Module 3

NoSQL Databases

Ismael Navas-Delgado – University of Málaga (ismael@uma.es)

Table of contents

- Introduction
  - JSON
  - MongoDB Basics
  - MongoDB Atlas
  - MongoDB Compass
  - MongoDB Python
  - MongoDB simple queries

Table of contents

- Introduction
- **JSON**
  - MongoDB Basics
  - MongoDB Atlas
  - MongoDB Compass
  - MongoDB Python
  - MongoDB simple queries

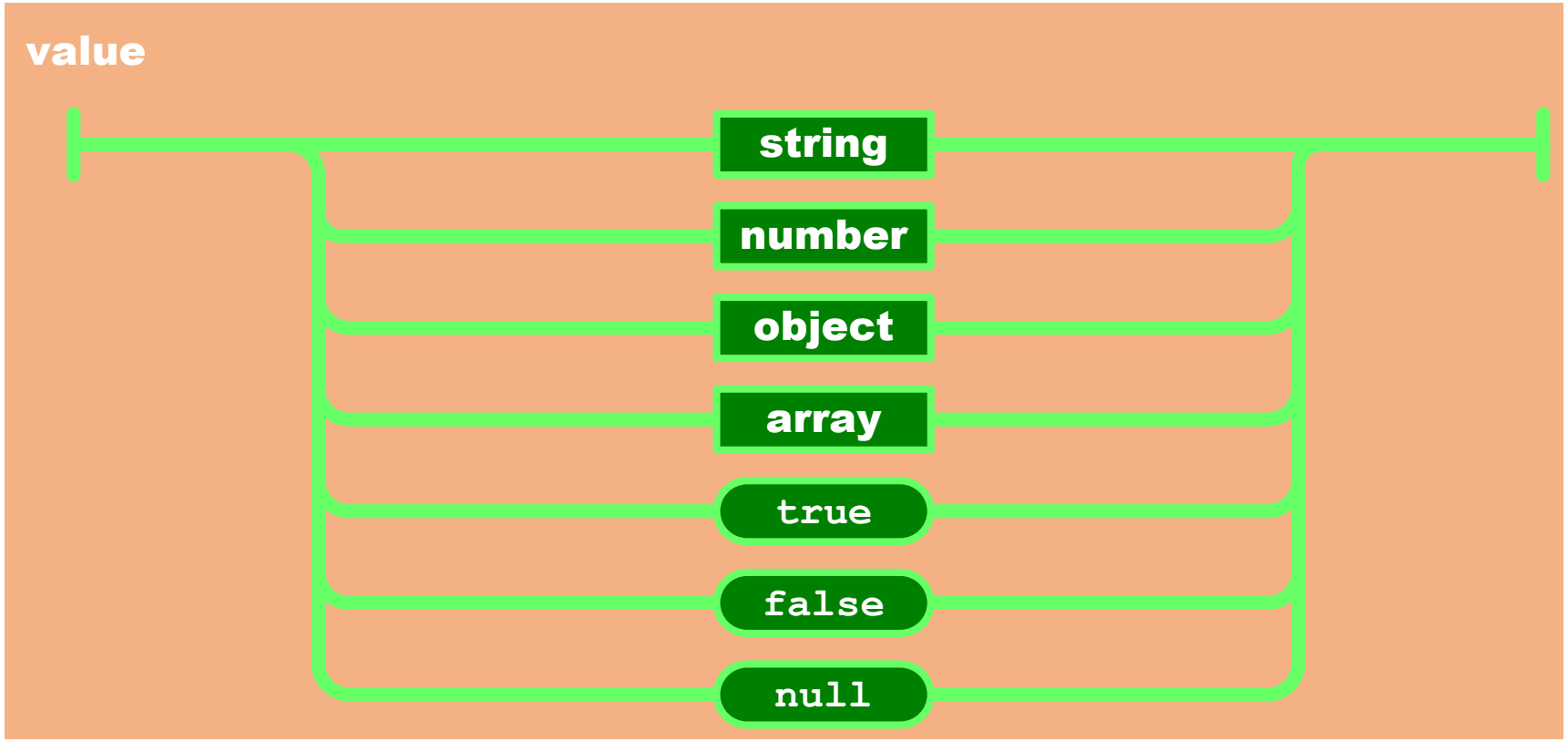
Values

- Strings
- Numbers
- Booleans

- Objects
- Arrays

- **`null`**

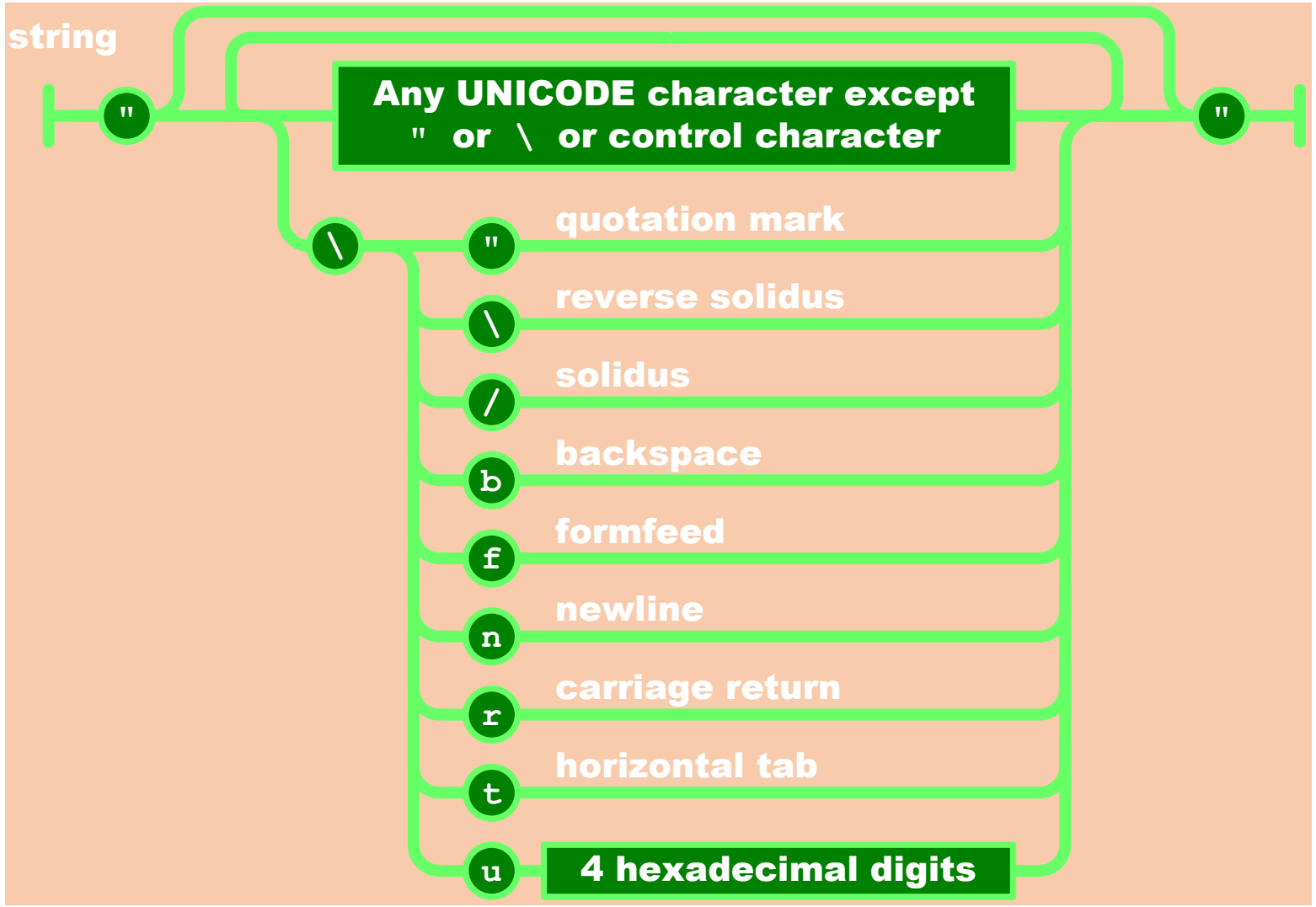
Value



Strings

- Sequence of 0 or more Unicode characters
- No separate character type
  - A character is represented as a string with a length of 1
- Wrapped in "double quotes"
- Backslash escapement

String

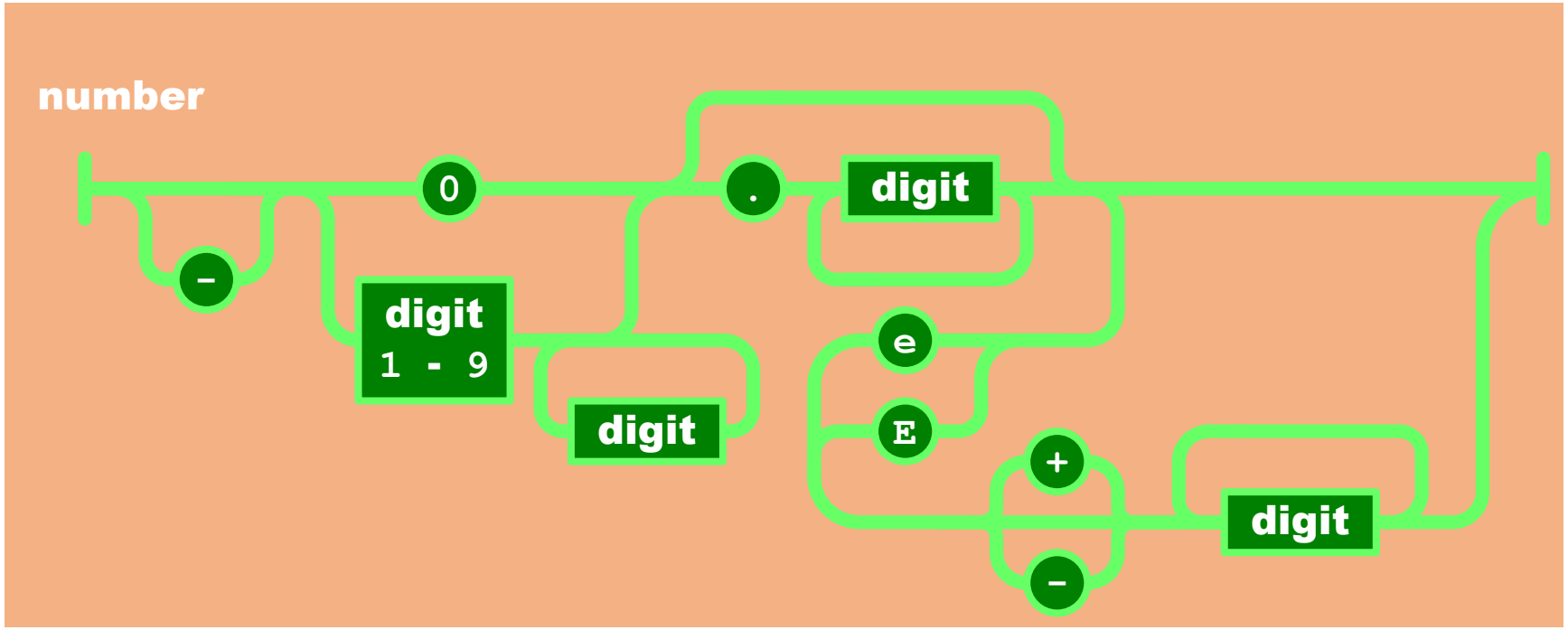

string
"
Any UNICODE character except
" or \ or control character
"
\
"
quotation mark
\
reverse solidus
/
solidus
b
backspace
f
formfeed
n
newline
r
carriage return
t
horizontal tab
u
4 hexadecimal digits

Numbers

- Integer
- Real
- Scientific

- No octal or hex
- No **`NaN`** or **`Infinity`**
  - Use **`null`** instead

Number



Booleans

- `true`
- `false`

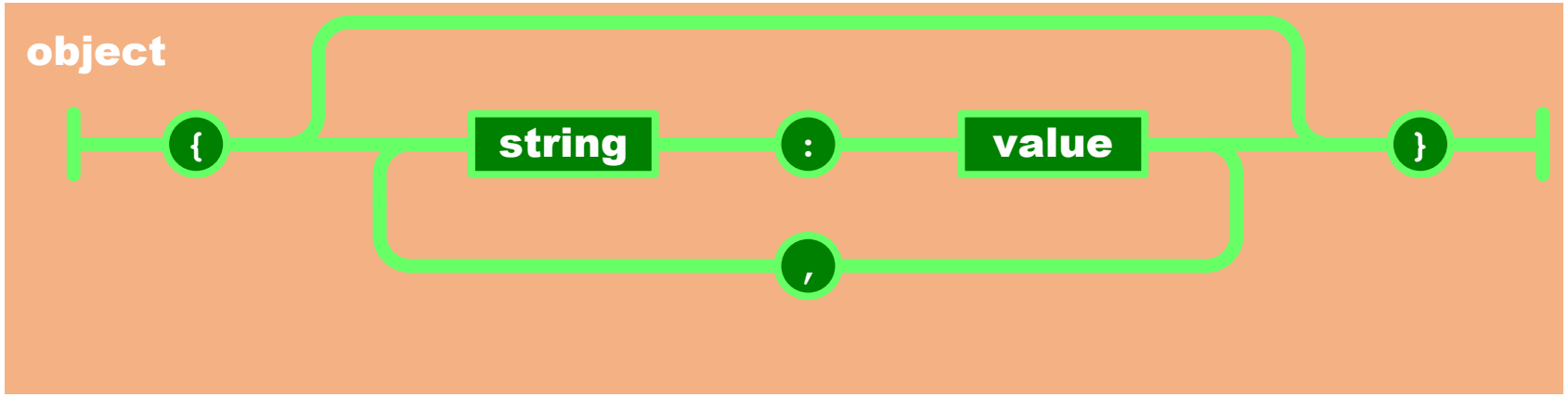
`null`

- A value that isn't anything

Object

- Objects are unordered containers of key/value pairs
- Objects are wrapped in **{ }**
- **,** separates key/value pairs
- **:** separates keys and values
- Keys are strings
- Values are JSON values
  - struct, record, hashtable, object

Object



Object


```
{"name":"Jack B. Nimble","at large":
true,"grade":"A","level":3,
"format":{"type":"rect","width":1920,
"height":1080,"interlace":false,
"framerate":24}}
```

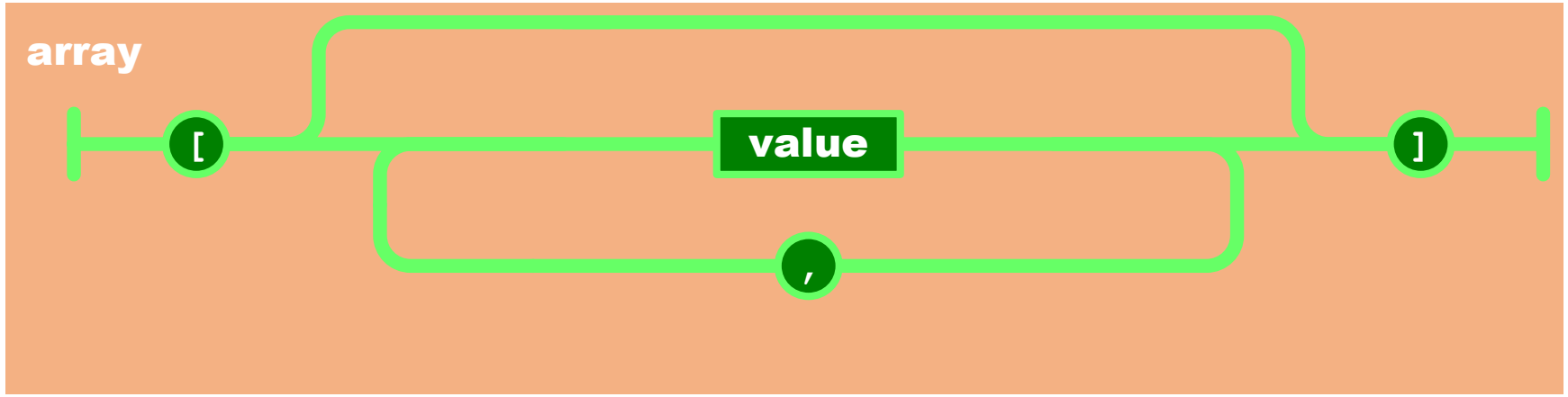
Object

```
{
    "name":      "Jack B. Nimble",
    "at large": true,
    "grade":     "A",
    "format": {
        "type":      "rect",
        "width":     1920,
        "height":    1080,
        "interlace": false,
        "framerate": 24
    }
}
```

Array

- Arrays are ordered sequences of values
- Arrays are wrapped in `[]`
- `,` separates values
- JSON does not talk about indexing.
  - An implementation can start array indexing at 0 or 1.

Array



Array

```
["Sunday", "Monday", "Tuesday", "Wednesday",
 "Thursday", "Friday", "Saturday"]
```

```
[
  • [0, -1, 0],
  • [1, 0, 0],
  • [0, 0, 1]
]
```

Arrays vs Objects

- Use objects when the key names are arbitrary strings.
- Use arrays when the key names are sequential integers.
- Don't get confused by the term Associative Array.

MongoDB Documents

JSON

```
{
        "FirstName": "Ismael"
        "LastName": "Navas Delgado"
}
```

XML

```
<person>
        <firstname>Ismael</firstname>
        <lastname>
                Navas Delgado
        </lastname>
</person>
```

MongoDB Documents

JSON

```
{
        "FirstName": "Ismael"
        "LastName": "Navas Delgado"
}
```

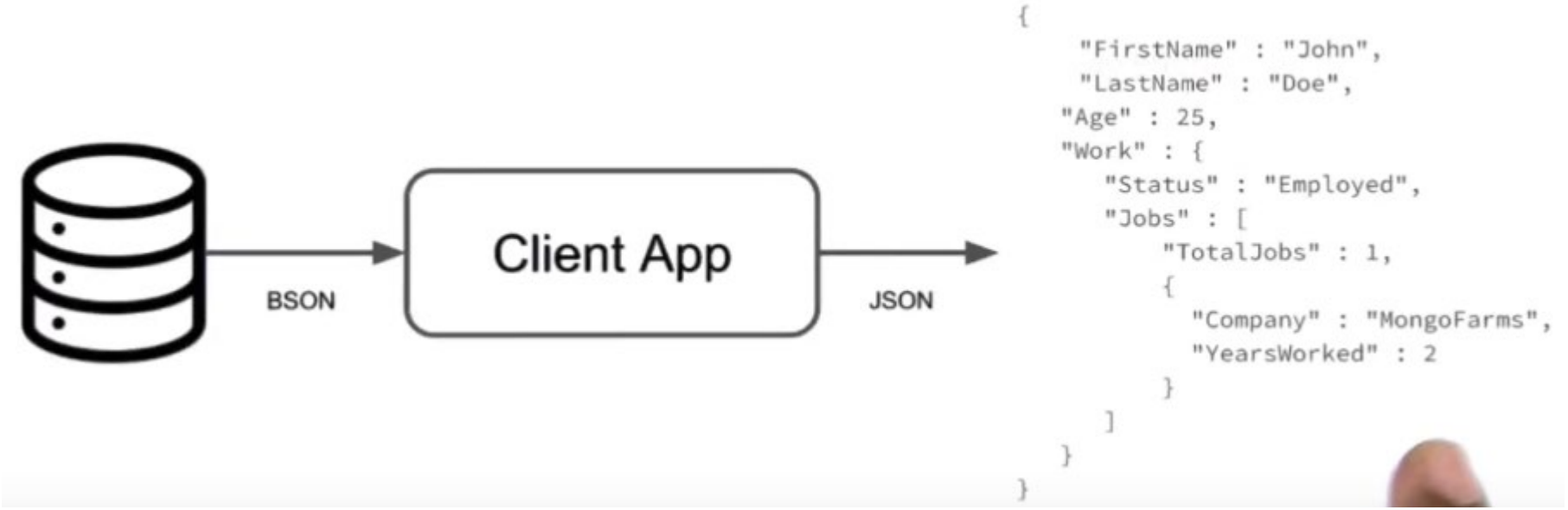
Relational

FirstName (String)	LastName (String)
Ismael	Navas Delgado

MongoDB JSON Limitations

- MongoDB Documents (~tuples) are restricted to 16MB size.
- If you try to insert or update a document with bigger size you'll get an error.

MogoDB BSON (Binary JSON)



MongoDB BSON

BSON Data types

- Double
- String
- Object (Document)
- Array
- Binary Data
- Undefined
- ObjectId
- Boolean
- Date
- Null
- Regular Expression
- DBPointer
- JavaScript
- Symbol
- JavaScript (with scope)_
- 32-bit integer
- Timestamp
- 64-bit integer
- Decimal128
- MinKey
- MaxKey

JSON Data types

- Object <document>
- String
- Number
- Array
- Boolean
- Null

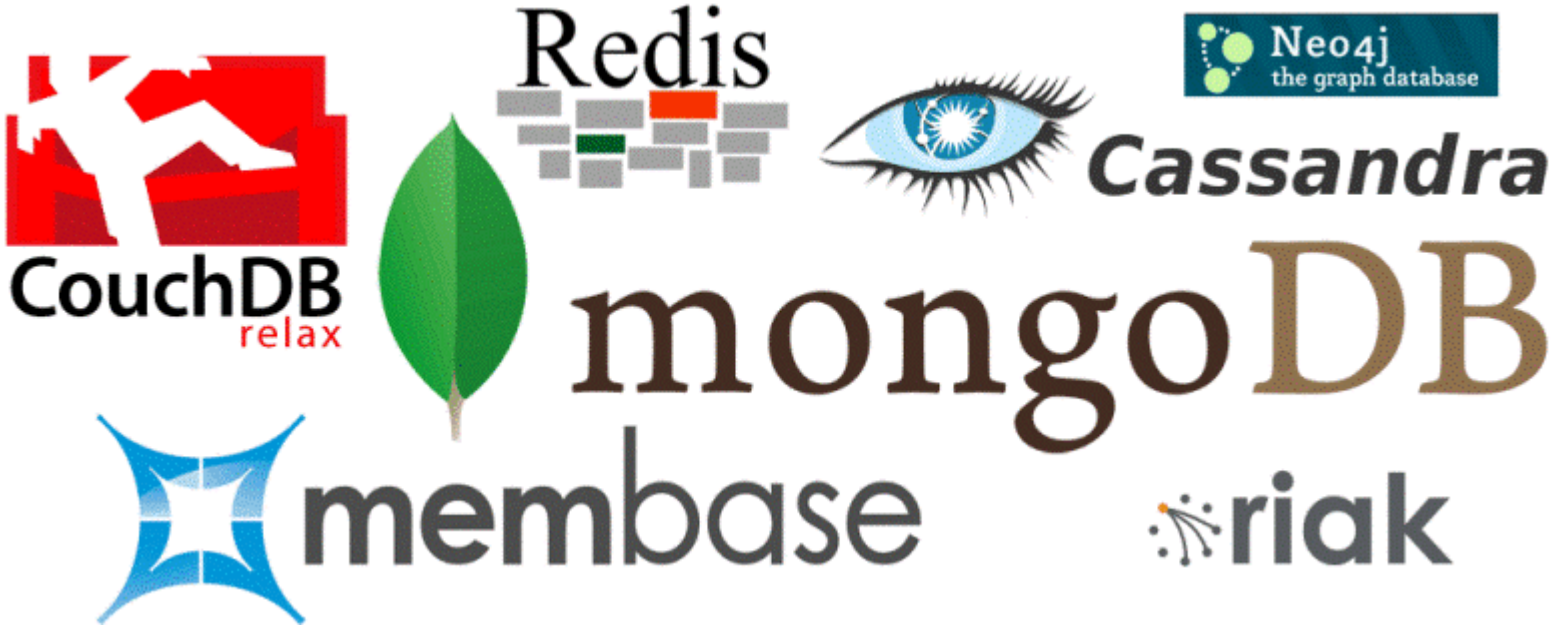
MongoDB BSON in Python → Python Dictionaries

```
new_account = {
    "account_holder": "Ismael Navas",
    "account_id": "ES2134757394785",
    "account_type": "checking",
    "balance": 123456,
    "last_updated": datetime.datetime.utcnow()
}
```

Table of contents

- Introduction
  - JSON
  - **MongoDB Basics**
  - MongoDB Atlas
  - MongoDB Compass
  - MongoDB Python
  - MongoDB simple queries

Introduction



Introduction

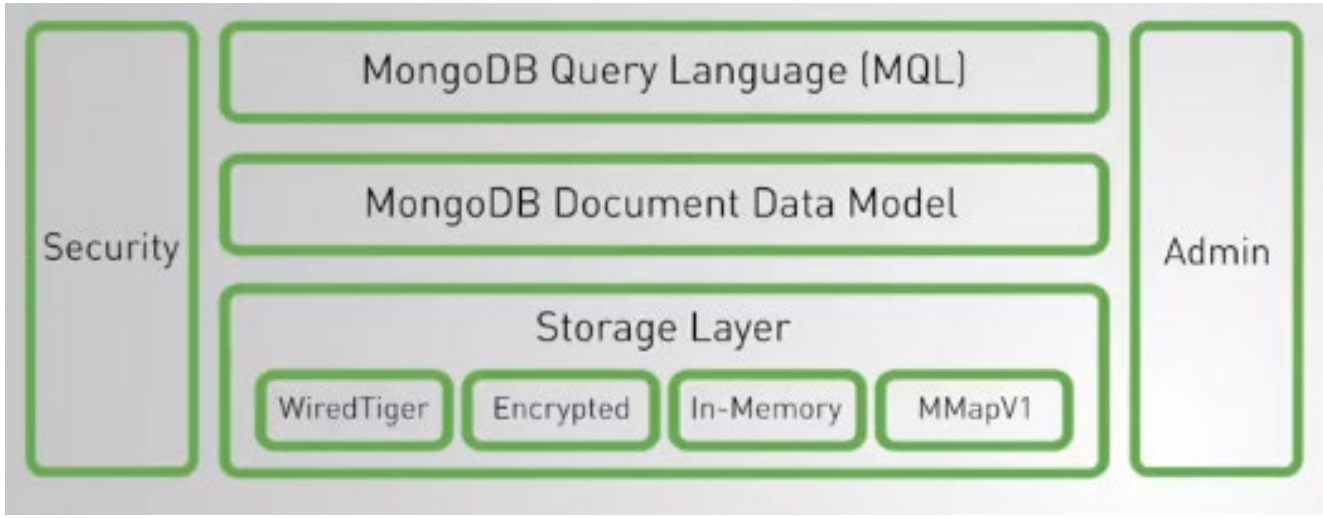
- Document Oriented Databases
  - Lotus Notes
- Data model: document collections (JSON, XML, BSON) with key-value pairs
  - Examples: CouchDB, **MongoDB**
  - Good for:
    - Natural data modelling
    - Programmer friendly
    - Agile development
    - Web oriented: CRUD

Introduction

- MongoDB (from “humongous”, that is, really big) is a document-oriented NoSQL DB
- MongoDB stores BSON (Binary JSON) documents with dynamic schema, making data integration an easy issue

Introduction

- Combines the best of key-value, document databases and relational databases
- Uses JSON and has its own query language
- Implemented in C++
- Used by SourceForge, Bit.ly, Foursquare and GitHub
- URL: http://www.mongodb.org/

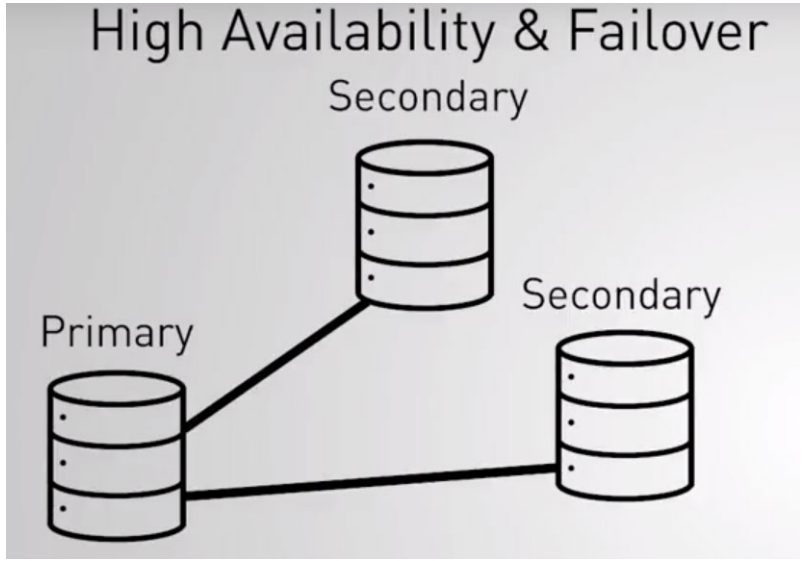


Introduction

- Ad hoc queries
  - MongoDB supports field searches, range queries and regular expressions
  - Queries can return specific fields, a document of a built results using JavaScript
- Indexing
  - Any field can be indexed in MongoDB, similar to the indexes in relational databases

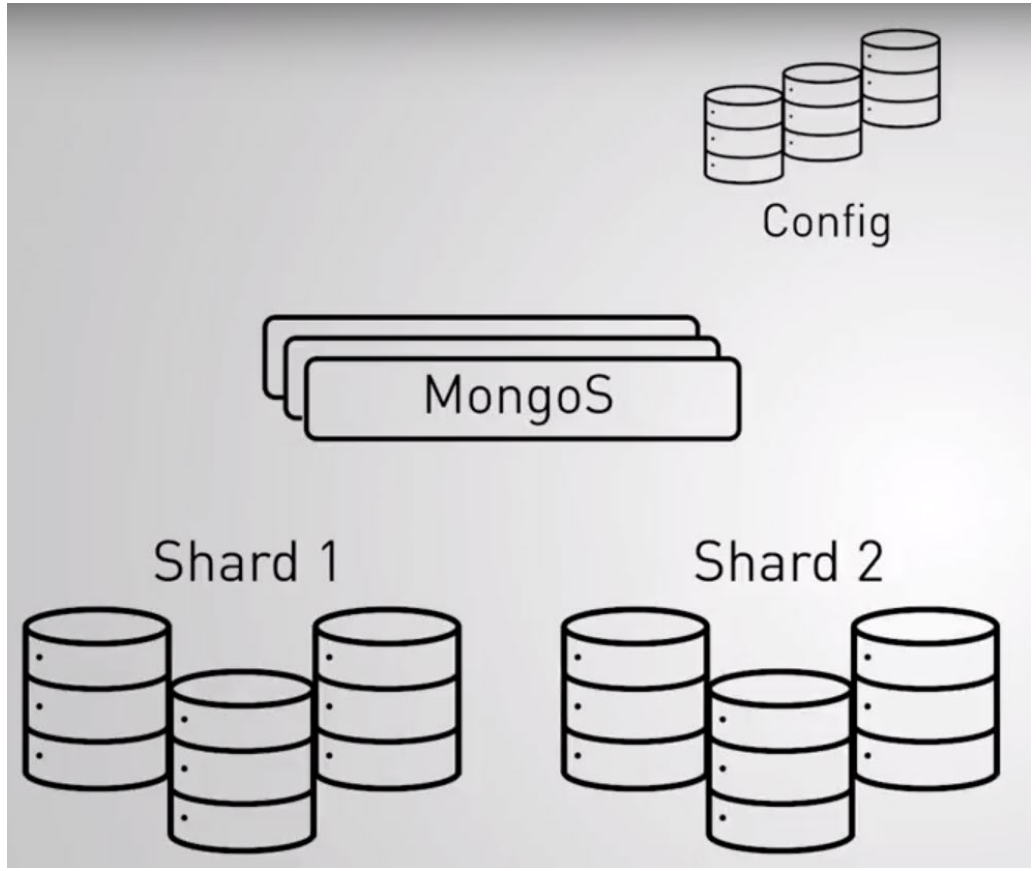
Introduction

- Replication
  - MongoDB support replication based on master-slave schemas
    - Master can run read and write operations (can be more than one)
    - Slave can copy master data and read them
      - If the master is not accessible, the slave can chose a different master dynamically



Introduction

- Load balancing
  - MongoDB can scale horizontally
    - The developer can chose a shard key that is used to distribute the data of a collection.
    - Data is distributed according to the shard key range of values
    - A shard is a master with one or more slaves
  - MongoDB can be executed in multiple servers, balancing the load and duplicating the data



Introduction

- MongoDB stores the document structure of JSON using dynamic BSON files. So there is no predefined schema
- Data elements are documents that are stored in collections
  - A collection can have several documents
    - Collections are like tables and documents like tuples
    - Documents in a collection can have different structure, but it is recommended that they have similar structures with some variations if needed

Introduction

- The document structure is as simple as a set of key-value pairs
  - Values can be numbers, strings, binary data (such as images) or any combination of key-value pairs

Introduction

- BSON is the format used by mongo for storage and data exchange. It is a binary representation
- BSON is designed for less storage needs and better performance
- The long fields include a size field for better reading of these data, so some BSON files are bigger in disk than the equivalent data in JSON
- BSON data types are JSON data types plus Date and Byte Arrays

Introduction

- A BSON object is an ordered list of elements. Each element has a name, type and value. Names are strings and types can be:
  - String
  - Integer 32 or 64 bits
  - Float 64 bits IEEE 754
  - Date (millisecond in Unix format)
  - Byte arrays
  - Boolean
  - Null
  - BSON object
  - BSON array
  - Regular expression
  - JavaScript code

Terminology

SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row	document or BSON document
column	field
index	index
table joins	embedded documents and linking
primary key	primary key
Specify any unique column or column combination as primary key.	In MongoDB, the primary key is automatically set to the _id field.
aggregation (e.g. group by)	aggregation pipeline

Example


```
{
    "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
    "Last Name": "PELLERIN",
    "First Name": "Franck",
    "Age": 29,
    "Address": {
        "Street": "1 chemin des Loges",
        "City": "VERSAILLES"
    }
}
```

Example

```
{
"_id" : 1,
"name" : { "first" : "John", "last" : "Backus" },
"contribs" : [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ],
"awards" : [
        {
          "award" : "W.W. McDowell Award",
          "year" : 1967,
          "by" : "IEEE Computer Society"
        },
        { "award" : "Draper Prize",
          "year" : 1993,
          "by" : "National Academy of Engineering"
        }
]
}
```

Table of contents

- Introduction
  - JSON
  - MongoDB Basics
  - **MongoDB Atlas**
  - MongoDB Compass
  - MongoDB Python
  - MongoDB simple queries

MongoDB Atlas

- Cluster
- IPs
- Users

Security

Authentication

Verifies the **Identity** of a user

Answers the question:
Who are you?

Authorization

Verifies the **privileges** of a user

Answers the question:
What do you have access to?

Security

Authentication Mechanisms

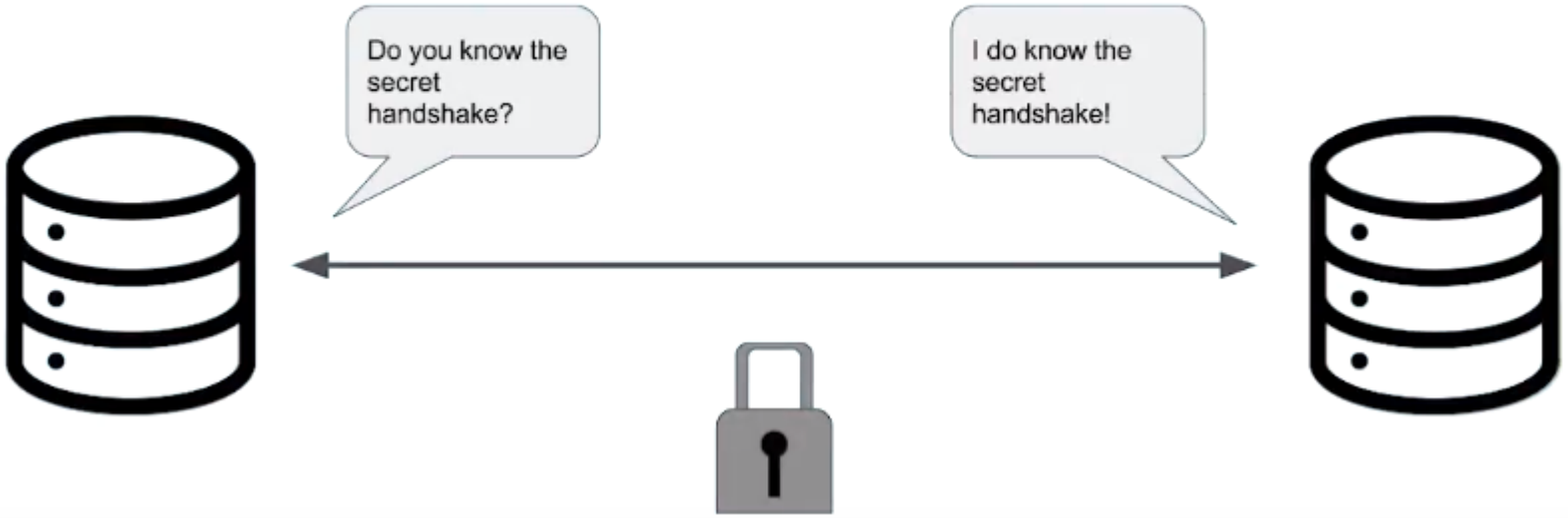
- SCRAM
- X.509

MongoDB Enterprise Only

- LDAP
- KERBEROS

Security

Cluster Authentication Mechanisms



Security

Authorization: Role Based Access Control

- Each user has one or more **Roles**
- Each **Role** has one or more **Privileges**
- A **Privilege** represents a group of **Actions** and the **Resources** those actions apply to

Authorization

- **Built-in roles**

- Database User
- Database Administration
- Cluster Administration
- Backup/Restore
- Super User

read	root
readWrite	
dbAdmin	
userAdmin	
dbOwner	
	clusterAdmin
backup	clusterManager
restore	clusterMonitor
	hostManager

Authorization

- **Built-in roles**

• Database User	readAnyDatabase readWriteAnyDatabase
• Database Administration	dbAdminAnyDatabase userAdminAnyDatabase
• Super User All Database	root

Authorization

- **Built-in roles**
  - **userAdmin**
  - dbOwner
  - dbAdmin

userAdmin

changeCustomData	grantRole
changePassword	revokeRole
createRole	setAuthenticationRestriction
createUser	viewRole
dropRole	viewUser
dropUser	

Authorization

- **Built-in roles**
  - userAdmin
  - dbOwner
  - **dbAdmin**

dbAdmin

collStats	...
dbHash	bypassDocumentValidation
dbStats	collMod
killCursors	collStats
listIndexes	compact
listCollections	convertToCapped

Authorization

- **Built-in roles**
  - userAdmin
  - **dbOwner**
  - dbAdmin

dbOwner

The database owner can perform any administrative action on the database.

This role combines the privileges granted by the **readWrite**, **dbAdmin** and **userAdmin** roles.

```
db.grantRolesToUser( "dba",  [ { db: "playground", role: "dbOwner"  } ] )
```

```
db.runCommand( { rolesInfo: { role: "dbOwner", db: "playground" },
showPrivileges: true} )
```

Table of contents

- Introduction
  - JSON
  - MongoDB Basics
  - MongoDB Atlas
  - **MongoDB Compass**
  - MongoDB Python
  - MongoDB simple queries

Table of contents

- Introduction
  - JSON
  - MongoDB Basics
  - MongoDB Atlas
  - MongoDB Compass
  - **MongoDB Python**
  - MongoDB simple queries

CRUD

- **(CREATE) Create documents**
  - **db.collection.insert()**
- (READ) Query documents
  - db.collection.find()
- (UPDATE) Update documents
  - db.collection.update()
- (DELETE) Delete documents
  - db.collection.remove()

PyMongo Basics

```
import datetime
from pymongo import MongoClient
client = MongoClient("…")
database = client.Master2022
account_collection = database.accounts
new_account = {
    "account_holder": "Ismael Navas",
    "account_id": "ES2134757394785",
    "account_type": "checking",
    "balance": 123456,
    "last_updated": datetime.datetime.utcnow()
}
result = account_collection.insert_one(new_account)
document_id = result.inserted_id
print(f"_id inserted: {document_id}")
client.close()
```

Table of contents

- Introduction
  - JSON
  - MongoDB Basics
  - MongoDB Atlas
  - MongoDB Compass
  - MongoDB Python
  - **MongoDB simple queries**

CRUD

- (CREATE) Create documents
  - db.collection.insert()
- **(READ) Query documents**
  - **db.collection.find()**
- (UPDATE) Update documents
  - db.collection.update()
- (DELETE) Delete documents
  - db.collection.remove()

Collections

- MongoDB stores documents in collections.
- Collections are analogous to tables in relational databases.
  - Unlike a table, however, a collection does not require its documents to have the same schema.
- In MongoDB, documents stored in a collection must have a unique_id field that acts as a primary key.

CRUD

- Restaurants collection:


```
{
 "address": {
   "building": "1007",
   "coord": [ -73.856077, 40.848447 ],
   "street": "Morris Park Ave",
   "zipcode": "10462"
 },
 "borough": "Bronx",
 "cuisine": "Bakery",
 "grades": [
   { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
   { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
   { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
   { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
   { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
 ],
 "name": "Morris Park Bake Shop",
 "restaurant_id": "30075445"
}
```

CRUD

- Retrieve the dataset from:
  - **https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/primer-dataset.json**
    - save to a file named primer-dataset.json
- Mongo Compass

mongoDB Compass

Create Database

Database Name

AGAPA

Collection Name

restaurantes

☐ **Time-Series**

Time-series collections efficiently store sequences of measurements over a period of time. Learn More

> **Additional preferences** (e.g. Custom collation, Capped, Clustered collections)

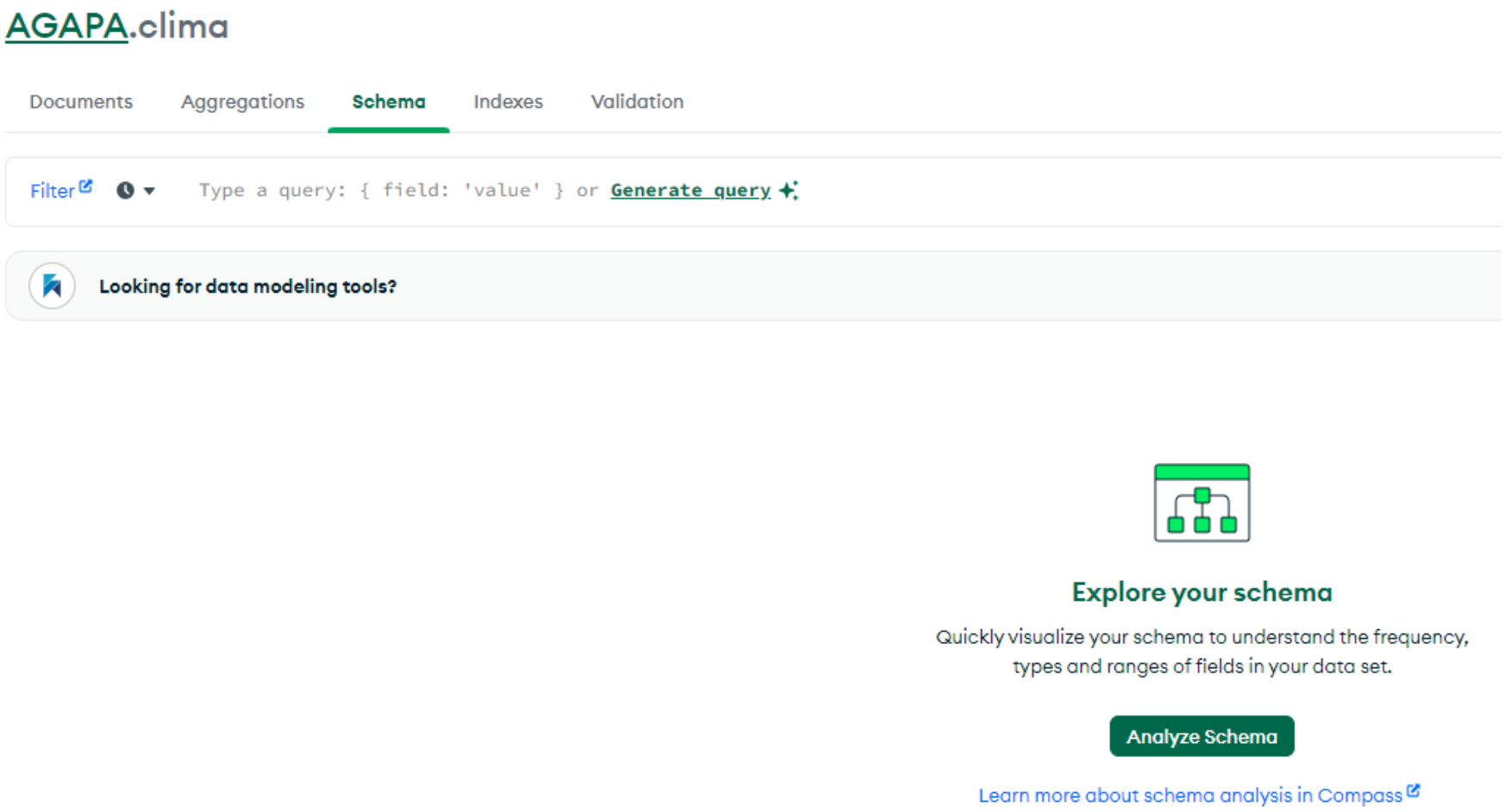
Cancel | Create Database

This collection has no data

It only takes a few seconds to import data from a JSON or CSV file.

Import Data

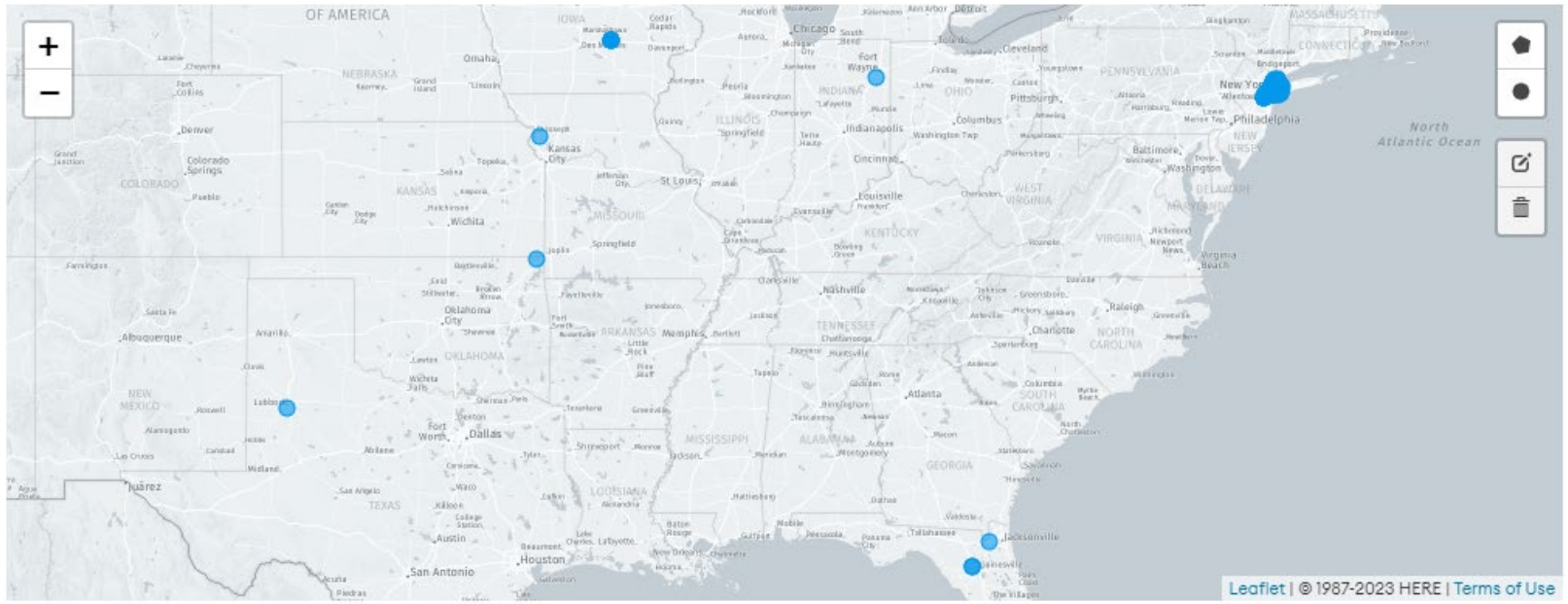
1 – 20 of 25359


AGAPA.clima
Documents
Aggregations
Schema
Indexes
Validation
Filter
Type a query: { field: 'value' } or Generate query
Looking for data modeling tools?
Explore your schema
Quickly visualize your schema to understand the frequency, types and ranges of fields in your data set.
Analyze Schema
Learn more about schema analysis in Compass

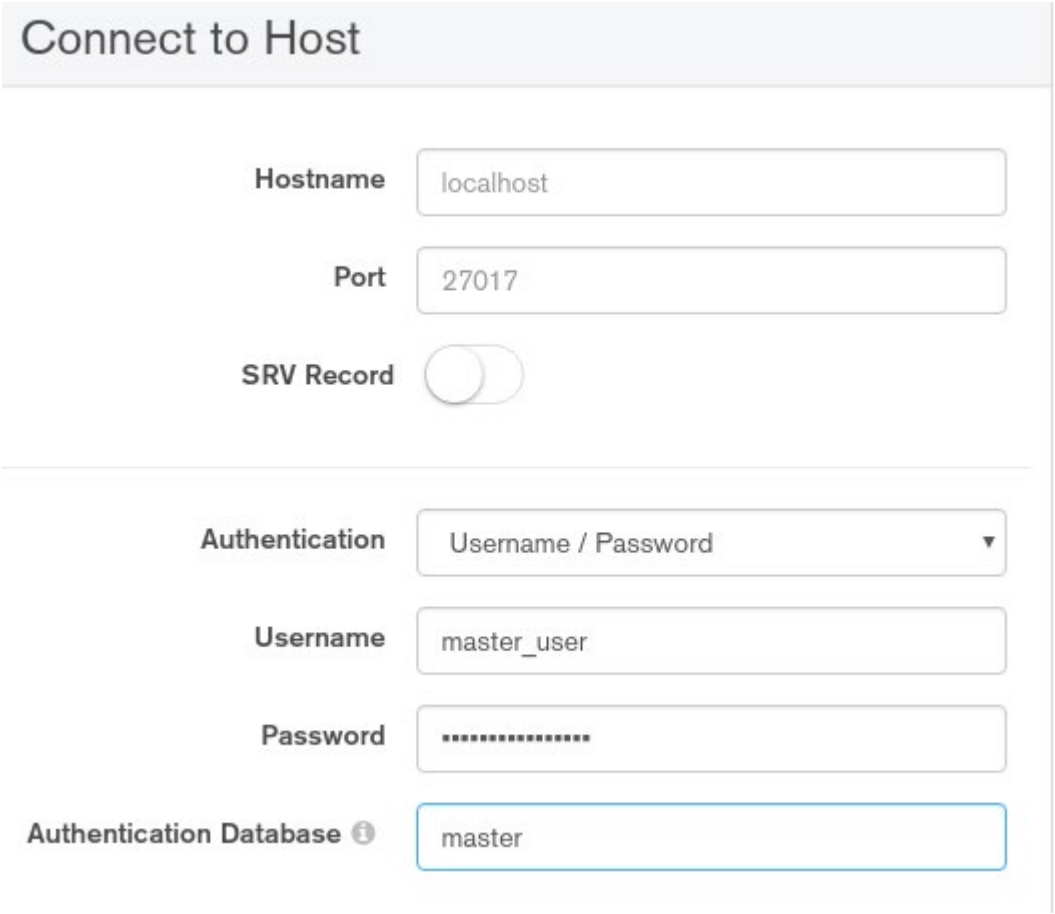
borough

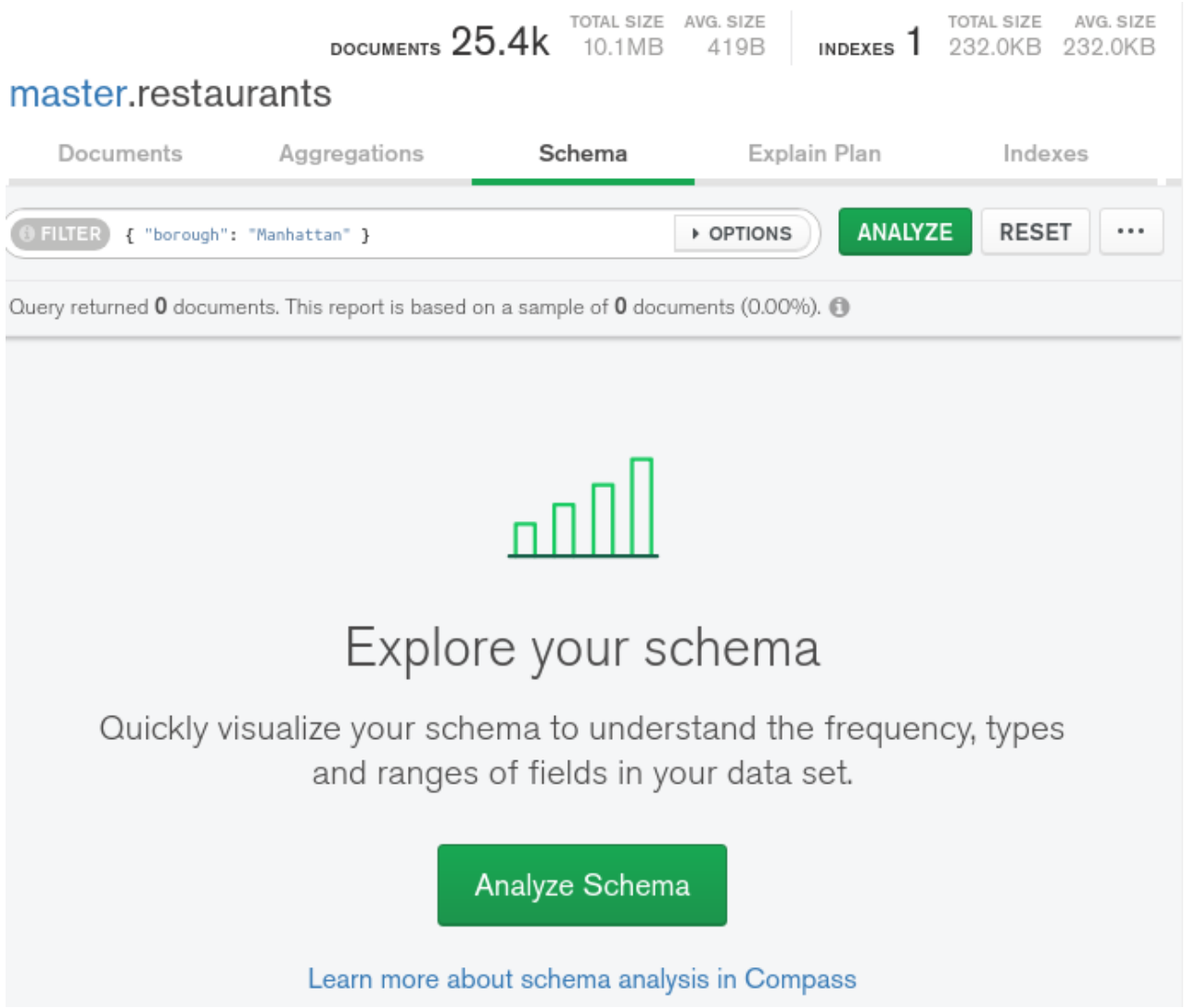
string

Manhattan | Queens | Brooklyn | Bronx | S



Mongo Compass

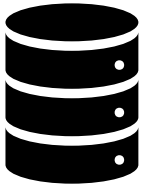




Mongo Atlas (Servicio en la nube)

Mongo Atlas Web

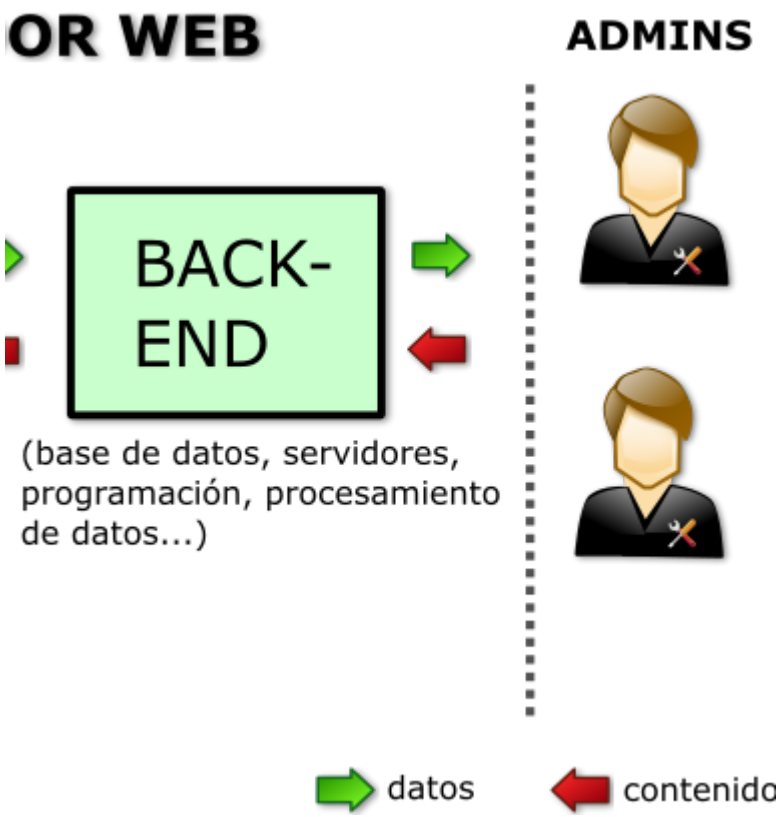
Admin & Test



Design & Test

Mongo Atlas Cluster

Mongo Compass



CRUD

- You can use the find() method to issue a query to retrieve data from a collection in MongoDB. All queries in MongoDB have the scope of a single collection.
- Queries can return all documents in a collection or only the documents that match a specified filter or criteria. You can specify the filter or criteria in a document and pass as a parameter to the find() method.
- The find() method returns query results in a cursor, which is an iterable object that yields documents.

CRUD

- Query for All Documents in a Collection
- To return all documents in a collection, call the find() method without a criteria document. For example, the following operation queries for all documents in the restaurants collection.
  - db.restaurants.find()
- The result set contains all documents in the restaurants collection.

CRUD

- Specify Equality Conditions
  - The query condition for an equality match on a field has the following form:
    - { <field1>: <value1>, <field2>: <value2>, ... }
  - If the <field> is a top-level field and not a field in an embedded document or an array, you can either enclose the field name in quotes or omit the quotes.
  - If the <field> is in an embedded document or an array, use dot notation to access the field. With dot notation, you must enclose the dotted name in quotes.

CRUD

- Query by a Top Level Field
  - The following operation finds documents whose borough field equals "Manhattan".
    - db.restaurants.find({ "borough": "Manhattan" })
  - The result set includes only the matching documents.
  - Uso de variables:

```
var c = db.restaurants.find( { "borough": "Manhattan" } )
c.hasNext()
c.next()
```

CRUD

- Query by a Top Level Field
  - The following operation finds documents whose borough field equals "Manhattan".
    - db.restaurants.find({ "borough": "Manhattan" })
  - The result set includes only the matching documents.
  - Uso de variables:

```
var c = db.restaurants.find( { "borough": "Manhattan" } )
c.hasNext()
c.next()
```

python™ mongoDB Compass

CRUD

- Query by a Field in an Embedded Document
  - To specify a condition on a field within an embedded document, use the dot notation. Dot notation requires quotes around the whole dotted field name. The following operation specifies an equality condition on the zipcode field in the address embedded document.
    - db.restaurants.find({ "address.zipcode": "10075" })
  - The result set includes only the matching documents.
- Limit the number of results with limit(N)
  - db.restaurants.find({ "address.zipcode": "10075" }).limit(1)

CRUD

- Query by a Field in an Array
  - The grades array contains embedded documents as its elements. To specify a condition on a field in these documents, use the dot notation. Dot notation requires quotes around the whole dotted field name. The following queries for documents whose grades array contains an embedded document with a field grade equal to "B".
    - db.restaurants.find({ "grades.grade": "B" })
  - The result set includes only the matching documents.
- Use .pretty for pretty print out:
  - db.restaurants.find({ "grades.grade": "B" }).pretty()

CRUD

- Query operators:
  - https://docs.mongodb.org/manual/reference/operator/query/
- OR

```
db.restaurants.find(
    { $or:
        [
        { "address.zipcode": "10075" },
        { "address.zipcode": " 10019 " }
        ]
    }
)
```

  - db.restaurants.find({ $or: [{ "address.zipcode": "10075" }, { "address.zipcode": " 10019 " }] })

CRUD

- Query operators:
  - https://docs.mongodb.org/manual/reference/operator/query/
- GT, LT, …

```
db.restaurants.find(
    { $and:
        [
        { "address.zipcode": { $gt: "10019" } },
        { "address.zipcode": { $lt: "10022" } }
        ]
    }
)
```

  - db.restaurants.find({ $and: [{ "address.zipcode": { $gt: "10019" } }, { "address.zipcode": { $lt: "10022" } }] })

CRUD

- Sort Query Results
  - To specify an order for the result set, append the sort() method to the query. Pass to sort() method a document which contains the field(s) to sort by and the corresponding sort type, e.g. 1 for ascending and -1 for descending.
  - For example, the following operation returns all documents in the restaurants collection, sorted first by the borough field in ascending order, and then, within each borough, by the "address.zipcode" field in ascending order:
    - db.restaurants.find().sort({ "borough": 1, "address.zipcode": 1 })

AI in Mongo Compass

Filter Type a query: { field: 'value' }

{ "borough": "Manhattan" }

1 – 20 of 10259

Filter {"borough": "Manhattan"}

restaurantes que estén en Manhattan

Filter Type a query: { field: 'value' }

{ "borough": "Manhattan" }

1 - 20 of 10259

Filter {"borough": "Manhattan"}

restaurantes que estén en Manhattan

Filter {"borough": "Manhattan", "cuisine": "Italiana"}

restaurantes en Manhattan de comida Italiana

Filter {"address.zipcode": "10075"}

restaurantes que estén en el código postal 10075

Filter {"address.zipcode": "10075"}

restaurantes que estén en el código postal 10075

```
_id: "30191841"
address: Object
  building: "351"
  coord: Array (2)
  street: "West   57 Street"
  zipcode: "10019"
borough: "Manhattan"
cuisine: "Irish"
grades: Array (4)
name: "Dj Reynolds Pub And Restaurant"
```

Filter {"grades.grade": "B"}

restaurantes que tangan una valoración de B

Filter {"grades.grade": "B"}

restaurantes que tangan una valoración de B

```
_id: "30191841"
address: Object
  building: "351"
  coord: Array (2)
  street: "West   57 Street"
  zipcode: "10019"
borough: "Manhattan"
cuisine: "Irish"
grades: Array (4)
  0: Object
    date: 2014-09-06T00:00:00.000+00:00
    grade: "A"
    score: 2
  1: Object
  2: Object
  3: Object
name: "Dj Reynolds Pub And Restaurant"
```

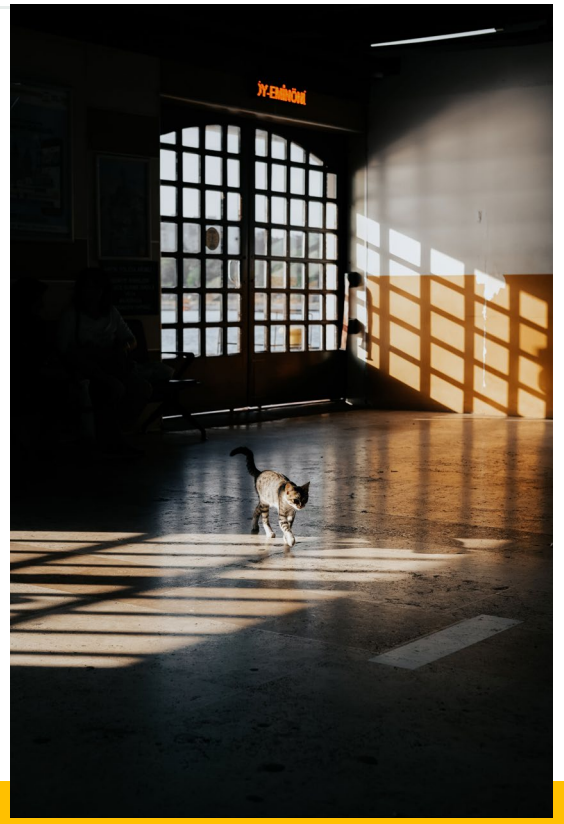
Filter {"grades.grade": "B"}

restaurantes que tangan una valoración de B

```
_id: "30191841"
address: Object
  building: "351"
  coord: Array (2)
  street: "West   57 Street"
  zipcode: "10019"
borough: "Manhattan"
cuisine: "Irish"
grades: Array (4)
  0: Object
    date: 2014-09-06T00:00:00.000+00:00
    grade: "A"
    score: 2
  1: Object
  2: Object
  3: Object
name: "Dj Reynolds Pub And Restaurant"
```

Filter { "grades.grade": { "$eq": "B" } }

restaurantes que todas sus valoraciones sean B




```
_id: "30075445"
address: Object
  building: "1007"
  coord: Array (2)
  street: "Morris Park Ave"
  zipcode: "10462"
borough: "Bronx"
cuisine: "Bakery"
grades: Array (5)
  0: Object
    date: 2014-03-03T00:00:00.000+00:00
    grade: "A"
    score: 2
  1: Object
    date: 2013-09-11T00:00:00.000+00:00
    grade: "A"
    score: 6
  2: Object
    date: 2013-01-24T00:00:00.000+00:00
    grade: "A"
    score: 10
  3: Object
    date: 2011-11-23T00:00:00.000+00:00
    grade: "A"
    score: 9
  4: Object
    date: 2011-03-10T00:00:00.000+00:00
    grade: "B"
    score: 14
name: "Morris Park Bake Shop"
```

Filter { "grades.grade": { "$eq": "B" } }

restaurantes que todas sus valoraciones sean B

```
{"grades": {
        $not: {
                $elemMatch: {"grade": { $ne: "B" }}
}
        }
}
```


```
_id: "30075445"
address: Object
  building: "1007"
  coord: Array (2)
  street: "Morris Park Ave"
  zipcode: "10462"
borough: "Bronx"
cuisine: "Bakery"
grades: Array (5)
  0: Object
    date: 2014-03-03T00:00:00.000+00:00
    grade: "A"
    score: 2
  1: Object
    date: 2013-09-11T00:00:00.000+00:00
    grade: "A"
    score: 6
  2: Object
    date: 2013-01-24T00:00:00.000+00:00
    grade: "A"
    score: 10
  3: Object
    date: 2011-11-23T00:00:00.000+00:00
    grade: "A"
    score: 9
  4: Object
    date: 2011-03-10T00:00:00.000+00:00
    grade: "B"
    score: 14
name: "Morris Park Bake Shop"
```

Filter { "grades.grade": { "$eq": "B" } }

restaurantes que todas sus valoraciones sean B

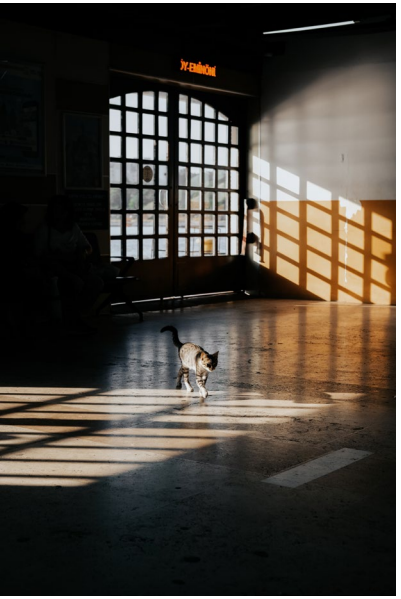
```
{"grades": {
        $not: {
                $elemMatch: {"grade": { $ne: "B" }}
}
            }
}
```

```
_id: "40386287"
address: Object
borough: "Manhattan"
cuisine: "Chinese"
grades: Array (4)
  0: Object
     date: 2014-06-17T00:00:00.000+00:00
     grade: "B"
     score: 16
  1: Object
     date: 2013-10-16T00:00:00.000+00:00
     grade: "B"
     score: 21
  2: Object
     date: 2013-04-09T00:00:00.000+00:00
     grade: "B"
     score: 17
  3: Object
     date: 2012-07-13T00:00:00.000+00:00
     grade: "B"
     score: 23
name: "Winnie'S Bar"
```

Filter

```
{ "address.zipcode": { "$in": ["10075", "10019"] }, "cuisine": "restaurantes" }
```

restaurantes en el código postal 10075 o 10019



Filter

```
{"address.zipcode": {"$in": ["10075", "10019"]}}
```

restaurantes en el código postal 10075 o 10019

Filter

```
{
  $or: [
    { "address.zipcode": "10075" },
    { "address.zipcode": "10019" }
  ]
}
```

restaurantes en el código postal 10075 o 10019 usando $or

Filter

```
{
  "address.zipcode": {
    "$gt": "10019",
    "$lt": "10022"
  },
  "cuisine": "restaurantes"
}
```

201

restaurantes en el código postal mayor de 10019 y menor de 10022

Filter

```
{
  $and: [{"address.zipcode": {
        $gt: "10019"}},
          {"address.zipcode": {
        $lt: "10022"
      }
    }
  ]
}
```

201

restaurantes en el código postal mayor de 10019 y menor de 10022 usando $and

6430

Filter

```
{ "address.zipcode": { "$gt": "10019" }, "address.zipcode": { "$lt": "10022" } }
```

201

201

6430