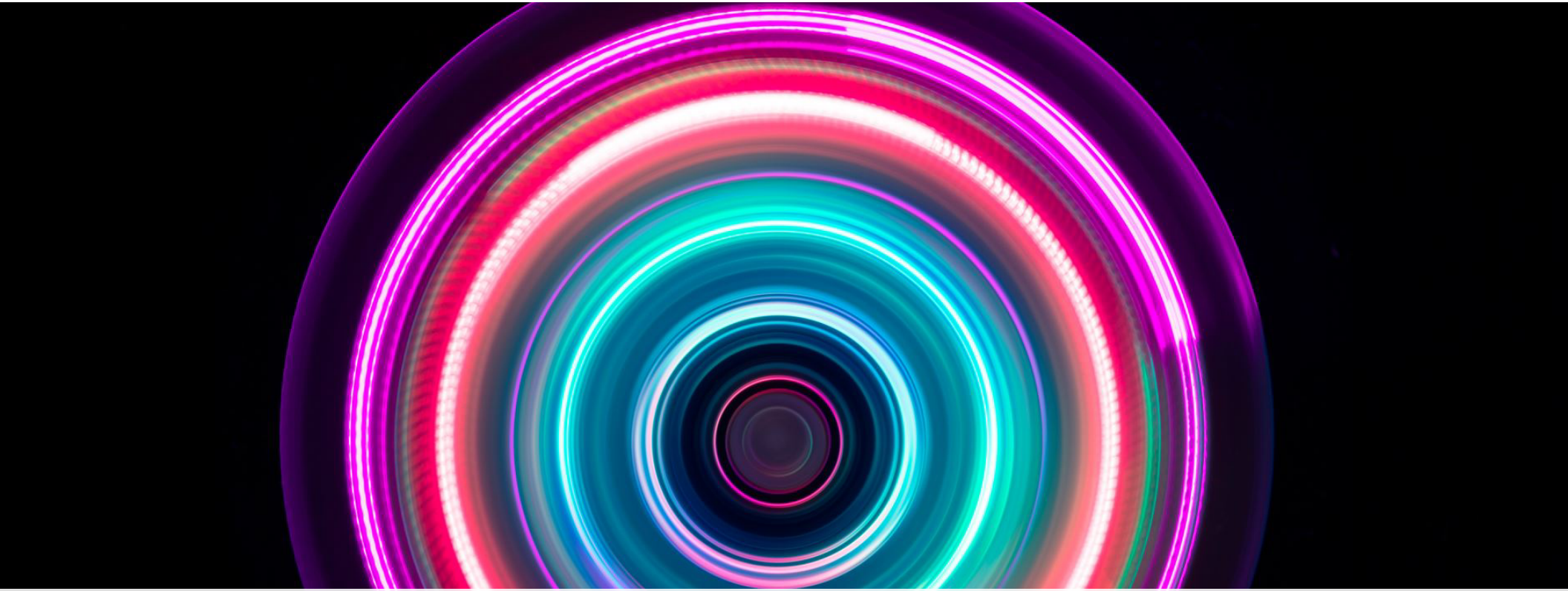


10 NOSQL (MONGO DB) AND OBJECT ORIENTED DATABASES

CS 340: INTRODUCTION TO DATABASE SYSTEMS

Adapted from Dr. Omar Alomeir
2023

Course instructor:
Dr. Maram Alajlan.

LEARNING GOALS

CLO1: Demonstrate the knowledge in designing, updating, and querying basic NoSQL-Database Systems. (Knowledge & Understanding)

Chapter objectives:

- Define the key concepts behind NoSQL database systems.
- Define the key concepts behind Object Oriented database systems.
- Compare relational and non-relational database concepts.
- Create a MongoDB database and perform basic operations.

TOPICS

1. Introduction of a NoSQL Database (MongoDB) including, installation, MongoDB Compass and Atlas
2. MongoDB Elements: Database, Collections and Documents
3. Perform basic NO SQL CRUD operations on Documents
   a. Creating documents
   b. Updating documents
   c. Deleting documents
   d. Querying documents
4. MongoDB Compass
5. Introduction of Object Oriented Databases.

RELATIONAL DATABASES

- The purpose of relational databases is the storage and retrieval of highly structured data used mainly in an organization.
- Relational model has persisted (with changes) since the 1970s.
- A relational database typically stores structured data in tables (relations) containing specific data and types of data.
- Relational databases use Structured Query Language (SQL).

NON-RELATIONAL DATABASES

Since the explosion of the internet and digital media in the 1990s and 2000s, new computer applications have emerged with significantly different database requirements. Example applications include:

- Social media
- Internet of things
- Web advertising

Data generated by new internet and multimedia applications is commonly called big data and differs from transactional data in four ways:

- **Volume**. Typical size ranges from terabytes to petabytes (million billion bytes), occasionally reaching exabytes (billion billion bytes).
- **Velocity**. Big data is generated at extremely high rates. Facebook users upload roughly a billion photos per day, or 10,000 per second.
- **Variety**. Variety means both unstructured and rapidly changing data types. Unstructured data refers to information embedded in complex data types like images, video, GPS coordinates, and natural language.
- **Veracity**. Transactional data is typically created by an organization's employees or trusted partners. Big data is often generated by the public. Consequently, the accuracy of big data varies much more than transactional data.

NON-RELATIONAL

Mainly driven by changes in the web, as well as new technologies such as Google's BigTable and MapReduce.

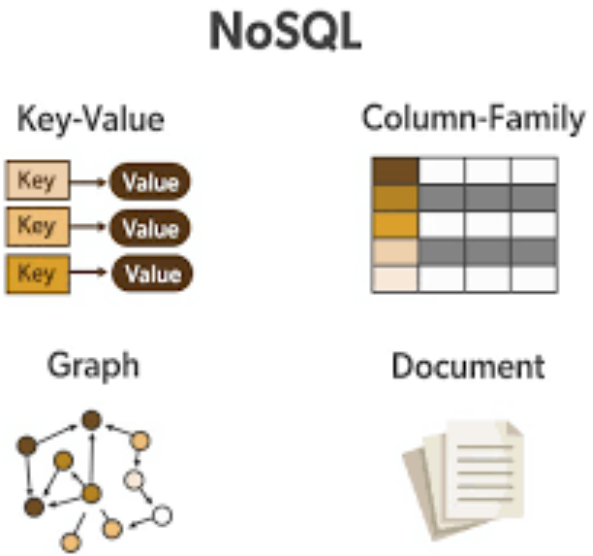
A move away from relational databases to something more flexible, that can handle large volumes of unstructured data. Simpler "horizontal" scaling to clusters of machines (which is a problem for relational databases). More control over availability.

Starting around 2009, settling down recently to somewhere in the middle.

Some NoSQL systems now support some form of SQL.

Relational systems support features introduced in NoSQL systems.

NOSQL EXAMPLES



NON-RELATIONAL DATABASES (NOSQL)-MONGODB

Non-relational databases are schema-less and are referred to as "NoSQL," which stands for **Non SQL** and sometimes as **Not Only SQL.**

A non-relational database stores data in a non-tabular form, and tends to be more flexible than the traditional, SQL-based, relational database structures.

NoSQL systems are also more flexible and have relaxed guarantees (The ACID requirements are relaxed in many big data applications).

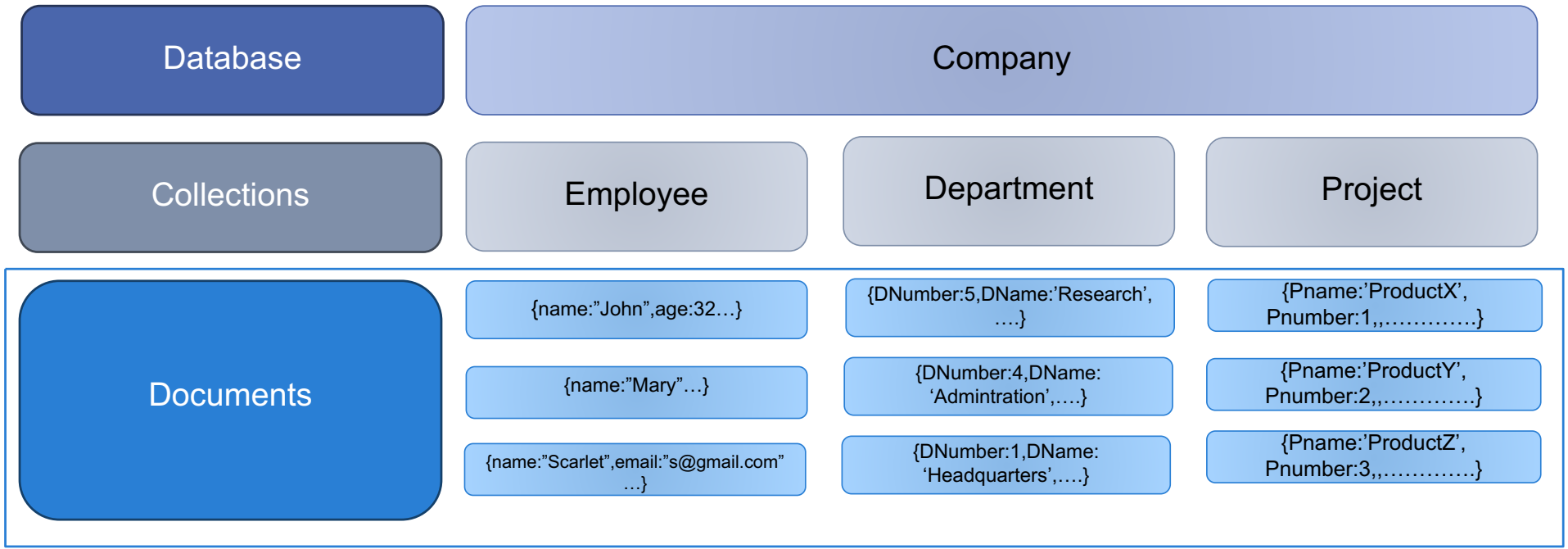
A document can be highly detailed while containing a range of different types of information in different formats.

MONGODB DOCUMENT EXAMPLE

```
{
        "_id" : "888665555",
       "Fname" : "James",
        "Minit" : "E",
       "LName" : "Borg",
        "dBirth" : "10-Nov-2007",
       "Address" : "450 Stone Houston TX",
       "Sex" : "M",
        "Salary" : 55000,
       "Dnumber" : "5",
       "Dependents" : [
                {
                    "DepName" : "Ann",
                     "DepSec" : "F",
                      "Relation" : "Daughter"
                }
          ]
}
```

Composite

NON-RELATIONAL DATABASES (NOSQL)-MONGODB



NON-RELATIONAL DATABASES (NOSQL)-MONGODB

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields (attributes). Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data. The following table shows the relationship of RDBMS terminology with MongoDB.

QUICK COMPARISON OF ELEMENTS

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)
Database Server and Client	
mysql/sqlplus	mongo

MongoDB

- No Schema
- Schema evolution easy
- Denormalized data
- No Joins
- Fetching is fast

Relational DB

- Rigid Schema
- Schema evolutions hard
- Normalized data
- Joins
- Fetching joined data is slower

MONGODB FEATURES

- Flexible Schema
- Easy to Scale
- Handles unstructured data
- Distributed database.

JSON IN MONGODB

JSON stands for **J**ava**S**cript **O**bject **N**otation. JSON is a standard text-based format for representing structured data based on JavaScript object syntax. It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).

JSON is a lightweight format for storing and transporting data

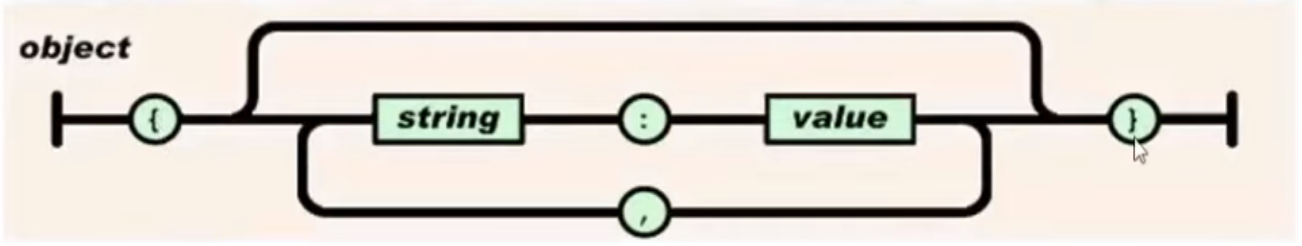
JSON is often used when data is sent from a server to a web page. JSON is "self-describing" and easy to understand

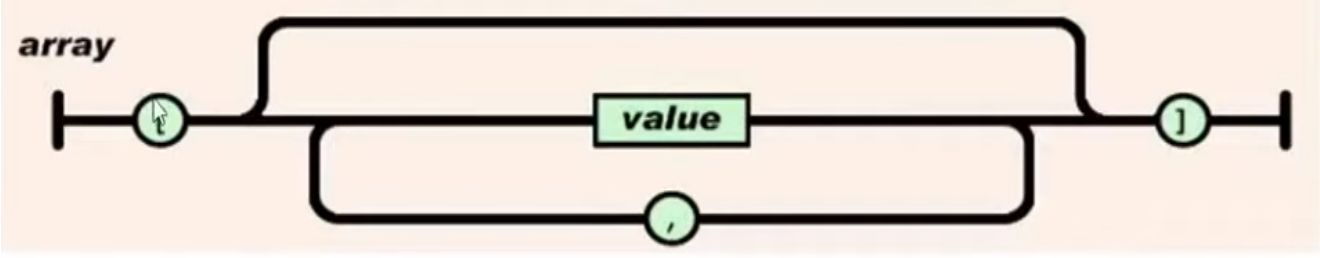
JavaScript objects are simple associative containers, wherein a string key is mapped to a value (which can be a number, string, function, or even another object). This simple language trait allowed JavaScript objects to be represented remarkably simply in text:

JSON Syntax Rules

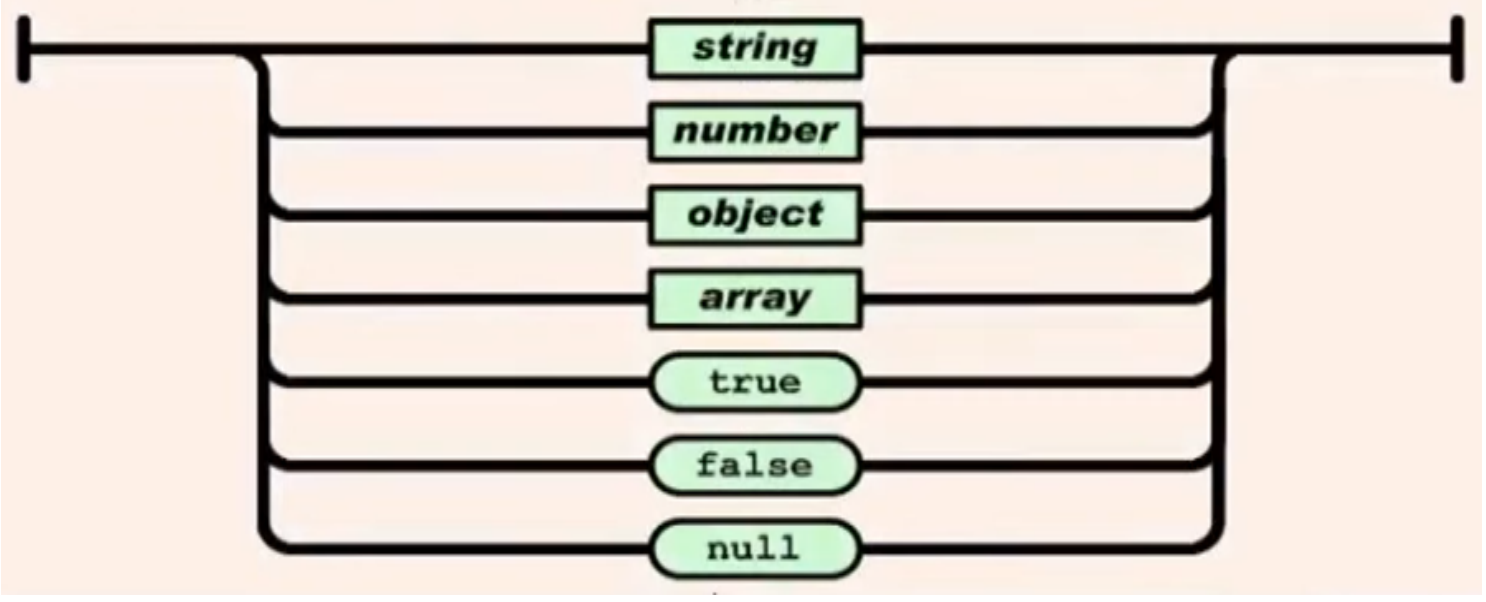
- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects

JSON OBJECT AND ARRAY





JSON DATA TYPES



JSON USED IN MONGODB

JSON data is written as name/value pairs, just like JavaScript object properties.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

"firstName":"John“

JSON objects are written inside curly braces.

Just like in JavaScript, objects can contain multiple name/value pairs:

{"firstName":"John", "lastName":"Doe"}

JSON names require double quotes. JavaScript names do not.

JSON USED IN MONGODB

JSON arrays are written inside square brackets.

Just like in JavaScript, an array can contain objects:

```
"employees":[
        { "firstName": "John",   "lastName": "Doe"},
        { "firstName": "Anna",   "lastName": "Smith"},
        { "firstName": "Peter",   "lastName": "Jones"}
]
```

In the example above, the object "employees" is an array. It contains three objects.

Each object is a record of a person (with a first name and a last name).

DISADVANTAGE OF JSON

However, there are several issues that make JSON less than ideal for usage inside of a database.

1. JSON is a text-based format, and text parsing is very slow
2. JSON's readable format is far from space-efficient, another database concern
3. JSON only supports a limited number of basic data types

DISADVANTAGE OF JSON

In order to make MongoDB JSON-first, but still high-performance and general-purpose, BSON (a superset of JSON with some more data types, most importantly binary byte array) was invented to bridge the gap: a binary representation to store data in JSON format, optimized for speed, space, and flexibility.

Does MongoDB use BSON, or JSON?

MongoDB stores data in BSON format both internally, and over the network, but that doesn't mean you can't think of MongoDB as a JSON database. Anything you can represent in JSON can be natively stored in MongoDB, and retrieved just as easily in JSON.

For example, MongoDB allows developers to query and manipulate objects by specific keys inside the JSON/BSON document, even in nested documents many layers deep into a record and create high performance indexes on those same keys and values.

Consider the following JSON document:

```
{
   "hello" : "world"
}
```

It's BSON equivalent will be:

```
\x16\x00\x00\x00              // Size of the Document
\x02                          // 0x02 = type String
hello\x00                     // field name
\x06\x00\x00\x00world\x00     // field value
\x00                          // Used to represent end of object
```

DATA TYPES USED IN MONGODB

MongoDB supports many datatypes. Some of them are −

String: This is the most commonly used datatype to store the data.

Integer: This type is used to store a numerical value. Integer can be 32 bit or 64 bit.

Boolean: This type is used to store a boolean (true/ false) value.

Double: This type is used to store floating point values.

Arrays: This type is used to store arrays or list or multiple values into one key.

Object: This datatype is used for embedded documents.

Null : a Null value.

Date : This datatype is used to store the current date or time in UNIX time format.

Object ID: This datatype is used to store the document's ID

CRUD Operations in MongDB

Create

Update

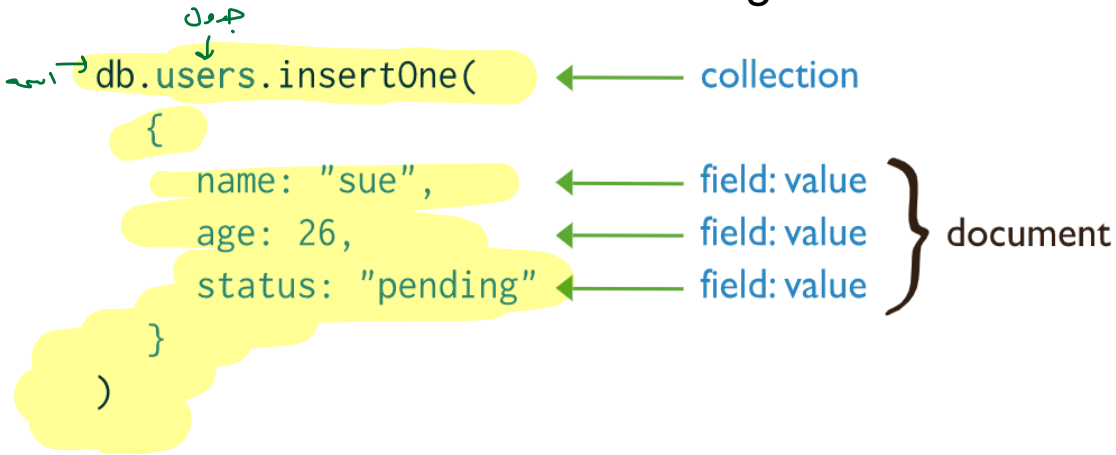
Read

Delete

CREATE OPERATIONS

Create or insert operations add new documents to a collection. If the collection does not currently exist, insert operations will create the collection.

In MongoDB, insert operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

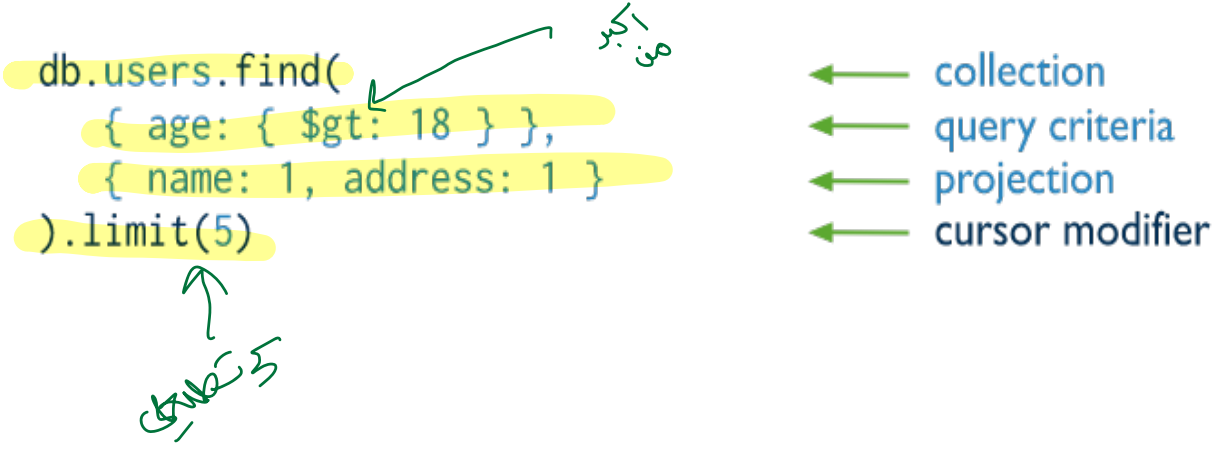


READ OPERATIONS

Read operations retrieve documents from a collection; i.e. query a collection for documents. MongoDB provides the following methods to read documents from a collection:

db.collection.find()

You can specify query filters or criteria that identify the documents to return.

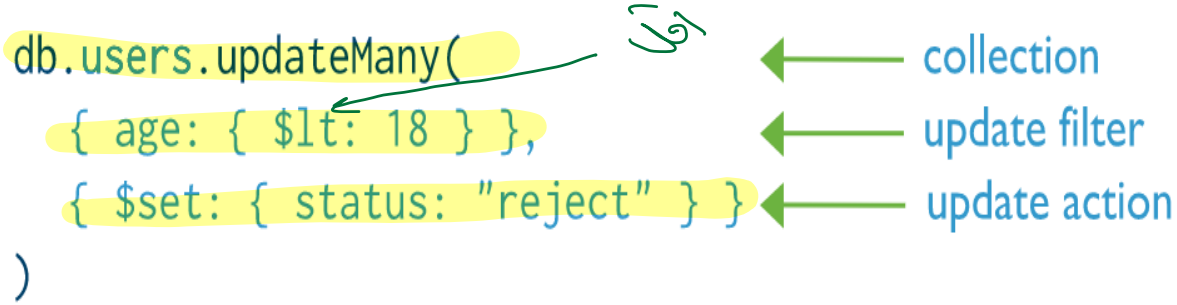


UPDATE OPERATIONS

Update operations modify existing documents in a collection. MongoDB provides the following methods to update documents of a collection:

- db.collection.updateOne()
- db.collection.updateMany()
- Db.collection.replaceOne()

You can specify criteria, or filters, that identify the documents to update. These filters use the same syntax as read operations.



UPDATE OPERATIONS

With **replaceOne()** you can only replace the entire document, while **updateOne()** allows for updating fields.

Since replaceOne() replaces the entire document - fields in the old document not contained in the new will be lost.

With updateOne() new fields can be added without losing the fields in the old document.

DELETE OPERATIONS

Delete operations remove documents from a collection. MongoDB provides the following methods to delete documents of a collection:

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

In MongoDB, delete operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

You can specify criteria, or filters, that identify the documents to remove. These filters use the same syntax as read operations.

```
db.users.deleteMany(          <-- collection
   { status: "reject" }       <-- delete filter
)
```

MONGODB COMPASS AND ATLAS

The GUI for MongoDB.

Compass is an interactive tool for querying, optimizing, and analyzing your MongoDB data in a visual environment.

Compass is free to use and source available, and can be run on macOS, Windows, and Linux.

Download Compass

We will get to know MongoDB Atlas in a lab this Tuesday and Wednesday in sha’ Allah, stay tuned.

OO DATABASE MANAGEMENT SYSTEMS

What is an Object-Oriented Database? A database model that integrates **object-oriented programming** principles.

- Stores data as **objects** instead of traditional tables, similar to how data is represented in object-oriented programming.

Key Characteristics

- **Objects**: Data is stored as complex objects with attributes and methods.
- **Classes & Inheritance**: Supports **class hierarchies** and **inheritance** to reuse structures.
- **Encapsulation**: Combines data and behavior, keeping them together as in programming objects.
- **Persistence**: Objects remain in the database, even after the application ends.

OODBMS HISTORY

The concept of **Object-Oriented Database Management Systems (OODBMS)** emerged in the 1980s, driven by the need for databases that could handle complex data types beyond the capabilities of traditional **Relational Database Management Systems (RDBMS)**.

Unlike RDBMS, which organizes data in rows and columns, OODBMS was designed to store data as **objects**—mirroring the structures used in **Object-Oriented Programming (OOP)**.

PostgreSQL and Its OO Roots:

- **PostgreSQL**, originally developed at the University of California, Berkeley, in the late 1980s, began as an **object-relational database**.
- Initially named **Postgres** (a successor to the Ingres project), it was designed to support complex data structures and allow users to define their own data types and relationships.
- Postgres was among the early databases to offer **object-oriented features**, such as inheritance and support for user-defined types, blurring the line between traditional

RDBMS VS OODBMS: EXAMPLE

Customer {

```
    customerID: String
    name: String
    address: String
    orders: List<Order>
```

}

CustomerID	Name	Address
C001	John	USA
C002	Emma	Canada

Order {

```
    orderID: String
    date: Date
    amount: Float
    customer: Customer
```

}

OrderID	Date	Amount	CustomerID (FK)
O1001	2023-01-01	250.00	C001
O1002	2023-02-01	150.00	C002
O1003	2023-03-05	300.00	C001

RDBMS VS OODBMS (2)

Aspect	Object-Oriented Database (OODBMS)	Relational Database (RDBMS)
Data Representation	Objects and classes, mirroring object-oriented design	Tables with rows and columns
Relationships	Direct object references (e.g., list of Order in Customer)	Foreign keys and JOIN operations
Schema Flexibility	Flexible, allows inheritance and polymorphism	Fixed schema, modifications require altering table design