

# Practice 4: MongoDB Basics

Big Data System Design

Part 1

# PRACTICE

# Database

## ❖ Database is a physical container for collections

- Commands
  - use DATABASE\_NAME
  - show dbs
  - db.dropDatabase()

## ❖ Practice 1

- **show dbs**

## ❖ Practice 2

- **use blogging**

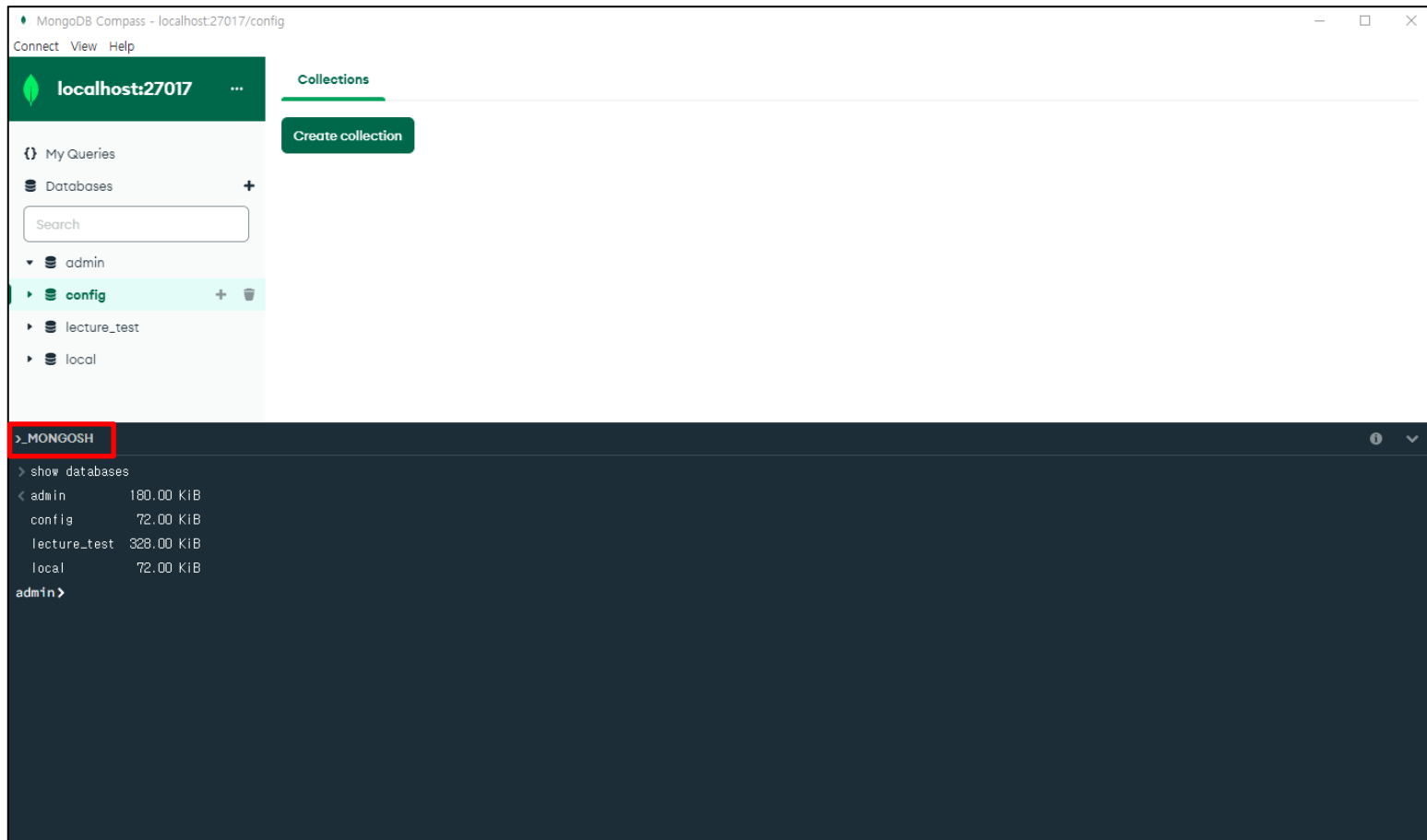
## ❖ Practice 3

- **db**

# Database

## ❖ MongoDB Compass

- You can use Compass Shell (Mongosh) in the lower left corner



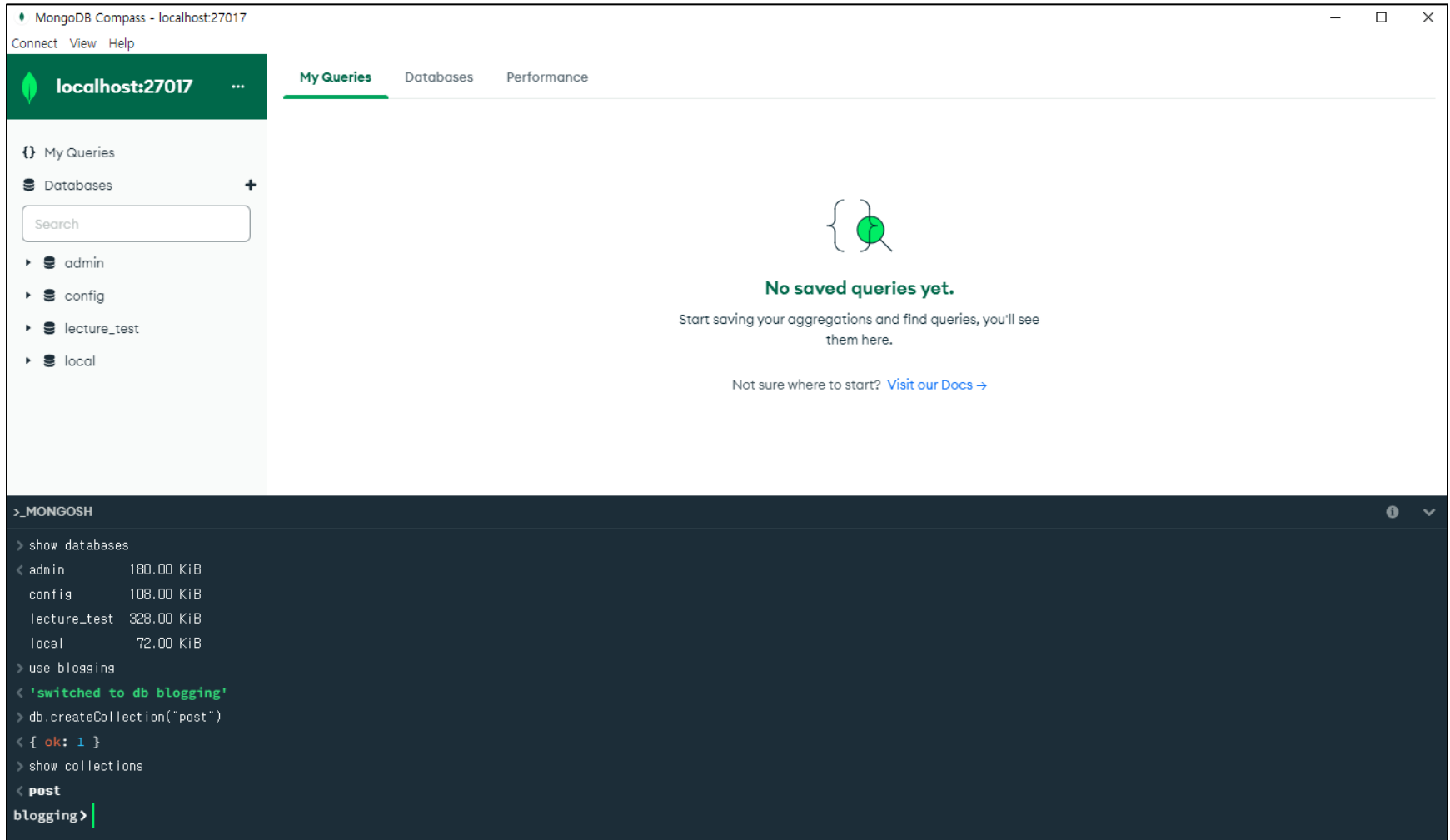
# Collection

❖ Collection is a group of MongoDB documents similar to tables of RDBMS

- Commands
  - `db.createCollection(name, options)`
  - `show collections`
  - `db.COLLECTION_NAME.drop()`
- Practice 4
  - **`show collections`**
- Practice 5
  - **`db.createCollection("post")`**

# Collection

## ❖ Result of Practice 4 and Practice 5

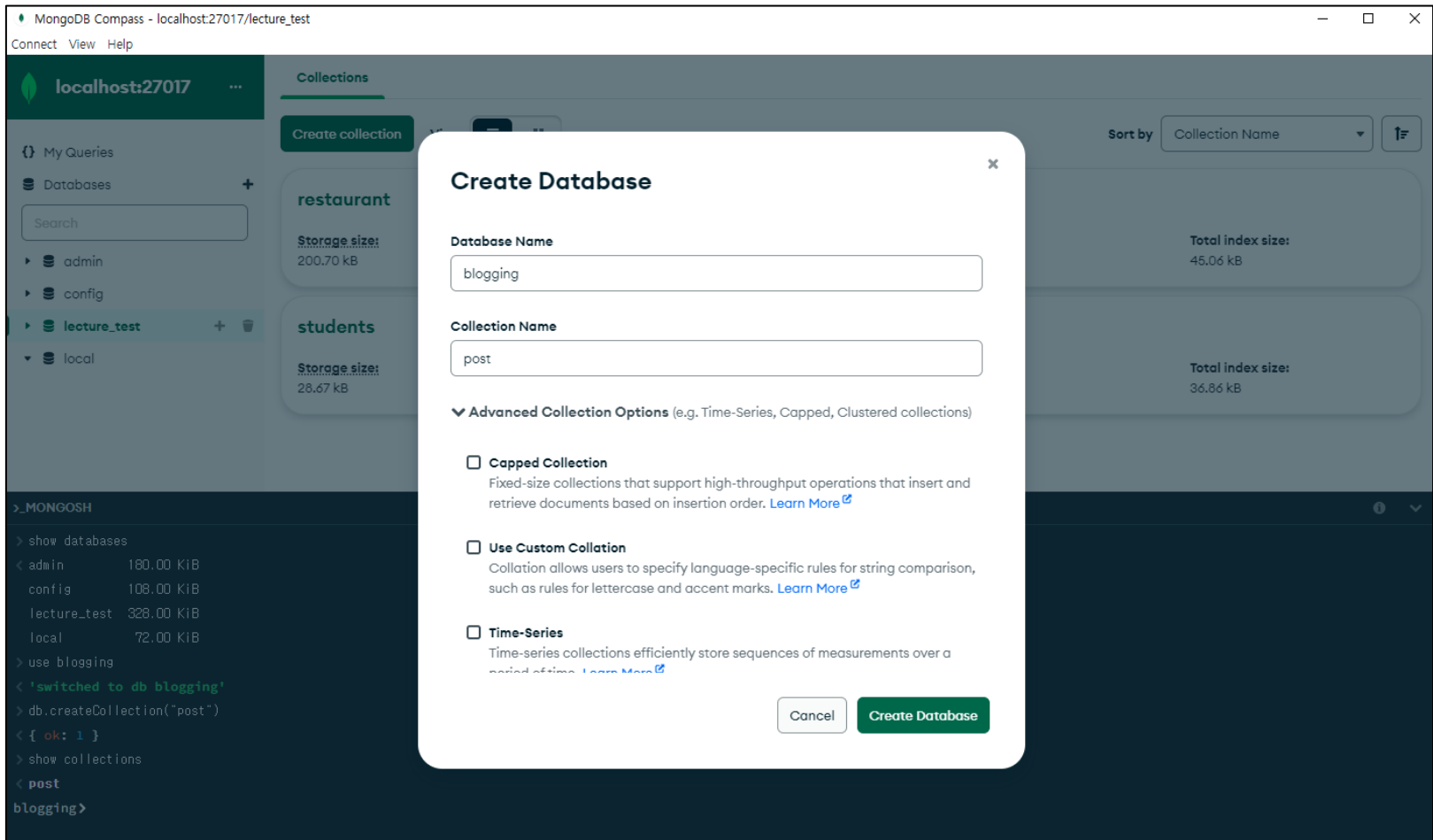


The screenshot displays the MongoDB Compass application window. The title bar reads "MongoDB Compass - localhost:27017". The top navigation bar includes "Connect", "View", and "Help". The left sidebar shows the connection "localhost:27017" and a list of databases: "admin", "config", "lecture\_test", and "local". The main panel is titled "My Queries" and shows a message: "No saved queries yet. Start saving your aggregations and find queries, you'll see them here." Below this message is a link: "Not sure where to start? [Visit our Docs](#) →". At the bottom, a terminal window titled ">\_MONGOSH" shows the following commands and output:

```
> show databases
< admin      180.00 KIB
  config     108.00 KIB
  lecture_test 328.00 KIB
  local       72.00 KIB
> use blogging
< 'switched to db blogging'
> db.createCollection("post")
< { ok: 1 }
> show collections
< post
blogging>
```

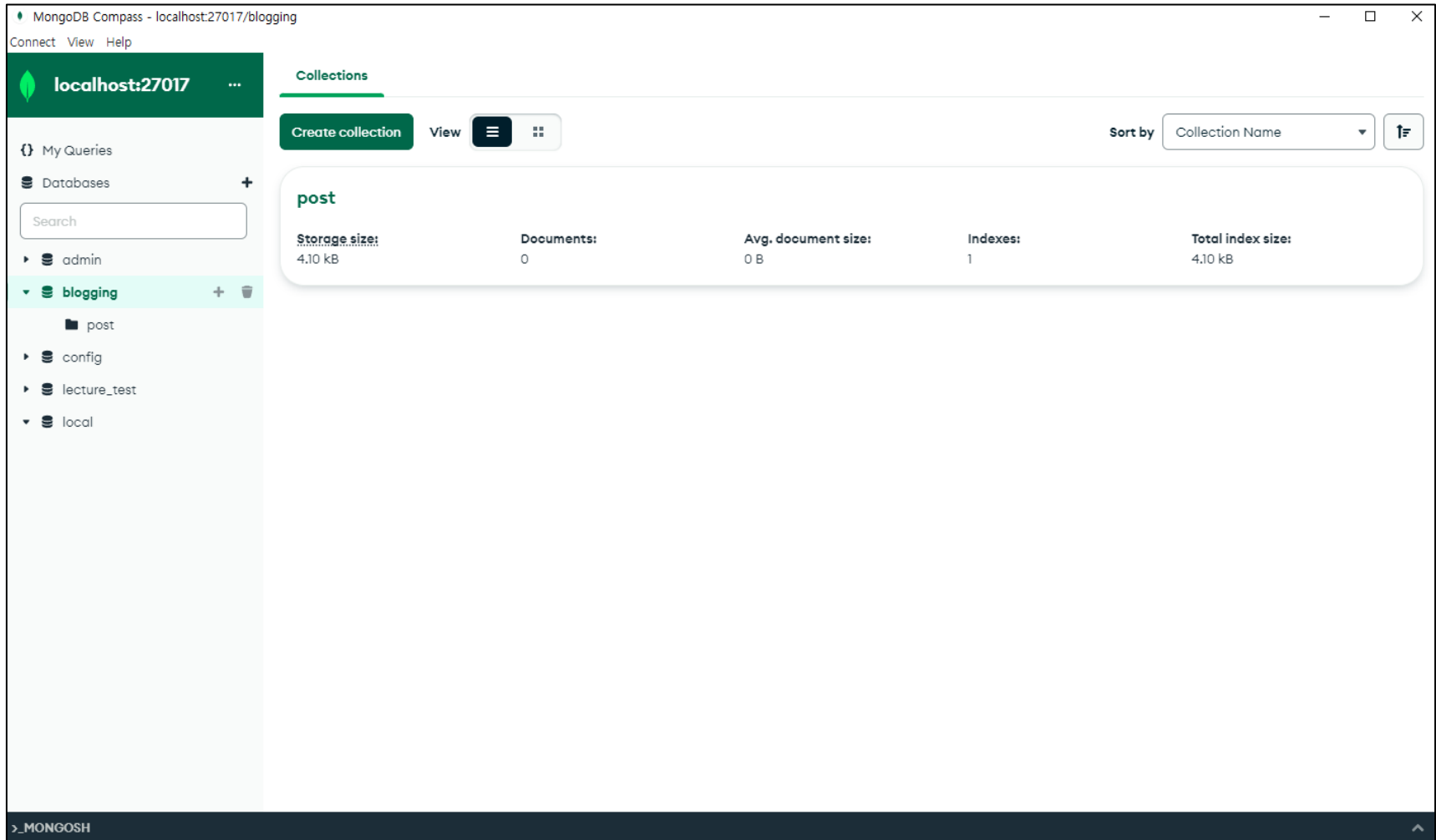
# Collection

## ❖ Creating database and collection in MongoDB Compass



# Collection

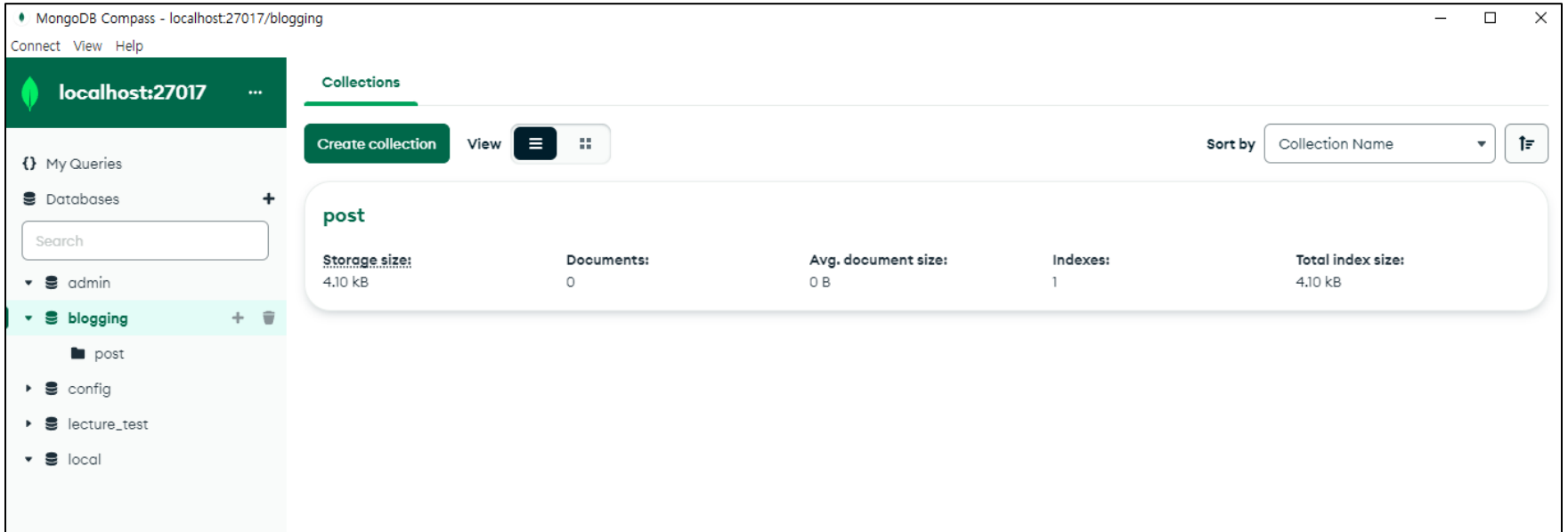
❖ New database with POST collection is created





# Collection

❖ New TAG collection is created by inserting a document



The screenshot shows the MongoDB Compass interface. On the left sidebar, the 'blogging' database is selected, showing a collection named 'post'. The main panel displays the 'Collections' tab for the 'blogging' database, showing the 'post' collection with the following statistics:

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
4.10 kB	0	0 B	1	4.10 kB

At the bottom, the MongoDB Shell (MONGOSH) is open, showing the following commands and output:

```
>_MONGOSH
> show collections
< post
> db.post.drop()
< true
> db.tag.insertOne({"tag": "Big data"})
< {
  acknowledged: true,
  insertedId: ObjectId("642279cac74ae345123ac1d0")
}
> show collections
< post
  tag
blogging>
```

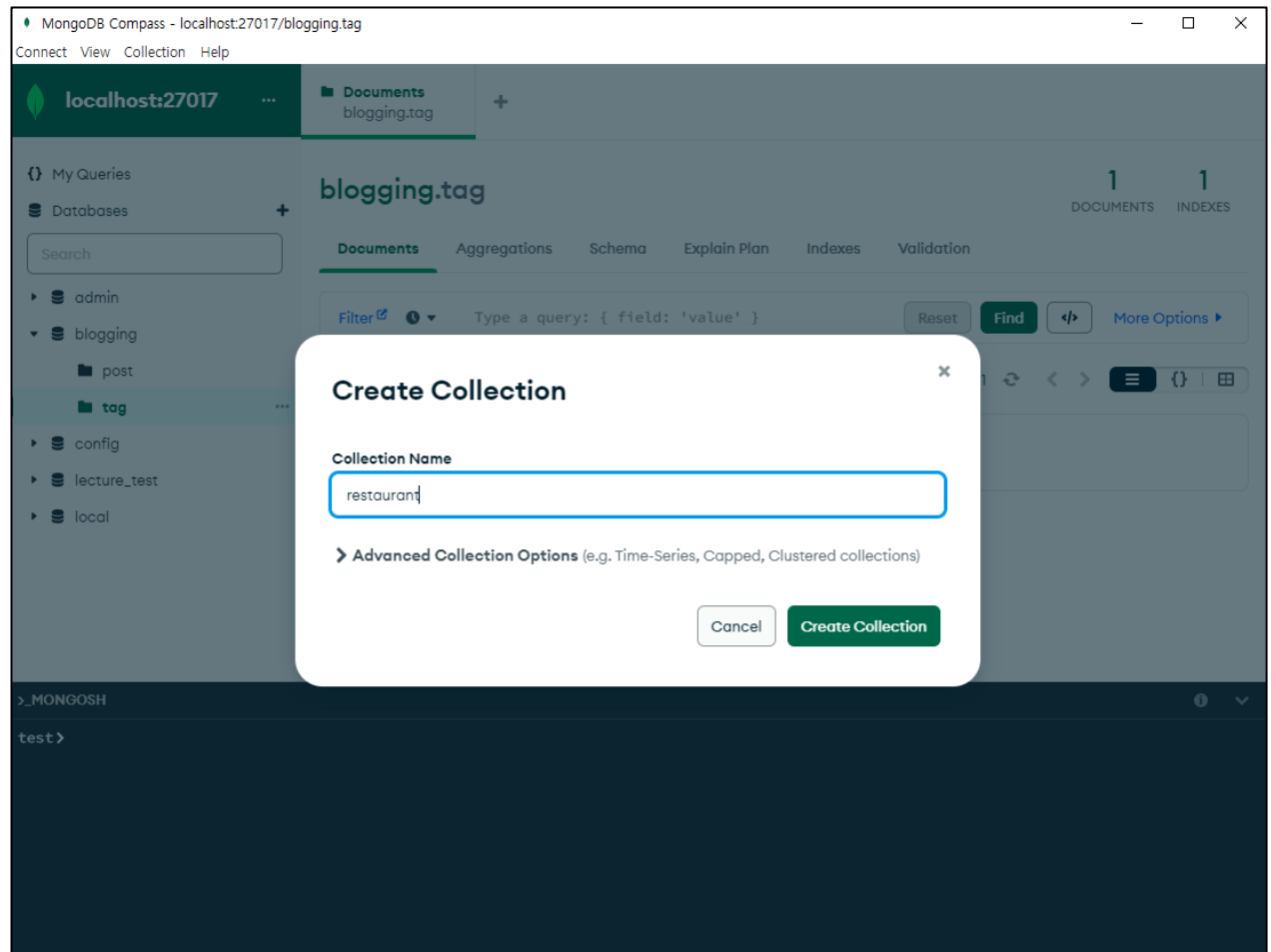
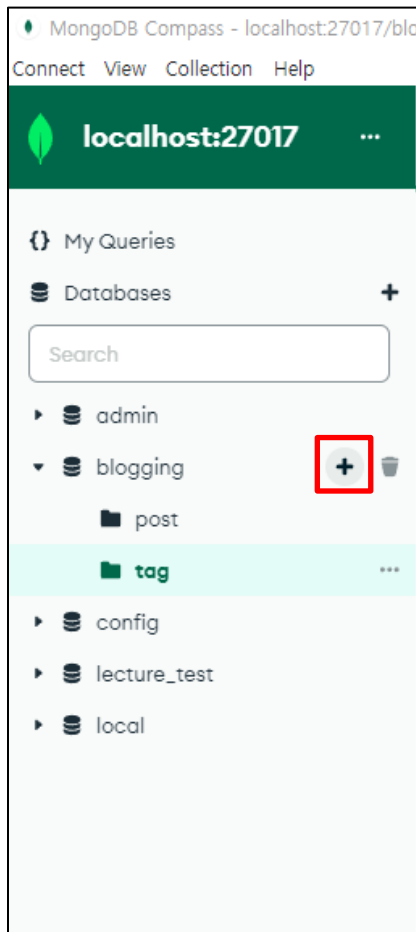
# Collection

## ❖ Explore your schema

- You can easily visualize your schema to understand the frequency, types and ranges of fields in your data
- Practice 6
  - ① Create new 'restaurant' collection
  - ② Download 'restaurant' JSON dataset from eCampus
  - ③ Add JSON data on 'restaurant' collection
  - ④ Click 'schema' option and run analysis
  - ⑤ Check the results

# Collection

## ❖ Explore your schema



# Collection

## ❖ Explore your schema

성공: restaurant JSON dataset 이(가) 생성되었습니다.

Lecture 4 - MongoDB Overview

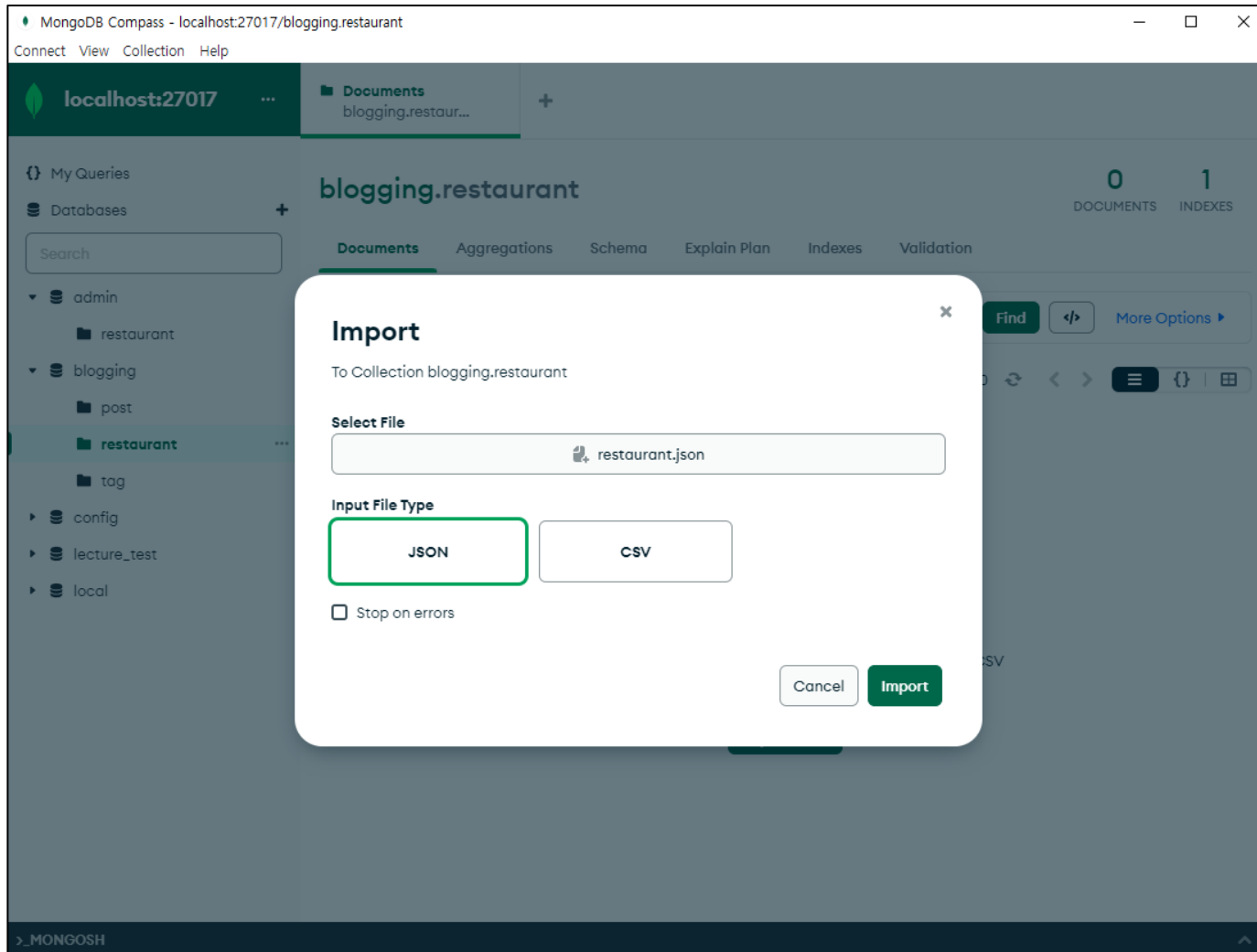
콘텐츠 생성 ▼ 평가 ▼ 도구 ▼ 파트너 콘텐츠 ▼

**MongoDB Overview** ▼  
설정: 통계 추적

**restaurant JSON dataset** ▼  
설정: 통계 추적

# Collection

## ❖ Explore your schema



# Collection

## ❖ Explore your schema

The screenshot shows the MongoDB Compass interface for the `blogging.restaurant` collection. The `Schema` tab is selected and highlighted with a red box. The interface displays the collection's schema, including fields like `_id`, `URL`, `address`, `address line 2`, `name`, `outcode`, `postcode`, `rating`, and `type_of_food`. The `rating` field is shown as a float (5.5), and the `type_of_food` field is shown as a string ("Chinese").

localhost:27017 ... Documents blogging.restaur...

My Queries Databases Search

admin restaurant blogging post restaurant tag config lecture\_test local

blogging.restaurant 2.5k 1 DOCUMENTS INDEXES

Documents Aggregations Schema Explain Plan Indexes Validation

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

ADD DATA EXPORT COLLECTION 1 - 20 of 2548

```
{
  "_id": ObjectId('55f14312c7447c3da7051b26'),
  "URL": "http://www.just-eat.co.uk/restaurants-cn-chinese-cardiff/menu",
  "address": "228 City Road",
  "address line 2": "Cardiff",
  "name": ".CN Chinese",
  "outcode": "CF24",
  "postcode": "3JH",
  "rating": 5,
  "type_of_food": "Chinese"
}
```

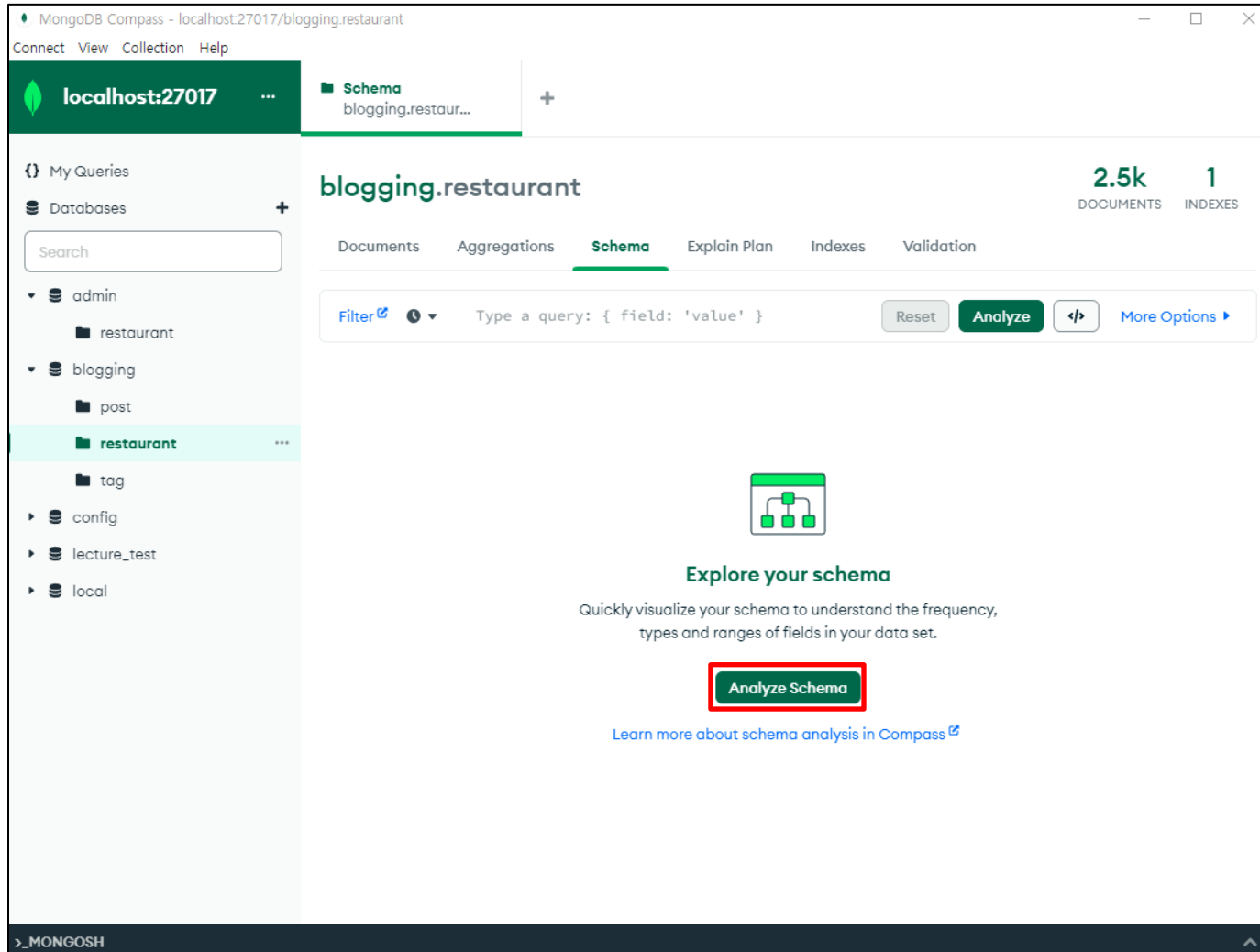
```
{
  "_id": ObjectId('55f14312c7447c3da7051b27'),
  "URL": "http://www.just-eat.co.uk/restaurants-atthai-ss9/menu",
  "address": "376 Rayleigh Road",
  "address line 2": "Essex",
  "name": "@ Thai",
  "outcode": "SS9",
  "postcode": "SPT",
  "rating": 5.5,
  "type_of_food": "Thai"
}
```

```
{
  "_id": ObjectId('55f14312c7447c3da7051b28'),
  "URL": "http://www.just-eat.co.uk/restaurants-atthairestaurant/menu",
  "address": "376 Rayleigh Road, Homersfield",
  "address line 2": "Essex",
  "name": "@ Thai",
  "outcode": "SS9",
  "postcode": "SPT",
  "rating": 5.5,
  "type_of_food": "Thai"
}
```

>\_MONGOSH

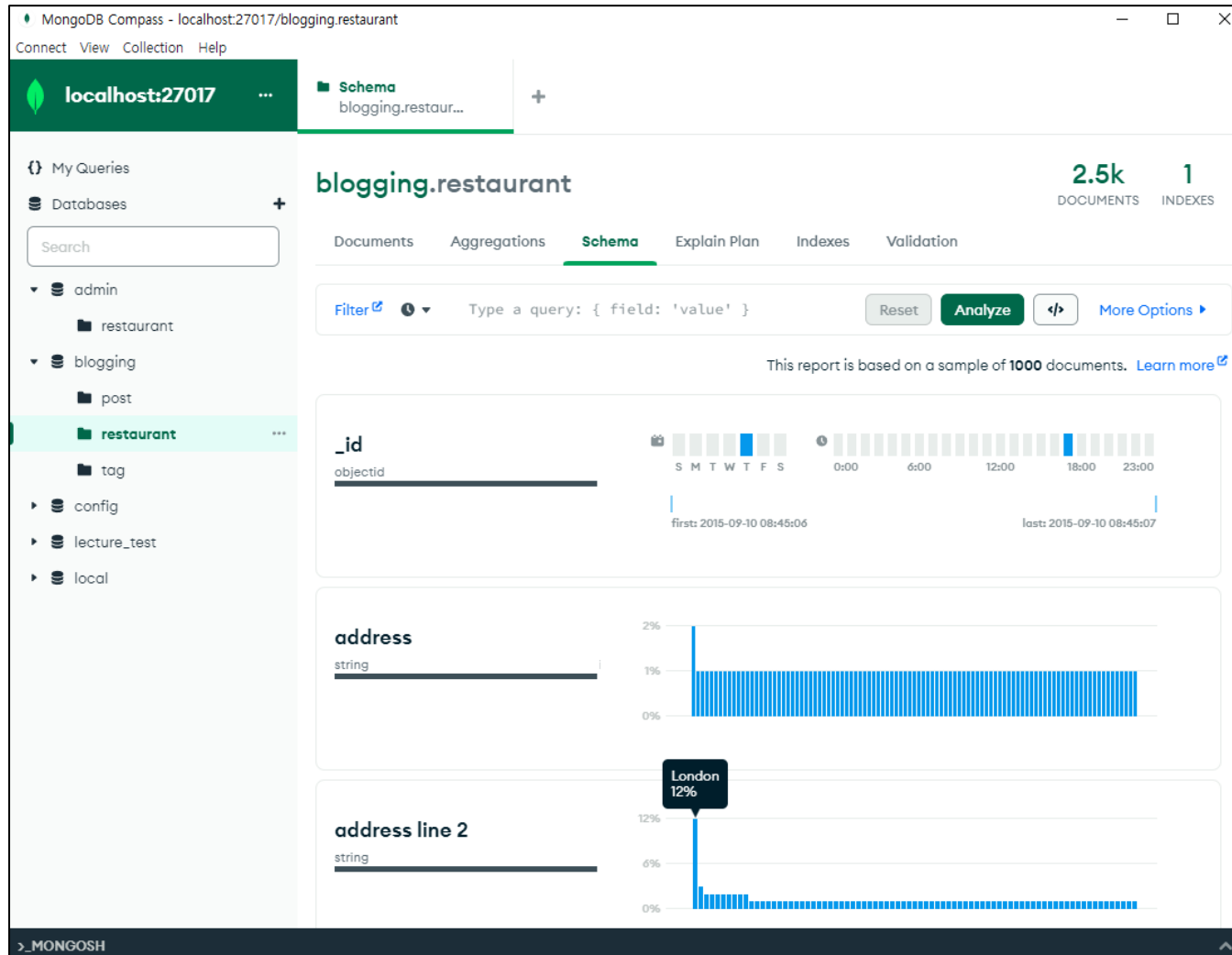
# Collection

## ❖ Explore your schema



# Collection

## ❖ Explore your schema

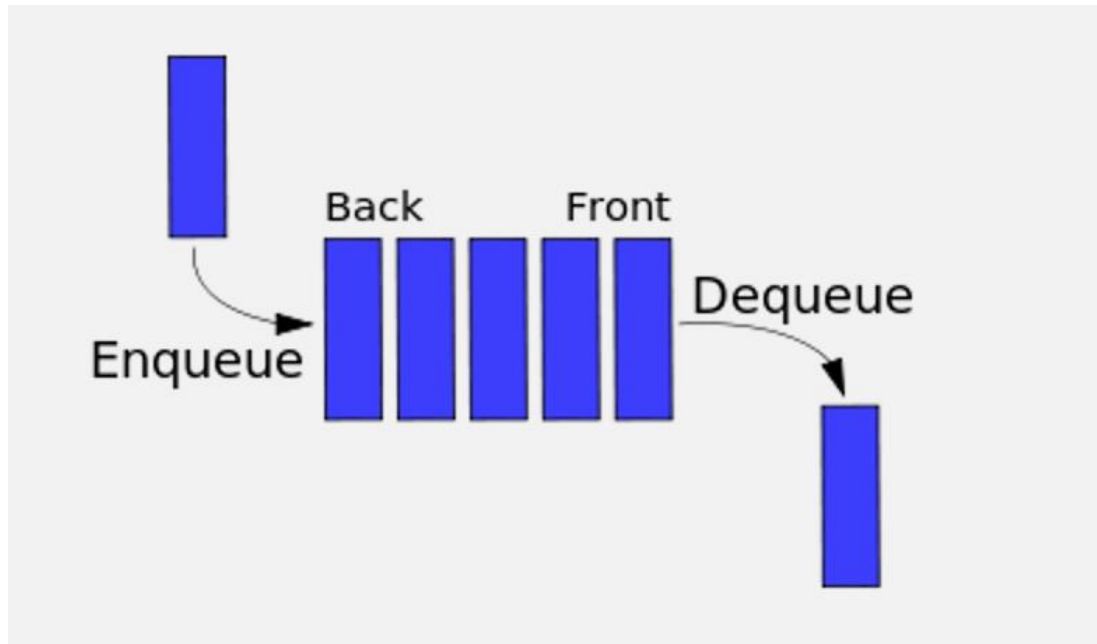




# Collection

## ❖ Capped collection

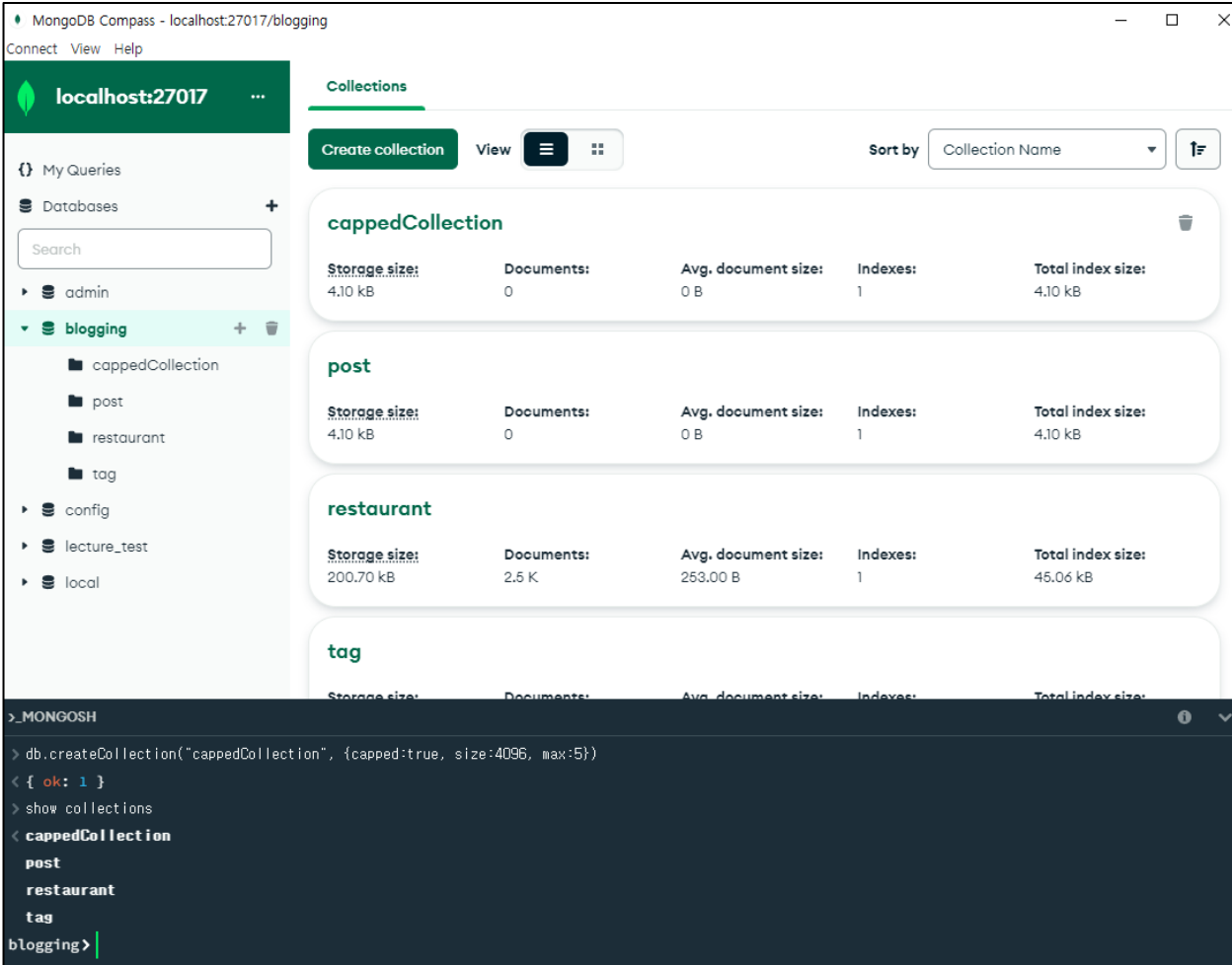
- To create a capped collection (Practice 7)
  - **`db.createCollection("cappedCollection",{capped:true, size:4096, max: 5})`**



# Collection

## ❖ Capped collection

- You can create a capped collection from Mongo Shell



The screenshot shows the MongoDB Compass interface for the 'localhost:27017/blog' database. The left sidebar shows the database structure with 'blogging' selected. The main panel displays a list of collections with their respective statistics.

Collection Name	Storage size	Documents	Avg. document size	Indexes	Total index size
cappedCollection	4.10 kB	0	0 B	1	4.10 kB
post	4.10 kB	0	0 B	1	4.10 kB
restaurant	200.70 kB	2.5 K	253.00 B	1	45.06 kB
tag					

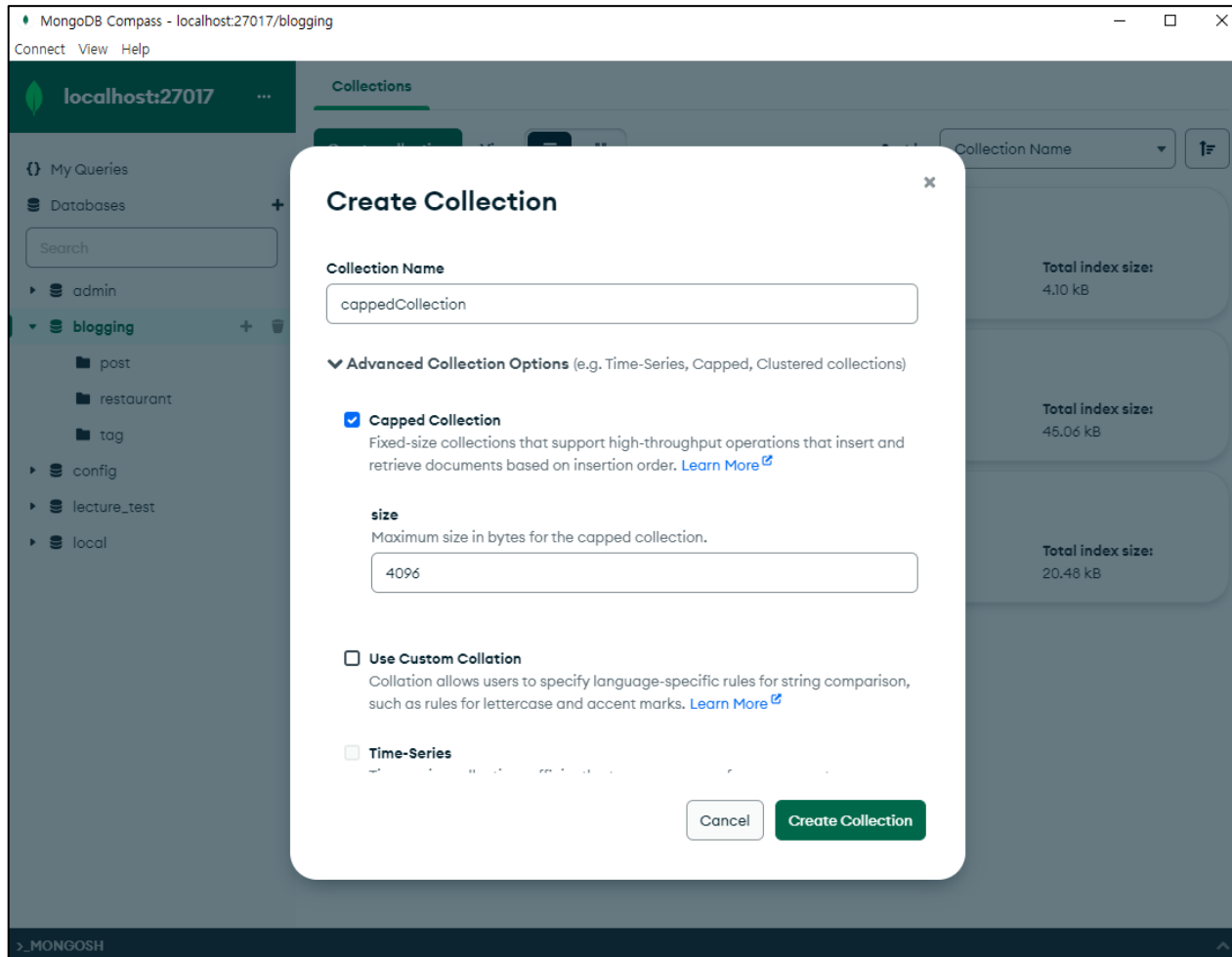
Below the screenshot, the MongoDB Shell command to create a capped collection is shown:

```
> MONGOSH
> db.createCollection("cappedCollection", {capped:true, size:4096, max:5})
< { ok: 1 }
> show collections
< cappedCollection
  post
  restaurant
  tag
blogging>
```

# Collection

## ❖ Capped collection

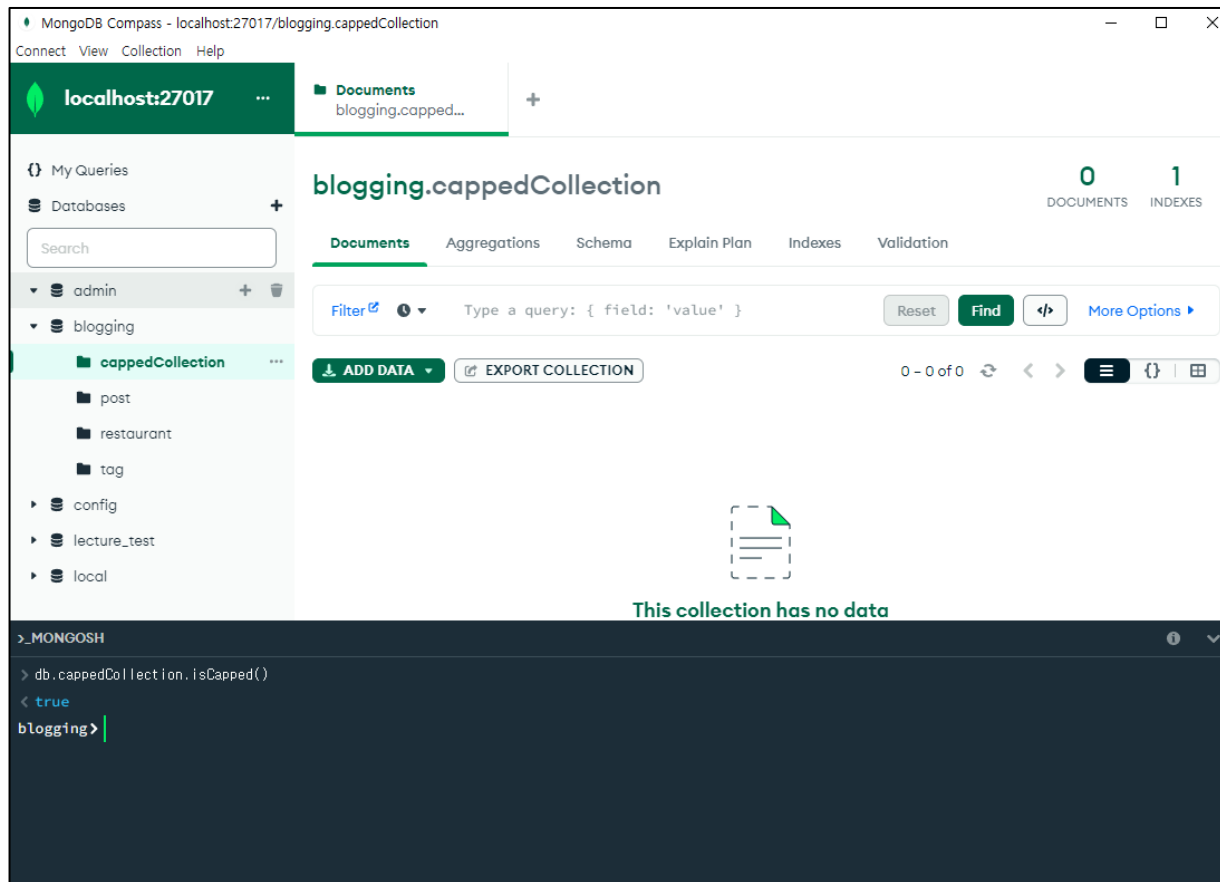
- You can also use Compass GUI for creating the capped collections



# Collection

## ❖ Capped collection

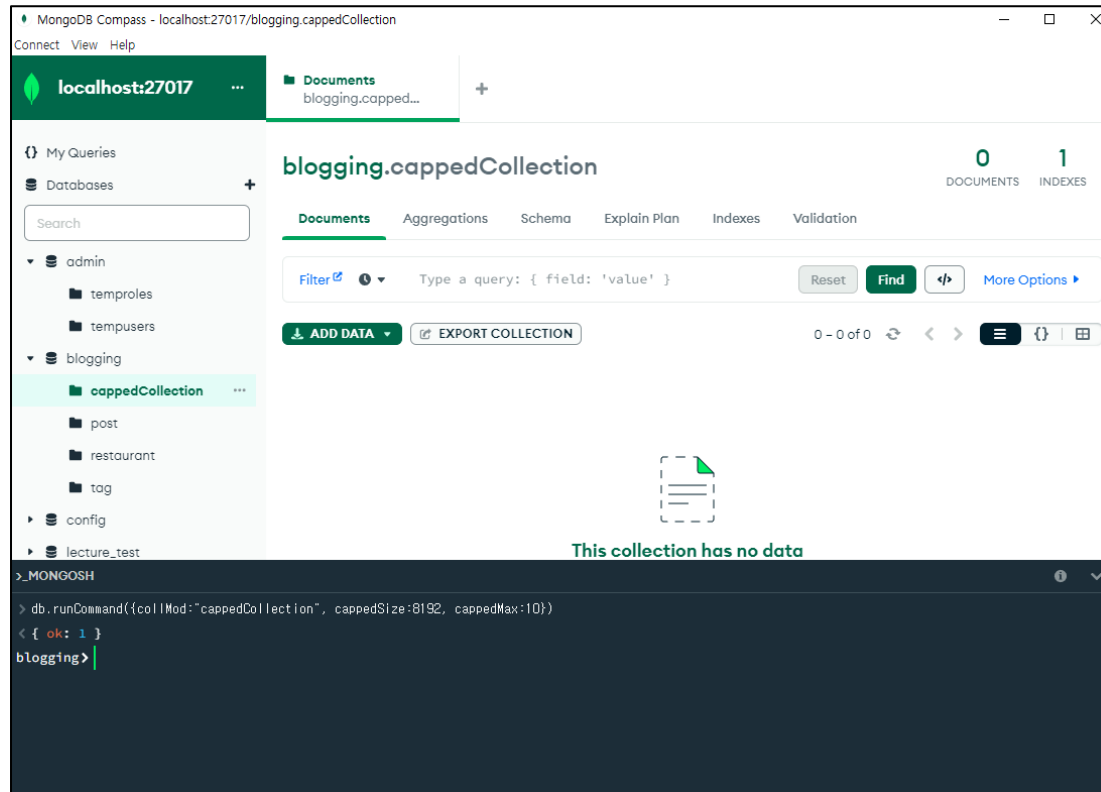
- Check capped collection
  - **db.cappedCollection.isCapped()**



# Collection

## ❖ Capped collection

- Change options of a capped collection
  - **`db.runCommand({collMod:"cappedCollection", cappedSize: 8192, cappedMax: 10})`**



# Document

❖ A document is a set of key-value pairs

- Commands

- insertOne()
- insertMany()

- Practice 8

- **db.post.insertOne({  
 title: 'Why Does Data Science Need to Be Successful?',  
 description: 'Big Data challenges',  
 by: 'Sherry Tiao',  
 url: 'https://blogs.oracle.com/bigdata/data-science-successful',  
 likes: 3  
})**

# Document

## ❖ Result of Practice 8

The screenshot displays the MongoDB Compass interface. The top bar shows the connection to 'localhost:27017' and the selected database 'blogging.post'. The left sidebar lists the databases and collections, with 'blogging.post' selected. The main panel shows the 'blogging.post' collection with 0 documents and 1 index. The 'Documents' tab is active, displaying a single document with the following fields:

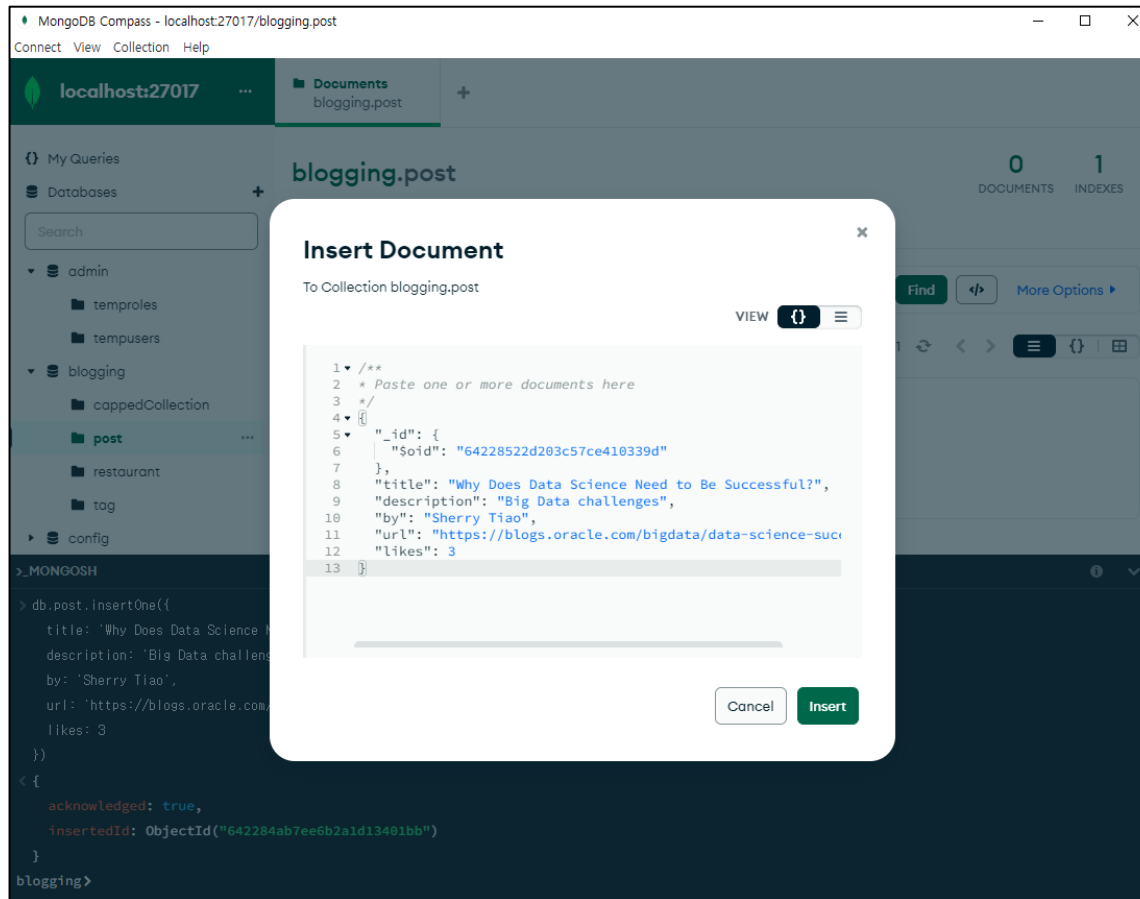
```
{
  "_id": ObjectId('642284ab7ee6b2a1d13401bb'),
  "title": "Why Does Data Science Need to Be Successful?",
  "description": "Big Data challenges",
  "by": "Sherry Tiao",
  "url": "https://blogs.oracle.com/bigdata/data-science-successful",
  "likes": 3
}
```

Below the document, the MongoDB shell output is visible, showing the successful insertion of the document:

```
> db.post.insertOne({
  title: 'Why Does Data Science Need to Be Successful?',
  description: 'Big Data challenges',
  by: 'Sherry Tiao',
  url: 'https://blogs.oracle.com/bigdata/data-science-successful',
  likes: 3
})
< {
  acknowledged: true,
  insertedId: ObjectId("642284ab7ee6b2a1d13401bb")
}
blogging>
```

# Document

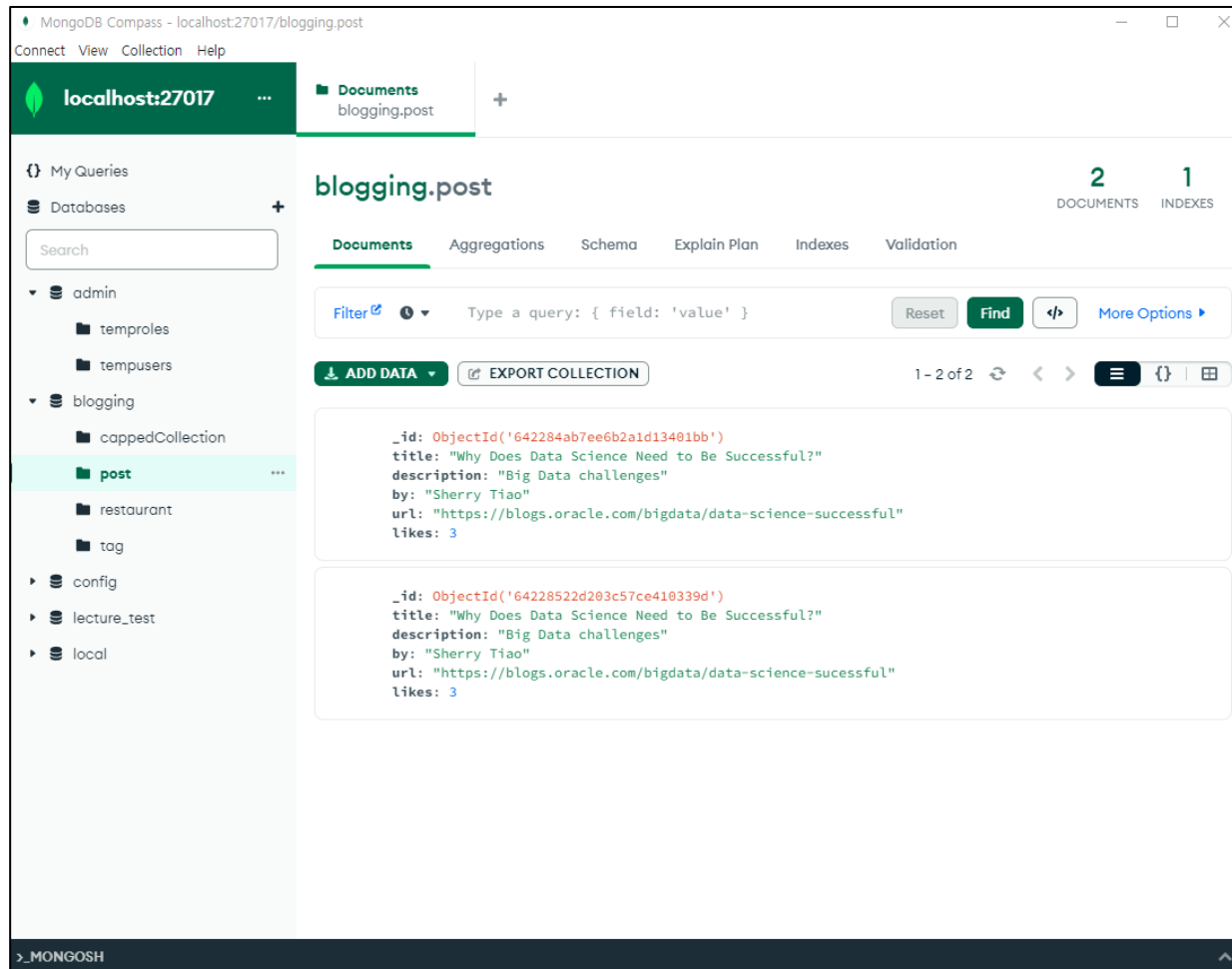
- ❖ Use Compass GUI to insert the document
  - ADD DATA -> Insert Document
    - NOTE: Pay attention to quotations





# Document

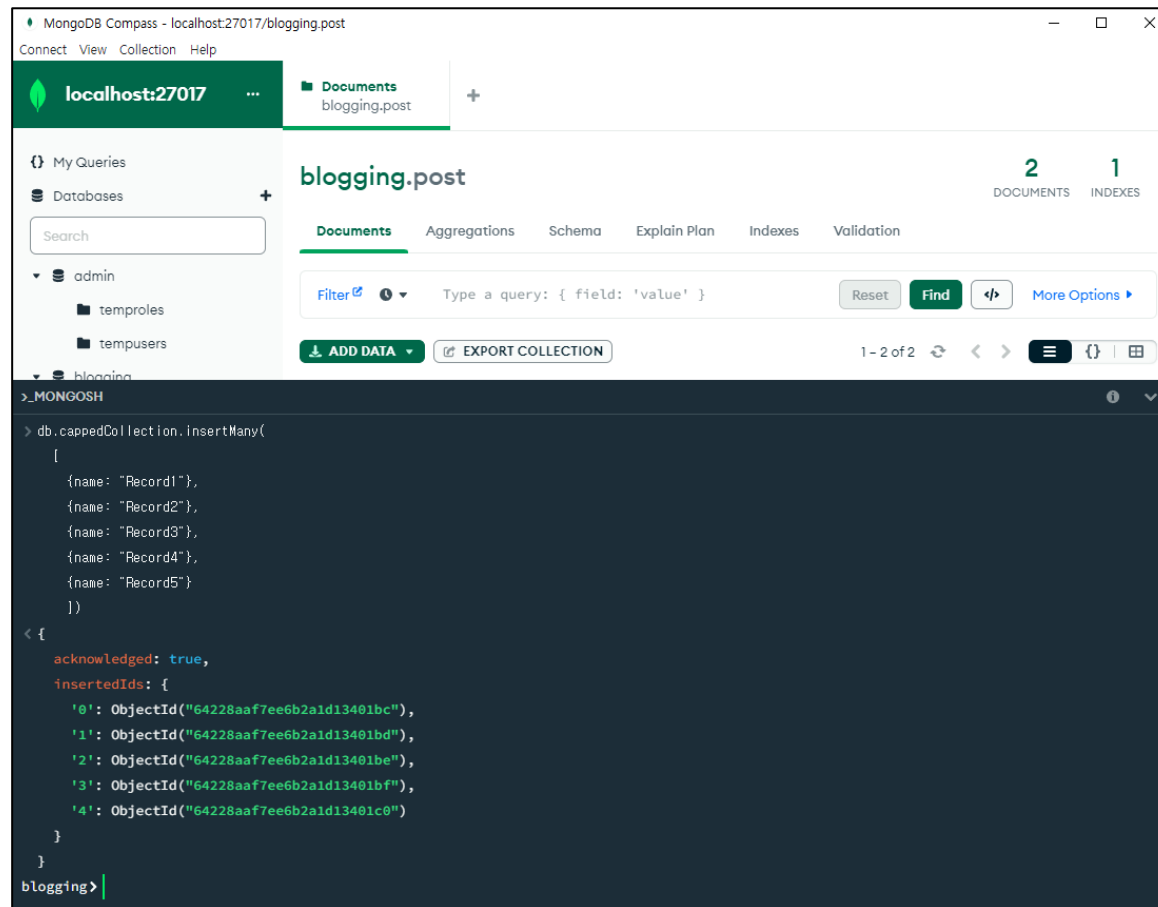
- ❖ Use Compass GUI to insert the document
  - Check out the result



# Document

## ❖ insertMany()

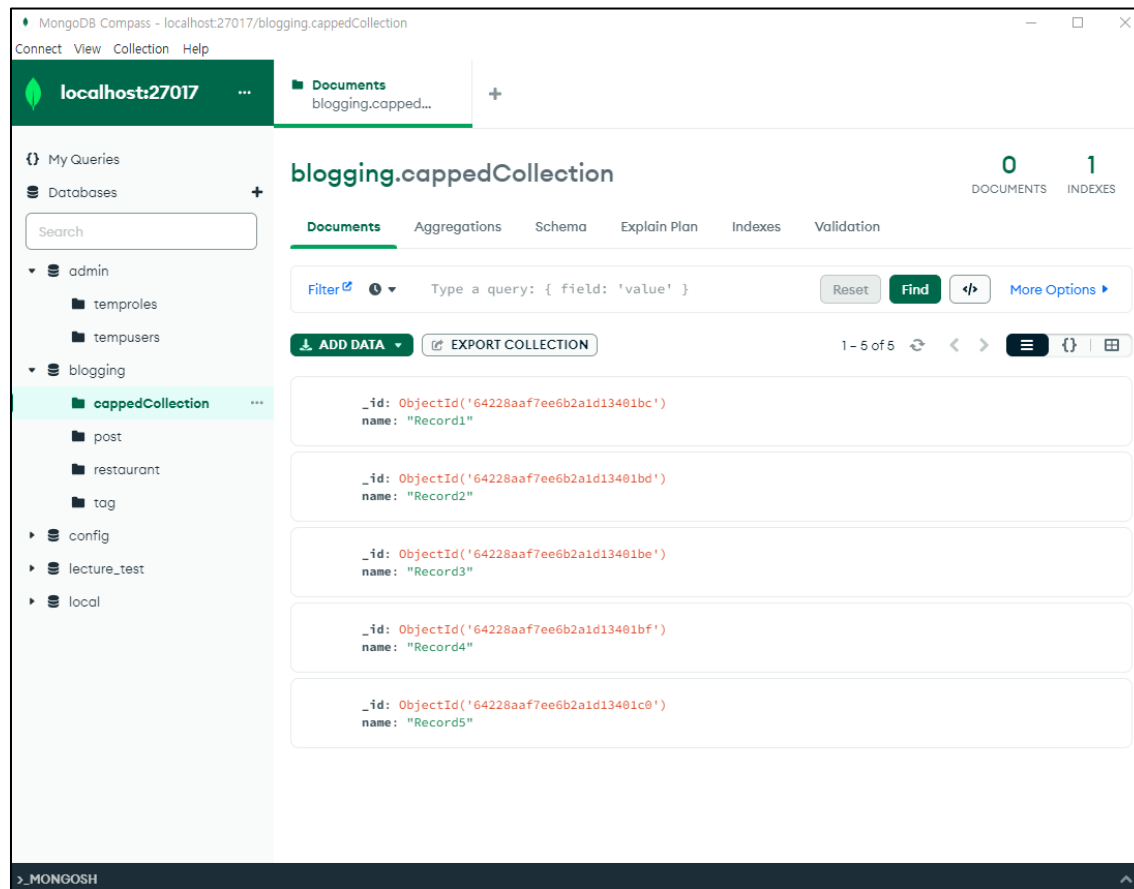
- As the name suggests, you can use insertMany() to insert multiple documents



# Document

## ❖ insertMany()

- As the name suggests, you can use insertMany() to insert multiple documents



# Document

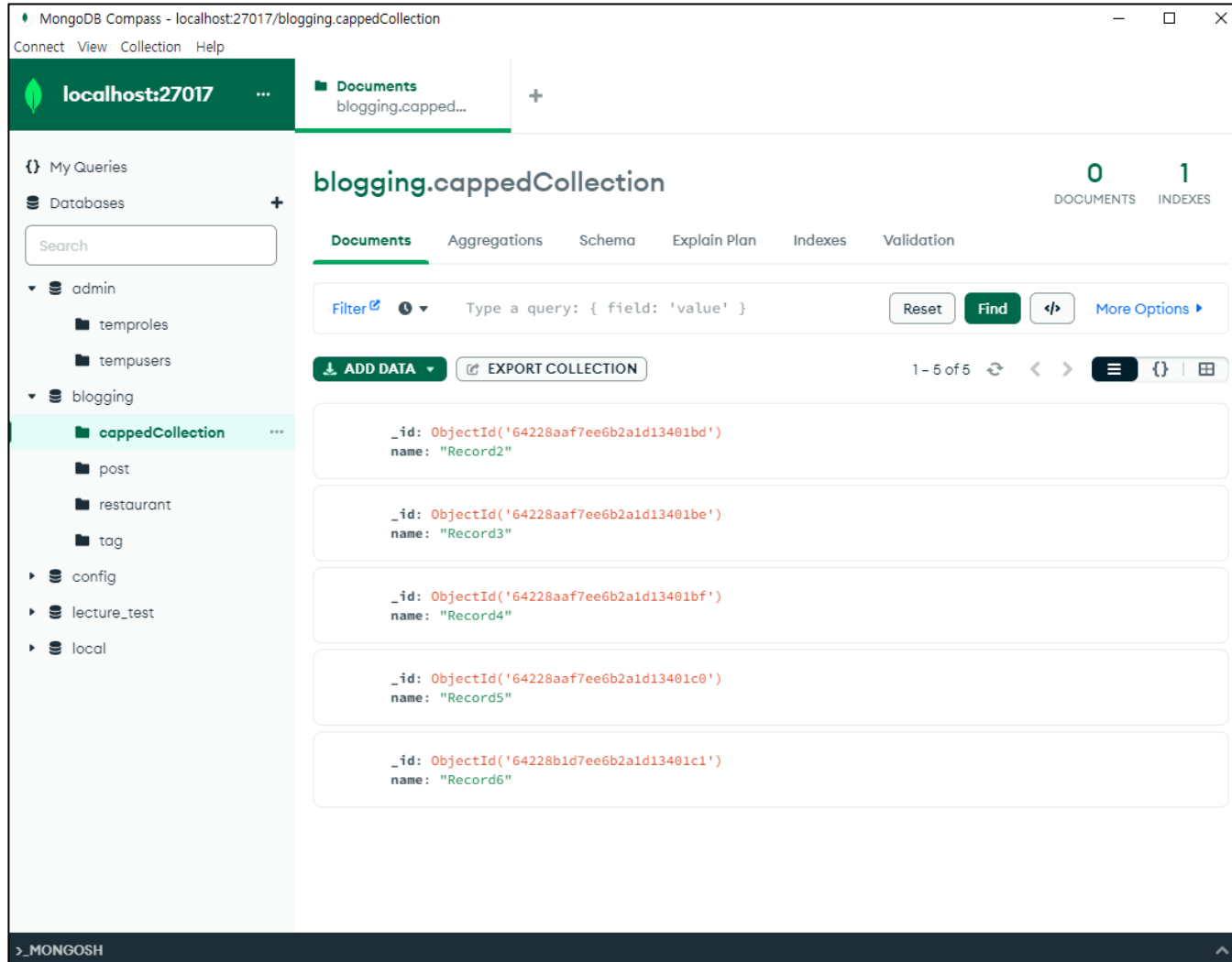
## ❖ Test Capped Collection by inserting Record6

The screenshot displays the MongoDB Compass interface for the `localhost:27017` connection. The left sidebar shows the database structure with `blogging.cappedCollection` selected. The main panel shows the `blogging.cappedCollection` documents tab, displaying three records: `Record1`, `Record2`, and `Record3`. The document counts are 0 for documents and 1 for indexes. Below the main panel, a terminal window shows the MongoDB command to insert a new record into the capped collection.

```
>_MONGOSH
> db.cappedCollection.insertOne(
  {name: "Record6"})
< {
  acknowledged: true,
  insertedId: ObjectId("64228b1d7ee6b2a1d13401c1")
}
blogging>
```

# Document

## ❖ Test Capped Collection by inserting Record6

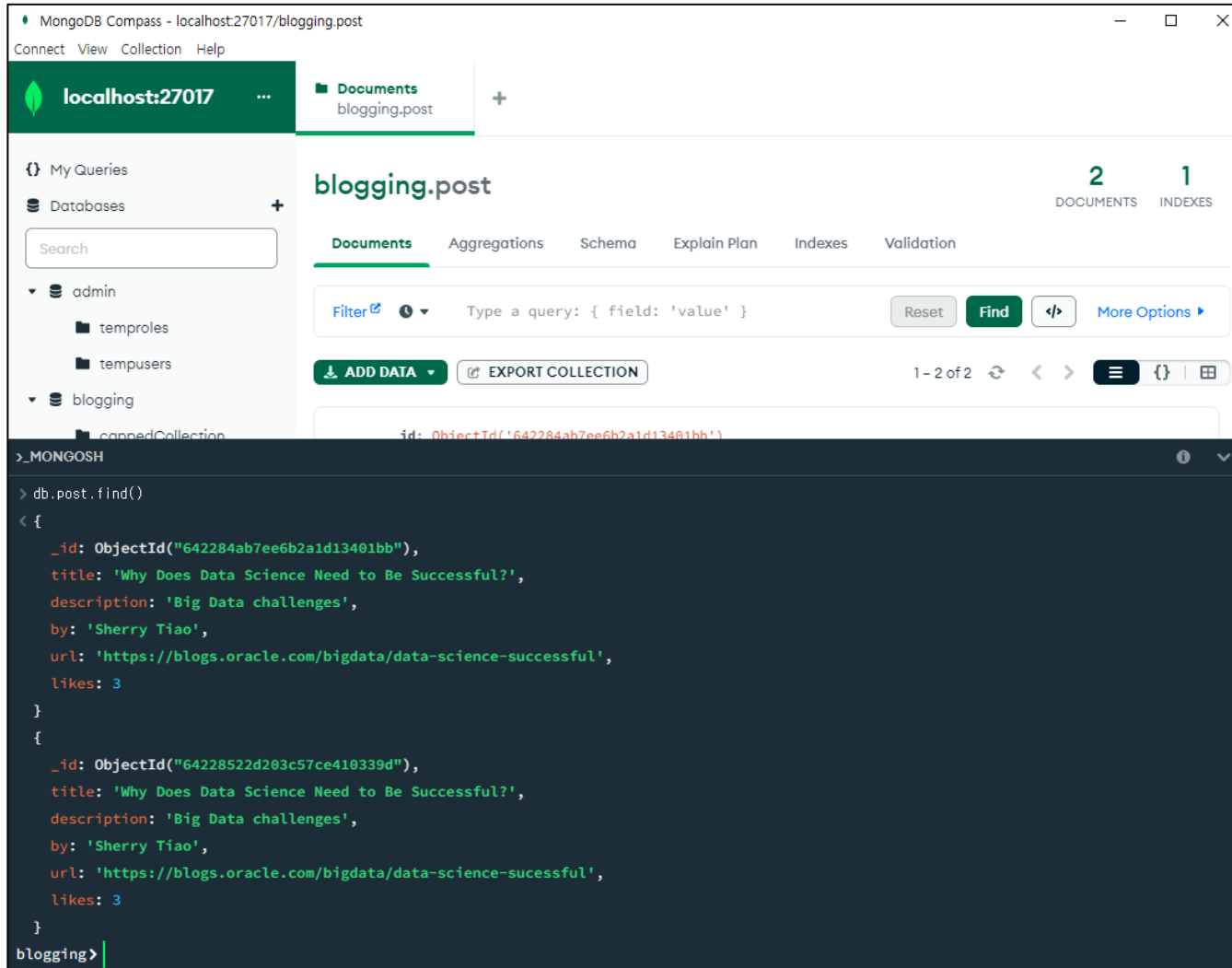


# Query Document

- ❖ Select all documents in a collection
  - To query the post collection, simply define the following select criteria
    - **db. post.find()**
  - To display the results in a formatted way, you can use pretty() method
    - *db.COLLECTION\_NAME.find().pretty()*
    - **db. post.find().pretty()**
      - Compare difference between them
  - This operation corresponds to the following SQL statement
    - **SELECT \* FROM post**

# Query Document

## ❖ Result of querying POST collection in MongoDB Shell



The screenshot displays the MongoDB Compass interface and the MongoDB Shell. The top section shows the MongoDB Compass window with the 'blogging.post' collection selected. The 'Documents' tab is active, showing 2 documents and 1 index. The 'Find' button is highlighted, and the query field contains the default query: `{ field: 'value' }`. Below the query field, there are buttons for 'ADD DATA' and 'EXPORT COLLECTION'. The bottom section shows the MongoDB Shell with the command `db.post.find()` executed, resulting in two documents being returned. The documents are displayed in a JSON format, showing fields like `_id`, `title`, `description`, `by`, `url`, and `likes`.

**MongoDB Compass Interface:**

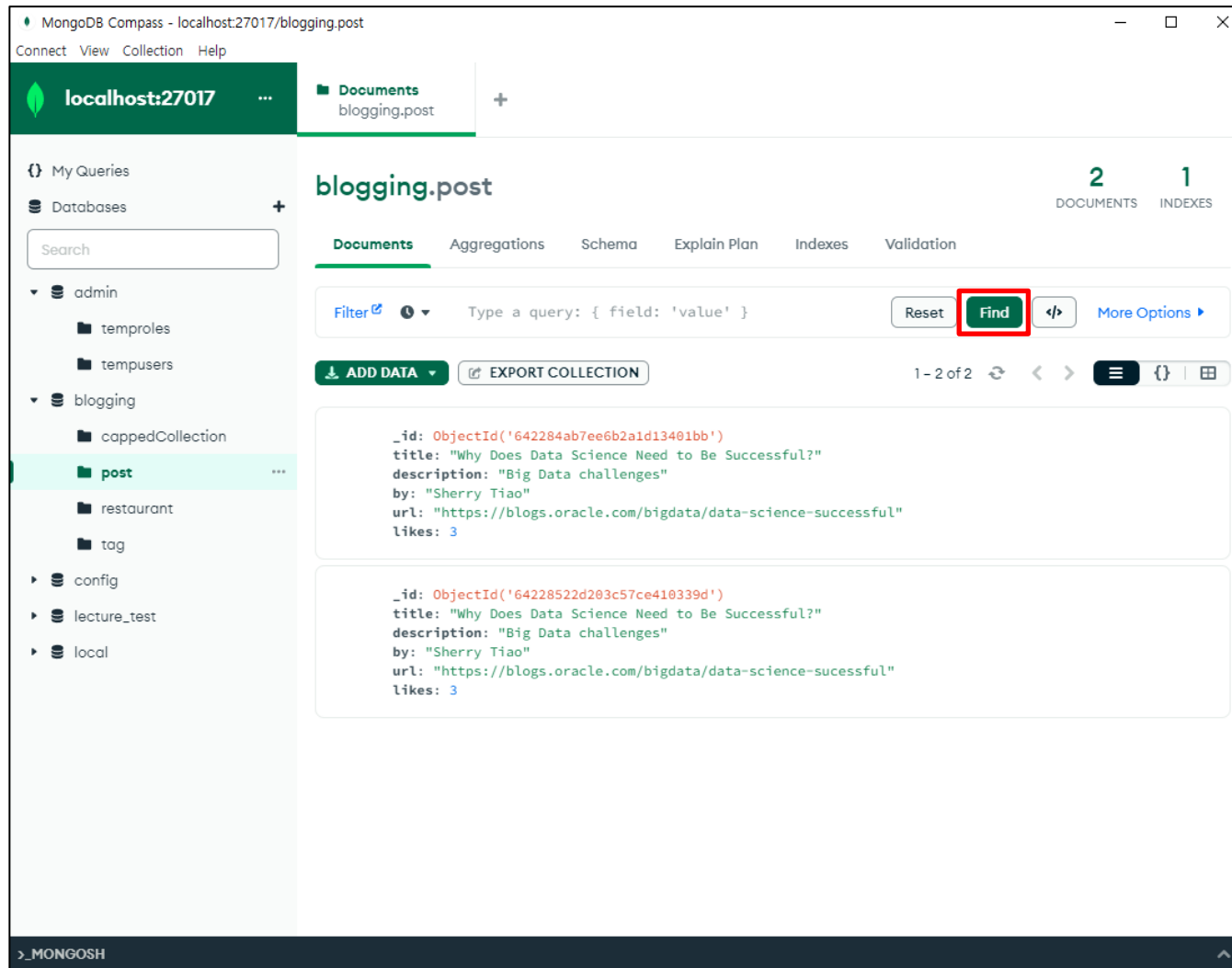
- Database: localhost:27017
- Collection: blogging.post
- Documents: 2
- Indexes: 1
- Query: `{ field: 'value' }`
- Buttons: Filter, Reset, Find, More Options
- Buttons: ADD DATA, EXPORT COLLECTION
- Page: 1 - 2 of 2

**MongoDB Shell Output:**

```
>_MONGOSH
> db.post.find()
< {
  _id: ObjectId("642284ab7ee6b2a1d13401bb"),
  title: 'Why Does Data Science Need to Be Successful?',
  description: 'Big Data challenges',
  by: 'Sherry Tiao',
  url: 'https://blogs.oracle.com/bigdata/data-science-successful',
  likes: 3
}
{
  _id: ObjectId("64228522d203c57ce410339d"),
  title: 'Why Does Data Science Need to Be Successful?',
  description: 'Big Data challenges',
  by: 'Sherry Tiao',
  url: 'https://blogs.oracle.com/bigdata/data-science-sucessful',
  likes: 3
}
blogging>
```

# Query Document

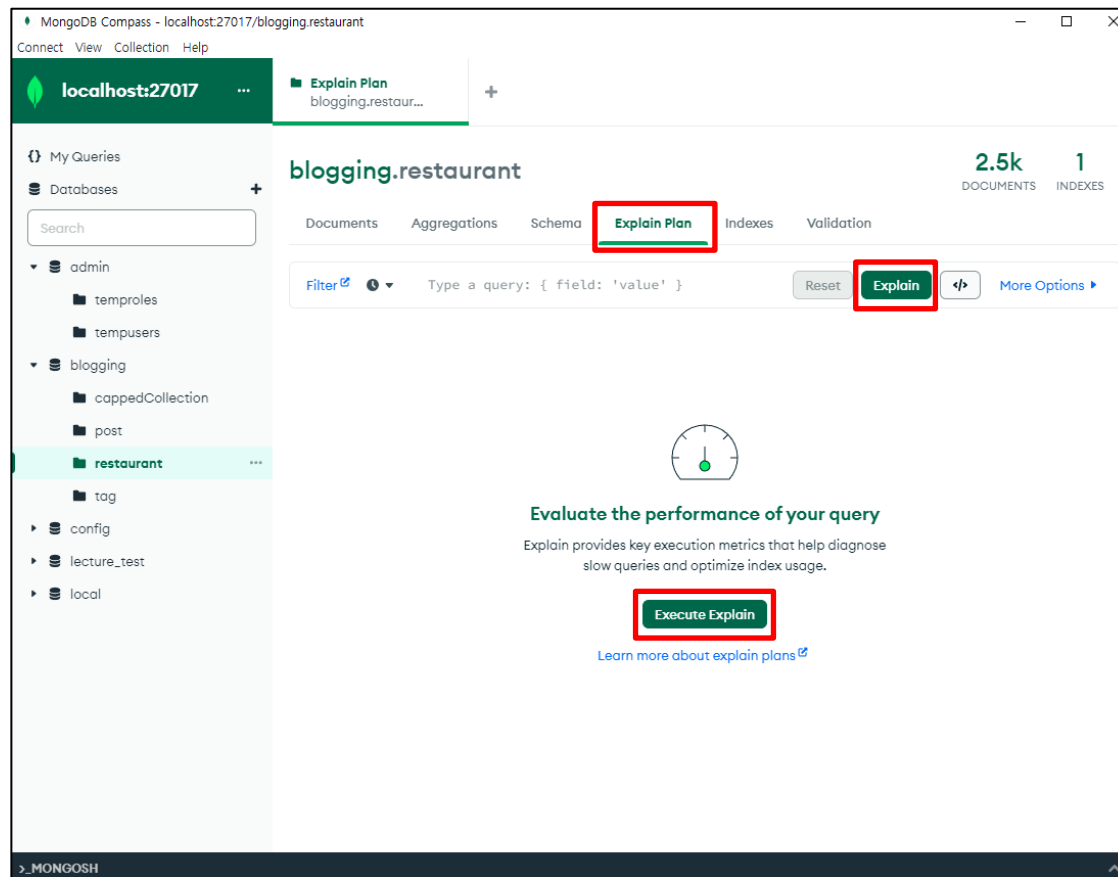
- ❖ You can also avoid writing complex queries by Compass GUI





# Query Document

- ❖ Evaluate the performance of your query
  - Explain provides key execution metrics that help diagnose slow queries and optimize index usage (use restaurant collection)



# Query Document

## ❖ Evaluate the performance of your query

The screenshot shows the MongoDB Compass interface for a local instance at localhost:27017. The database is 'blogging' and the collection is 'restaurant'. The 'Explain Plan' tab is selected, showing a query performance summary for a query that returns 2548 documents. The summary indicates that no index is available for this query, resulting in a COLLSCAN operation. The execution time is 0 ms, and 2548 documents were examined.

**Query Performance Summary**

Documents Returned: <b>2548</b>	Actual Query Execution Time (ms): <b>2</b>
Index Keys Examined: <b>0</b>	Sorted in Memory: <b>no</b>
Documents Examined: <b>2548</b>	<b>⚠ No index available for this query.</b>

**COLLSCAN**

nReturned: **2548** Execution Time: **0 ms**

Documents Examined: **2548**

[DETAILS](#)

# Query Document

## ❖ Evaluate the performance of your query

The screenshot shows the MongoDB Compass interface for a query on the `blogging.restaurant` collection. The query is `{ "address.line.2": "London" }`. The interface displays the query performance summary and the execution plan.

**Query Performance Summary**

Metric	Value
Documents Returned	345
Index Keys Examined	0
Documents Examined	2548
Actual Query Execution Time (ms)	1
Sorted in Memory	no
Index Available	No index available for this query.

**Execution Plan**

The execution plan is a **COLLSCAN** (Collection Scan). The execution time is 0 ms. The number of documents returned is 345, and the number of documents examined is 2548.

**Visual Tree**

The visual tree shows the query execution plan. The root node is a **COLLSCAN** node. The execution time is 0 ms. The number of documents returned is 345, and the number of documents examined is 2548.

# Final Task

❖ Do the following tasks for the final task

1. Clean up the post collection
2. Change the post collection into capped collection
  - With maximum number of documents 5
3. Test the post collection by inserting at least 5 blogs
  - Use insertMany() function
4. Insert one more document into the post collection
  - Use insertOne() function
  - Check the behavior of the post collection
5. Perform queries and check out the performance using 'restuarants' dataset

Questions?

**SEE YOU NEXT TIME!**