WELCOME

MongoDB Developer Workshop

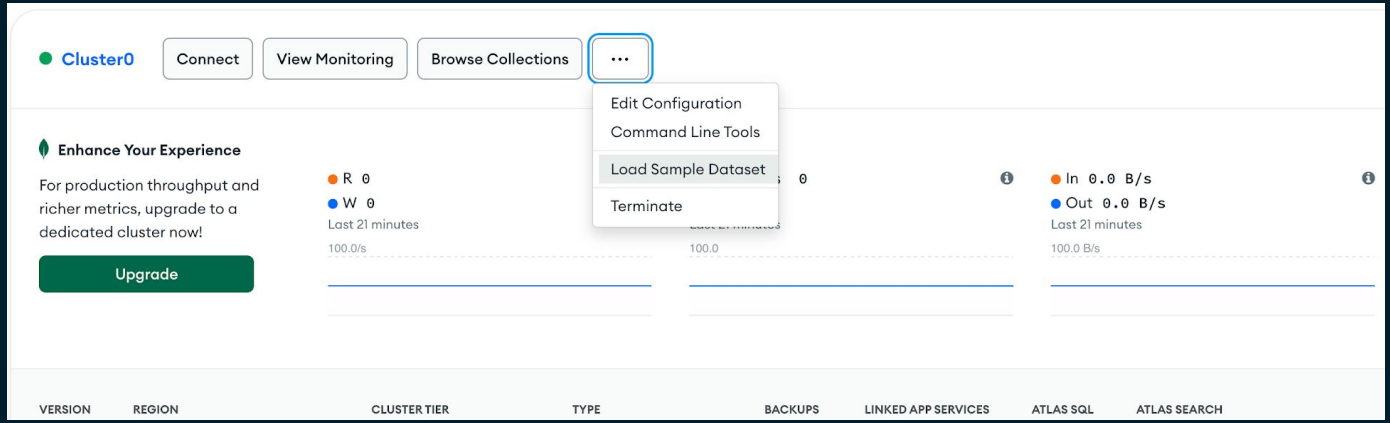
GitHub to Follow Along

Go to this GH repo (https://github.com/nickgogan/MongoDBAtlasDeveloperDay) to find a README.md file with:

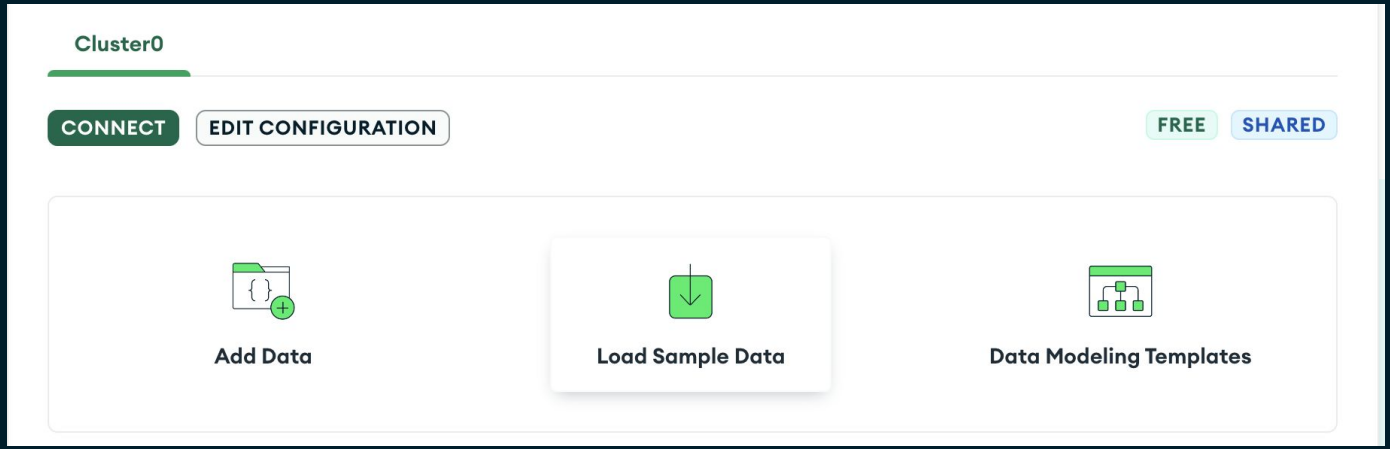
- Prerequisites
- Plaintext instructions
- Properly formatted code to copy/paste or refer to as you go through the exercises

Load Sample Dataset

Yours could look like this:

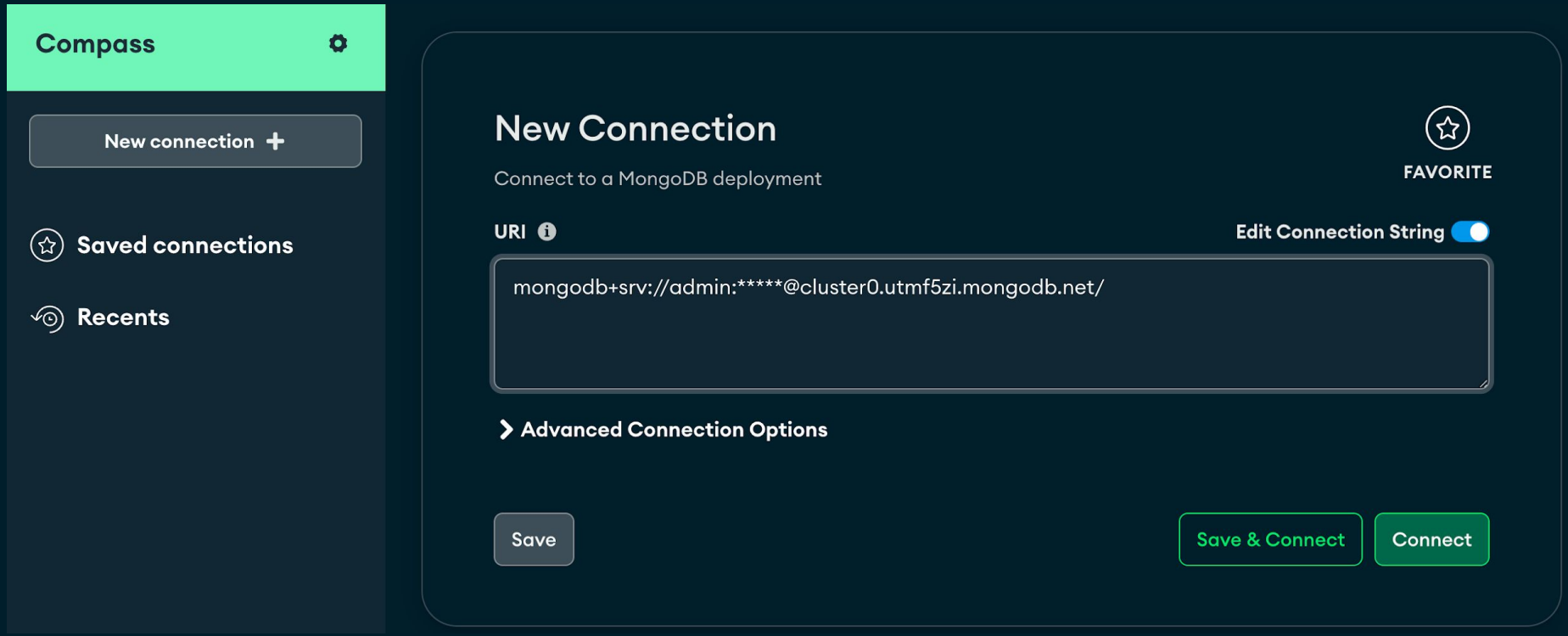


Or this:



Get Atlas Connection String

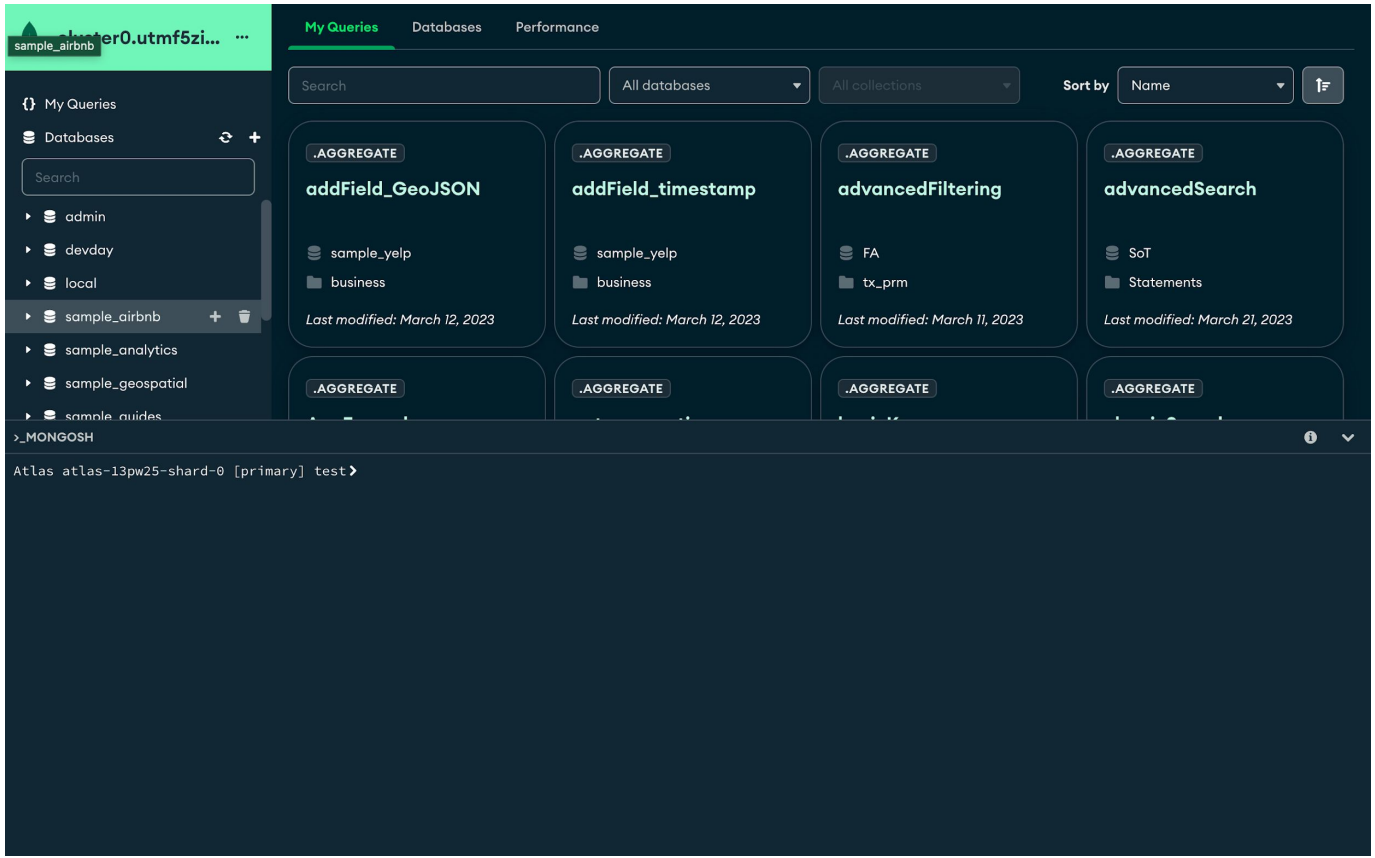
Connect to Atlas using Compass



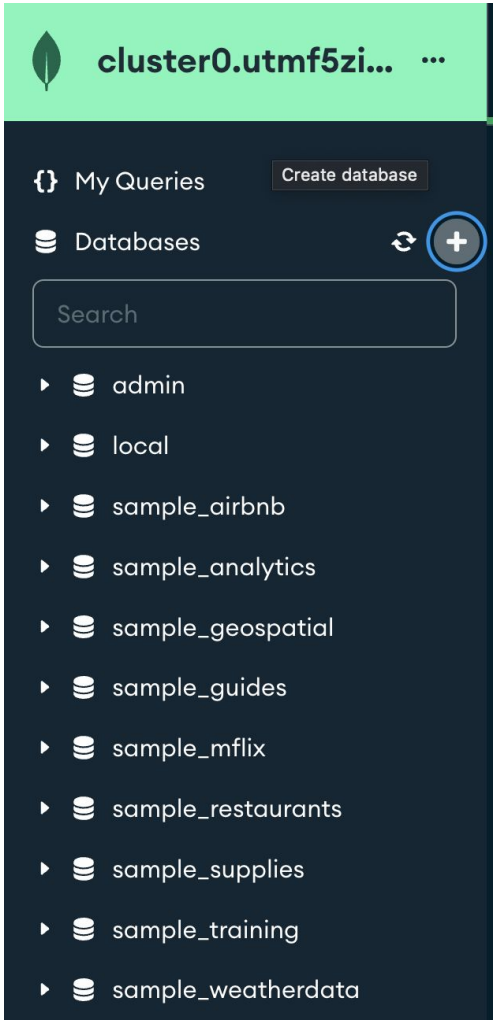
Compass GUI & Shell Overview

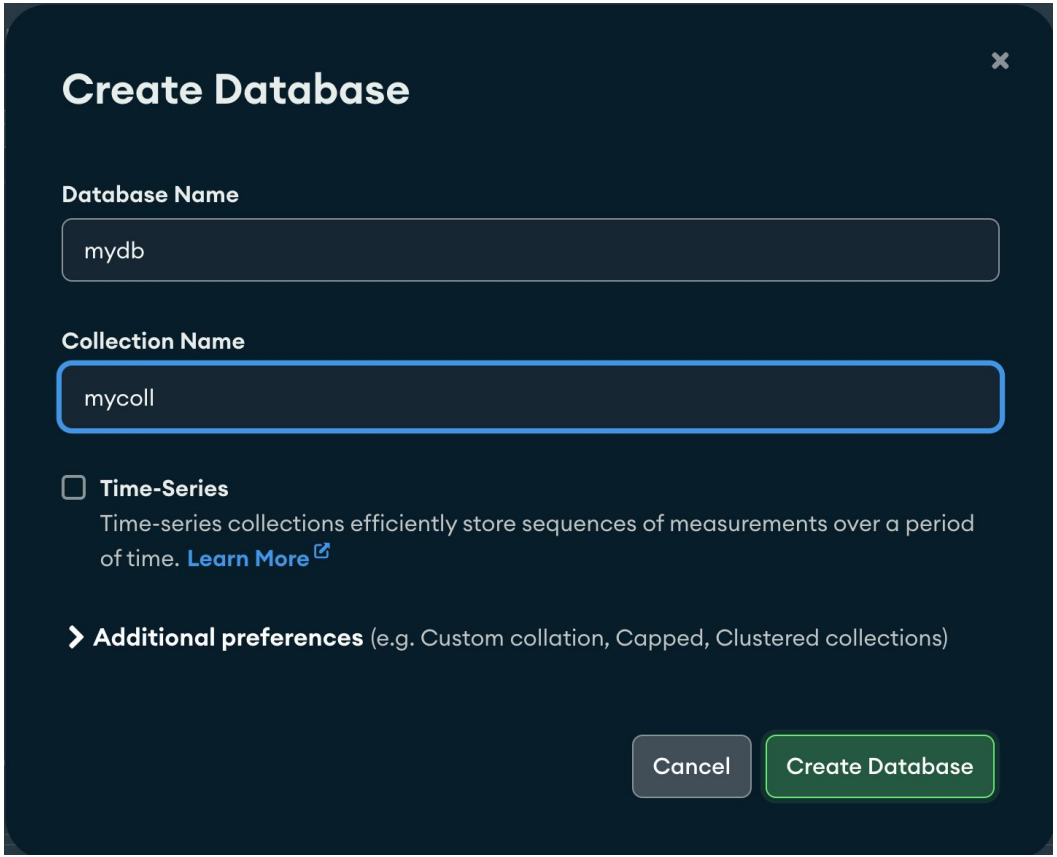
GUI

Shell



Exercise 1: CRUD





SQL-to-MQL

MQL Reference - Writes (Complete Guide)

SQL	MQL
`INSERT INTO people(user_id, age, status)` `VALUES ("bcd001", 45, "A")`	`db.people.insertOne( person )`

`UPDATE people`
`SET status = "C"`
`WHERE age > 25`
	`db.people.updateMany(`
	`  { age: { $gt: 25 } },`
	`  { $set: { status: "C" } }`
	`)`

`DELETE FROM people WHERE status = "D"`	`db.people.deleteMany({status: "D"})`

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

Exercise 1: CRUD

Insert Command Syntax

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

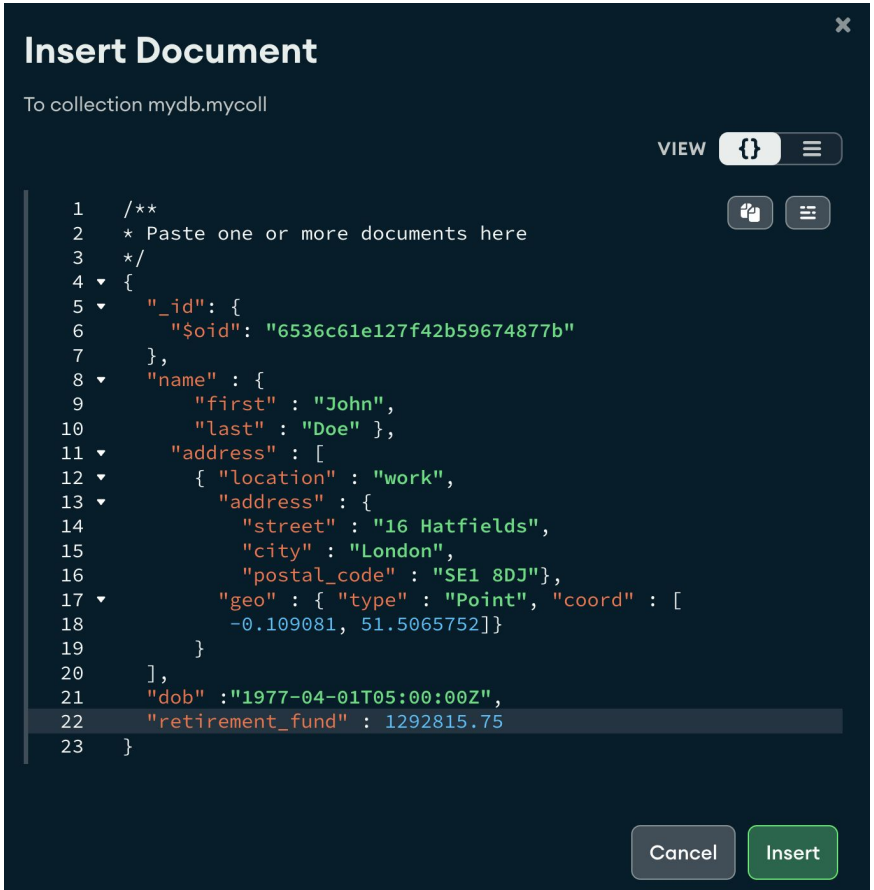
`db.<collectionName>.insertMany([{k:v},{k:v}],{options})`
InsertMany, which accepts an array of documents

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

```
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

- `$set` - adds / updates fields
- `$unset` - removes fields
- `$inc` - increments a field
- `$pull` / `$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

Exercise 3: CRUD

Delete Command Syntax

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

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

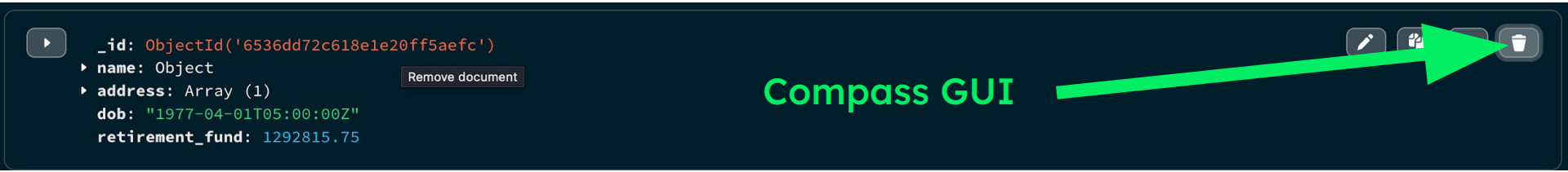
`{filter}` - Which document(s) to delete (i.e. where-clause).

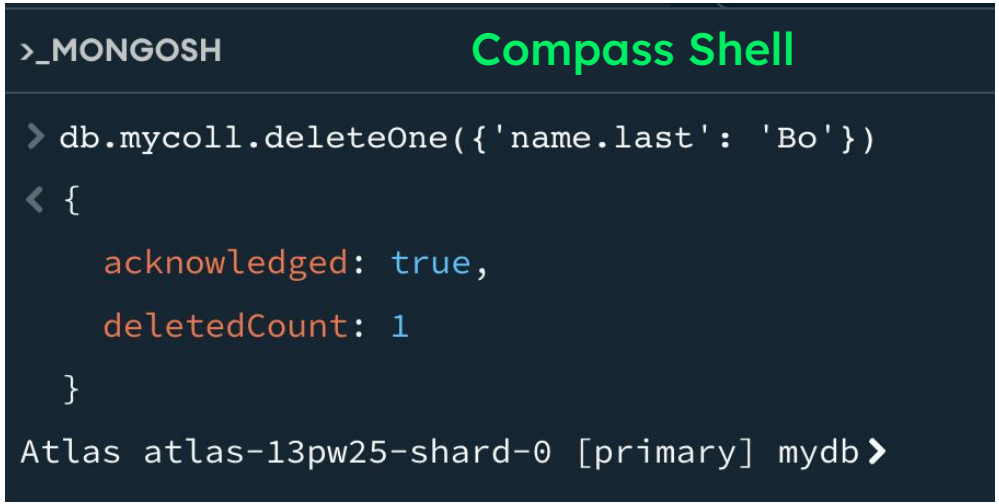
`{options}` - Options include things like write concern (a mongoDB parameter for durability / performance tradeoffs) collation (for language support) and hint (index hinting)

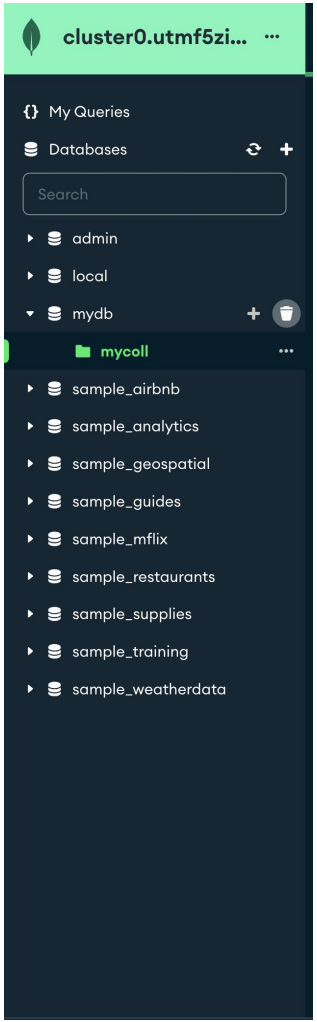
Delete this document

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

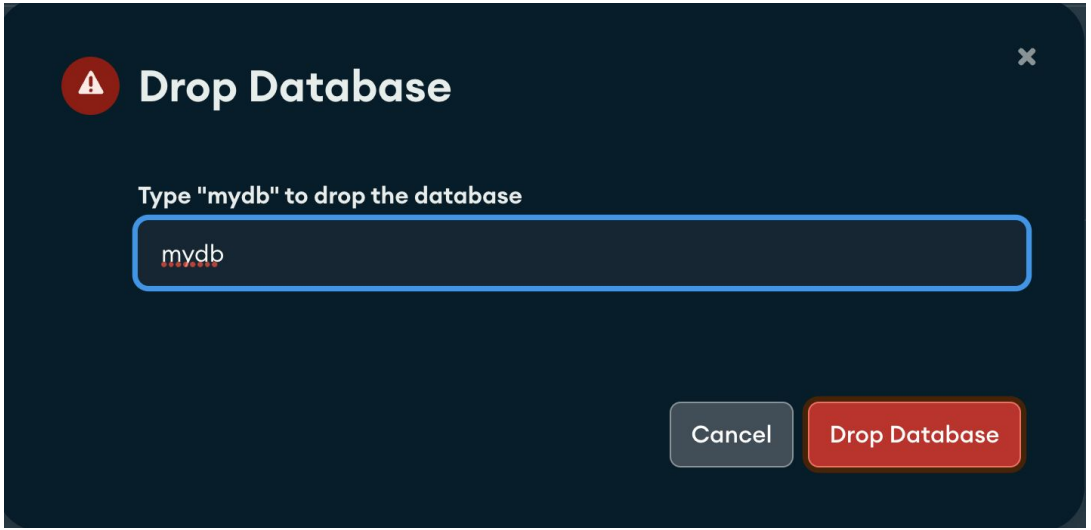
Answer 3: CRUD







Exercise 4: CRUD - Drop the database



SQL-to-MQL

MQL Reference - Reads (Complete Guide)

SQL	MQL
`SELECT * FROM people`	`db.people.find()`

`SELECT id, user_id, status`
`FROM people`
	`db.people.find(`
	`  { }`
	`).project({ user_id: 1, status: 1 })`

`SELECT * FROM people` `WHERE status = "A"`	`db.people.findOne( { status: "A" } )`
`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

- `$eq` (equals) - Similar to a normal query
- `$gt`/`$gte`/`$lt`/`$lte` - Greater [and equal] / less than [and equal]: important operator for range queries
- `$in:[]` / `$nin:[]` (in / not in): Multiple value equality match
- `$ne` - Not equal
- `$not`: Inverses the logic of a query
- `$or`: Boolean OR
- `$and`: Boolean AND. Usually used for specific nesting situations
- `$exists`: Filters based on field existence
- `$type`: Filters based on BSON type
- `$elemMatch`: 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
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:

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.

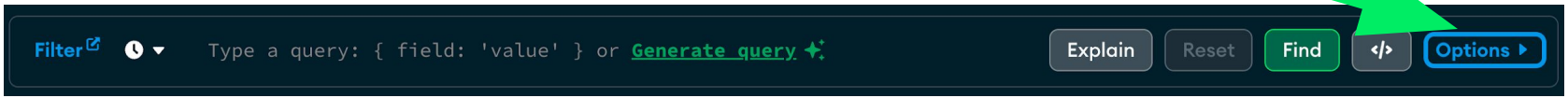
Example: db.movies.find({title: “Regeneration”}, {_id: 0, title: 1, fullplot: 1})

`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



`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

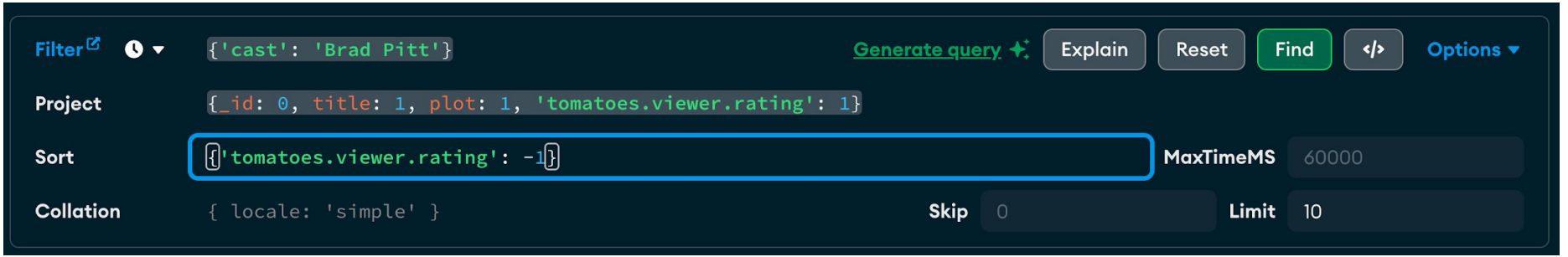
Find the top 10 movies by Rotten Tomatoes rating with Brad Pitt in it.

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

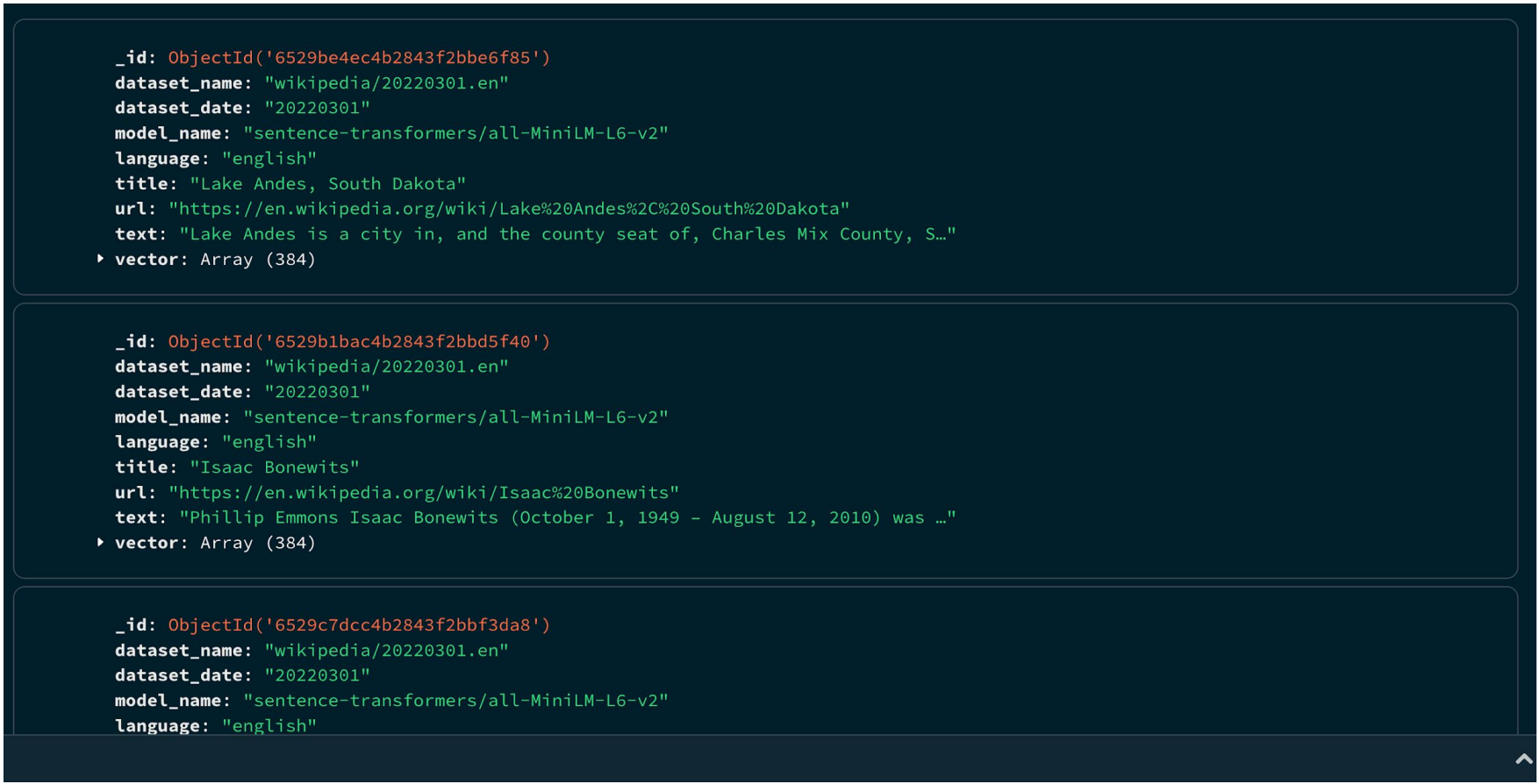


Compass Shell

```
> 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)
```

Import wikipedia_tiny.json via Compass

Wikipedia Collection Schema



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

Index features

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

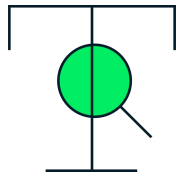
Case-insensitive indexes

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

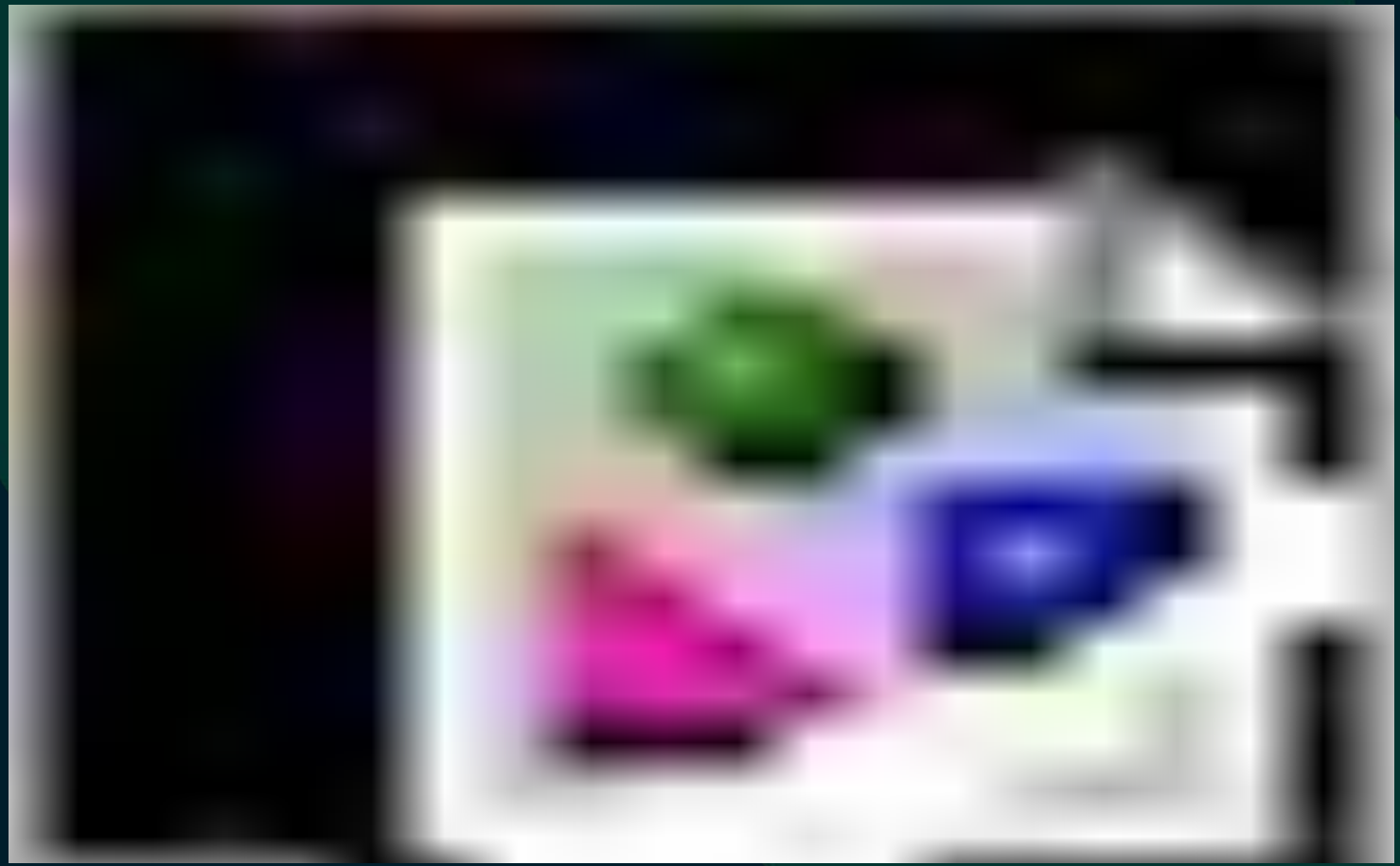
Indexing

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

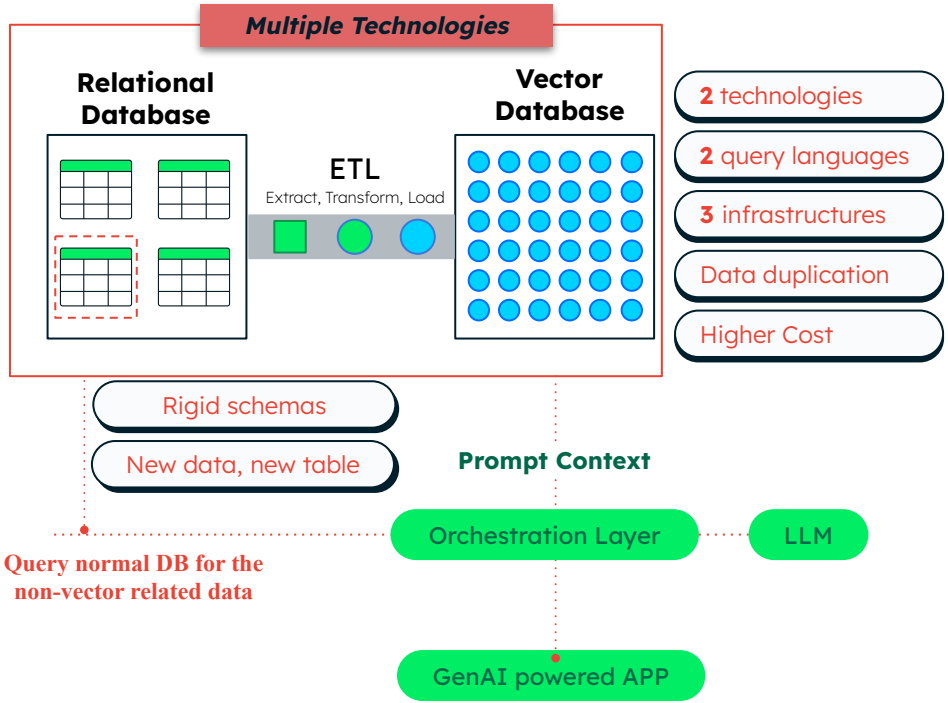
Querying

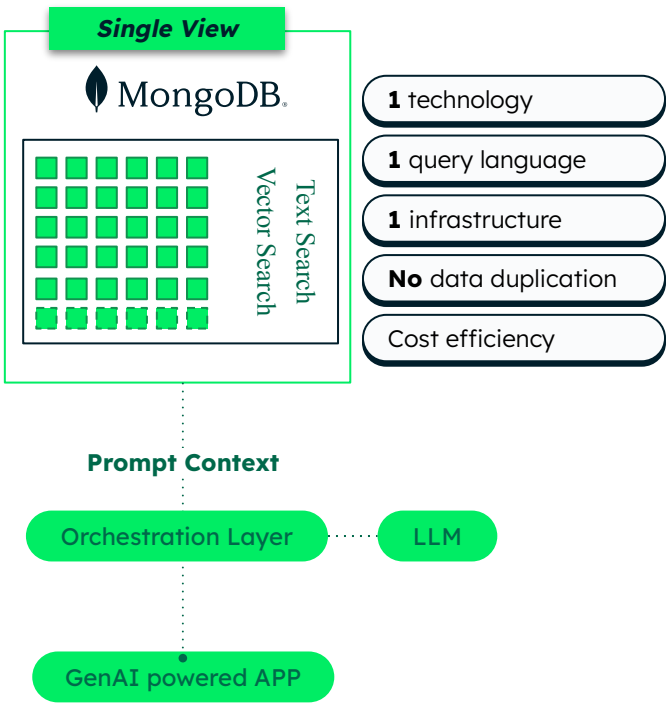
- Rich Query
- **Fuzzy Matching**
- Scoring
- Autocomplete
- Faceting / Filtering
- Highlighting
- Synonyms
- moreLikeThis
- **Semantic search**

Create Vector Search Index in Atlas UI

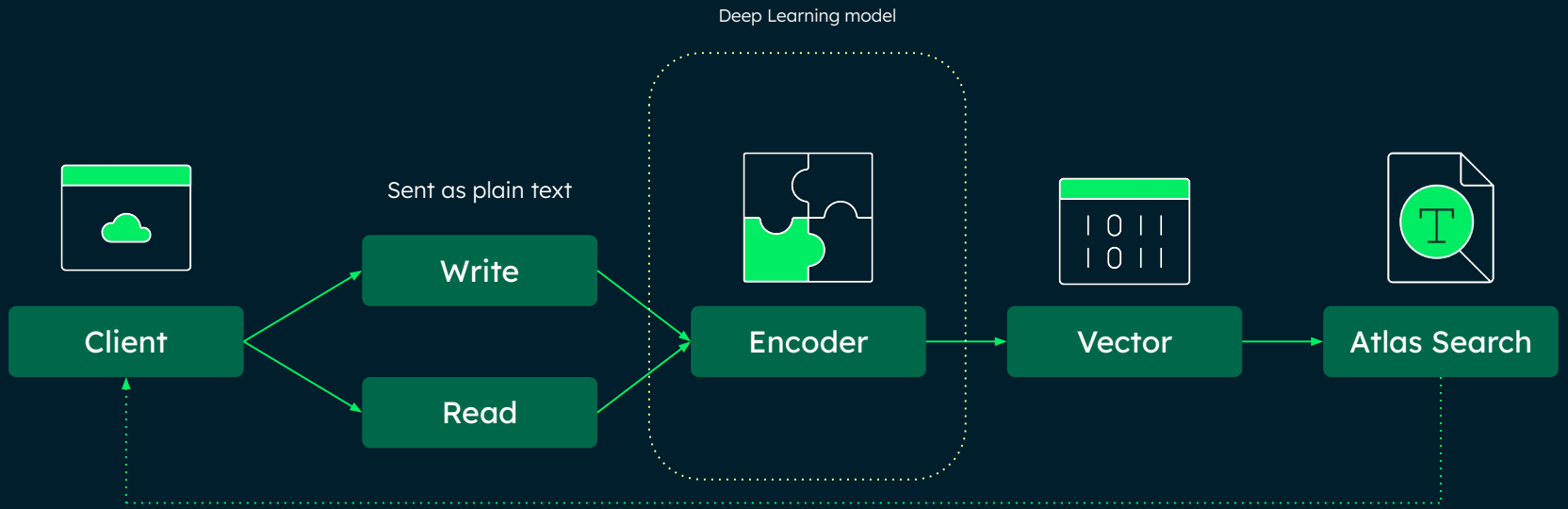


Atlas Vector Search





Semantic Search Data Workflow



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

Vector Search Index Definition

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

Semantic Search

Back in the Compass Shell, perform these sequentially:

```
Use devday

let vector = <vector> //vector can be found in the workshop.txt file

let agg = [{
  $vectorSearch: {
    queryVector: vector,
    Path: 'vector',
    numCandidates: 50,
    index: 'vector',
    limit: 10,
  }
}]

db.wikipedia.aggregate(agg)
```

Semantic Search with Filter

Modify the previous query to include the filter component in green below:

```
agg = [{
      $vectorSearch: {
        queryVector: vector,
        Path: ‘vector’,
        numCandidates: 50,
        index: ‘vector’,
        limit: 10,
        filter: {
          $and: [{
           'language': {$eq:'english'}
          }]
        }
      }
    }]

    db.wikipedia.aggregate(agg)
```

Thank you