DBS101 Database Systems Fundamentals

Lesson 14

Royal University of Bhutan

Learning Outcomes

1. Evaluate data modeling in document databases.
2. Apply CRUD operations on MongoDB database.
3. Differentiate between Embedding and Referencing.
4. Create databases in MongoDB Atlas.

Data Modeling in Document Databases(MongoDB)

- One format to model and query data.

"Data that is accessed together should be stored together"

Data Modeling in Document Databases(MongoDB)

Documents

- Displayed in JSON.
- Stored as BJSON(Binary JSON): An extension of JSON that provides additional features like support for JSON data types, date,numbers, ObjectID, etc.

Data Modeling in Document Databases(MongoDB)

Documents

- ObjectID
  - Special data type in mongodb, it is used to create unique identifiers.
  - Every document requires an “_id” field, which acts like a primary key.
  - If a inserted document does not contain a “_id” field Mongodb automatically generates a ObjectID for the “_id” field.

Data Modeling in Document Databases(MongoDB)

Cardinalities

1. One to One: One to one relation between attributes in a document.

Eg: **{"_id: 2,**

"title": "Pride and Prejudice",

"author": "Jane Austen"}

Field "title" is tied to one "author"

Data Modeling in Document Databases(MongoDB)

Cardinalities

2. One to Many: One to many relation between attributes in a document.

Data Modeling in Document Databases(MongoDB)

Eg: {“_id: 2,

“title”: “DUNE 2”,

“Cast”:[

{“actor”: “Zendaya”},

{“actor”: “Timothee”}]

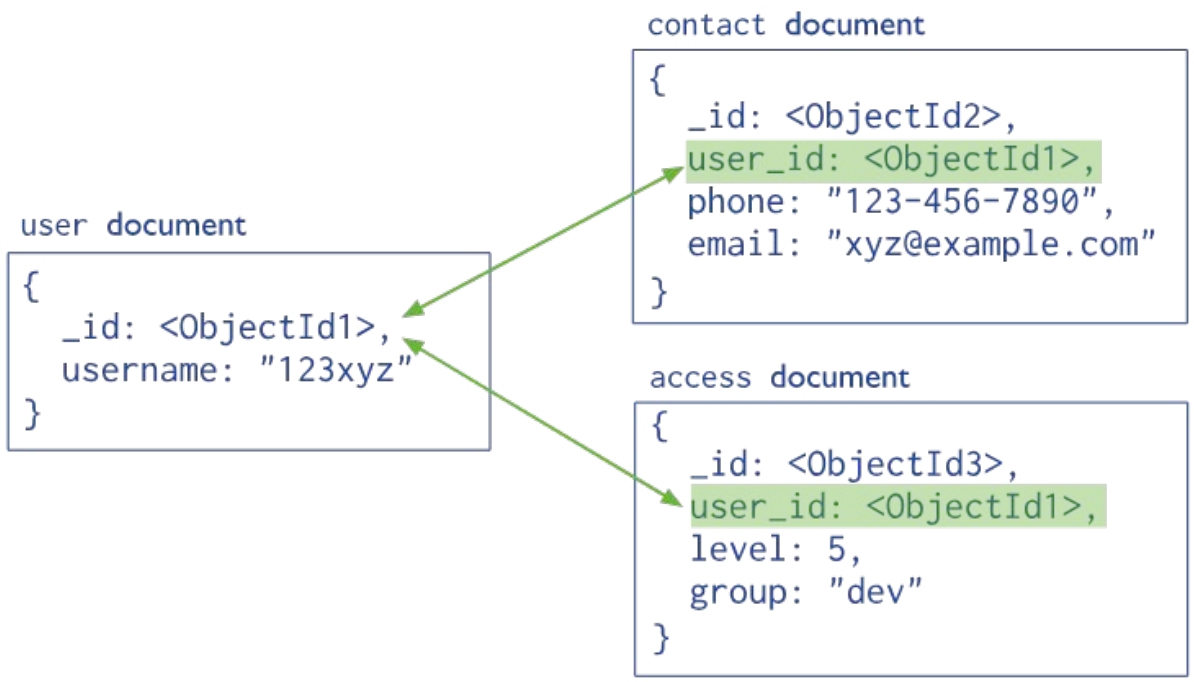
}

Field “title” is tied to one more than one “actors”
- Nested Arrays are a good way to represent one to many relationships.

Data Modeling in Document Databases(MongoDB)

Cardinalities

3. Many to Many: Documents referring other documents.



Data Modeling in Document Databases(MongoDB)

user document

```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

contact document

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

access document

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Data Modeling in Document Databases(MongoDB)

Embedding

- Inserting related data into one document.
- Improves query performance.
- Allows developers to update related data in a single write operation.

Data Modeling in Document Databases(MongoDB)

Embedding

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
           }
}
```

Embedded sub-document

Embedded sub-document

Data Modeling in Document Databases(MongoDB)

Embedding

- Create a database: Blog

```
use Blog
```

Data Modeling in Document Databases(MongoDB)

```
// Inserting a post with embedded user details
db.posts.insertOne({
  _id: ObjectId(1),
  title: "Post1",
  content: "Embedded post",
  user: {
    _id: ObjectId(1),  // Embedded user id
    username: "user1"  // Embedded username
  }
})
```

Data Modeling in Document Databases(MongoDB)

```
// Query details

db.posts.findOne({title: "Post1"})
```

Data Modeling in Document Databases(MongoDB)

Limitations of Embedding

- Embedding data into a single document can create large documents.(Maximum size of document: 16 MB)
- Continuously adding data without limits creates **unbound documents.**
- Large documents need to be read in full into memory which may slow down applications(Slower query performance and Excessive use of Memory).

Data Modeling in Document Databases(MongoDB)

Referencing

- Referring other documents in the collection or in other collections.
- Used when storing related data in other documents or other collections.
- Stores “_id” ObjectID as a link between two documents.

Data Modeling in Document Databases(MongoDB)

Referencing

- Also known as **linking or data normalization** in MongoDB.
- Lessens duplication of data.
- Smaller documents are created.

Data Modeling in Document Databases(MongoDB)

contact document

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

user document

```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

access document

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Data Modeling in Document Databases(MongoDB)

```
// Inserting a user
db.users.insertOne({
  _id: 1,
  username: "user1"})
```

Data Modeling in Document Databases(MongoDB)

```
// Inserting a post with a reference to the user
db.posts.insertOne({
  _id: 2,
  title: "Post2",
  content: "Hello World",
  user: 1})
```

Data Modeling in Document Databases(MongoDB)

```
// Querying post
db.posts.findOne({title: “Post2”})
```

Note: Mongodb creates new ObjectIDs each time you use the keyword to insert values.

Input for ObjectID should be a 24 character hex string, 12 byte Uint8Array, or an integer.

Data Modeling in Document Databases(MongoDB)

Limitations of Referencing

- Querying from multiple documents can cost extra resources(Memory).
- Read operations are slower(Performed using aggregations).

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

Scaling data Models

- Query patterns need to align with data models for optimum efficiency of
  - Query Result Time
  - Memory Usage
  - CPU Usage
  - Storage

Scaling data Models

- Avoid **Unbounded Documents**
  - Unbounded documents are documents that grow infinitely.
  - Unbounded documents result in poor query and write performance.
  - Results in excessive use of memory.

Schema anti-patterns

Schema design patterns are guidelines that help developers plan,organize and model data.

Schema anti-patterns result in:

- Sub-optimal performance
- Non-scalable solutions

Schema anti-patterns

Common schema anti-patterns:

- Massive arrays
- Massive number of collections
- Bloated documents
- Unnecessary indexes
- Queries without indexes
- Data that is accessed together but stored in different collections.

Open MongoDB Compass:

10 Minutes

Load Sample Dataset:

Download all json files

Using MongoDB compass

- Create a database “analytics” and collections “accounts”
- Load the accounts.json file to accounts collections to insert data.
- Repeat the same steps for customers.js and transactions.js

localhost:27017

My Queries

Performance

Databases

Search

Crea

Create Database

Database Name

analytics

Collection Name

accounts

☐ **Time-Series**

Time-series collections efficiently store sequences of measurements over a period of time. Learn More

Additional preferences (e.g. Custom collation, Capped, Clustered collections)

Cancel | Create Database

This collection has no data

It only takes a few seconds to import data from a JSON or CSV file.

Import Data

CRUD Operations on MongoDB

Read Operations

- find() : Lets one view all results for a condition.

```
db.customers.find({name: "Elizabeth Ray"} )
```

OR To get a specific document use $eq operator

```
db.customers.find({name: {$eq :"Elizabeth Ray"}})
```

CRUD Operations on MongoDB

Read Operations

- $in operator: Allows one to select all documents that have a field value equal to any of the values specified in the array.

```
db.accounts.find({limit:{$in: [10000,9000]}})
```

CRUD Operations on MongoDB

Read Operations

- Comparison operators:
  - $gt: Greater than
  - $gte: Greater than or equal to
  - $lt: Less than
  - $lte: Less than or equal to

```
db.accounts.find({limit:{$gte: 9000}})
```

CRUD Operations on MongoDB

Read Operations

- Access sub-document fields using “.” notation:

  Syntax: “field.nested-field”

  db.accounts.find({“products.0”: “InvestmentStock”})

Note: As no nested-field name was given for the selection of products an index was automatically generated.

CRUD Operations on MongoDB

Read Operations

- $elemMarch: Only returns values which are part of an **array**.

```
db.accounts.find({products: {$elemMatch: {$eq: "InvestmentStock"}}})
```

CRUD Operations on MongoDB

Read Operations

Logical operators

- $and: Find all results that match a criteria.

```
db.accounts.find({$and:[{"products.0":"CurrencyService"}
,{"products.1": "InvestmentStock"}]})
```

CRUD Operations on MongoDB

Read Operations

Logical operators

- $or: Find all results that match one of the given criterias.

```
db.accounts.find({$or:[{"products.0":"CurrencyService"},{"products.0": "InvestmentStock"}]})
```

CRUD Operations on MongoDB

```
db.accounts.find({
  $or: [
  { "products.0":"CurrencyService" },
  { "products.0": "InvestmentStock" }],
  $or: [
  { "products.1": "CurrencyService" },
  { "products.1": "InvestmentStock" }]
  })
```

CRUD Operations on MongoDB

The operator operations overwrite themselves, hence one needs to use **$and** operator when using multiple operators.

CRUD Operations on MongoDB

Update Operations

replaceOne(): replaces the first matching document in the collection that matches the filter, using the replacement document.

Syntax:

db.collections.replaceOne(filter,replacement,options)

db.customers.replaceOne({username: 'fmiller'},{email: "dummy@gmail.com"})

Output: matchedCount: 1,modifiedCount: 1,

CRUD Operations on MongoDB

Update Operations

updateOne(): Updates one document, has set and push operators.

- **$set** Operator: Replaces value to a field in the document, can also add a new field and value
- **$push** Operator: Appends a value to an array, if field is absent,$push adds the array field with the value as its element.

CRUD Operations on MongoDB

Update Operations

updateOne():

```
db.customers.updateOne({username:"charleshudson"},
{$set: {address: "thimphu"}})
```

Check:

```
db.customers.findOne({username:"charleshudson"})
```

CRUD Operations on MongoDB

Update Operations

updateOne():

```
db.accounts.updateOne({account_id:557378},{$push:
{products:"Software"}})
```

Check:

```
db.accounts.findOne({account_id:557378})
```

CRUD Operations on MongoDB

Update Operations

updateOne():

upsert option: Inserts a document with provided information if matching documents do not exist.

```
db.accounts.updateOne({account_id:5},{$set:{products:"Software"}},{upsert:true})
db.accounts.findOne({account_id:5})
```

CRUD Operations on MongoDB

Update Operations

findandModify: Returns the document that has just been updated

Syntax

```
findandModify({query:{},update:{},new:true})
```

CRUD Operations on MongoDB

Update Operations

findandModify: Returns the document that has just been updated

Syntax

```
findandModify({query:{},update:{},new:true})
```

CRUD Operations on MongoDB

Update Operations

findandModify: Returns the document that has just been updated

```
db.customers.findAndModify({
  query:{username:"serranobrian"},
  update:{$set:{name:"Loday"}},
  new:true})
```

CRUD Operations on MongoDB

Update Operations

updateMany(): Updates all documents in a collection that fulfills a certain criteria.

```
db.accounts.updateMany({limit: 10000},{$set:{limit: 11000}})
```

Check:

```
db.accounts.find()
```

CRUD Operations on MongoDB

Update Operations

updateMany(): Updates all documents in a collection that fulfills a certain criteria.

- This is not an all or nothing operation.
- It will not rollback updates, if some of the updates were not fulfilled.

CRUD Operations on MongoDB

Delete Operations

deleteOne(): Deletes one document that fulfills the criteria.

```
db.customer.findOne({username:"loday"})

db.customers.deleteOne({username:"loday"})

db.customer.findOne({username:"loday"})
```

CRUD Operations on MongoDB

Cursor: Pointer to the result set of a query.

find(): returns a cursor that points to the document.

Cursor Methods:

- Cursor.sort(): Sorts the result query in ascending or descending order.
- Cursor.limit(): Limits the number of results displayed.

CRUD Operations on MongoDB

Sort

```
db.customers.find().sort({name:1})
```

1: Ascending order

-1: Descending order

```
db.customers.find().sort({name:1,email:1})
```

CRUD Operations on MongoDB

Limit

```
db.customers.find().sort({name:1}).limit(3)
```

CRUD Operations on MongoDB

Projections

From the find() method, the number of fields retrieved to display can also be modified.

db.collections.find(<filter>,<document>,<options>)

-Specifying the second document field will limit the number of fields shown.

CRUD Operations on MongoDB

Projections

```
db.customers.find({},{username:1,name:1})
```

- Setting value to 0 will exclude a field
- Exclusion and Inclusion cannot be done in the same query except for "_id" field.

```
db.customers.find({},{username:1,name:1,_id:0})
```

CRUD Operations on MongoDB

Counting Documents

```
db.customers.countDocuments()
```

```
db.accounts.countDocuments({limit:{$gte:9000}})
```

Break 5 Minutes

MongoDB Atlas

- A cloud database service which provides multi-model databases over the Internet.

Clustering: A technique that involves creating multiple copies of databases across server.

- MongoDB Atlas Cluster: It is similar to a database on the cloud which allows for horizontal scaling(database sharding).

MongoDB Atlas

- The sharding method of horizontal scaling involves dividing a large database into smaller, more manageable pieces (called shards) and then distributing the shards across multiple machines.

MongoDB Atlas

- The sharding method of horizontal scaling involves dividing a large database into smaller, more manageable pieces (called shards) and then distributing the shards across multiple machines.

MongoDB Atlas

- Log into MongoDB using your college email address.

- Hope everyone has registered for MongoDB University. MongoDB University provides students will free access to courses and gives out Atlas credits.

account.mongodb.com/account/login

MongoDB.

Log in to your account

Don't have an account? Sign Up

Google

GitHub

Or with email and password

Email Address

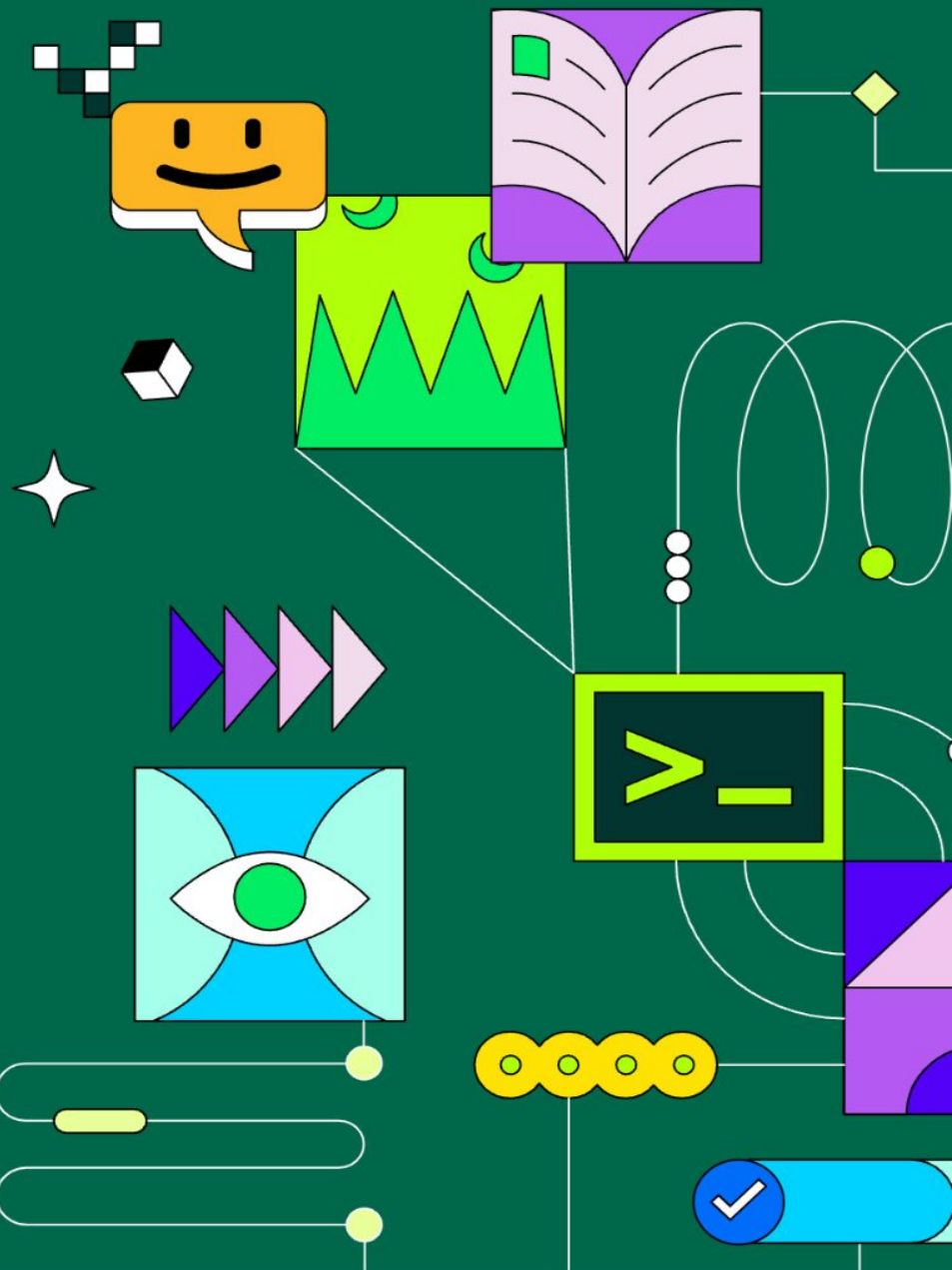
Next

MongoDB.local is coming to NYC!

Come to our flagship user conference for product announcements, technical deep-dives, and expert advice.

Enjoy 50% off with code WEB50.

Get 50% off →



MongoDB Atlas

Click on **browse collections** to view your databases and collections on Atlas.

Database Deployments

Create deployment

Cluster0

Connect | View info | Edit configuration

Data Size: 134.44 MB

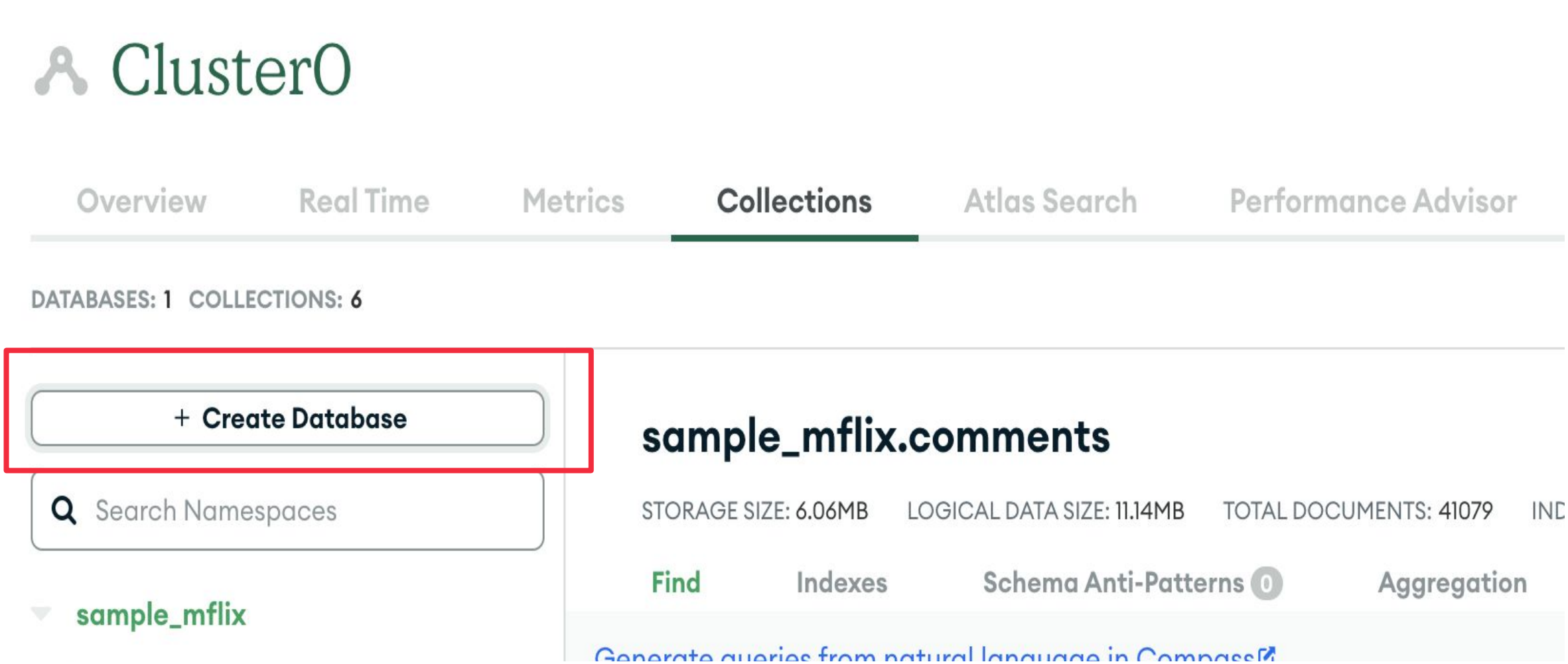
Browse collections →

View monitoring →

+ Add Tag

MongoDB Atlas

Click on **create database** to create a new database healthInformation.


Cluster0
Overview
Real Time
Metrics
Collections
Atlas Search
Performance Advisor
DATABASES: 1 COLLECTIONS: 6
+ Create Database
Search Namespaces
sample_mflix
sample_mflix.comments
STORAGE SIZE: 6.06MB
LOGICAL DATA SIZE: 11.14MB
TOTAL DOCUMENTS: 41079
IND
Find
Indexes
Schema Anti-Patterns 0
Aggregation

CST

Create Database

Database name

healthInformation

Collection name

patient

Additional Preferences

Select

Cancel

Create

Cluster0

VERSION 7.0.8 REGION AWS Mumbai (ap-south-1)

Overview | Real Time | Metrics | Collections | Atlas Search | Performance Advisor | Online Archive | Cmd Line Tools

DATABASES: 2 COLLECTIONS: 7

VISUALIZE YOUR DATA | REFRESH

+ Create Database

Search Namespaces

healthInformation

patient

sample_mflix

healthInformation.patient

STORAGE SIZE: 4KB LOGICAL DATA SIZE: 0B TOTAL DOCUMENTS: 0 INDEXES TOTAL SIZE: 4KB

Find | Indexes | Schema Anti-Patterns 0 | Aggregation | Search Indexes

Generate queries from natural language in Compass

INSERT DOCUMENT

Filter

```
Type a query: { field: 'value' }
```

Reset | Apply | Options

QUERY RESULTS: 0

MongoDB Atlas

Click on insert document to add a document to your collection patient.

- You can toggle between options to add your input as json or table entry format.

MongoDB Atlas

healthInformation.patient

STORAGE SIZE: 4KB LOGICAL DATA SIZE: 0B TOTAL DOCUMENTS: 0 INDEXES TOTAL SIZE: 4KB

Find Indexes Schema Anti-Patterns 0 Aggregation Search Indexes

Generate queries from natural language in Compass

INSERT DOCUMENT

Filter Type a query: { field: 'value' } Reset Apply Options

Insert Document

To collection patient

VIEW {} ≡

```
1   _id: ObjectId('662515c6dbde2a3cdee805d7')        ObjectId
```

Cancel Insert

Insert Document

To collection patient

VIEW

```
1 {"_id":{"$oid":"662515c6dbde2a3cdee805d7"},
2  "name": "Dolma",
3  "Current_Address": "Gedu"}
```

Cancel Insert

QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('662515c6dbde2a3cdee805d7')
name : "Dolma"
Current_Address : "Gedu"
```

MongoDB Atlas

- You can go to command line to install Atlas cli or other tools to manage your cluster.

Connect To Your Cluster

Methods to connect your application to your cluster via MongoShell, URI, or Compass can be found in the connect modal.

Connect Instructions

Manage Your Cluster From the Atlas Command Line

Create and manage MongoDB Atlas resources from your command line and easily automate them using scripts. Learn more

Install Atlas CLI

Utilities to manage your Cluster From the Command Line

Use command line utilities to import and export data, restore backups, and view diagnostics

Install MongoDB Database Tools

Binary Import and Export Tools

Connecting to MongoDB Atlas through Cli

MongoDB Atlas

Click on Network Access under Security to add your IP to access your cluster from the cloud.

- Without adding your IP to the network access list your laptop/workstation will not be authorised to access the cluster.

echnology
an

SECURITY

Quickstart

Backup

Database Access

Network Access

Network Access

IP Access List | Peering | Private Endpoint

ADD CURRENT IP ADDRESS

Current IP Address not added. You will not be able to connect to databases from this address.

Do not show me again

Network Access

IP Access List | Peering | Private Endpoint

Current IP Address added!
Visit **Network Access** to modify your settings.

MongoDB Atlas

Connection String:

- It is a URI that helps a client connect to a standalone cluster, replica sets or sharded clusters.

In our case, the connection string will help us gain access to our Atlas cluster.

Note: The connection string is also used by a client application to connect to a database server.

MongoDB Atlas

Connection String Syntax:

```
mongodb+srv://[username:password@]host[/[defaultauthdb]
[?options]]
```

Data Services

Database Deployments

Create deployment

•••

Cluster0

Connect | View info | Edit configuration

Data Size: 134.44 MB

Browse collections →

View monitoring →

Connect to your application

Drivers
Access your Atlas data using MongoDB's native drivers (e.g. Node.js, Go, etc.)

Access your data through tools

Data Explorer
Browse your Atlas collections without leaving the UI

Compass
Explore, modify, and visualize your data with MongoDB's GUI

Shell
Quickly add & update data using MongoDB's Javascript command-line interface

CS

Connect to Cluster0

Set up connection security — Choose a connection method — 3 Connect

I don't have the MongoDB Shell installed | I have the MongoDB Shell installed

1. Select your operating system and download the MongoDB Shell

macOS

Install via HomeBrew

Homebrew is a package manager for macOS.

```
brew install mongosh
```

See more installation options

2. Run your connection string in your command line

Use this connection string in your application

```
mongosh "mongodb+srv://cluster0.fvzdga6.mongodb.net/" --apiVersion 1 --username
```

You will be prompted for the password for the Database User, **paldenongmocst**. When entering your password,

Copy and Paste to terminal/Commandline

Recap:

- Document Model in MongoDB
- Data Modeling
- Embedding vs Referencing
- MongoDB Compass
- CRUD in MongoDB
- MongoDB Atlas

Next class:
Document Databases
- Indexing
- Aggregation
- Transactions
- Database Sharding