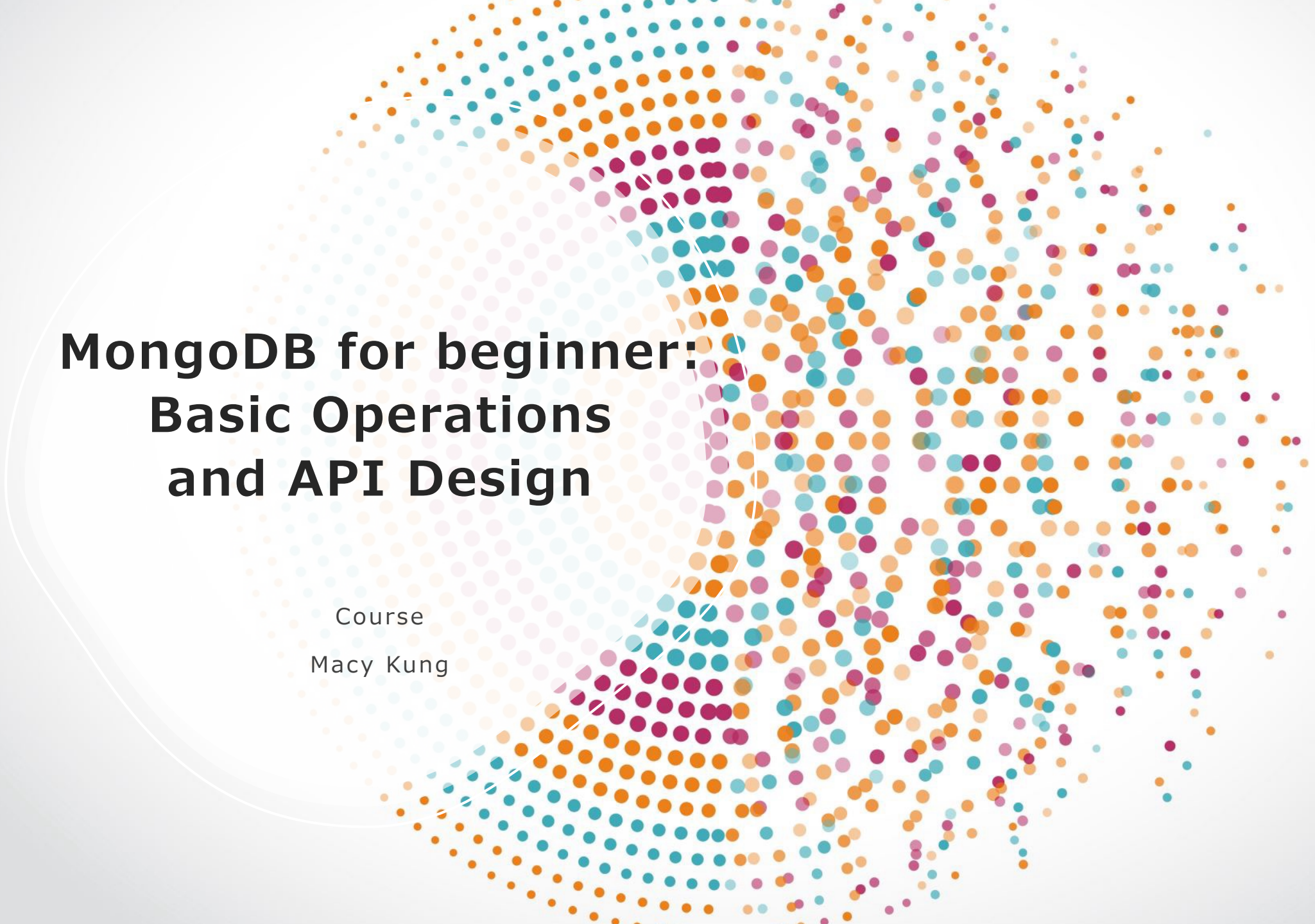


MongoDB for beginner: Basic Operations and API Design

Course

Macy Kung

Welcome to the Course

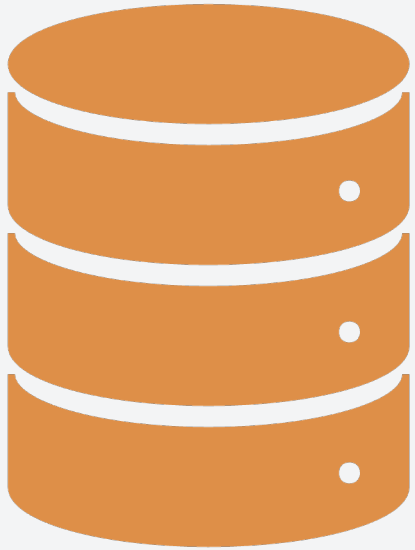
- Introduction to NoSQL and MongoDB
- Knowledge Notes
- Set Up & Installation
- Inserting Documents
- Mongo Query Language
- Deleting Documents
- Updating Documents
- Data Modeling
- Introduction to REST API

Introduction to NoSQL Database

What is a Database?

What is an SQL Database?

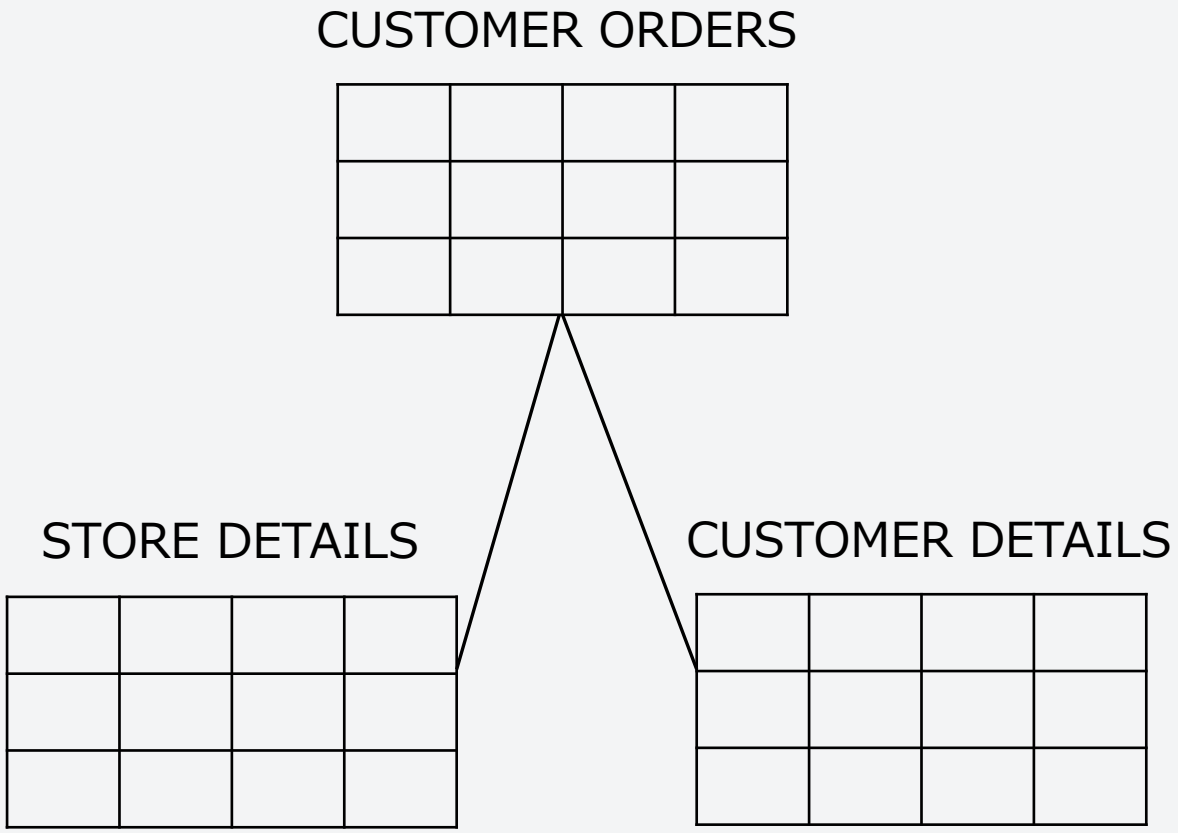
What is a NoSQL Database?



What is a Database

A database is defined as an organized collection of structured information or data typically stored electronically.

What is an SQL Database

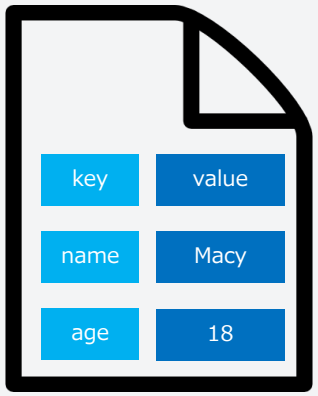


What is a NoSQL Database

NoSQL (also refers to Not only SQL, non-SQL or non-relational) is a database which gives you a way to manage the data which is in a non-relational form i.e. which is not structured in a tabular manner and does not possess tabular relationships.

Types of NoSQL Database

Document



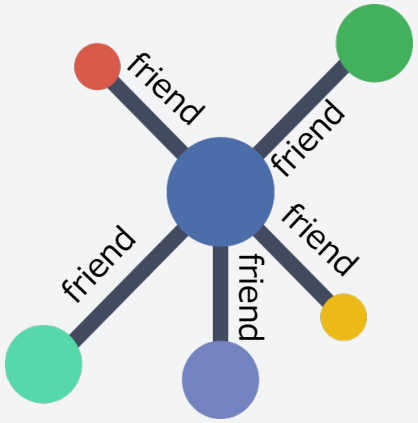
Apache, MongoDB, CouchDB

Key Value

key	value
Student_1	name@Laura @A4- tel@45789
Student_2	name@Chirs @B7- tel@12349

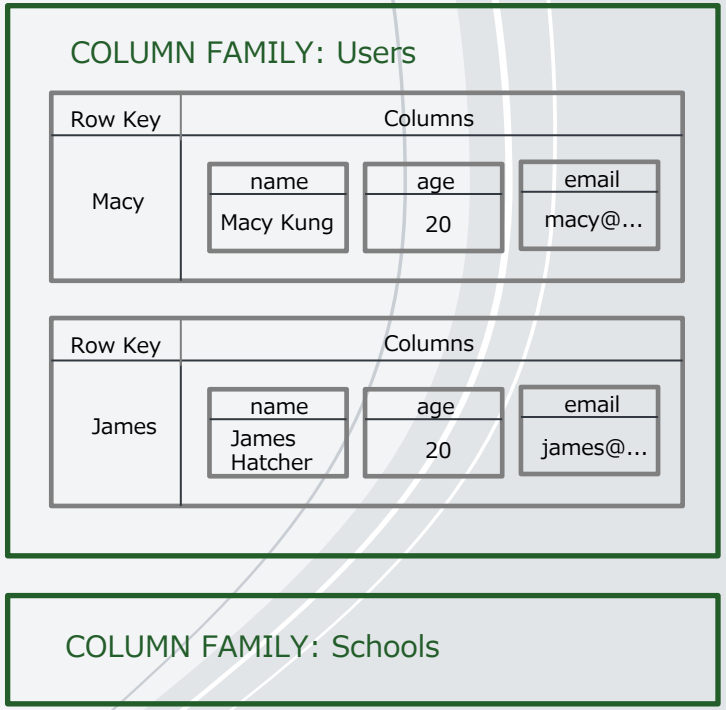
Redis, Dynamo, ZooKeeper

Graph



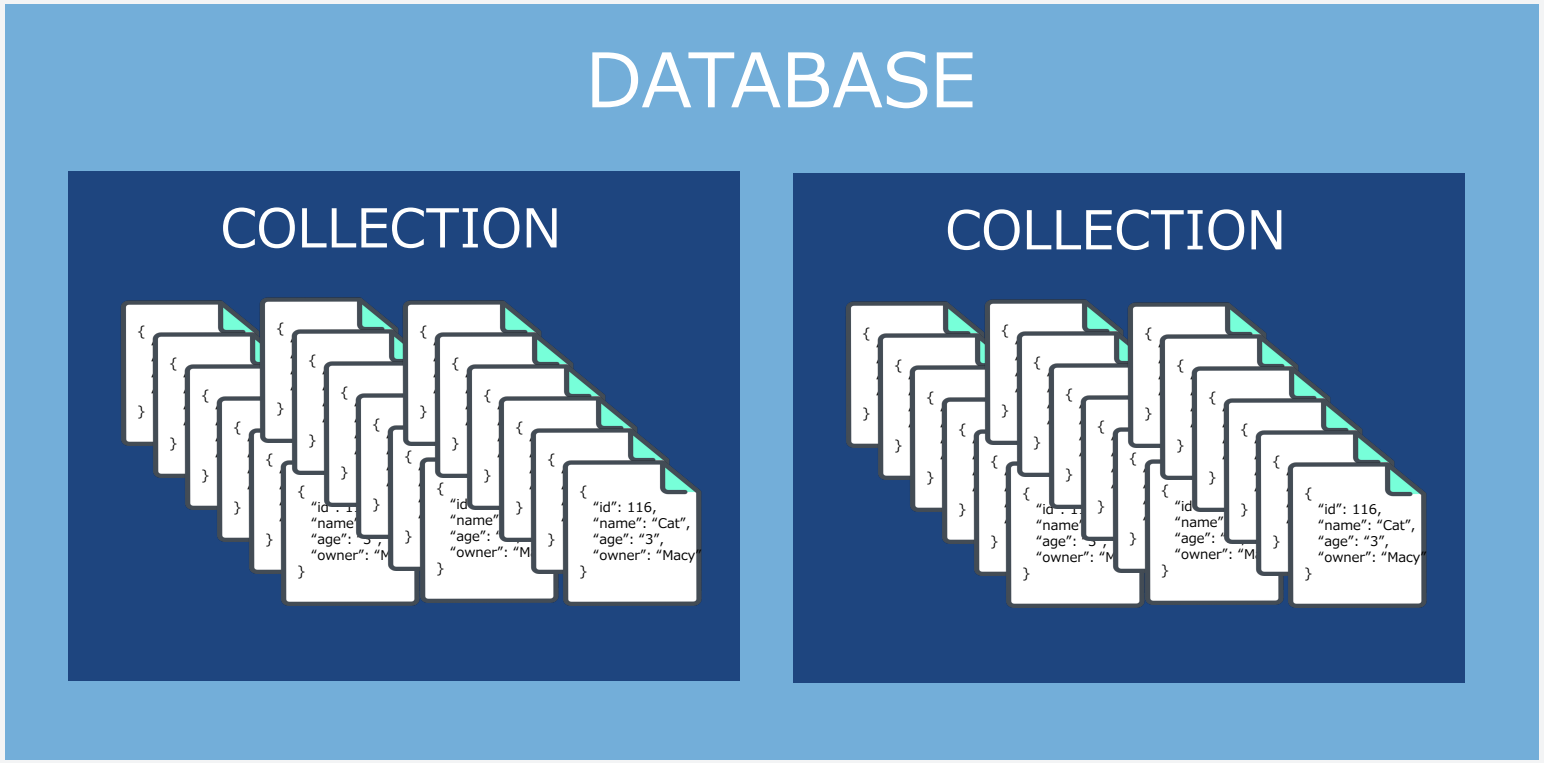
Neo4j, AllegroGraph, InfiniteGraph

Column Family



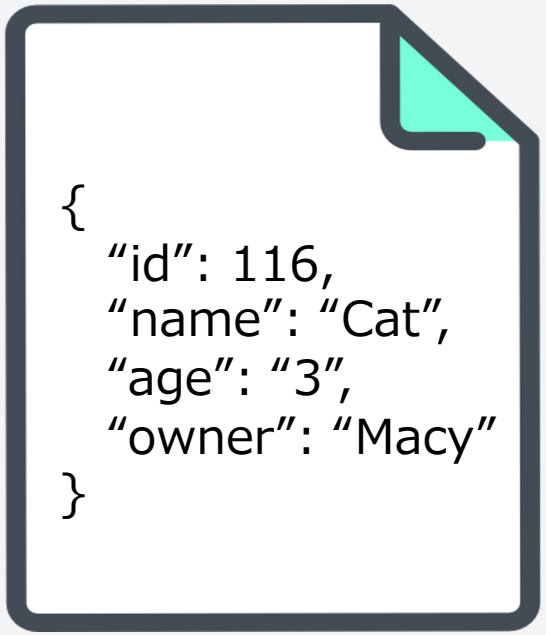
Cassandra, Hbase, Google BigTable

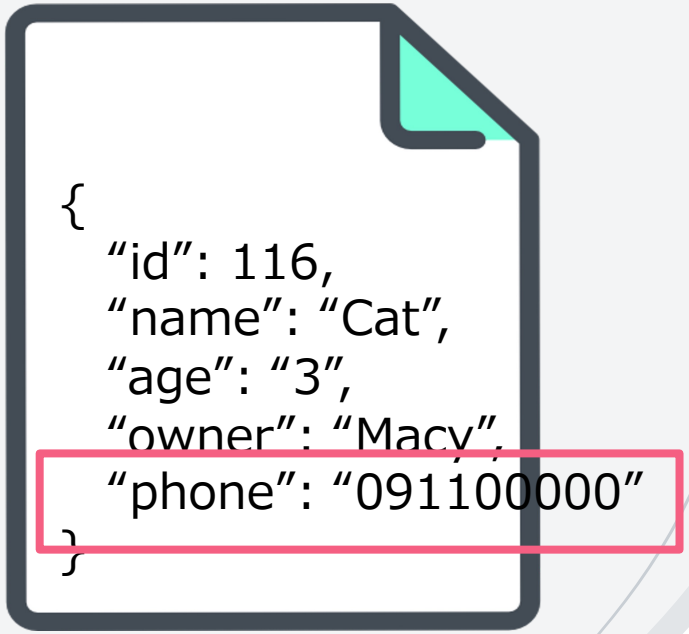
What is MongoDB




```
{
  “field”: “value”,
  “field”: “value”,
  “field”: “value”,
  “field”: “value”
}
```

What is MongoDB



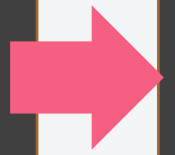


Documents are **polymorphic** i.e. they don't have a fixed structure. Changes can be made easily to an individual document structure e.g. new field value pairs can be added.

Previous example in SQL database

SQL

ID	NAME	AGE	OWNER
1	Cookie	3	Macy
2	Cin cin	7	Cindy
3	Dao	6	Lien
4	Pi	1	Yao

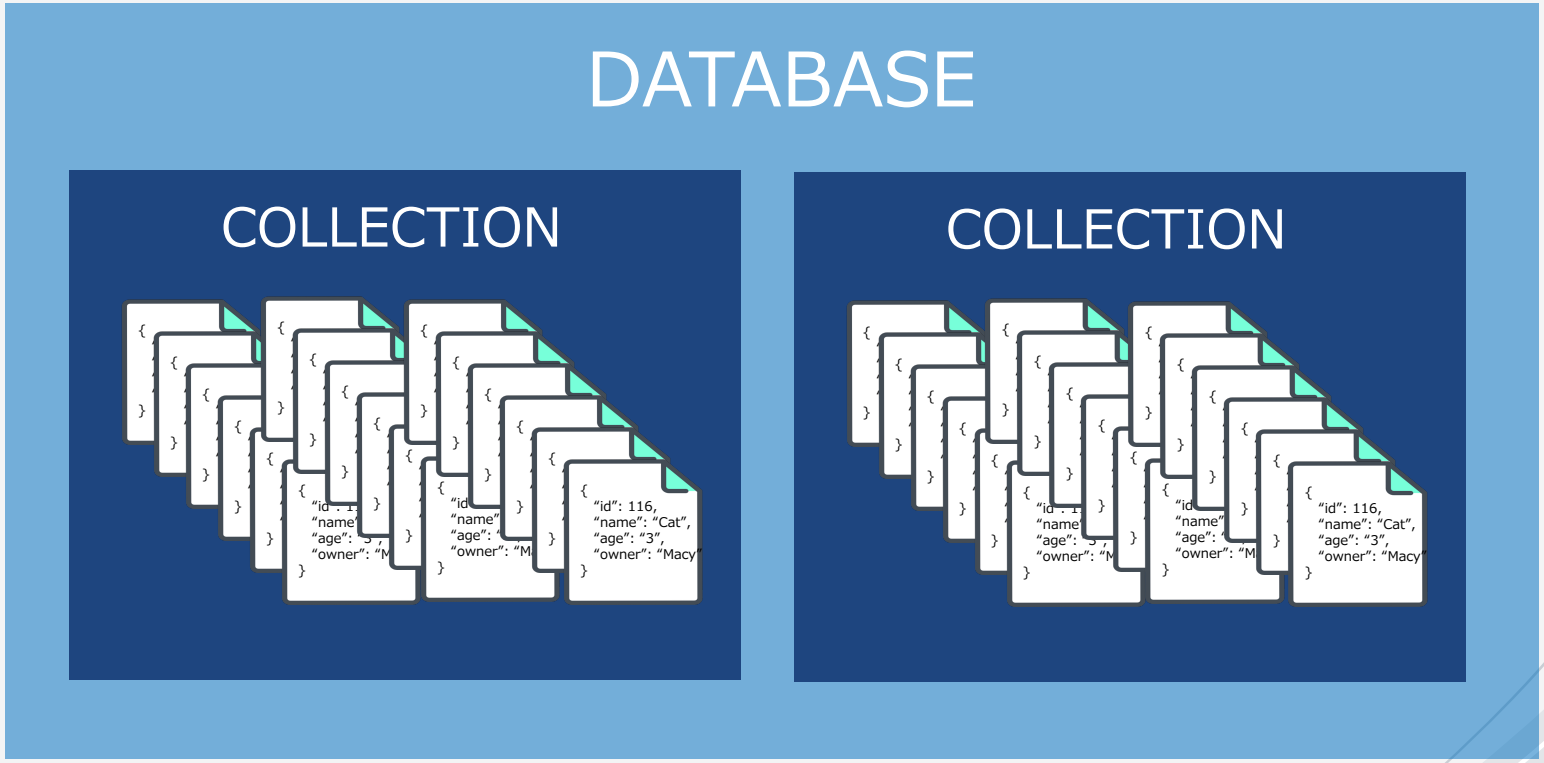


ID	NAME	AGE	OWNER	PHONE
1	Cookie	3	Macy	098100
2	Cin cin	7	Cindy	null
3	Dao	6	Lien	null
4	Pi	1	Yao	091888

What is MongoDB

In general, data is stored more efficiently in MongoDB than in SQL because it is schema-less resulting in less data redundancy and more efficiency

Documents in MongoDB



Documents in MongoDB

Documents in MongoDB

Documents in MongoDB

```
{
    "_id": ObjectId(11666),
    "name": "Cat",
    "phone": ["02-0034241", "091133442"]
    "address": {
        "street": "No.1 Neihu road",
        "city": "Taipei",
        "postal code": "114"
     }
}
```

An array is a data structure consisting of a collection of elements
With an array data type, we can store multiple values in a single key of the document.

Documents in MongoDB

```
{
    "_id": ObjectId(11666),
    "name": "Cat",
    "age": "3",
    "owner": "Macy"
}
```

```
{
    "_id": ObjectId(11666),
    "name": "Cat",
    "age": "3",
    "owner": "Macy",
    "phone": "091100000"
}
```

```
{
    "_id": ObjectId(11666),
    "name": "Cat",
    "age": "3",
    "owner": "Macy",
    "cell phone": "091100000"
}
```

Not the same field

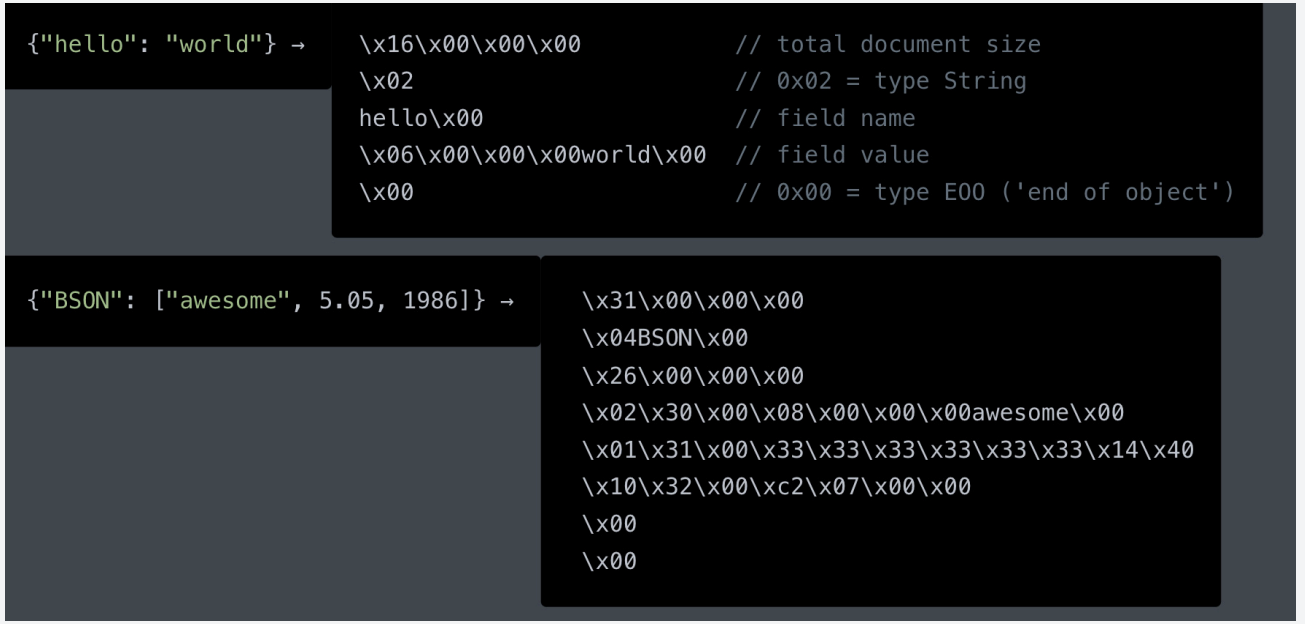
MongoDB - Datatypes

NUMERICAL:
Numbers such as integers and decimals

DATE:
Date and time values

STRING:
Charater / text values

👉 Datatypes Reference



What is BSON?
Does MongoDB use BSON, or JSON?

SQL vs NoSQL: comparison table

- SQL (Structured Query Language) is a traditional Relation database management system(RDBMS).
- SQL uses a fixed schema, where the structure of the tables must be defined before data can be inserted.
- In SQL, data is stored in tables with rows and columns.
- A great choice if you have structured data and need a traditional relational database

- MongoDB is a document-oriented NoSQL database.
- MongoDB uses a dynamic schema, where documents can have different fields.
- MongoDB uses a more expressive query language based on JSON
- In MongoDB, data is stored in collections of JSON-like documents.
- An ideal choice if you have unstructured and/or structured data with the potential for rapid growth

Replication

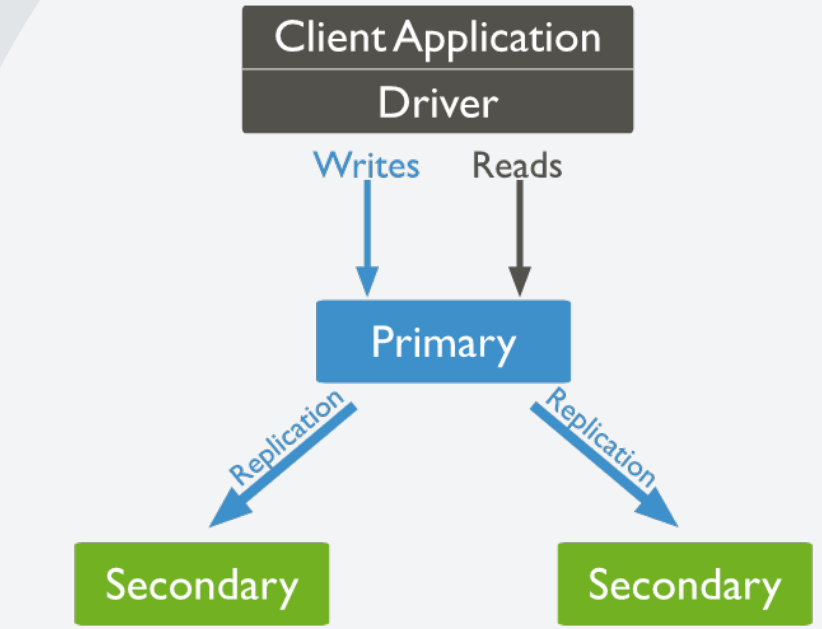
- A replica set in MongoDB is a group of **mongod** processes that maintain the same data set. Replica sets provide redundancy and high availability, and are the basis for all production deployments.

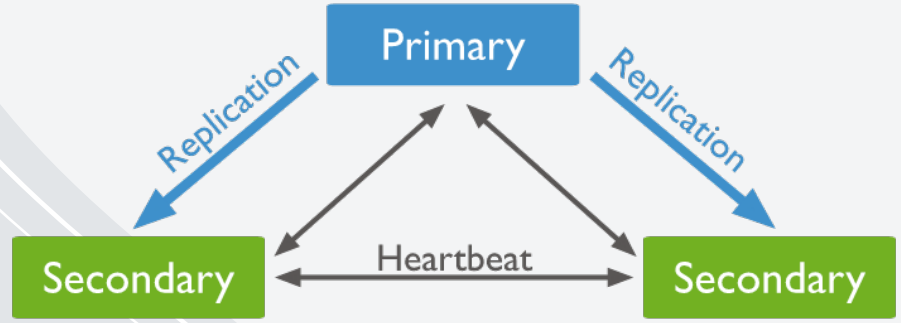
- There are 3 instance in a replica set and each instance contains a full copy of the data in each instance

Replication

- Of the data bearing nodes, one and only one member is deemed the primary node, while the other nodes are deemed secondary nodes. If the primary is unavailable, an eligible secondary will hold an election to elect itself the new primary.

- **mongod** is the primary daemon process for the MongoDB system. It handles data requests, manages data access, and performs background management operations.





Sharded Cluster

- Sharding is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations.

Sharded cluster consists of the following components

SHARD

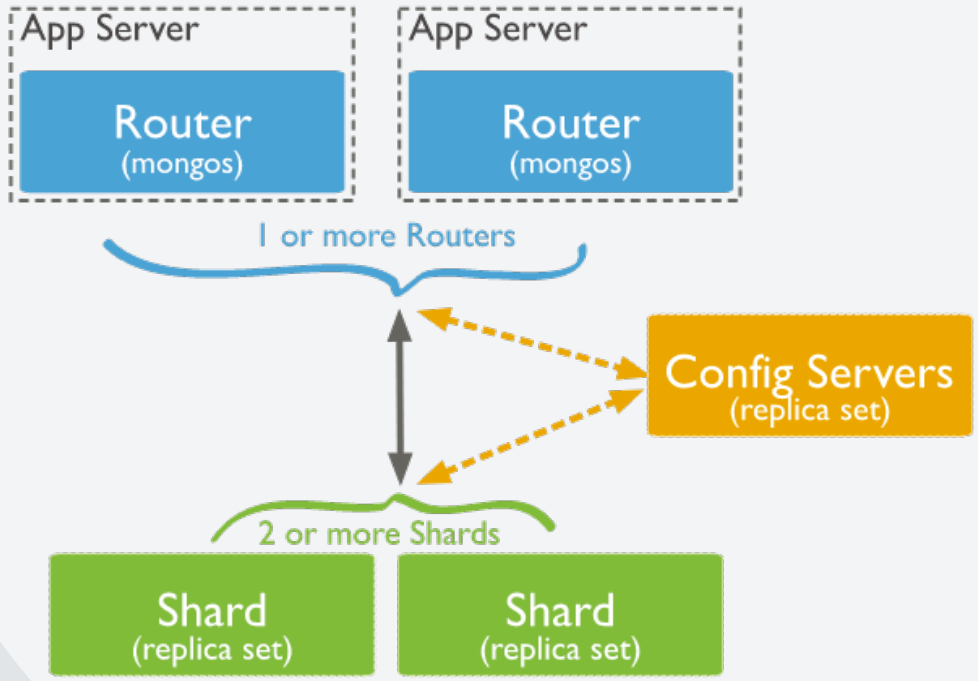
Containing a partition of the data

MONGOS

Interface between any application using MongoDB and the sharded cluster

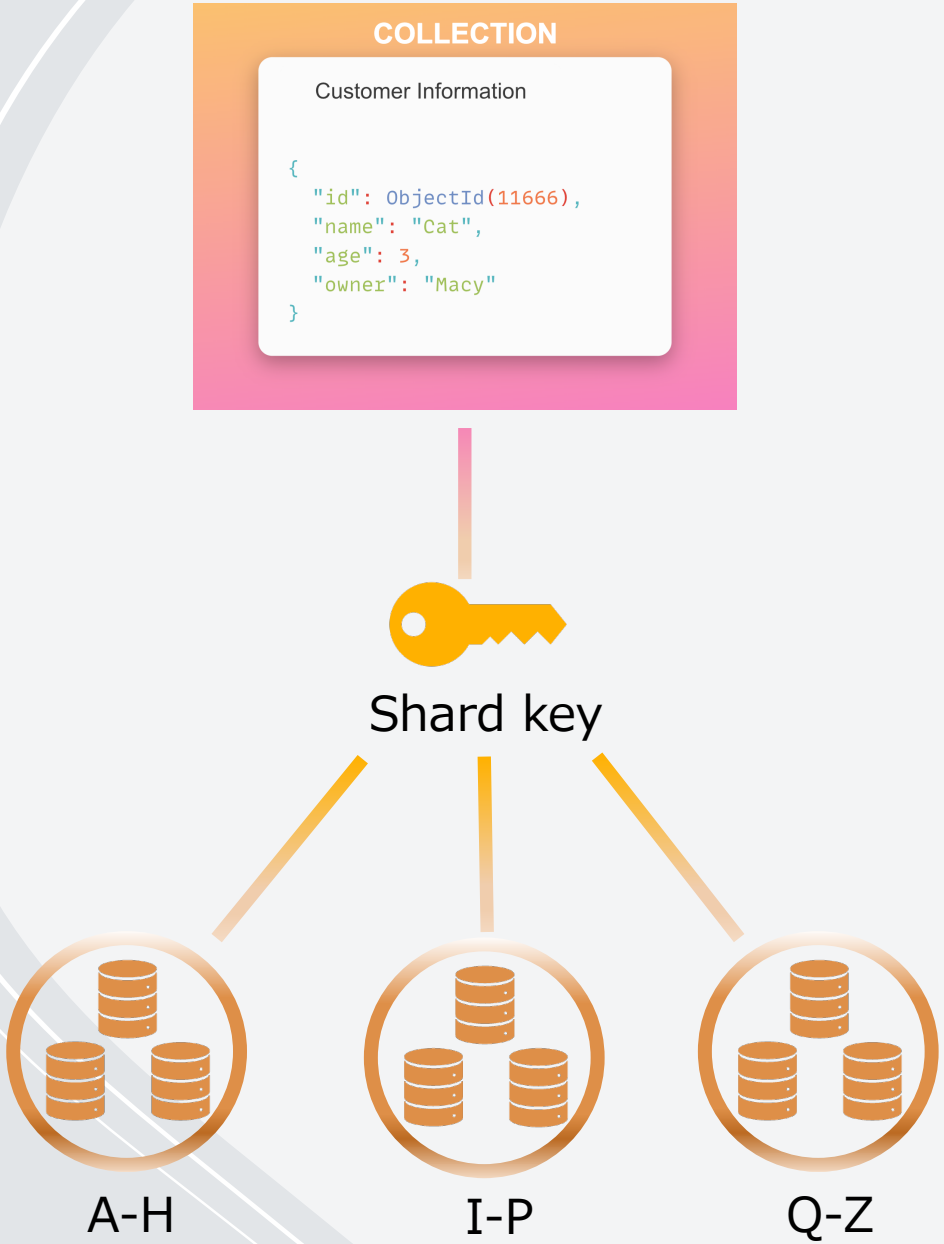
CONFIG SERVERS

Store metadata and configuration settings



Natively Distributed System

- MongoDB uses the **shard key** to distribute the collection's documents across shards.
- The shard key consists of a field or multiple fields in the documents.



You get a basic knowledge of MongoDB

- What’s NoSQL database
- MongoDB with format of document
- A replica set typically consists of __ machine(s)?
- A typical Relational Database has a fixed schema that requires defining the data structure before inserting data, which can be less flexible compared to MongoDB, which allows a more flexible and dynamic data structure with its schema-less design.

Advatages of MongoDB Database

- Less complexity vs relational databases
- Schema-less
- Easier to maintain
- Easy to scale

When to use NoSQL

- To handle a huge volume of structured, semi-structured and unstructured data.
- If your relational database is not capable enough to scale up to your traffic at an acceptable cost.
- If you want to have an efficient, scale-out architecture in place of an expensive and monolithic architecture.
- If you have local data transactions that need not be very durable.
- If you are going with schema-less data and want to include new fields without any ceremony.
- When your priority is easy scalability and availability.

When to Avoid NoSQL

- In such cases like financial transactions, etc., you may go with SQL databases.
- If consistency is a must and if there aren’t going to be any large-scale changes in terms of the data volume, then going with the SQL database is a better option.

Set Up & Installation

- Click Try Free, SignUp to MongoDB Atlas with Google Account
- MongoDB Atlas Quick Tour
- Create database
- Create Users
- Network access
- Connect to MongoDB using VS Code
- How can I connect to my MongoDB database from VS Code?

Create Database

When using the use directive to specify a database, if the database does not exist, MongoDB will automatically create it.

Create Database

```
use DATABASE_NAME
```

Create Database

```
> use test_tv
switched to db test_tv
> db
test_tv
>
```

Create Collection

Collections are similar to tables in a relational database. The steps to create a collection are as follows:

Create Collection

```
use test_tv
db.createCollection("dramas")
```

Inserting Documents - insertOne

insertOne

```
db.collection.insertOne(
  <document or array of documents>
)

Inserts a single document into a collection.
Returns an object that contains
the status of the operation.
```

Return :

```
db.dramas.insertOne(
  {
    "_id": ObjectId("a23t4464732"),
    "name": "腦筋急轉彎2",
    "episodes": [
      {
        "season": 1,
        "episode": 1,
        "link": "http://link1.com",
        "description": "第1集簡介"
      }
    ],
    "end_date": "2024-07-30 00:00:00",
    "release_date": "2024-06-13 00:00:00",
    "description": "深入探索踏上青少女時期的萊莉內心，全新情緒角色也將全面解鎖",
    "poster": "http://poster1.jpg",
    "category": "電影"
  }
)
```

Inserting Documents - insertMany

insertMany

```
db.collection.insertMany(
  <document or array of documents>
)
```

Inserts multiple documents into a collection.

Return :

```
db.dramas.insertMany(
  {
    "name": "腦筋急轉彎2",
    "episodes": [
      {
        "season": 1,
        "episode": 1,
        "link": "http://link1.com",
        "description": "第1集簡介"
      }
    ],
    "end_date": "2024-07-30 00:00:00",
    "release_date": "2024-06-13 00:00:00",
    "description": "深入探索踏上青少女時期的萊莉內心，全新情緒角色也將全面解鎖",
    "poster": "http://poster1.jpg",
    "category": "電影"
  },
  {
    "name": "荒野機器人",
    "episodes": [
      {
        "season": 1,
        "episode": 1,
        "link": "http://link1.com",
        "description": "第1集簡介"
      }
    ],
    "end_date": "2024-11-20 00:00:00",
    "release_date": "2024-10-10 00:00:00",
    "description": "一個機器人簡稱羅茲的羅茲森7134號機器人，因為船難被困在一座無人荒島，...",
    "poster": "http://poster1.jpg",
    "category": "電影"
  }
)
```

Using the find Methods

find

```
db.collection.find({query},{projection})

Returns all documents in a collection
Query and projection are optional
- Query : Specifies selection filter.
- Projection: Specifies the fields to return
  in the documents that match the query filter.
  1: true, query this field
  0: false, ignore this field
```

find

```
1 MongoQL:
2 db.dramas.find({"category": "電影"},{"name":1})
3
4 Return:
5 [
6   {_id:ObjectId("dshth343"), name: "腦筋急轉彎2"},
7   {_id:ObjectId("dsadaw4t"), name: "荒野機器人"}
8 ]
9
```

Using the findOne Methods

findOne

```
db.collection.findOne({query},{projection})

Returns one document from the collection
If multiple documents satisfy the query,
the methods returns the first document.
```

find

```
1 MongoQL:
2 db.dramas.findOne({"category": "電影"},{"name":1})
3
4 Return:
5 [
6   {_id:ObjectId("dshth343"), name: "腦筋急轉彎2"}
7 ]
8
```

Case Sensitivity in MongoDB

Objects in MongoDB are case sensitive.

For example, the findOne method (performed in the sample_training database on the shoes collection) should be as follows:

db.dramas.findOne() – this is the correct syntax

Db.dramas.findOne() – will not work because Db contains an upper case character

db.Dramas.findOne() – will not work because the collection is saved as “trips” not “Trips”, case sensitivity matters

db.dramas.FindOne() – will not work because case sensitivity also matters on methods. It should be “findOne “not “FindOne”

Which of the following is correct

The findOne method performed in the database on the shoes collection

a. db.dramas.findOne()

b. Db.dramas.findOne()

c. db.Dramas.findOne()

d. db.dramas.FindOne()

e. All of the above

From creating a database to querying data

1. **Create Databse**

   use test_tv

2. **Create Collection**

   db.createCollection("dramas");

3. **Insert data**

   db.dramas.insertMany([{ name: "Squid Game",

   category: "亞洲",…]);

4. **Query Data : 動漫**

   db.dramas.find({ category: "動漫" });

Material: https://github.com/windwaterbo/mongodb_coding_for_beginners/blob/main/2024/practice.md

Use of $ in Mongo query language

- Precedes operators in Mongo query language
- Precedes field values
- Aggregation Pipeline

Equal to

{“size”:{$eq:”UK8”}}

{field:{operator : value}}

Logical Operators

$and

Return all documents that match all conditions

$or

Return all documents that match any of the conditions

$nor

Return all documents that fail all of the conditions

{operator: [{condition1},{condition2},..]}

$not

Return all documents that do not match the expression

{operator: [{condition1}]}

Comparison Operators

$eq

Equal to
{“size”:”UK8”}
{“size”:{$eq:”UK8”}}

$ne

Not equal to

$gt

Greater than

$gte

Greater than
Or Equal to

$lt

Less than

$lte

Less than
Or equal to

$in

values specified in an array
{“name”:{$in:[“寄生蟲”]}}

$nin

Not in

Elements Operators

$exists

Return documents that contain the specified field

db.collection.find({field:{$exists:<boolean>}})

$type

Return documents that contain values of a specified data type

db.collection.find({field:{$type:<BSON type>}})

db.collection.find({field:{$type:[<BSON type1>, <BSON type2>,..]}})

Upcoming movie list query

1. **Query Data：即將上線**

```
db.dramas.find({release_date: { $gte: "2024-05-24 00:00:00" }})
```

Query Embedded Documents

You can perform read operations using the db.collection.find() method

- Match an Embedded/Nested Document
- Specify Equality Match on a Nested Field
- Specify Match using Query Operator
- Specify AND Condition

Match an Embedded/Nested Document

To specify an equality condition on a field that is an embedded/nested document, use the query filter document

{ <field>: <value>}

where <value> is the document to match.

```
db.inventory.find({ size: { h: 14, w: 21, uom: "cm" } })
```

documents

```
[
  { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
  { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
  { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
  { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
  { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]
```

Specify Equality Match on a Nested Field

The following example selects all documents where the field uom nested in the size field equals "in"

```
db.inventory.find({ "size.uom": "in" })
```

documents

```
[
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]
```

Specify Match using Query Operator

A query filter document can use the query operators to specify conditions in the following form:

{ <field1>: { <operator1>: <value1> }, ... }

```
db.inventory.find({ "size.h": { $lt: 15 } })
```

documents

```
[
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]
```

Specify AND Condition

The following query selects all documents where the nested field h is less than 15, the nested field uom equals "in", and the status field equals "D"

```
db.inventory.find({ "size.h": { $lt: 15 }, "size.uom": "in", status: "D" })
```

documents

```
[
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]
```

Query an Array of Embedded Documents

- Query for a Document Nested in an Array
- Specify a Query Condition on a Field Embedded in an Array of Documents
- Use the Array Index to Query for a Field in the Embedded Document
- A Single Nested Document Meets Multiple Query Conditions on Nested Fields
- Combination of Elements Satisfies the Criteria

Query for a Document Nested in an Array

The following example selects all documents where an element in the instock array matches the specified document

```
db.inventory.find( { "instock": { warehouse: "A", qty: 5 } } )
```

documents

```
[
   { item: "journal", instock: [ { warehouse: "A", qty: 5 }, { warehouse: "C", qty: 15 } ] },
   { item: "notebook", instock: [ { warehouse: "C", qty: 5 } ] },
   { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 15 } ] },
   { item: "planner", instock: [ { warehouse: "A", qty: 40 }, { warehouse: "B", qty: 5 } ] },
   { item: "postcard", instock: [ { warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 } ] }
)
```

Specify a Query Condition on a Field Embedded in an Array of Documents

If you do not know the index position of the document nested in the array, concatenate the name of the array field, with a dot (.) and the name of the field in the nested document.

The following example selects all documents where the instock array has at least one embedded document that contains the field qty whose value is less than or equal to 20

```
db.inventory.find( { 'instock.qty': { $lte: 20 } } )
```

documents

```
[
   { item: "journal", instock: [ { warehouse: "A", qty: 5 }, { warehouse: "C", qty: 15 } ] },
   { item: "notebook", instock: [ { warehouse: "C", qty: 5 } ] },
   { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 15 } ] },
   { item: "planner", instock: [ { warehouse: "A", qty: 40 }, { warehouse: "B", qty: 5 } ] },
   { item: "postcard", instock: [ { warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 } ] }
)
```

Use the Array Index to Query for a Field in the Embedded Document

Using dot notation, you can specify query conditions for field in a document at a particular index or position of the array. The array uses zero-based indexing.

The following example selects all documents where the instock array has as its first element a document that contains the field qty whose value is less than or equal to 20

```
db.inventory.find( { 'instock.0.qty': { $lte: 20 } } )
```

documents

```
[
   { item: "journal", instock: [ { warehouse: "A", qty: 5 }, { warehouse: "C", qty: 15 } ] },
   { item: "notebook", instock: [ { warehouse: "C", qty: 5 } ] },
   { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 15 } ] },
   { item: "planner", instock: [ { warehouse: "A", qty: 40 }, { warehouse: "B", qty: 5 } ] },
   { item: "postcard", instock: [ { warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 } ] }
)
```

A Single Nested Document Meets Multiple Query Conditions on Nested Fields

Use $elemMatch operator to specify multiple criteria on an array of embedded documents such that at least one embedded document satisfies all the specified criteria.

The following example queries for documents where the instock array has at least one embedded document that contains both the field qty equal to 5 and the field warehouse equal to A

```
db.inventory.find( { "instock": { $elemMatch: { qty: 5, warehouse: "A" } } } )
```

documents

```
[
   { item: "journal", instock: [ { warehouse: "A", qty: 5 }, { warehouse: "C", qty: 15 } ] },
   { item: "notebook", instock: [ { warehouse: "C", qty: 5 } ] },
   { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 15 } ] },
   { item: "planner", instock: [ { warehouse: "A", qty: 40 }, { warehouse: "B", qty: 5 } ] },
   { item: "postcard", instock: [ { warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 } ] }
)
```

The following example queries for documents where
the instock array has at least one embedded document that
contains the field qty that is greater than 10 and less than or equal to 20

```
db.inventory.find( { "instock": { $elemMatch: { qty: { $gt: 10, $lte: 20 } } } } )
```

documents

```
[
   { item: "journal", instock: [ { warehouse: "A", qty: 5 }, { warehouse: "C", qty: 15 } ] },
   { item: "notebook", instock: [ { warehouse: "C", qty: 5 } ] },
   { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 15 } ] },
   { item: "planner", instock: [ { warehouse: "A", qty: 40 }, { warehouse: "B", qty: 5 } ] },
   { item: "postcard", instock: [ { warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 } ] }
)
```

Combination of Elements Satisfies the Criteria

If the compound query conditions on an array field do not use the $elemMatch operator, the query selects those documents whose array contains any combination of elements that satisfies the conditions.

For example, the following query matches documents where any document nested in the instock array has the qty field greater than 10 and any document (but not necessarily the same embedded document) in the array has the qty field less than or equal to 20

```
db.inventory.find( { "instock.qty": { $gt: 10,  $lte: 20 } } )
```

documents

```
[
   { item: "journal", instock: [ { warehouse: "A", qty: 5 }, { warehouse: "C", qty: 15 } ] },
   { item: "notebook", instock: [ { warehouse: "C", qty: 5 } ] },
   { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 15 } ] },
   { item: "planner", instock: [ { warehouse: "A", qty: 40 }, { warehouse: "B", qty: 5 } ] },
   { item: "postcard", instock: [ { warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 } ] }
)
```

The following example queries for documents where the instock array has at least one embedded document that contains the field qty equal to 5 and at least one embedded document (but not necessarily the same embedded document) that contains the field warehouse equal to A

```
db.inventory.find( { "instock.qty": 5, "instock.warehouse": "A" } )
```

documents

```
[
   { item: "journal", instock: [ { warehouse: "A", qty: 5 }, { warehouse: "C", qty: 15 } ] },
   { item: "notebook", instock: [ { warehouse: "C", qty: 5 } ] },
   { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 15 } ] },
   { item: "planner", instock: [ { warehouse: "A", qty: 40 }, { warehouse: "B", qty: 5 } ] },
   { item: "postcard", instock: [ { warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 } ] }
)
```

Deleting Documents - deleteOne

deleteOne

```
db.collection.deleteOne(<filter>)

The first document that matches
 the filter condition will be deleted
```

deleteOne

MongoQL:

```
db.dramas.deleteOne({"_id": ObjectId("dsadaw4t")})
```

Return:

```
{ "acknowledged" : true, "deletedCount" : 1 }
```

Delete Documents - deleteMany

deleteMany

```
db.collection.deleteMany(<filter>)

All documents that match the condition
will be deleted.
```

deleteMany

MongoQL:

```
db.dramas.deleteMany({"category":"電影"})
```

Return:

```
{ "acknowledged" : true, "deletedCount" : 2 }
```

Updating Documents

updateOne

```
db.collection.updateOne(
 <filter>,<update>,<options>
)
```

Update one will update the first documents that match the filter criteria.

updateMany

```
db.collection.updateMany(
 <filter>,<update>,<options>
)
```

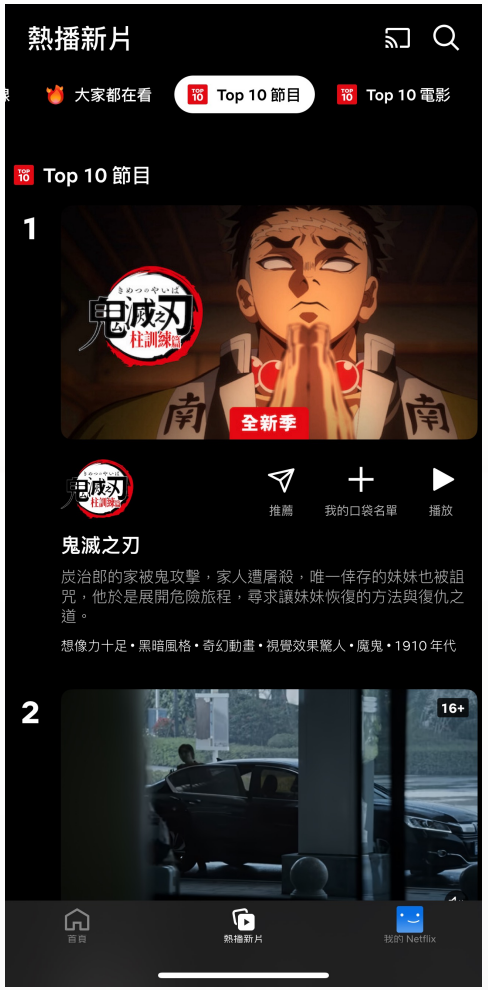
Update many will update all documents that match the filter criteria.

$set

db.collection.updateOne({filter},{$set:{field:value,…},{options}}

db.collection.updateMany({filter},{$set:{field:value,…},{options}}

🏠📝 Multi-document update exercise: add top10 tags to episodes



1. **Update Data : Top10**

```
db.dramas.updateMany(
{ name: { $in: ["淚之女王", "葬送的芙利蓮", "出租車司機", "肌肉魔法使"] } },
{ $set: { tag: ["top10"] } })
```

2. **Query Top10**

```
db.dramas.find({ tag: "top10" })
```

Update Operators

$unset

Removes the specified field from a document.

db.collection.updateMany({filter},{$unset:{field:value,…}},{options})

$inc

Increments the value of the field by the specified amount.

db.collection.updateMany({filter},{$inc:{field:value,…}},{options})

$rename

Renames a field.

db.collection.updateMany({filter},{$rename:{field:value,…}},{options})

$push

Adds an item to an array.

db.collection.updateMany({filter},{$push:{field:value,…}},{options})

$pull

Remove items from an array.

db.collection.updateMany({filter},{$pull:{field:value,…}},{options})

Please refer to this link for additional information on MongoDB Update Operators

Comparing SQL and NoSQL: Syntax Differences Between MySQL and MongoDB

MySQL	MongoDB
INSERT	
INSERT INTO dramas (name, category, release_date, end_date, description, poster) VALUES ('犯罪都市3：無法無天', '亞洲', '2024-06-02', '2024-12-30', '第三部《犯罪都市》系列，主要講述了"野獸警察"馬錫道對抗財閥繼承人和日本黑幫老大的故事。', 'http://poster1.jpg');	db.dramas.insertOne({ name: "犯罪都市3：無法無天", category: "亞洲", release_date: "2024-06-02", episodes: [{season: 1, episode: 1, link: "http://link1.com", description: "第1集簡介"},{season: 1, episode: 2, link: "http://link2.com", description: "第2集簡介"}], end_date: "2024-12-30", description: "第三部《犯罪都市》系列，主要講述了"野獸警察"馬錫道對抗財閥繼承人和日本黑幫老大的故事。", poster: "http://poster1.jpg" });
FIND	
SELECT category, release_date FROM dramas WHERE name = '犯罪都市3：無法無天';	db.dramas.find({name: "犯罪都市3：無法無天"}, {category: 1, release_date: 1});
UPDATE	
UPDATE dramas SET category = '動作' WHERE name = '犯罪都市3：無法無天';	db.dramas.update({name: "犯罪都市3：無法無天"}, {$set: {category: "動作"}});
DELETE	
DELETE FROM dramas WHERE category = '電影';	db.dramas.deleteMany({category: "電影"});

Data Modeling Considerations

- The right schema design can eanble you to make the most out of your database
- When designing your database it is important to consider:

Data Use and Performance

Determine important queries

Read vs Write

Data Model Design

Embedding vs Referencing

user document

```
{
  _id: <ObjectId1>,
  username: "Chan",
  contact: {
      phone: "0922111444",
      email: "chan0616@example.com"
  },
  access: {
      level: 5,
      group: "dev"
  }
}
```

user document

```
{
  _id: <ObjectId1>,
  username: "Chan"
}
```

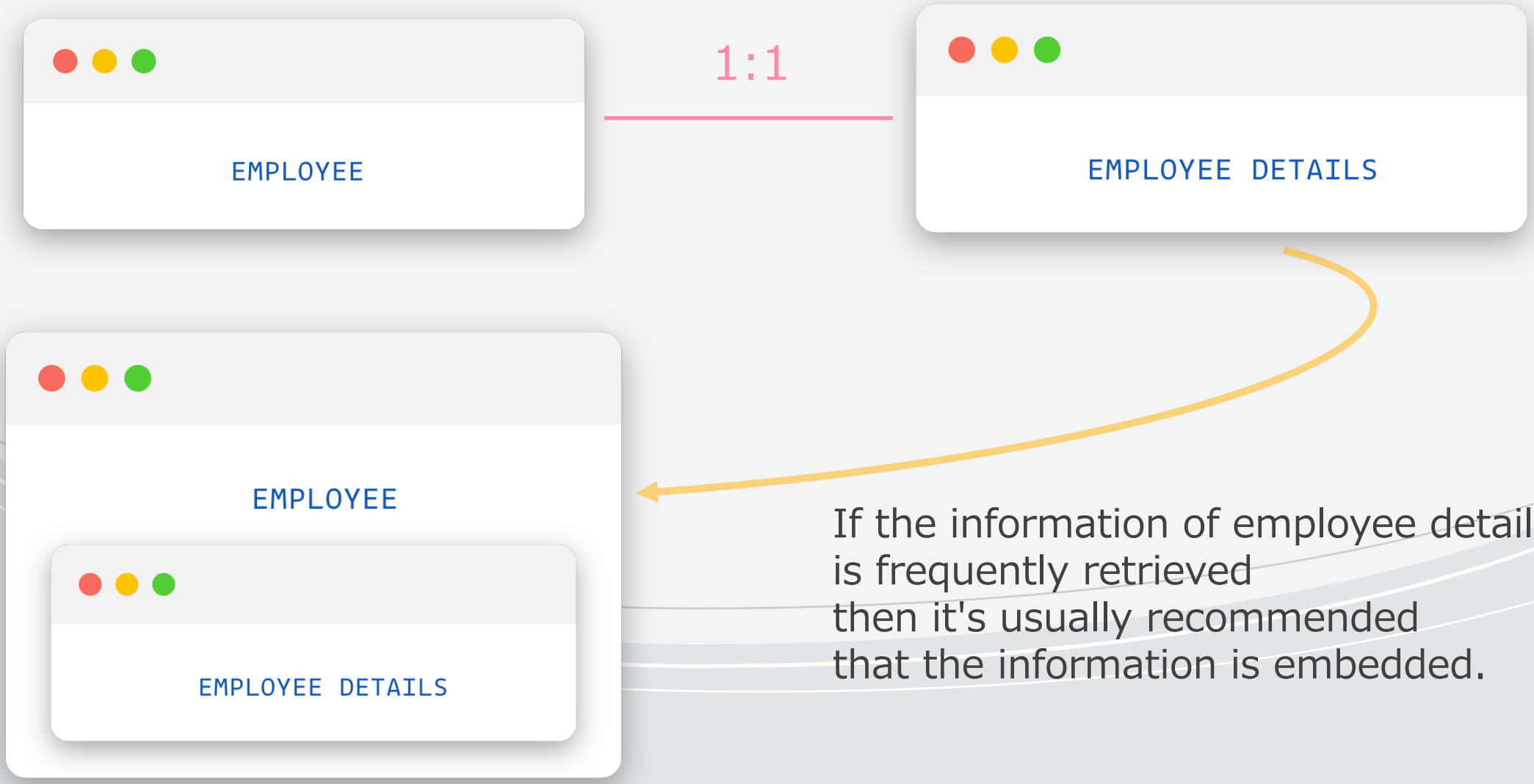
contact document

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "0922111444",
  email: "chan0616@example.com"
}
```

access document

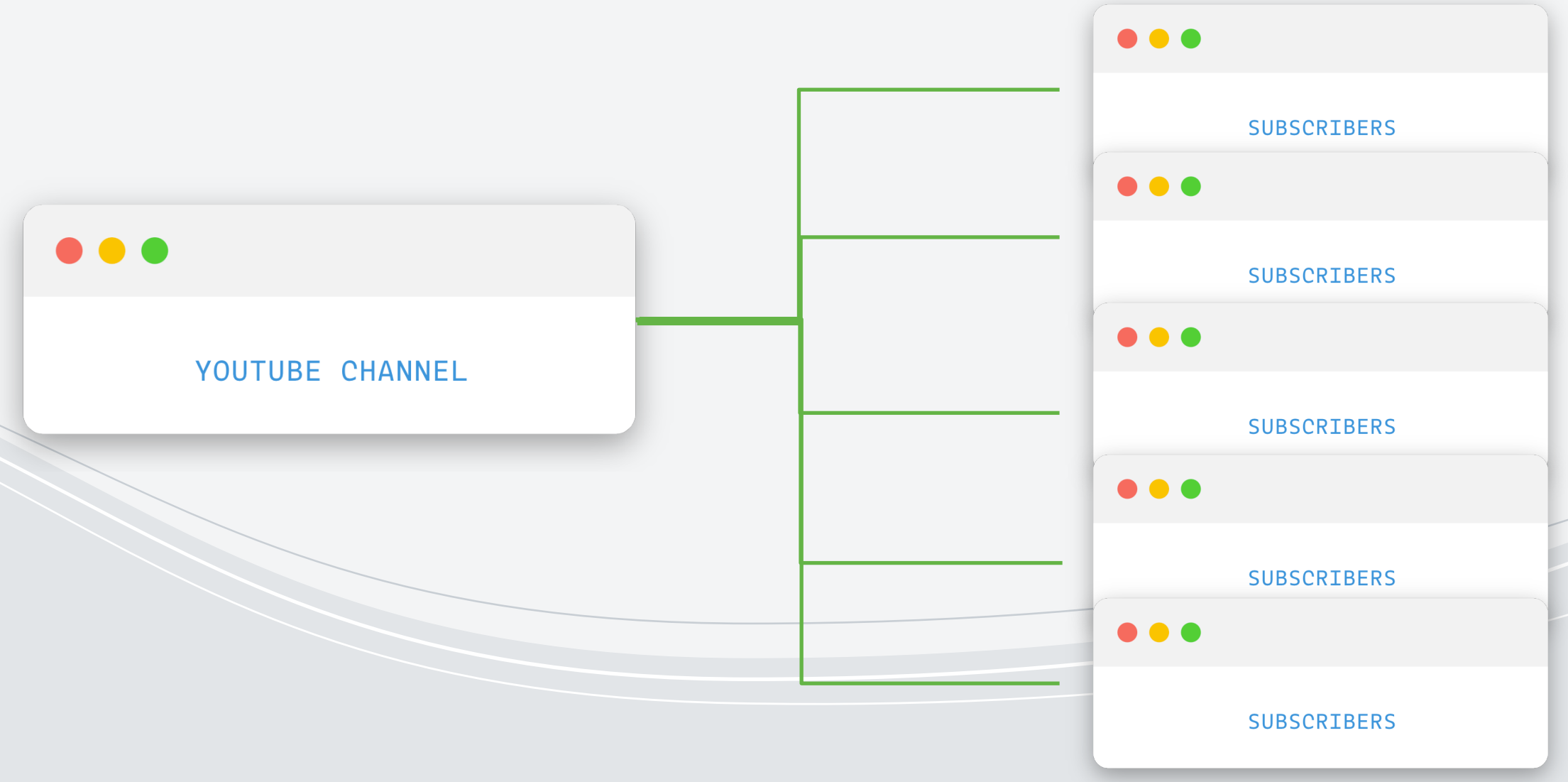
```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

1:1 Relationship

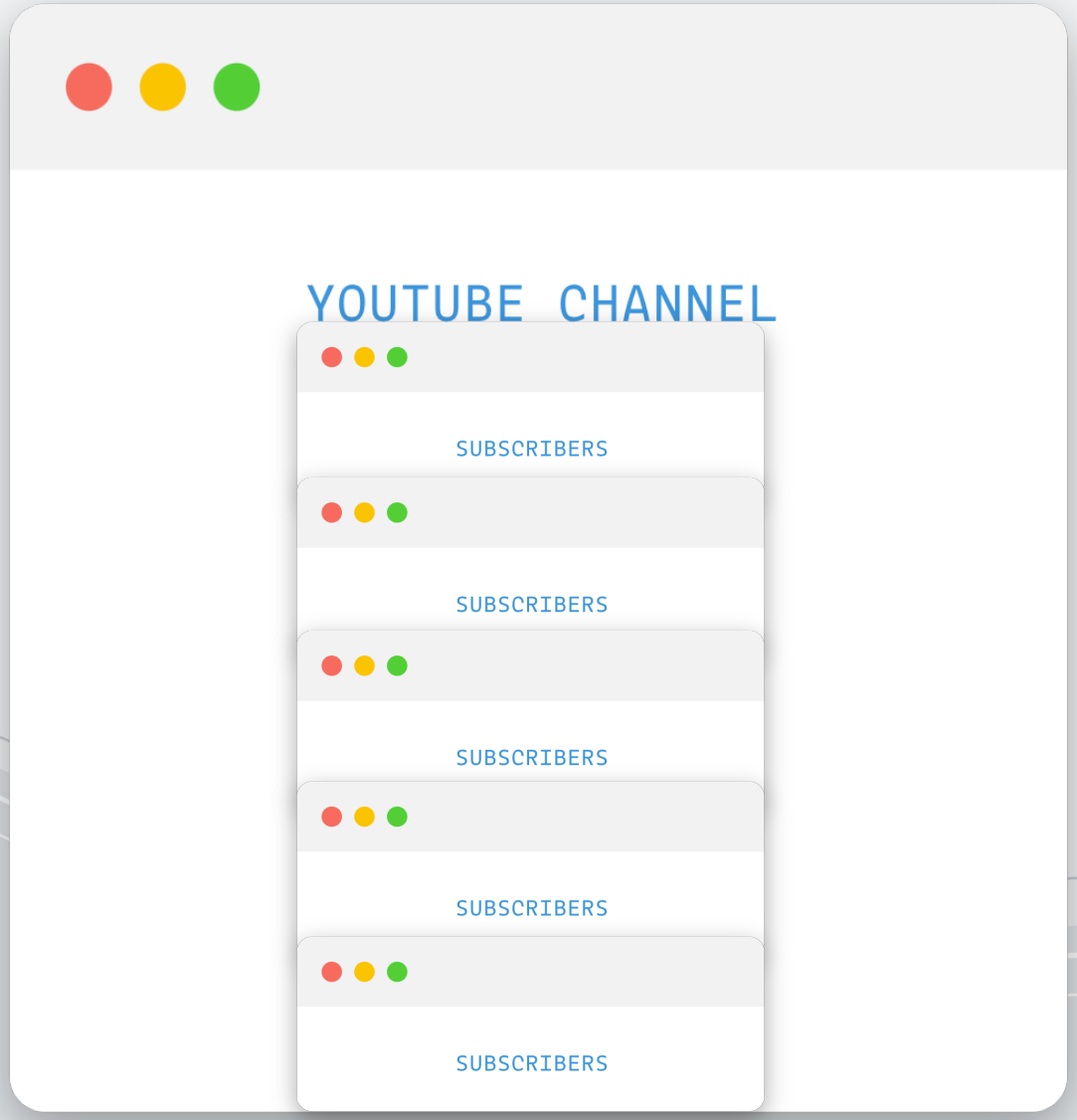


If the information of employee details
is frequently retrieved
then it's usually recommended
that the information is embedded.

1:Many Relationship

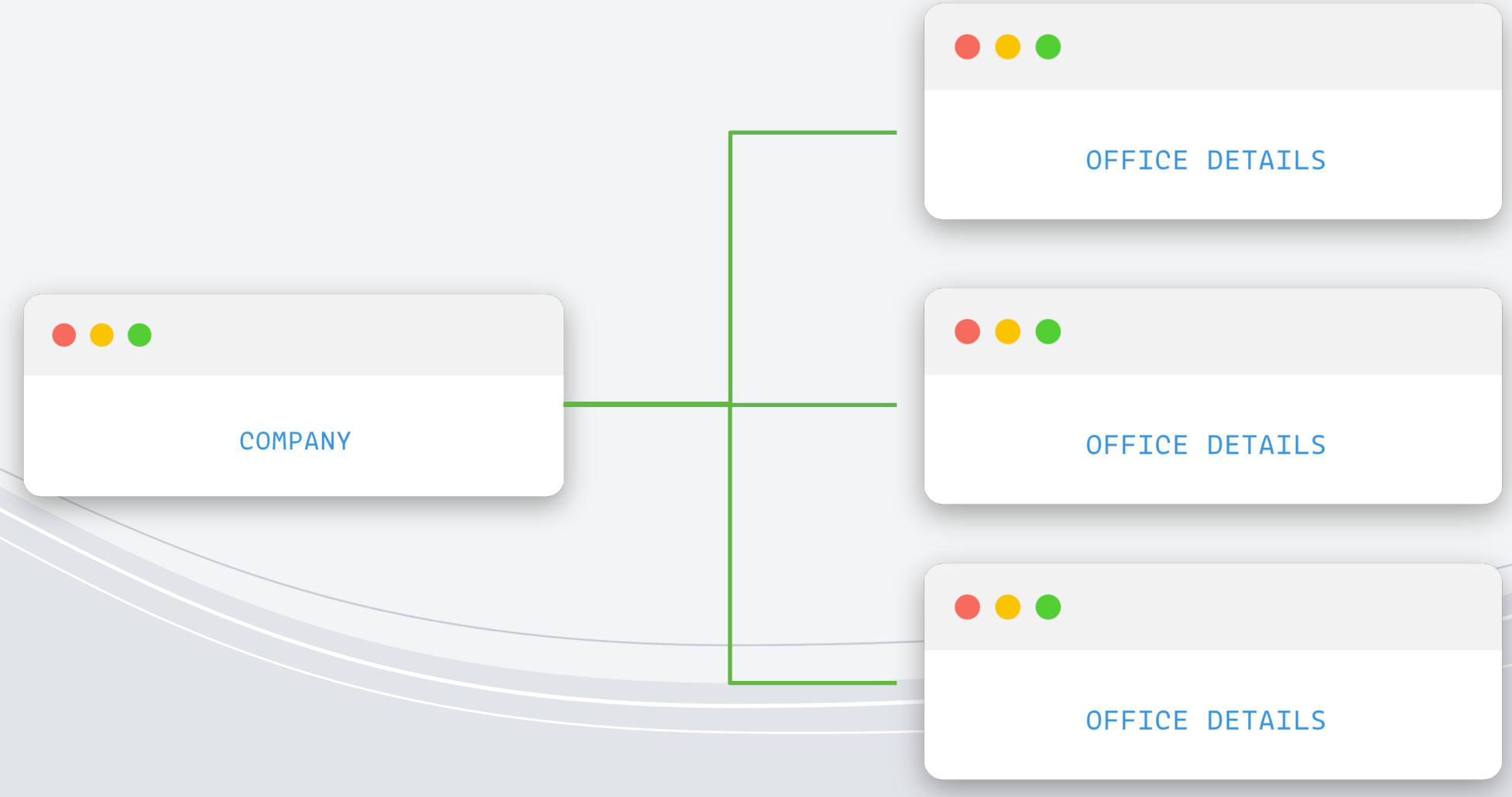


1:Many Relationship

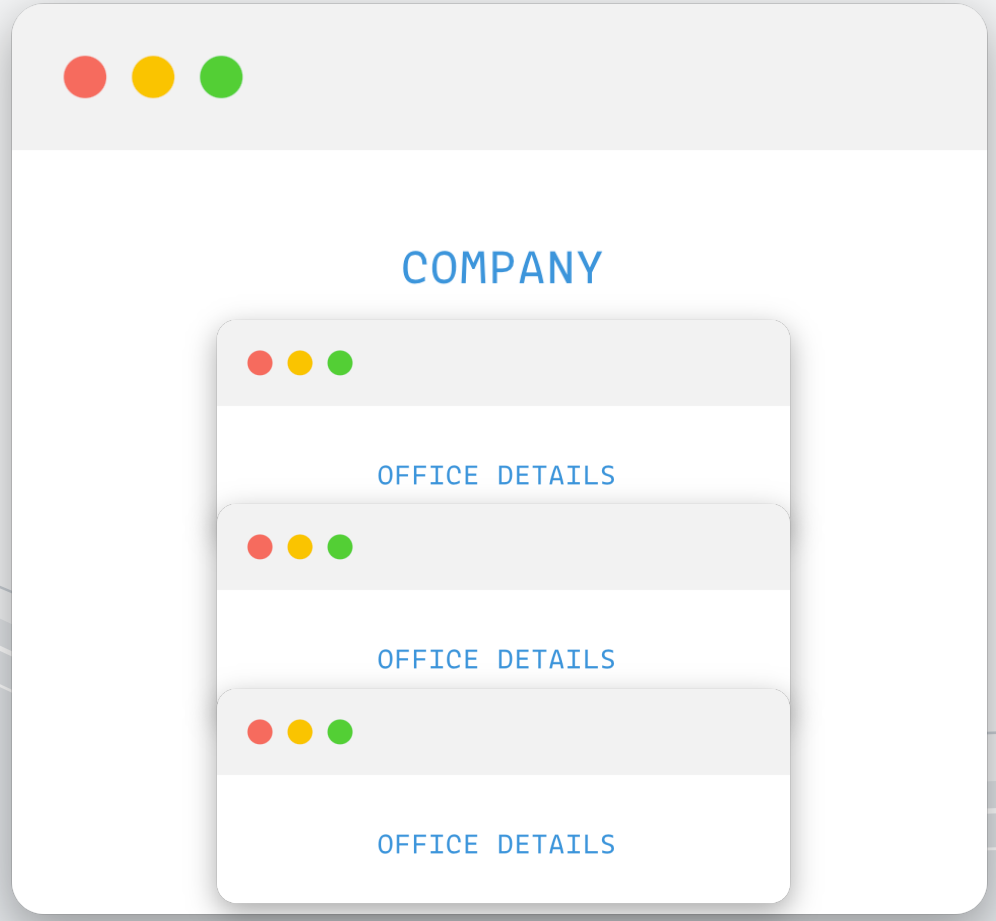


Too many embedded documents

1:Many Relationship

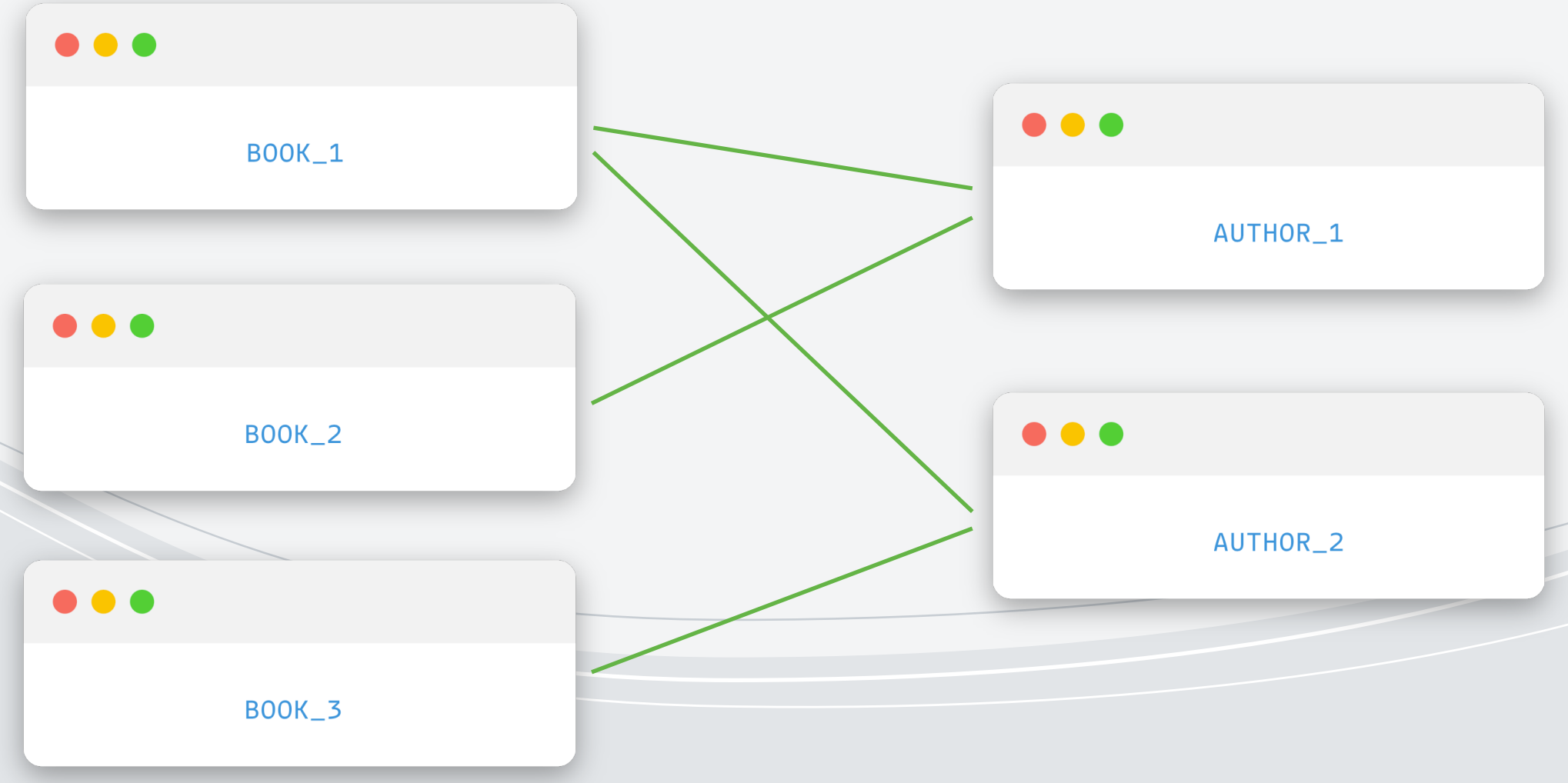


1:Many Relationship

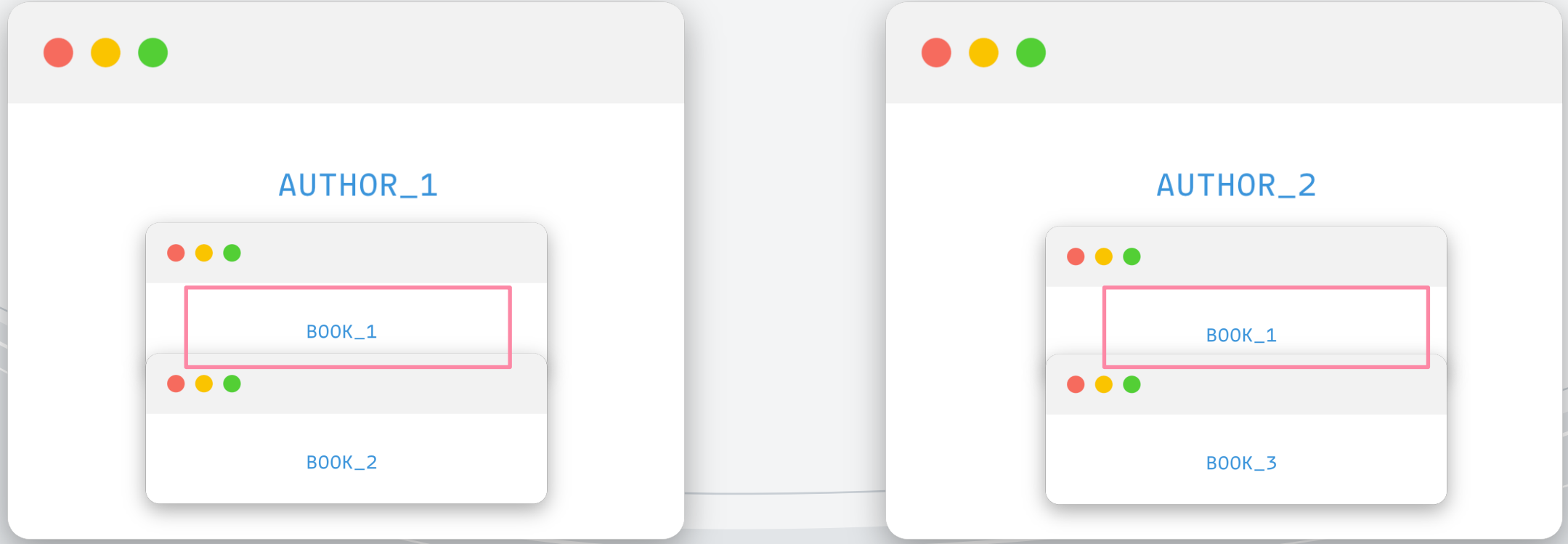


More suited to embedding documents

Many:Many Relationship



Many:Many Relationship



Building with Pattern

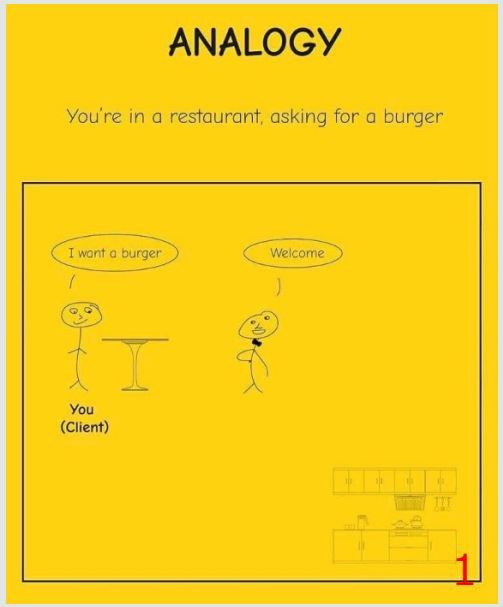
Here is a very useful resource on Schema Patterns.

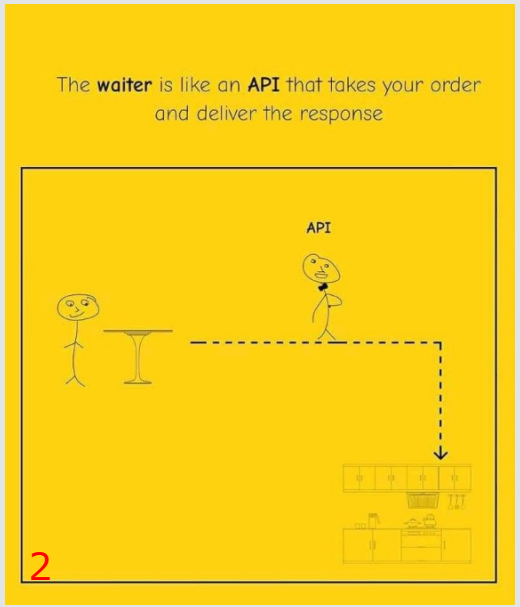
Schema Patterns are very useful when deciding how to model your data in MongoDB so please be sure to check it out.

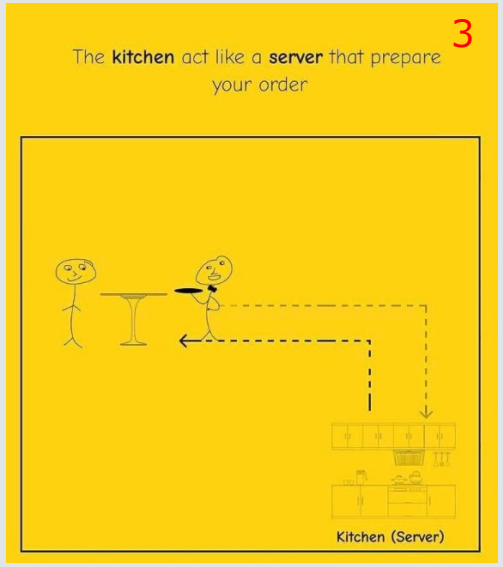
🏠📝 Data Modeling and Operations: My Pocket List

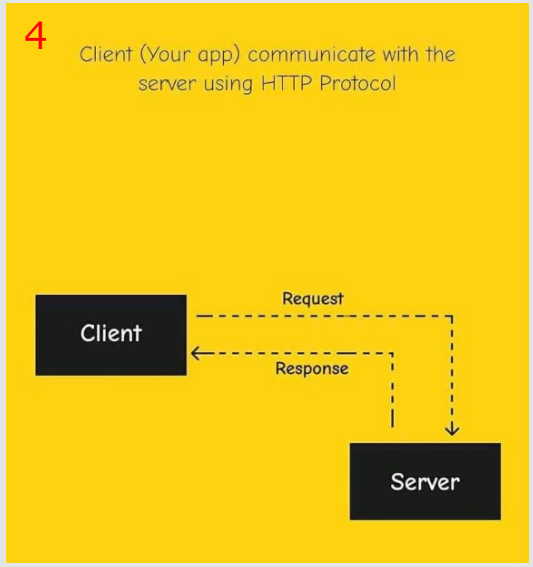
1. 建立 Collection: **users**
2. 新增 users information (至少兩個 user)
   - **account, password, name, followedDramas**
3. 更新 users 的追蹤片單 **followedDramas**
4. 查詢 user 的 片單資訊, 只列出 **follwedDramas**
   - tips: find methods and **specific projection**

From: coder.journey







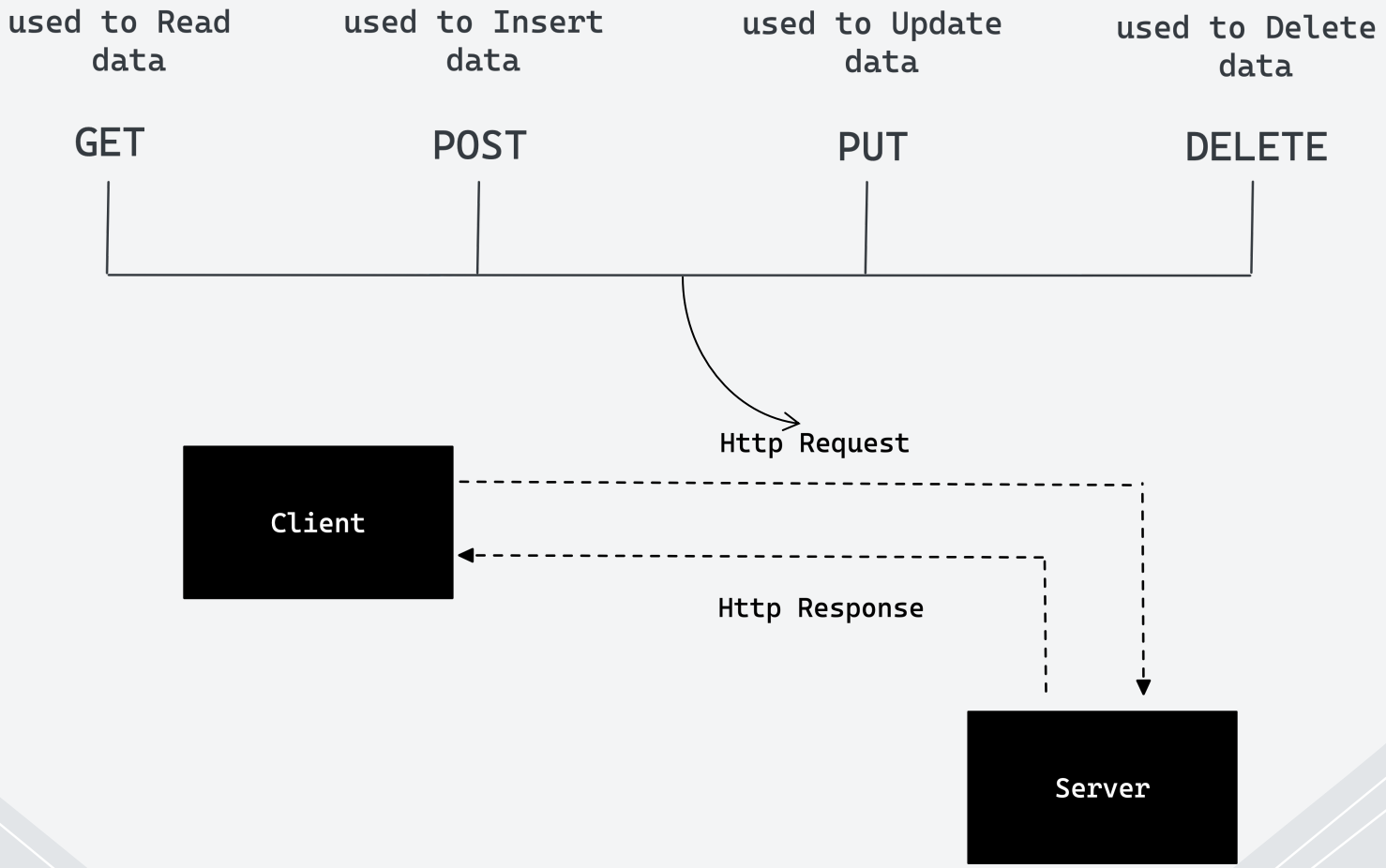


1. Think of yourself in a restaurant, where you are the client asking for a service. This scenario will help us understand how REST APIs function in a software environment.
2. In our restaurant analogy, the waiter acts as the API. Just as the waiter takes your order to the kitchen and brings back your food, an API receives requests from the client and delivers data from the server.
3. The kitchen represents the server. It's responsible for preparing your order based on the request sent by the waiter (API). The server processes these requests and sends back the appropriate response.
4. The client (your application) communicates with the server using the HTTP protocol, which is the fundamental technology that enables the exchange of data on the web.

From: coder.journey

HTTP Methods

The client 4 methods to communicate with the server



From: coder.journey

Example of Get Request

http://restaurant.com/api/meals

Called Resource

From: coder.journey

Best Practice for better naming

1. Don’t use verbs

GET	/~~getAllDramas~~	/dramas
POST	/~~createNewDramas~~	/dramas
DELETE	/~~deleteAllDramas~~	/dramas

2. Use plural nouns

/dramas instead of /drama

/users instead of /user

3. Use sub-resources for relations

```
GET  /dramas/1111              #Return the drama that id is 1111
GET  /users/viewingHistory     #Return user’s  viewing history
```

Best Practice for better naming

4. Filtering

```
GET   /dramas?category=anime   #Return a list of category anime of  dramas
```

5. Sorting

Allow ascending and descending sorting over multiple fields.

```
GET   /dramas?sort=-release_date
```

6. Version your API

```
https://example.com/v1/dramas
```

Insert dramas and Query drama with specific category

1. Insert data

   POST /api/v1/dramas

   db.dramas.insertMany([{ name: "Squid Game",

   category: "亞洲",…]);

2. Query Data : category=動漫

   GET /api/v1/dramas?category=anime

   db.dramas.find({ category: "動漫" });

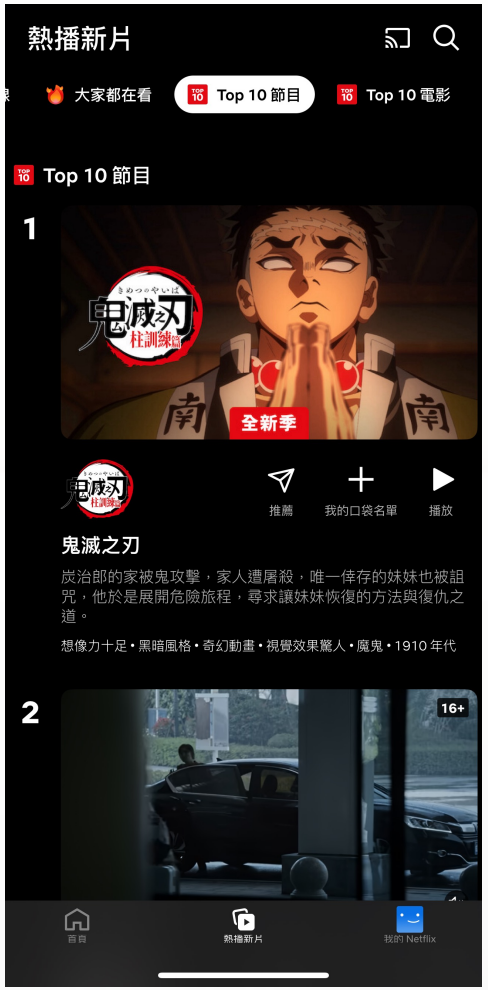
Upcoming movie list query

1.Query Data：即將上線

```
GET /api/v1/dramas
    ?release_date=2024-05-24&operator=gte
```

```
db.dramas.find({release_date: { $gte: "2024-05-24
00:00:00" }})
```

🏠📝 Multi-document update exercise: add top10 tags to episodes



1. Update Data : Top10

   PUT /api/v1/dramas/tags

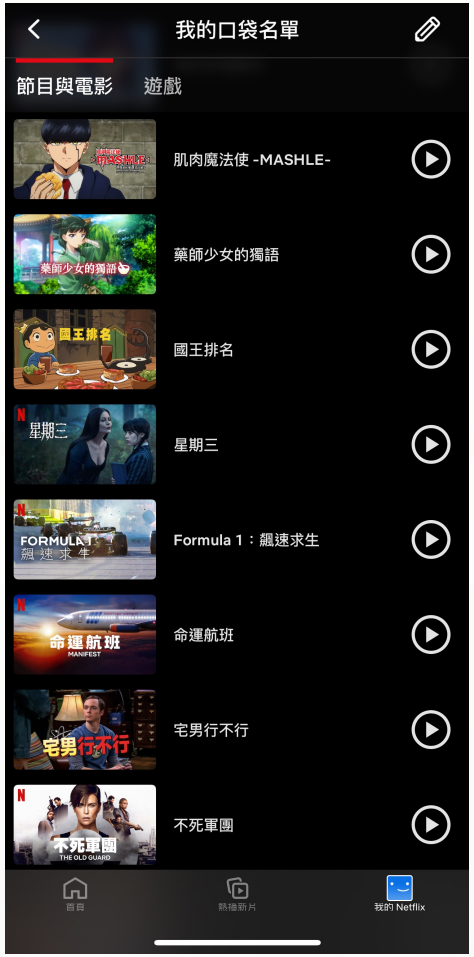
   ```
   db.dramas.updateMany(
   { name: { $in: ["淚之女王", "葬送的芙利蓮", "出租車司機", "肌肉魔法使"] } },
   { $set: { tag: ["top10"] } })
   ```

2. Query Top10

   GET /api/v1/dramas/tags/top10

   ```
   db.dramas.find({ tag: "top10" })
   ```

🏠📝 Data Modeling and Operations: My Pocket List



1. 建立 Collection: **users**
2. 新增 user information (至少兩個 user)

   - account, password, name, followedDramas

   POST /api/v1/users
3. 更新 users 的追蹤片單 **followedDramas**

   PUT /api/v1/users
4. 查詢 user 的 片單資訊, 只列出 **follwedDramas**

   - tips: find methods and **specific projection**

   GET /api/v1/users/followedDramas

Happy Learning