MongoDB

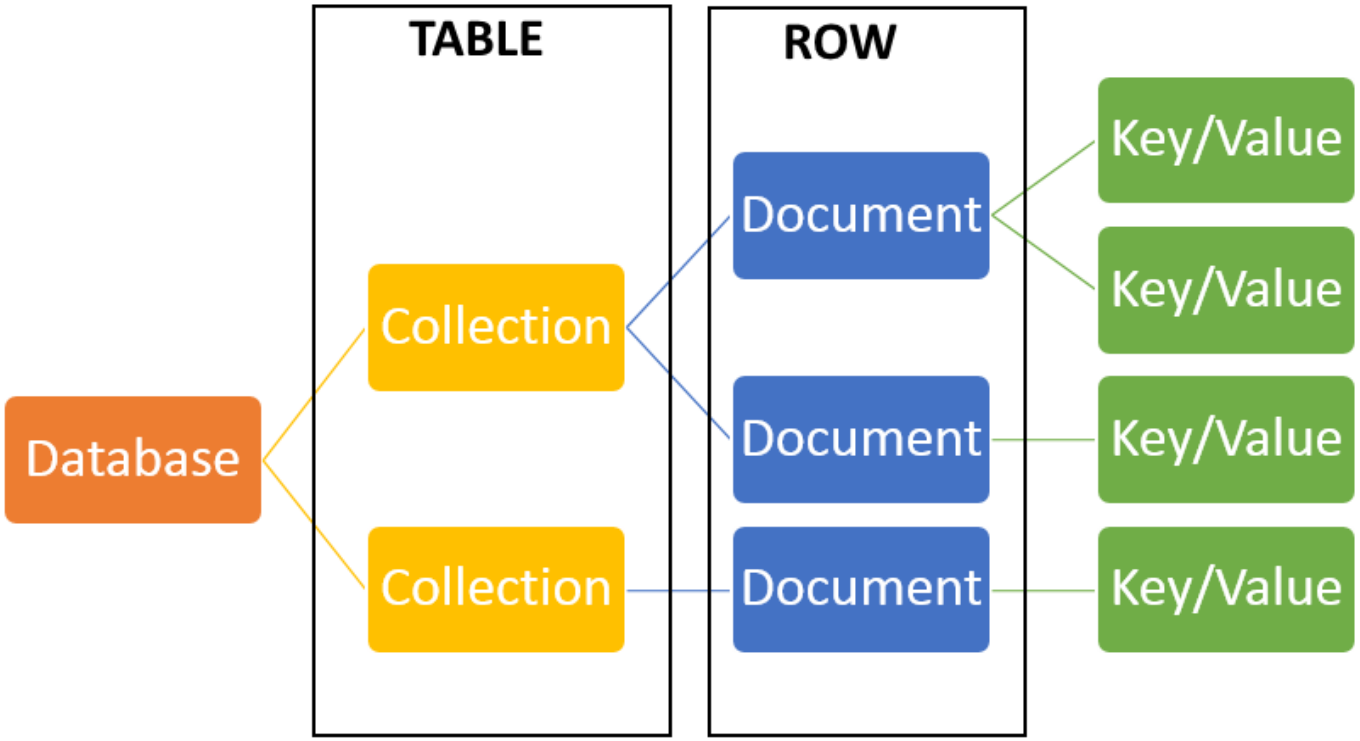
Introduction

- MongoDB is an open-source database developed by MongoDB, Inc. (https://www.mongodb.com)
- MongoDB stores data in JSON-like (BSON) documents that can vary in structure.
- Related information is stored together for fast query access through the MongoDB query language.
- MongoDB uses dynamic schemas.

History

- 2007 - First developed (by 10gen)
- 2009 - Become Open Source (version 1.2)
- 2010 - Considered production ready (version 1.4)
- *2013 - MongoDB Closes $150 Million in Funding*
- 2014 - Latest stable version (version 2.6)
- 2015 – Version 3.0
- 2018 – Version 4.0
- 2021 – Version 5.0
- 2022 – Version 6.0
- August, 15 2023 – Version 7.0

MongoDB structure



Terminology and Concepts

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

SQL to Aggregation Mapping Chart

SQL Terms, Functions, and Concepts	MongoDB Aggregation Operators
WHERE	$match
GROUP BY	$group
HAVING	$match
SELECT	$project
ORDER BY	$sort
LIMIT	$limit
SUM()	$sum
COUNT()	$sum/$sortByCount
join	$lookup
SELECT INTO NEW_TABLE	$out
MERGE INTO TABLE	$merge
UNIONALL	$unionWith

MongoDB - Advantages

- Flexible Data Model
- Expressive Query Syntax
- Easy to Learn
- Performance
- Scalable and Reliable
- Reactive Streams Drivers
- Documentation
- Text Search
- Server-Side Script
- Documents = Objects

MongoDB – The bad

- Joins not Supported
- High Memory Usage
- Limited Data Size
- Limited Nesting
- No Triggers
- Duplicate Data

Insert document

- db.collection.insertOne()
- db.collection.insertMany()

SQL INSERT Statements	MongoDB insertOne() Statements

```
INSERT INTO people(user_id,
                   age,
                   status)
VALUES ("bcd001",
        45,
        "A")
```

```
db.people.insertOne(
   { user_id: "bcd001", age: 45, status: "A" }
)
```

```
try {
   db.products.insertMany( [
      { item: "card", qty: 15 },
      { item: "envelope", qty: 20 },
      { item: "stamps" , qty: 30 }
   ] );
} catch (e) {
   print (e);
}
```

Find document(s)

db.collection.find(query, projection)

SQL SELECT Statements	MongoDB find() Statements
`SELECT *` `FROM people`	`db.people.find()`

`SELECT id,`
`user_id,`
`status`
`FROM people`
	`db.people.find(`
	`{ },`
	`{ user_id: 1, status: 1 }`
	`)`

`SELECT user_id, status`
`FROM people`
	`db.people.find(`
	`{ },`
	`{ user_id: 1, status: 1, _id: 0 }`
	`)`

`SELECT *`
`FROM people`
`WHERE status = "A"`
	`db.people.find(`
	`{ status: "A" }`
	`)`

```
SELECT user_id, status
FROM people
WHERE status = "A"
```

```
db.people.find(
    { status: "A" },
    { user_id: 1, status: 1, _id: 0 }
)
```

```
SELECT *
FROM people
WHERE status != "A"
```

```
db.people.find(
    { status: { $ne: "A" } }
)
```

```
SELECT *
FROM people
WHERE status = "A"
AND age = 50
```

```
db.people.find(
    { status: "A",
      age: 50 }
)
```

```
SELECT *
FROM people
WHERE status = "A"
OR age = 50
```

```
db.people.find(
    { $or: [ { status: "A" } ,
             { age: 50 } ] }
)
```

```
SELECT *                    db.people.find(
FROM people                     { age: { $gt: 25 } }
WHERE age > 25              )
```

```
SELECT *                    db.people.find(
FROM people                     { age: { $lt: 25 } }
WHERE age < 25              )
```

```
SELECT *                    db.people.find(
FROM people                     { age: { $gt: 25, $lte: 50 } }
WHERE age > 25              )
AND   age <= 50
```

```
SELECT *                    db.people.find( { user_id: /bc/ } )
FROM people
WHERE user_id like "%bc%"   -or-

                            db.people.find( { user_id: { $regex: /bc/ } } )
```

```
SELECT *
FROM people
WHERE user_id like "bc%"
```

```
db.people.find( { user_id: /^bc/ } )
```

-or-

```
db.people.find( { user_id: { $regex: /^bc/ } } )
```

```
SELECT *
FROM people
WHERE status = "A"
ORDER BY user_id ASC
```

```
db.people.find( { status: "A" } ).sort( { user_id: 1 } )
```

```
SELECT *
FROM people
WHERE status = "A"
ORDER BY user_id DESC
```

```
db.people.find( { status: "A" } ).sort( { user_id: -1 } )
```

```
SELECT COUNT(*)
FROM people
```

```
db.people.count()
```

or

```
db.people.find().count()
```

```
SELECT COUNT(user_id)
FROM people
```

```
db.people.count( { user_id: { $exists: true } } )
```

or

```
db.people.find( { user_id: { $exists: true } } ).count()
```

```
SELECT COUNT(*)
FROM people
WHERE age > 30
```

```
db.people.count( { age: { $gt: 30 } } )
```

or

```
db.people.find( { age: { $gt: 30 } } ).count()
```

```
SELECT DISTINCT(status)
FROM people
```

```
db.people.distinct( "status" )
```

```
SELECT *
FROM people
LIMIT 1
```

```
db.people.findOne()
```

or

```
db.people.find().limit(1)
```

```
SELECT *
FROM people
LIMIT 5
SKIP 10
```

```
db.people.find().limit(5).skip(10)
```

Explain query

```
EXPLAIN SELECT *
FROM people
WHERE status = "A"
```

```
db.people.find( { status: "A" } ).explain()
```

Others criteria

- limit()
- skip()
- explain()
- sort()
- count()
- pretty()
- …

Update document

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

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

db.collection.replaceOne(<filter>, <replacement>, <options>)

SQL Update Statements	MongoDB updateMany() Statements

`UPDATE people`
`SET status = "C"`
`WHERE age > 25`
	`db.people.updateMany(`
	`{ age: { $gt: 25 } },`
	`{ $set: { status: "C" } }`
	`)`

`UPDATE people`
`SET age = age + 3`
`WHERE status = "A"`
	`db.people.updateMany(`
	`{ status: "A" } ,`
	`{ $inc: { age: 3 } }`
	`)`

Delete document

- db.collection.deleteMany()
- db.collection.deleteOne()

SQL Delete Statements	MongoDB deleteMany() Statements
`DELETE FROM people` `WHERE status = "D"`	`db.people.deleteMany( { status: "D" } )`
`DELETE FROM people`	`db.people.deleteMany({})`

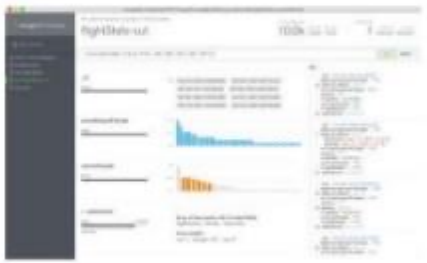
Drop databse

- MongoDB db.dropDatabase() command is used to drop a existing database.
- This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

```
>use mydb
switched to db mydb
>db.dropDatabase()
>{ "dropped" : "mydb", "ok" : 1 }
>
```

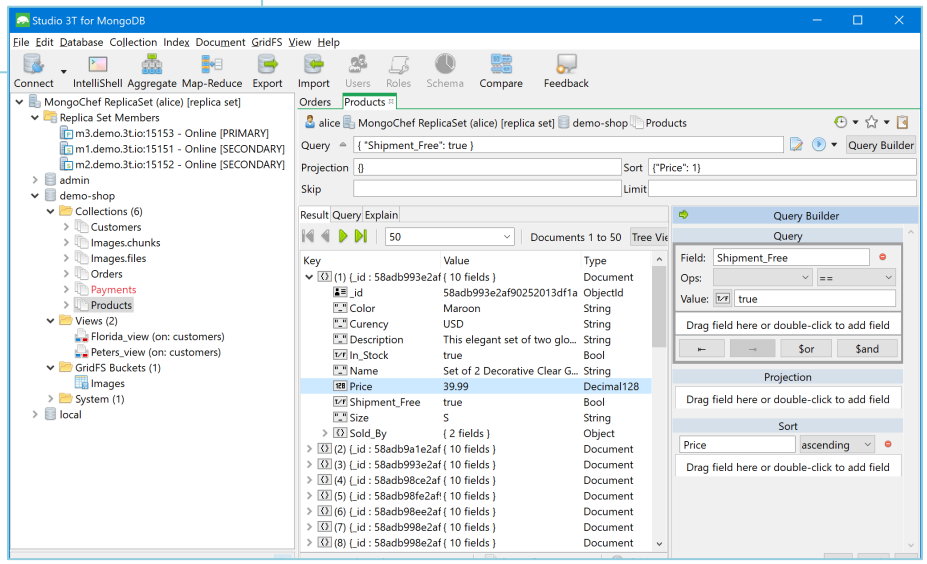
Using Management tools

MongoDB Compass



For fast schema discovery and visual construction of ad-hoc queries

- Visualize schema
  - Frequency of fields
  - Frequency of types
  - Determine validator rules
- View Documents
- Graphically build queries
- Authenticated access



Authentication enable

- Grant permission to users to authenticate
  - Central database
  - Each database
- Policies:
  - readAnyDatabase -> qa/tester
  - readWriteAnyDatabase -> developer
  - userAdminAnyDatabase -> admin
  - dbAdminAnyDatabase -> admin

Authentication enable

1. Create admin database
2. Add admin user

```
> use admin
switched to db admin
> db.createUser(
...   {
...     user: "admin",
...     pwd: "abc123",
...     roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
...   }
... )
Successfully added user: {
        "user" : "admin",
        "roles" : [
                {
                        "role" : "userAdminAnyDatabase",
                        "db" : "admin"
                }
        ]
}
```

3. Client logon:

   mongo -u "admin" -p "abc123" -authenticationDatabase "admin"

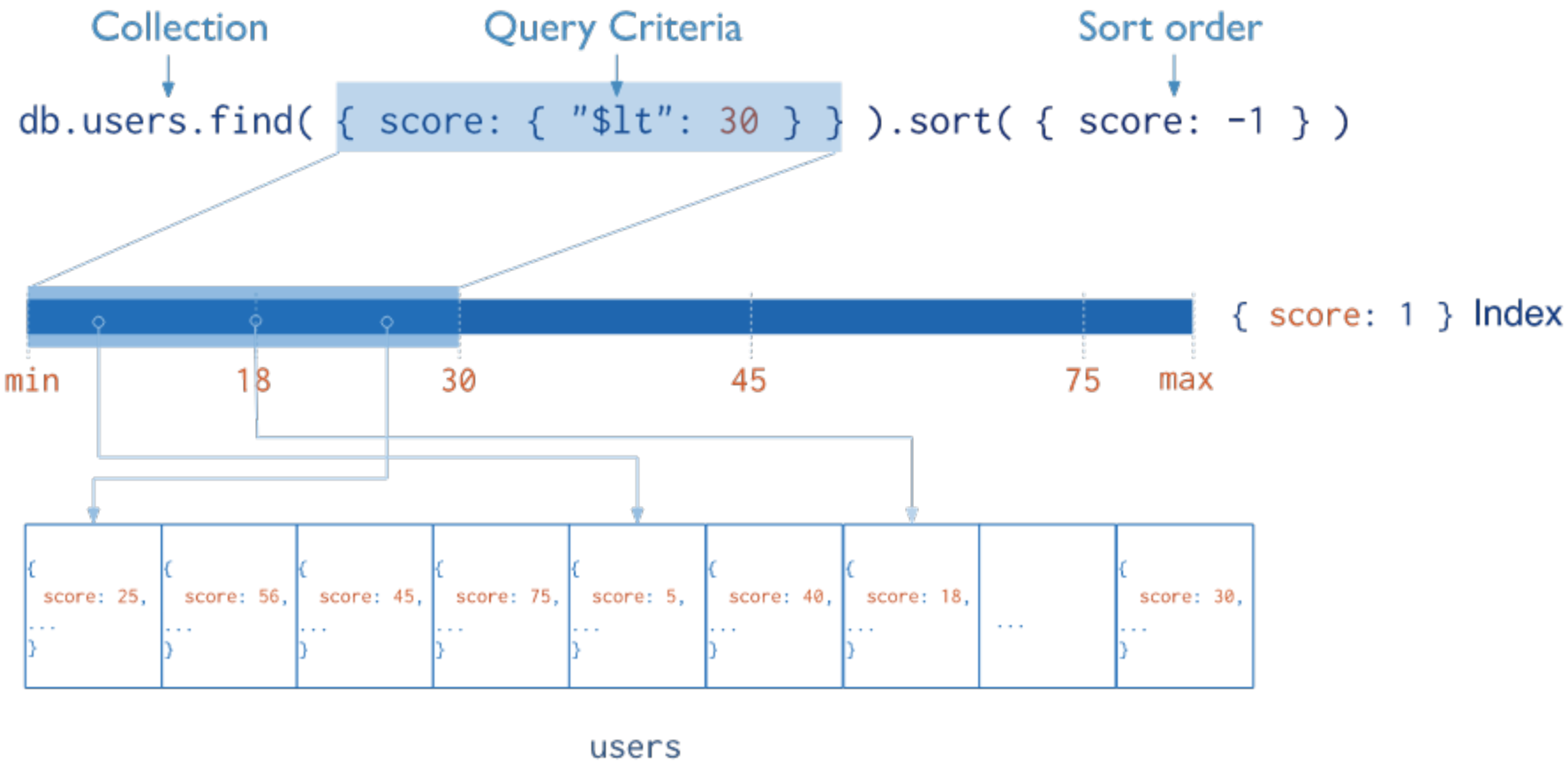
MongoDB Java Drivers

- Driver:
  https://www.mongodb.com/docs/drivers/java-drivers/
- Sync
  https://www.mongodb.com/docs/drivers/java/sync/current/
- MongoDB Java Reactive Streams
  https://www.mongodb.com/docs/drivers/reactive-streams/

MongoDB - Indexes

- Indexes support the efficient execution of queries in MongoDB. Without indexes, MongoDB must perform a collection scan, i.e. scan every document in a collection, to select those documents that match the query statement.
- If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect.
- MongoDB defines indexes at the collection level and supports indexes on any field or sub-field of the documents in a MongoDB collection.

MongoDB - Indexes



MongoDB - Indexes

Default _id Index

- MongoDB creates a unique index on the _id field during the creation of a collection.

Create an Index

- Creates indexes on collections:

  db.collection.createIndex(<keys>, <options>)

  Options:

  - An ascending index: 1
  - A descending index: -1

MongoDB - Indexes

Index Types (1)

- **Single Field:** MongoDB supports the creation of user-defined ascending/descending indexes on a single field of a document.
- **Compound Indexes:** MongoDB supports compound indexes, where a single index structure holds references to multiple fields within a collection's documents.
- **Multikey Indexes:** To index a field that holds an array value, MongoDB creates an index key for each element in the array.

MongoDB - Indexes

Index Types (2)

- **Text Indexes:** MongoDB provides text indexes to support text search queries on string content.

  To create index on a field that contains a string or an array of string elements, include the field and specify the string literal "text" in the index document.

  Ex: db.people.createIndex({firstname: "text" })

- **Wildcard Indexes:** MongoDB 4.2 introduces wildcard indexes for supporting queries against unknown or arbitrary fields.
  - Create a wildcard index on a field:

    db.collection.createIndex({ "fieldA.$" : 1 })**
  - Create a Wildcard Index on All Fields:

    db.collection.createIndex({ "$" : 1 })**

MongoDB - Indexes

Index Properties

- **Unique Indexes:** A unique index ensures that the indexed fields do not store duplicate values.

  Create a Unique Index:

  ```
  db.collection.createIndex( <keys>, { unique: true } )
  ```

- **Partial Indexes:** Partial indexes only index the documents in a collection that meet a specified filter expression.

  To create a partial index, use **db.collection.createIndex()** method with the partialFilterExpression option.

  Ex:
  ```
  db.restaurants.createIndex(
    { cuisine: 1, name: 1 },
    { partialFilterExpression: { rating: { $gt: 5 } } }
  )
  ```

MongoDB - Indexes

Index Properties

For example, the following operation creates a compound index that indexes only the documents with a rating field greater than 5.

```
db.restaurants.createIndex(
   { cuisine: 1, name: 1 },
   { partialFilterExpression: { rating: { $gt: 5 } } }
)
```

MongoDB - Indexes

Manage Indexes

- View Existing Indexes: db.collection.getIndexes()
- Remove Indexes:
  - Remove Specific Index: db.collection.dropIndex()
  - Remove All Indexes: db.collection.dropIndexes()

Ref:

- https://www.mongodb.com/docs/drivers/java/sync/current/fundamentals/indexes/