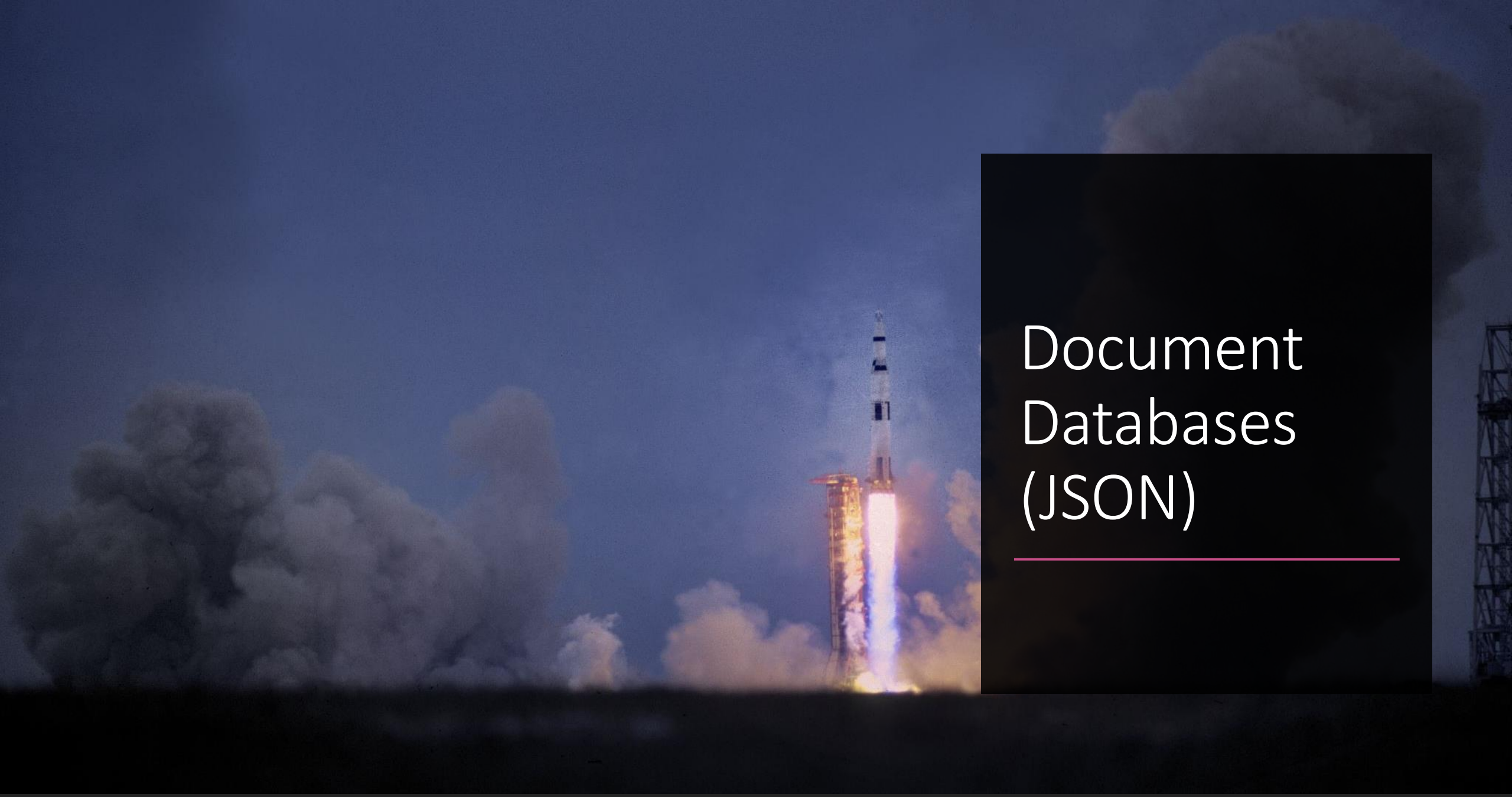


Document Databases (JSON)

Document database

Definition

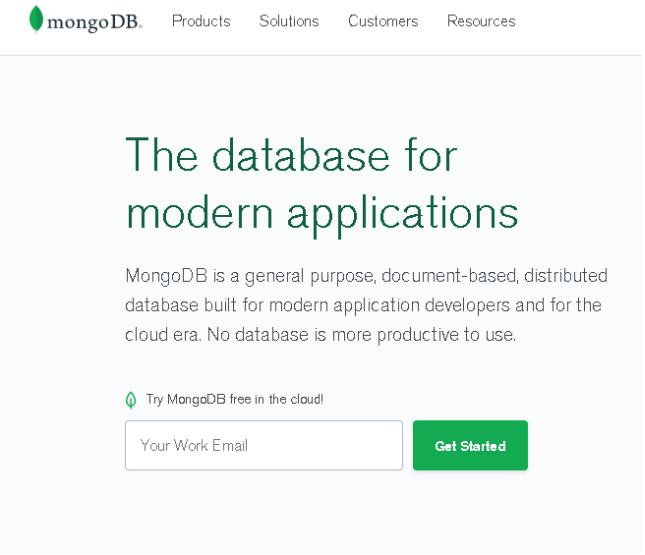
- A place to store documents.
- Documents are self-describing structures and usually similar to each other, but they don’t have to be the same.

Pros/Cons

- Documents can vary from each other and still belong to the same collection

Example

- MongoDB, Couchbase



JSON Documents

JavaScript Object Notation

Lightweight data-interchange format.

Easily readable by humans and machines.

System-independent.

Based on a subset of JavaScript.

Language independent but based on C-family of language conventions.

JSON Structure

Objects:

- Base building block.
- Unordered set of name/value pairs.
  - Records, structs, dicts, hash tables, associative arrays
- Delimited by { and }
- Each name is followed by :
- Name/value pairs separated by ,
- Values can be a string, number, boolean, object or array.

Strings:

- Delimited by “ “
- Sequence of zero or more characters.

Number

- Integer, fraction, exponent

Boolean

- True, false or null

Arrays:

- Ordered collection of values
  - Vectors, lists, sequences
- Delimited by [and]
- Values separated by ,

Sample Student JSON


```
{
  "id": 1,
  "name": "John Doe",
  "age": 20,
  "gender": "male",
  "major": "Computer Science",
  "GPA": 3.5,
  "courses": [
    {
      "courseId": 101,
      "courseName": "Introduction to Computer Science",
      "instructor": "Prof. Smith",
      "credits": 3,
      "grade": "A"
    },
    {
      "courseId": 201,
      "courseName": "Data Structures and Algorithms",
      "instructor": "Prof. Johnson",
      "credits": 4,
      "grade": "B+"
    }
  ]
}
```

MongoDB (JSON Document database)

Features

- Open source
- Cross-platform
- Leading NoSQL db
- Stores data in JSON-like documents
- Document model maps to the objects in your application code
- Queries, indexing, and real time aggregation

MongoDB (Document database)

Pros

- Rich query support
- Indexing
- Replication
- Load balancing
- File storage
- Aggregation

Cons

- Weak transactional support
- Lack of referential integrity
- 16 MB max document size
- Max 64 indexes per collection
- Max 12 nodes per replica set
- Max 512 bytes for shard keys

Core characteristics

Data structure

- Basic element: BSON (Binary JSON) document
- Collection: Multiple documents
- Database: Multiple collections

Data types

- Null
- Boolean
- Number
- String
- Date
- Array

Functionality

- Aggregation
  - Group values from multiple documents
- Transactions
  - Atomic operations in single documents
- Indexing
  - Improve efficiency in queries
- Sharding
  - Distribute data across multiple machines

BSON vs JSON

Binary encoded JavaScript Object Notation.

Not (directly) readable by humans, easier to parse for machines.

- Encodes type and length information.

Supports more data types than JSON.

- Integer: byte, int32, int64, uint64
- Float: double, decimal128
- Time/Date: timestamp, date
- Misc: objectid, regex, javascript,

MongoDB organization

- DB is a container of collections

```
> use sample_db;
switched to db sample_db
```

- Collection is a container of documents
  - Equivalent to a table in SQL
  - Documents in the same collection should have a similar or related purpose
  - Schemaless

```
> db.createCollection("users_profile");
{ "ok" : 1 }
```

- Documents are a collection of key-value pairs
  - Special attribute _id
    - Automatically generated by MongoDB if not provided

MongoDB/SQL

SQL	MongoDB
Database	Database
Table	Collection
Row	Document
Column	Field

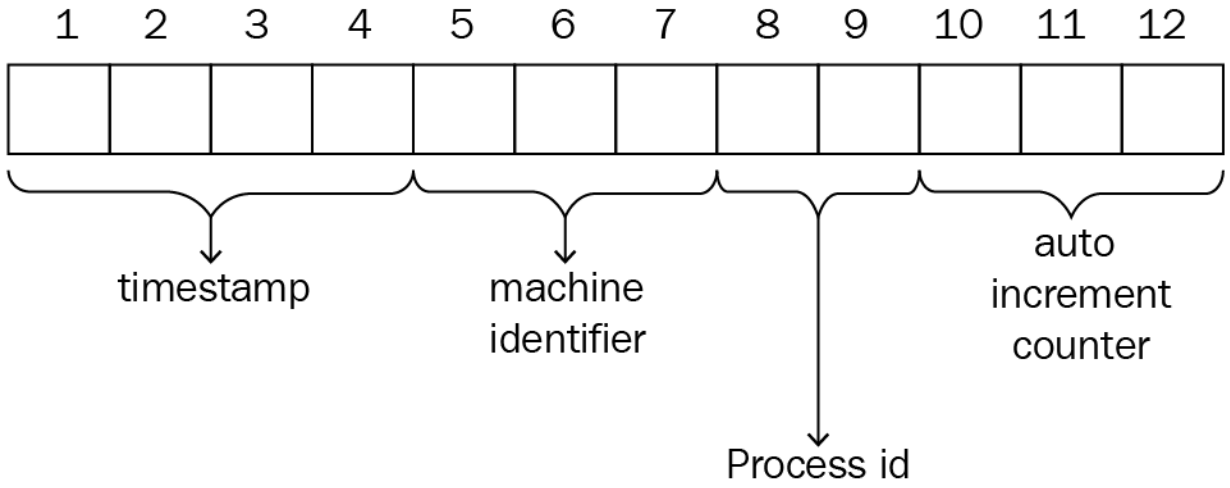
Basic functionality

Basic commands

- show dbs
- show collections
- use DATABASE
- db.createCollection(“collection”)
- db.COLLECTION.insertOne(DATA)
- db.COLLECTION.find(QUERY)
- db.COLLECTION.aggregate(STAGES)

Identifiers

- Consist of 12 bytes:
  - 4-byte value representing the seconds since the Unix epoch
  - 5-byte random value
  - 3-byte counter, starting with a random value.



Mongo Query Language

Create operation

Inserts new document into collection

- If the collection does not exist, create it

Atomic at document level

db.collection.insertOne(DATA)

db.collection.insertMany([DATA_ARRAY])

```
db.users_profile.insertOne({
          userId : 1,
          firstName : "John",
          lastName : "Richard",
          age : 26,
          email : "john2992@mail.com"
 })
```

```
/* 1 */
{
    "acknowledged" : true,
    "insertedId" : ObjectId("596ce72cbc5266c2c9c44e8d")
}
```

Collection methods

Methods

- find(query, projection)
- countDocuments(query, options)
- distinct(field, query, options)

- Query: Specifies selection filter using query operators.
- Projection: Specifies the fields to return in the documents that match the query filter

Queries

General syntax

- db.COLLECTION.find(QUERY, PROJECTION)
  - QUERY is a document where you define your filters
  - PROJECTION is a document where you define the fields to extract from the documents that fulfill the query

Selecting all the documents in a collection

- db.COLLECTION.find() #SELECT * FROM COLLECTION;

Basic querying

- db.COLLECTION.find({<field>:<value>}) #SELECT * FROM COLLECTION WHERE <field>=<value>;
- db.COLLECTION.find({<field>: {<operator>:<value>}}) #SELECT * FROM COLLECTION WHERE <field> <operator> <value>;

Basic projection

- db.COLLECTION.find({}, {<field>:<value>, <field2>:<value>}) #SELECT field, field2 FROM COLLECTION;
- Value = 1 to include field, 0 to hide (cannot be mixed except for _id)

Query operators

Selectors

- Comparison
  - $eq, $gt, $gte, $in, $lt, $lte, $ne
- Logical
  - $and, $or, $nor, $not
- Element
  - $exists, $type
- Evaluation
  - $expr, $mod, $regex, $text, $where
- Array
  - $all, $size, $elemMatch

Projections

- $
- $elemMatch
- $meta
- $slice

Multiple conditions

Compound queries

- They can specify conditions for more than one field in the collection's documents.

AND conditions

```
db.COLLECTION.find({
                    <field>:<value>,
                    <field2>:<value>
        })
```

```
SELECT *
FROM COLLECTION
WHERE <field> = <value>
AND <field2> = <value;
```

OR conditions

```
db.COLLECTION.find({
        $or: [{<field>:<value>}, {<field2>:<value>}]
})
```

```
SELECT *
FROM COLLECTION
WHERE <field> = <value>
OR <field2> = <value;
```

Examples

- db.users_profile.find({
  firstName : {
  $in :{“John”, “Kedar”}
  }
  })

- SELECT * FROM user_profiles WHERE firstName in('John', 'Kedar’);

- db.users_profile.find({
  firstName : “John”,
  age : { $lt : 29 }
  })

- SELECT * FROM user_profiles WHERE firstName='John' AND age<30;

- db.users_profile.find({
  $or : [
  { firstName : “John” },
  { age : { $lt : 30 } }
  ]
  })

- SELECT * FROM user_profiles WHERE firstName='John' or age<30

db.users_profile.find({
“firstName” : “John”,
$or : [
{ age : { $lt : 30 } }
{ “lastName” : /^s/ },
]
})

SELECT * FROM user_profiles WHERE firstName='John’ AND age<30 OR lastName like 's%';

Querying arrays

To specify equality condition on an array, use the query document { <field>: <value> } where <value> is the exact array to match, including the order of the elements.

- db.COLLECTION.find({ <array_field> : [<value1>, <value2>, …] })

If, instead, you wish to find an array that contains values, without regard to order or other elements in the array, use the $all operator:

- db.COLLECTION.find({ <array_field> : { $all : [<value1>, <value2>, …] } })

To query if the array field contains at least one element with the specified value, use the filter { <field>: <value> } where <value> is the element value.

- db.COLLECTION.find({ <array_field> : <value> })

Specifying conditions on array elements

To specify conditions on the elements in the array field, use query operators in the query filter document:

- db.COLLECTION.find({ <array_field> : { <operator> : <value> } })

Compound filter conditions will match documents where all conditions are met by a combination of one or more array elements:

- db.COLLECTION.find({ <array_field> : { <operator1> : <value1>, <operator2> : <value2>, … } })

Use $elemMatch operator to specify multiple criteria on the elements of an array such that at least one array element satisfies all the specified criteria:

- db.COLLECTION.find({ <array_field> : { $elemMatch : {<operator1> : <value1>, <operator2> : <value2>, … } } })

Use the $size operator to query for arrays by number of elements.

- db.inventory.find({ <array_field> : { $size: <size> } })

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

- db.COLLECTION.find({ <array_field>.<index> : <value> })

Examples

{ "title": "A Beginner's Guide to MongoDB",

“author": “Jane Smith",

"tags": ["MongoDB", "NoSQL", "Databases"] }

{ "title": "10 Tips for Writing Clean Code",

“author": “John Smith",

"tags": ["Programming", "Best Practices", "Clean Code"] }

{ "title": "Building a RESTful API with Express",

"body": "Express is a popular framework for building...",

"tags": ["Node.js", "Express", "APIs"] }

- Find all blog posts with the tag "MongoDB":
  - db.blogposts.find({ tags: "MongoDB" });
- Find all blog posts with the tag "Programming" or "Node.js":
  - db.blogposts.find({ tags: { $in: ["Programming", "Node.js"] } });
- Find all blog posts with at least three tags:
  - db.blogposts.find({ tags: { $size: 3 } });
- Find all unique tags used in the collection:
  - db.blogposts.distinct("tags");
- Find all blog posts that were written by "Jane Smith" and have the tag "MongoDB".
  - db.blogposts.find({ author: "Jane Smith", tags: { $elemMatch: { $eq: "MongoDB" } }});

Querying nested documents

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.COLLECTION.find({ <outer_field>: { <field> : <value>, <field> : <value>, … } })
- With only match documents with outer_field's value EXACTLY like provided document (including order)

To specify a query condition on fields in an embedded/nested document, use dot notation.

- db.COLLECTION.find({ <outer_field>.<inner_field> : <value> })

Also, possible to query arrays of documents. For instance:

- db.COLLECTION.find({ <array_field> : { <inner_field> : <value>, … } })
- db.COLLECTION.find({ <array_field>.<inner_field> : <value> })
- db.COLLECTION.find({ <array_field>.<inner_field> : { <operator> : <value> } })
- db.COLLECTION.find({ <array_field> : { $elemMatch : { <inner_field> : <value>, … } } })
- db.COLLECTION.find({ <array_field>.<index>.<inner_field> : <value>)

Examples


```
{  "name": "John Doe",
  "email": "johndoe@example.com",
  "contact": {
    "phone": "123-456-7890",
    "address": {
      "street": "123 Main St",
      "city": "Anytown",
      "state": "CA",
      "zip": "12345" } } },
```

```
{ "name": "Jane Smith",
  "email": "janesmith@example.com",
  "contact": {
    "phone": "987-654-3210",
    "address": {
      "street": "456 Oak St",
      "city": "Othertown",
      "state": "NY",
      "zip": "54321“ } } }
```


Find all users who live in California:

- db.users.find({ "contact.address.state": "CA" });

Find all users who have a phone number that starts with "123":

- db.users.find({ "contact.phone": /^123/ });

Find all users who live in "Anytown" or "Othertown":

- db.users.find({ "contact.address.city": { $in: ["Anytown", "Othertown"] } });

Examples

{ "title": "Getting started with MongoDB",

"content": "MongoDB is a popular NoSQL database...",

"comments": [

{ "name": "John", "email": "john@example.com", "comment": "Great article!“ },

{ "name": "Jane", "email": "jane@example.com", "comment": "Thanks for the helpful tips.“ }

] },

{ "title": "Building RESTful APIs with Node.js",

"content": "Node.js is a popular platform for building RESTful APIs...",

"comments": [

{ "name": "Bob", "email": "bob@example.com", "comment": "I'm excited to try this out!“ },

{ "name": "Alice", "email": "alice@example.com", "comment": "This is exactly what I needed.“ }

] }

Find all blog posts that have at least one comment by "John":

- db.blogposts.find({ "comments": { $elemMatch: { "name": "John" } } });

Find all blog posts that have a comment by "Jane" with the email "jane@example.com":

- db.blogposts.find({ "comments": { $elemMatch: { "name": "Jane", "email": "jane@example.com" } } });

Find all blog posts that have at least one comment with the word "helpful" in the comment:

- db.blogposts.find({ "comments": { $elemMatch: { "comment": /helpful/i } } });

Update

Update(<filter>, <update>, <options>)

- Filter: a pattern to match
- Update: the changes to be made to documents that match the pattern
- Options: additional parameters

db.collection.updateOne()

db.collection.updateMany()

db.collection.replaceOne()

- Replaces full document, except for _id field

Examples

```
db.users_profile.updateOne(
        { userId : 1 },
        { $set : {
                age : 30
        }
})

db.users_profile.updateMany(
        { age : 28 },
        { $set : {age : 35}
})
```

UPDATE users_profile SET age = 35 WHERE age = 28;

```
Db.users_profile.replaceOne(
        { userId : 1 },
        {
                userId : 1,
                firsName : “Sam”,
                lastName : “Billings”,
                age : 28,
                email : “sam292@gmail.com”
})
```

Delete operation

Matches and deletes documents

db.collection.deleteOne(<filter>)

db.collection.deleteMany(<filter>)

Examples

db.users_profile.deleteMany({})

- Deletes every document

DELETE from users_profie;

```
db.users_profile.deleteOne({
        userId : 1
})
```

DELETE FROM users_profile WHERE userId = 1;

Activity

AirBnB

Mongo DB has available sample databases. One of them has information about reviews given by Airbnb users.

For this exercise, perform the following actions:

1. Get access (either local or remote) to the **sample_airbnb** database
2. Perform queries to obtain the following information
    1. Determine the total number of documents available
    2. Determine the number of documents whose type of property is a house
    3. Determine the number of documents of properties in Portugal
    4. Determine the number of documents with at least 10 reviews
    5. Determine the names of the properties with more than 300 reviews
    6. Determine the different countries available

Answers

1. Determine the total number of documents available
    1. 869
2. Determine the number of documents whose type of property is a house
    1. 95
3. Determine the number of documents of properties in Portugal
    1. 82
4. Determine the number of documents with at least 10 reviews
    1. 432
5. Determine the names of the properties with more than 300 reviews
    1. "O'Porto Studio | Historic Center"
    2. "The Ohana at Volcanoes National Park!"
    3. "B & B Room Yoga Garden's Place"
    4. "ABEL'S IN DOWNTOWN - THE PLACE TO BE 1 OF 2"
    5. "Near Airport private room, 2 bedroom granny flat**"
    6. “5*Super host Apartment+terrace 8 min from Ramblas"
6. Determine the different countries available
    1. "United States",
    2. "Brazil",
    3. "Canada",
    4. "Portugal",
    5. "Hong Kong",
    6. "Turkey",
    7. "Spain",
    8. "Australia“

Additional exercises

Perform queries to obtain the following information

- Determine the number of properties with Wifi
- Determine the number of properties with TV
- Determine the number of properties with Wifi or TV
- Determine the number of properties with Wifi or TV in Spain

Aggregation

Aggregation

Definition

- Operation to group values from multiple documents together.
- Besides, they can perform different operations on the grouped data to return a single result

Types

- Aggregation pipeline
- Map-reduce function*
- Single purpose aggregation method

Summary: Aggregation pipeline

Definition

- It's a sequence of stages.
- Each stage transforms the documents as they pass through the pipeline

Stages

- $match
  - Filters the documents according to a given expression
- $project
  - Filters the fields in all the documents (adds or removes fields)
- $group
  - Groups documents according to an expression
- $bucket
  - Groups documents according to a list of thresholds.
- $lookup
  - Left outer join to another collection
- $unwind
  - Deconstructs an array and returns one document per value
- $count
  - Determines the number of documents
- $sort
  - Reorders the documents according to a given field
- $limit
  - Select only the given number of documents
- $out
  - Writes the resulting documents to a collection

Filtering documents

Command: $match

Definition: Filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage.

Syntax:

```
{ $match: { <query> } }
```

Filtering fields

Command: $project

Definition: Passes along the documents with the requested fields to the next stage in the pipeline. The specified fields can be existing fields from the input documents or newly computed fields.

Syntax: { $project: { <specification(s)> } }

Syntax	Description
<field>: 1	Includes the field
<field>: 0	Exludes the field
_id: 0	Excludes the ID
<field>: <expression>	Adds a new field or resets the value of an existing field.

Example

Obtain only the names of the properties in Spain

```
db.listingsAndReviews.aggregate([
        { $match: {"address.country": "Spain"} },
        { $project: {"name": 1, _id: 0} }
]);
```


{ "name" : "Nice room in Barcelona Center" }
{ "name" : "Grand apartment Sagrada Familia" }
{ "name" : "Cozy bedroom Sagrada Familia" }
{ "name" : "Park Guell apartment with terrace" }
{ "name" : "DOUBLE ROOM for 1 or 2 ppl" }
{ "name" : "Trendy apartment in BCN centre" }
{ "name" : "Great location in Barcelona" }
{ "name" : "Room in the very ♥ of BCN - Unbeatable location" }
{ "name" : "Triple room Barcelona Guell park" }
{ "name" : "Alcam Colón 42 Apartment" }
{ "name" : "Alcam Colon 52 Apartment" }
{ "name" : "" }
{ "name" : "MONTJUIC FOUNTAINS APARTMENT" }
{ "name" : "Spacious and quiet duplex apartment in Poble Sec" }
{ "name" : "2 confortables habitaciones en el centro de BCN!" }
{ "name" : "GRACE BEAUTIFUL LOFT IN BARCELONA-At least 32 days" }
{ "name" : "Amazing NEW Food Lover Balcony, close to FIRA, MWC" }
{ "name" : "habitación doble tranquila con luz" }
{ "name" : "1 or 2 rooms with private terrace in shared flat" }
{ "name" : "SPACIOUS RAMBLA CATALUÑA" }

Grouping

Command: $group

Definition: Groups input documents by the specified _id expression and for each distinct grouping, outputs a document.

Syntax:

```
{
 $group:
  {
   _id: <expression>, // Group By Expression
   <field1>: { <accumulator1> : <expression1> },
   ...
  }
}
```

Accumulator operators:

$avg, $first, $last, $max, $min, $sum, $push

Examples

Determine the number of properties per country

```
db.listingsAndReviews.aggregate([
    { $group: {
        _id: "$address.country",
        "total": { $sum: 1}
      }
    }
]);
```

```
{ "_id" : "Canada", "total" : 69 }
{ "_id" : "Hong Kong", "total" : 71 }
{ "_id" : "Brazil", "total" : 167 }
{ "_id" : "Australia", "total" : 69 }
{ "_id" : "United States", "total" : 242 }
{ "_id" : "Portugal", "total" : 82 }
{ "_id" : "Turkey", "total" : 89 }
{ "_id" : "Spain", "total" : 80 }
```

Examples

Determine the lowest and highest prices for a property on every country.

```
db.listingsAndReviews.aggregate([
        { $group: {
                _id: "$address.country",
                "min_price": {$min: {$toDouble: "$price"} },
                "max_price": {$max: {$toDouble: "$price"} }
                }
        }
]);
```

```
{ "_id" : "Canada", "min_price" : 20, "max_price" : 240 }
{ "_id" : "Hong Kong", "min_price" : 181, "max_price" : 11681 }
{ "_id" : "Brazil", "min_price" : 48, "max_price" : 11190 }
{ "_id" : "Australia", "min_price" : 36, "max_price" : 999 }
{ "_id" : "Portugal", "min_price" : 14, "max_price" : 200 }
{ "_id" : "United States", "min_price" : 34, "max_price" : 1000 }
{ "_id" : "Spain", "min_price" : 15, "max_price" : 805 }
{ "_id" : "Turkey", "min_price" : 37, "max_price" : 1213 }
```

Examples

The average number of reviews per country

```
db.listingsAndReviews.aggregate([
        { $group: {
                _id: "$address.country",
                "reviews": {$avg: "$number_of_reviews"},
                }
        }
]);
```

```
{ "_id" : "Canada", "reviews" : 18.144927536231883 }
{ "_id" : "Hong Kong", "reviews" : 39.098591549295776 }
{ "_id" : "Brazil", "reviews" : 10.946107784431138 }
{ "_id" : "Australia", "reviews" : 48.231884057971016 }
{ "_id" : "United States", "reviews" : 50.36776859504132 }
{ "_id" : "Portugal", "reviews" : 70.58536585365853 }
{ "_id" : "Turkey", "reviews" : 12.910112359550562 }
{ "_id" : "Spain", "reviews" : 51.5125 }
```

Examples

Determine the number of properties having Wifi service on every country.

```
db.listingsAndReviews.aggregate([
    {$match: {"amenities": {$all: ["Wifi"]}}},
    {$group: {
            _id: "$address.country",
            "total": { $sum: 1}
        }}
]);
```

```
{ "_id" : "Portugal", "total" : 80 }
{ "_id" : "United States", "total" : 237 }
{ "_id" : "Spain", "total" : 78 }
{ "_id" : "Australia", "total" : 63 }
{ "_id" : "Turkey", "total" : 82 }
{ "_id" : "Hong Kong", "total" : 62 }
{ "_id" : "Canada", "total" : 68 }
{ "_id" : "Brazil", "total" : 154 }
```

Creating buckets

Command: $bucket

Definition: Categorizes incoming documents into groups, called buckets, based on a specified expression and bucket boundaries.

Syntax:

```
{ $bucket: {
    groupBy: <expression>,
    boundaries: [ <lowerbound1>, <lowerbound2>, ... ],
    default: <literal>,
    output: {
      <output1>: { <$accumulator expression> },
      ...
      <outputN>: { <$accumulator expression> }
    }
  }
}
```

Boundaries:

An array of values based on the groupBy expression that specify the boundaries for each bucket. Each adjacent pair of values acts as the inclusive lower boundary and the exclusive upper boundary for the bucket.

Example

Determine the number of properties in Spain for each of the following prices' ranges: 0-100, 100-300, 300-500, 500+

```
db.listingsAndReviews.aggregate([
        {$match: {"address.country" : "Spain"}},
        {$bucket: {
                groupBy: "$price",
                boundaries: [0,100,300,500],
                default: "500+",
                output: {"total": {$sum:1}}
        }}
]);
```

```
{ "_id" : 0, "total" : 55 }
{ "_id" : 100, "total" : 19 }
{ "_id" : 300, "total" : 3 }
{ "_id" : "500+", "total" : 3 }
```

Making joins

Command: $lookup

Definition: Performs a left outer join to a collection in the same database to filter in documents from the “joined” collection for processing.

Syntax:

```
{ $lookup:
    {
      from: <collection to join>,
      localField: <field from the input documents>,
      foreignField: <field from the documents of the "from" collection>,
      as: <output array field>
    }
}
```

Examples

Id	Item	Price	Quantity
1	Almonds	12	2
2	Pecans	20	1
3			

ID	SKU	Description	Instock
1	Almonds	Product 1	120
2	Bread	Product 2	80
3	Cashews	Product 3	60
4	Pecans	Product 4	70
5	Null	Incomplete	
6			

Example

Obtain the inventory details of every product in an order

```
db.orders.aggregate([
 {
   $lookup: {
      from: "inventory",
      localField: "item",
      foreignField: "sku",
      as: "inventory_docs"
   }
 }
])
```

```
{
  "_id" : 1,
  "item" : "almonds",
  "price" : 12,
  "quantity" : 2,
  "inventory_docs" : [
    { "_id" : 1, "sku" : "almonds", "description" : "product 1",
"instock" : 120 }
  ]
}
…
```

Unwinding values in an array

Command: $unwind

Definition: Deconstructs an array field from the input documents to output a document for each element. Each output document is the input document with the value of the array field replaced by the element.

Syntax:

```
{ $unwind: <field path> }
```

Example:

Obtain the number of properties having each of the amenities:

```
db.listingsAndReviews.aggregate([
        {$unwind: "$amenities"},
        {$group: {
                _id: "$amenities",
                "total": {$sum: 1}
        }},
        {$sort: {"total": -1}}
]);
```

```
{ "_id" : "Wifi", "total" : 5303 }
{ "_id" : "Essentials", "total" : 5048 }
{ "_id" : "Kitchen", "total" : 4951 }
{ "_id" : "TV", "total" : 4295 }
{ "_id" : "Hangers", "total" : 4226 }
{ "_id" : "Hair dryer", "total" : 3900 }
{ "_id" : "Washer", "total" : 3877 }
{ "_id" : "Shampoo", "total" : 3709 }
{ "_id" : "Iron", "total" : 3692 }
{ "_id" : "Laptop friendly workspace", "total" : 3442 }
{ "_id" : "Air conditioning", "total" : 3431 }
```

Information resources

Mongo DB

- Documentation
  - https://docs.mongodb.com/manual/introduction/
- Aggregation
  - https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/#aggregation-pipeline-operator-reference
  - https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/

Sharding

Sharding

Definition

- Method for distributing data across multiple machines.
- It's specially used to support deployments with very large data sets and high throughput operations.

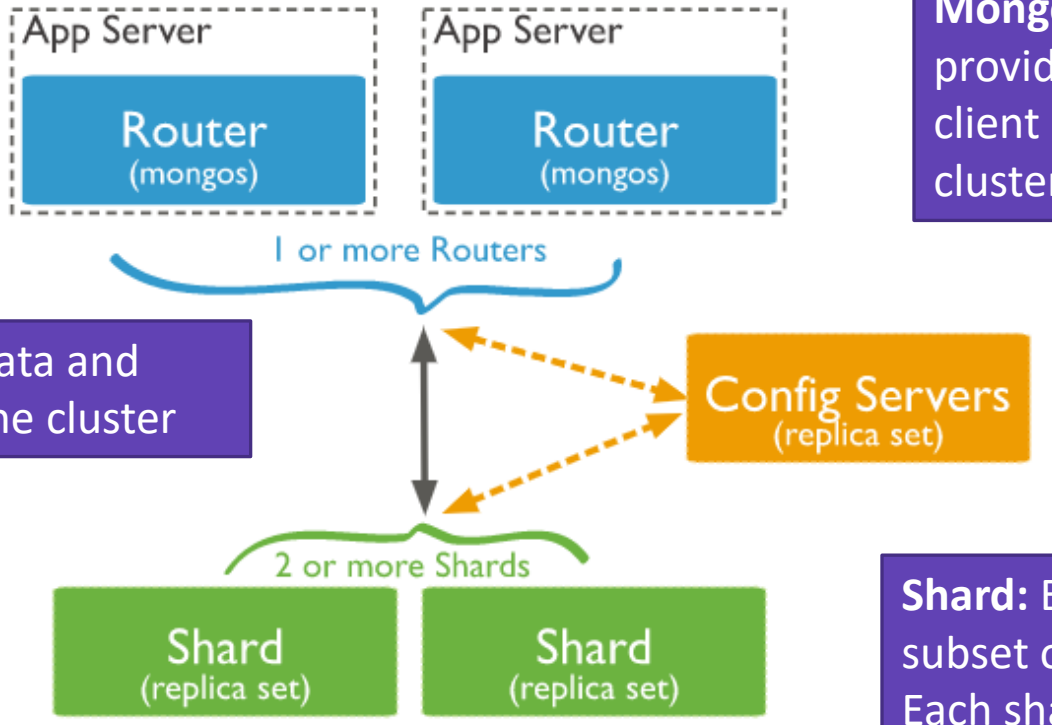
Scenarios

- Database systems with large data sets or high throughput applications challenge the server's capacity
  - High query rates can exhaust the CPU capacity of the server
  - Working set sizes larger than the system's RAM stress the I/O capacity of disk drives

Methods for addressing system growth

- ***Vertical Scaling.*** Increase the capacity of a server (use a more powerful CPU, add more RAM, or increase the amount of storage space)
- ***Horizontal Scaling.*** Involves dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required.

Sharding architecture



Mongos: Act as query routers, providing an interface between client applications and the sharded cluster.

Config server: Store metadata and configuration settings for the cluster

Shard: Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set.

Information resources

Mongo DB

- Documentation
  - https://docs.mongodb.com/manual/introduction/
- Collection methods
  - https://docs.mongodb.com/manual/reference/method/js-collection/
- Aggregation
  - https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/#aggregation-pipeline-operator-reference