

MongoDB Developer Workshop

If you have any questions or issues during the session, please raise your virtual hand and a MongoDB expert will send you a 1:1 chat to assist.

Checkpoint 0

Who did not get the chance to set up their Atlas profile? A MongoDB expert can walk you through the process.

Checkpoint 0

Who did not get a chance to install Compass or get the wikipedia_tiny.json file?

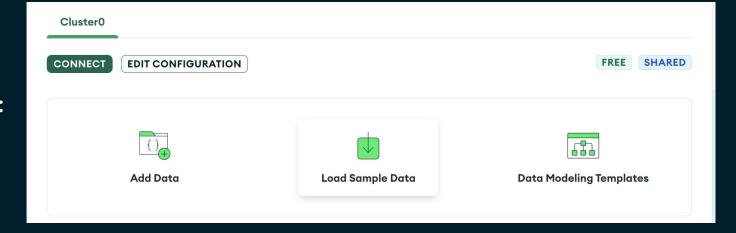
Load Sample Dataset



Yours could look like this:

Cluster0 Connect Enhance Your Experience For production throughput and richer metrics, upgrade to a dedicated cluster now! Upgrade	Prowse Collections R 0 W 0 Last 21 minutes	Edit Configuration Command Line Tools Load Sample Dataset Terminate	; 0	0	• In 0.0 B/s • Out 0.0 B/s Last 21 minutes 100.0 B/s	0
VERSION REGION	CLUSTER TIER T	үре ва	CKUPS LINKED APP SER	/ICES	ATLAS SQL ATLAS SEARCH	

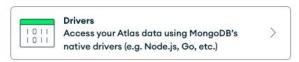
Or this:





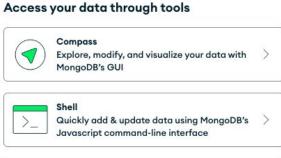


Connect to your application



MongoDB for VS Code

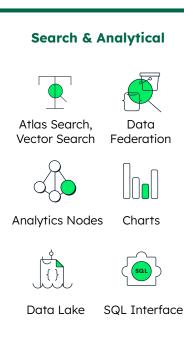
your VS Code environment

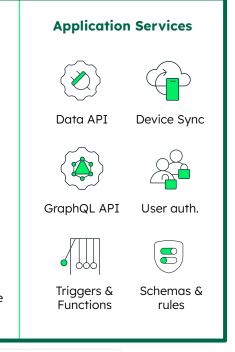




Work with your data in MongoDB directly from







Out-of-the-Box Integrations Kafka, Spark, Kubernetes Operator, Terraform Provider...and many more



Naturally maps to objects in code

- Eliminates requirements to use ORMs
- Breaks down complex interdependencies between developer and DBAs teams

Represent data of any structure

- Polymorphic: each document can contain different fields
- Modify the schema at any time (without downtime)
- · Guardrails with schema validation

Strongly typed for ease of processing

Over 20 binary encoded JSON data types

Access by <u>idiomatic drivers in all major programming</u> <u>language</u>

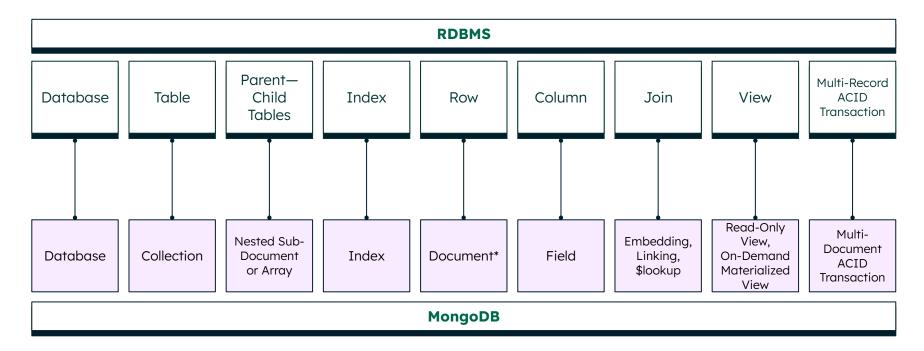
Document Data Model

Related data contained in a single, rich document

```
" id" : ObjectId("5ad88534e3632e1a35a58d00"),
  "name" : {
    "first": "John",
   "last" : "Doe" },
  "address" : [
   { "location" : "work",
      "address" : {
        "street": "16 Hatfields",
       "city": "London",
        "postal_code" : "SE1 8DJ"},
      "geo" : { "type" : "Point", "coord" : [
         -0.109081, 51.5065752]}},
   {...}
  "dob" : ISODate("1977-04-01T05:00:00Z"),
  "retirement_fund" : NumberDecimal("1292815.75"),
  "vector": [0.605620826459227, 0.4372084666070962,
0.9186864251789797, 0.632518798575036, ...]
```

Some terminology

A comparison



^{*} Documents contain more entity data than a relational database row

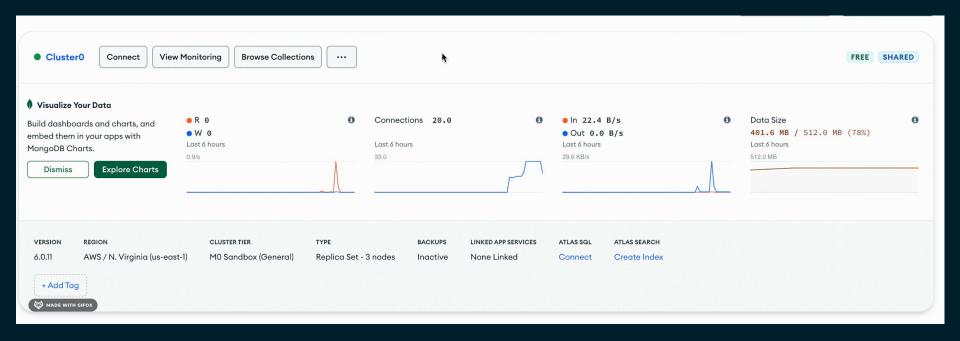


Checkpoint 1

Sample dataset loaded

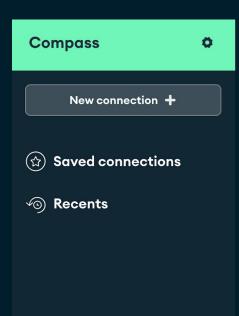


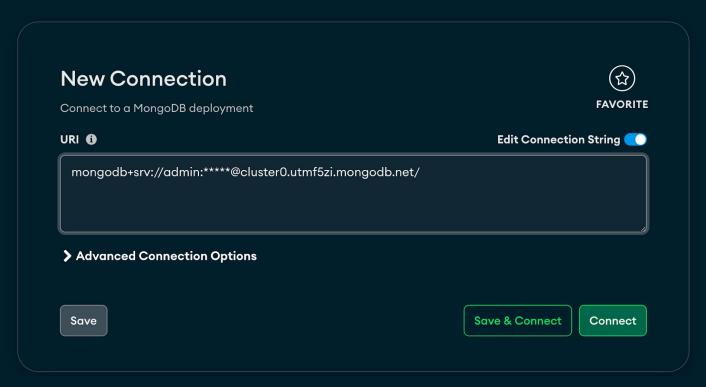
Get Atlas Connection String





Connect to Atlas using Compass



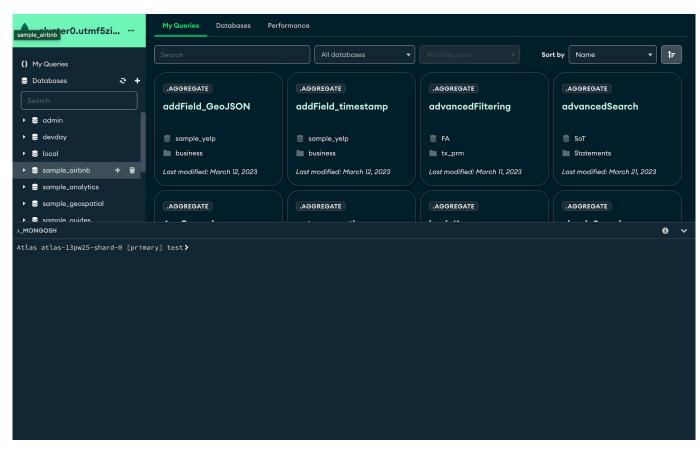


Compass GUI & Shell Overview





Shell





() My Queries Create database

- ▶ **⊜** ad<u>min</u>

Databases

- ▶ **S** local
- ▶ ≡ sample_airbnb
- sample_analytics
- ▶ **≥** sample_geospatial
- sample_guides
- sample_mflix
- sample_restaurants
- sample_supplies
- sample_training
- sample_weatherdata

Exercise 1: CRUD



Cı	reate Database
Dat	abase Name
m	ydb
Coll	ection Name
m	ycoll
	Time-Series Time-series collections efficiently store sequences of measurements over a perio of time. Learn More ^ど
> A	Additional preferences (e.g. Custom collation, Capped, Clustered collections)
	Cancel Create Databas

MQL Reference - Writes (Complete Guide)

SQL	MQL
<pre>INSERT INTO people(user_id, age, status) VALUES ("bcd001", 45, "A")</pre>	db.people.insertOne(person)
UPDATE people SET status = "C" WHERE age > 25	<pre>db.people.updateMany({ age: { \$gt: 25 } }, { \$set: { status: "C" } })</pre>
DELETE FROM people WHERE status = "D"	<pre>db.people.deleteMany({status: "D"})</pre>

Exercise 1: CRUD



```
db.<collectionName>.insertOne({key:value}, {options})
Insert, which takes as an argument a single document to insert
db.<collectionName>.insertMany([{k:v},{k:v}],{options})
InsertMany, which accepts an array of documents
```

Exercise 1: CRUD

Insert Command Syntax

```
db.<collectionName>.insertOne({key:value}, {options})
Insert, which takes as an argument a single document to insert
```

db.<collectionName>.insertMany([{k:v},{k:v}],{options})

InsertMany, which accepts an array of documents

Add me to your new collection:

```
"name" : {
    "first": "John",
    "last" : "Doe" },
  "address" : [
    { "location" : "work",
      "address" : {
        "street": "16 Hatfields",
        "city": "London",
        "postal_code" : "SE1 8DJ"},
      "geo" : { "type" : "Point", "coord" : [
       -0.109081, 51.5065752]}
"dob" :"1977-04-01T05:00:00Z",
"retirement_fund" : 1292815.75
```

Answer 1: CRUD

•

Compass GUI

Compass Shell

```
Insert Document
To collection mydb.mycoll
                                                    VIEW {}
        * Paste one or more documents here
          " id": {
            "$oid": "6536c61e127f42b59674877b"
              "first" : "John",
             "last" : "Doe" },
            "address" : [
   11 •
   12 🕶
              { "location" : "work",
               "address" : {
   13 ▼
   14
                 "street" : "16 Hatfields",
   15
                 "city": "London",
   16
                 "postal_code" : "SE1 8DJ"},
                "geo" : { "type" : "Point", "coord" : [
   17 🕶
                -0.109081, 51.5065752]}
   18
   19
   20
   21
          "dob" :"1977-04-01T05:00:00Z",
   23
                                                    Cancel
                                                              Insert
```

```
mydb > db.mycoll.insertOne(){
          "name" : {
            "first": "John",
            "last" : "Doe" },
          "address" : [
            { "location" : "work",
              "address" : {
                "street": "16 Hatfields",
                "city" : "London",
                "postal_code" : "SE1 8DJ"},
              "geo" : { "type" : "Point", "coord" : [
               -0.109081, 51.5065752]}
        ],
        "dob" :"1977-04-01T05:00:00Z",
        "retirement_fund" : 1292815.75
```

Exercise 2: CRUD



Update Command Syntax

```
db.<collectionName>.updateOne({filter},{update},{options})
db.<collectionName>.updateMany({filter},{update},{options})
```

{filter} - Which documents to update (i.e. where-clause).

{update} - Either a replacement or a modification.

{options} - notably, {upsert:true | false}. If the record exists, update it. If not, insert the contents of {update} as a document if it makes sense.

Common update operators

- \$set adds / updates fields
- <u>\$unset</u> removes fields
- \$inc increments a field
- \$\frac{\\$pull}{\} / \frac{\\$push}{\} removes or inserts into an array
- \$convert

Exercise 2: CRUD



Hint: Use dot notation to access nested fields!

Add this address to your record:

```
{
  "location" : "home",
  "address" : {
     "street" : "123 ABC Street",
     "city" : "AAA",
     "postal_code" : "111111"
  }
}
```

Start with:

```
let newAddress = {...above json...}
```

Change John Doe's lastName to "Bo":

```
{
  "location" : "home",
  "address" : {
     "street" : "123 ABC Street",
     "city" : "AAA",
     "postal_code" : "111111"
  }
}
```

Answer 2: CRUD



```
>_MONGOSH
> let newAddress = {
    "location" : "home",
    "address" : {
                                                Compass Shell
     "street": "123 ABC Street",
     "city" : "AAA",
     "postal code" : "111111"
 db.mycoll.updateOne({'name.last': 'Doe'},{$set: {'name.last': 'Bo'},$push: {'address': newAddress}})
Atlas atlas-13pw25-shard-0 [primary] mydb≯
```

Exercise 3: CRUD

Delete Command Syntax

```
db.collection.deleteOne({filter},{options})
db.collection.deleteMany({filter},{options})

{filter} - Which document(s) to delete (i.e.
where-clause).

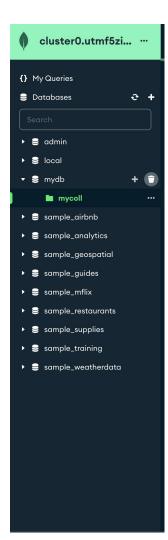
{options} - Options include things like write concern (a mongoDB parameter for durability / performance tradeoffs) collation (for language support) and hint (index hinting)
```

Delete this document

```
"name" : {
    "first" : "John",
    "last" : "Bo" },
    "address" : [...]
],
    "dob" :"1977-04-01T05:00:00Z",
    "retirement_fund" : 1292815.75
}
```

Answer 3: CRUD

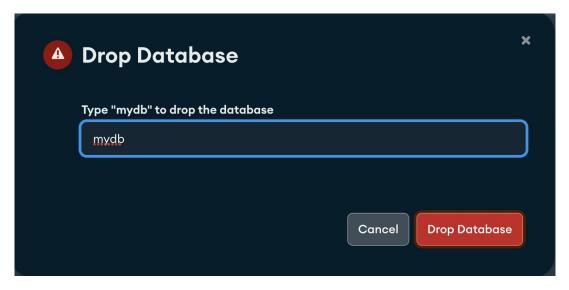






Exercise 4: CRUD - Drop the database

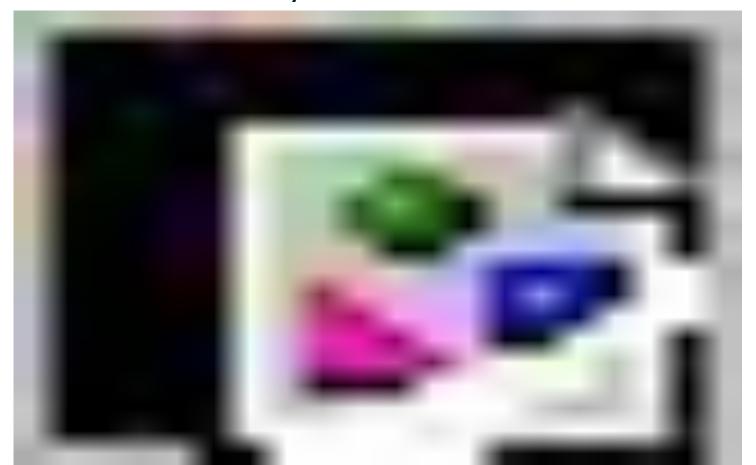




MQL Reference - Reads (Complete Guide)

SQL	MQL
SELECT * FROM people	db.people.find()
SELECT id, user_id, status FROM people	<pre>db.people.find({ }).project({ user_id: 1, status: 1 })</pre>
SELECT * FROM people WHERE status = "A"	<pre>db.people.findOne({ status: "A" })</pre>
SELECT * FROM people WHERE status = "A" AND age = 50	db.people.find({status: "A", age: 50})

Switch to the sample_mflix.movies collection & analyze the schema





Exercise 5: CRUD



Find Command Syntax

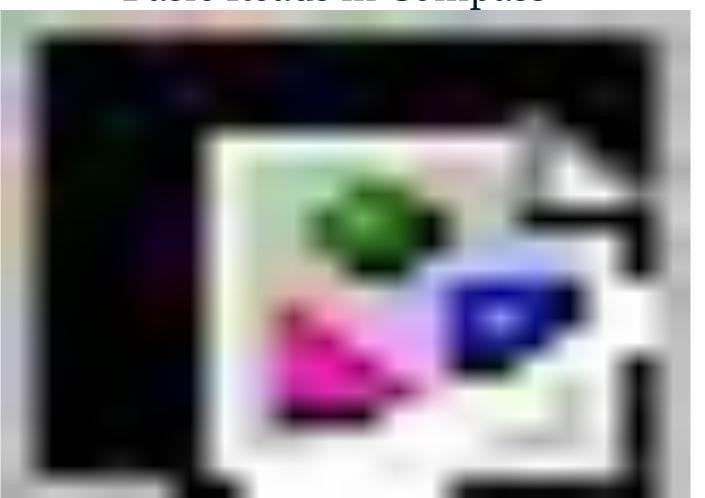
```
db.<collectionName>.findOne({filter},{projection})
Returns the first document matching the optional {filter}
```

db.<collectionName>.find({filter},{projection})
Returns a cursor(iterable pointer to many documents)

Common find (aka filter) operators

- <u>\$eq</u> (equals) Similar to a normal query
- \$gt/\$gte/\$1t/\$1te Greater [and equal] / less than [and equal]: important operator for range queries
- \$\frac{\sin}{\sin}:[] / \$\frac{\sin}{\sin}:[] (in / not in): Multiple value equality match
- \$ne Not equal
- <u>\$not</u>: Inverses the logic of a query
- \$or: Boolean OR
- <u>\$and</u>: Boolean AND. Usually used for specific nesting situations
- \$exists: Filters based on field existence
- \$type: Filters based on BSON type
- <u>\$elemMatch</u>: Query on objects within arrays

Basic Reads in Compass



Exercise 5: CRUD



Hint: Use dot notation to access nested fields!

- 1. Find 1 movie with the title "Blacksmith Scene"
- 2. Find movies released in 1991 with Brad Pitt
- 3. Find all movies released after 1991
- Find all records where the runtime fields does not exist:
- 5. Find all movies where the Rotten Tomatoes rating is greater than 3.4:

Answer 5: CRUD



Hint: Use dot notation to access nested fields!

- 1. Find 1 movie with the title "Blacksmith Scene"
- 2. Find movies released in 1991 with Brad Pitt
- 3. Find all movies released after 1991
- 4. Find all records where the runtime fields does not exist:
- 5. Find all movies where the Rotten Tomatoes rating is greater than 3.4:

- 1. db.movies.find({title: "Regeneration"})
- 2. db.movies.find({year:1991, cast:'Brad
 - Pitt'})
- 3. db.movies.find({year: {\$gt: 1991}})
- 4. db.movies.find({runtime: {\$exists:
- false}})
- 5. db.movies.find({'tomatoes.viewer.rating':{
 \$gte:3.4}})

CRUD: Sort, Limit, Skip, Project

```
db.<collectionName>.find({filter}): Returns a cursor (iterable pointer to many documents)
db.<collectionName>.find({filter}, {projection}): Returns only the specified fields.
Example: db.movies.find({title: "Regeneration"}, {_id: 0, title: 1, fullplot: 1})
db.<collectionName>.find({filter},{projection}).sort([{'<fieldName>':<1|-1>}])
db.<collectionName>.find({filter},{projection}).skip(<int>)
db.<collectionName>.find({filter},{projection}).skip().limit(<int>)
```

CRUD: Sort, Limit, Skip, Project

```
Type a query: { field: 'value' } or Generate query ★
                                                             Explain
                                                                          Find
db.<collectionName>.find({filter}): Returns a cursor (iterable pointer to many documents)
db.<collectionName>.find({filter}, {projection}): Returns only the specified fields.
Example: db.movies.find({title: "Regeneration"}, {_id: 0, title: 1, fullplot: 1})
db.<collectionName>.find({filter},{projection}).sort([{'<fieldName>':<1|-1>}])
db.<collectionName>.find({filter},{projection}).skip(<int>)
db.<collectionName>.find({filter},{projection}).skip().limit(<int>)
```

Exercise 6: CRUD

•

Find the top 10 movies by Rotten Tomatoes rating with Brad Pitt in it. Sort by highest rated.

Answer 6: CRUD



Find the top 10 movies by Rotten Tomatoes rating with Brad Pitt in it. Sort by highest rated.

Compass GUI

```
Filter 2
               {'cast': 'Brad Pitt'}
                                                                                            Explain
                                                                          Generate query ★
                                                                                                       Reset
                                                                                                                Find
                                                                                                                              Options ▼
               {_id: 0, title: 1, plot: 1, 'tomatoes.viewer.rating': 1}
Project
               {| tomatoes.viewer.rating': -1}
                                                                                                      MaxTimeMS
Sort
Collation
                                                                                                           Limit 10
               { locale: 'simple' }
                                                                                 Skip 0
```

Compass Shell

```
> let filter = {'cast': 'Brad Pitt'}
> let project = {'cast': 'Brad Pitt'}
> let sort = {'tomatoes.viewer.rating': -1}
> let limit = 10
Atlas atlas-13pw25-shard-0 [primary] sample_mflix > db.movies.find(filter, project).sort(sort).limit(10)
```

Checkpoint 2

MongoDB Query Fundamentals Done



Indexes in MongoDB



Index types

Primary index

Every collection has a primary key index

Compound index

Index against multiple keys in the document

Multikey index

Index into arrays

Wildcard index

Auto-index all matching fields, sub-documents & arrays

Text indexes

Support for text searches

Geospatial indexes

2d & 2d sphere indexes for spatial geometries

Hashed indexes

Hashed based values for sharding

<u>Index features</u>

TTL indexes

Single field indexes, when expired delete the document

Unique indexes

Ensures value is not duplicated

Partial indexes

Expression based indexes, allowing indexes on subsets of data

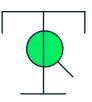
Case-insensitive indexes

Supports text search using case insensitive search

Sparse indexes

Only index documents which have the given field

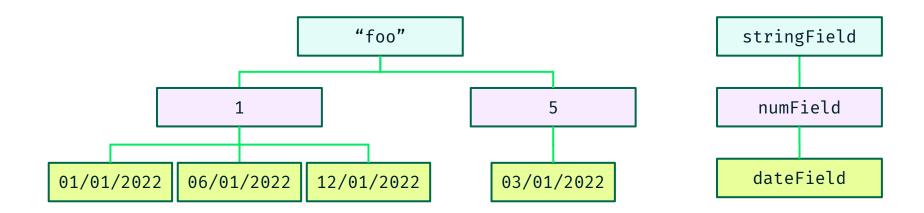
Atlas Search \$SEARCH - single command integrating Lucene engine





Index Structure Example

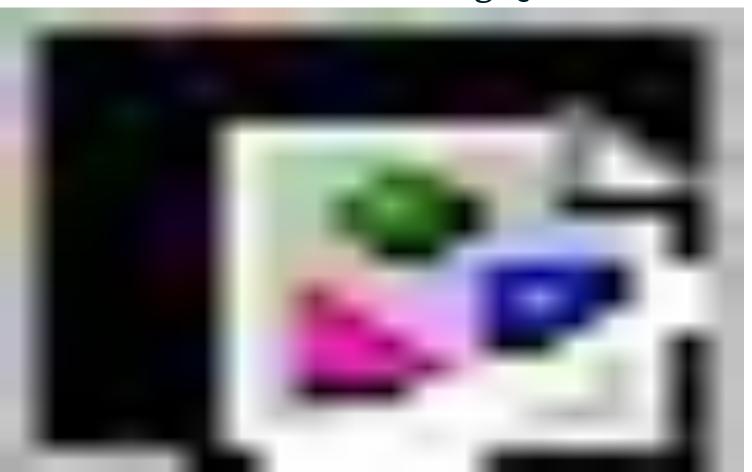
Index - db.<collection>.createIndex({stringField: 1, numField: 1, dateField: 1})



ISODate is formatted as "mm/dd/yyyy" for presentation purposes

Exercise 7: Understanding Queries & Indexes

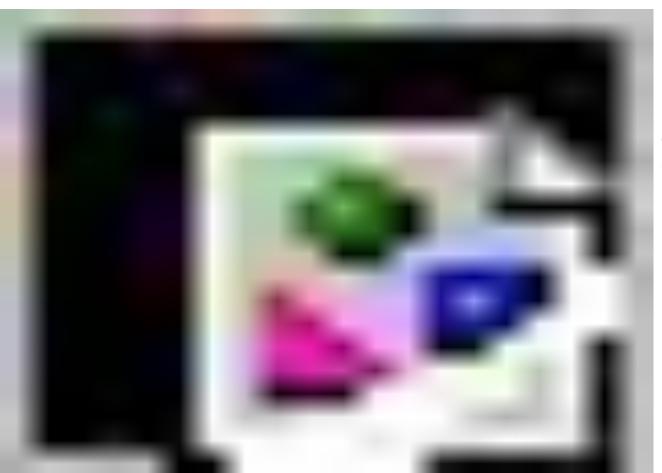




db.movies.find({
 'tomatoes.viewer
 .rating':{\$gte:3
 .4}}).explain()

Exercise 7: Create an Index





db.movies.createIndex({'
tomatoes.viewer.rating':
1})

Exercise 7: Re-Examine the Query





db.movies.find({
 'tomatoes.viewer
 .rating':{\$gte:3
 .4}}).explain()

The Aggregation Framework

Advanced data processing pipeline for transformations and analytics

Example:

```
db.orders.aggregate( [
   { $match:
            { status: "A" } },
   { $group:
             { _id: "$cust_id", total: { $sum: "$amount" } } },
   { $sort: { "customer.id" : 1 } },
   { $project: { "customer.name.first_name" : 1 , acctStatus : 1 } }
```

Over 25 Stages and 100 Operators:

https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/

\$addFields

\$bucket

\$bucketAuto \$collStats

\$count \$facet

\$geoNear

\$graphLookup \$group

\$indexStats \$limit

\$listSessions

\$lookup \$match

\$merge

\$out

\$planCacheStats

\$project \$redact

\$replaceRoot

\$replaceWith \$sample

\$search

\$set \$skip

\$sort

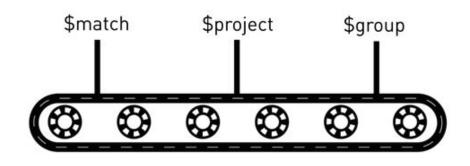
\$sortByCount

\$unionWith



Retrieval operations can filter:

- What Documents
- What Fields
- What Order



Aggregation allows us to compute new data & support complex manipulation of documents such as:

- Calculating new/virtual fields
- Grouping & summarizing values
- Reshaping documents



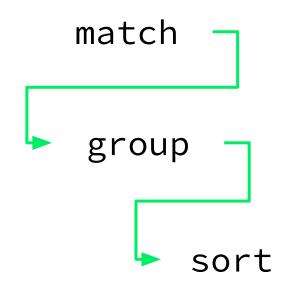
Syntax Comparison

```
db.listingsAndReviews.find( //find() - functions, arguments and APIs
  { "address.country": "Canada" },
  { "host.host_total_listings_count": 1, "host.host_name": 1})
  .sort({"host.host_total_listings_count": -1}).limit(1)
db.listingsAndReviews.aggregate([ //aggregate() - an array of objects
  {$match: {"address.country": "Canada", "price": { $lt: 200, $gt: 100
}}},
  {\project: \{\text{"host.host_total_listings_count\text{": 1, \text{"host.host_name\text{": 1}}\},
  {\$sort: \{ \"host.host_total_listings_count\": \[ -1 \] \},
  {$limit: 1}])
```

*nix Pipes

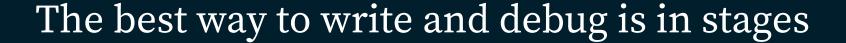
MQL Pipelines

ps ax | ¬ grep mongod | ¬ head 1



Basic Aggregation Stages

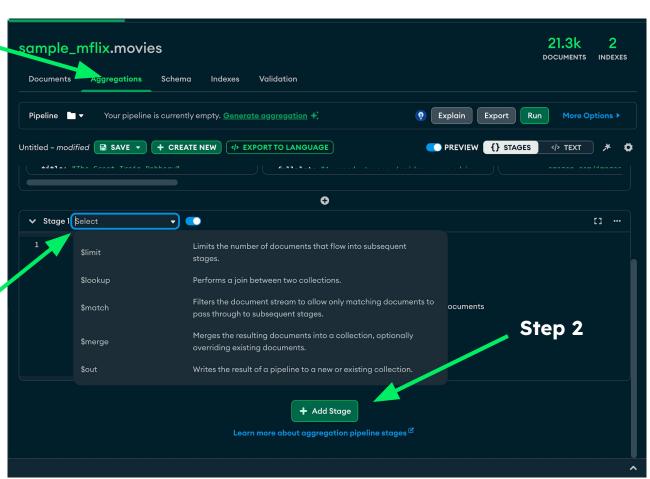
When using these at the start of a pipeline, the query optimizer transforms them to a find()



```
only_Canada = { $match: { "address.country": "Canada" }}
host_Nlstngs_name = {\$project: {"host.host_total_listings_count": 1,
                                "host.host name": 1 }}
most_lstngs = {$sort: {"price": -1}}
first_place = {$limit: 1}
db.listingsAndReviews.aggregate([only_Canada, host_Nlstngs_name,
                                             most_lstngs, first_place])
```

Aggregation Pipeline Builder

Step 1



Step 3

Exercise 8: Aggregation 101

•

Translate this MQL into an Aggregation Pipeline

```
let filter = {'cast': 'Brad Pitt'}
let project = {_id: 0, title: 1}
let sort = {'tomatoes.viewer.rating': -1}
let limit = 10
db.movies.find(filter, project).sort(sort).limit(limit)
```

Answer 8: Aggregation 101

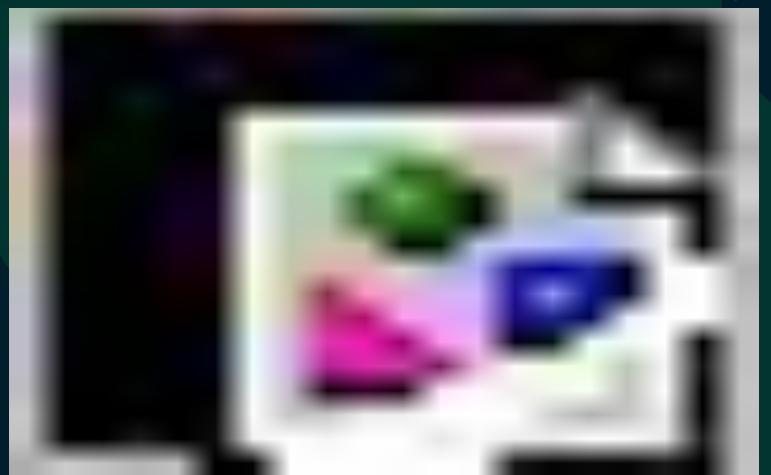
Translate this MQL into an Aggregation Pipeline

```
let filter = {'cast': 'Brad Pitt'}
let project = {_id: 0, title: 1}
let sort = {'tomatoes.viewer.rating': -1}
let limit = 10
db.movies.find(filter, project).sort(sort).limit(limit)
       let pipeline = [
         {$match: { 'cast': "Brad Pitt",}},
         {project: { _id: 0, title: 1}},
         {$sort: { "tomatoes.viewer.rating": -1}},
         {$limit: 10}
       db.movies.aggregate(pipeline)
```

Checkpoint 3

Queries and Indexing done

Exercise 9: Import wikipedia_tiny





Indexes in MongoDB



Index types

Primary index

Every collection has a primary key index

Compound index

Index against multiple keys in the document

Multikey index

Index into arrays

Wildcard index

Auto-index all matching fields, sub-documents & arrays

Text indexes

Support for text searches

Geospatial indexes

2d & 2d sphere indexes for spatial geometries

Hashed indexes

Hashed based values for sharding

<u>Index features</u>

TTL indexes

Single field indexes, when expired delete the document

Unique indexes

Ensures value is not duplicated

Partial indexes

Expression based indexes, allowing indexes on subsets of data

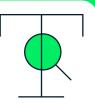
Case-insensitive indexes

Supports text search using case insensitive search

Sparse indexes

Only index documents which have the given field

Atlas Search \$SEARCH - single command integrating Lucene engine



Atlas Search Capabilities Summary



Search features are all about making data more **discoverable** and **relevant** for end users

Indexing

- Support for 40+ languages
- Multiple data types
- Custom Analyzers
- Automated ETL
- Autocomplete
- Faceting
- Stored source
- Vector store

Querying

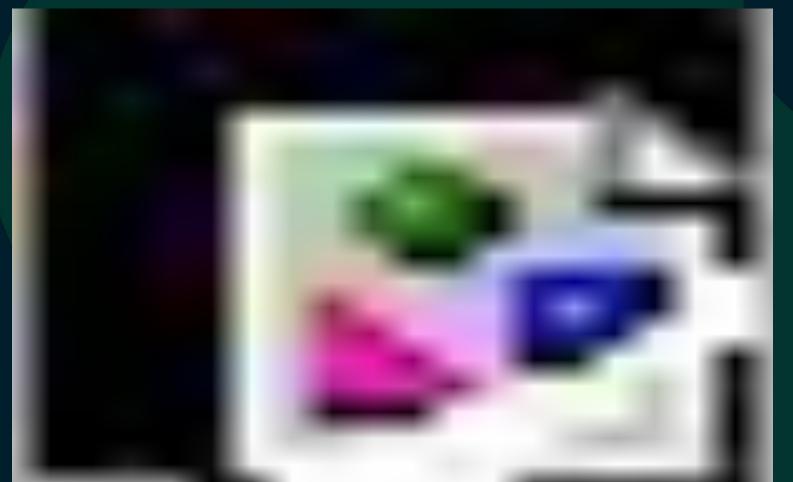
- Rich Query
- Fuzzy Matching
- Scoring
- Autocomplete
- Faceting / Filtering
- Highlighting
- Synonyms
- moreLikeThis
- Semantic search

Wikipedia Collection Schema



```
_id: ObjectId('6529be4ec4b2843f2bbe6f85')
 dataset_name: "wikipedia/20220301.en"
 dataset_date: "20220301"
 model_name: "sentence-transformers/all-MiniLM-L6-v2"
 language: "english"
 title: "Lake Andes, South Dakota"
 url: "https://en.wikipedia.org/wiki/Lake%20Andes%2C%20South%20Dakota"
 text: "Lake Andes is a city in, and the county seat of, Charles Mix County, S..."
vector: Array (384)
 _id: ObjectId('6529b1bac4b2843f2bbd5f40')
 dataset_name: "wikipedia/20220301.en"
 dataset_date: "20220301"
 model name: "sentence-transformers/all-MiniLM-L6-v2"
 language: "english"
 title: "Isaac Bonewits"
 url: "https://en.wikipedia.org/wiki/Isaac%20Bonewits"
 text: "Phillip Emmons Isaac Bonewits (October 1, 1949 - August 12, 2010) was ..."
▶ vector: Array (384)
 _id: ObjectId('6529c7dcc4b2843f2bbf3da8')
 dataset_name: "wikipedia/20220301.en"
 dataset_date: "20220301"
 model_name: "sentence-transformers/all-MiniLM-L6-v2"
 language: "english"
```

Exercise 10: Create Atlas Search Index



Exercise 10: Query Atlas Search with <u>\$search</u>



Fuzzy Example

```
$search = {
  index: 'default',
 text: {
     query: '<user input>',
     path: 'fullplot',
     fuzzy: {
       maxEdits: 2,
       prefixLength: 0
$project = {
 score: {
    $meta: "searchScore",
 },
 highlights: {
    $meta: "searchHighlights",
```

Exercise 10: Query Atlas Search with <u>\$search</u>



Fuzzy Example

```
$search = {
  index: 'default',
  text: {
     query: '<user input>',
     path: 'fullplot',
     fuzzy: {
       maxEdits: 2,
       prefixLength: 0
$project = {
  score: {
    $meta: "searchScore",
 },
 highlights: {
    $meta: "searchHighlights",
```

Highlight Example

```
index: 'default',
text: {
   query: '<user input>',
   path: 'fullplot',
   fuzzy: {
     maxEdits: 2,
     prefixLength: 0
highlight: {
   path: "fullplot"
```

Compound Example

```
index: "demo_mflix_movies",
compound: {
  should: {
    text: {
      path: "fullplot",
      query: "werewolves",
      fuzzy: {
        maxEdits: 2,
      },
  mustNot: {
    text: {
      path: "fullplot",
      query: "vampires",
    },
```

Exercise 11: Create Optimized Search Index

- In the Atlas UI, go back to the Data Services Tab
- Click Browse collections in Database Deployments
- Enter the 'Search' Tab and Click on 'Create a Search Index'
- Select 'Visual Editor' as the Configuration Method
- Click on Next
- Fill in the details for the index:
 - a. Index Name: optimized
 - b. Database and Collection: devday.wikipedia
 - c. Next
- Click Refine Your Index
 - a. Select lucene.english for each Analyzer
 - b. Disable Dynamic Mapping
 - Click Add Field Mapping and Map both title and text fields to String, keeping the other defaults
- Click on Save Changes and then Create Search Index



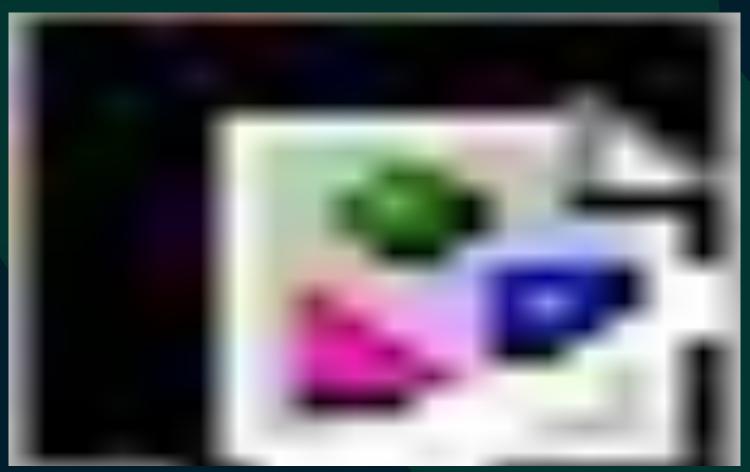
Notice the difference in index size

devday.wikipedia

Indexes Used: 2 of 3.

Name	Index Fields	Status 19	Size	Documents
optimized	"text", "title"	ACTIVE View status details	Primary Node: 32.26MB	Primary Node: 5,000 (100%) indexed of 5,000 total
default	[dynamic]	ACTIVE View status details	Primary Node: 76.04MB	Primary Node: 5,000 (100%) indexed of 5,000

Exercise 11: Vector Search Index •





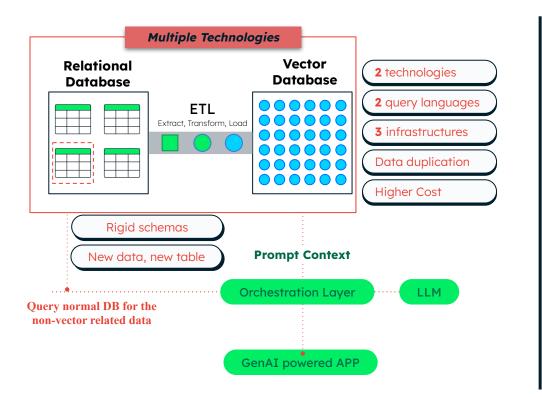
Vector Search Index Definition

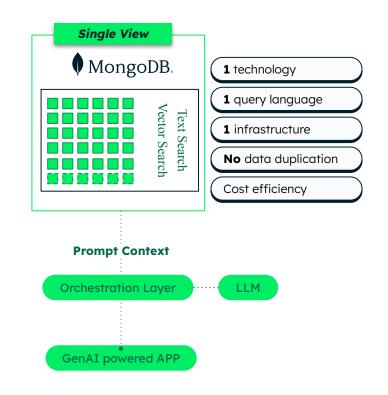


```
"mappings": {
  "dynamic": false,
  "fields": {
    "language": [{
        "type": "token",
        "normalizer": "lowercase"
      },
        "type": "string"
    "vector": {
      "dimensions": 384,
      "similarity": "cosine",
      "type": "knnVector"
```

Atlas Vector Search

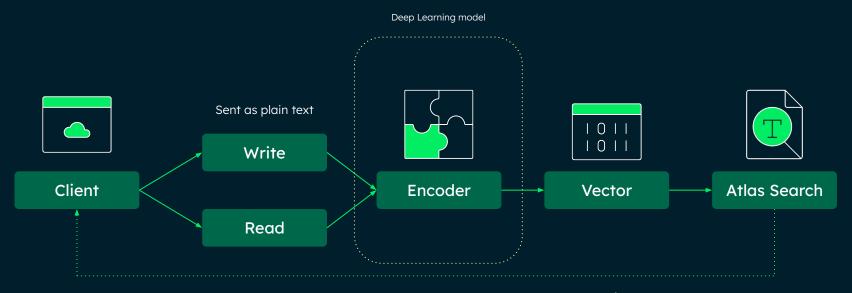






Semantic Search Data Workflow





Documents returned as they're stored, typically remove vectors via \$project

Exercise 11: Semantic Search

```
Use devday
let vector = <vector>
let agg = [{
  $vectorSearch: {
    queryVector: vector,
    Path: 'vector',
    numCandidates: 50,
    index: 'vector',
    limit: 10,
}]
db.wikipedia.aggregate(agg)
```

Exercise 12: Semantic Search with Filter

```
let agg = [{
  $vectorSearch: {
    queryVector: vector,
    Path: 'vector',
    numCandidates: 50,
    index: 'vector',
    limit: 10,
    filter: {
      $and: [{
       'language': {$eq:'english'}
      }]
db.wikipedia.aggregate(agg)
```

What you accomplished today

- Explored what a document database is and how to work with it in MongoDB Atlas:
 - Basic CRUD operations
 - Reads with 1+ filters, with sorting, projecting, and limiting results
 - How to analyze queries and commands
 - How to create and leverage database indexes
 - The **aggregation framework** and how it differs from the Query API (i.e. MQL)
- Imported a **multi-lingual**, **vectorized dataset** in your cluster
- Performed full-text search using Atlas Search
- Performed **vector search**:
 - Semantic searches using cosine similarity
 - Combined search filtering and vector search to optimize accuracy & performance

Thank you