MONGODB

EGCO321 DATABASE SYSTEMS

KANAT POOLSAWASD
DEPARTMENT OF COMPUTER ENGINEERING
MAHIDOL UNIVERSITY

WHAT IS MONGODB ? (1)

- Developed by 10gen, founded in 2007
- A document oriented, NoSQL database
  - Hash - based, schema - less database
    - No Data Definition Language
    - In practice, this means you can store hashes with any keys and values that you choose
      - Keys are a basic data type but in reality stored as strings
      - Document Identifiers (_id) will be created for each document, field name reserved by system
    - Application tracks the schema and mapping
    - Uses BSON format
      - Based on JSON – B stands for Binary

WHAT IS MONGODB ? (2)

- Supports APIs (drivers) in many computer languages
  - JavaScript, Python, Ruby, Perl, Java, Java Scala, C#, C++, Haskell, Erlang

FUNCTIONALITY OF MONGODB

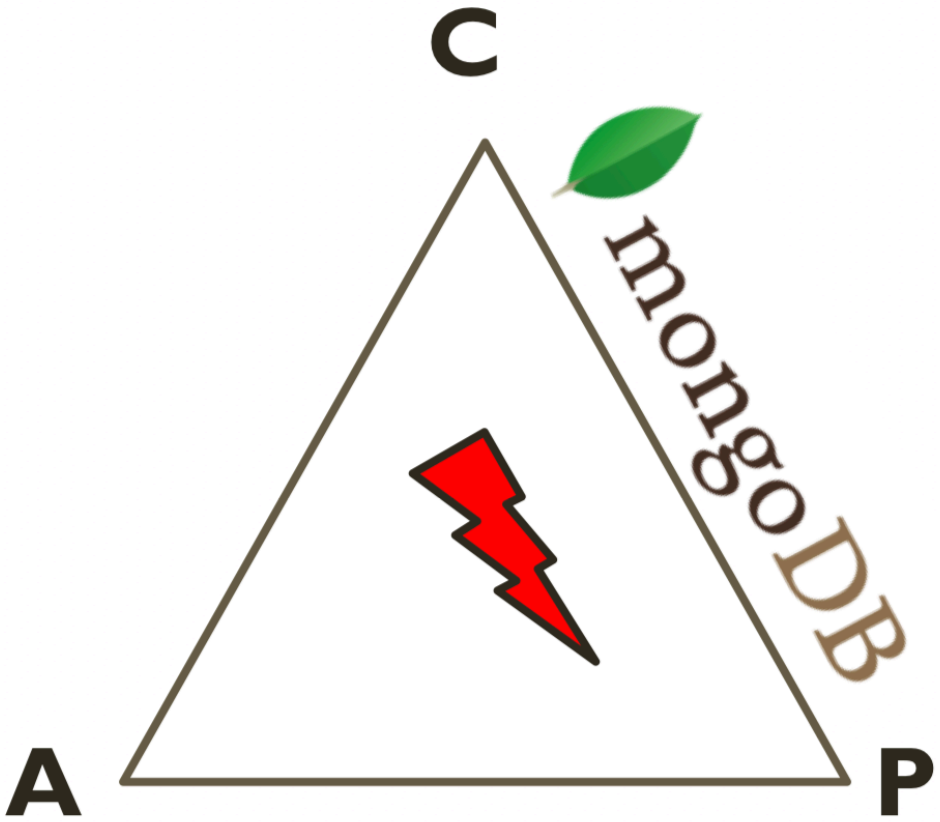
- Dynamic Schema (No DDL)
- Document - based database
- Secondary indexes
- Query language via an API
- Atomic writes and fully - consistent reads
  - If system configured that way
- Master - slave replication with automated failover (replica sets)
- Built-in horizontal scaling via automated range - based partitioning of data (sharding)
- No joins nor transactions

WHY USE MONGODB ?

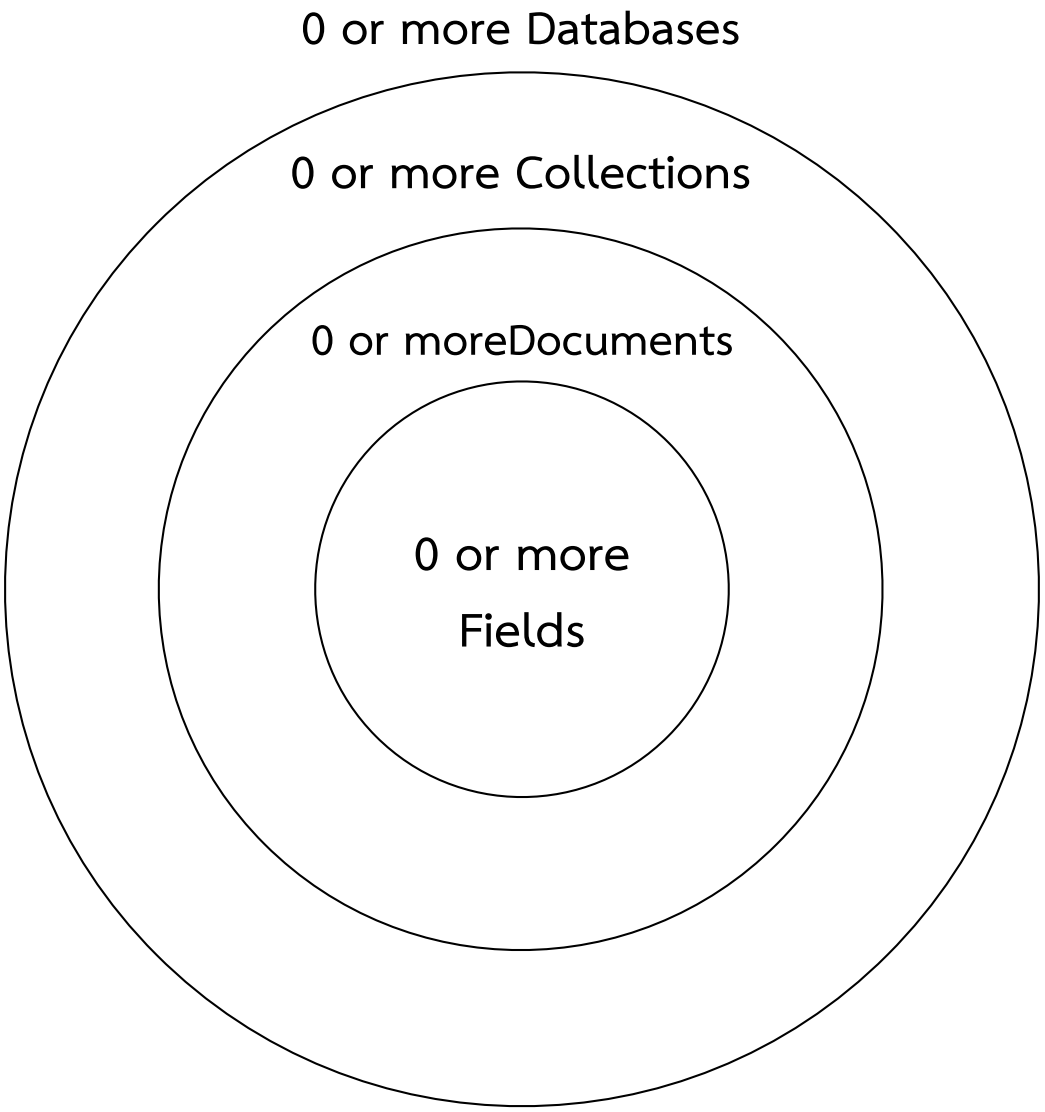
- Simple queries
- Functionality provided applicable to most web applications
- Easy and fast integration of data (No ERD diagram)
- Not well suited for heavy and complex transactions systems

MONGODB: CAP APPROACH

- Focus on Consistency and Partition Tolerance.



MONGODB: HIERARCHICAL OBJECT



RELATIONAL DB CONCEPTS TO NOSQL

Relational DB	MongoDB
Database	Database
Table, View	Collection
Row	Document (BSON)
Column	Field
Index	Index
Join	Embedded
Foreign Key	Reference
Partition	Shard

MONGODB PROCESSES AND CONFIGURATION

- Mongod – Database instance
- Mongos - Sharding processes
  - Analogous to a database router.
  - Processes all requests
  - Decides how many and which mongod s should receive the query
  - Mongos collates the results, and sends it back to the client.
- Mongo – an interactive shell (a client)
  - Fully functional JavaScript environment for use with a MongoDB
- You can have one mongos for the whole system no matter how many mongods you have OR you can have one local mongos for every client if you wanted to minimise network latency.

CHOICES MADE FOR DESIGN OF MONGODB

- Scale horizontally over commodity hardware
  - Lots of relatively inexpensive servers
- Keep the functionality that works well in RDBMSs
  - Ad hoc queries
  - Fully featured indexes
  - Secondary indexes
- What doesn't distribute well in RDB?
  - Long running multi - row transactions
  - Joins
  - Both artifacts of the relational data model (row x column)

BSON FORMAT

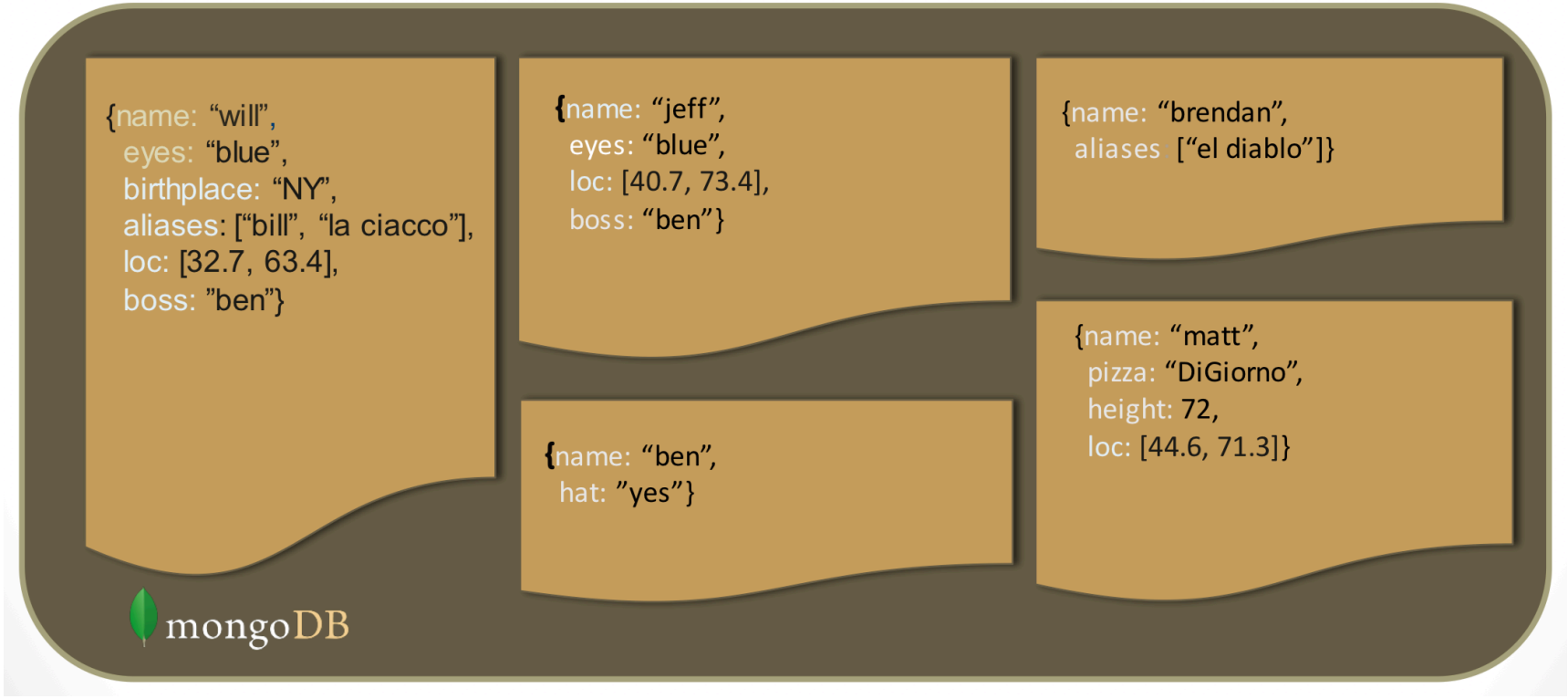
- Binary-encoded serialization of JSON - like documents
- Zero or more key/value pairs are stored as a single entity
- Each entry consists of a field name, a data type, and a value
- Large elements in a BSON document are prefixed with a length field to facilitate scanning

THE _ID FIELD

- By default, each document contains an _id field. This field has a number of special characteristics:
  - Value serves as primary key for collection.
  - Value is unique, immutable, and may be any non-array type.
  - Default data type is ObjectId, which is “small, likely unique, fast to generate, and ordered.” Sorting on an ObjectId value is roughly equivalent to sorting on creation time.

SCHEMA FREE

- MongoDB does not need any pre - defined data schema
- Every document in a collection could have different data
  - Addresses NULL data fields

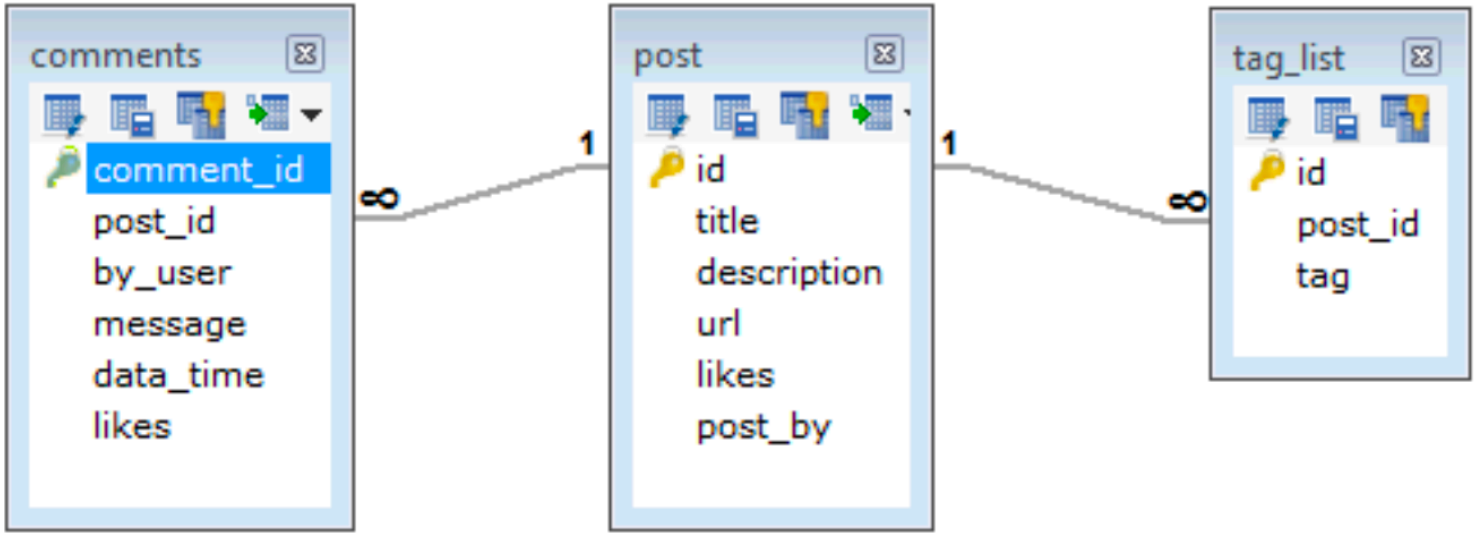


RDBMS VS MONGODB (1)

- Suppose a client needs a database design for his blog/ website and see the differences between RDBMS and MongoDB schema design. Website has the following requirements.
  - Every post has the unique title, description and url.
  - Every post can have one or more tags.
  - Every post has the name of its publisher and total number of likes.
  - Every post has comments given by users along with their name, message, data-time and likes.
- On each post, there can be zero or more comments.

RDBMS VS MONGODB (2)

- In RDBMS schema, design for above requirements will have minimum three tables.



RDBMS VS MONGODB (3)

- While in MongoDB schema, design will have one collection post and the following structure:

```
{
   _id: POST_ID
   title: TITLE_OF_POST,
   Description: POST_DESCRIPTION,
   by: POST_BY,
   Url: URL_OF_POST,
   Tags: [TAG1, TAG2, … ],
   likes: TOTAL_LIKES
   comments: [ { user:COMMENT_BY,
                 message: TEXT,
                 dateCreate: DATE_TIME,
                 Like: LIKES },
               { user:COMMENT_BY,
                 message: TEXT,
                 dateCreate: DATE_TIME,
                 Like: LIKES }
             ]
}
```

MONGODB - DATABASE

- MongoDB **use DATABASE_NAME** is used to create database.
  - The command will create a new database if it doesn't exist, otherwise it will return the existing database.
- MongoDB **show dbs;** command is used to show all database.
- MongoDB **db.dropDatabase()** command is used to drop a existing database.

MONGODB - COLLECTION

- MongoDB **db.createCollection(name)** is used to create collection.
- Example:

```
db.createCollection("mycollection")
```

- MongoDB's **db.collection.drop()** is used to drop a collection from the database.

MONGODB - DATA TYPE

- MongoDB supports many datatypes. Some of them are:
  - String
  - Integer
  - Boolean
  - Double
  - Arrays
  - Timestamp
  - Object
  - etc.

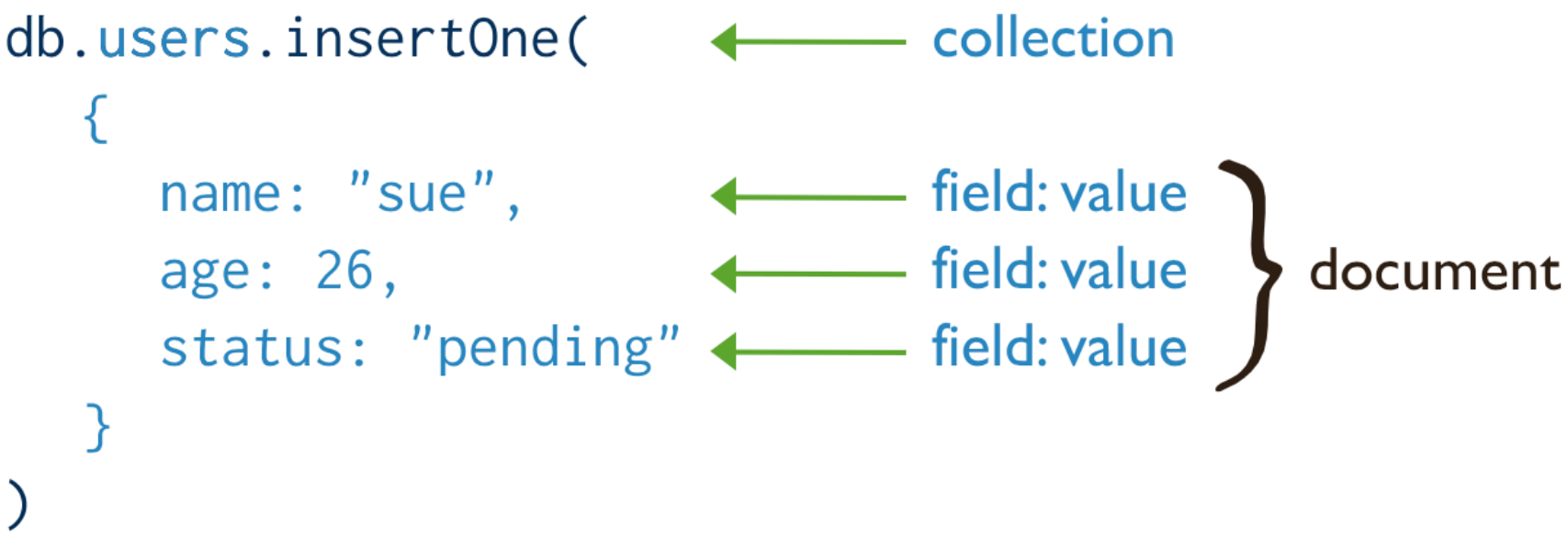
MONGODB - CREATE (1)

- Create Operations
  - Create or insert operations add new documents to a collection. If the collection does not currently exist, insert operations will create the collection.
  - MongoDB provides the following methods to insert documents into a collection:
    - `db.collection.insert()*`
    - `db.collection.insertOne()`
    - `db.collection.insertMany()`

* Old version before 3.2

MONGODB - CREATE (2)

- Create Operation Example



MONGODB - READ (1)

- Read Operations
  - Read operations retrieves documents from a collection; i.e. queries a collection for documents. MongoDB provides the following methods to read documents from a collection:

```
db.collection.find(<query>,<project>)
db.collection.find(<query>,<project>).pretty()
```

  - Example:

```
db.courses.find({room:"6273"}).pretty();
```

MONGODB - READ (2)

- To query the document on the basis of some condition, you can use following operations

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:<value>}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}}).pretty()	where likes > 50
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}}).pretty()	where likes != 50

MONGODB - READ (3)

- In the find() method, if you pass multiple keys by separating them by ',' then MongoDB treats it as AND condition. Following is the basic syntax of AND

```
db.collection.find({key1:value1, key2:value2})
```

- Example:

```
db.courses.find({room:"6273", code:”EGCO343"});
```

MONGODB - READ (4)

- To query documents based on the OR condition, you need to use $or keyword. Following is the basic syntax of OR

```
db.collection.find(
   {
      $or: [ {key1: value1}, {key2:value2} ]
   }
)
```

- Example:

```
db.courses.find({$or:[{room:”6273"},{room:"6275"}]});
```

MONGODB - READ (5)

- Using AND and OR together

```
db.courses.find(
 {"credit": {$gte:3},
   $or: [{"instructor":null},
         {"instructor":"Kanat Poolsawasd"}]
 }
);
```

MONGODB - READ (6)

- $regex provides **regular expression** capabilities for pattern matching strings in queries.
- To use **$regex**, use one of the following syntaxes:

```
{ <field>: { $regex: /pattern/, $options: '<options>' } }

{ "<field>": { "$regex": "pattern", "$options": "<options>" } }

{ <field>: { $regex: /pattern/<options> } }
```

MONGODB - READ (7)

- Example:

Use Case	Operator	Example
Contains	`$regex`	`{ name: { $regex: "john" } }`
Case-insensitive	`$regex + $options:"i"`	`{ name: { $regex: "john", $options: "i" } }`
Starts with	`^pattern`	`{ name: { $regex: /^john/i } }`
Ends with	`pattern$`	`{ name: { $regex: /john$/i } }`
Full-text search	`$text`	`{ $text: { $search: "john" } }`

MONGODB - UPDATE (1)

- Update Operations
  - Update operations modify existing documents in a collection. MongoDB provides the following methods to update documents of a collection:
    - `db.collection.update()*`
    - `db.collection.updateOne()`
    - `db.collection.updateMany()`
    - `db.collection.replaceOne()`

* Old version before 3.2

MONGODB - UPDATE (2)

- Update Operation Example

```
db.users.updateMany(
   { age: { $lt: 18 } },
   { $set: { status: "reject" } }
)
```

collection

update filter

update action

MONGODB - DELETE (1)

- Delete Operations
  - Delete operations remove documents from a collection. MongoDB provides the following methods to delete documents of a collection:
    - `db.collection.remove()*`
    - `db.collection.deleteOne()`
    - `db.collection.deleteMany()`

* Old version before 3.2

MONGODB - DELETE (2)

- Delete Operation Example

```
db.users.deleteMany(
   { status: "reject" }
)
```

MONGODB - PROJECTION

- MongoDB's `find()` method, explained in MongoDB Query Document accepts second optional parameter that is list of fields that you want to retrieve.
- In MongoDB, when you execute `find()` method, then it displays all fields of a document.
- To limit this, you need to set a list of fields with value 1 or 0. (1 is used to show the field while 0 is used to hide the fields.)

```
db.collection.find({<query>},{KEY:1})
```

- Example:

```
db.courses.find({room:”6273”},{“code":1,_id:0})
```

MONGODB - LIMIT RECORDS

- To limit the records in MongoDB, you need to use limit() method.
- The method accepts one number type argument, which is the number of documents that you want to be displayed.

```
db.collection.find(<query>).limit(NUMBER)
```

- Example:

```
db.courses.find({room:”6273”},{“code”:1,_id:0}).limit(1);
```

MONGODB - SORTING

- To sort documents in MongoDB, you need to use `sort()` method.
- The method accepts a document containing a list of fields along with their sorting order.
- To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

```
db.collection.find(<query>).sort({KEY:1})
```

- Example:

```
db.courses.find({room:"6273"},
  {"code":1,_id:0}).sort({"code":-1});
```

MONGODB - AGGREGATION (1)

- Aggregations operations process data records and return computed results.
- Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.
- In SQL `COUNT(*)` and with `GROUP BY` is an equivalent of mongodb aggregation.

MONGODB - AGGREGATION (2)

Stage	Purpose	Example
$match	WHERE	{ $match: { status: "active" } }
$group	GROUP BY	{ $group: { _id: "$category", total: { $sum: "$price" } } }
$sort	ORDER BY	{ $sort: { total: -1 } }
$project	Projection	{ $project: { name: 1, age: 1, _id: 0 } }
$limit	LIMIT	{ $limit: 5 }
$unwind	Array to multiple rows	{ $unwind: "$tags" }

MONGODB - AGGREGATION (3)

- In UNIX command, shell pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on.
- MongoDB also supports same concept in aggregation framework. There is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents (or the final resulting JSON document at the end of the pipeline).
- This can then in turn be used for the next stage and so on.

MONGODB - AGGREGATION (4)

- Example:

```
db.orders.insertMany([
  { _id: 1, customer: "Alice", total: 500, status:
"complete" },
  { _id: 2, customer: "Bob", total: 300, status:
"pending" },
  { _id: 3, customer: "Charlie", total: 700, status:
"complete" },
  { _id: 4, customer: "Alice", total: 200, status:
"pending" }
]);
```

MONGODB - AGGREGATION (5)

- Example (Cont.):

```
db.orders.aggregate([
  { $match: { status: "complete" } },
  { $count: "total_complete_orders" }
]);
```

- Result:

```
{ total_complete_orders: 2 }
```

MONGODB - AGGREGATION (6)

- Example:

```
db.orders.aggregate([
  { $group: { _id: "$status", count: { $sum: 1 } } },
  { $sort: { count: -1 } }
]);
```

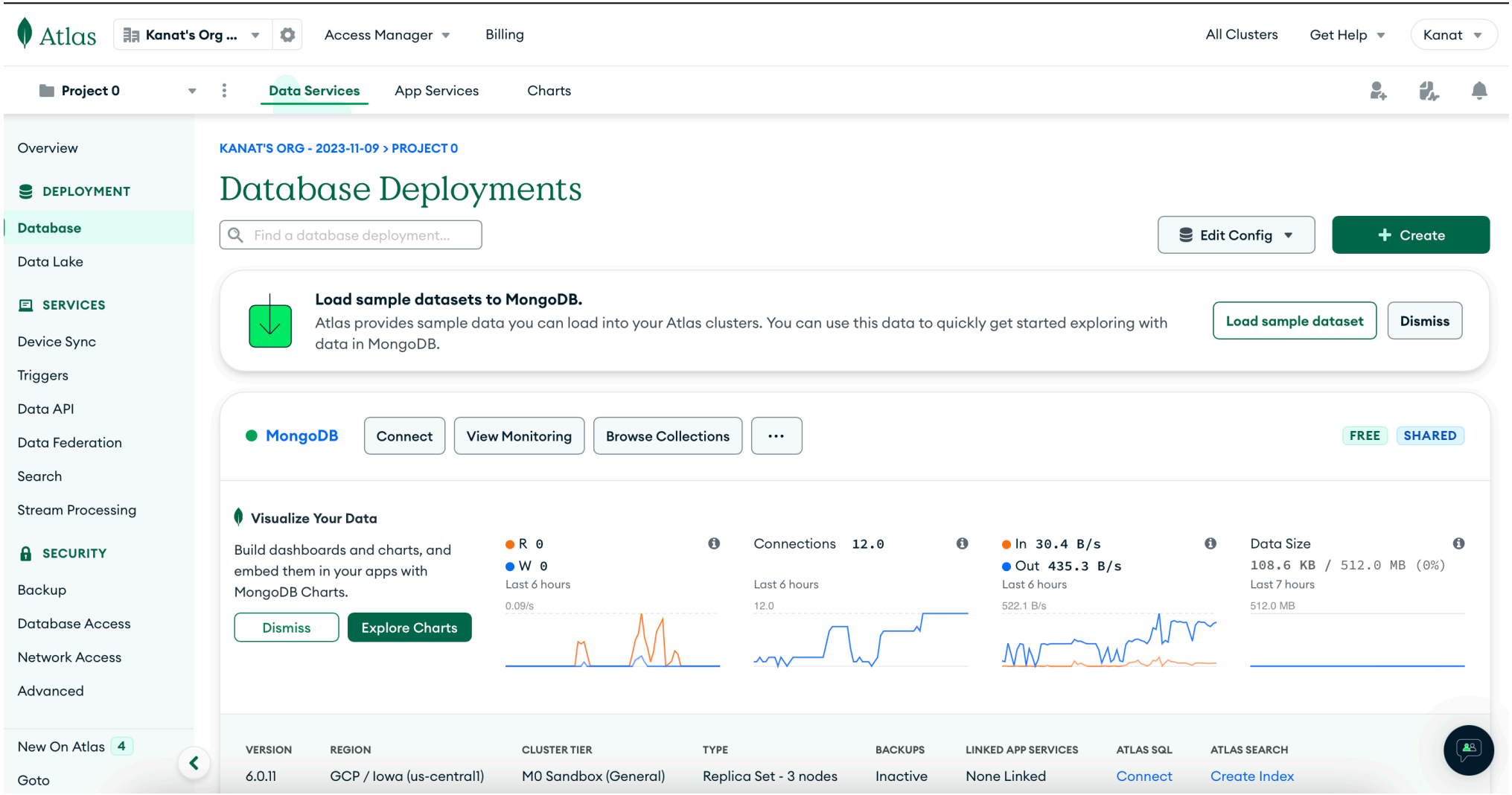
- Result:

```
{
  _id: 'complete',
  count: 2
}
{
  _id: 'pending',
  count: 2
}
```

MONGODB - AGGREGATION (7)

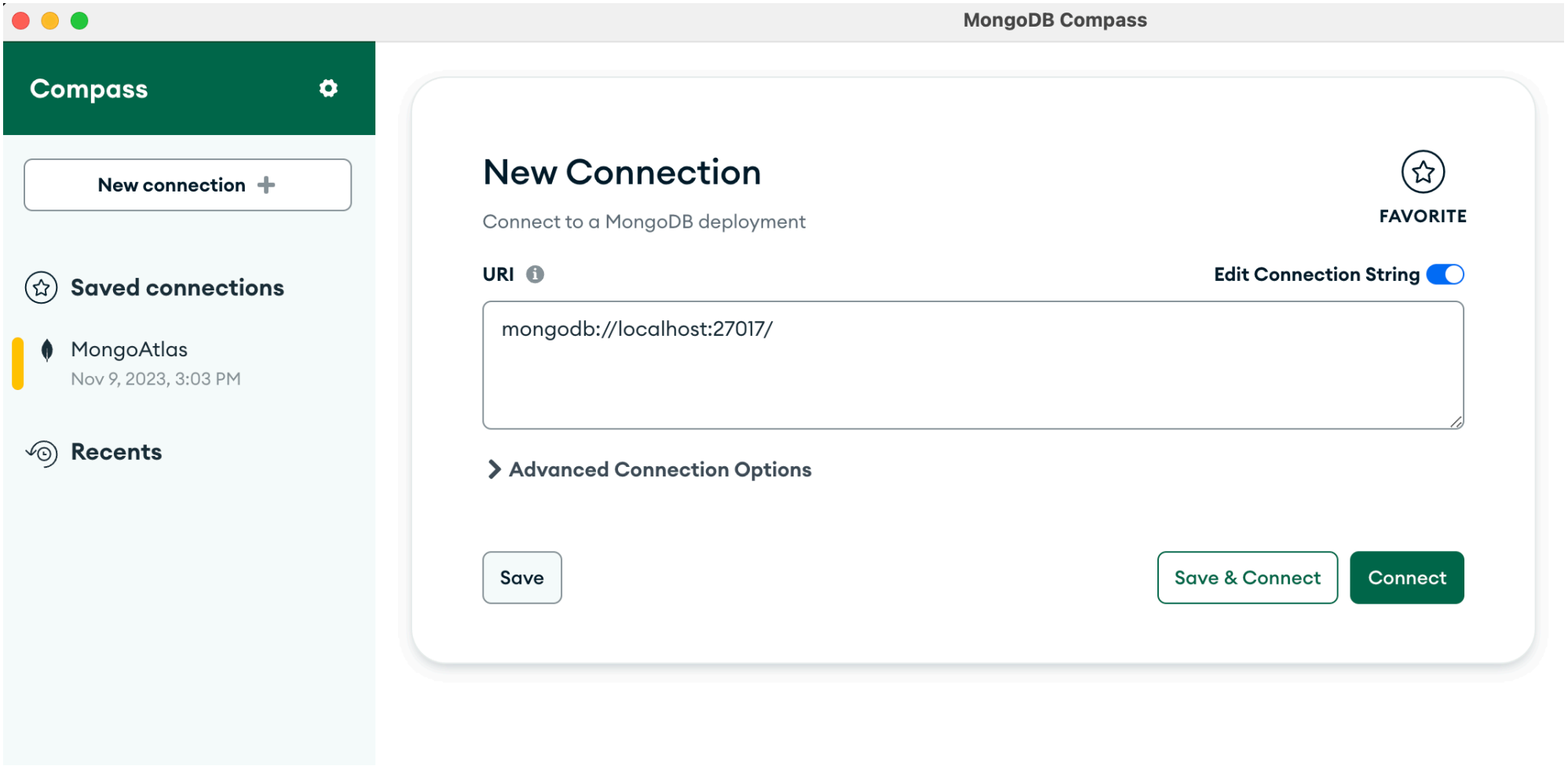
Expression	Example
$sum	db.collection.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : “$likes"}}}])
$avg	db.collection.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : “$likes"}}}])
$min	db.collection.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : “$likes"}}}])
$max	db.collection.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : “$likes"}}}])
$first	db.collection.aggregate([{$group : {_id : "$by_user", first_url : {$first : “$url"}}}])
$last	db.collection.aggregate([{$group : {_id : "$by_user", first_url : {$last : “$url"}}}])

MONGODB ATLAS

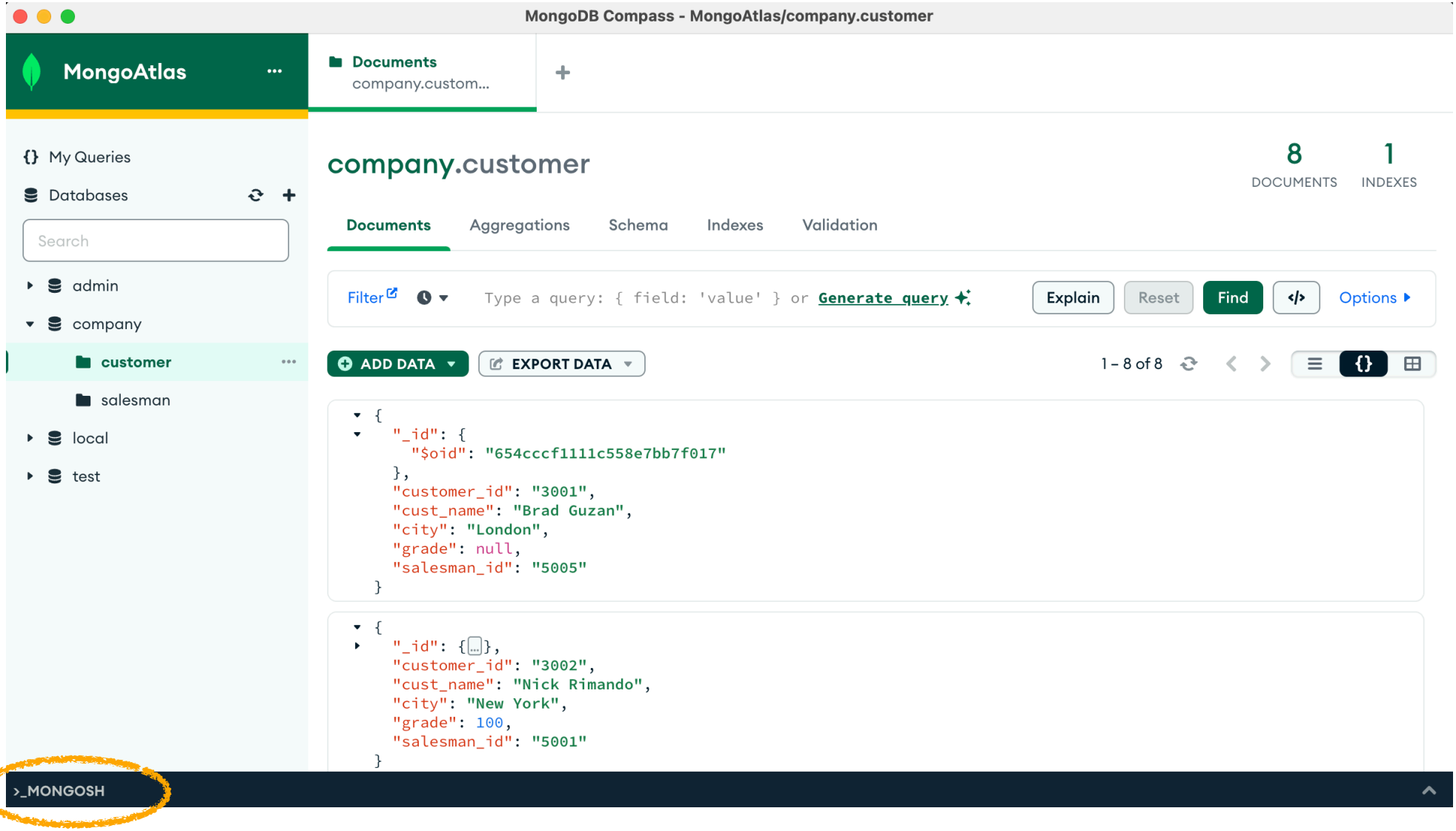


* https://cloud.mongodb.com

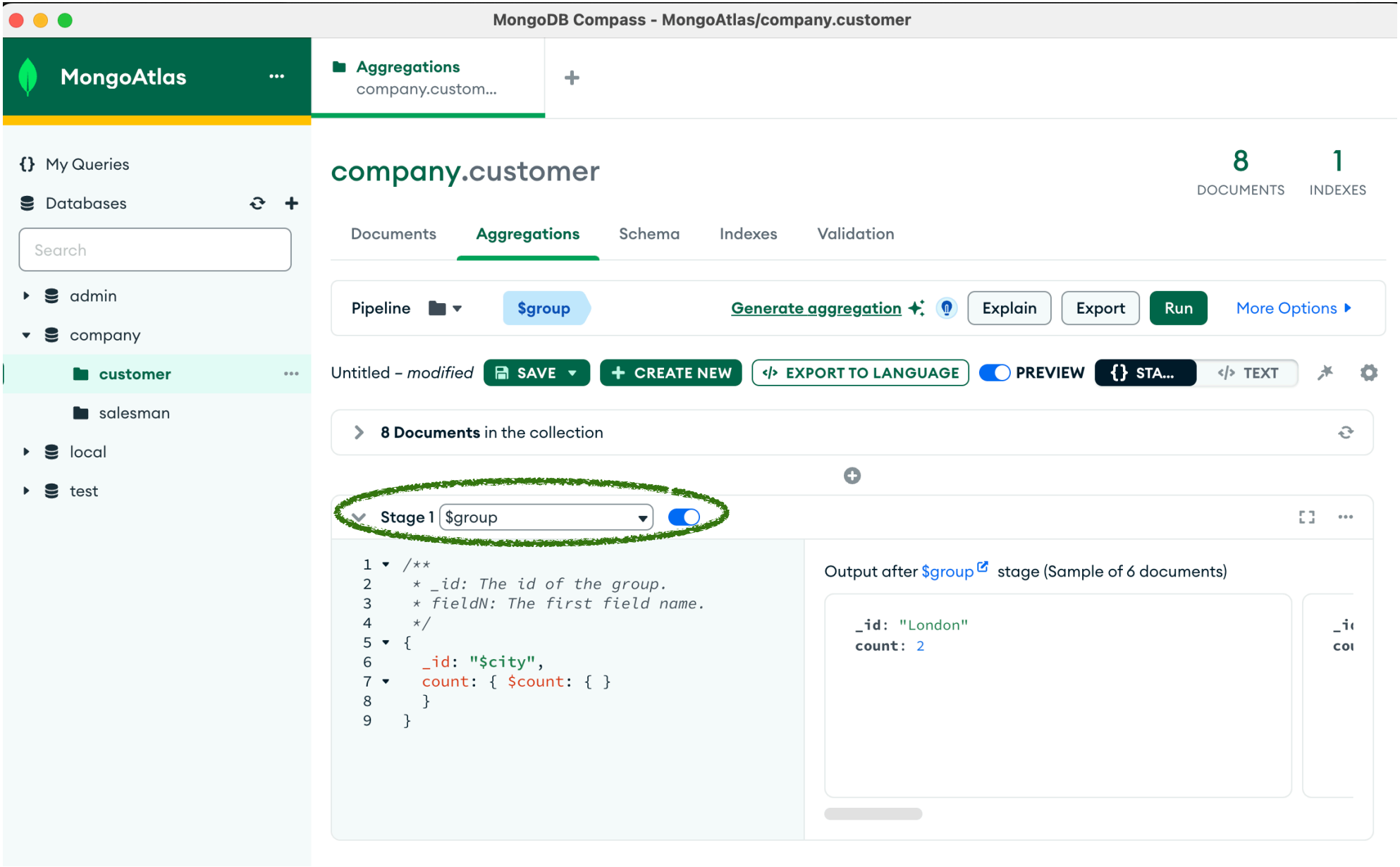
MONGODB COMPASS (1)



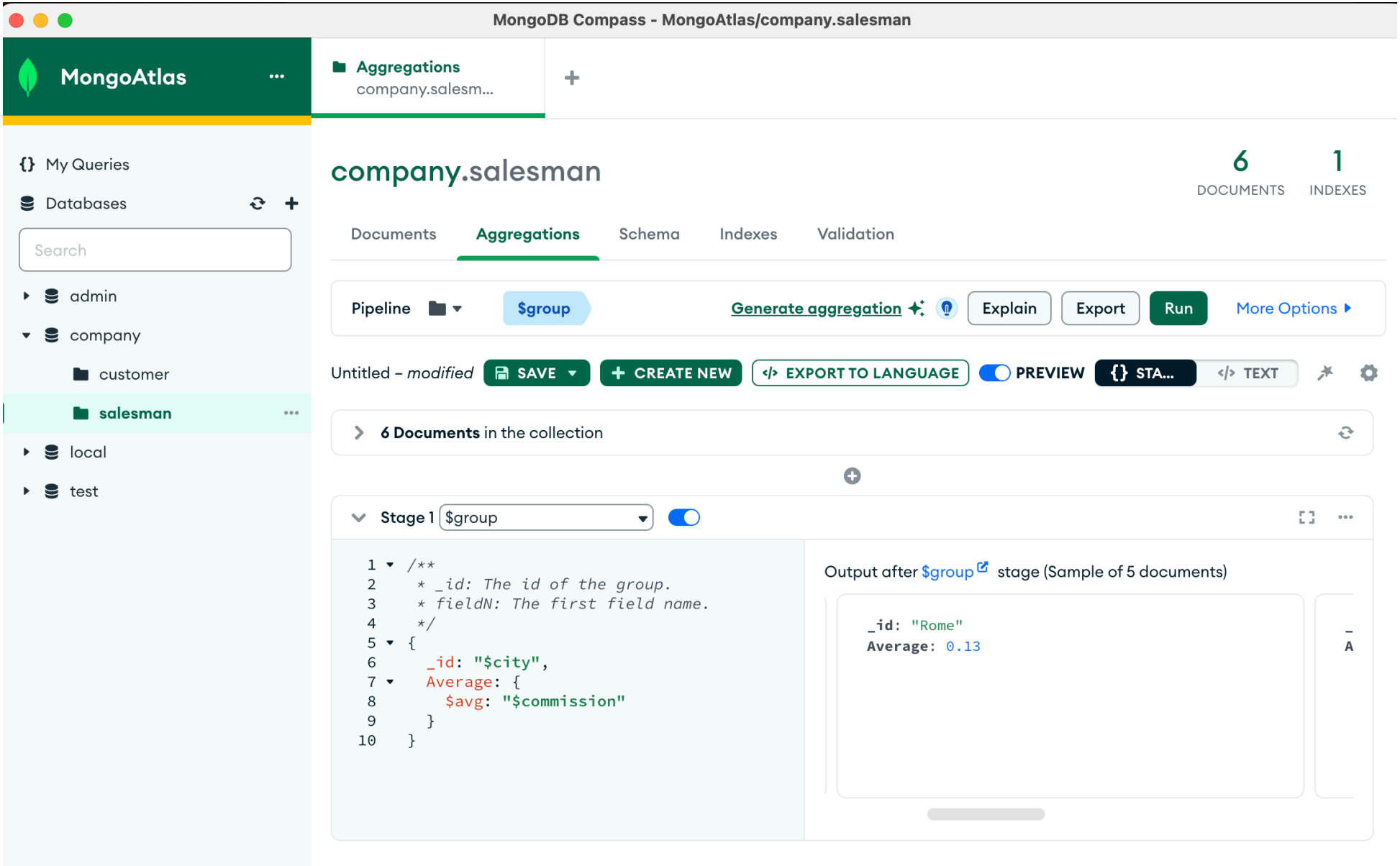
MONGODB COMPASS (2)



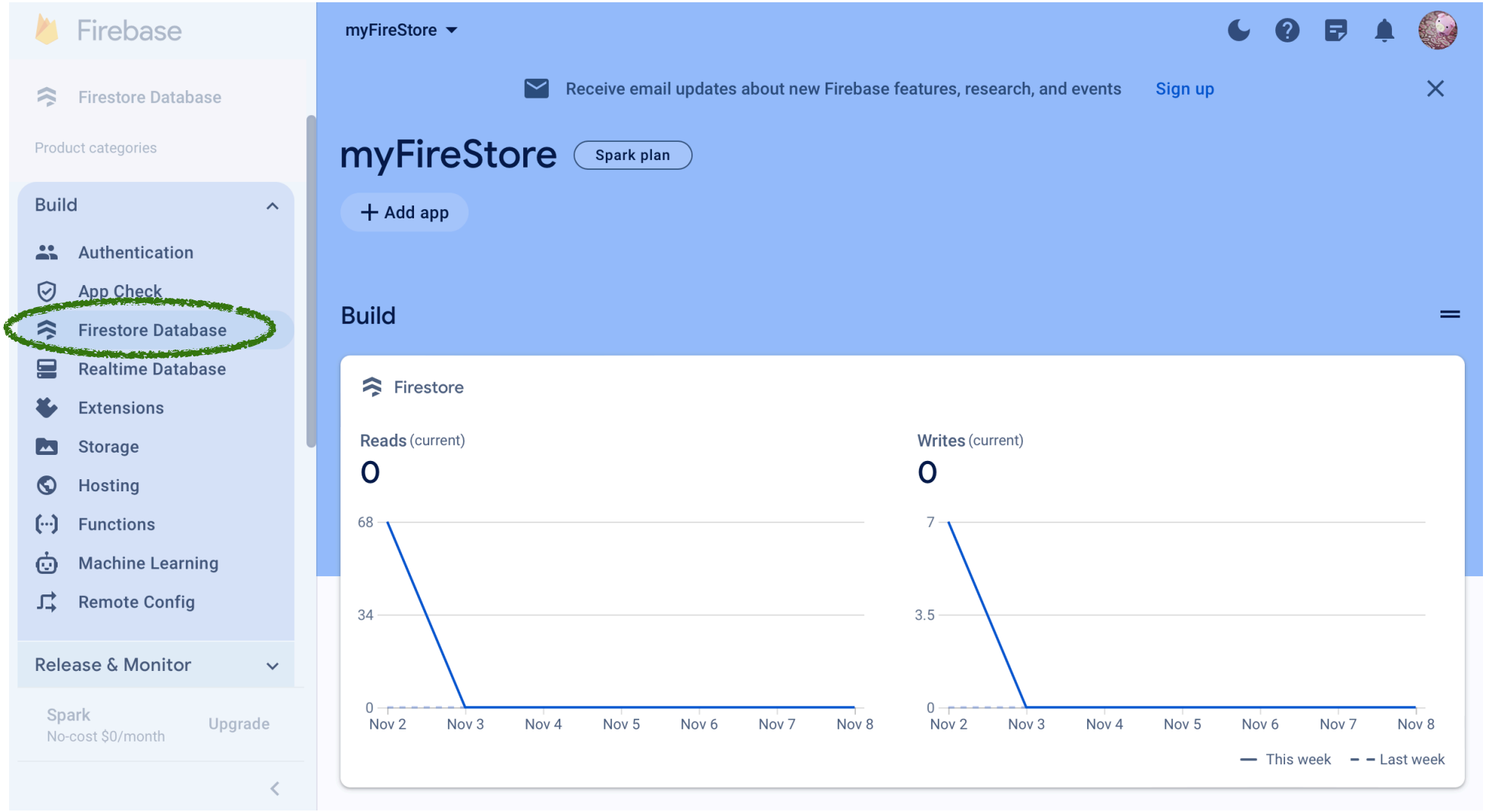
MONGODB COMPASS (3)



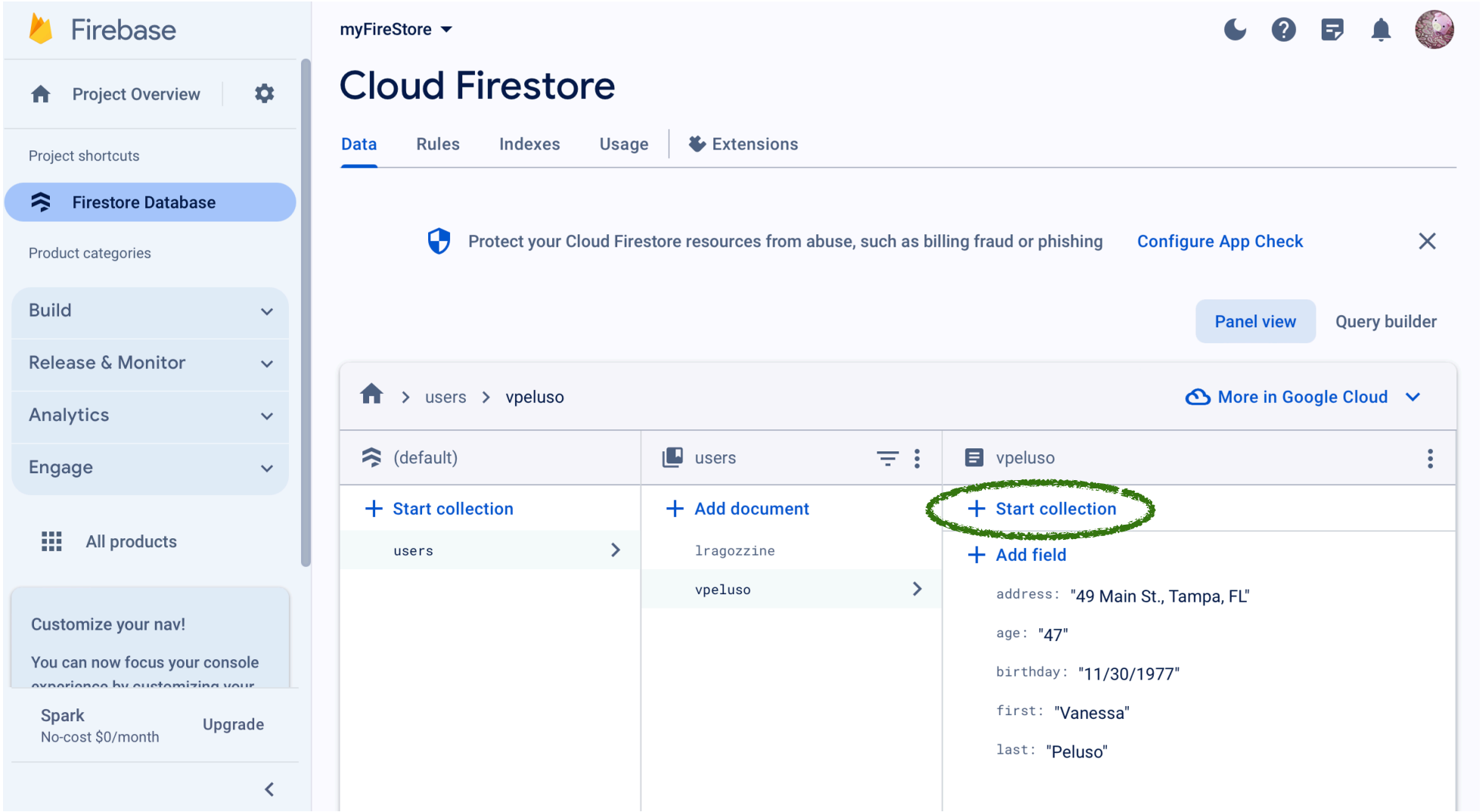
MONGODB COMPASS (4)



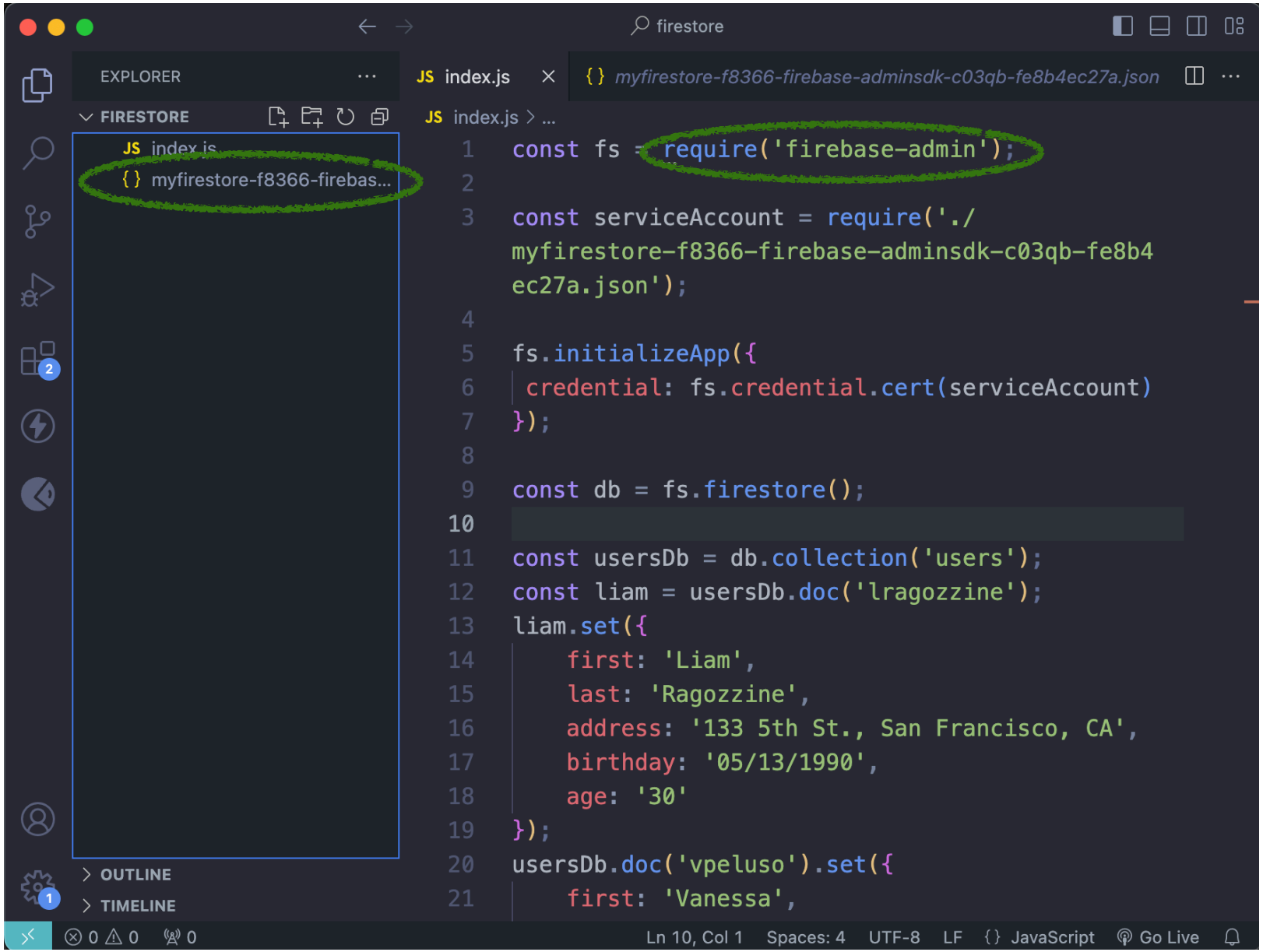
GOOGLE FIREBASE (1)



GOOGLE FIREBASE (2)



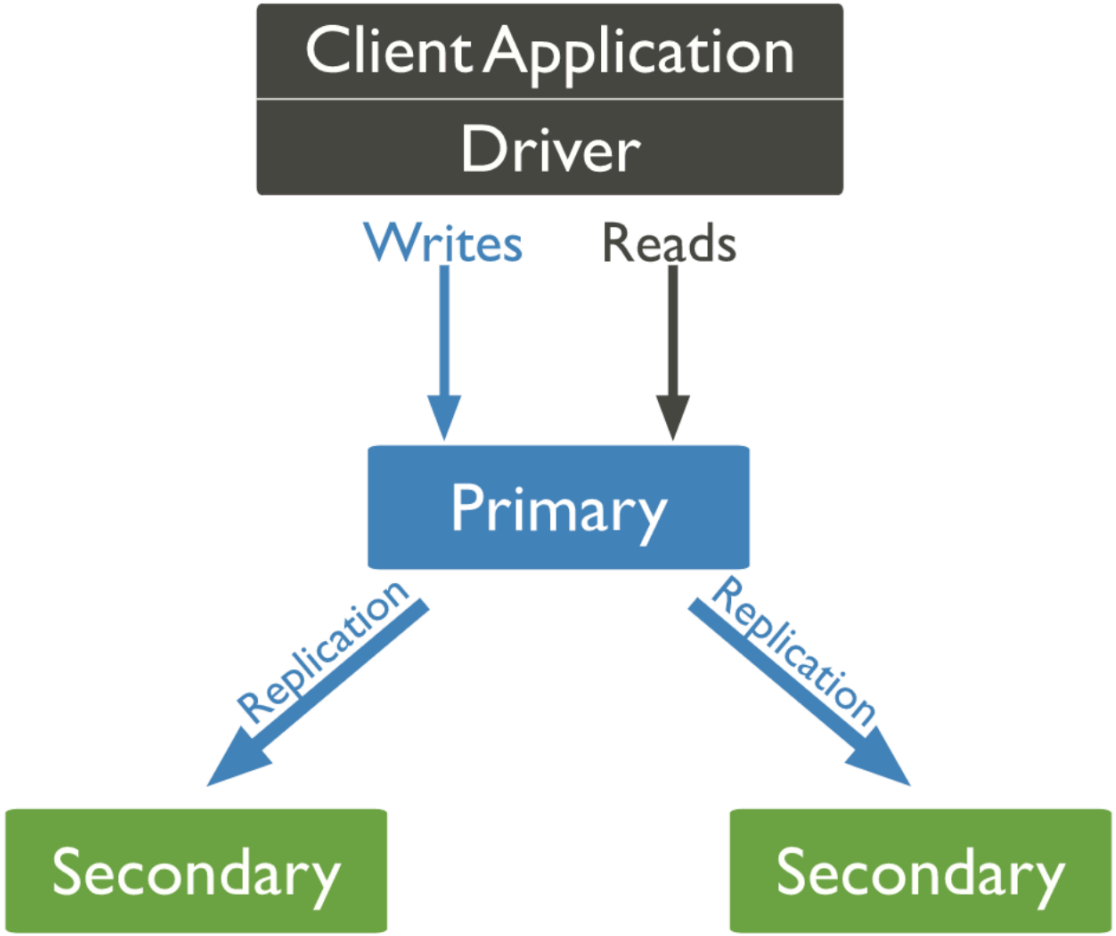
GOOGLE FIREBASE (3)



MONGODB - REPLICATION (1)

- Replication is the process of synchronizing data across multiple servers, that provides redundancy and increases data availability with multiple copies of data on different database servers.
- Replication protects a database from the loss of a single server, and also allows you to recover from hardware failure and service interruptions.
- With additional copies of the data, you can dedicate one to disaster recovery, reporting, or backup.

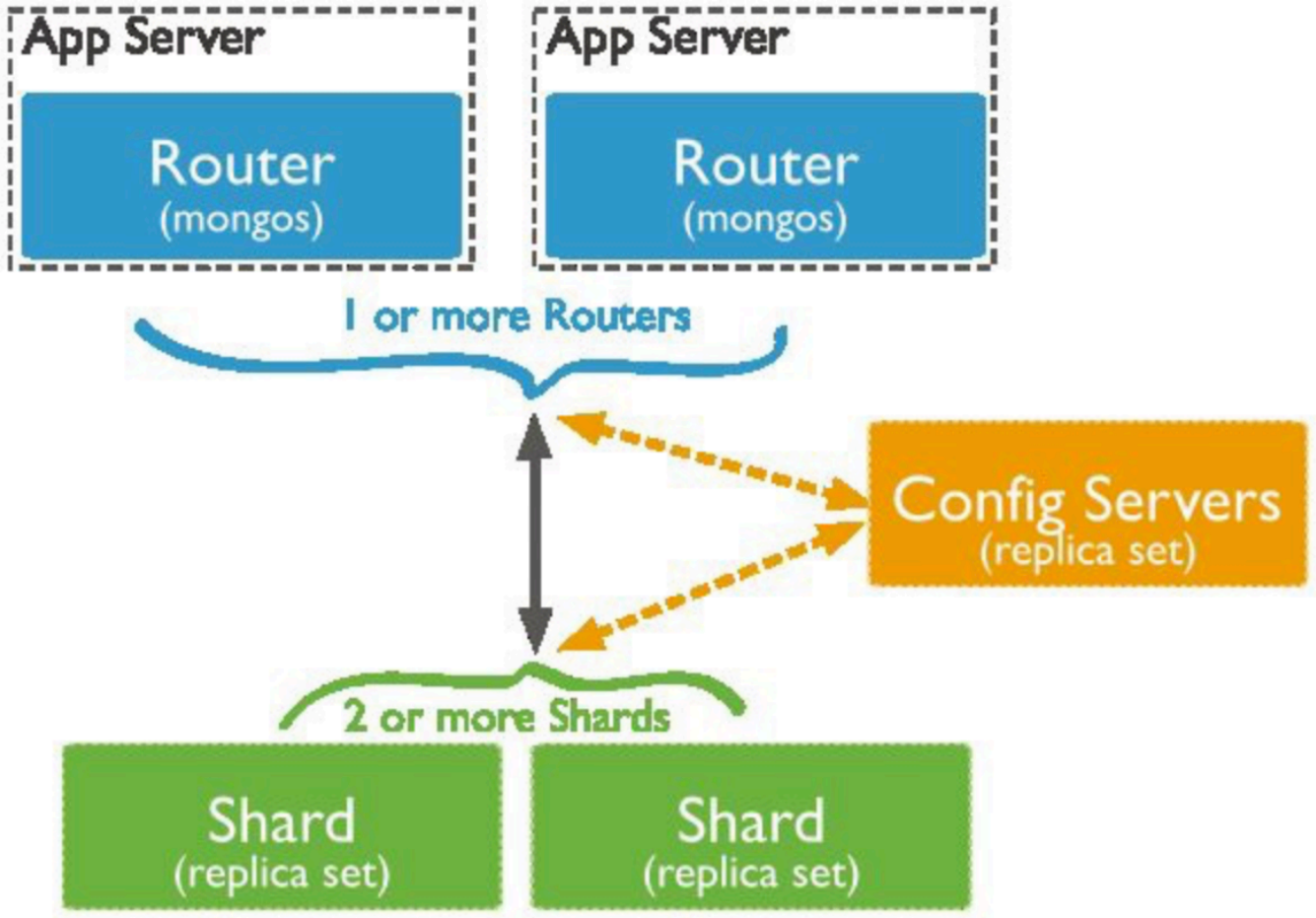
MONGODB - REPLICATION (2)



MONGODB - SHARDING (1)

- Sharding is the process of storing data records across multiple machines and it is MongoDB's approach to meeting the demands of data growth.
- As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput.
- Sharding solves the problem with horizontal scaling. With sharding, you add more machines to support data growth and the demands of read and write operations.

MONGODB - SHARDING (2)

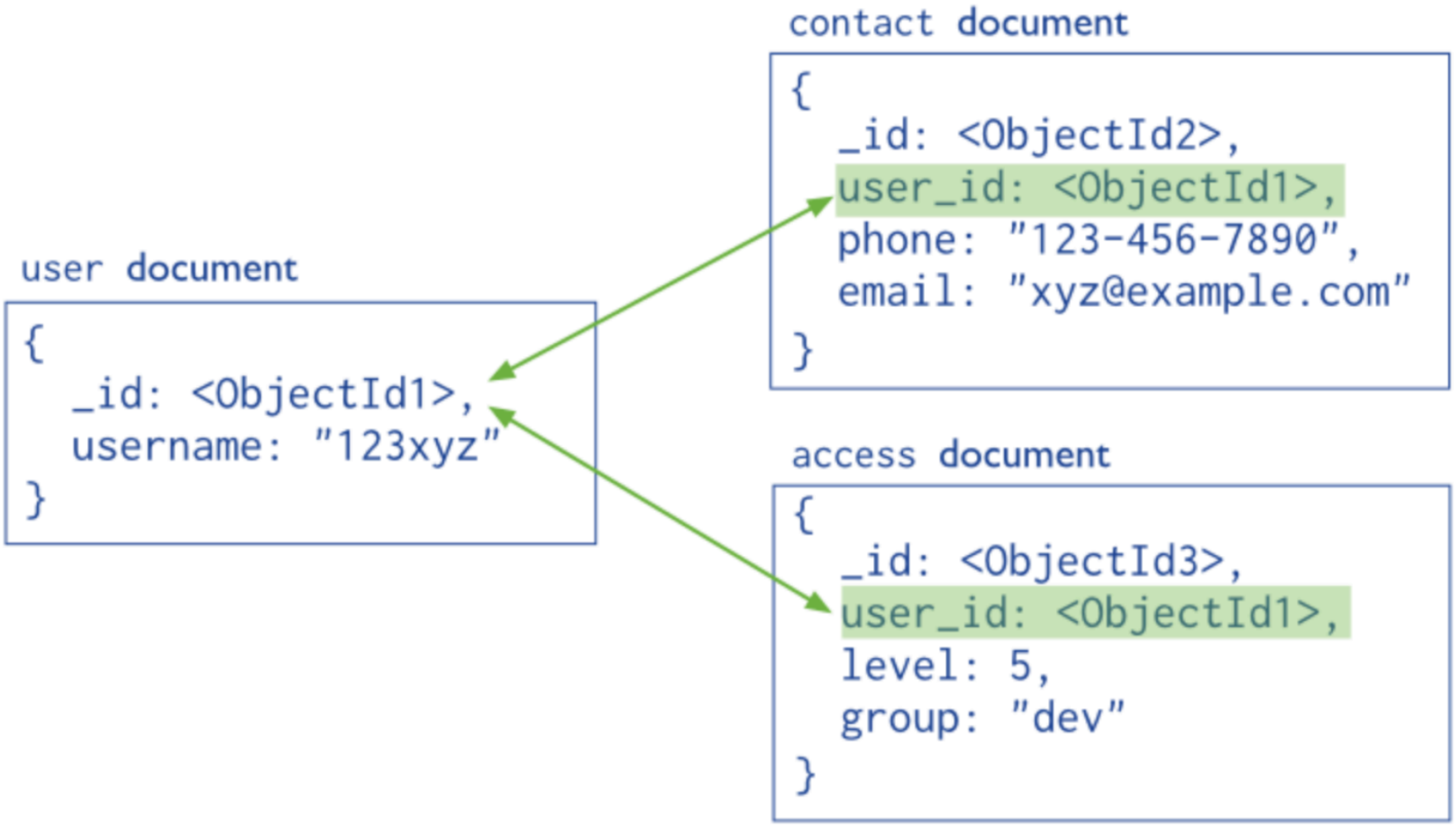


RELATIONSHIPS

- Relationships in MongoDB represent how various documents are logically related to each other. Relationships can be modeled via **Embedded** (Nested Document) and **Referenced** (Lookup) approaches. Such relationships can be either 1:1, 1:N, N:1 or N:N.
- Let us consider the case of storing addresses for users. So, one user can have multiple addresses making this a 1:N relationship.

EMBEDDED DATA MODELS

REFERENCED DATA MODELS



$LOOKUP (REFERENCED)

- $lookup is an aggregation stage in MongoDB that allows you to join data from one collection to another.
- It is similar to SQL JOIN (e.g., INNER JOIN or LEFT JOIN) but designed for document-oriented databases.
- Syntax

```
db.collection.aggregate([
  {
    $lookup: {
    from: "<target_collection>",       //Collection to join
    localField: "<field_in_current>", //Field from the current
    foreignField: "<field_in_target>",//Field in the target
      as: "<output_array_field>"       //Name of the field to
                                         store the joined data
    }
  }
```

$LOOKUP (REFERENCED)

- Example:

```
orders:
[
  { _id: 1, customer_id: 101, total: 500 },
  { _id: 2, customer_id: 102, total: 700 }
]

customers:
[
  { _id: 101, name: "Alice" },
  { _id: 102, name: "Bob" }
]
```

$LOOKUP (REFERENCED)

- Example (Cont.):

```
db.orders.aggregate([
  {
    $lookup: {
      from: “customers”,           //target collection
      localField: “customer_id”, //current->orders
      foreignField: “_id”,         //target->customers
      as: “customer_info”          //new field name
    }
  }
]);
```

$LOOKUP (REFERENCED)

- Result:

```
{
  _id: 1,
  customer_id: 101,
  total: 500,
  customer_info: [
    {
      _id: 101,
      name: 'Alice'
    }
  ]
}
```

```
{
  _id: 2,
  customer_id: 102,
  total: 700,
  customer_info: [
    {
      _id: 102,
      name: 'Bob'
    }
  ]
}
```

GUIDELINES

- Embed when:
  - Related data is always accessed together.
  - The data is small and bounded. (1:1 or 1:Few)
  - You want simpler queries and faster reads.
- Reference + $lookup when:
  - Related data grows independently.
  - You want to avoid large documents.
  - Data changes frequently, and you want minimal duplication.
  - You need flexibility in querying relationships.

DATA MODELING GUIDELINES

Factor	Embedding	Referencing
Relationship	One-to-one or one-to-few	One-to-many or many-to-many
Access	Always read together	Accessed separately or on demand
Size	Small and bounded	Large, unbounded, or frequently growing
Update	Changes rarely and usually as a whole	Updated frequently or independently
Read Performance	Need fastest read performance (all data in one doc)	Read performance can tolerate join overhead
Write performance	Occasional large document writes are acceptable	Many independent writes or updates required
Duplication	Some data duplication acceptable	Want to avoid duplication or redundant storage
Growth	Embedded array won't grow indefinitely	Related data grows large or unbounded over time

TRADE-OFFS

- Using smaller documents containing more frequently-accessed data reduces the overall size of the working set.
- These smaller documents result in improved read performance for the data that the application accesses most frequently.

PRACTICE IN DESIGN

- Use the data from the **company** database to design an appropriate data structure in MongoDB.
  - customer
  - orders
  - salesman
- Choose either **embedding** or **referencing** and explain why you chose that format.
- Write at least two documents for each collection of sample data.

ASSIGNMENT 5

- จากฐานข้อมูล univdb ใน MySQL
- หากต้องการเก็บข้อมูลดังกล่าวใน MongoDB ควรออกแบบโครงสร้างการเก็บข้อมูลอย่างไร
- ให้แต่ละกลุ่มเลือก Embedding หรือ Referencing พร้อมให้เหตุผลว่าทำไมถึงเลือกรูปแบบดังกล่าว
- เขียนตัวอย่างข้อมูลมาอย่างน้อย collection ละ 2 documents
- ส่งตัวอย่างข้อมูลที่ออกแบบพร้อมคำอธิบายในรูปแบบ pdf