hloeg/endpoint/data/v1/action/insertOne' \
--header 'Content-Type: application/json' \
--header 'Access-Control-Request-Headers: *' \
--header 'api-key:
G76S5a1SDokn3f8s2S8uCdCL4Uezfs67NnvdgN35Hw42M8x2ej75pyGU9RbJNKg5' \
--data-raw '{
  "collection": "products",
  "database": "onlineshop",
  "dataSource": "Cluster0",
  "document": {
    "_id": 15,
    "pname": "inserted",
    "price": 100,
    "status": {
      "published": false,
      "revision": 1
    }
  }
}'
```



```
}
}'
```

```
$ curl --request POST \
'https://eu-central-1.aws.data.mongodb-api.com/app/data-hloeg/endpoint/data/v1/action/insertOne' \
--header 'Content-Type: application/json' \
--header 'Access-Control-Request-Headers: *' \
--header 'api-key: G76S5a1SDokn3f8s2S8uCdCL4Uezfs67NnvDgN35Hw42M8x2ej75pyGU9RbJNKg5' \
--data-raw '{
  "collection": "products",
  "database": "onlineshop",
  "dataSource": "Cluster0",
  "document": {
    "_id": 15,
    "pname": "inserted",
    "price": 100,
    "status": {
      "published": false,
      "revision": 1
    }
  }
}'
{"insertedId":15}
```

4. updateOne

```
curl --request POST \
'https://eu-central-1.aws.data.mongodb-api.com/app/data-
hloeg/endpoint/data/v1/action/updateOne' \
--header 'Content-Type: application/json' \
--header 'Access-Control-Request-Headers: *' \
--header 'api-key:
G76S5a1SDokn3f8s2S8uCdCL4Uezfs67NnvDgN35Hw42M8x2ej75pyGU9RbJNKg5' \
--data-raw '{
  "collection": "products",
  "database": "onlineshop",
  "dataSource": "Cluster0",
  "filter": { "_id": 15 },
  "update": {
    "$set": {
      "status": {
        "published": true,
        "revision": 2
      }
    }
  }
}'
```

```
$ curl --request POST \
  'https://eu-central-1.aws.data.mongodb-api.com/app/data-hloeg/endpoint/data/v1/action/updateOne' \
  --header 'Content-Type: application/json' \
  --header 'Access-Control-Request-Headers: *' \
  --header 'api-key: G76S5alSDokn3f8s2S8uCdCL4Uezfs67NnvdgN35Hw42M8x2ej75pyGU9RbJNKg5' \
  --data-raw '{
    "collection": "products",
    "database": "onlineshop",
    "dataSource": "Cluster0",
    "filter": { "_id": 15 },
    "update": {
      "$set": {
        "status": {
          "published": true,
          "revision": 2
        }
      }
    }
  }'
{"matchedCount":1,"modifiedCount":1}
```

5. deleteOne

```
curl --location --request POST 'https://eu-central-1.aws.data.mongodb-
api.com/app/data-hloeg/endpoint/data/v1/action/deleteOne' \
--header 'Content-Type: application/json' \
--header 'Access-Control-Request-Headers: *' \
--header 'api-key:
G76S5alSDokn3f8s2S8uCdCL4Uezfs67NnvdgN35Hw42M8x2ej75pyGU9RbJNKg5' \
--data-raw '{
  "collection": "products",
  "database": "onlineshop",
  "dataSource": "Cluster0",
  "filter": { "_id": 15}
}'
```

```
$ curl --location --request POST 'https://eu-central-1.aws.data.mongodb-api.com/app/data-hloeg/endpoint/data/v1/action/deleteOne' \
--header 'Content-Type: application/json' \
--header 'Access-Control-Request-Headers: *' \
--header 'api-key: G76S5alSDokn3f8s2S8uCdCL4Uezfs67NnvdgN35Hw42M8x2ej75pyGU9RbJNKg5' \
--data-raw '{
  "collection": "products",
  "database": "onlineshop",
  "dataSource": "Cluster0",
  "filter": { "_id": 15}
}'
{"deletedCount":1}
```