

mongoDB

Bộ môn: Kỹ Thuật Phần Mềm

Giáo viên: Trần Thế Trung.

Email: tranthetrong@iuh.edu.vn

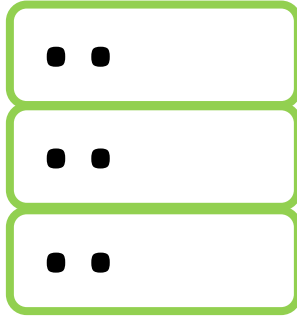


2- MongoDB: Import, Export, Querying data

1. Giới thiệu về định dạng data của mongodb khi lưu trữ trong hệ thống và hiển thị.
2. Giới thiệu về **JSON**.
3. Giới thiệu về **BSON**.
4. Import/Export data (*trên Atlas và mongo shell*).
5. Tương tác dữ liệu trên Atlas.
6. Truy vấn dữ liệu với mongo shell.

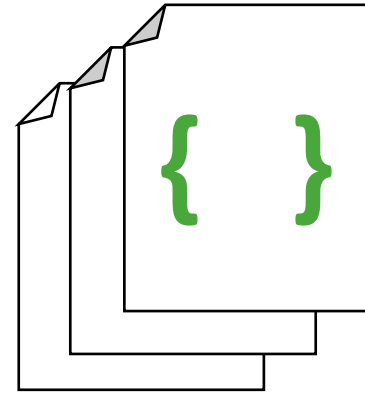


1. How Does MongoDB Store Data?



Presentation

How are documents presented in memory?



Correct syntax

What is the correct syntax for documents?

2. JSON (*Javascript Standard Object Notation*)

- When you view or update documents in MongoDB Shell → you are working with JSON
- JSON format:
 - Start and end with `{ }`
 - Separate each **key** and **value** with colon `:`
 - Separate each **key:value** pair with comma `,`
 - **Key** must be surrounded by quotation mark `" "`
 - In MongoDB **"keys"** also called **"fields"**.

```
{
  "_id": 1,
  "name" : { "first" : "John", "last" : "Backus" },
  "contribs" : [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ],
  "awards" : [
    {
      "award" : "W.W. McDowell Award",
      "year" : 1967,
      "by" : "IEEE Computer Society"
    }, {
      "award" : "Draper Prize",
      "year" : 1993,
      "by" : "National Academy of Engineering"
    }
  ]
}
```

JSON Values

Numbers: no quotes.

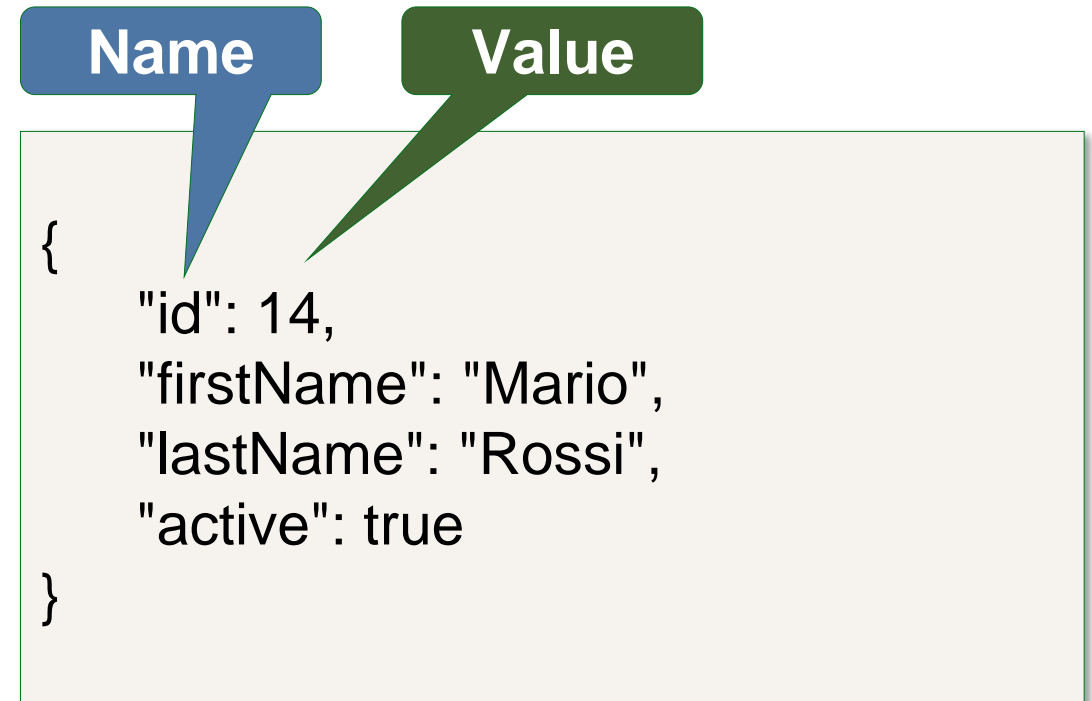
String: in double quotes.

Boolean: true, false.

Nested JSON object.


Array.

Null.



Nested JSON Objects

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "address" : {
    "street" : "100 Main St",
    "city" : "Philadelphia",
    "state" : "Pennsylvania",
    "zip" : "19103",
    "country" : "USA"
  }
}
```



JSON Arrays

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "languages" : ["Java", "C#", "Python", "Javascript"]
}
```

Pros of JSON

Friendly

Readable

Familiar

Cons of JSON

Text-based

Space-Consuming

Limited of data types

3. **BSON** (*Binary JSON*)

- Bridges the gap between binary representation and JSON format
- A binary representation to store data in JSON format
- Optimized for:
 - Speed
 - Space
 - Flexibility
- To achieve high performance

```
_id[0a2<E0<00
saleDate"0uHLitems00mnameprinter
papertags%0office1stationaryprice0
<0quantity1rnamenotepadtags00office
1writing2schoolprice0
<0quantity20namepenstagsB0writing1o
ffice2school3stationaryprice0<0qua
ntity3pname
backpacktags-0school1travel2kidspri
ce[<0quantity4rnamenotepadtags00off
ice1writing2schoolprice7<0quantity5
xname
envelopestags40
stationary1office2generalprice0<0qu
antity6xname
```


JSON

Encoding

UTF-8 String

Data Support

String, Boolean,
Number, Array

Readability

Human and Machine

BSON

Encoding

Binary

Data Support

String, Boolean, Number (Integer, long, float..),
Array, Date, Raw Binary

Readability

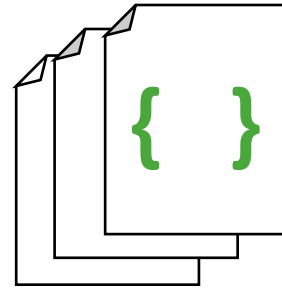
Machine only

Summary



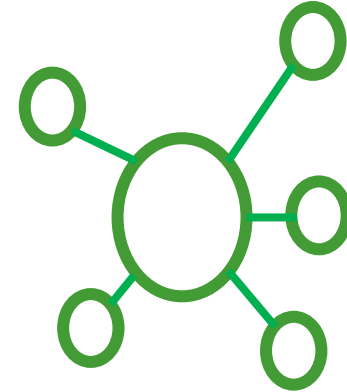
BSON

MongoDB stores data in BSON, internally and over the network



JSON

Can be natively stored and retrieved in MongoDB



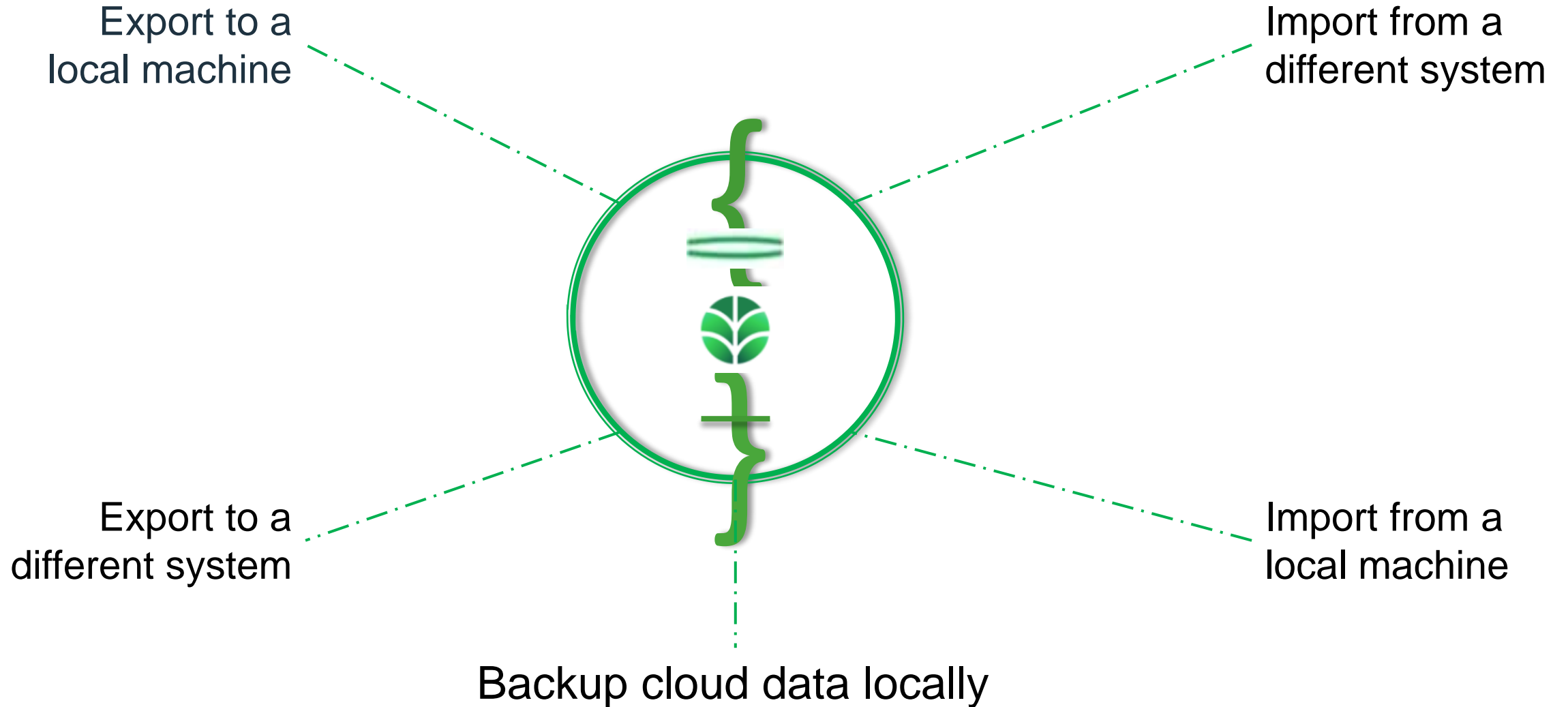
Additional Features

BSON , provides additional features like speed and flexibility

4. Import/Export Data



Interacting with Atlas Cluster



Interacting with Atlas Cluster

Data is stored in **BSON** but viewed in **JSON** → Which format we're going to use? (export)

JSON

mongoexport

mongoimport

BSON

mongodump

mongorestore

Export (BSON)

Syntax: **mongodump** <options>

1. From localhost:

```
mongodump -d dbtest -c restaurants -o backup
```

connect to a local MongoDB instance running on port 27017 and use the default settings to export the content (no parameters with all databases and collections)

2. From Atlas:

```
mongodump --uri  
mongodb+srv://mongobasic:pass@cluster0.msr5i.mongodb.net/sample_restaurants  
-c restaurants -o backup
```

creates a dump file that contains only the collection named restaurants of sample_restaurants database

Export (BSON)

mongodump --help

<options>:

- d *database to use,*
- c *collection to use,*
- u *username for authentication,*
- p *password for authentication,*
- o *output directory,*
- host *mongodb host to connect,*
- port *server port (can also use -h hostname:port),*
- ...

Import (BSON)

Syntax: **mongorestore** <options> <directory or file to restore>

1. From localhost:

mongorestore -d backup -c restaurants backup/dbtest/restaurants.bson

restores the collection named restaurants in the database backup from the corresponding files located in the backup/dbtest/restaurants.bson

2. From Atlas:

mongorestore --uri mongodb+srv://mongobasic:pass@cluster0.msr5i.mongodb.net -d backup -c restaurants backup/dbtest/restaurants.bson

restore from a backup/dbtest/ directory to MongoDB Atlas Cluster.

Export (JSON)

Syntax: **mongoexport** <options>

1. From localhost:

```
mongoexport -d dbtest -c restaurants -o res.json
```

export the restaurants collection of dbtest db to the res.json output file from a local MongoDB instance running on port 27017.

2. From Atlas:

```
mongoexport --uri  
mongodb+srv://mongobasic:pass@cluster0.msr5i.mongodb.net/sample_restaurants  
-c restaurants -o res.json
```

connect to a MongoDB Atlas Cluster and export the restaurants collection of sample_restaurant db to res.json output file.

Import (JSON)

Syntax: **mongoimport** <options> <file to import>

1. From localhost:

Mongoimport --drop -d backup -c restaurants res.json

Mongoimport -h localhost:27017 --drop -d backup -c restaurants res.json

import the json data from the res.json file into the collection restaurants in the backup db to a local mongod instance running on port 27017.

--drop: before restoring the collections from the dumped backup, drops the collections from the target database. does not drop collections that are not in the backup.

2. From Atlas:

mongoimport --uri mongodb+srv://mongobasic:pass@cluster0.msr5i.mongodb.net/backup
--drop -c restaurants res.json

Import from the res.json file to MongoDB Atlas Cluster.

5- Data Explorer



Data Interaction using the Atlas UI

The screenshot displays the MongoDB Atlas interface for a project named 'ABC'. The left sidebar shows navigation options under 'DEPLOYMENT' (Databases, Triggers, Data Lake) and 'SECURITY' (Database Access, Network Access, Advanced). The main content area is titled 'Database Deployments' for the 'MONGO_BASIC' deployment. A search bar is present. Below the search bar, there are four buttons: 'Sandbox', 'Connect', 'View Monitoring', and 'Browse Collections'. The 'Browse Collections' button is highlighted with a green box and a red hand icon. Below these buttons, there are four monitoring charts for the last 6 hours:

- Reads (R):** 0 reads, 0 writes. Last 6 hours: 0.008/s.
- Connections:** 0 connections. Last 6 hours: 9.0.
- I/O:** In: 0.0 B/s, Out: 0.0 B/s. Last 6 hours: 18.3 KB/s.
- Data Size:** 326.5 MB. Last 10 days: 512.0 MB.

Data Interaction using the Atlas UI

The screenshot displays the MongoDB Atlas interface. At the top, the navigation bar includes 'mongo_basic', 'Atlas', 'Realm', and 'Charts'. Below this, a secondary navigation bar lists various tools: 'Overview', 'Real Time', 'Metrics', 'Collections' (which is highlighted with a green underline), 'Search', 'Profiler', 'Performance Advisor', 'Online Archive', and 'Command Line Tools'. On the left side, a sidebar shows 'DATABASES: 8' and 'COLLECTIONS: 21'. Below this, there is a '+ Create Database' button and a 'NAMESPACES' section. The namespaces list includes 'sample_airbnb', 'sample_analytics', 'sample_geospatial', 'sample_mflix', 'sample_restaurants', 'sample_supplies', 'sample_training' (highlighted in green), and 'sample_weatherdata'. A red hand icon points to 'sample_training'. The main content area is titled 'sample_training.zips' and shows 'COLLECTION SIZE: 3.13MB', 'TOTAL DOCUMENTS: 29470', and 'INDEXES TOTAL SIZE: 348KB'. Below this, there are tabs for 'Find', 'Indexes', 'Schema Anti-Patterns 0', 'Aggregation', and 'Search Indexes'. An 'INSERT DOCUMENT' button is located on the right. A 'FILTER' input field contains the JSON query: `{"filter": "example"}`. To the right of the filter are 'Find' and 'Reset' buttons. Below the filter, it says 'QUERY RESULTS 1-20 OF MANY'. A sample document is displayed in a light gray box:

```
{
  "_id": ObjectId("5c8eccc1caa187d17ca6ed1c"),
  "city": "BRIERFIELD",
  "zip": "35035",
  "loc": Object,
  "pop": 1282,
  "state": "AL"
}
```

Data Interaction using the Atlas UI

ABC

Access Manager

Billing

All Clusters

Get Help

Tri

mongo_basic

Atlas

Realm

Charts

ABC > MONGO_BASIC > DATABASES

Sandbox

VERSION
4.4.7

REGION
AWS Singapore (ap-southeast-1)

Overview

Real Time

Metrics

Collections

Search

Profiler

Performance Advisor

Online Archive

Command Line Tools

DATABASES: 8 COLLECTIONS: 21

VISUALIZE YOUR DATA

REFRESH

+ Create Database

NAMESPACES

sample_airbnb

sample_analytics

accounts

customers

transactions

sample_geospatial

Feature Requests

sample_mflix

sample_analytics

DATABASE SIZE: 15.79MB INDEX SIZE: 144KB TOTAL COLLECTIONS: 3

CREATE COLLECTION

Collection Name	Documents	Documents Size	Documents Avg	Indexes	Index Size	Index Avg
accounts	1746	218KB	128B	1	68KB	68KB
customers	500	191.22KB	392B	1	20KB	20KB
transactions	1746	15.39MB	9.03KB	1	56KB	56KB

Data Interaction using the Atlas UI

The screenshot displays the MongoDB Atlas web interface. At the top, there's a navigation bar with 'ABC', 'Access Manager', 'Billing', 'All Clusters', 'Get Help', and a user menu. Below this, a secondary bar shows 'mongo_basic', 'Atlas' (selected), 'Realm', and 'Charts'. The left sidebar contains a '+ Create Database' button and a 'NAMESPACES' search bar. Underneath, a tree view lists namespaces: 'sample_airbnb', 'sample_analytics' (expanded), 'accounts', 'customers' (highlighted with a green box), and 'transactions'. Other namespaces like 'sample_geospatial', 'sample_mflix', 'sample_restaurants', 'sample_supplies', 'sample_training', and 'sample_weatherdata' are also visible. The main content area has tabs for 'Find' (active), 'Indexes', 'Schema Anti-Patterns 0', 'Aggregation', and 'Search Indexes'. Above the 'Find' tab, a box shows 'sample_analytics.customers' with statistics: 'COLLECTION SIZE: 191.22KB', 'TOTAL DOCUMENTS: 500', and 'INDEXES TOTAL SIZE: 20KB'. Below the tabs, there's a 'FILTER' input with the text '{"filter": "example"}' and buttons for 'Find' and 'Reset'. The 'Find' button is highlighted with a green box. Below the filter, it says 'QUERY RESULTS 1-20 OF MANY'. A document is displayed in a box, with a green arrow pointing from the 'customers' namespace in the sidebar to this document. The document fields are: '_id: ObjectId("5ca4bbcea2dd94ee58162a6f")', 'username: "andrewhamilton"', 'name: "Gary Nichols"', 'address: "633 Miller Turnpike Jonathanland, OR 62874"', 'birthdate: 1993-02-25T17:37:29.000+00:00', 'email: "laura34@yahoo.com"', '> accounts: Array', and '> tier_and_details: Object'. To the right of the document, there are icons for 'Edit', 'Copy', 'Duplicate', and 'Delete', with the text 'Edit Copy Duplicate Delete' overlaid in red.

Data Interaction using the Atlas UI

The screenshot displays the MongoDB Atlas web interface. The top navigation bar includes the Atlas logo, a menu icon, and tabs for 'Realm' and 'Charts'. On the right, there are icons for user management, document management, and notifications. The left sidebar shows a list of collections: 'sample_analytics', 'sample_geospatial', 'sample_mflix', 'sample_restaurants', 'sample_supplies', 'sample_training', 'sample_weatherdata', and a collapsed 'test' collection. Under 'test', there are sub-items 'bios', 'inventory', and 'products', which is currently selected and highlighted with a green bar. The main panel shows the 'products' collection with a filter bar containing the text 'FILTER {"filter": "example"}'. To the right of the filter bar are 'Find' and 'Reset' buttons. Above the 'Find' button is an 'INSERT DOCUMENT' button, which is highlighted with a green box and a red hand icon. Below the filter bar, the query results are displayed, showing two documents. The first document has fields: '_id: 103', 'sku: "xyz789"', and 'description: "Multiple line description"'. The second document has fields: '_id: 102', 'sku: "xyz456"', and 'description: "There are Many spaces before line"'.

Atlas

Realm

Charts

sample_analytics

sample_geospatial

sample_mflix

sample_restaurants

sample_supplies

sample_training

sample_weatherdata

test

bios

inventory

products

INSERT DOCUMENT

FILTER {"filter": "example"}

Find

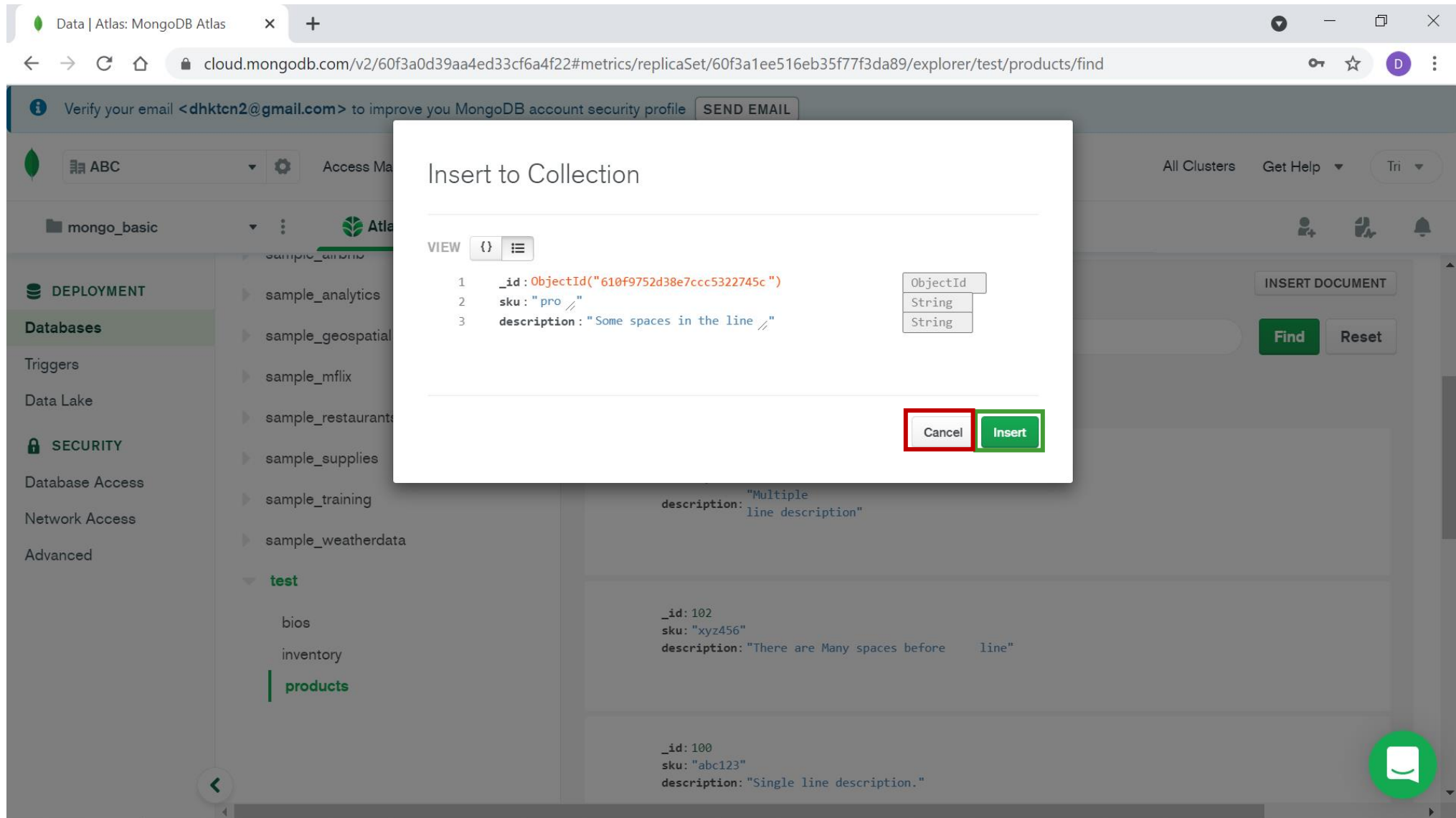
Reset

QUERY RESULTS 1-7 OF 7

```
_id: 103
sku: "xyz789"
description: "Multiple
line description"
```

```
_id: 102
sku: "xyz456"
description: "There are Many spaces before line"
```


Data Interaction using the Atlas UI



Data Interaction using the Atlas UI

The screenshot displays the MongoDB Atlas interface. On the left sidebar, the 'sample_training' collection is selected, and the 'zips' field is highlighted. The main panel shows a query filter: `{ "state": "NY" }`. Below the filter, the query results are displayed, showing three documents. Each document contains fields for `_id`, `city`, `zip`, `loc`, `pop`, and `state`. Red hand-drawn arrows indicate the flow of interaction: one arrow points from the 'zips' field in the sidebar to the filter box, and three arrows point from the 'zip' field in each of the three result documents to the filter box. The 'Find' button is also highlighted with a red hand-drawn arrow.

Atlas Realm Charts

sample_mflix

sample_restaurants

sample_training

companies

grades

inspections

routes

trips

zips

sample_weatherdata

INSERT DOCUMENT

FILTER `{ "state": "NY" }`

Find Reset

QUERY RESULTS 1-20 OF MANY

`_id: ObjectId("5c8eccc1caa187d17ca72f89")`
`city: "FISHERS ISLAND"`
`zip: "06390"`
`> loc: Object`
`pop: 329`
`state: "NY"`

`_id: ObjectId("5c8eccc1caa187d17ca72f8b")`
`city: "NEW YORK"`
`zip: "10003"`
`> loc: Object`
`pop: 51224`
`state: "NY"`

`_id: ObjectId("5c8eccc1caa187d17ca72f8c")`
`city: "GOVERNORS ISLAND"`
`zip: "10004"`
`> loc: Object`
`pop: 3593`
`state: "NY"`

Data Interaction using the Atlas UI

The screenshot displays the MongoDB Atlas interface. On the left sidebar, the 'zips' collection is selected under the 'sample_training' dataset. The main panel shows a query filter: `{ "state": "NY", "city": "ALBANY" }`. Below the filter, the query results are displayed, showing 1-7 of 7 results. A red hand icon points to the 'Find' button. The results are as follows:

Result
<pre>_id: ObjectId("5c8eccc1caa187d17ca731d0") city: "ALBANY" zip: "12204" > loc: Object pop: 6927 state: "NY"</pre>
<pre>_id: ObjectId("5c8eccc1caa187d17ca731d7") city: "ALBANY" zip: "12209" > loc: Object pop: 10008 state: "NY"</pre>
<pre>_id: ObjectId("5c8eccc1caa187d17ca731d5") city: "ALBANY" zip: "12207" > loc: Object pop: 2709 state: "NY"</pre>

6- MongoDB Shell (**mongosh**) **Query** Documents



MongoDB Shell (**mongosh**)

- A fully functional JavaScript and Node.js *(support editing and running scripts)*.
- You can use the MongoDB Shell to queries and operations directly with your database.

Connect to a Deployment:

- Connect to local MongoDB instance on default port:
 - **mongosh**
- Connect to local MongoDB instance on non default port:
 - **mongosh --port 28015** or **mongosh --host localhost:28015**
- Connect mongoDB instance on a remote host:
 - **mongosh "mongodb://mongodb0.example.com:28015"**
- Connecting to Atlas:
 - **mongosh "mongodb+srv://cluster0.msr5i.mongodb.net/myFirstDatabase" -u mongobasic**

Mongosh Usage

- List the databases available to the user:
 - `show dbs`
- Switch/Create databases:
 - `use db_name`
- List of all collections for current database:
 - `show collections`

Find Method

- **find()** selects documents in a collection or view and returns a cursor to the selected documents

Syntax: `db.collectionname.find(query, projection)`

Parameter	Type	Description
query	document	Optional. Specifies selection filter using <u>query operators</u> . To return all documents in a collection, omit this parameter or pass an empty document ({}).
projection	document	Optional. Specifies the fields to return in the documents that match the query filter. To return all fields in the matching documents, omit this parameter

Find Examples

- The examples in this section use documents from the [bios collection](#) where the documents generally have the form:

```
{
  "_id" : value,
  "name" : { "first" : string, "last" : string },           // embedded document
  "birth" : ISODate,
  "death" : ISODate,
  "contribs" : [ string, ... ],                             // Array of Strings
  "awards" : [
    { "award" : string, "year": number, "by": string }     // Array of embedded
    ...                                                     documents
  ]
}
```


Find Examples

- Find all documents in a collection:
 - `db.bios.find()`
- Find all documents in the bios collection where `_id` **equals** 5:
 - `db.bios.find({ _id: 5 })`
- Find all documents in the bios collection where the field last in the name embedded document **equals** "Hopper":
 - `db.bios.find({ "name.last": "Hopper" })`
to access fields in an embedded document, use dot notation
- To return all documents in the bios collection where the embedded document name is **exactly** { first: "Yukihiro", last: "Matsumoto" } including the order:
 - `db.bios.find({ name: { first: "Yukihiro", last: "Matsumoto" } })`

Find Examples

- To return all documents in the bios collection where the embedded document name **contains** a field first with the value "Yukihiko" and a field last with the value "Matsumoto":

- `db.bios.find({ "name.first" : "Yukihiko", "name.last" : "Matsumoto" })`

The query would match documents with name fields that held either of the following values:

`{ first: "Yukihiko", aka: "Matz", last: "Matsumoto" }`

`{ last: "Matsumoto", first: "Yukihiko" }`

- To return documents in the bios collection where the **array field** contains the element "UNIX":
 - `db.bios.find({ "contribs" : "UNIX" })`

Find Examples

- To query for all documents where the field tags value is an array with exactly two elements, "A" and "B", in the specified order:
 - `db.inventory.find({ tags: ["A", "B"] })` *//match an array*
- To find an array that contains both the elements "A" and "B", without regard to order or other elements in the array, use the \$all operator
 - `db.inventory.find({ tags: { $all: ["A", "B"] } })`
- To returns documents in the bios collection where the awards array **contains an element** with award field **equals** "Turing Award":
 - `db.bios.find({ "awards.award": "Turing Award" })`

Query Operators

Name	Syntax	Description and Example
\$eq	{ field: { \$eq: value } }	db.inventory.find({ qty: { \$eq: 20 } }) <i>Matches values that are equal to a specified value.</i>
\$gt	{ field: { \$gt: value } }	db.inventory.find({ qty: { \$gt: 20 } }) <i>Matches values that are greater than a specified value.</i>
\$gte	{ field: { \$gte: value } }	db.inventory.find({ qty: { \$gte: 20 } }) <i>Matches values that are greater than or equal to a specified value.</i>
\$in	{ field: { \$in: [value1, value2, ... valueN] } }	db.inventory.find({ qty: { \$in: [5, 15] } }) db.bios.find({ contribs: { \$in: ["ALGOL", "Lisp"] } }) <i>Matches any of the values specified in an array.</i>
\$lt	{ field: { \$lt: value } }	db.inventory.find({ qty: { \$lt: 20 } }) <i>Matches values that are less than a specified value.</i>
\$lte	{ field: { \$lte: value } }	db.inventory.find({ qty: { \$lte: 20 } }) <i>Matches values that are less than or equal to a specified value.</i>

Query Operators

Name	Syntax	Description and Example
\$ne	{field: { \$ne: value } }	db.inventory.find({ qty: { \$ne: 20 } }) <i>Matches all values that are not equal to a specified value.</i>
\$nin	{field: { \$nin: [value1, value2, ... valueN] } }	db.inventory.find({ qty: { \$nin: [5, 15] } }) <i>Matches none of the values specified in an array.</i>
\$and	{ \$and: [{ exp1 }, { exp2 }, ..., { expN }] }	db.inventory.find({ \$and: [{ price: { \$ne: 1.99 } }, { price: { \$exists: true } }] }) <i>implicit AND operation</i> - db.inventory.find({ price: { \$ne: 1.99, \$exists: true } }) <i>Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.</i>
\$or	{ \$or: [{ exp1 }, { exp2 }, ... , { expN }] }	db.inventory.find({ \$or: [{ quantity: { \$lt: 20 } }, { price: 10 }] }) <i>Joins query clauses with a logical OR returns all documents that match the conditions of either clause.</i>

Query Operators

Name	Syntax	Description and Example
\$not	{ field: { \$not: { operator-expression } } }	db.inventory.find({ price: { \$not: { \$gt: 1.99 } } }) <i>Inverts the effect of a query expression and returns documents that do not match the query expression.</i>
\$nor	{ \$nor: [{ exp1 }, { exp2 }, ... { expN }] }	db.inventory.find({ \$nor: [{ price: 1.99 }, { qty: 2 }] }) <i>Joins query clauses with a logical NOR returns all documents that fail to match both clauses.</i>
\$exists	{ field: { \$exists: <boolean> } }	db.inventory.find({ qty: { \$exists: true, \$nin: [5, 15] } }) <i>When <boolean> is true, \$exists matches the documents that contain the field, including documents where the field value is null</i>

Query Operators

Name	Syntax	Description and Example
\$regex	<pre>{ field: { \$regex: /pattern/<options> } }</pre> or <pre>{ field: { \$regex: /pattern/, \$options: '<options>' } }</pre> or <pre>{ field: { \$regex: 'pattern', \$options: '<options>' } }</pre>	<p>- matches all documents where the sku field is like "%789":</p> <pre>db.inventory.find({ sku: { \$regex: /789\$/ } })</pre> <p>- use the i option to perform a case-insensitive match for documents with sku value that starts with ABC</p> <pre>db.inventory.find({ sku: { \$regex: /^ABC/i } })</pre> <p>- use the m option to match lines starting with the letter S for multiline strings:</p> <pre>db.inventory.find({ description: { \$regex: /^S/m } })</pre> <p>- Use the s option to allow the dot character (i.e. .) to match all characters including new line as well as the i option to perform a case-insensitive match:</p> <pre>db.inventory.find({ description: { \$regex: /m.*line/, \$options: 'si' } })</pre>

Query Operators

Name	Syntax	Description and Example
\$all	{ field: { \$all: [value1, value2 ...] } }	- To find an array that contains both the elements “A” and “B”, without regard to order or other elements in the array: db.inventory.find({ tags: { \$all: [“A”, “B”] } })
\$size	{ field: { \$size: number } }	db.inventory.find({ tags: { \$size: 3 } }) Selects documents if the array field is a specified size.

Query Operators

Name	Syntax	Description and Example
\$elemMatch	<code>{ field: { \$elemMatch: {exp1, exp2, ..., expN} } }</code>	<p>- . The following <code>find()</code> operation queries for all documents where the value of the <code>zipcode</code> field is 63109. The <code>\$elemMatch</code> projection returns only the first matching element of the <code>students</code> array where the <code>school</code> field has a value of 102:</p> <pre>db.schools.find({ zipcode: "63109" }, { students: { \$elemMatch: { school: 102 } } })</pre> <p>- and the <code>age</code> field is greater than 10</p> <pre>db.schools.find({ zipcode: "63109" }, { students: { \$elemMatch: { school: 102, age: { \$gt: 10 } } } })</pre> <p>Selects documents if element in the array field matches all the specified <code>\$elemMatch</code> conditions.</p>

count()

- Counts the number of documents referenced by a cursor. Append the count() method to a find() query to return the number of matching documents.
- Example:
 - db.restaurants.find({ 'address.zipcode': '11369'}).count() *// the result is 5*
 - or
 - db.restaurants.count({'address.zipcode': '11369'}) *// the result is 5*

limit()

- Use limit() to maximize performance and prevent MongoDB from returning more results than required for processing.
- Example:
 - db.restaurants.find({ 'address.zipcode': '11369'}).limit(3) *// the result is 3*

skip()

- The skip() method controls the starting point of the results set.

- Example:

- `db.restaurants.find({ 'address.zipcode': '11369' }).skip(1)` *// the result is 4*

sort()

- The sort() method orders the documents in the result set

- Example:

- `db.restaurants.find({ 'address.zipcode': '11369' }).sort({name: 1})`
- `db.restaurants.find({ 'address.zipcode': '11369' }, {name: 1}).sort({name: 1})`

1: ascending
-1: descending

projection

Projection

- The projection parameter determines which fields are returned in the matching documents. The projection parameter takes a document of the following form:

`{ field1: value, field2: value, ... }`

`field: <1 or true>` Specifies the inclusion of a field. Non-zero integers are also treated as true.

`field: <0 or false>` Specifies the exclusion of a field.

*You **cannot mix zeros and ones** in a single projection. **you can mix ones and zeros** is when you're specifically asking to exclude the `_id` field*

- Example:

- `db.restaurants.find({ 'address.zipcode': '11369', {name: 1}`)
- `db.restaurants.find({ 'address.zipcode': '11369', {_id: 0, _id: 1, name: 1}`)
- `db.restaurants.find({ 'address.zipcode': '11369', {_id: 1, _id: 0, name: 1}`)
- `db.restaurants.find({ 'address.zipcode': '11369', {name: 1, name: 0}`)

Question?

