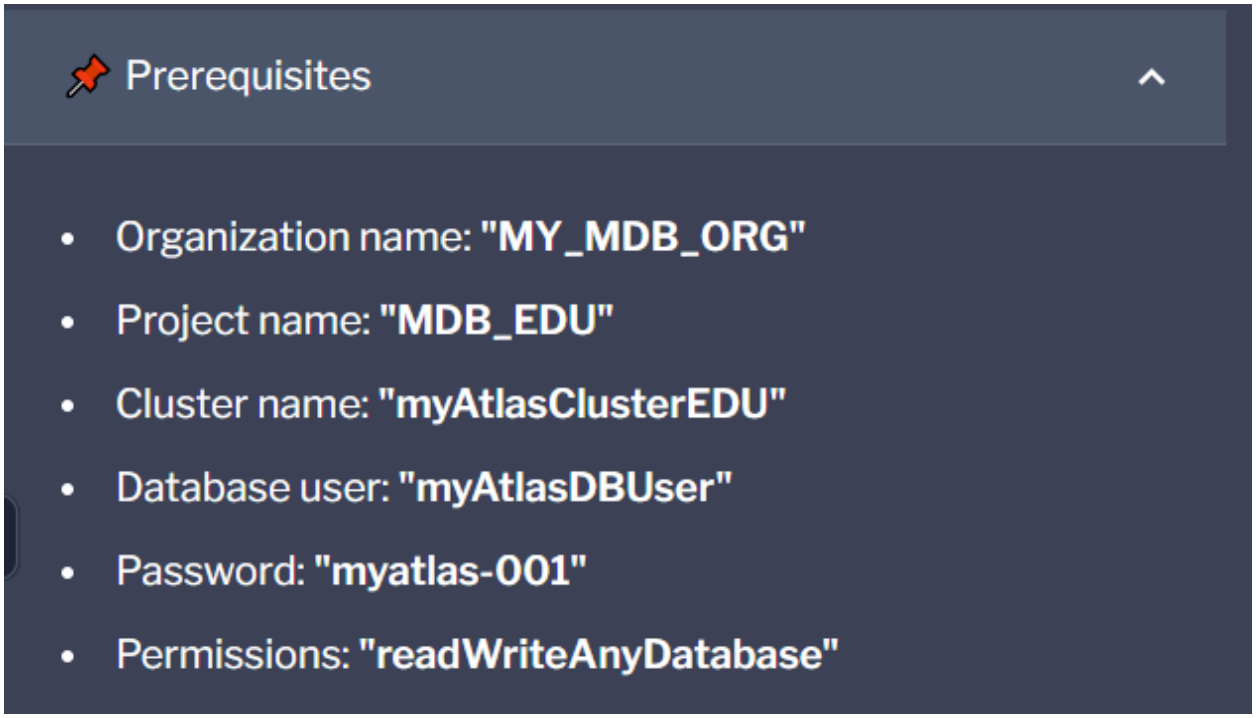
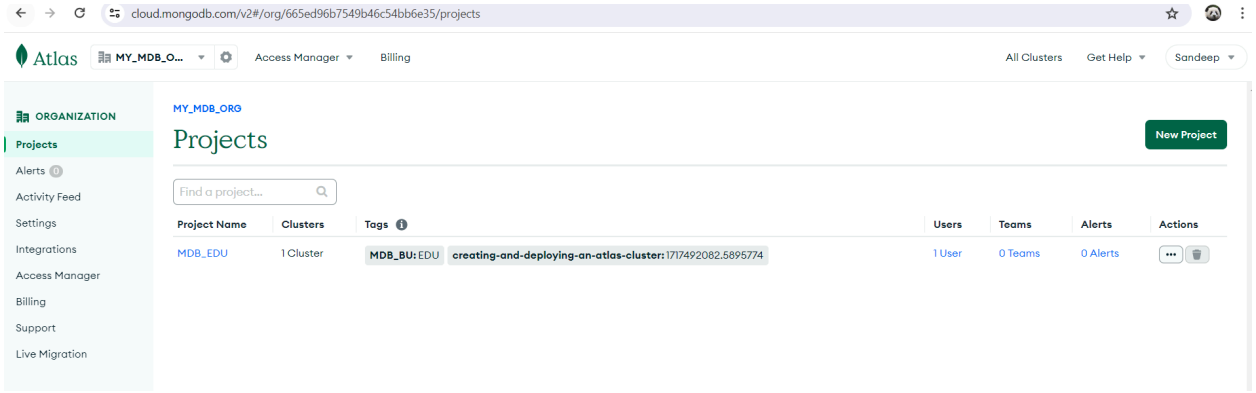


$ root@mongodb:~$ atlas auth login

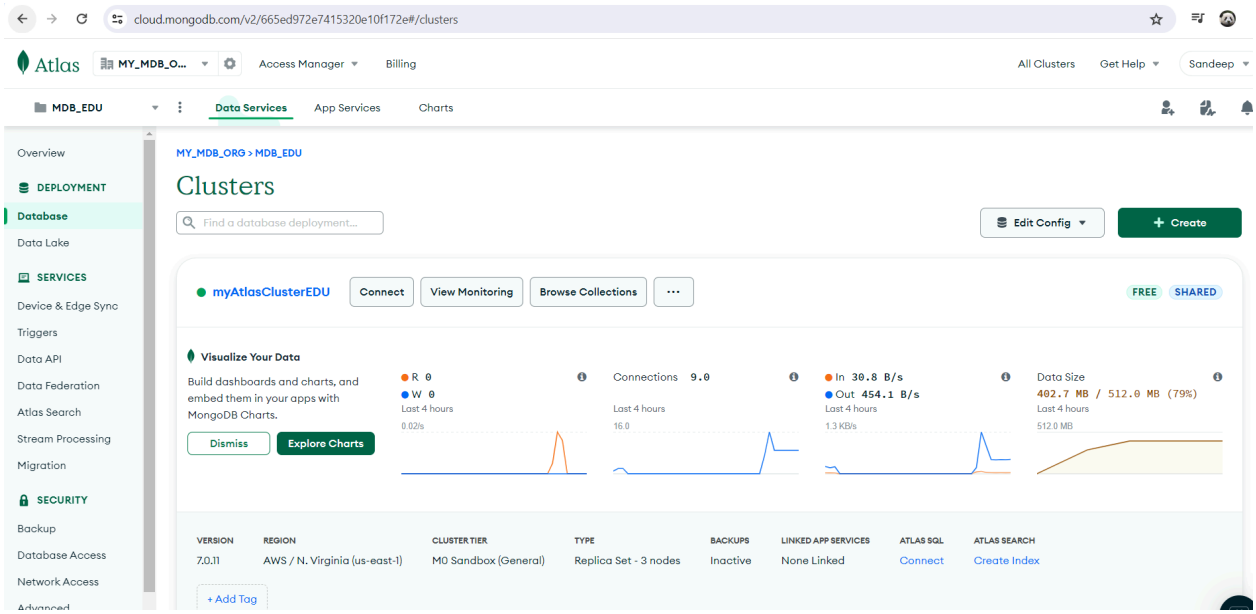
// Cluster Creation

```
root@mongodb:~$ atlas setup --clusterName myAtlasClusterEDU --provider AWS --currentIp --skipSampleData --username myAtlasDBU
ser --password myatlas-001 | tee atlas_cluster_details.txt

[Default Settings]
Cluster Name:                          myAtlasClusterEDU
Cloud Provider and Region:             AWS - US_EAST_1
Database User Username:                myAtlasDBUser
Allow connections from (IP Address):   34.34.155.175
? Do you want to set up your Atlas database with default settings? Yes
We are deploying myAtlasClusterEDU...

Store your database authentication access details in a secure location:
Database User Username: myAtlasDBUser
Database User Password: myatlas-001

Creating your cluster... [It's safe to 'Ctrl + C']
.....Cluster created.
Your connection string: mongodb+srv://myatlasclusteredu.ymqgri1.mongodb.net

MongoDB Shell detected. Connecting to your Atlas cluster:
$ mongosh -u myAtlasDBUser -p myatlas-001 mongodb+srv://myatlasclusteredu.ymqgri1.mongodb.net
Current Mongosh Log ID: 665ed9e432a6eb7f612202d7
Connecting to:          mongodb+srv://<credentials>@myatlasclusteredu.ymqgri1.mongodb.net/?appName=mongosh+2.2.5
Using MongoDB:          7.0.11
Using Mongosh:          2.2.5

For mongosh info see: https://docs.mongodb.com/mongodb-shell/


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/leg
al/privacy-policy).
You can opt-out by running the disableTelemetry() command.

Atlas atlas-x02tt5-shard-0 [primary] test> ls
```

// Organization–Project



// Cluster in Portal

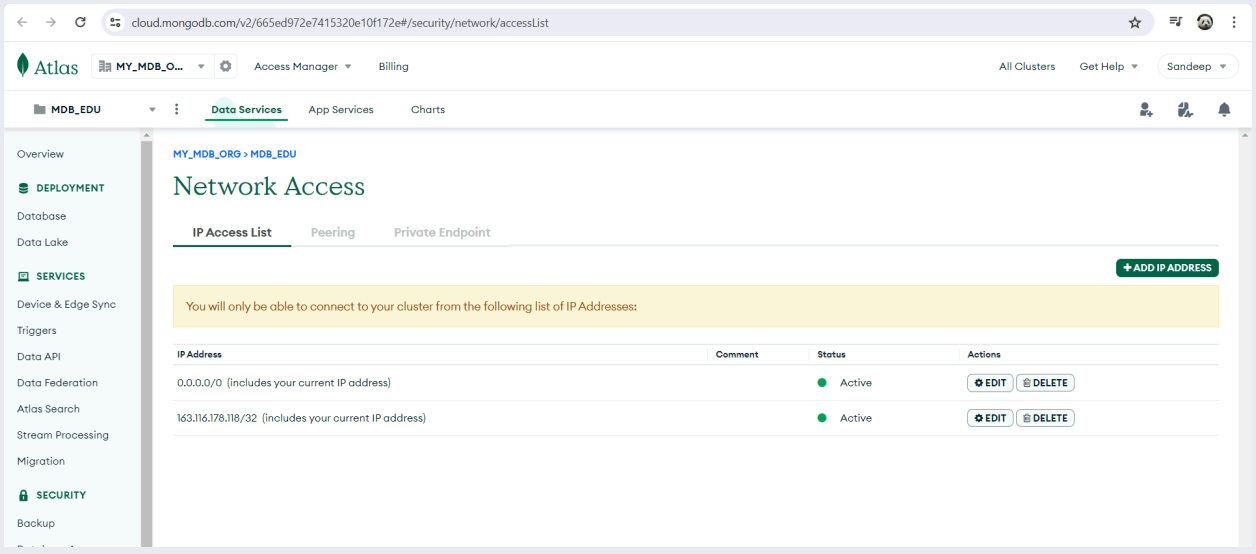


// Load the Sample Dataset Into Your Atlas Cluster

$ atlas clusters sampleData load myAtlasClusterEDU

```
root@mongodb:~$ atlas clusters sampleData load myAtlasClusterEDU
Sample Data Job 665edb002d31fe533a80d5b1 created.
root@mongodb:~$
```

// Network access list, add your IP. Adding from anywhere is not good idea

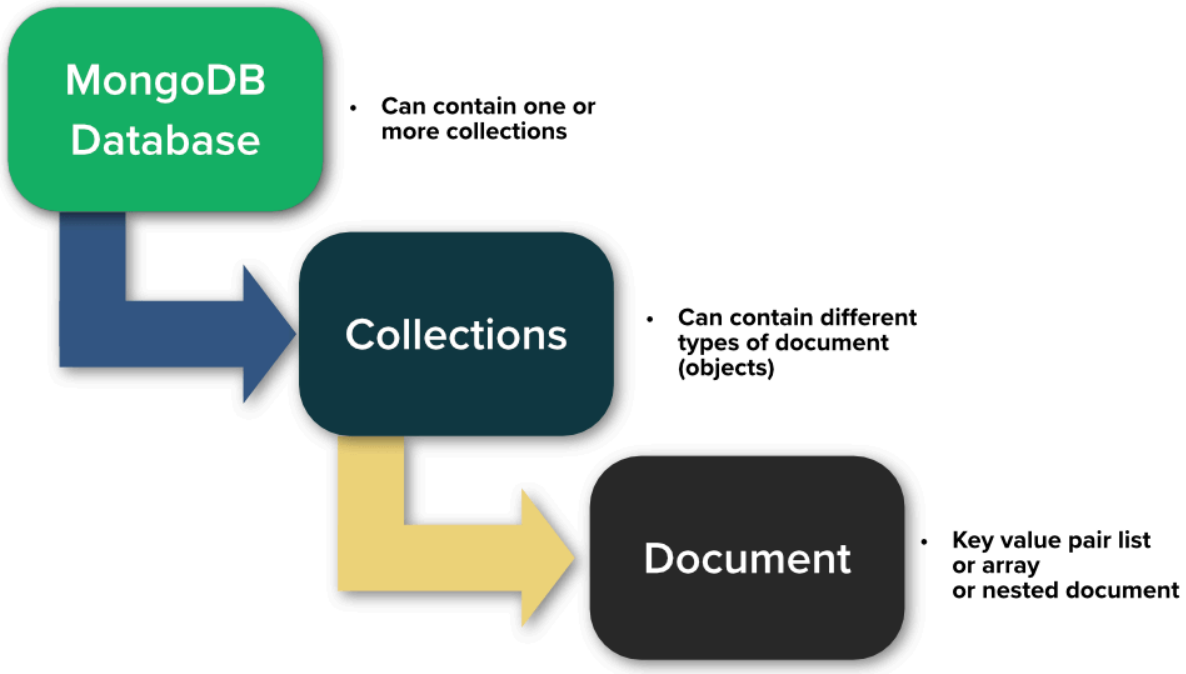


// Connection with VS Code

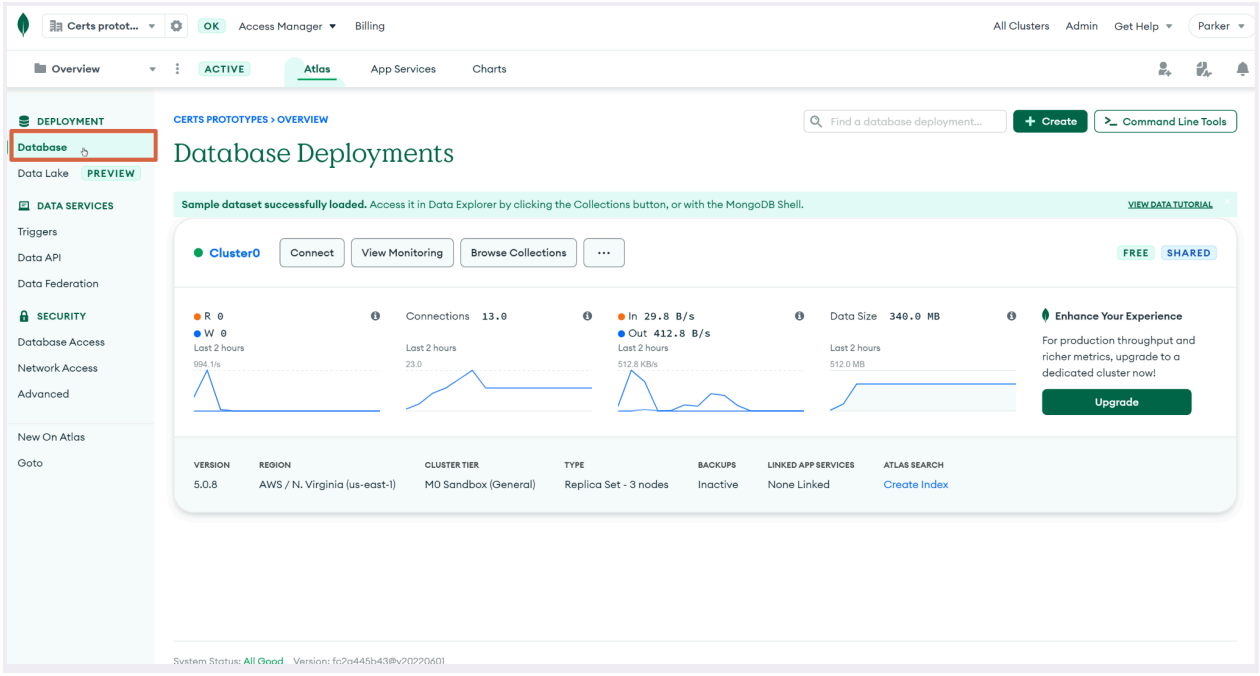
$ mongodb+srv://myAtlasDBUser:<password>@myatlasclusteredu.ymqgri1.mongodb.net/

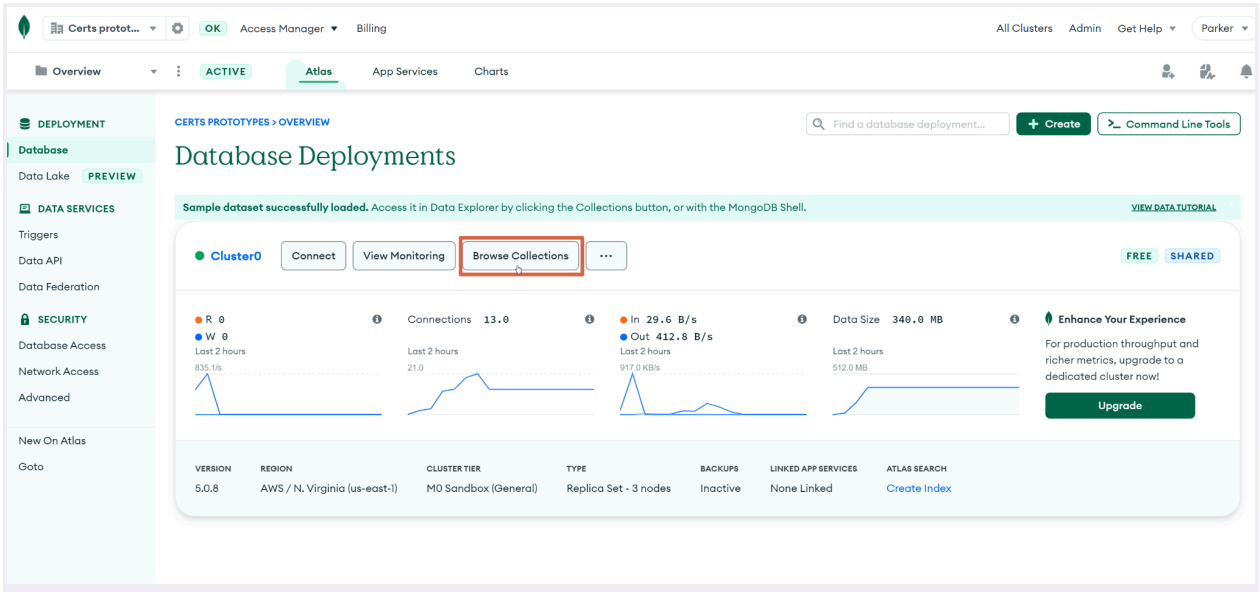
$ mongodb+srv://myAtlasDBUser:myatlas-001@myatlasclusteredu.ymqgri1.mongodb.net/

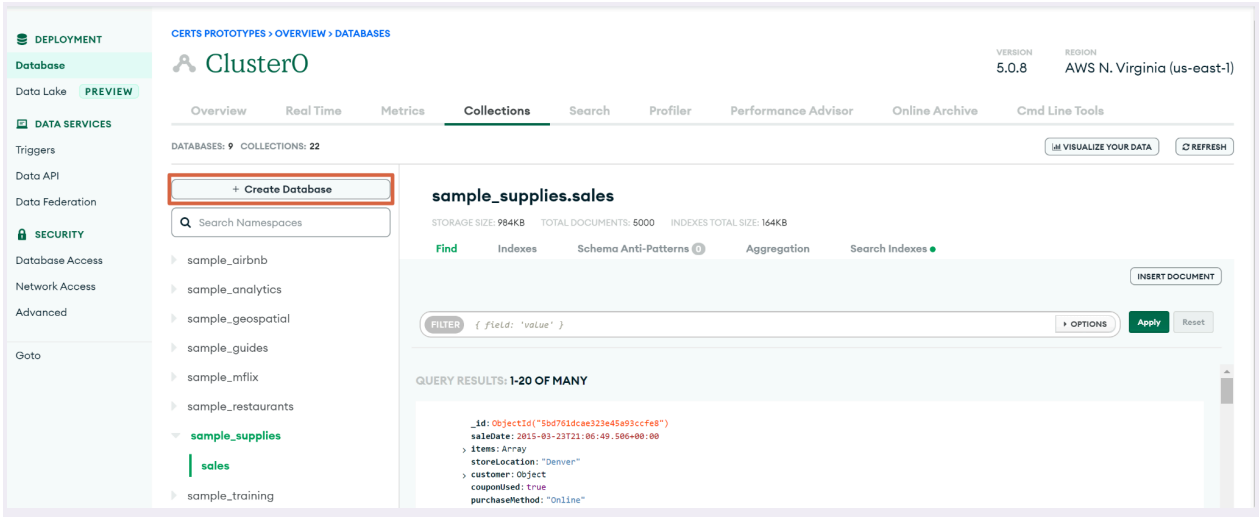
Document: Basic unit of data in MongoDB (Displayed **JSON but stored in BSON**). **Flexible schema** model. Supports Polymorphic data.
Collection: Grouping of documents
Database: Container for collection
Atlas: Developer data platform. The MongoDB database is the foundation of MongoDB Atlas. The additional functionality that Atlas offers—such as full-text search, data visualization, data lake storage, and mobile device sync—are built on top of data stored in cloud-hosted MongoDB database deployments.

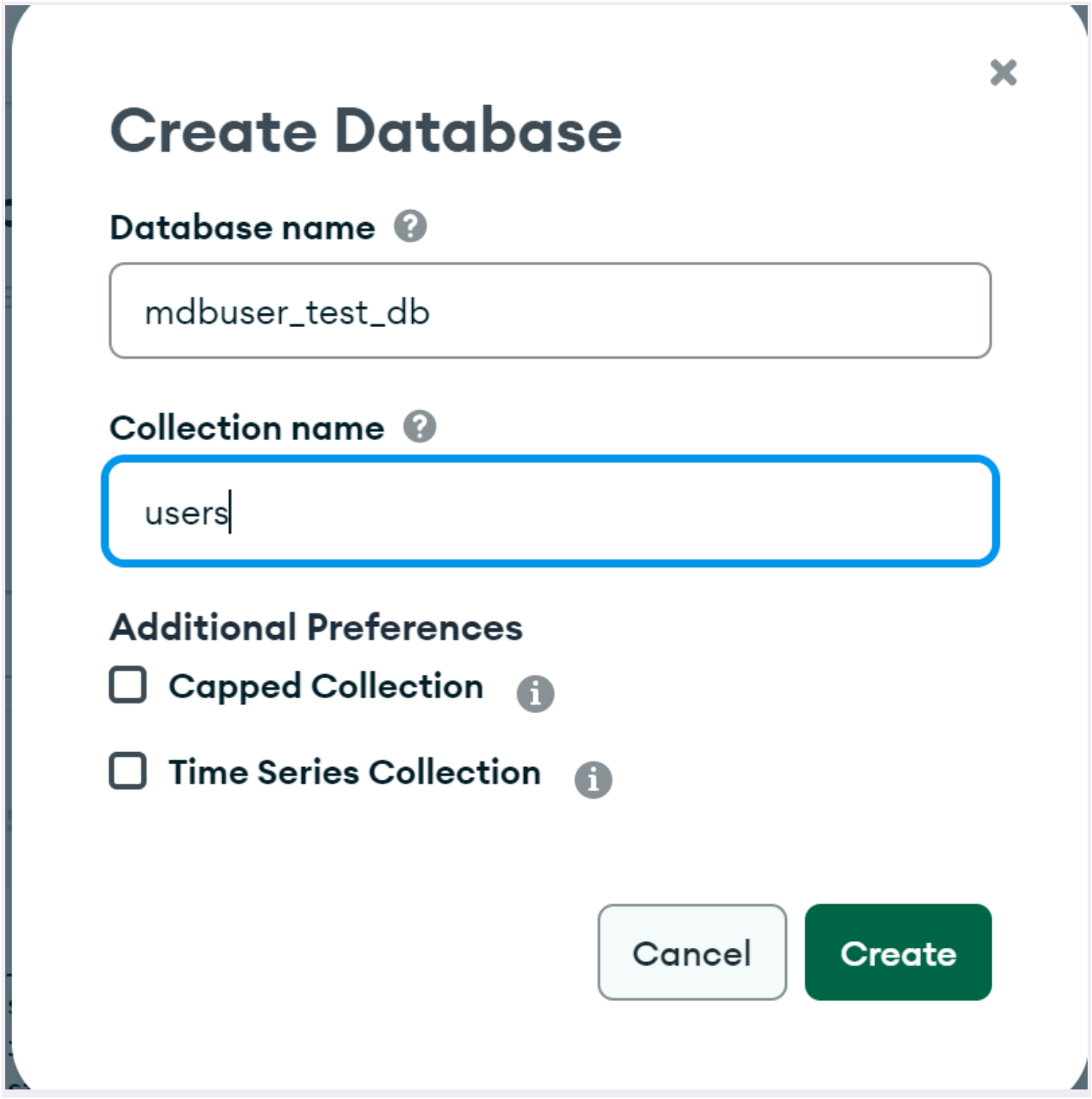


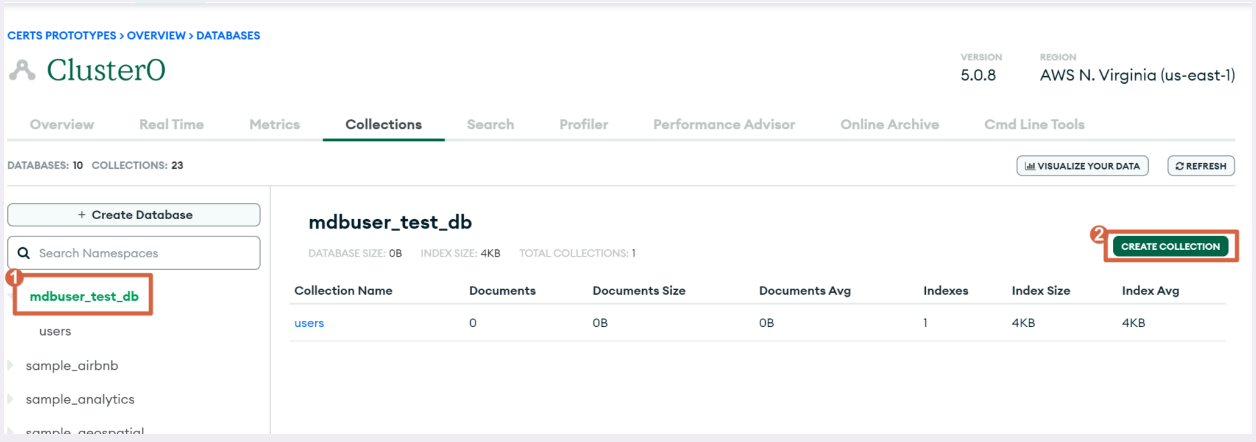
`_id` is a required field in every MongoDB document. If an inserted document does not include an `_id` field, MongoDB will automatically create the `_id` field and populate it with a value of type `ObjectId`

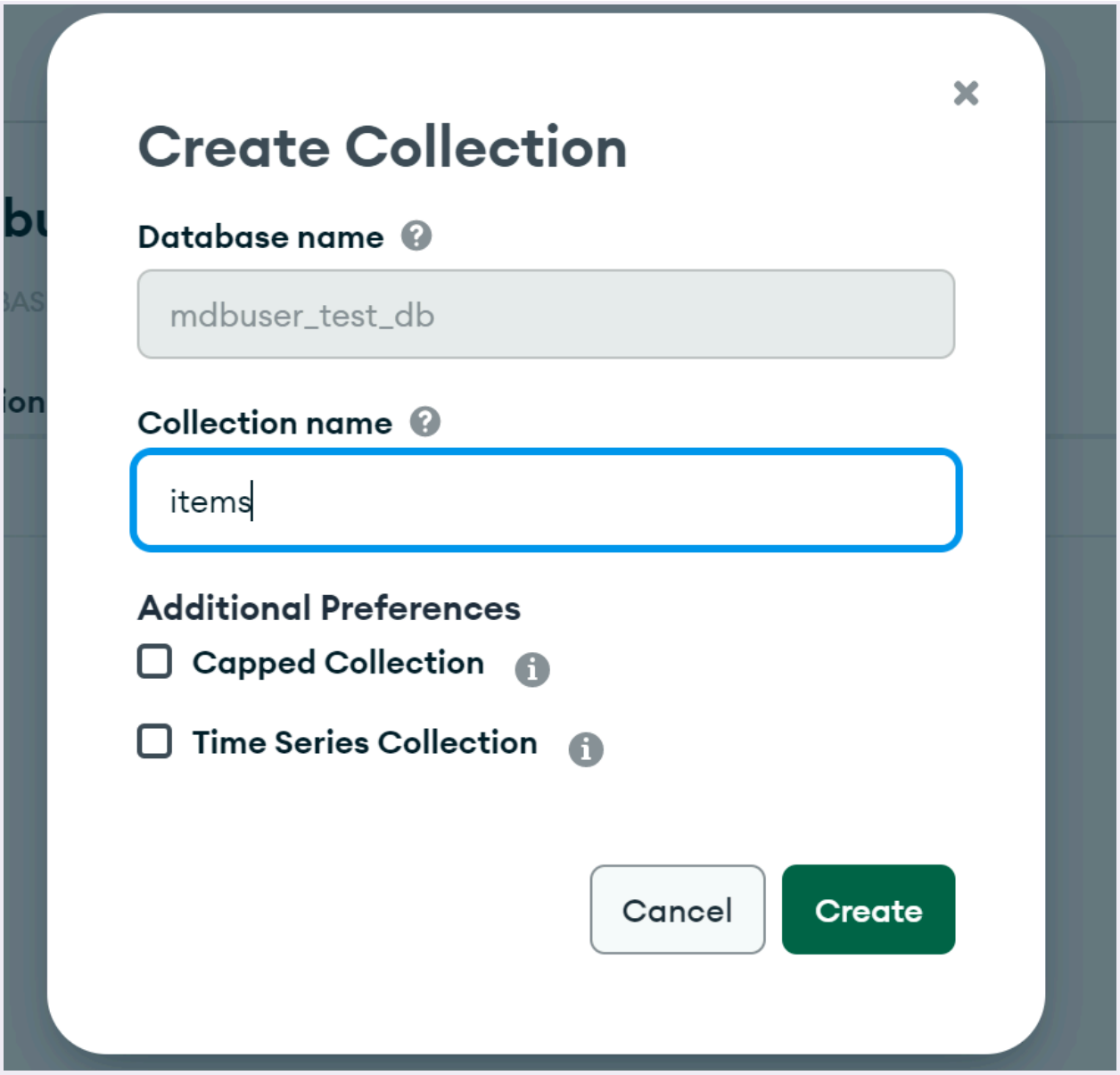

Certs protot...
OK
Access Manager
Billing
All Clusters
Admin
Get Help
Parker
Overview
ACTIVE
Atlas
App Services
Charts
DEPLOYMENT
Database
Data Lake
PREVIEW
DATA SERVICES
Triggers
Data API
Data Federation
SECURITY
Database Access
Network Access
Advanced
New On Atlas
Goto
CERTS PROTOTYPES > OVERVIEW
Find a database deployment...
Create
Command Line Tools
Database Deployments
Sample dataset successfully loaded. Access it in Data Explorer by clicking the Collections button, or with the MongoDB Shell.
VIEW DATA TUTORIAL
Cluster0
Connect
View Monitoring
Browse Collections
FREE
SHARED
R 0
W 0
Last 2 hours
Connections 13.0
In 29.8 B/s
Out 412.8 B/s
Data Size 340.0 MB
Enhance Your Experience
For production throughput and richer metrics, upgrade to a dedicated cluster now!
Upgrade
VERSION
REGION
CLUSTER TIER
TYPE
BACKUPS
LINKED APP SERVICES
ATLAS SEARCH
5.0.8
AWS / N. Virginia (us-east-1)
M0 Sandbox (General)
Replica Set - 3 nodes
Inactive
None Linked
Create Index

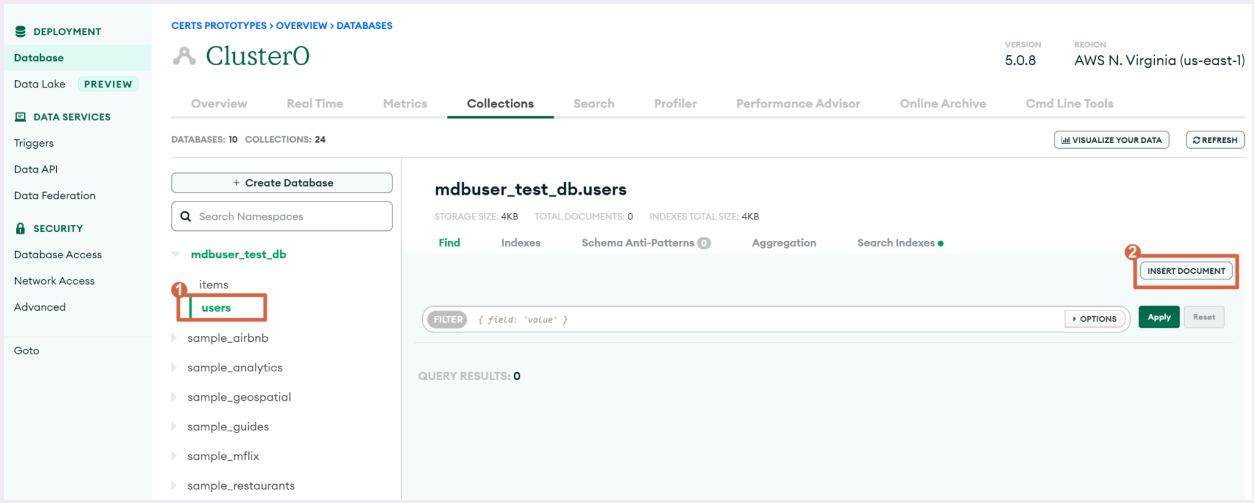

Certs protot...
OK
Access Manager
Billing
All Clusters
Admin
Get Help
Parker
Overview
ACTIVE
Atlas
App Services
Charts
DEPLOYMENT
Database
Data Lake
PREVIEW
DATA SERVICES
Triggers
Data API
Data Federation
SECURITY
Database Access
Network Access
Advanced
New On Atlas
Goto
CERTS PROTOTYPES > OVERVIEW
Find a database deployment...
Create
Command Line Tools
Database Deployments
Sample dataset successfully loaded. Access it in Data Explorer by clicking the Collections button, or with the MongoDB Shell.
VIEW DATA TUTORIAL
Cluster0
Connect
View Monitoring
Browse Collections
FREE
SHARED
R 0
W 0
Last 2 hours
Connections 13.0
In 29.6 B/s
Out 412.8 B/s
Data Size 340.0 MB
Enhance Your Experience
For production throughput and richer metrics, upgrade to a dedicated cluster now!
Upgrade
VERSION
REGION
CLUSTER TIER
TYPE
BACKUPS
LINKED APP SERVICES
ATLAS SEARCH
5.0.8
AWS / N. Virginia (us-east-1)
M0 Sandbox (General)
Replica Set - 3 nodes
Inactive
None Linked
Create Index


DEPLOYMENT
Database
Data Lake
PREVIEW
DATA SERVICES
Triggers
Data API
Data Federation
SECURITY
Database Access
Network Access
Advanced
Goto
CERTS PROTOTYPES > OVERVIEW > DATABASES
Cluster0
VERSION
5.0.8
REGION
AWS N. Virginia (us-east-1)
Overview
Real Time
Metrics
Collections
Search
Profiler
Performance Advisor
Online Archive
Cmd Line Tools
DATABASES: 9 COLLECTIONS: 22
VISUALIZE YOUR DATA
REFRESH
+ Create Database
Search Namespaces
sample_airbnb
sample_analytics
sample_geospatial
sample_guides
sample_mflix
sample_restaurants
sample_supplies
sales
sample_training
sample_supplies.sales
STORAGE SIZE: 984KB
TOTAL DOCUMENTS: 5000
INDEXES TOTAL SIZE: 164KB
Find
Indexes
Schema Anti-Patterns 0
Aggregation
Search Indexes
INSERT DOCUMENT
FILTER
{ field: 'value' }
OPTIONS
Apply
Reset
QUERY RESULTS: 1-20 OF MANY
_id: ObjectId("5bd761dcae323e45a93ccfe8")
saleDate: 2015-03-23T21:06:49.506+00:00
items: Array
storeLocation: "Denver"
customer: Object
couponUsed: true
purchaseMethod: "Online"

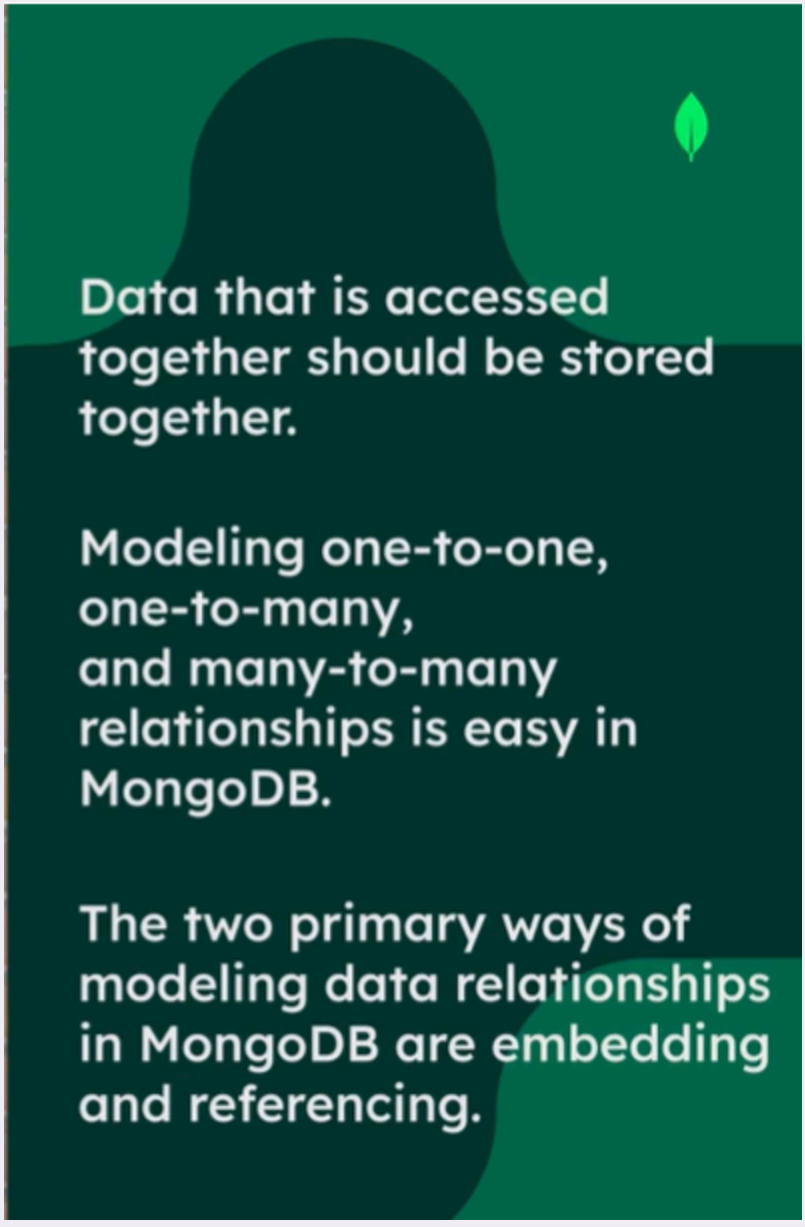

Create Database
Database name
mdbuser_test_db
Collection name
users
Additional Preferences
Capped Collection
Time Series Collection
Cancel
Create


CERTS PROTOTYPES > OVERVIEW > DATABASES
Cluster0
VERSION
5.0.8
REGION
AWS N. Virginia (us-east-1)
Overview
Real Time
Metrics
Collections
Search
Profiler
Performance Advisor
Online Archive
Cmd Line Tools
DATABASES: 10 COLLECTIONS: 23
VISUALIZE YOUR DATA
REFRESH
+ Create Database
Search Namespaces
1
mdbuser_test_db
users
sample_airbnb
sample_analytics
mdbuser_test_db
DATABASE SIZE: 0B INDEX SIZE: 4KB TOTAL COLLECTIONS: 1
2
CREATE COLLECTION
Collection Name
Documents
Documents Size
Documents Avg
Indexes
Index Size
Index Avg
users
0
0B
0B
1
4KB
4KB


Create Collection
Database name
mdbuser_test_db
Collection name
items
Additional Preferences
Capped Collection
Time Series Collection
Cancel
Create


DEPLOYMENT
Database
Data Lake PREVIEW
DATA SERVICES
Triggers
Data API
Data Federation
SECURITY
Database Access
Network Access
Advanced
Goto
CERTS PROTOTYPES > OVERVIEW > DATABASES
Cluster0
VERSION
5.0.8
REGION
AWS N. Virginia (us-east-1)
Overview
Real Time
Metrics
Collections
Search
Profiler
Performance Advisor
Online Archive
Cmd Line Tools
DATABASES: 10 COLLECTIONS: 24
VISUALIZE YOUR DATA
REFRESH
+ Create Database
Search Namespaces
mdbuser_test_db
items
users
sample_airbnb
sample_analytics
sample_geospatial
sample_guides
sample_mflix
sample_restaurants
mdbuser_test_db.users
STORAGE SIZE: 4KB
TOTAL DOCUMENTS: 0
INDEXES TOTAL SIZE: 4KB
Find
Indexes
Schema Anti-Patterns
Aggregation
Search Indexes
INSERT DOCUMENT
FILTER
{ field: 'value' }
OPTIONS
Apply
Reset
QUERY RESULTS: 0


Insert to Collection
VIEW
{}
1 _id : ObjectId("62a112c6e144e53fdb387223")
2 name : "Parker"
3 age : 28
ObjectId
String
Int32
Cancel
Insert



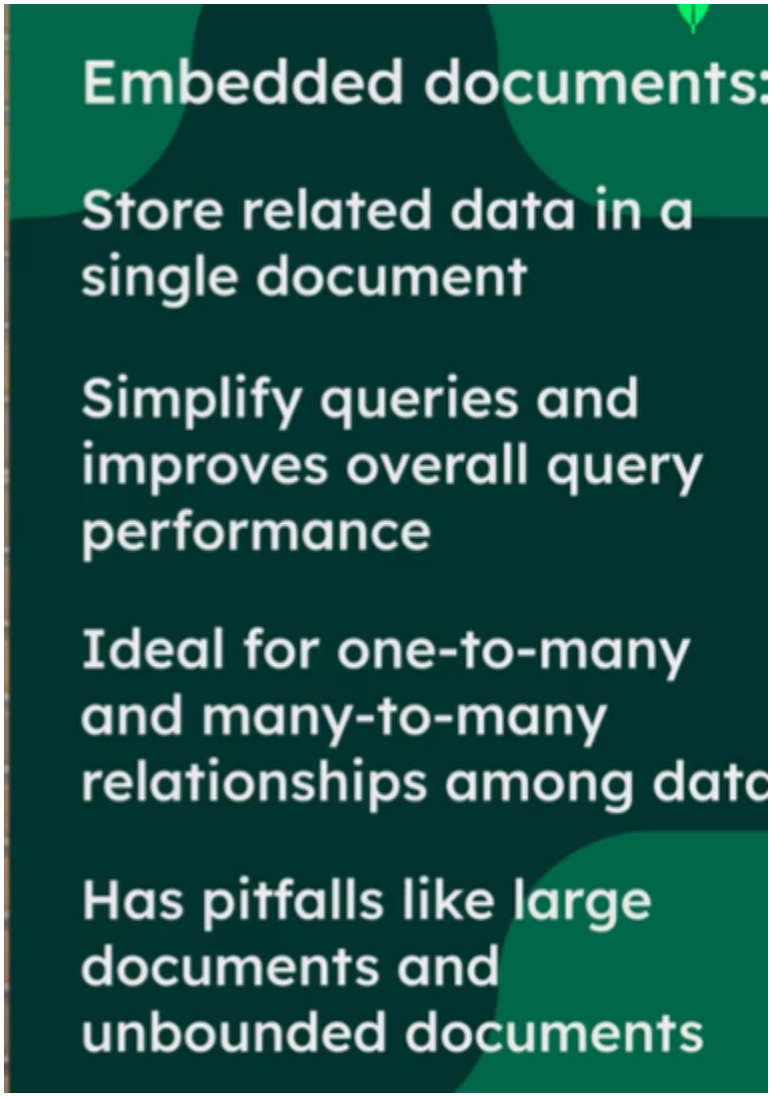
.

Modeling data relationship:

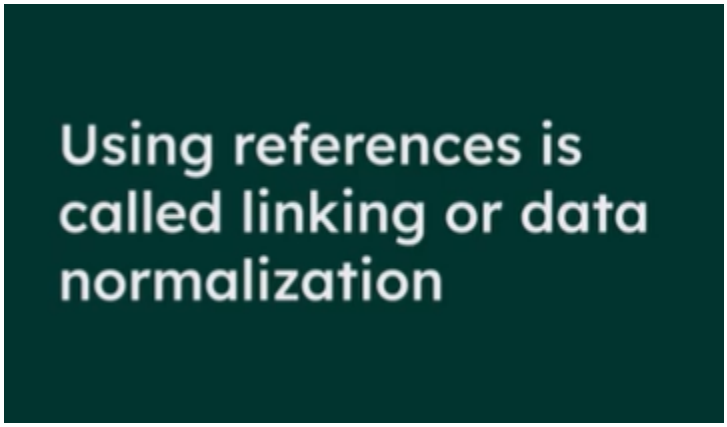
1. Embedded documents/nested documents.Done using array (unlike one-one relationship with key, value pair)

Limitation: - **Large documents** need to be loaded to memory

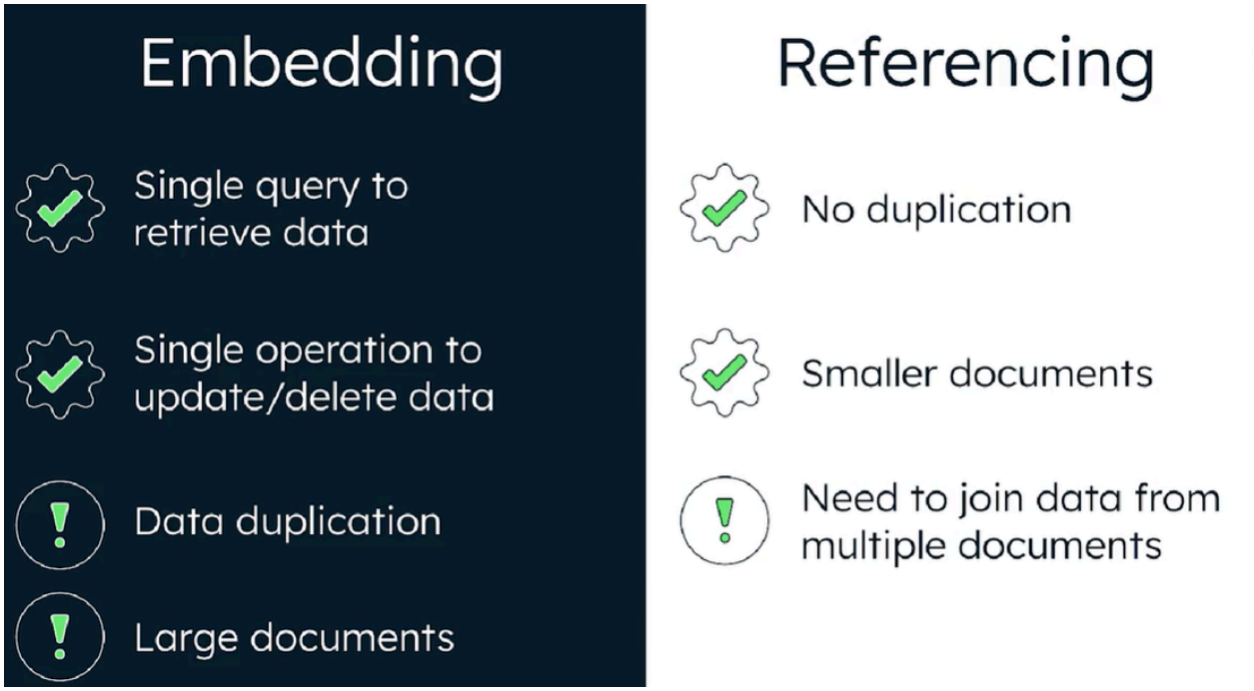
- **Unbounded documents** where BSON stored as larger than 16 MB



2. Referencing: Done in separate document and **referenced with Obect_id**



```
{
  "student": "John Smith",
   "student_id": "001",
   "age": "18",
   "email": "johnsmith@mongodb.edu",
   "grade_level": "freshman",
   "gpa": "4.0",
   "street": "3771 McClintock Ave",
   "city": "Los Angeles",
   "state": "CA",
   "zip": "90089",
   "emergency_contact_name": "Mary Smith",
   "emergency_contact_relation": "Mother",
“courses”:
{
 "course_id" : “CS150”,
 "course_name" : "MongoDB101."
}
{
 "course_id" : “CS177”,
 "course_name" : "Introduction to Programming in Python."
}
```

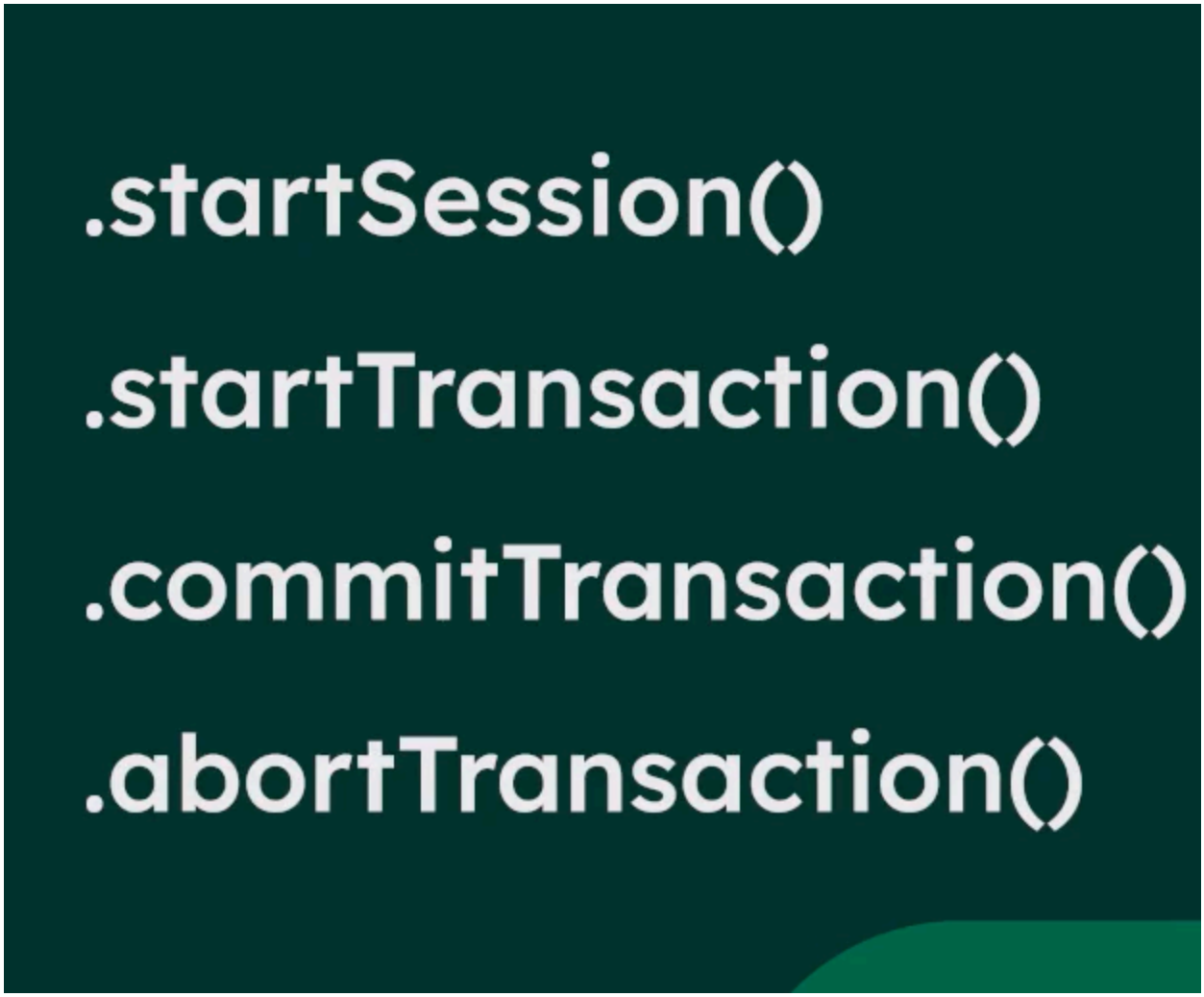

Embedding
Single query to retrieve data
Single operation to update/delete data
Data duplication
Large documents
Referencing
No duplication
Smaller documents
Need to join data from multiple documents

ACID Transaction

A group of database operations that will be completed as a unit or not at all

```
Atlas atlas-a847ea-shard-0 [primary] bank> const session = db.getMongo().startSession()

Atlas atlas-a847ea-shard-0 [primary] bank> session.startTransaction()

Atlas atlas-a847ea-shard-0 [primary] bank> const account = session.getDatabase('bank').getCollection('accounts')

Atlas atlas-a847ea-shard-0 [primary] bank> account.updateOne( { account_id: "MDB740836066"}, {$inc: { balance: -30 }})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Atlas atlas-a847ea-shard-0 [primary] bank> account.updateOne( { account_id: "MDB963134500"}, {$inc: { balance: 30 }})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Atlas atlas-a847ea-shard-0 [primary] bank> session.commitTransaction()
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1657293864, i: 7 }),
    signature: {
      hash: Binary(Buffer.from("5d29f78e6b41ec7233df9db846eda5f8000968c8", "hex"), 0),
      keyId: Long("7081342091082596360")
    }
```

```
Atlas atlas-a847ea-shard-0 [primary] bank> db.accounts.find( { account_id: "MDB740836066" } )
[
  {
    _id: ObjectId("62c843bdcd93ba83edcc7614"),
    account_id: 'MDB740836066',
    account_holder: 'Naja Petersen',
    account_type: 'savings',
    balance: 2861.8,
    transfers_complete: [ 'TR784553031', 'TR728134708', 'TR396066257' ]
  }
]
Atlas atlas-a847ea-shard-0 [primary] bank> db.accounts.find( { account_id: "MDB963134500" } )
[
  {
    _id: ObjectId("62c843bdcd93ba83edcc7603"),
    account_id: 'MDB963134500',
    account_holder: 'Florence Taylor',
    account_type: 'checking',
    balance: 2628.57,
    transfers_complete: [
      'TR652631048',
      'TR846259280',
      'TR412957052',
      'TR519499915',
      'TR767971921',
      'TR717427175',
      'TR263422717'
    ]
  }
]
Atlas atlas-a847ea-shard-0 [primary] bank>
```



######################

Multi-Document Transactions

ACID transactions in MongoDB are typically used only by applications where values are exchanged between different parties, such as banking or business applications. If you find yourself in a scenario where a multi-document transaction is required, it's very likely that you will complete a transaction with one of MongoDB's drivers. For now, let's focus on completing and canceling multi-document transactions in the shell to become familiar with the steps.

Using a Transaction: Video Code

Here is a recap of the code that's used to complete a multi-document transaction:

```
const session = db.getMongo().startSession()

session.startTransaction()

const account = session.getDatabase('< add database name here>').getCollection('<add collection name here>')

//Add database operations like .updateOne() here

session.commitTransaction()
```

Aborting a Transaction

If you find yourself in a scenario that requires you to roll back database operations before a transaction is completed, you can abort the transaction. Doing so will roll back the database to its original state, before the transaction was initiated.

Aborting a Transaction: Video Code

Here is a recap of the code that's used to cancel a transaction before it completes:

```
const session = db.getMongo().startSession()

session.startTransaction()

const account = session.getDatabase('< add database name here>').getCollection('<add collection name here>')

//Add database operations like .updateOne() here

session.abortTransaction()
```

```
##########################
Current Mongosh Log ID: 6661a3cd0435dc9905a26a12
Connecting to:
mongodb+srv://<credentials>@instruqttest.3xfvk.mongodb.net/bank?appName=mongos
h+2.2.6
Using MongoDB:          6.0.15
Using Mongosh:          2.2.6

For mongosh info see: https://docs.mongodb.com/mongodb-shell/


To help improve our products, anonymous usage data is collected and sent to MongoDB
periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

Atlas atlas-d3opcw-shard-0 [primary] bank> const session =
db.getMongo().startSession()

Atlas atlas-d3opcw-shard-0 [primary] bank> session.startTransaction()

Atlas atlas-d3opcw-shard-0 [primary] bank> const account =
session.getDatabase('bank').getCollection('accounts')

Atlas atlas-d3opcw-shard-0 [primary] bank> const account =
session.getDatabase('bank').getCollection('accounts')

Atlas atlas-d3opcw-shard-0 [primary] bank> account.insertOne({
...   account_id: "MDB454252264",
...   account_holder: "Florence Taylor",
...   account_type: "savings",
...   balance: 100.0,
...   transfers_complete: [],
...   last_updated: new Date()
... })
{
  acknowledged: true,
  insertedId: ObjectId('6661a3e50435dc9905a26a13')
}
Atlas atlas-d3opcw-shard-0 [primary] bank> account.updateOne( { account_id:
"MDB963134500" }, {$inc: { balance: -100.00 }})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
```

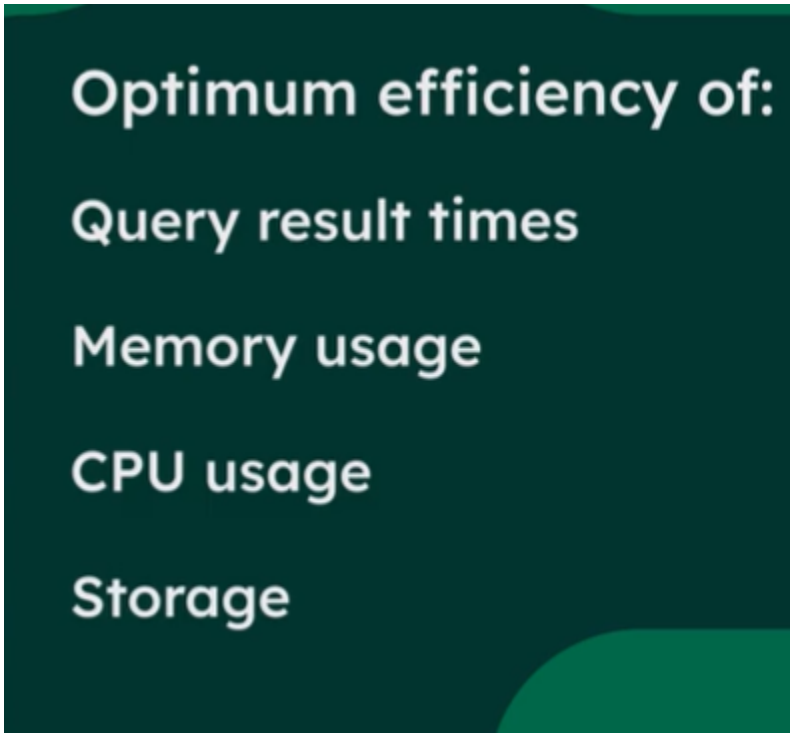
```
  modifiedCount: 1,
  upsertedCount: 0
}
Atlas atlas-d3opcw-shard-0 [primary] bank> session.commitTransaction()

Atlas atlas-d3opcw-shard-0 [primary] bank>
```

##

Create abort.json:

```
var session = db.getMongo().startSession()

session.startTransaction()

var account = session.getDatabase('bank').getCollection('accounts')

account.updateOne( { account_id: "MDB740836066" }, {$inc: { balance: 100 }})

account.updateOne( { account_id: "MDB963134500" }, {$inc: { balance: -5 }})

session.abortTransaction()


##############################################
```

Scaling data model:



Schema Design anti-patterns

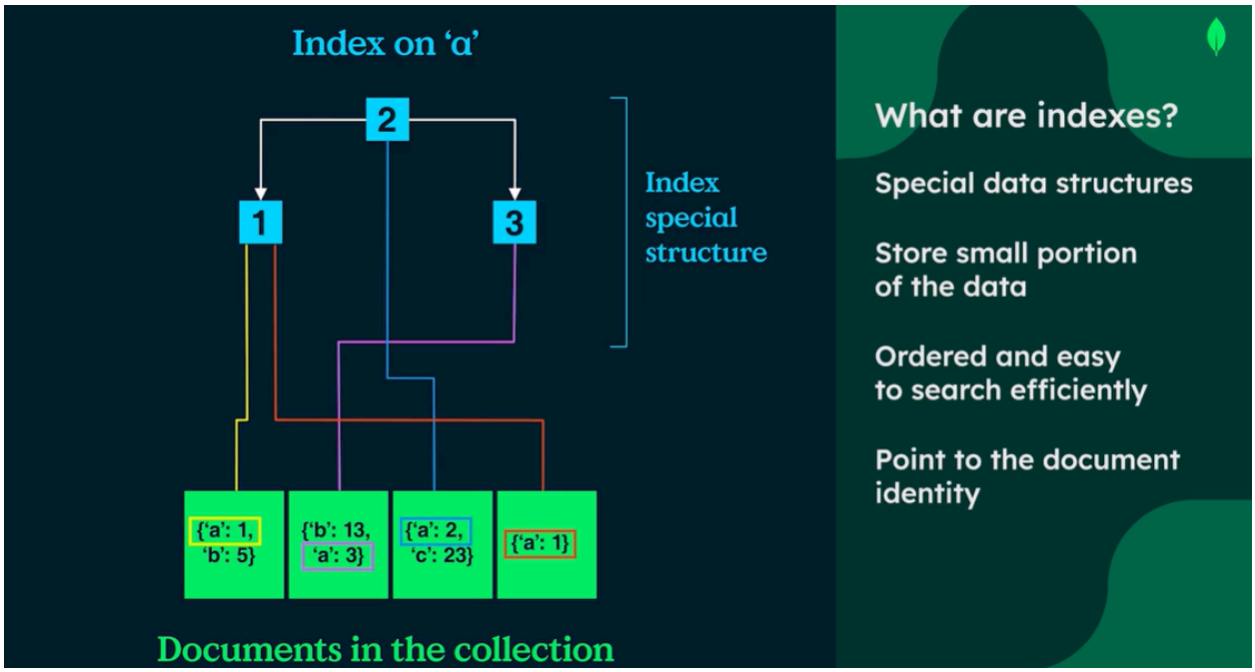
Connecting Python to MongoDB Database Drivers

- Drivers/Libraries: PyMongo (Synchronous) & Motor(Asynchronous)

```
from pymongo import MongoClient
# Set the value of MONGODB_URI to your Atlas connection string.
MONGODB_URI =
'mongodb+srv://myAtlasDBUser:myatlas-001@myatlasclusteredu.ymqgri1.mongodb.net/?retryWrites=
true&w=majority&appName=myAtlasClusterEDU'

# Connect to your MongoDB cluster.
        client = MongoClient(MONGODB_URI)

# List all databases in the cluster.
        for db_name in client.list_database_names():
           print(db_name)
```

```
from pymongo import MongoClient

MONGODB_URI = "mongodb+srv://user1:<password

client = MongoClient(MONGODB_URI)

for db_name in client.list_database_names():
    print(db_name)
```

Indexing: Done on collection. To improve query performance, reduce disk I/O, reduce resource required
Limitation: Write performance is redueced



Indexes:

Data structures for fast data retrieval

Support equality matches and range-based query operations, and return sorted results

Come with a write-performance cost

Common types are single field and compound indexes

Multikey indexes operate on an array field

Creating a Single Field Index

Create a Single Field Index

Use

`createIndex()`

to create a new index in a collection. Within the parentheses of

`createIndex()`

, include an object that contains the field and sort order.

```
db.customers.createIndex({
  birthdate: 1
})
```

Create a Unique Single Field Index

Add

`{unique:true}`

as a second, optional, parameter in

`createIndex()`

to force uniqueness in the index field values. Once the unique index is created, any inserts or updates including duplicated values in the collection for the index field/s will fail.

```
db.customers.createIndex({
  email: 1
},
{
  unique:true
})
```

MongoDB only creates the unique index if there is no duplication in the field values for the index field/s.

Unique indexes ensure that indexed fields do not store duplicate values. In this example, MongoDB will return a duplicate key error if you attempt to insert a new document with an email that already exists in the collection, as the `Unique` constraint was set to `true`.

View the Indexes used in a Collection

Use

`getIndexes()`

to see all the indexes created in a collection.

```
db.customers.getIndexes()
```

Check if an index is being used on a query

Use

`explain()`

in a collection when running a query to see the Execution plan. This plan provides the details of the execution stages (IXSCAN , COLLSCAN, FETCH, SORT, etc.).

- **The `IXSCAN` stage indicates the query is using an index and what index is being selected.**
- **The `COLLSCAN` stage indicates a collection scan is performed, not using any indexes.**
- **The `FETCH` stage indicates documents are being read from the collection.**
- **The `SORT` stage indicates documents are being sorted in memory.**

```
db.customers.explain().find({
  birthdate: {
    $gt:ISODate("1995-08-01")
    }
  })
db.customers.explain().find({
  birthdate: {
    $gt:ISODate("1995-08-01")
    }
  }).sort({
    email:1
    })
```

$ db.<collection>.getIndexes()

$ Atlas atlas-d3opcw-shard-0 [primary] bank> db.transfers.getIndexes()
⇒ [{ v: 2, key: { _id: 1 }, name: '_id_' }]

Understanding Multikey Indexes

Create a Single field Multikey Index

Use

`createIndex()`

to create a new index in a collection. Include an object as parameter that contains the array field and sort order. In this example `accounts` **is an array field.**

```
db.customers.createIndex({
  accounts: 1
})
```

View the Indexes used in a Collection

Use `getIndexes()` to see all the indexes created in a collection.

```
db.customers.getIndexes()
```

Check if an index is being used on a query

Use `explain()` in a collection when running a query to see the Execution plan. This plan provides the details of the execution stages (IXSCAN , COLLSCAN, FETCH, SORT, etc.).

- The `IXSCAN` stage indicates the query is using an index and what index is being selected.
- The `COLLSCAN` stage indicates a collection scan is perform, not using any indexes.
- The `FETCH` stage indicates documents are being read from the collection.
- The `SORT` stage indicates documents are being sorted in memory.

```
db.customers.explain().find({
  accounts: 627788
  })
```

###

```
db.customers.createIndex({accounts:1})

db.customers.createIndex({email:1,
    accounts:1})
```

####

$ Atlas atlas-d3opcw-shard-0 [primary] bank> db.accounts.createIndex({ account_id: 1 }, { unique: true })
⇒ account_id_1
####

```
Current Mongosh Log ID: 666052b3fc1dc1dc7ea26a12
Connecting to:
mongodb+srv://<credentials>@instruqttest.3xfvk.mongodb.net/bank?appName=mongosh+2.2.6
Using MongoDB:          6.0.15
Using Mongosh:          2.2.6

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

$ Atlas atlas-d3opcw-shard-0 [primary] bank> db.accounts.explain().find({ account_id:
"MDB829000996" })
⇒ {
  explainVersion: '1',
  queryPlanner: {
    namespace: 'bank.accounts',
    indexFilterSet: false,
    parsedQuery: { account_id: { '$eq': 'MDB829000996' } },
    queryHash: 'C7AD3977',
    planCacheKey: 'E975A254',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { account_id: 1 },
        indexName: 'account_id_1',
        isMultiKey: false,
        multiKeyPaths: { account_id: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { account_id: [ '["MDB829000996", "MDB829000996"]' ] }
      }
    },
    rejectedPlans: []
  },
  command: {
    find: 'accounts',
    filter: { account_id: 'MDB829000996' },
    '$db': 'bank'
  },
```

```
  serverInfo: {
    host: 'atlas-d3opcw-shard-00-01.3xfvk.mongodb.net',
    port: 27017,
    version: '6.0.15',
    gitVersion: '7494119c41ca4e13b493e9f048df4032164e860e'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1717588665, i: 2 }),
    signature: {
      hash: Binary.createFromBase64('YzGGeIB9HI0jjZ+TSEdMO5l6CXk=', 0),
      keyId: Long('7322486994171330561')
    }
  },
  operationTime: Timestamp({ t: 1717588665, i: 2 })
}
Atlas atlas-d3opcw-shard-0 [primary] bank>
####
```

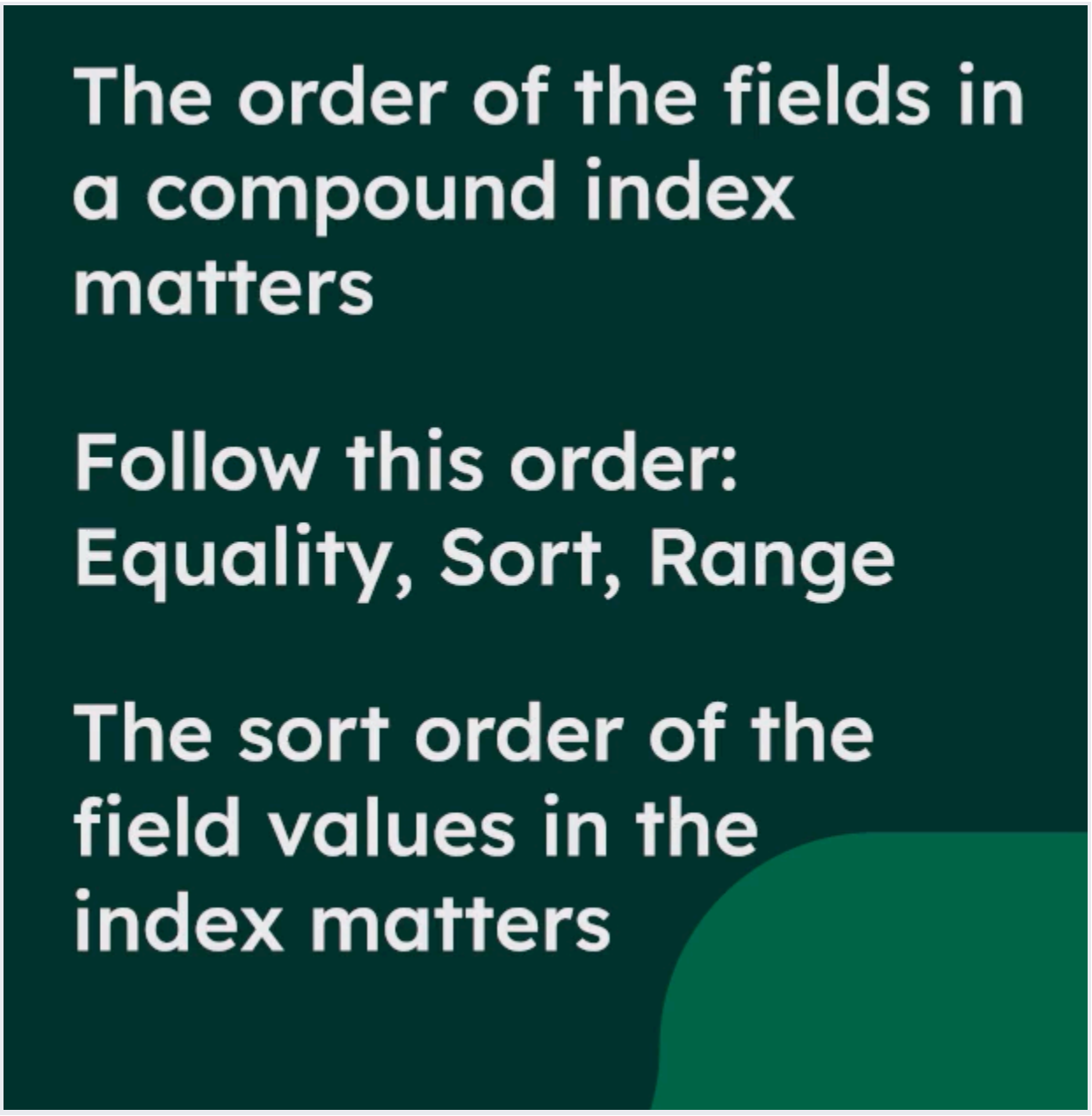
```
Atlas atlas-d3opcw-shard-0 [primary] bank> db.accounts.explain().find({
transfers_complete: { $in: ["TR617907396"] } })
⇒ {
  explainVersion: '1',
  queryPlanner: {
    namespace: 'bank.accounts',
    indexFilterSet: false,
    parsedQuery: { transfers_complete: { '$eq': 'TR617907396' } },
    queryHash: 'D341FD99',
    planCacheKey: '061129EA',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { transfers_complete: 1 },
        indexName: 'transfers_complete_1',
        isMultiKey: true,
        multiKeyPaths: { transfers_complete: [ 'transfers_complete' ] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { transfers_complete: [ '["TR617907396", "TR617907396"]' ] }
      }
    },
    rejectedPlans: []
  },
  command: {
    find: 'accounts',
    filter: { transfers_complete: { '$in': [ 'TR617907396' ] } },
    '$db': 'bank'
  },
  serverInfo: {
    host: 'atlas-d3opcw-shard-00-01.3xfvk.mongodb.net',
    port: 27017,
    version: '6.0.15',
    gitVersion: '7494119c41ca4e13b493e9f048df4032164e860e'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
```

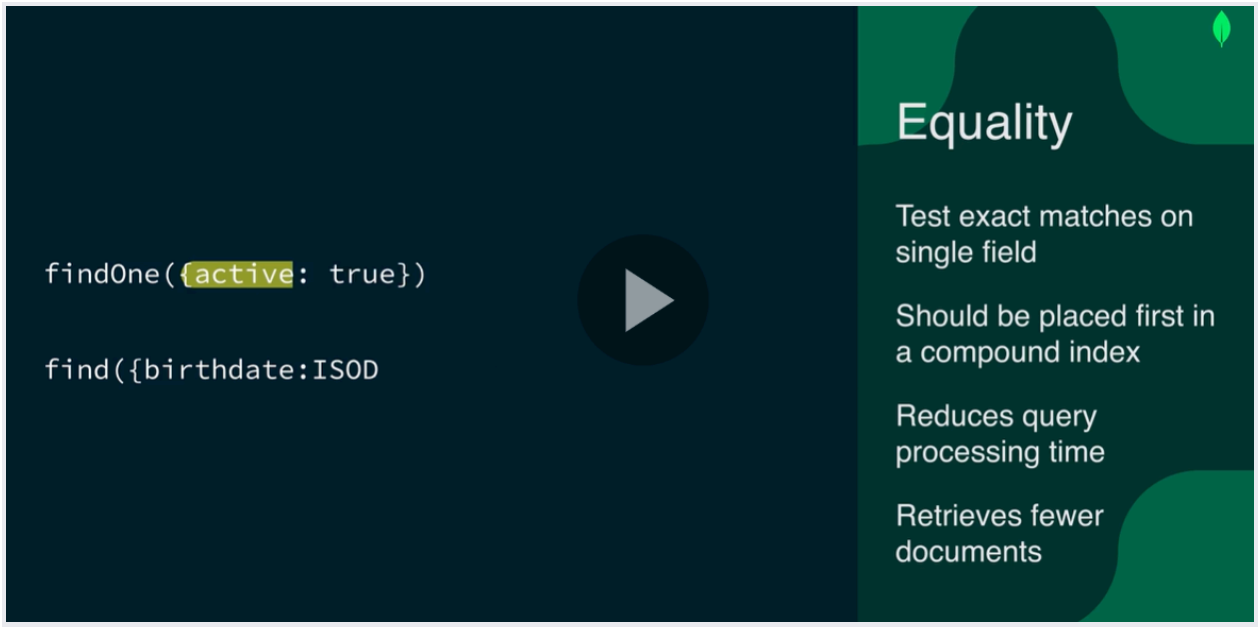
```
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1717589028, i: 7 }),
    signature: {
      hash: Binary.createFromBase64('hP5l64OOCkBmwJrv9tGWBHnclXI=', 0),
      keyId: Long('7322486994171330561')
    }
  },
  operationTime: Timestamp({ t: 1717589028, i: 7 })
}
Atlas atlas-d3opcw-shard-0 [primary] bank> // create a multikey index on the
`transfers_complete` field:

Atlas atlas-d3opcw-shard-0 [primary] bank> db.accounts.createIndex({
transfers_complete: 1 })
transfers_complete_1
Atlas atlas-d3opcw-shard-0 [primary] bank>

query that finds a specific `completed_transfers` array elementollowed by the find
method to view the winningPlan for a

Atlas atlas-d3opcw-shard-0 [primary] bank> db.accounts.explain().find({
transfers_complete: { $in: ["TR617907396"] } })
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'bank.accounts',
    indexFilterSet: false,
    parsedQuery: { transfers_complete: { '$eq': 'TR617907396' } },
    queryHash: 'D341FD99',
    planCacheKey: '061129EA',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
```

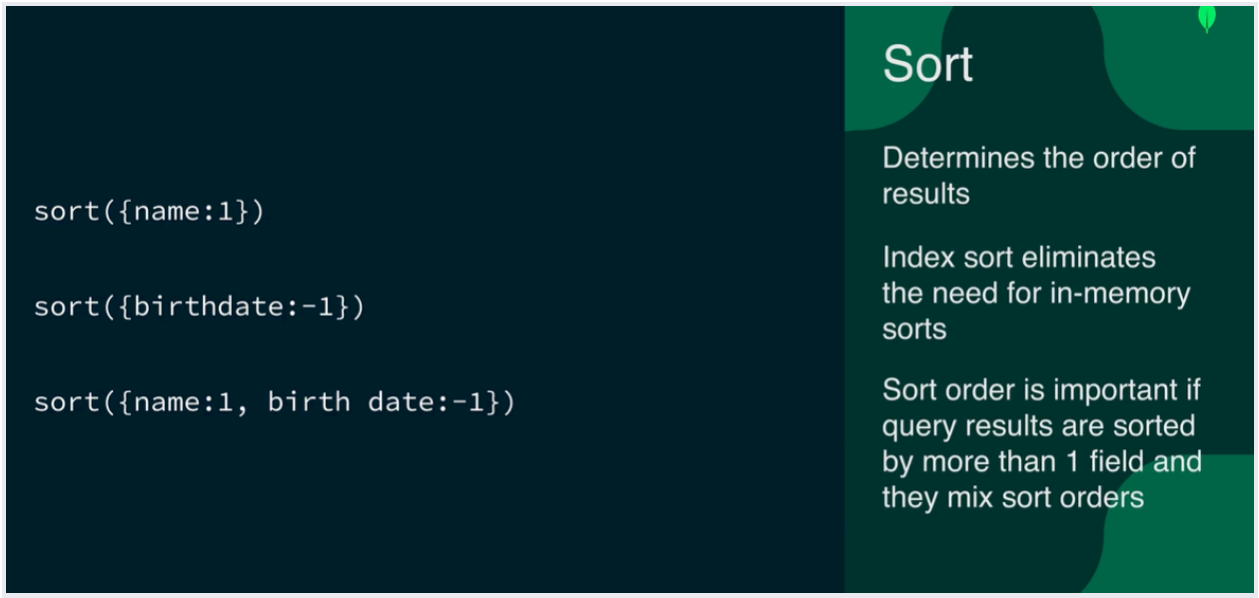
```
    inputStage: {
      stage: 'IXSCAN',
      keyPattern: { transfers_complete: 1 },
      indexName: 'transfers_complete_1',
      isMultiKey: true,
      multiKeyPaths: { transfers_complete: [ 'transfers_complete' ] },
      isUnique: false,
      isSparse: false,
      isPartial: false,
      indexVersion: 2,
      direction: 'forward',
      indexBounds: { transfers_complete: [ '["TR617907396", "TR617907396"]' ] }
    }
  },
  rejectedPlans: []
},
command: {
  find: 'accounts',
  filter: { transfers_complete: { '$in': [ 'TR617907396' ] } },
  '$db': 'bank'
},
serverInfo: {
  host: 'atlas-d3opcw-shard-00-01.3xfvk.mongodb.net',
  port: 27017,
  version: '6.0.15',
  gitVersion: '7494119c41ca4e13b493e9f048df4032164e860e'
},
serverParameters: {
  internalQueryFacetBufferSizeBytes: 104857600,
  internalQueryFacetMaxOutputDocSizeBytes: 104857600,
  internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
  internalDocumentSourceGroupMaxMemoryBytes: 104857600,
  internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
  internalQueryProhibitBlockingMergeOnMongoS: 0,
  internalQueryMaxAddToSetBytes: 104857600,
  internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
},
ok: 1,
'$clusterTime': {
  clusterTime: Timestamp({ t: 1717589052, i: 1 }),
  signature: {
    hash: Binary.createFromBase64('z/73z7uytw648KVAO5jqmk/enMs=', 0),
    keyId: Long('7322486994171330561')
  }
```

```
  },
  operationTime: Timestamp({ t: 1717589052, i: 1 })
}
Atlas atlas-d3opcw-shard-0 [primary] bank>
```

A compound index is an index that contains references to multiple fields within a document. Compound indexes are created by adding a comma-separated list of fields and their corresponding sort order to the index definition.







Working with Compound Indexes

Review the code below, which demonstrates how to create a compound index in a collection.

Create a Compound Index

Use `createIndex()` to create a new index in a collection. Within the parentheses of `createIndex()`, include an object that contains two or more fields and their sort order.

```
db.customers.createIndex({
  active:1,
  birthdate:-1,
  name:1
})
```

Order of Fields in a Compound Index

The order of the fields matters when creating the index and the sort order. It is recommended to list the fields in the following order: Equality, Sort, and Range.

- **Equality: field/s that matches on a single field value in a query**
- **Sort: field/s that orders the results by in a query**
- **Range: field/s that the query filter in a range of valid values**

The following query includes an equality match on the active field, a sort on birthday (descending) and name (ascending), and a range query on birthday too.

```
db.customers.find({
  birthdate: {
    $gte:ISODate("1977-01-01")
    },
    active:true
    }).sort({
      birthdate:-1,
      name:1
      })
```

Here's an example of an efficient index for this query:

```
db.customers.createIndex({
  active:1,
  birthdate:-1,
  name:1
})
```

View the Indexes used in a Collection

Use

`getIndexes()`

to see all the indexes created in a collection.

`db.customers.getIndexes()`

Check if an index is being used on a query

Use

`explain()`

in a collection when running a query to see the Execution plan. This plan provides the details of the execution stages (IXSCAN , COLLSCAN, FETCH, SORT, etc.). Some of these are:

- **The `IXSCAN` stage indicates the query is using an index and what index is being selected.**
- **The `COLLSCAN` stage indicates a collection scan is performed, not using any indexes.**
- **The `FETCH` stage indicates documents are being read from the collection.**
- **The `SORT` stage indicates documents are being sorted in memory.**

```
db.customers.explain().find({
  birthdate: {
    $gte:ISODate("1977-01-01")
    },
  active:true
  }).sort({
    birthdate:-1,
    name:1
    })
```

Cover a query by the Index

An Index covers a query when MongoDB does not need to fetch the data from memory since all the required data is already returned by the index.

In most cases, we can use projections to return only the required fields and cover the query. Make sure those fields in the projection are in the index.

By adding the projection

`{name:1,birthdate:1,_id:0}`

in the previous query, we can limit the returned fields to only

`name`

and

`birthdate`

. These fields are part of the index and when we run the `explain()` command, the execution plan shows only two stages:

- IXSCAN - Index scan using the compound index
- PROJECTION_COVERED - All the information needed is returned by the index, no need to fetch from memory

```
db.customers.explain().find({
  birthdate: {
    $gte:ISODate("1977-01-01")
    },
  active:true
  },
  {name:1,
    birthdate:1,
    _id:0
```

```
  }).sort({
     birthdate:-1,
     name:1
     })
```

####
db.accounts.explain().find({ account_holder:"Andrea", balance:{ $gt:5 }}, { account_holder: 1, balance: 1, account_type:1, _id: 0}).sort({ balance: 1 })r

⇒
Current Mongosh Log ID: 66605728bd03ba063aa26a12
Connecting to:
mongodb+srv://<credentials>@instruqttest.3xfvk.mongodb.net/bank?appName=mongosh+2.2.6
Using MongoDB: 6.0.15
Using Mongosh: 2.2.6

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

Atlas atlas-d3opcw-shard-0 [primary] bank> db.accounts.explain().find({ account_holder:"Andrea", balance:{ $gt:5 }}, { account_holder: 1, balance: 1, account_type:1, _id: 0}).sort({ balance: 1 })


```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'bank.accounts',
    indexFilterSet: false,
    parsedQuery: {
      '$and': [
        { account_holder: { '$eq': 'Andrea' } },
        { balance: { '$gt': 5 } }
      ]
    },
    queryHash: 'AD6630BF',
    planCacheKey: 'AD6630BF',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
```

```
    winningPlan: {
      stage: 'SORT',
      sortPattern: { balance: 1 },
      memLimit: 104857600,
      type: 'simple',
      inputStage: {
        stage: 'PROJECTION_SIMPLE',
        transformBy: { account_holder: 1, balance: 1, account_type: 1, _id: 0 },
        inputStage: {
          stage: 'COLLSCAN',
          filter: {
            '$and': [
              { account_holder: { '$eq': 'Andrea' } },
              { balance: { '$gt': 5 } }
            ]
          },
          direction: 'forward'
        }
      }
    },
    rejectedPlans: []
  },
  command: {
    find: 'accounts',
    filter: { account_holder: 'Andrea', balance: { '$gt': 5 } },
    sort: { balance: 1 },
    projection: { account_holder: 1, balance: 1, account_type: 1, _id: 0 },
    '$db': 'bank'
  },
  serverInfo: {
    host: 'atlas-d3opcw-shard-00-01.3xfvk.mongodb.net',
    port: 27017,
    version: '6.0.15',
    gitVersion: '7494119c41ca4e13b493e9f048df4032164e860e'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
```

```
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1717589830, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('6mZ/mb270MIMuxyW3YpsVwMTQMk=', 0),
      keyId: Long('7322486994171330561')
    }
  },
  operationTime: Timestamp({ t: 1717589830, i: 1 })
}
Atlas atlas-d3opcw-shard-0 [primary] bank>



####################################
// create a compound index using the `account_holder`, `balance` and `account_type`
fields:
db.accounts.createIndex({ account_holder: 1, balance: 1, account_type: 1 })

// Use the explain method to view the winning plan for a query
db.accounts.explain().find({ account_holder: "Andrea", balance:{ $gt :5 }}, {
account_holder: 1, balance: 1, account_type:1, _id: 0}).sort({ balance: 1 })



####################################################
```

The `hideIndex()` command hides an index. By hiding an index, you'll be able to assess the impact of removing the index on query performance. MongoDB does not use hidden indexes in queries but continues to update their keys. This allows you to assess if removing the index affects the query performance and unhide the index if needed. Unhiding an index is faster than recreating it. In this example, you would use the command `db.customers.hideIndex({email:1})`

Deleting an Index

Review the code below, which demonstrates how to delete indexes in a collection.

View the Indexes used in a Collection

Use

`getIndexes()`

to see all the indexes created in a collection. There is always a default index in every collection on _id field. This index is used by MongoDB internally and cannot be deleted.

```
db.customers.getIndexes()
```

Delete an Index

Use

`dropIndex()`

to delete an existing index from a collection. Within the parentheses of

`dropIndex()`**, include an object representing the index key or provide the index name as a string.**

Delete index by name:

```
db.customers.dropIndex(
  'active_1_birthdate_-1_name_1'
)
```

Delete index by key:

```
db.customers.dropIndex({
```

```
  active:1,
  birthdate:-1,
  name:1
})
```

Delete Indexes

Use

`dropIndexes()`

to delete all the indexes from a collection, with the exception of the default index on _id.

`db.customers.dropIndexes()`

The

`dropIndexes()`

command also can accept an array of index names as a parameter to delete a specific list of indexes.

```
db.collection.dropIndexes([
  'index1name', 'index2name', 'index3name'
  ])
```

Atlas Search (Relevance based search)... Apache Lucene

What type of index is written below? (Select one.)

```
{
  "analyzer": "lucene.standard",
  "searchAnalyzer": "lucene.standard",
  "mappings": {
    "dynamic": true
  }
}
```

Correct Answer

A. A search index

Your Answer: Correct ✔

Correct.

A search index is used to describe how the application search algorithm should work. You can customize this with Atlas Search.

-

```
{
  "analyzer": "lucene.standard",
  "searchAnalyzer": "lucene.standard",
  "mappings": {
    "dynamic": false,
    "fields": {
      "company": {
        "type": "string",
        "analyzer": "lucene.whitespace",
      },
      "employees": {
        "type": "string",
        "analyzer": "lucene.standard"
      }
    }
  }
}
```

- A search with a **dynamic index** will query against all of the fields, including nested fields. Dynamic field mapping is used to search all of the fields for the search term, with **equal weight** placed on all **fields.**
- **Static index**

Create a json file named "search_index.json"

```
{
  "name": "sample_supplies-sales-static",
  "searchAnalyzer": "lucene.standard",
  "analyzer": "lucene.standard",
  "collectionName": "sales",
  "database": "sample_supplies",
  "mappings": {
    "dynamic": false,
    "fields": {
      "storeLocation": {
        "type": "string"
      }
    }
  }
}
```

$ root@mongodb:~$ **atlas clusters search indexes create --clusterName myAtlasClusterEDU -f /app/search_index.json**
⇒ Index sample_supplies-sales-static created.
$ root@mongodb:~$ **atlas clusters search indexes list --clusterName myAtlasClusterEDU --db sample_supplies --collection sales**
⇒ #ID NAME DATABASE COLLECTION TYPE
666192a50239bd39492a2d7e sample_supplies-sales-static sample_supplies sales search
root@mongodb:~$ #
##

Using $search and Compound Operators

The compound operator within the $search aggregation stage allows us to give weight to different field and also filter our results without having to create additional aggregation stages. The four options for the compound operator are **"must", "mustNot, "should", and "filter".**

"must" will exclude records that do not meet the criteria. "mustNot" will exclude results that do meet the criteria. "should" will allow you to give weight to results that do meet the criteria so that they appear first. "filter" will remove results that do not meet the criteria.

```
$search {
  "compound": {
    "must": [{
      "text": {
        "query": "field",
        "path": "habitat"
      }
    }],
    "should": [{
      "range": {
        "gte": 45,
        "path": "wingspan_cm",
        "score": {"constant": {"value": 5}}
      }
    }]
  }
}
```

################################

1. Create a `$search` stage that uses the index `sample_supplies-sales-static` with query value of `London` and a path wildcard of `*`.
2. Create a `$set` stage that adds a new field `score` to the document to represent the Atlas Search score `{ $meta: "searchScore" }`.
3. Review the documents outputted and note they now include a score, the `object_id`, and the fields that matched the search term. The score is based on how much that record matches the search term and on the rules described in the index. In this case, with a static mapping for the index, the field `storeLocation` is what is used for the score calculation.

```
$ mongosh -u myAtlasDBUser -p myatlas-001 $MY_ATLAS_CONNECTION_STRING/sample_supplies

$ db.sales.aggregate([
 {
  $search: {
   index: 'sample_supplies-sales-static',
   text: {
    query: 'London', path: { 'wildcard': '*' }
   } } },
 {
  $set: {
   score: { $meta: "searchScore" }
   }
 }
])
```

#####In Terminal#####
root@mongodb:~$ mongosh -u myAtlasDBUser -p myatlas-001
$MY_ATLAS_CONNECTION_STRING/sample_supplies
Current Mongosh Log ID: 6661979a4f56e256c7a26a12
Connecting to:
mongodb+srv://<credentials>@myatlasclusteredu.ymqgri1.mongodb.net/sample_supplies?appName=mongosh+2.2.6
Using MongoDB: 7.0.11
Using Mongosh: 2.2.6

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

Atlas atlas-x02tt5-shard-0 [primary] sample_supplies> db.sales.aggregate([
... {

```
...     $search: {
...       index: 'sample_supplies-sales-static',
...       text: {
...         query: 'London', path: { 'wildcard': '*' }
...       } } },
...   {
...     $set: {
...       score: { $meta: "searchScore" }
...       }
...   }
... ])
[
```

```
⇒  {
  _id: ObjectId('5bd761dcae323e45a93ccff3'),
  saleDate: ISODate('2015-07-22T02:45:20.727Z'),
  items: [
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('21.46'),
      quantity: 5
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('21.82'),
      quantity: 1
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('34.43'),
      quantity: 3
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('24.11'),
      quantity: 1
    }
  ],
  storeLocation: 'London',
  customer: {
    gender: 'F',
    age: 49,
    email: 'merto@betosiv.pm',
```

```
    satisfaction: 3
  },
  couponUsed: false,
  purchaseMethod: 'In store',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd018'),
  saleDate: ISODate('2017-09-28T08:35:45.703Z'),
  items: [
    {
      name: 'laptop',
      tags: [ 'electronics', 'school', 'office' ],
      price: Decimal128('1531.69'),
      quantity: 5
    },
    {
      name: 'printer paper',
      tags: [ 'office', 'stationary' ],
      price: Decimal128('37'),
      quantity: 10
    },
    {
      name: 'binder',
      tags: [ 'school', 'general', 'organization' ],
      price: Decimal128('25.02'),
      quantity: 10
    },
    {
      name: 'binder',
      tags: [ 'school', 'general', 'organization' ],
      price: Decimal128('17.1'),
      quantity: 7
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('6.1'),
      quantity: 2
    },
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
      price: Decimal128('5.65'),
      quantity: 4
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
```

```
      price: Decimal128('10.6'),
      quantity: 5
    }
  ],
  storeLocation: 'London',
  customer: { gender: 'M', age: 71, email: 'ep@jedofi.gd', satisfaction: 5 },
  couponUsed: false,
  purchaseMethod: 'Online',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd021'),
  saleDate: ISODate('2016-02-06T09:35:02.820Z'),
  items: [
    {
      name: 'binder',
      tags: [ 'school', 'general', 'organization' ],
      price: Decimal128('23.14'),
      quantity: 10
    }
  ],
  storeLocation: 'London',
  customer: { gender: 'F', age: 53, email: 'we@bedijohaz.gp', satisfaction: 3 },
  couponUsed: false,
  purchaseMethod: 'In store',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd037'),
  saleDate: ISODate('2017-03-18T14:43:24.028Z'),
  items: [
    {
      name: 'backpack',
      tags: [ 'school', 'travel', 'kids' ],
      price: Decimal128('105.7'),
      quantity: 5
    },
    {
      name: 'laptop',
      tags: [ 'electronics', 'school', 'office' ],
      price: Decimal128('1350.8'),
      quantity: 3
    },
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
      price: Decimal128('40.79'),
      quantity: 2
    },
```

```
  {
    name: 'binder',
    tags: [ 'school', 'general', 'organization' ],
    price: Decimal128('23.44'),
    quantity: 8
  },
  {
    name: 'envelopes',
    tags: [ 'stationary', 'office', 'general' ],
    price: Decimal128('6.92'),
    quantity: 4
  },
  {
    name: 'notepad',
    tags: [ 'office', 'writing', 'school' ],
    price: Decimal128('27.6'),
    quantity: 3
  },
  {
    name: 'printer paper',
    tags: [ 'office', 'stationary' ],
    price: Decimal128('19.28'),
    quantity: 1
  },
  {
    name: 'notepad',
    tags: [ 'office', 'writing', 'school' ],
    price: Decimal128('22.31'),
    quantity: 3
  },
  {
    name: 'envelopes',
    tags: [ 'stationary', 'office', 'general' ],
    price: Decimal128('5.39'),
    quantity: 10
  },
  {
    name: 'pens',
    tags: [ 'writing', 'office', 'school', 'stationary' ],
    price: Decimal128('14.79'),
    quantity: 3
  }
],
storeLocation: 'London',
customer: {
  gender: 'F',
  age: 29,
  email: 'jokakre@pocuuwe.kw',
  satisfaction: 4
```

```
  },
  couponUsed: false,
  purchaseMethod: 'In store',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd03b'),
  saleDate: ISODate('2013-01-17T17:44:45.354Z'),
  items: [
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('15.02'),
      quantity: 1
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('24'),
      quantity: 4
    },
    {
      name: 'binder',
      tags: [ 'school', 'general', 'organization' ],
      price: Decimal128('10.74'),
      quantity: 5
    },
    {
      name: 'printer paper',
      tags: [ 'office', 'stationary' ],
      price: Decimal128('28.57'),
      quantity: 3
    },
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
      price: Decimal128('39.66'),
      quantity: 4
    },
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('5.59'),
      quantity: 3
    },
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('7.32'),
```

```
      quantity: 9
    },
    {
      name: 'backpack',
      tags: [ 'school', 'travel', 'kids' ],
      price: Decimal128('173.97'),
      quantity: 2
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('23.89'),
      quantity: 3
    }
  ],
  storeLocation: 'London',
  customer: { gender: 'M', age: 53, email: 'osi@wenowu.vu', satisfaction: 4 },
  couponUsed: true,
  purchaseMethod: 'In store',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd067'),
  saleDate: ISODate('2015-01-27T18:23:15.509Z'),
  items: [
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('29.73'),
      quantity: 3
    },
    {
      name: 'binder',
      tags: [ 'school', 'general', 'organization' ],
      price: Decimal128('20.76'),
      quantity: 3
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('12.77'),
      quantity: 5
    },
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
      price: Decimal128('23.55'),
      quantity: 2
    },
```

```
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('16.42'),
      quantity: 1
    },
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('15.05'),
      quantity: 8
    },
    {
      name: 'laptop',
      tags: [ 'electronics', 'school', 'office' ],
      price: Decimal128('864.31'),
      quantity: 4
    },
    {
      name: 'binder',
      tags: [ 'school', 'general', 'organization' ],
      price: Decimal128('12.18'),
      quantity: 8
    },
    {
      name: 'backpack',
      tags: [ 'school', 'travel', 'kids' ],
      price: Decimal128('86.32'),
      quantity: 1
    },
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
      price: Decimal128('52.55'),
      quantity: 2
    }
  ],
  storeLocation: 'London',
  customer: { gender: 'M', age: 69, email: 'lu@cougva.cy', satisfaction: 3 },
  couponUsed: false,
  purchaseMethod: 'Online',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd07a'),
  saleDate: ISODate('2015-02-15T17:41:53.206Z'),
  items: [
    {
      name: 'printer paper',
```

```
      tags: [ 'office', 'stationary' ],
      price: Decimal128('32.38'),
      quantity: 4
    },
    {
      name: 'laptop',
      tags: [ 'electronics', 'school', 'office' ],
      price: Decimal128('1282.77'),
      quantity: 3
    },
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('21.39'),
      quantity: 5
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('17.9'),
      quantity: 2
    },
    {
      name: 'backpack',
      tags: [ 'school', 'travel', 'kids' ],
      price: Decimal128('180.33'),
      quantity: 1
    }
  ],
  storeLocation: 'London',
  customer: {
    gender: 'M',
    age: 66,
    email: 'rewluclil@zufi.ca',
    satisfaction: 4
  },
  couponUsed: false,
  purchaseMethod: 'Online',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd07e'),
  saleDate: ISODate('2017-06-22T21:53:26.728Z'),
  items: [
    {
      name: 'backpack',
      tags: [ 'school', 'travel', 'kids' ],
      price: Decimal128('176.37'),
      quantity: 5
```

```
    },
    {
      name: 'printer paper',
      tags: [ 'office', 'stationary' ],
      price: Decimal128('30.27'),
      quantity: 8
    },
    {
      name: 'laptop',
      tags: [ 'electronics', 'school', 'office' ],
      price: Decimal128('696.85'),
      quantity: 3
    }
  ],
  storeLocation: 'London',
  customer: {
    gender: 'F',
    age: 56,
    email: 'galvuhamo@popudhuv.td',
    satisfaction: 5
  },
  couponUsed: false,
  purchaseMethod: 'Phone',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd084'),
  saleDate: ISODate('2013-11-08T15:01:32.331Z'),
  items: [
    {
      name: 'laptop',
      tags: [ 'electronics', 'school', 'office' ],
      price: Decimal128('709.07'),
      quantity: 1
    },
    {
      name: 'printer paper',
      tags: [ 'office', 'stationary' ],
      price: Decimal128('47.88'),
      quantity: 1
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('33.61'),
      quantity: 3
    }
  ],
  storeLocation: 'London',
```

```
  customer: { gender: 'F', age: 54, email: 'rulu@vaju.la', satisfaction: 2 },
  couponUsed: false,
  purchaseMethod: 'In store',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd088'),
  saleDate: ISODate('2015-10-20T16:40:43.714Z'),
  items: [
    {
      name: 'printer paper',
      tags: [ 'office', 'stationary' ],
      price: Decimal128('20.2'),
      quantity: 1
    },
    {
      name: 'binder',
      tags: [ 'school', 'general', 'organization' ],
      price: Decimal128('26.68'),
      quantity: 4
    },
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
      price: Decimal128('37.27'),
      quantity: 5
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('26.77'),
      quantity: 3
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('32.23'),
      quantity: 2
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('17.85'),
      quantity: 3
    },
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
      price: Decimal128('72.24'),
```

```
      quantity: 2
    },
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('14.2'),
      quantity: 2
    },
    {
      name: 'backpack',
      tags: [ 'school', 'travel', 'kids' ],
      price: Decimal128('170.72'),
      quantity: 5
    }
  ],
  storeLocation: 'London',
  customer: {
    gender: 'M',
    age: 44,
    email: 'movodfa@tuddat.ng',
    satisfaction: 4
  },
  couponUsed: false,
  purchaseMethod: 'In store',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd090'),
  saleDate: ISODate('2013-02-13T14:07:15.955Z'),
  items: [
    {
      name: 'printer paper',
      tags: [ 'office', 'stationary' ],
      price: Decimal128('12.04'),
      quantity: 3
    },
    {
      name: 'backpack',
      tags: [ 'school', 'travel', 'kids' ],
      price: Decimal128('140.41'),
      quantity: 4
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('34.41'),
      quantity: 3
    },
    {
```

```
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('26.28'),
      quantity: 4
    },
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('11.95'),
      quantity: 5
    },
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
      price: Decimal128('55.82'),
      quantity: 1
    },
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('14.25'),
      quantity: 3
    },
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
      price: Decimal128('50.21'),
      quantity: 5
    }
  ],
  storeLocation: 'London',
  customer: {
    gender: 'F',
    age: 26,
    email: 'uchocga@jucuv.mh',
    satisfaction: 4
  },
  couponUsed: false,
  purchaseMethod: 'In store',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd094'),
  saleDate: ISODate('2016-09-26T12:47:34.933Z'),
  items: [
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
      price: Decimal128('63.89'),
```

```
      quantity: 3
    },
    {
      name: 'binder',
      tags: [ 'school', 'general', 'organization' ],
      price: Decimal128('12.05'),
      quantity: 7
    }
  ],
  storeLocation: 'London',
  customer: {
    gender: 'M',
    age: 47,
    email: 'gijudow@turuwim.do',
    satisfaction: 4
  },
  couponUsed: false,
  purchaseMethod: 'Online',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd0be'),
  saleDate: ISODate('2016-08-15T15:41:39.260Z'),
  items: [
    {
      name: 'printer paper',
      tags: [ 'office', 'stationary' ],
      price: Decimal128('10.31'),
      quantity: 3
    },
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
      price: Decimal128('44.8'),
      quantity: 3
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('19.05'),
      quantity: 1
    },
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('18.59'),
      quantity: 4
    },
    {
```

```
      name: 'binder',
      tags: [ 'school', 'general', 'organization' ],
      price: Decimal128('24.2'),
      quantity: 1
    },
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('24.48'),
      quantity: 10
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('25.23'),
      quantity: 3
    },
    {
      name: 'backpack',
      tags: [ 'school', 'travel', 'kids' ],
      price: Decimal128('135.72'),
      quantity: 2
    }
  ],
  storeLocation: 'London',
  customer: { gender: 'M', age: 28, email: 'lo@bakeve.sh', satisfaction: 4 },
  couponUsed: false,
  purchaseMethod: 'Online',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd0c7'),
  saleDate: ISODate('2017-12-11T02:18:28.861Z'),
  items: [
    {
      name: 'backpack',
      tags: [ 'school', 'travel', 'kids' ],
      price: Decimal128('44.65'),
      quantity: 4
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('18.73'),
      quantity: 5
    },
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
```

```
      price: Decimal128('72.08'),
      quantity: 4
    },
    {
      name: 'binder',
      tags: [ 'school', 'general', 'organization' ],
      price: Decimal128('13.35'),
      quantity: 3
    },
    {
      name: 'laptop',
      tags: [ 'electronics', 'school', 'office' ],
      price: Decimal128('1075.7'),
      quantity: 3
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('12.27'),
      quantity: 4
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('15.18'),
      quantity: 4
    }
  ],
  storeLocation: 'London',
  customer: { gender: 'F', age: 35, email: 'ujci@jumpu.as', satisfaction: 5 },
  couponUsed: false,
  purchaseMethod: 'Online',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd0df'),
  saleDate: ISODate('2017-02-24T14:36:50.910Z'),
  items: [
    {
      name: 'laptop',
      tags: [ 'electronics', 'school', 'office' ],
      price: Decimal128('868.83'),
      quantity: 4
    },
    {
      name: 'backpack',
      tags: [ 'school', 'travel', 'kids' ],
      price: Decimal128('112.75'),
      quantity: 1
```

```
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('13.07'),
      quantity: 2
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('14.36'),
      quantity: 5
    },
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
      price: Decimal128('38.23'),
      quantity: 3
    },
    {
      name: 'printer paper',
      tags: [ 'office', 'stationary' ],
      price: Decimal128('43.71'),
      quantity: 9
    },
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('15.95'),
      quantity: 6
    },
    {
      name: 'binder',
      tags: [ 'school', 'general', 'organization' ],
      price: Decimal128('13.11'),
      quantity: 9
    }
  ],
  storeLocation: 'London',
  customer: { gender: 'F', age: 34, email: 'anieti@conmu.cw', satisfaction: 4 },
  couponUsed: false,
  purchaseMethod: 'In store',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd0f0'),
  saleDate: ISODate('2013-02-24T15:44:08.600Z'),
  items: [
    {
```

```
    name: 'pens',
    tags: [ 'writing', 'office', 'school', 'stationary' ],
    price: Decimal128('20.29'),
    quantity: 4
  },
  {
    name: 'notepad',
    tags: [ 'office', 'writing', 'school' ],
    price: Decimal128('20.48'),
    quantity: 4
  },
  {
    name: 'envelopes',
    tags: [ 'stationary', 'office', 'general' ],
    price: Decimal128('22.12'),
    quantity: 4
  },
  {
    name: 'binder',
    tags: [ 'school', 'general', 'organization' ],
    price: Decimal128('17.73'),
    quantity: 3
  },
  {
    name: 'laptop',
    tags: [ 'electronics', 'school', 'office' ],
    price: Decimal128('1483.86'),
    quantity: 1
  },
  {
    name: 'pens',
    tags: [ 'writing', 'office', 'school', 'stationary' ],
    price: Decimal128('30.1'),
    quantity: 1
  },
  {
    name: 'binder',
    tags: [ 'school', 'general', 'organization' ],
    price: Decimal128('29.17'),
    quantity: 3
  },
  {
    name: 'printer paper',
    tags: [ 'office', 'stationary' ],
    price: Decimal128('41.86'),
    quantity: 2
  }
],
storeLocation: 'London',
```

```
  customer: { gender: 'F', age: 19, email: 'gi@binvozfo.at', satisfaction: 4 },
  couponUsed: true,
  purchaseMethod: 'In store',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd120'),
  saleDate: ISODate('2013-12-04T07:57:09.171Z'),
  items: [
    {
      name: 'binder',
      tags: [ 'school', 'general', 'organization' ],
      price: Decimal128('12.17'),
      quantity: 6
    },
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
      price: Decimal128('59.14'),
      quantity: 1
    },
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('5.51'),
      quantity: 10
    },
    {
      name: 'binder',
      tags: [ 'school', 'general', 'organization' ],
      price: Decimal128('22.86'),
      quantity: 4
    },
    {
      name: 'backpack',
      tags: [ 'school', 'travel', 'kids' ],
      price: Decimal128('168.73'),
      quantity: 3
    },
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('8.68'),
      quantity: 5
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('19.57'),
```

```
      quantity: 1
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('32.96'),
      quantity: 3
    }
  ],
  storeLocation: 'London',
  customer: { gender: 'M', age: 47, email: 'bamu@jakasi.jm', satisfaction: 4 },
  couponUsed: false,
  purchaseMethod: 'In store',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd131'),
  saleDate: ISODate('2017-03-28T11:06:27.079Z'),
  items: [
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
      price: Decimal128('47.64'),
      quantity: 2
    },
    {
      name: 'laptop',
      tags: [ 'electronics', 'school', 'office' ],
      price: Decimal128('1507.67'),
      quantity: 2
    },
    {
      name: 'binder',
      tags: [ 'school', 'general', 'organization' ],
      price: Decimal128('26.78'),
      quantity: 2
    },
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('22.73'),
      quantity: 4
    },
    {
      name: 'backpack',
      tags: [ 'school', 'travel', 'kids' ],
      price: Decimal128('147.25'),
      quantity: 3
    },
```

```
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('20.41'),
      quantity: 7
    }
  ],
  storeLocation: 'London',
  customer: {
    gender: 'M',
    age: 66,
    email: 'fimute@duhadavu.edu',
    satisfaction: 4
  },
  couponUsed: false,
  purchaseMethod: 'In store',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd13f'),
  saleDate: ISODate('2014-11-30T18:03:06.614Z'),
  items: [
    {
      name: 'notepad',
      tags: [ 'office', 'writing', 'school' ],
      price: Decimal128('17.63'),
      quantity: 5
    }
  ],
  storeLocation: 'London',
  customer: { gender: 'F', age: 63, email: 'ce@zoslemu.aw', satisfaction: 1 },
  couponUsed: false,
  purchaseMethod: 'Phone',
  score: 0.8888952136039734
},
{
  _id: ObjectId('5bd761dcae323e45a93cd151'),
  saleDate: ISODate('2013-07-15T08:05:25.382Z'),
  items: [
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('16.8'),
      quantity: 3
    },
    {
      name: 'backpack',
      tags: [ 'school', 'travel', 'kids' ],
      price: Decimal128('137.84'),
```

```
      quantity: 5
    },
    {
      name: 'envelopes',
      tags: [ 'stationary', 'office', 'general' ],
      price: Decimal128('19.41'),
      quantity: 9
    },
    {
      name: 'pens',
      tags: [ 'writing', 'office', 'school', 'stationary' ],
      price: Decimal128('20.71'),
      quantity: 3
    },
    {
      name: 'laptop',
      tags: [ 'electronics', 'school', 'office' ],
      price: Decimal128('738.23'),
      quantity: 5
    }
  ],
  storeLocation: 'London',
  customer: { gender: 'M', age: 27, email: 'vag@lez.sv', satisfaction: 5 },
  couponUsed: false,
  purchaseMethod: 'Online',
  score: 0.8888952136039734
 }
]
Type "it" for more
Atlas atlas-x02tt5-shard-0 [primary] sample_supplies>
[ mongodb ][                    (0*bash)                    ][ 06/06 11:04 ]
```

##

Grouping Search Results by Using Facets: $searchMeta and facet

$searchMeta is an aggregation stage for Atlas Search where the metadata related to the search is shown. This means that if our search results are broken into buckets, using facet, we can see that in the $searchMeta stage, because those buckets are information about how the search results are formatted.

```
$searchMeta: {
   "facet": {
      "operator": {
         "text": {
         "query": ["Northern Cardinal"],
         "path": "common_name"
         }
      },
      "facets": {
         "sightingWeekFacet": {
            "type": "date",
            "path": "sighting",
            "boundaries": [ISODate("2022-01-01"),
               ISODate("2022-01-08"),
               ISODate("2022-01-15"),
               ISODate("2022-01-22")],
            "default" : "other"
         }
      }
   }
}
```

"facet" is an operator within $searchMeta. "operator" refers to the search operator - the query itself. "facets" operator is where we put the definition of the buckets for the facets.

##

Create file search_index.json

```
{
    "name": "sample_supplies-sales-facets",
    "searchAnalyzer": "lucene.standard",
    "analyzer": "lucene.standard",
    "collectionName": "sales",
    "database": "sample_supplies",
    "mappings": {
        "dynamic": true,
        "fields": {
        "purchaseMethod": [
            {
            "dynamic": true,
            "type": "document"
            },
```

```
            {
            "type": "string"
            }
        ],
        "storeLocation": [
            {
            "dynamic": true,
            "type": "document"
            },
            {
            "type": "string"
            }
        ]
        }
    }
}
```

root@mongodb:~$ atlas clusters search indexes create --clusterName myAtlasClusterEDU -f /app/search_index.json
⇒ Index sample_supplies-sales-facets created.
root@mongodb:~$ atlas clusters search indexes list --clusterName myAtlasClusterEDU --db sample_supplies
--collection sales
⇒ ID NAME DATABASE COLLECTION TYPE
6661977f2c545368645b880c sample_supplies-sales-static sample_supplies sales search
6661999a5d94c44bf51cb399 sample_supplies-sales-facets sample_supplies sales search
root@mongodb:~$

####

```
"facet": {
  "operator": {
    "text": {
      "query": ["In store"],
      "path": "purchaseMethod"
    }
  }
}
```

```
"facet": {
  "operator": {
    "text": {
      "query": ["In store"],
      "path": "purchaseMethod"
    }
  }}
```

Python

CRUD Operations

- **insertone()** inserts one document in the collection

```
Atlas atlas-4e9k8f-shard-0 [primary] training> db.grades.insertOne( {
...   student_id: 546799,
...      scores: [
...        {
.....           type: "quiz",
.....           score: 50,
.....         },
...        {
.....           type: "homework",
.....           score: 70,
.....         }
...      ]
... } )
{
  acknowledged: true,
  insertedId: ObjectId("62f138664985719fd9fb98f6")
}
Atlas atlas-4e9k8f-shard-0 [primary] training>
```

- **insertmany()**

```
Atlas atlas-4e9k8f-shard-0 [primary] training> db.grades.insertMany([
...   {
.....     student_id: 546789,
.....     scores: [
.....       {
.......         type: "quiz",
.......         score: 50,
.......     },
.....       {
.......         type: "homework",
.......         score: 70,
.......     }
.....     ],
.....     class_id: 551,
.....   },
...   {
.....     student_id: 777777,
.....     scores: [
.....       {
.......         type: "quiz",
.......         score: 72,
.......     }
.....     ],
.....     class_id: 550,
.....   },
...   {
.....     student_id: 223344,
.....     scores: [
.....       {
.......         type: "exam",
.......         score: 45,
```

- Insert a Single Document

Use insertOne() to insert a document into a collection. Within the parentheses of insertOne(), include an object that contains the document data. Here's an example:

```
db.grades.insertOne({
  student_id: 654321,
  products: [
    {
      type: "exam",
      score: 90,
    },
    {
      type: "homework",
      score: 59,
    },
    {
      type: "quiz",
      score: 75,
    },
    {
```

```
      type: "homework",
      score: 88,
    },
  ],
  class_id: 550,
})
```

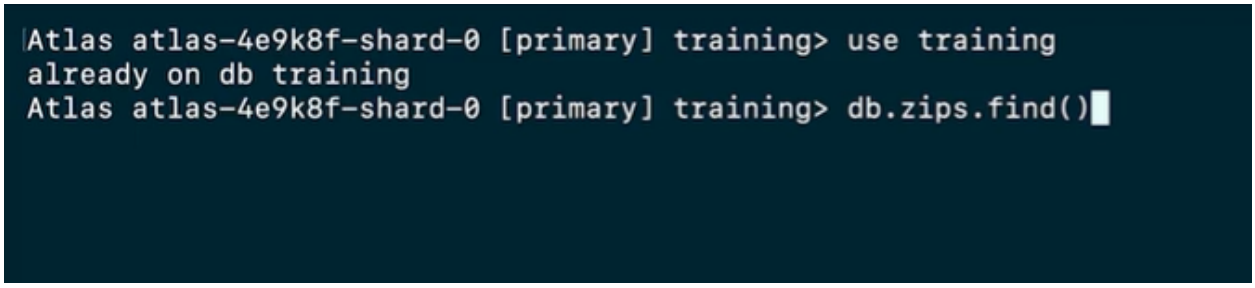
- **Insert Multiple Documents**

Use insertMany() to insert multiple documents at once. Within insertMany(), include the documents within an array. Each document should be separated by a comma. Here's an example:

```
db.grades.insertMany([
  {
    student_id: 546789,
    products: [
      {
        type: "quiz",
        score: 50,
      },
      {
        type: "homework",
        score: 70,
      },
      {
        type: "quiz",
        score: 66,
      },
      {
        type: "exam",
        score: 70,
      },
    ],
    class_id: 551,
  },
  {
    student_id: 777777,
    products: [
      {
        type: "exam",
        score: 83,
      },
      {
        type: "quiz",
        score: 59,
      },
      {
        type: "quiz",
        score: 72,
      },
```

```
    {
      type: "quiz",
      score: 67,
    },
  ],
  class_id: 550,
},
{
  student_id: 223344,
  products: [
    {
      type: "exam",
      score: 45,
    },
    {
      type: "homework",
      score: 39,
    },
    {
      type: "quiz",
      score: 40,
    },
    {
      type: "homework",
      score: 88,
    },
  ],
  class_id: 551,
},
])
```

- db.collection. find()



```
Current Mongosh Log ID: 66649433c6af68f533a26a12
Connecting to:          mongodb+srv://<credentials>@instruqttest.3xfvk.mongodb.net/sample_analytics?appName=mongosh+2.2.6
Using MongoDB:          6.0.15
Using Mongosh:          2.2.6

For mongosh info see: https://docs.mongodb.com/mongodb-shell/


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/leg
al/privacy-policy).
You can opt-out by running the disableTelemetry() command.

Atlas atlas-d3opcw-shard-0 [primary] sample_analytics> db.accounts
sample_analytics.accounts
Atlas atlas-d3opcw-shard-0 [primary] sample_analytics> db.accounts.insertOne(
...   {
...     account_id: 111333,
...     limit: 12000,
...     products: [
...       "Commodity",
...       "Brokerage"
...       ],
...     "last_updated": new Date()
...   }
... )
{
  acknowledged: true,
  insertedId: ObjectId('66649450c6af68f533a26a13')
}
Atlas atlas-d3opcw-shard-0 [primary] sample_analytics>
```

####

```
Atlas atlas-d3opcw-shard-0 [primary] sample_analytics> db.accounts.insertMany(
...   [
...     {
...       "account_id": 111333,
...       "limit": 12000,
...       "products": [
...       "Commodity",
...       "Brokerage"
...       ],
...       "last_updated": new Date()
...     },
...     {
...       "account_id": 678943,
...       "limit": 8000,
...       "products": [
...       "CurrencyService",
...       "Brokerage",
...       "InvestmentStock"
...       ],
...       "last_updated": new Date()
...     },
...     {
...       "account_id": 321654,
...       "limit": 10000,
...       "products": [
...       "Commodity",
...       "CurrencyService"
...       ],
...       "last_updated": new Date()
...     }
...   ]
... )
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('666495202358ebe109a26a13'),
    '1': ObjectId('666495202358ebe109a26a14'),
    '2': ObjectId('666495202358ebe109a26a15')
```

###

Find a Document with Equality

- When given equality with an _id field, the find() command will return the specified document that matches the _id. Here's an example:

db.zips.find({ **_id: ObjectId("5c8eccc1caa187d17ca6ed16**") })

- Find a Document by Using the $in Operator

Use the $in operator to select documents where the value of a field equals any value in the specified array. Here's an example:

- db.zips.find({ city: { $in: ["PHOENIX", "CHICAGO"] } })

Eg: db.sales.find({ storeLocation: { $in: ["London", "New York"] } }) // All the documents with city london and New York

#####

Remember, to access subdocuments, you must use the syntax “field.nestedfield”, which includes quotation marks.

```
Atlas atlas-4e9k8f-shard-0 [primary] training> db.sales.find({ "items.price": { $gt: 50 } })
[
  {
    _id: ObjectId("62d18ebfe46fce3f14998fcd"),
    items: [
      {
        name: 'copier',
        tags: [ 'stationary', 'office', 'general' ],
        price: Decimal128("1250.99"),
        quantity: 1
      }
    ],
    customer: { gender: 'M', age: 67, email: 'bernie@com.com', satisfaction: 5 }
  },
  {
    _id: ObjectId("62d1e52ee46fce3f14998fce"),
    items: [
      {
        name: 'laptop',
        tags: [ 'stationary', 'office', 'general' ],
        price: Decimal128("829.99"),
        quantity: 3
      }
    ],
    customer: { gender: 'M', age: 65, email: 'johndo@ha.ck', satisfaction: 5 }
  }
]
Atlas atlas-4e9k8f-shard-0 [primary] training>
```

```
Atlas atlas-4e9k8f-shard-0 [primary] training> db.sales.find({ "customer.age": { $gte: 65 } })
[
  {
    _id: ObjectId("62d18ebfe46fce3f14998fcd"),
    items: [
      {
        name: 'copier',
        tags: [ 'stationary', 'office', 'general' ],
        price: Decimal128("1250.99"),
        quantity: 1
      }
    ],
    customer: { gender: 'M', age: 67, email: 'bernie@com.com', satisfaction: 5 }
  },
  {
    _id: ObjectId("62d1e52ee46fce3f14998fce"),
    items: [
      {
        name: 'laptop',
        tags: [ 'stationary', 'office', 'general' ],
        price: Decimal128("829.99"),
        quantity: 3
      }
    ],
    customer: { gender: 'M', age: 65, email: 'johndo@ha.ck', satisfaction: 5 }
  }
]
Atlas atlas-4e9k8f-shard-0 [primary] training>
```

Finding Documents by Using Comparison Operators

Review the following comparison operators: $gt, $lt, $lte, and $gte.

$gt

Use the $gt operator to match documents with a field greater than the given value. For example:

db.sales.find({ "items.price": { $gt: 50}})

$lt
Use the $lt operator to match documents with a field less than the given value. For example:

```
db.sales.find({ "items.price": { $lt: 50}})
```

$lte
Use the $lte operator to match documents with a field less than or equal to the given value. For example:

```
db.sales.find({ "customer.age": { $lte: 65}})
```

$gte
Use the $gte operator to match documents with a field greater than or equal to the given value. For example:

```
db.sales.find({ "customer.age": { $gte: 65}})
```

######

Query array in document
$elemMatch{}

```
Atlas atlas-4e9k8f-shard-0 [primary] training> db.accounts.find({
...     products: {
.....         $elemMatch: { $eq: "InvestmentStock" }
.....   }
... })
[
  {
    _id: ObjectId("5ca4bbc7a2dd94ee58162a1f"),
    account_id: 135875,
    limit: 10000,
    products: [ 'Derivatives', 'CurrencyService', 'InvestmentStock' ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee581624e3"),
    account_id: 298562,
    limit: 10000,
    products: [
      'Derivatives',
      'Commodity',
      'InvestmentFund',
      'Brokerage',
      'InvestmentStock'
    ]
  }
]
Atlas atlas-4e9k8f-shard-0 [primary] training>
```

Querying on Array Elements in MongoDB
Review the following code, which demonstrates how to query array elements in MongoDB.

Find Documents with an Array That Contains a Specified Value
In the following example, "InvestmentFund" is not enclosed in square brackets, so MongoDB returns all documents within the products array that contain the specified value.

db.accounts.find({ products: "InvestmentFund"})
Find a Document by Using the $elemMatch Operator
Use the $elemMatch operator to find all documents that contain the specified subdocument. For example:

```
db.sales.find({
  items: {
    $elemMatch: { name: "laptop", price: { $gt: 800 }, quantity: { $gte: 1 } },
  },
})
###
```

Create a query that matches all documents with a `transactions` sub document that contains an `amount` less than or equal to $4500 and a `transaction_code` of "sell" in the `transactions` collection

⇒

```
db.transactions.find({
   transactions: {
     $elemMatch: { amount: { $lte: 4500 }, transaction_code: "sell" },
   },
 })
####
```

Finding Documents by Using Logical Operators

Review the following logical operators: implicit $and, $or, and $and.

Find a Document by Using Implicit $and

Use implicit $and to select documents that match multiple expressions. For example:

```
db.routes.find({ "airline.name": "Southwest Airlines", stops: { $gte: 1 } })
```

Find a Document by Using the $or Operator

Use the $or operator to select documents that match at least one of the included expressions. For example:

```
db.routes.find({
  $or: [{ dst_airport: "SEA" }, { src_airport: "SEA" }],
})
```

Find a Document by Using the $and Operator

Use the $and operator to use multiple $or expressions in your query.

```
db.routes.find({
  $and: [
    { $or: [{ dst_airport: "SEA" }, { src_airport: "SEA" }] },
    { $or: [{ "airline.name": "American Airlines" }, { airplane: 320 }] },
  ]
})
```

#######

Replacing a Document in MongoDB

```
Atlas atlas-4e9k8f-shard-0 [primary] library> db.books.replaceOne(
...   { _id: ObjectId("62c5e671541e2c6bcb528308") },
...   {
.....     title: "Deep Dive into React Hooks",
.....     ISBN: "0-3182-1299-4",
.....     thumbnailUrl: "http://via.placeholder.com/640x360",
.....     publicationDate: ISODate("2022-07-28T02:20:21.000Z"),
.....     authors: ["Ada Lovelace"],
.....   }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Atlas atlas-4e9k8f-shard-0 [primary] library>
```

To replace documents in MongoDB, we use the replaceOne() method. The replaceOne() method takes the following parameters:

filter: A query that matches the document to replace.
replacement: The new document to replace the old one with.
options: An object that specifies options for the update.
In the previous video, we use the _id field to filter the document. In our replacement document, we provide the entire document that should be inserted in its place. Here's the example code from the video:

```
db.books.replaceOne(
 {
  _id: ObjectId("6282afeb441a74a98dbbec4e"),
 },
 {
  title: "Data Science Fundamentals for Python and MongoDB",
  isbn: "1484235967",
  publishedDate: new Date("2018-5-10"),
  thumbnailUrl:
   "https://m.media-amazon.com/images/I/71opmUBc2wL._AC_UY218_.jpg",
  authors: ["David Paper"],
  categories: ["Data Science"],  })
```

```
Atlas atlas-4e9k8f-shard-0 [primary] library> use library
already on db library
Atlas atlas-4e9k8f-shard-0 [primary] library> db.podcasts.findOne({ title: "The MongoDB Podcast"})
{
  _id: ObjectId("6261a92dfee1ff300dc80bf1"),
  title: 'The MongoDB Podcast',
  platforms: [ 'Apple Podcasts', 'Spotify' ],
  year: 2022,
  hosts: [ ],
  premium_subs: 4152,
  downloads: 2,
  podcast_type: 'audio'
}
Atlas atlas-4e9k8f-shard-0 [primary] library>
```

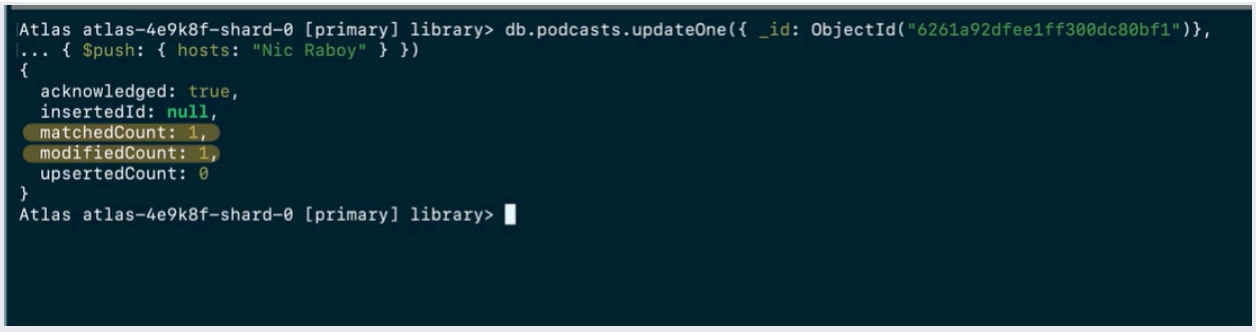
$set operator to replace or update value:

```
Atlas atlas-4e9k8f-shard-0 [primary] library> db.podcasts.updateOne(
...   { _id: ObjectId("6261a92dfee1ff300dc80bf1") },
...   { $set: { subscribers: 98562 } } )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Atlas atlas-4e9k8f-shard-0 [primary] library>
```

$upser operator to update a document when it does not exist before

```
Atlas atlas-4e9k8f-shard-0 [primary] library> db.podcasts.updateOne(
...   { title: "The Developer Hub" },
...   { $set: { topics: ["databases", "MongoDB"] } },
...   { upsert: true })
{
  acknowledged: true,
  insertedId: ObjectId("6303e097fc41e97a569ef868"),
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 1
}
Atlas atlas-4e9k8f-shard-0 [primary] library>
```

$ push operator: Add element to an array



Updating MongoDB Documents by Using updateOne()

The updateOne() method accepts a filter document, an update document, and an optional options object. MongoDB provides update operators and options to help you update documents. In this section, we'll cover three of them: $set, upsert, and $push.

$set

The $set operator replaces the value of a field with the specified value, as shown in the following code:

```
db.podcasts.updateOne(
  {
    _id: ObjectId("5e8f8f8f8f8f8f8f8f8f8f8"),
  },

  {
    $set: {
      subscribers: 98562,
    },
  }
)
```

upsert

The upsert option creates a new document if no documents match the filtered criteria. Here's an example:

```
db.podcasts.updateOne(
  { title: "The Developer Hub" },
  { $set: { topics: ["databases", "MongoDB"] } },
  { upsert: true }
)
```

$push

The $push operator adds a new value to the hosts array field. Here's an example:

```
db.podcasts.updateOne(
```

{ _id: ObjectId("5e8f8f8f8f8f8f8f8f8f8f8") },
{ $push: { hosts: "Nic Raboy" } }

```
For mongosh info see: https://docs.mongodb.com/mongodb-shell/

Atlas atlas-xbzsi3-shard-0 [primary] bird_data> db.birds.updateOne(
...   { _id: ObjectId("6268471e613e55b82d7065d7") },
...   {
...     $push: {
...       diet: { $each: ["newts", "opossum", "skunks", "squirrels"] },
...     },
...   }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Atlas atlas-xbzsi3-shard-0 [primary] bird_data> root@mongodb:~$
```

```
For mongosh info see: https://docs.mongodb.com/mongodb-shell/

Atlas atlas-xbzsi3-shard-0 [primary] bird_data> db.birds.updateOne(
...   {
...     common_name: "Robin Redbreast",
...   },
...   {
...     $inc: {
...       "sightings": 1,
...     },
...     $set: {
...       last_updated: new Date(),
...     },
...   },
...   {
...     upsert: true,
...   }
... )
{
  acknowledged: true,
  insertedId: ObjectId('6664a63de652ceb43abb1000'),
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 1
}
Atlas atlas-xbzsi3-shard-0 [primary] bird_data>
```

Updating MongoDB Documents by Using `findAndModify()`

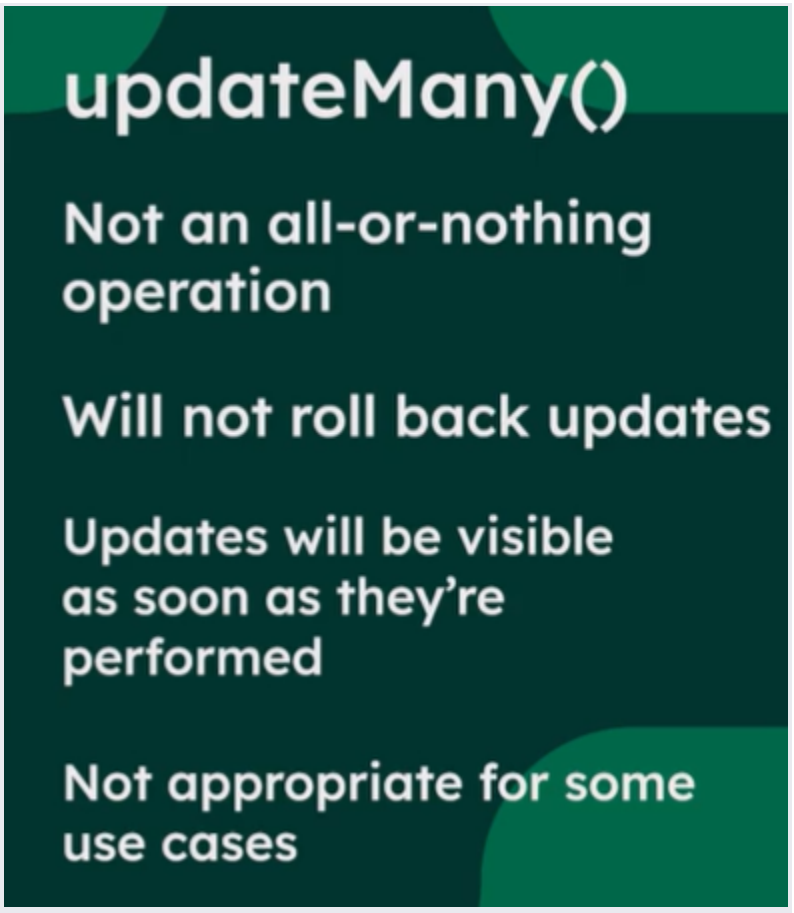
The `findAndModify()` method is used to find and replace a single document in MongoDB. It accepts a filter document, a replacement document, and an optional options object. It saves round trip to server twice. The following code shows an example:

```
db.podcasts.findAndModify({
  query: { _id: ObjectId("62c7361f4bf2017af21ccdde") },
  update: { $inc: { downloads: 1 } },
  new: true
})
```

#####

Updatemany()

```
Atlas atlas-4e9k8f-shard-0 [primary] library> db.books.updateMany(
...   { publishedDate: { $lt: ISODate("2019-01-01T08:00:00.000Z") } },
...   { $set: { status: "LEGACY" } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 353,
  modifiedCount: 353,
  upsertedCount: 0
}
Atlas atlas-4e9k8f-shard-0 [primary] library>
```



###

Deleting Documents in MongoDB

To delete documents, use the deleteOne() or deleteMany() methods. Both methods accept a filter document and an options object.

Delete One Document
The following code shows an example of the deleteOne() method:

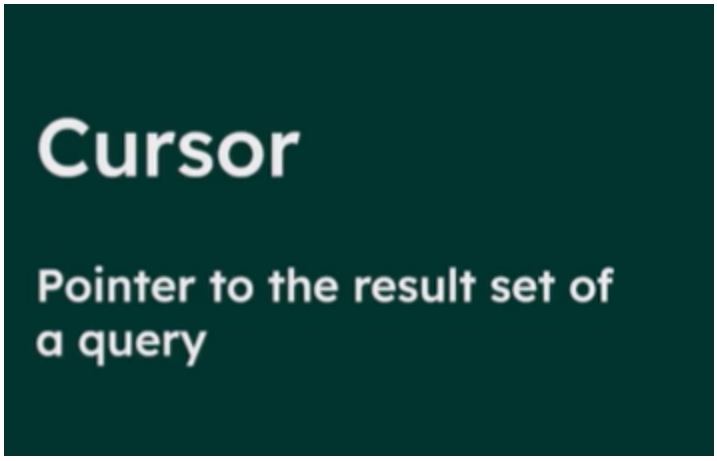
db.podcasts.deleteOne({ _id: Objectid("6282c9862acb966e76bbf20a") })

Delete Many Documents
The following code shows an example of the deleteMany() method:

db.podcasts.deleteMany({category: “crime”})
##############
MongoDB CRUD Operations: Replace and Delete Documents

Replaced a single document by using **db.collection.replaceOne()**.
Updated a field value by using the $set update operator in **db.collection.updateOne()**.
Added a value to an array by using the $push update operator in **db.collection.updateOne()**.
Added a new field value to a document by using the upsert option in **db.collection.updateOne().**
Found and modified a document by using **db.collection.findAndModify()**.
Updated multiple documents by using **db.collection.updateMany()**.
Deleted a document by using **db.collection.deleteOne()**.

######



cursor.sort()

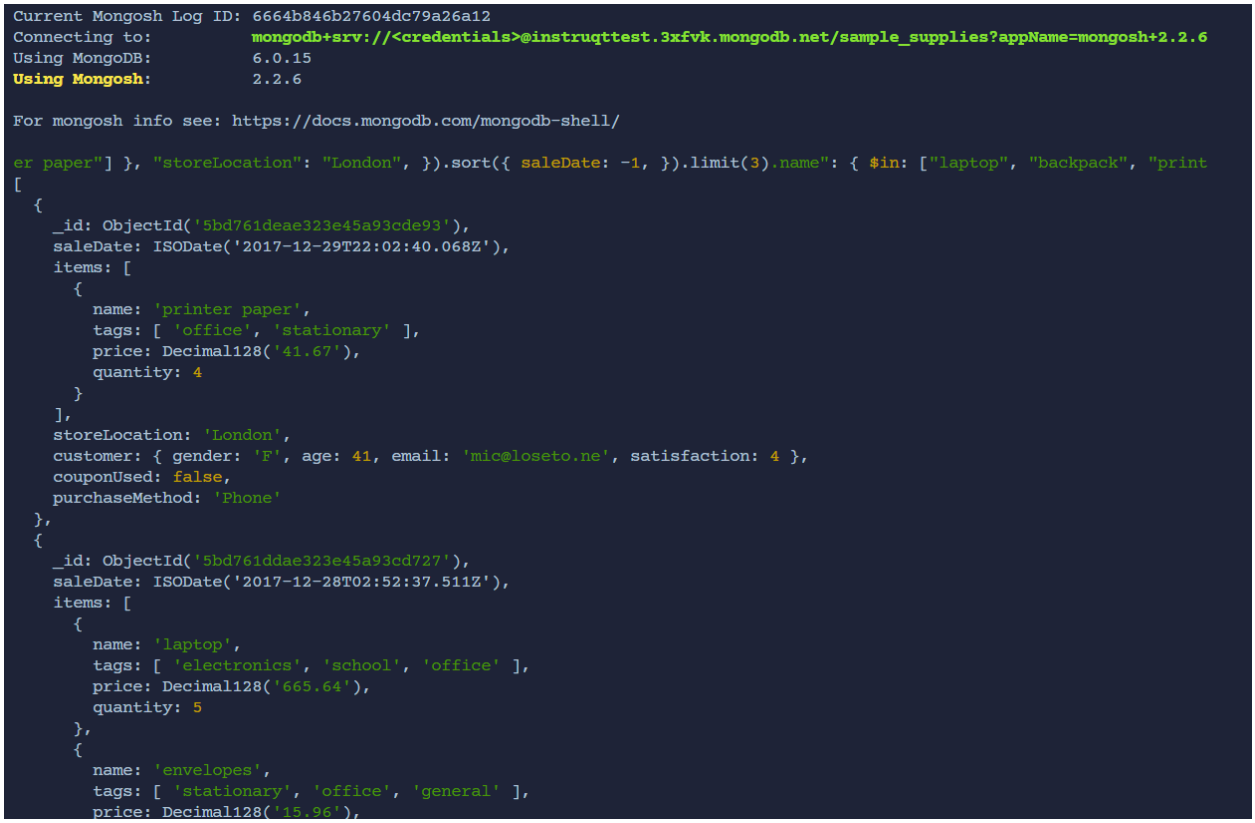
The sort() method takes an object consisting of one or more key-value pairs, in which the keys are the fields to sort by and the values represent the sort direction (1 or -1 to specify an ascending or descending sort order respectively.)

```
Atlas atlas-a847ea-shard-0 [primary] training> db.companies.find({category_code:"music"},{name:1}).sort({name:1})
[
  { _id: ObjectId("52cdef7e4bab8bd67529ba19"), name: 'Audocs' },
  { _id: ObjectId("52cdef7d4bab8bd675299bb3"), name: 'Band Metrics' },
  { _id: ObjectId("52cdef7d4bab8bd675298de2"), name: 'Bandsintown' },
  { _id: ObjectId("52cdef7e4bab8bd67529a67f"), name: 'Club Cooee' },
  { _id: ObjectId("52cdef7d4bab8bd675298d51"), name: 'MIKESTAR' },
  { _id: ObjectId("52cdef7c4bab8bd675297fed"), name: 'Myxer' },
  { _id: ObjectId("52cdef7d4bab8bd675298d6e"), name: 'Nimbit' },
  {
    _id: ObjectId("52cdef7f4bab8bd67529c6e6"),
    name: 'OfficialVirtualDJ'
  },
  { _id: ObjectId("52cdef7c4bab8bd6752980f2"), name: 'OurStage' },
  { _id: ObjectId("52cdef7d4bab8bd675299e42"), name: 'Radionomy' },
  { _id: ObjectId("52cdef7d4bab8bd675299042"), name: 'Rhapsody' },
  { _id: ObjectId("52cdef7c4bab8bd675297d98"), name: 'Slacker' },
  { _id: ObjectId("52cdef7d4bab8bd675299dcd"), name: 'Smule' },
  { _id: ObjectId("52cdef7d4bab8bd67529891b"), name: 'Spotify' },
  { _id: ObjectId("52cdef7f4bab8bd67529be8f"), name: 'Stereomood' },
  { _id: ObjectId("52cdef7c4bab8bd675297f53"), name: 'Thumbplay' },
  { _id: ObjectId("52cdef7c4bab8bd6752981f7"), name: 'blinkbox music' },
  { _id: ObjectId("52cdef7c4bab8bd675297f83"), name: 'imeem' }
]
Atlas atlas-a847ea-shard-0 [primary] training>
```

cursor.limit(): Will improve performance of query

```
Atlas atlas-a847ea-shard-0 [primary] training> db.companies.find({category_code:"music"},{name:1,number_of_employees:1}).sort({number_of_employees:-1})
.limit(3)
[
  {
    _id: ObjectId("52cdef7d4bab8bd67529891b"),
    name: 'Spotify',
    number_of_employees: 5000
  },
  {
    _id: ObjectId("52cdef7d4bab8bd675299042"),
    name: 'Rhapsody',
    number_of_employees: 150
  },
  {
    _id: ObjectId("52cdef7f4bab8bd67529c6e6"),
    name: 'OfficialVirtualDJ',
    number_of_employees: 102
  }
]
Atlas atlas-a847ea-shard-0 [primary] training>
```

Sorting and Limiting Query Results in MongoDB



Review the following code, which demonstrates how to sort and limit query results.

Sorting Results

Use cursor.sort() to return query results in a specified order. Within the parentheses of sort(), include an object that specifies the field(s) to sort by and the order of the sort. Use 1 for ascending order, and -1 for descending order.

Syntax:

db.collection.find(<query>).sort(<sort>)

Example:

// Return data on all music companies, sorted alphabetically from A to Z.

db.companies.find({ category_code: "music" }).sort({ name: 1 });

To ensure documents are returned in a consistent order, include a field that contains unique values in the sort. An easy way to do this is to include the _id field in the sort. Here's an example:

// Return data on all music companies, sorted alphabetically from A to Z. Ensure consistent sort order

db.companies.find({ category_code: "music" }).sort({ name: 1, _id: 1 });

Limiting Results

Use cursor.limit() to specify the maximum number of documents the cursor will return. Within the parentheses of limit(), specify the maximum number of documents to return.

Syntax:

db.companies.find(<query>).limit(<number>)

Example:

// Return the three music companies with the highest number of employees. Ensure consistent sort order.

```
db.companies
  .find({ category_code: "music" })
  .sort({ number_of_employees: -1, _id: 1 })
  .limit(3);
```

###

Projection: Returning Specific Data from a Query in MongoDB

```
Atlas atlas-a847ea-shard-0 [primary] training> db.inspections.findOne()
{
  _id: ObjectId("56d61033a378eccde8a8354f"),
  id: '10021-2015-ENFO',
  certificate_number: 9278806,
  business_name: 'ATLIXCO DELI GROCERY INC.',
  date: 'Feb 20 2015',
  result: 'No Violation Issued',
  sector: 'Cigarette Retail Dealer - 127',
  address: { city: 'RIDGEWOOD', zip: 11385, street: 'MENAHAN ST', number: 1712 }
}
Atlas atlas-a847ea-shard-0 [primary] training>
```

Include:

```
db.collection.find(<query>, {<field>:1, <field>:1})
```

Exclude:

```
db.collection.find(<query>, {<field>:0, <field>:0})
```

Review the following code, which demonstrates how to return selected fields from a query.

Add a Projection Document

To specify fields to include or exclude in the result set, add a projection document as the second parameter in the call to db.collection.find().

Syntax:

db.collection.find(<query>, <projection>)

Include a Field

To include a field, set its value to 1 in the projection document.

Syntax:

db.collection.find(<query>, { <field> : 1 })

Example:

// Return all restaurant inspections - business name, result, and _id fields only
db.inspections.find(
  { sector: "Restaurant - 818" },
  { business_name: 1, result: 1 }
)
Exclude a Field
To exclude a field, set its value to 0 in the projection document.

Syntax:

db.collection.find(query, { <field> : 0, <field>: 0 })
Example:

// Return all inspections with result of "Pass" or "Warning" - exclude date and zip code
db.inspections.find(
  { result: { $in: ["Pass", "Warning"] } },
  { date: 0, "address.zip": 0 }
)
While the _id field is included by default, it can be suppressed by setting its value to 0 in any projection.

// Return all restaurant inspections - business name and result fields only
db.inspections.find(
  { sector: "Restaurant - 818" },
  { business_name: 1, result: 1, _id: 0 }
####

Counting Documents in a MongoDB Collection

Use `db.collection.countDocuments()` to count the number of documents that match a query. `countDocuments()` takes two parameters: a query document and an options document.

Syntax:

```
db.collection.countDocuments( <query>, <options> )
```

The query selects the documents to be counted.

Examples:

```
// Count number of docs in trip collection
db.trips.countDocuments({})
// Count number of trips over 120 minutes by subscribers
db.trips.countDocuments({ tripduration: { $gt: 120 }, usertype:
"Subscriber" })
###########
```

MongoDB CRUD Operations in Python

- BSON document are represented as Python dictionaries
- Pymongo and bson package
-

Drivers > Python > Apps > insert_single.py > ...

```python
import datetime
import os

from dotenv import load_dotenv
from pymongo import MongoClient

# Load config from .env file
load_dotenv()
MONGODB_URI = os.environ["MONGODB_URI"]

# Connect to MongoDB cluster with MongoClient
client = MongoClient(MONGODB_URI)

# Get reference to 'bank' database
db = client.bank

# Get reference to 'accounts' collection
accounts_collection = db.accounts

new_account = {
    "account_holder": "Linus Torvalds",
    "account_id": "MDB829001337",
    "account_type": "checking",
    "balance": 50352434,
    "last_updated": datetime.datetime.utcnow(),
}

# Write an expression that inserts the 'new_account' document into the 'accounts' collection.
result = accounts_collection.insert_one(new_account)

document_id = result.inserted_id
print(f"_id of inserted document: {document_id}")

client.close()
```

-

Inserting a Document in Python Applications

Review the following code, which demonstrates how to insert a single document and multiple documents into a MongoDB collection by using PyMongo.

Insert One Document

To insert a single document into a collection, append insert_one() to the collection object. The insert_one() method accepts a document as an argument and returns a result. In this example, we use the result to print the _id value of the inserted document.

In the following code, the document that's being inserted is stored in a variable called new_account. This variable is declared just above the expression that inserts the document.

```
# Get reference to 'bank' database
db = client.bank

# Get reference to 'accounts' collection
accounts_collection = db.accounts

new_account = {
   "account_holder": "Linus Torvalds",
   "account_id": "MDB829001337",
   "account_type": "checking",
   "balance": 50352434,
   "last_updated": datetime.datetime.utcnow(),
}

# Write an expression that inserts the 'new_account' document into the 'accounts' collection.
result = accounts_collection.insert_one(new_account)

document_id = result.inserted_id
print(f"_id of inserted document: {document_id}")

client.close()
```

Insert Multiple Documents

To insert more than one document into a collection, append the insert_many() method to the collection object. The insert_many() method accepts an iterable of documents as an argument and returns a result. In this example, we use the result to print out the number of documents inserted and their _id values.

In the following code, the accounts to be inserted are stored in a list variable called new_accounts. This variable is declared just above the expression that inserts the documents.

```
# Get reference to 'bank' database
db = client.bank

# Get a reference to 'accounts' collection
accounts_collection = db.accounts

new_accounts = [
```

```
    {
        "account_id": "MDB011235813",
        "account_holder": "Ada Lovelace",
        "account_type": "checking",
        "balance": 60218,
    },
    {
        "account_id": "MDB829000001",
        "account_holder": "Muhammad ibn Musa al-Khwarizmi",
        "account_type": "savings",
        "balance": 267914296,
    },
]

# Write an expression that inserts the documents in 'new_accounts' into the 'accounts'
collection.
result = accounts_collection.insert_many(new_accounts)

document_ids = result.inserted_ids
print("# of documents inserted: " + str(len(document_ids)))
print(f"_ids of inserted documents: {document_ids}")

client.close()
###
```

```python
import os
import pprint

from dotenv import load_dotenv
from pymongo import MongoClient

# Import ObjectId from bson package (part of PyMongo distribution) to enable querying by ObjectId
from bson.objectid import ObjectId

# Load config from .env file
load_dotenv()
MONGODB_URI = os.environ["MONGODB_URI"]

# Connect to MongoDB cluster with MongoClient
client = MongoClient(MONGODB_URI)

# Get reference to 'bank' database
db = client.bank

# Get a reference to the 'accounts' collection
accounts_collection = db.accounts

# Query by ObjectId
document_to_find = {"_id": ObjectId("62d6e04ecab6d8e1304974ae")}

# Write an expression that retrieves the document matching the query constraint in the 'accounts' collection.
result = accounts_collection.find one(document to find)
pprint.pprint(result)

client.close()
```

###

Querying a MongoDB Collection in Python Applications

Query for a Single Document

To return a single document that matches a query, append find_one() to the collection object. The find_one() method can accept a filter argument that specifies the query to be performed. In this example, the filter is assigned to the document_to_find variable.

find_one() returns the first document that matches the query, or it returns None if there are no matches. In this example, we use the result to print out the returned document.

```
# Get reference to 'bank' database
db = client.bank

# Get a reference to the 'accounts' collection
accounts_collection = db.accounts

# Query by ObjectId
document_to_find = {"_id": ObjectId("62d6e04ecab6d8e1304974ae")}

# Write an expression that retrieves the document matching the query constraint in the
'accounts' collection.
result = accounts_collection.find_one(document_to_find)
pprint.pprint(result)

client.close()
```

Query for Multiple Documents

To return all documents that match a query, append find() to the collection object. The find() method can accept a filter argument that specifies the query to be performed. In this example, the filter is assigned to the documents_to_find variable.

find() returns a Cursor instance, which allows us to iterate over all matching documents. In this example, we use the cursor to print out the documents matched by this query, as well as the total number of documents that are found.

```
# Get reference to 'bank' database
db = client.bank

# Get a reference to the 'accounts' collection
accounts_collection = db.accounts

# Query
documents_to_find = {"balance": {"$gt": 4700}}

# Write an expression that selects the documents matching the query constraint in the
'accounts' collection.
cursor = accounts_collection.find(documents_to_find)

num_docs = 0
for document in cursor:
    num_docs += 1
    pprint.pprint(document)
    print()
print("# of documents found: " + str(num_docs))

client.close()

####
```

Updating Documents in Python Applications

Update a Single Document

To update a single document that matches a query, append update_one() to the collection object.

The update_one() method has two required parameters: a filter document that matches the document to update, and an update document that specifies the modifications to apply. In this example, the filter is assigned to the document_to_update variable, and the update is assigned to the add_to_balance variable.

update_one() returns a result. In this example, we use the result to print a confirmation of the number of documents that were updated.

We also print the target document before and after the update. This way, we can confirm that the specified modification has been applied.

Here's the code:

```
# Get reference to 'bank' database
db = client.bank

# Get reference to 'accounts' collection
accounts_collection = db.accounts

# Filter
document_to_update = {"_id": ObjectId("62d6e04ecab6d8e130497482")}

# Update
add_to_balance = {"$inc": {"balance": 100}}

# Print original document
pprint.pprint(accounts_collection.find_one(document_to_update))

# Write an expression that adds to the target account balance by the specified amount.
result = accounts_collection.update_one(document_to_update, add_to_balance)
print("Documents updated: " + str(result.modified_count))

# Print updated document
pprint.pprint(accounts_collection.find_one(document_to_update))

client.close()
```

Update Multiple Documents

To update all documents that match a query, append update_many() to the collection object.

The update_many() method also has two required parameters: a filter document that matches the document to update, and an update document that specifies the modifications to apply. In this example, the filter is assigned to the select_accounts variable, and the update is assigned to the set_field variable. Note that in this update, we define a new field and set its initial value.

update_many() returns a result. In this example, we use the result to print out a confirmation of the number of documents that matched and were modified by the operation. We also print out one sample document after the update. This way, we can confirm that the specified modification has been applied.

Here's the code:

```
# Get reference to 'bank' database
db = client.bank

# Get reference to 'accounts' collection
accounts_collection = db.accounts

# Filter
select_accounts = {"account_type": "savings"}

# Update
set_field = {"$set": {"minimum_balance": 100}}

# Write an expression that adds a 'minimum_balance' field to each savings acccount and sets its value to 100.
result = accounts_collection.update_many(select_accounts, set_field)

print("Documents matched: " + str(result.matched_count))
print("Documents updated: " + str(result.modified_count))
pprint.pprint(accounts_collection.find_one(select_accounts))

client.close()
###
```

Deleting Documents in Python Applications

Delete a Single Document

To delete a single document that matches a query, append delete_one() to the collection object.

The delete_one() method has one required parameter, which is a filter that matches the document to delete. In the following example, the filter is assigned to the document_to_delete variable.

Note that if delete_one() is called with an empty query filter document, then it matches and deletes the first document in the collection.

delete_one() returns a result. In this example, we use the result to print the count of documents that were deleted.

We also query for the target document before and after the delete operation as an additional confirmation.

Here's the code:

```
# Get reference to 'bank' database
db = client.bank

# Get a reference to the 'accounts' collection
accounts_collection = db.accounts

# Filter by ObjectId
document_to_delete = {"_id": ObjectId("62d6e04ecab6d8e130497485")}

# Search for document before delete
print("Searching for target document before delete: ")
pprint.pprint(accounts_collection.find_one(document_to_delete))

# Write an expression that deletes the target account.
result = accounts_collection.delete_one(document_to_delete)

# Search for document after delete
print("Searching for target document after delete: ")
pprint.pprint(accounts_collection.find_one(document_to_delete))
```

```
print("Documents deleted: " + str(result.deleted_count))

client.close()
```

##

Delete Multiple Documents

To delete all documents that match a query, append delete_many() to the collection object.

The delete_many() method has one required parameter, which is a filter document that matches the document to delete. In the following example, the filter is assigned to the documents_to_delete variable.

Note that if we call delete_many() with an empty query filter document, then all documents in the collection will be deleted.

delete_many() returns a result. In this example, we use the result to print the count of documents that were deleted.

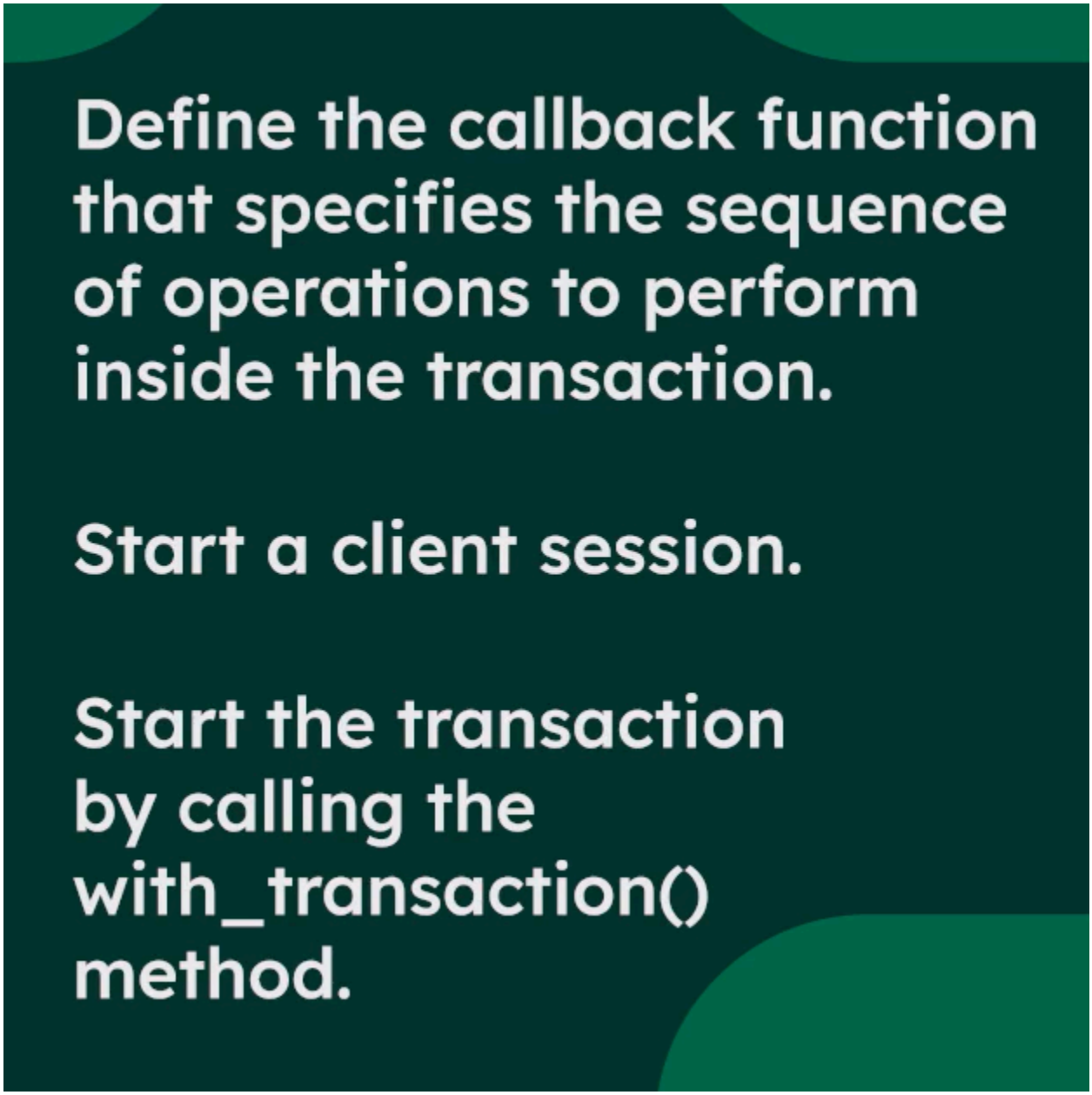
We also query for a sample target document before and after the delete operation as an additional confirmation.

Here's the code:

```
# Get reference to 'bank' database
db = client.bank

# Get a reference to the 'accounts' collection
accounts_collection = db.accounts

# Filter for accounts with balance less than $2000
documents_to_delete = {"balance": {"$lt": 2000}}

# Search for sample document before delete
print("Searching for sample target document before delete: ")
pprint.pprint(accounts_collection.find_one(documents_to_delete))

# Write an expression that deletes the target accounts.
result = accounts_collection.delete_many(documents_to_delete)

# Search for sample document after delete
```

```
print("Searching for sample target document after delete: ")
pprint.pprint(accounts_collection.find_one(documents_to_delete))

print("Documents deleted: " + str(result.deleted_count))

client.close()
###
```

Transaction in Python



Creating MongoDB Transactions in Python Applications

The `with_transaction()` method starts the transaction, runs the callback, then commits or cancels the transaction.

Steps:

Create a Transaction

Note that you must have an active database connection to create a transaction. Once you're connected to a database, complete the following steps:

Define the callback that specifies the sequence of operations to perform inside the transaction. Make sure to do the following:

In the callback, include the required parameter, which is the client session. You can also add additional parameters for your particular transaction if needed. In the following example, there are four keyword parameters specific to this bank transfer: transfer_id, account_id_receiver, account_id_sender, and transfer_amount.

Within the callback function, get references to the collections that the operations will take place on.

Write the transaction operations. Note that you must pass the session to each operation.

Start a client session by calling the start_session method on the client object in a with statement.

Carry out the transaction by calling with_transaction on the session object. with_transaction starts a transaction, runs the callback, and commits (or cancels if there's an error). For this step, consider the following:

with_transaction has one required parameter, which is the callback function that specifies the sequence of operations to perform inside the transaction. In the following example, we passed in the callback_wrapper function to pass the additional arguments to the callback: transfer_id, account_id_receiver, account_id_sender, and transfer_amount.

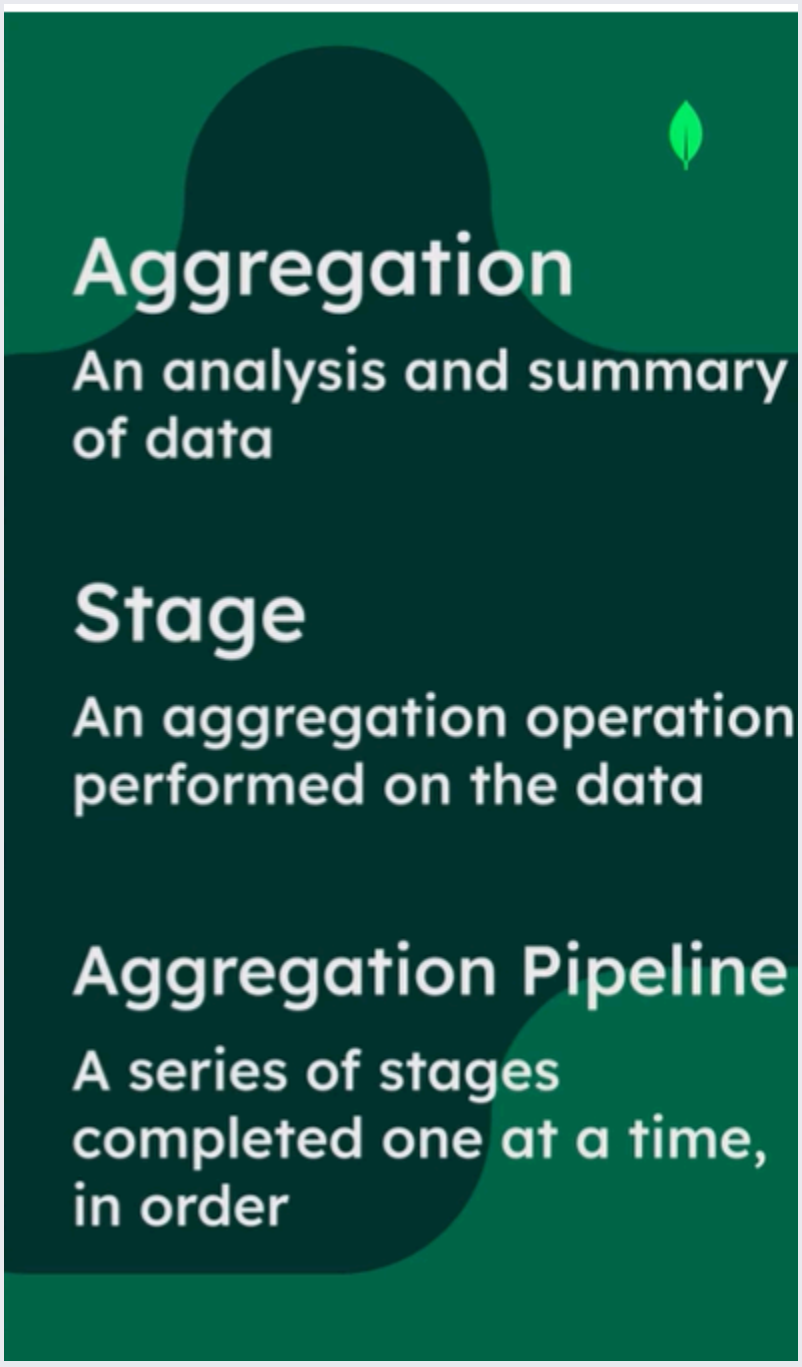
Note that the general best practice for passing arbitrary arguments to the callback is to use a lambda function. In this example, we used the callback wrapper for simplicity.

Here's the code to create the transaction:

```
# Connect to MongoDB cluster with MongoClient
client = MongoClient(MONGODB_URI)

# Step 1: Define the callback that specifies the sequence of operations to perform inside
the transactions.
def callback(
    session,
    transfer_id=None,
    account_id_receiver=None,
    account_id_sender=None,
    transfer_amount=None,
):

    # Get reference to 'accounts' collection
    accounts_collection = session.client.bank.accounts

    # Get reference to 'transfers' collection
    transfers_collection = session.client.bank.transfers

    transfer = {
        "transfer_id": transfer_id,
        "to_account": account_id_receiver,
        "from_account": account_id_sender,
        "amount": {"$numberDecimal": transfer_amount},
    }

    # Transaction operations
    # Important: You must pass the session to each operation

    # Update sender account: subtract transfer amount from balance and add transfer ID
# $inc operator increase/decreases the value
# $push operator is used for manipulating array
    accounts_collection.update_one(
        {"account_id": account_id_sender},
        {
            "$inc": {"balance": -transfer_amount},
            "$push": {"transfers_complete": transfer_id},
        },
```

```
        session=session,
    )

    # Update receiver account: add transfer amount to balance and add transfer ID
    accounts_collection.update_one(
        {"account_id": account_id_receiver},
        {
            "$inc": {"balance": transfer_amount},
            "$push": {"transfers_complete": transfer_id},
        },
        session=session,
    )

    # Add new transfer to 'transfers' collection
    transfers_collection.insert_one(transfer, session=session)

    print("Transaction successful")

    return


def callback_wrapper(s):
    callback(
        s,
        transfer_id="TR218721873",
        account_id_receiver="MDB343652528",
        account_id_sender="MDB574189300",
        transfer_amount=100,
    )


# Step 2: Start a client session
with client.start_session() as session:
    # Step 3: Use with_transaction to start a transaction, execute the callback, and commit
(or cancel on error)
    session.with_transaction(callback_wrapper)


client.close()
####
```

MongoDB Aggregation in Python



Introduction to MongoDB Aggregation

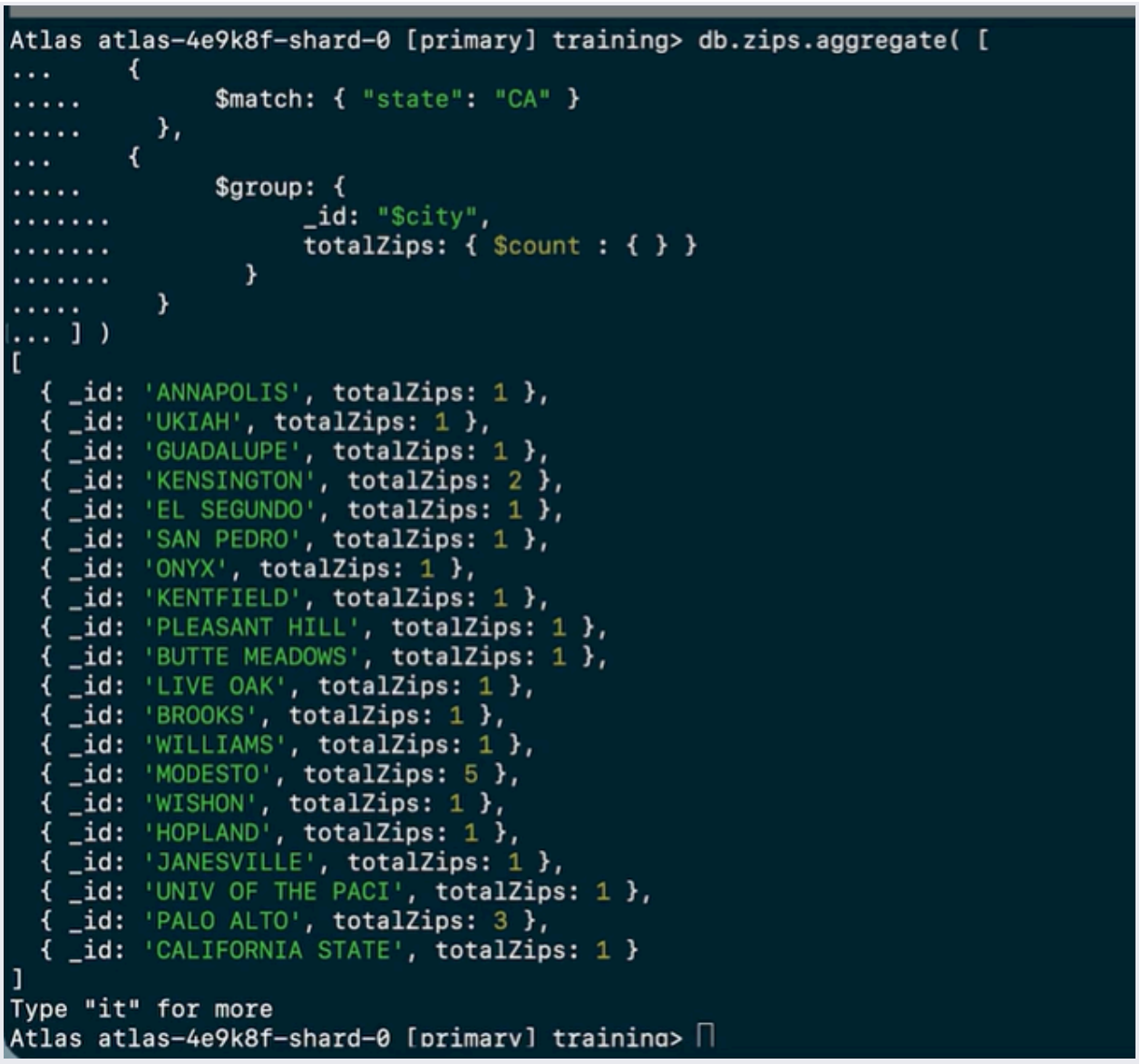
Definitions

- **Aggregation**: Collection and summary of data
- **Stage**: One of the built-in methods that can be completed on the data, but does not permanently alter it
- **Aggregation pipeline**: A series of stages completed on the data in order

Structure of an Aggregation Pipeline

```
db.collection.aggregate([
   {
      $stage1: {
         { expression1 },
         { expression2 }...
      },       $stage2: {
         { expression1 }...
      }
   }
])
```

$match helps in indexing at early stage of pipeline and $group helps to group based on unique id accompanied by accumulator



Using $match and $group Stages in a MongoDB Aggregation Pipeline

$match

The $match stage filters for documents that match specified conditions. Here's the code for $match:

```
{
  $match: {
    "field_name": "value"
  }
}
```

$group

The $group stage groups documents by a group key.

```
{
  $group:
    {
      _id: <expression>, // Group key
      <field>: { <accumulator> : <expression> }
    }
 }
```

$match and $group in an Aggregation Pipeline

The following aggregation pipeline finds the documents with a field named "state" that matches a value "CA" and then groups those documents by the group key "$city" and shows the total number of zip codes in the state of California. totalZips is an accumulator to count.

```
db.zips.aggregate([
{
   $match: {
      state: "CA"
    }
},
{
   $group: {
      _id: "$city",
      totalZips: { $count : { } }
   }
}
])
```

####
You have a collection that contains zip codes in the United States. Here’s a sample document from this collection:

```
_id: ObjectId('5c8eccc1caa187d17ca6eea2')
city: "EVERGREEN"
zip: "36401"
loc: Object
   y: 31.458009
   x: 86.925771
pop: 8556
state: "AL"
```

What will the output of this aggregation pipeline be? (Select one.)

```
db.zips.aggregate([
{
   $match: { "state": "TX" }
},
{
   $group: { "_id": "$city" }
}
])
```

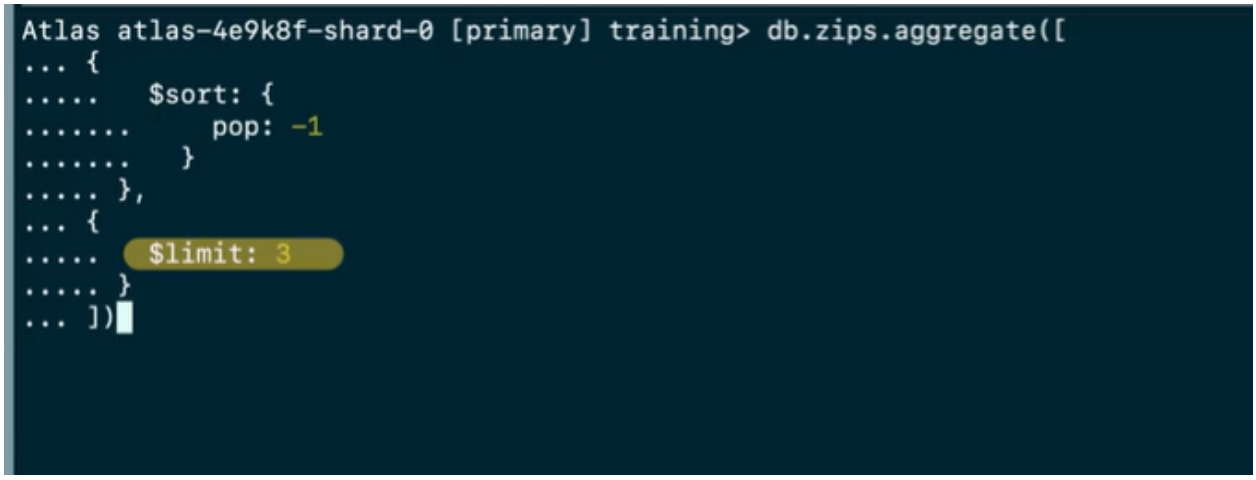
Correct Answer: A. One document for each city located in Texas (TX)Your Answer: Correct
Correct!

First, the $match stage finds all documents where the state is Texas (TX). Next, the $group stage groups documents according to a group key. The output of this stage is one document for each unique value of this group key. In this example, the output is one document for each city because the group key is the city.

###

```
Atlas atlas-d3opcw-shard-0 [primary] bird_data> db.sightings.aggregate([
...   {
...     $match: {
...         species_common: 'Eastern Bluebird'
...     }
...   }, {
...     $group: {
...         _id: '$location.coordinates',
...         number_of_sightings: { $count: {} }
...     }
...   }
... ])
[
  { _id: [ 40, -74 ], number_of_sightings: 3 },
  { _id: [ 41, -74 ], number_of_sightings: 1 },
  { _id: [ 40, -73 ], number_of_sightings: 1 }
]
Atlas atlas-d3opcw-shard-0 [primary] bird_data>
```

Using $sort and $limit Stages in a MongoDB Aggregation Pipeline



$sort
The $sort stage sorts all input documents and returns them to the pipeline in sorted order. We use 1 to represent ascending order, and -1 to represent descending order.

```
{
   $sort: {
      "field_name": 1
   }
}
```

$limit
The $limit stage returns only a specified number of records.

```
{  $limit: 5}
```

$sort and $limit in an Aggregation Pipeline
The following aggregation pipeline sorts the documents in descending order, so the documents with the greatest pop value appear first, and limits the output to only the first five documents after sorting.

```
db.zips.aggregate([
{
  $sort: {
    pop: -1
  }
},
{
  $limit:  5
}
])
```

Using $project, $count, and $set Stages in a MongoDB Aggregation Pipeline

```
Atlas atlas-4e9k8f-shard-0 [primary] training> use training
already on db training
Atlas atlas-4e9k8f-shard-0 [primary] training> db.zips.findOne()
{
  _id: ObjectId("5c8eccc1caa187d17ca6ed2c"),
  city: 'CROPWELL',
  zip: '35054',
  loc: { y: 33.506448, x: 86.280026 },
  pop: 4171,
  state: 'AL'
}
Atlas atlas-4e9k8f-shard-0 [primary] training> db.zips.aggregate([
... { $project: {
.... state:1,
.... zip:1,
..... population:"$pop",
..... _id:0
.....     }
.....  }
... ])
```

```
Atlas atlas-4e9k8f-shard-0 [primary] training> db.zips.aggregate([
...   { $set: {
.....
...  pop_2022: { $round: { $multiply: [1.0031, "$pop"] } }
...       }
...    }
... ]
...
```

$project

The $project stage specifies the fields of the output documents. 1 means that the field should be included, and 0 means that the field should be supressed. The field can also be assigned a new value.

```
{
   $project: {
      state:1,
      zip:1,
```

```
        population:"$pop",
        _id:0
    }
}
```

$set

The $set stage creates new fields or changes the value of existing fields, and then outputs the documents with the new fields.

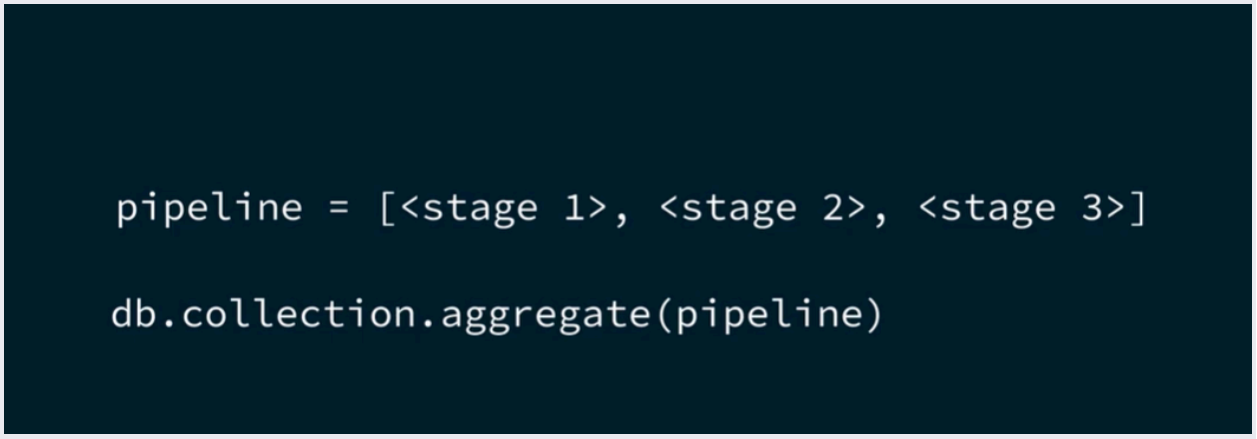
```
{
    $set: {
        place: {
            $concat:["$city",",","$state"]
        },
        pop:10000
     }
  }
```

$count

The $count stage creates a new document, with the number of documents at that stage in the aggregation pipeline assigned to the specified field name.

```
{
  $count: "total_zips"
}
```

```
Atlas atlas-4e9k8f-shard-0 [primary] training> db.zips.aggregate([
...   {
.....     $group: {
.......       _id: "$state",
.......       total_pop: { $sum: "$pop" }
.......     }
.....   },
...   {
.....     $match: {
.......         total_pop:{$lt:1000000}
.......     }
.....   },
...   {
.....     $out: "small_states"
.....
...   }
... ]
... )

Atlas atlas-4e9k8f-shard-0 [primary] training> show collections
accounts
grades
routes
sales
small_routes
small_sales
small_states
zip_codes
zips
Atlas atlas-4e9k8f-shard-0 [primary] training>
```



Using MongoDB Aggregation Stages with Python: $match and $group

`$match` and `$group` stages in a MongoDB aggregation pipeline by using PyMongo.

Using $match

Use the $match operator to select documents that match the specified query condition(s) and pass the matching documents to the next stage. $match takes a document that specifies the query.
$match should be placed early in a pipeline to reduce the number of documents that will be processed later in the pipeline.
Here's an example of the
$match
stage:

```
# Select accounts with balances of less than $1000.
select_by_balance = {"$match": {"balance": {"$lt": 1000}}}
```

Using $group

Use the $group stage to separate documents into groups. The $group stage must have an _id field that specifies the group key. The group key is preceded by a $ and enclosed in quotation marks.
A $group stage can include additional field(s) that are computed by using accumulator operators, such as $avg.
Here's an example of the
$group stage:

```
# Separate documents by account type and calculate the average balance for each account type.
separate_by_account_calculate_avg_balance = {
    "$group": {"_id": "$account_type", "avg_balance": {"$avg": "$balance"}}
}
```

Aggregation Example That Uses $match and $group

```
# Connect to MongoDB cluster with MongoClient
client = MongoClient(MONGODB_URI)

# Get reference to 'bank' database
db = client.bank

# Get reference to 'accounts' collection
accounts_collection = db.accounts

# Calculate the average balance of checking and savings accounts with balances of less than $1000.

# Select accounts with balances of less than $1000.
select_by_balance = {"$match": {"balance": {"$lt": 1000}}}

# Separate documents by account type and calculate the average balance for each account type.
separate_by_account_calculate_avg_balance = {
    "$group": {"_id": "$account_type", "avg_balance": {"$avg": "$balance"}}
}

# Create an aggegation pipeline using 'stage_match_balance' and 'stage_group_account_type'.
pipeline = [
    select_by_balance,
    separate_by_account_calculate_avg_balance,
]

# Perform an aggregation on 'pipeline'.
results = accounts_collection.aggregate(pipeline)

print()
print(
    "Average balance of checking and savings accounts with balances of less than $1000:", "\n"
)

for item in results:
    pprint.pprint(item)

client.close()
```

Using MongoDB Aggregation Stages with Python: $sort and $project

Using $sort

When we build queries by using the aggregation framework, each stage transforms or organizes data in a specific way. In this lesson, we focused on the $sort and $project stages.

Use the $sort operator to organize the input documents in **ascending or descending order**. $sort takes a document that specifies the field(s) to sort by and the respective sort order. To sort in ascending order, use the value of 1. For descending order, use the value of -1.

Here's an example of a $sort stage:

```
# Organize documents in order from highest balance to lowest.
organize_by_original_balance = {"$sort": {"balance": -1}}
```

Using $project

Use the $project stage to specify the fields returned by the aggregation. $project can be used to include or exclude existing fields by setting a field to 1 to include or 0 to exclude. It can also be used to add new fields or reset the value of existing fields.

To add a new field by using $project, specify the field name and set its value to an expression like this: <field>: <expression>. In this example, the new field name is gbp_balance. The expression contains the $divide arithmetic operator, the $balance field reference, and the conversion_rate_usd_to_gbp variable.

When creating an aggregation pipeline, the $project stage should usually be the last stage in a pipeline because it specifies the exact fields to be returned to the client.

Here's an example of a $project stage:

```
# Return only the account type & balance fields, plus a new field containing balance in Great British Pounds (GBP).
return_specified_fields = {
    "$project": {
        "account_type": 1,
        "balance": 1,
        "gbp_balance": {"$divide": ["$balance", conversion_rate_usd_to_gbp]},
        "_id": 0,
    }
}
###
```

Aggregation Example That Uses $match, $sort, and $project

```
# Connect to MongoDB cluster with MongoClient
client = MongoClient(MONGODB_URI)

# Get reference to 'bank' database
db = client.bank

# Get a reference to the 'accounts' collection
accounts_collection = db.accounts

# Return the account type, original balance, and balance converted to Great British Pounds (GBP)
# of all checking accounts with an original balance of greater than $1,500 US dollars, in order from highest
original balance to lowest.

# To calculate the balance in GBP, divide the original balance by the conversion rate
conversion_rate_usd_to_gbp = 1.3

# Select checking accounts with balances of more than $1,500.
select_accounts = {"$match": {"account_type": "checking", "balance": {"$gt": 1500}}}

# Organize documents in order from highest balance to lowest.
organize_by_original_balance = {"$sort": {"balance": -1}}

# Return only the account type & balance fields, plus a new field containing balance in Great British
Pounds (GBP).
return_specified_fields = {
    "$project": {
        "account_type": 1,
        "balance": 1,
        "gbp_balance": {"$divide": ["$balance", conversion_rate_usd_to_gbp]},
        "_id": 0,
    }
}

# Create an aggegation pipeline containing the four stages created above
pipeline = [
    select_accounts,
    organize_by_original_balance,
    return_specified_fields,
]

# Perform an aggregation on 'pipeline'.
results = accounts_collection.aggregate(pipeline)
```

```
print(
    "Account type, original balance and balance in GDP of checking accounts with original balance greater
than $1,500,"
    "in order from highest original balance to lowest: ", "\n"
)

for item in results:
    pprint.pprint(item)

client.close()


#########
```