



TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TPHCM

Khoa: Công nghệ thông tin

Môn: Hệ quản trị cơ sở dữ liệu

---

# BÁO CÁO ĐỒ ÁN

## TÌM HIỂU MONGODB

---

- Nhóm thực hiện
  - Nhóm 21
- Giảng viên hướng dẫn:
  - Phạm Thị Bạch Huệ

## BẢNG THÔNG TIN THÀNH VIÊN

Nhóm: 21

Họ và tên	MSSV	Email
Nguyễn Thị Ngọc Hân (Nhóm trưởng)	1712415	<a href="mailto:1712415@student.hcmus.edu.vn">1712415@student.hcmus.edu.vn</a>
Trịnh Đức Thanh	1712769	<a href="mailto:1712769@student.hcmus.edu.vn">1712769@student.hcmus.edu.vn</a>
Dương Khánh Vi	1712899	<a href="mailto:1712899@student.hcmus.edu.vn">1712899@student.hcmus.edu.vn</a>

## BẢNG PHÂN CÔNG CÔNG VIỆC VÀ ĐÁNH GIÁ

Người thực hiện	Công việc thực hiện	Mức độ hoàn thành	Đánh giá của nhóm
Nguyễn Thị Ngọc Hân – 1712415	Tìm các đặc điểm, tính năng đặc trưng.	100%	10/10
	Khai thác ngôn ngữ truy vấn.	100%	10/10
	Khai thác các tính năng cung cấp.	100%	10/10
Dương Khánh Vi - 1712899	Cách triển khai/cài đặt MongoDB.	100%	10/10
	Khai thác các tính năng cung cấp.	100%	10/10
	Cài đặt một CSDL minh họa.	100%	10/10
Trịnh Đức Thanh - 1712769	Mô hình dữ liệu.	100%	10/10
	Đánh giá các tính năng.	100%	10/10
	Đánh báo cáo	100%	10/10

## MỤC LỤC

A-	YÊU CẦU.....	1
B-	KẾT QUẢ.....	1
I.	Tổng quan về MongoDB .....	1
II.	Đặc điểm, tính năng đặc trưng .....	2
1.	Tính năng đặc trưng .....	2
2.	Một số đặc điểm của MongoDB .....	3
III.	Mô hình dữ liệu.....	4
1.	Giới thiệu Data Model MongoDB sử dụng .....	4
2.	Schema Validation .....	5
IV.	Cách triển khai/cài đặt MongoDB .....	8
1.	Tải MongoDB Shell .....	8
2.	Tải MongoDB Compass.....	14
V.	Khai thác ngôn ngữ truy vấn, các tính năng cung cấp của MongoDB .....	15
1.	Các kiểu dữ liệu trong mongoDB .....	15
2.	Một số câu lệnh dùng trong MongoDB so với SQL.....	17
3.	Chi tiết các lệnh Thêm/Xóa/Sửa/Truy vấn.....	19
4.	Function ( <i>chỉ có trong MongoDB Stitch</i> ).....	27
5.	Trigger ( <i>chỉ có trong MongoDB Stitch</i> ) .....	42
6.	Một số phương thức có sẵn trong MongoDB .....	43
7.	Chi tiết các hàm chính/hàm bổ trợ/toán tử có trong MongoDB .....	44
8.	Chỉ mục.....	66
9.	Sharding.....	70
10.	Map Reduce .....	72
11.	Replication .....	72
12.	Export/Import dữ liệu .....	77
13.	Điều khiển truy xuất đồng thời .....	88
14.	Xử lý giao tác.....	93
15.	Truy cập hệ thống/quản lý tài khoản .....	95
16.	Tạo Role, phân quyền .....	101



17.	Lưu trữ, phục hồi dữ liệu .....	103
18.	Cơ chế bảo mật .....	108
VI.	Cài đặt một CSDL minh họa .....	110
1.	Cài đặt CSDL bằng MongoDB Compass.....	110
2.	Cài đặt CSDL bằng MongoDB Shell .....	113
VII.	Đánh giá các tính năng của MongoDB .....	124
1.	Các tính năng hỗ trợ.....	124
2.	Ưu điểm và khuyết điểm .....	125
3.	So sánh MongoDB và SQL Server.....	128
C-	TÀI LIỆU THAM KHẢO .....	130

## A-YÊU CẦU

Sinh viên hãy khai thác HQT MongoDB theo các tiêu chí sau:

- Đặc điểm, tính năng đặc trưng (phù hợp cho việc tổ chức và lưu trữ dữ liệu dạng gì).
- Mô hình dữ liệu được sử dụng.
- Cách triển khai/ cài đặt MongoDB.
- Khai thác: ngôn ngữ truy vấn, các tính năng cung cấp (với vai trò là một HQT CSDL như: xử lý giao tác (đảm bảo tính ACID), có thể sử dụng trigger, cơ chế điều khiển đồng thời, cơ chế phục hồi dữ liệu, cơ chế bảo mật).
- Cài đặt một CSDL minh họa.
- Đánh giá các tính năng của MongoDB (có hỗ trợ tính năng gì, ưu khuyết điểm).

## B-KẾT QUẢ

### I. Tổng quan về MongoDB

MongoDB là một mã nguồn mở, dùng để thiết kế cơ sở dữ liệu để dễ dàng phát triển và mở rộng, cung cấp hiệu suất cao, tính sẵn sàng cao. Cấu trúc của Mongodbs bao gồm các cặp "thuộc tính" – "giá trị", MongoDB document tương tự như đối tượng JSON. Giá trị của thuộc tính bao gồm các thuộc tính giá trị cụ thể (số, chữ, ...), document, mảng, mảng các đối tượng.

Các tính năng chính - Hiệu suất cao: Hỗ trợ nhúng dữ liệu dạng mô hình dữ liệu giúp giảm thiểu hoạt động của server, Truy vấn dữ liệu sử dụng chỉ mục giúp tối ưu tốc độ truy vấn.

Dễ dàng tăng tính mở rộng: chế sharding tự động (tự động phân vùng dữ liệu trên máy chủ, động bộ hóa dữ liệu tốt MongoDB chạy trên hầu hết các nền tảng và hỗ trợ kiến trúc 64-bit để sử dụng sản xuất và cả hai 64-bit và 32-bit kiến trúc để thử nghiệm.

❖ Các hệ điều hành hỗ trợ:

	<b>MongoDB</b>	<b>MongoDB Enterprise</b>
Amazon Linux	Hỗ trợ	Hỗ trợ
Debian 7.1	Hỗ trợ	Hỗ trợ
RedHat/Cent OS 6.2+	Hỗ trợ	Hỗ trợ
SUSE 11	Hỗ trợ	Hỗ trợ
Ubuntu LTS 12.04	Hỗ trợ	Hỗ trợ

Ubuntu LTS 14.04	Hỗ trợ	Hỗ trợ
Windows Server 2012	Hỗ trợ	Hỗ trợ
Mac OSX 10.6+	Hỗ trợ	
RedHat/Cent OS 5.5+	Hỗ trợ	
RedHat/Cent OS 5.7+	Hỗ trợ	Hỗ trợ
Windows Server 2008 R2	Hỗ trợ	Hỗ trợ

Thuật ngữ được sử dụng trong MongoDB với thuật ngữ được sử dụng trong cơ sở dữ liệu SQL có một số sự khác biệt, dưới đây là bảng so sánh:

<b>SQL</b>	<b>MongoDB</b>
Table	Collection
Row	Document
Column	Field
Primary key	ObjectId
Index	Index
View	View
Nested table or object	Embedded document
Array	Array

## II. Đặc điểm, tính năng đặc trưng

### 1. Tính năng đặc trưng

- Ít Schema hơn: MongoDB là một cơ sở dữ liệu dựa trên Document, trong đó một Collection giữ các Document khác nhau. Số trường, nội dung và kích cỡ của Document này có thể khác với Document khác.
- Cấu trúc của một đối tượng là rõ ràng.
- Không có các Join phức tạp.
- Khả năng truy vấn sâu hơn. MongoDB hỗ trợ các truy vấn động trên các Document bởi sử dụng một ngôn ngữ truy vấn dựa trên Document mà mạnh mẽ như SQL.
- MongoDB dễ dàng để mở rộng.
- Việc chuyển đổi/ánh xạ của các đối tượng ứng dụng đến các đối tượng cơ sở dữ liệu là không cần thiết.
- Sử dụng bộ nhớ nội tại để lưu giữ phần công việc, giúp truy cập dữ liệu nhanh hơn.

- MongoDB sinh ra để tăng tốc độ truy xuất dữ liệu, phù hợp cho các ứng dụng cần tốc độ phản hồi nhanh(realtime như facebook chẳng hạn). Còn các tác nghiệp cần tính toàn vẹn dữ liệu(trong banking) thì Nosql sẽ ko bao giờ là 1 giải pháp cả mà người ta sẽ dùng Mysql.
- MongoDB không có tính ràng buộc, một điều tồi tệ trong Database vì vậy sẽ rất cần sự cẩn thận khi thao tác trên các collection có quan hệ dữ liệu với nhau.
- MongoDB đẩy trách nhiệm thao tác Database cho tầng ứng dụng nên sẽ tồn tài nguyên(tài nguyên bây giờ còn là vấn đề quá lớn nữa).
- MongoDb có thể mở rộng theo chiều ngang (scale out) phương pháp tăng cường khả năng lưu trữ và xử lý là dùng nhiều máy tính phân tán. MongoDB còn có thể mở rộng theo chiều dọc (scale up) tăng cấu hình server.
- MongoDB có thể dùng nhiều máy tính phân tán để lưu trữ dữ liệu nên chi phí sẽ rẻ hơn MySQL. MySQL sử dụng những máy chủ hàng khủng, độc quyền nên sẽ đắt đỏ hơn.

## 2. Một số đặc điểm của MongoDB

Thuộc tính	Đặc trưng
Lưu trữ hướng văn bản	Văn bản theo phong cách JSON với những lược đồ động đơn giản
Hỗ trợ đầy đủ chỉ mục	Chỉ mục trên bất kỳ các thuộc tính
Tính sao lặp và tính sẵn sàng cao	Mở rộng
Auto-sharding	Mở rộng theo chiều ngang mà không ảnh hưởng đến chức năng
Truy vấn	Đa dạng truy vấn dựa trên văn bản
GridFS	Lưu trữ file với bất kỳ kích cỡ mà không làm phức tạp ngăn xếp
Hỗ trợ thương mại	Hỗ trợ doanh nghiệp, đào tạo, tư vấn khái niệm cơ bản của MongoDB

❖ Các điểm lưu ý khi lựa chọn MongoDB:

- Nếu ứng dụng của có tính chất INSERT cao, bởi vì mặc định MongoDB có sẵn cơ chế ghi với tốc độ cao và an toàn.
- Ứng dụng của có tính chất INSERT cao, bởi vì mặc định MongoDB có sẵn cơ chế ghi với tốc độ cao và an toàn.
- Ứng dụng ở dạng thời gian thực nhiều, nghĩa là nhiều người thao tác với ứng dụng. Nếu trong quá trình load bị lỗi tại một điểm nào đó thì nó sẽ bỏ qua phần đó nên sẽ an toàn.

- Ứng dụng bạn có nhiều dữ liệu quá. Ví dụ, giả sử web có đến 10 triệu records thì đó là cơn ác mộng với MySQL. Bởi vì MongoDB có khả năng tìm kiếm thông tin liên quan cũng khá nhanh nên trường hợp này nên dùng nó.
- Máy chủ không có hệ quản trị CSDL, trường hợp này thường bạn sẽ sử dụng SQL LIFE hoặc là MongoDB.

### III. Mô hình dữ liệu

#### 1. Giới thiệu Data Model MongoDB sử dụng

MongoDB cung cấp mô hình dữ liệu Document store – mô hình dữ liệu chính. Ngoài ra, còn có Search engine – mô hình dữ liệu thứ cấp.

- Document: Trong mã ứng dụng, dữ liệu thường được biểu diễn dưới dạng một đối tượng hoặc văn bản dạng JSON (JavaScript Object Notations) vì đây là mô hình dữ liệu hiệu quả và trực quan cho các nhà phát triển. Document database giúp nhà phát triển dễ dàng lưu trữ và truy vấn dữ liệu trong một cơ sở dữ liệu bằng cách sử dụng cùng một định dạng mô hình văn bản họ sử dụng trong mã ứng dụng của mình. Tính chất linh hoạt, bán cấu trúc và phân cấp của các văn bản và cơ sở dữ liệu văn bản cho phép chúng phát triển phù hợp với yêu cầu của ứng dụng. Mô hình văn bản phát huy hiệu quả với danh mục, hồ sơ người dùng và hệ thống quản lý nội dung, nơi từng văn bản là duy nhất và phát triển theo thời gian. MongoDB là document database phổ biến cung cấp các API mạnh mẽ và trực quan để phát triển linh hoạt và lặp lại.

##### 1.1. Flexible Schema

Hệ quản trị cơ sở dữ liệu MongoDB là một mô hình dữ liệu linh hoạt. Không giống như các cơ sở dữ liệu SQL, phải xác định và triển khai sơ đồ của một bảng trước khi chèn dữ liệu, Collection trong MongoDB rất linh hoạt, không nhất thiết phải cho biết những thuộc tính có trong Collection, khi nào cần bổ sung sau vẫn được. Sự linh hoạt này tạo điều kiện cho việc hướng tới một cách lưu trữ như một thực thể hay một đối tượng. Mỗi Document có phù hợp với các trường dữ liệu của Collection chứa nó, ngay cả khi thay đổi gần như toàn bộ dữ liệu.

Tuy nhiên, trong thực tế, các Document trong một Collection chung một cấu trúc. Khi thiết kế cơ sở dữ liệu cần chú ý về cân bằng nhu cầu của các ứng dụng (tức là có nên tạo nhiều Collection lồng nhau), các đặc tính hiệu suất khi thực hiện truy vấn, và các mô hình dữ liệu. Khi thiết kế các mô hình dữ liệu, luôn luôn xem xét việc sử dụng các dữ liệu (ví dụ như truy vấn, cập nhật, và xử lý các dữ liệu) cũng như các cấu trúc vốn có của bản thân dữ liệu.

##### 1.2. Document Structure

###### a. Embedded Data Model

Với MongoDB, bạn có thể nhúng các dữ liệu liên quan vào một Structure hoặc Document, hay còn được gọi là mô hình "denormalized". Các mô hình

denormalized này cho phép các ứng dụng lấy và thao tác dữ liệu liên quan trong một database operation.

Sử dụng nhúng mô hình dữ liệu khi: Cơ sở dữ liệu có mối quan hệ giữa các thực thể (One-to-One và One-to-Many).

Trong MongoDB, nhiều trường hợp mô hình dữ liệu denormalized là tối ưu.

b. References

References lưu trữ các mối quan hệ giữa dữ liệu bằng cách chuyển các liên kết hoặc tham chiếu từ 1 document sang document khác. Nhìn rộng ra, ta có thể thấy đây là các mô hình dữ liệu normalized

**1.3. Atomicity of Write Operation**

a. Single Document Atomicity

Trong MongoDB, một thao tác ghi là một atomic ở cấp độ của một document.

Khi một thao tác ghi đơn (e.g. db.collection.updateMany()) sửa đổi nhiều tài liệu, việc sửa đổi từng tài liệu là atomic nhưng toàn bộ hoạt động khác không phải atomic.

b. Multi-Document Transactions

Đối với các trường hợp yêu cầu tính atomicity của việc đọc và ghi vào nhiều document (trong một hoặc nhiều collection), MongoDB hỗ trợ các giao dịch đa tài liệu (multi-document transactions) khác nhau trên các phiên bản khác nhau.

**2. Schema Validation**

**2.1. Data Model Concepts**

a. Thiết kế Data Model

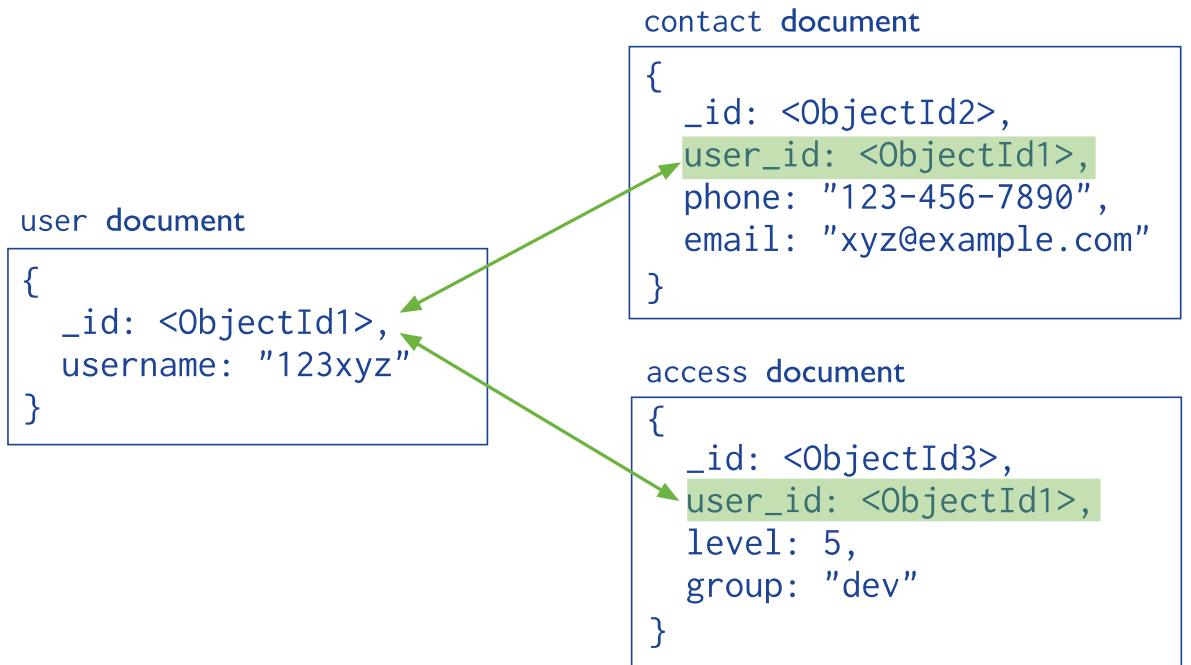
- ❖ Embedded Data Model

Ví dụ về mô hình denormalized:

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
    phone: "123-456-7890",
    email: "xyz@example.com"
  },
  access: {
    level: 5,
    group: "dev"
  }
}
```

❖ Chuẩn hóa Data Model.

Mô hình chuẩn hóa dữ liệu (normalized data model) được mô tả bằng cách sử dụng tham chiếu giữa các Document với nhau.



Để join collections, MongoDB cung cấp các giai đoạn tổng hợp:

- \$lookup
- \$graphLookup

MongoDB cũng cung cấp tham chiếu để join dữ liệu trên các collection.

b. *Operational Factors and Data Models*

Mô hình hóa dữ liệu ứng dụng cho MongoDB phụ thuộc vào dữ liệu, cũng như các đặc điểm của MongoDB.

❖ Document Growth.

Một số cập nhật các Document có thể tăng kích thước của văn bản. Những cập nhật này bao gồm đẩy yếu tố để một mảng (tức là \$push) và thêm các trường mới vào một tài liệu.

Khi sử dụng công cụ lưu trữ MMAPv1, Document Growth có thể được xem xét cho một Data Model của bạn. Đối với MMAPv1, nếu kích thước dữ liệu vượt quá không gian được chia cho Document đó, MongoDB sẽ di dời các Document trên ổ đĩa.

❖ Indexes.

Sử dụng chỉ số để cải thiện hiệu suất cho các truy vấn phổ biến. Xây dựng các chỉ số trên các thuộc tính mà xuất hiện thường xuyên trong các truy vấn và cho tất cả các yêu cầu đó trả lại kết quả được sắp xếp. MongoDB tự động tạo ra một chỉ số duy nhất trên các thuộc tính là \_id.

Khi tạo Indexes cần chú ý:

- Mỗi Indexes yêu cầu tối thiểu 8KB dữ liệu lưu trữ.
- Thêm một Indexes có một số tác động tiêu cực cho các hoạt động ghi. Vì mỗi khi thêm dữ liệu phải cập nhật chỉ số.
- Cần chú ý đến việc thiết lập kích thước cho từng những thuộc tính vì Indexes sẽ chiếm không gian ổ đĩa và ô nhớ.

## 2.2. ***Data Model Examples and Patterns***

a. *Mô hình quan hệ giữa các Document*

- Mô hình One-to-One quan hệ với nhung Document.
- Mô hình One-to-Many quan hệ với nhung Document.
- Mô hình One-to-Many quan hệ với tham chiếu giữa các Document.

b. *Mô hình cấu trúc cây*

- Mô hình cấu trúc cây với tham chiếu Parent.
- Mô hình cấu trúc cây với tham chiếu Child.
- Mô hình cấu trúc cây với Array of Ancestors.
- Mô hình cấu trúc cây với đường dẫn Materialized.
- Mô hình cấu trúc cây với Nested Set.

c. *Mô hình bối cảnh ứng dụng thực tế*

- Mô hình dữ liệu cho các Atomic Operations.

- Mô hình dữ liệu để hỗ trợ tìm kiếm từ khóa.
- Mô hình dữ liệu tiền tệ.
- Mô hình dữ liệu thời gian.

### **2.3. Data Model References**

- ❖ Format: gồm các trường (Field)

\$ref: giữ tên của bộ sưu tập nơi tài liệu được tham chiếu cư trú.

\$ id: chứa giá trị của trường \_id trong tài liệu được tham chiếu.

\$ db: Không bắt buộc.

Chỉ một số ứng dụng hỗ trợ tham chiếu \$db.

VD: { "\$ref" : <value>, "\$id" : <value>, "\$db" : <value> }

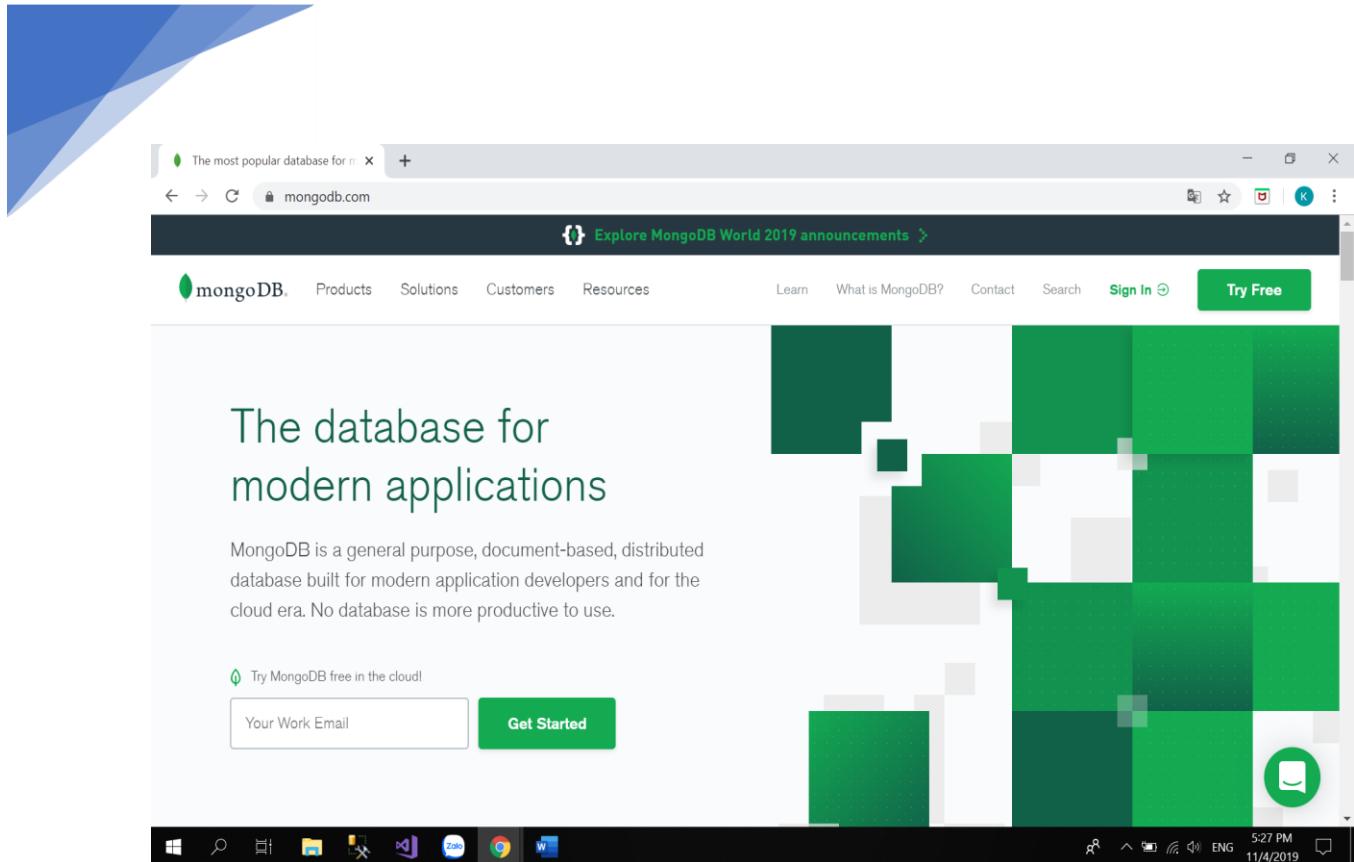
- ❖ Các ngôn ngữ lập trình hỗ trợ:

Ngôn ngữ lập trình	Tình trạng
C#	Hỗ trợ
Java	
Python	
Ruby	
Node.js	
Perl	Không hỗ trợ
C/C++	
PHP	
Haskell	
Scala	

## **IV. Cách triển khai/cài đặt MongoDB**

### **1. Tải MongoDB Shell**

- Bước 1:
- + Vào trang web của nhà phát triển là <https://www.mongodb.com/>



- + Vào mục Product trên thanh công chọn và chọn mục MongoDB Server (hoặc MongoDB Compass để tải phiên bản tạo database bằng giao diện). Ở đây nhóm em sẽ tạo cả 2 phiên bản để dễ dàng khai thác.

A screenshot of the MongoDB Products page. It shows a grid of products categorized into four groups: CLOUD, SOFTWARE, ANALYTICS, and SERVICES. Each group has several items listed with brief descriptions. At the bottom left, there's a "Cloud Migration" section with a "Move to the cloud with minimal downtime" link. The top navigation bar is identical to the homepage, and the bottom status bar shows the URL https://www.mongodb.com/what-is-mongodb, the time 5:28 PM, and the date 11/4/2019.

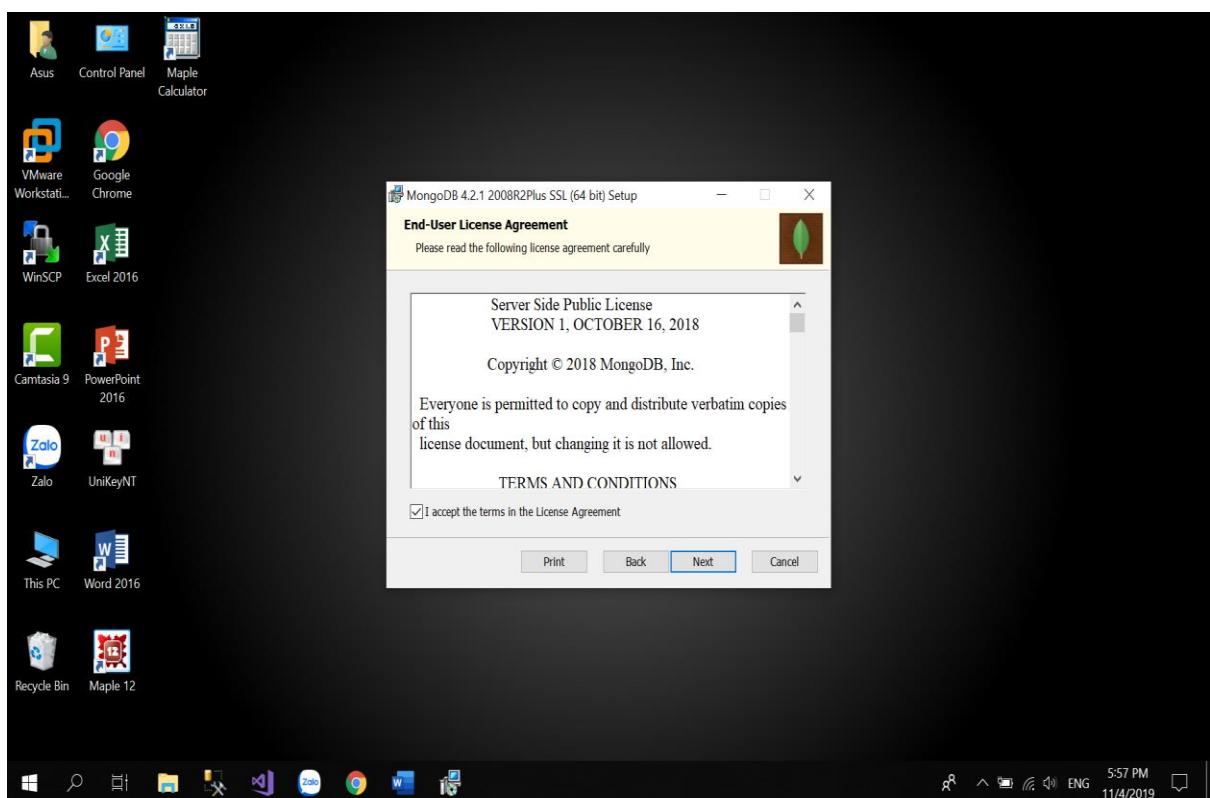
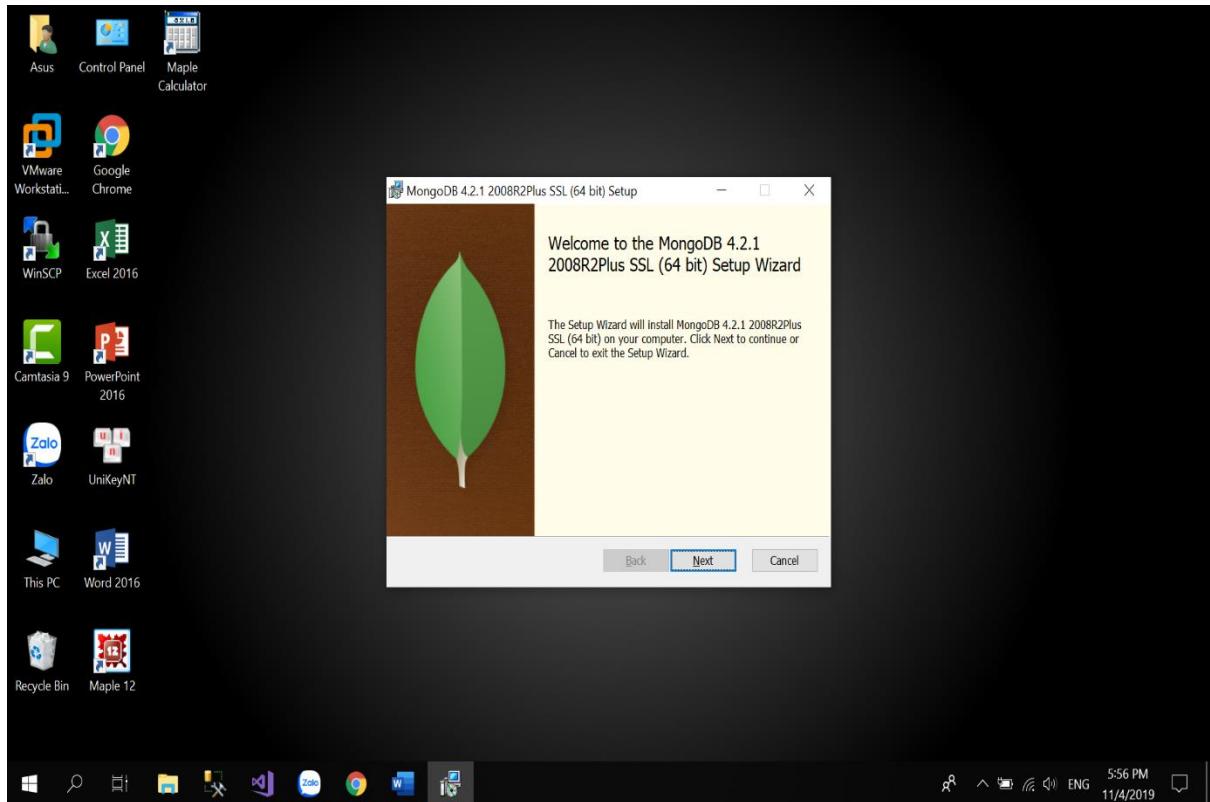
- + Chọn Download MongoDB

The screenshot shows the MongoDB homepage. At the top, there's a navigation bar with links for Products, Solutions, Customers, Resources, Learn, What is MongoDB?, Contact, Search, Sign In, and Try Free. Below the navigation is a large green header with the text "What Is MongoDB?". Underneath, a sub-header reads "MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need". There are two buttons at the bottom: "Start free with MongoDB Atlas" (green) and "Download MongoDB" (white). The URL in the browser address bar is [mongodb.com/what-is-mongodb](https://www.mongodb.com/what-is-mongodb).

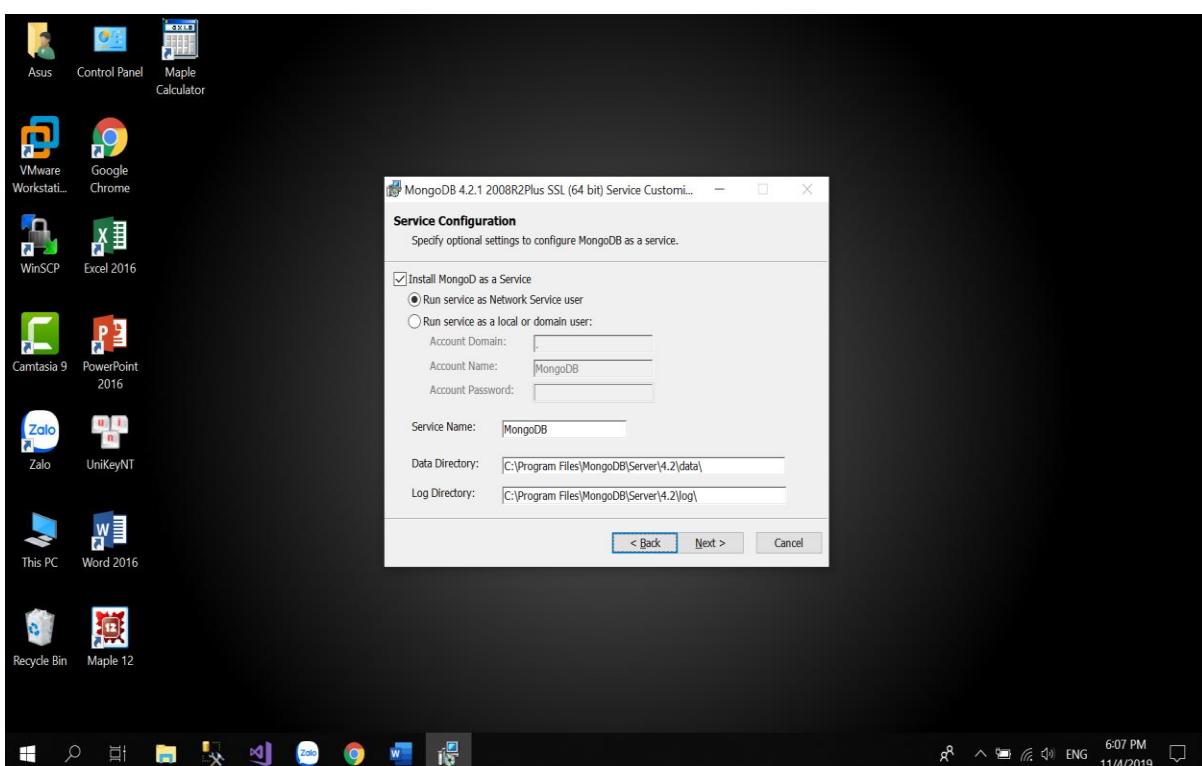
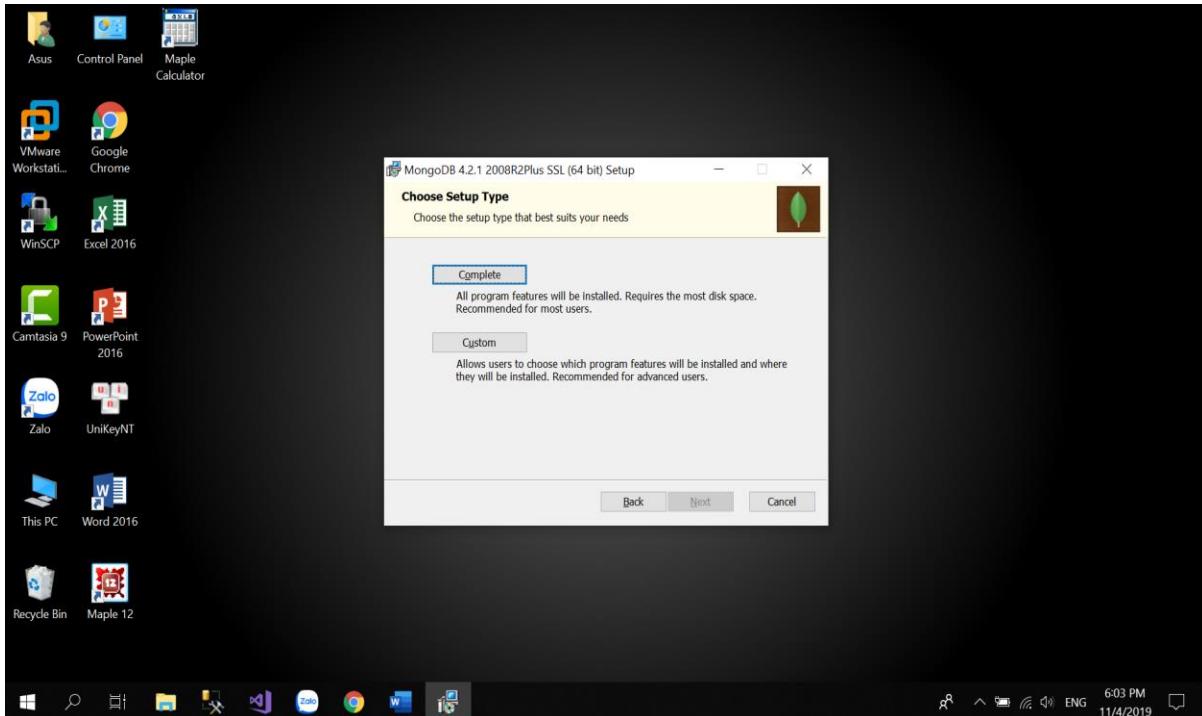
- + Chọn phiên bản Community và cấu hình phù hợp cho máy tính. Do máy tính nhóm sử dụng hệ điều hành Windows x64bit nên nhóm chọn OS là Windows x64 x64, phiên bản 4.2.1 (phiên bản mới nhất) và loại tập tin mà MSI (có thể chọn ZIP). Chọn Download

The screenshot shows the MongoDB Download Center page for the Community Server. It features two main options: "MongoDB Community Server" (highlighted with a green bar) and "MongoDB Enterprise Server". The "Community Server" section includes dropdown menus for "Version" (set to 4.2.1) and "OS" (set to Windows x64 x64). Below these are "Package" options (set to MSI) and a "Download" button. To the right, a sidebar lists links: Release notes, Changelog, All version binaries, Installation instructions, Download source (tgz), and Download source (zip). The URL in the browser address bar is [mongodb.com/download-center/community](https://www.mongodb.com/download-center/community). The taskbar at the bottom shows various application icons.

- Bước 2: Cài đặt vào máy sau khi tải về

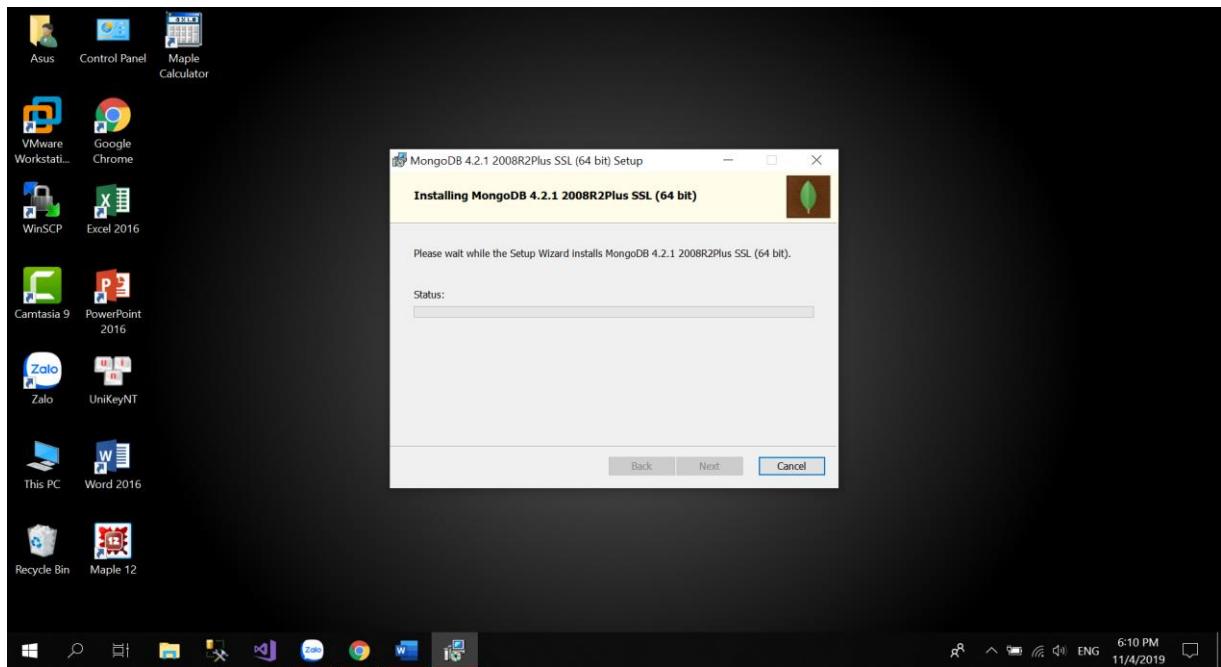


- + Chọn Setup Type là Complete (được khuyến khích cho mọi người dùng). Chọn Complete thì tất cả sẽ được cài đặt một cách tự động (cần có dung lượng lớn). Nếu chọn Custom thì phải chọn chương trình thực thi và vị trí của file.



**Chú ý:** MongoDB sẽ được chạy trên máy tính bằng Command Prompt

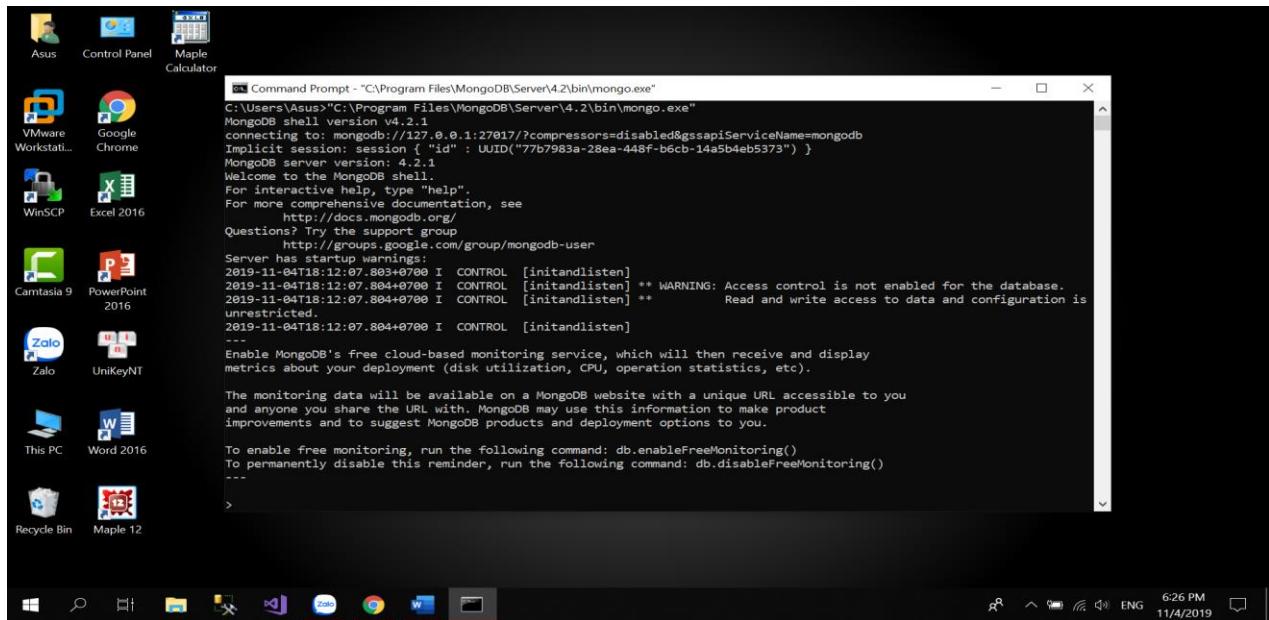
- + Nhấn Next và Install => Ứng dụng sẽ bắt đầu được cài đặt trên máy tính (đợi một vài phút để cài đặt)



- Bước 3: Setup ứng dụng vừa cài đặt trên Command Prompt
- + Vào trang web <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/> để copy đường dẫn
- + Chọn MongoDB Product => MongoDB Server => Installation => Installation MongoDB Community Edition => Install on Windows

The screenshot shows a web browser window with the MongoDB documentation for Windows installation. The URL in the address bar is <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/>. The page content is about installing MongoDB as a Windows Service. It includes instructions to start the MongoDB service upon successful installation and provides two methods to begin using MongoDB: either by opening a Windows Explorer/File Explorer and double-clicking on `mongo.exe`, or by opening a Command Interpreter with administrative privileges and running the command `C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe`. A red box highlights this command in a code block. The browser's address bar shows the URL, and the taskbar at the bottom includes icons for File Explorer, Task View, Start, and system status.

- + Mở Command Prompt và paste đường dẫn đã copy



+ MongoDB đã được cài đặt thành công trên máy tính

## 2. Tải MongoDB Compass

MongoDB Compass là phiên bản Visual của MongoDB, được khuyến khích cho mọi người dùng phổ thông vì tính tiện lợi của nó. Với MongoDB Compass, chúng ta có thể tạo CSDL, tạo user, tạo collection,... dễ dàng chỉ bằng thao tác.

Các bước làm cũng giống như tải MongoDB Shell, vào trang web

<https://www.mongodb.com/>, chọn MongoDB Compass, tải về phiên bản phù hợp với cấu hình của máy.

Version	Platforms	
1.19.12 (Stable)	Windows 64-bit (7+) (MSI)	Documentation

**Download**

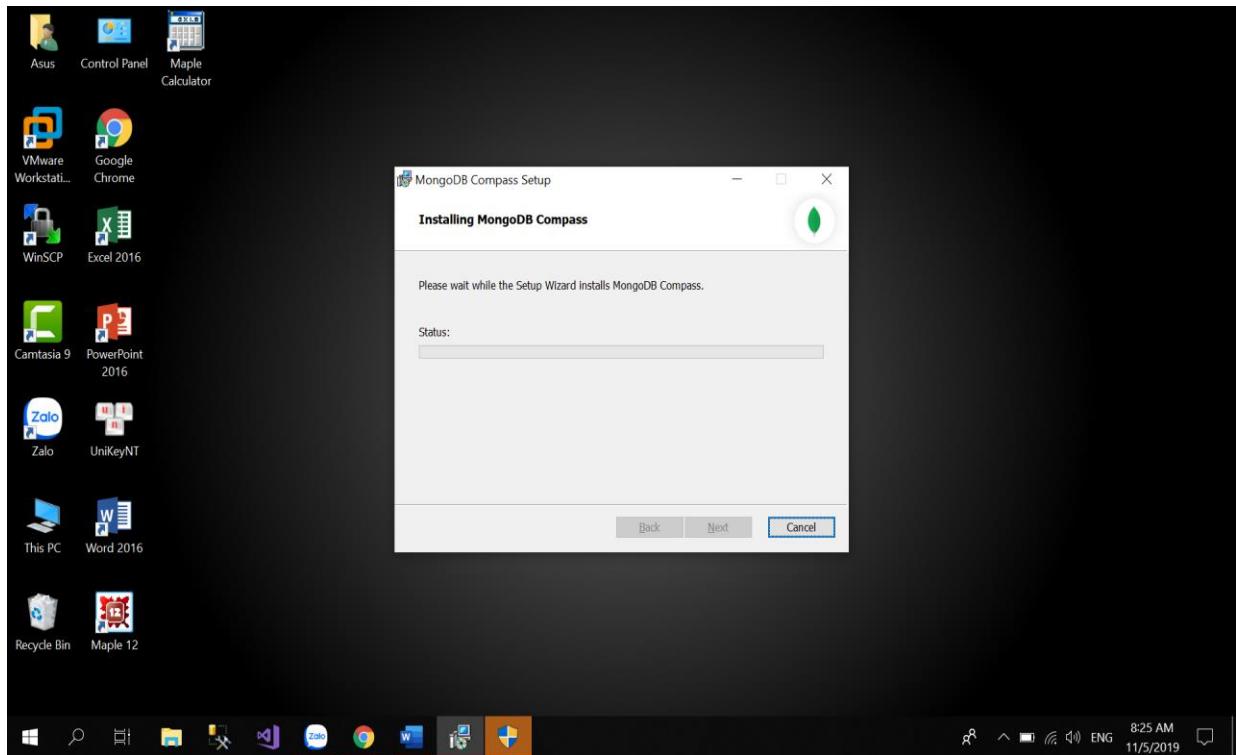
**Compass**  
The full version of MongoDB Compass, with all features and capabilities.

**Readonly Edition**  
This version is limited strictly to read operations, with all write and delete capabilities removed.

**Isolated Edition**  
This version disables all network connections except the connection to the MongoDB instance.

**Community Edition**  
This version is distributed with Community Server downloads and includes a subset of the features of Compass.

Sau khi tải, vào thư mục vừa tải và tiến hành cài đặt (nhấn chọn Next 3 lần và Install).  
Install vào máy có thể mất một vài phút.



## V. Khai thác ngôn ngữ truy vấn, các tính năng cung cấp của MongoDB

### ✚ Khai thác ngôn ngữ truy vấn

#### 1. Các kiểu dữ liệu trong mongoDB

Type	Number	Alias
Double	1	"double"
String	2	"string"
Object	3	"object"
Array	4	"array"
Binary data	5	"binData"
Undefined	6	"undefined"
ObjectId	7	"objectId"
Boolean	8	"bool"

Date	9	"date"
Null	10	"null"
Regular Expression	11	"regex"
DBPointer	12	"dbPointer"
JavaScript	13	"javascript"
Symbol	14	"symbol"
JavaScript (with scope)	15	"javascriptWithScope"
32-bit integer	16	"int"
Timestamp	17	"timestamp"
64-bit integer	18	"long"
Decimal128	19	"decimal"
Min key	-1	"minKey"
Max key	127	"maxKey"

- **Chuỗi:** Đây là kiểu dữ liệu được sử dụng phổ biến nhất để lưu giữ dữ liệu. Chuỗi trong MongoDB phải là UTF-8 hợp lệ.
- **Số nguyên:** Kiểu dữ liệu này được sử dụng để lưu một giá trị số. Số nguyên có thể là 32 bit hoặc 64 bit phụ thuộc vào Server của bạn.
- **Boolean:** Kiểu dữ liệu này được sử dụng để lưu giữ một giá trị Boolean (true/false).
- **Double:** Kiểu dữ liệu này được sử dụng để lưu các giá trị số thực dấu chấm động.
- **Min/ Max keys:** Kiểu dữ liệu này được sử dụng để so sánh một giá trị với các phần tử BSON thấp nhất và cao nhất.
- **Mảng:** Kiểu dữ liệu này được sử dụng để lưu giữ các mảng hoặc danh sách hoặc nhiều giá trị vào trong một key.
- **Timestamp:** Đánh dấu thời điểm một Document được sửa đổi hoặc được thêm vào.
- **Object:** Kiểu dữ liệu này được sử dụng cho các Document được nhúng vào.
- **Null:** Kiểu dữ liệu này được sử dụng để lưu một giá trị Null.

- **Symbol:** Kiểu dữ liệu này được sử dụng giống như một chuỗi
- **Date :** Kiểu dữ liệu này được sử dụng để lưu giữ date và time hiện tại trong định dạng UNIX time.
- **Object ID:** Kiểu dữ liệu này được sử dụng để lưu giữ ID của Document.
- **Binary data:** Kiểu dữ liệu này được sử dụng để lưu giữ dữ liệu nhị phân.
- **Code:** Kiểu dữ liệu này được sử dụng để lưu giữ JavaScrip code vào trong Document.
- **Regular expression:** Kiểu dữ liệu này được sử dụng để lưu giữ Regular Expression.

Ví dụ : Dữ liệu của collection bao gồm 2 documents MinKey và MaxKey: { "\_id" : 1, x : { "\$minKey" : 1 } } { "\_id" : 2, y : { "\$maxKey" : 1 } }

Câu lệnh truy vấn sau sẽ cho kết quả là một documents có \_id: 1:

```
db.data.find( { x: { $type: "minKey" } } )
```

hoặc có thể viết

```
db.data.find( { x: { $type: "-1" } } )
```

## 2. Một số câu lệnh dùng trong MongoDB so với SQL

Câu lệnh	SQL	MongoDB
Xem database trong hệ thống	(không có lệnh)	Show dbs
Create database	CREATE DATABASE TÊN CSDL	USE TÊN CSDL
Drop database	DROP TÊN CSDL	db.dropDatabase()
Create table	CREATE TABLE ten_bang( kieu_du_lieu cot1, kieu_du_lieu cot2, ..... kieu_du_lieu cotN,	db.createCollection('Tên')

	PRIMARY KEY( mot hoac nhieu cot );	
Xem collection (table)	Không có	show collections
Drop table	DROP TABLE people	db.people.drop()
Insert records into tables	INSERT INTO people(user_id, age, status) VALUES ("bcd001", 45, "A")	db.people.insertOne( { user_id: "bcd001", age: 45, status: "A" } )
Insert records into tables (một mảng nhiều ký tự)	Không thêm được	db.people.insertOne( { user_id: "bcd001", age: 45, status: "A", skills: {"PHP", "C++", "Ruby"} } )
Select	SELECT * FROM users	db.people.find() db.people.find().pretty()
	SELECT * FROM users WHERE age=33	db.users.find({age:33})
	SELECT danh sach_cot FROM ten_bang [WHERE dieu_kien] [ORDER BY cot1, cot2, .. cotN] [ASC   DESC];	db.collectionName.find().sort({KEY:1}) Trong đó, giá trị 1 là sắp xếp tăng dần và -1 là sắp xếp giảm dần db.users.find({age:33}).sort({name:1})
	SELECT * FROM users WHERE age=33 ORDER BY name	
	SELECT * FROM users WHERE age>33	db.users.find({age:{\$gt:33}})
	SELECT * FROM users WHERE age!=33	db.users.find({age:{\$ne:33}})
	SELECT * FROM people WHERE status = "A" OR age = 50	db.people.find( { \$or: [ { status: "A" } , { age: 50 } ] } )
	SELECT * FROM users WHERE name LIKE "Joe%"	db.users.find({name:/^Joe/})

	SELECT * FROM people WHERE user_id like "%bc%"	db.people.find( { user_id: /bc/ } ) db.people.find( { user_id: { \$regex: /bc/ } } )
	SELECT * FROM users WHERE age>33 AND age<=40	db.users.find({'age':{\$gt:33,\$lte:40}})
	SELECT * FROM users LIMIT 10 SKIP 20	db.users.find().limit(10).skip(20)
	SELECT a,b FROM users	db.people.find( { }, { a: 1, b: 1 } )
	SELECT COUNT(user_id) FROM people	db.people.count( { user_id: { \$exists: true } } ) db.people.find( { user_id: { \$exists: true } } ).count()
	SELECT DISTINCT last_name FROM users	db.users.distinct('last_name')
	SELECT COUNT(*y) FROM users	db.users.count()
	SELECT COUNT(*y) FROM users where AGE > 30	db.users.find({age: {'\$gt': 30}}).count()
	SELECT COUNT(AGE) from users	db.users.find({age: {'\$exists': true}}).count()
Explain	EXPLAIN SELECT * FROM users WHERE z=3	db.users.find({z:3}).explain()

- ❖ **Lưu ý:** Trong MongoDB các câu lệnh db.tên collection.insert({}) cần được viết trên notepad nhưng ta vẫn có cách khác, có thể viết trực tiếp trên Command Prompt:

```
> var Information = {}
> Information.StudentID = "1712777"
1712777
> Information.Name = "NNN"
NNN
> Information.Gender = "Female"
Female
> Information.DateOfBirth = "22/08/1999"
22/08/1999
> Information.email = "1712777@student.hcmus.edu.vn"
1712777@student.hcmus.edu.vn
> Information.skills = [ "Oracle", "MongoDB", "C#", "Python" ]
[ "Oracle", "MongoDB", "C#", "Python" ]
> db.Information.save(Information)
WriteResult({ "nInserted" : 1 })
>
```

### 3. Chi tiết các lệnh Thêm/Xóa/Sửa/Truy vấn

### 3.1. Thao tác thêm document

Các loại Insert	Cú pháp	Chức năng	Ví dụ
db.collection.insert()	db.collection.insert( <document or array of documents>, { writeConcern: <document>, ordered: <boolean> } )	Thêm một hay nhiều document trong collection	db.products.insert( { item: "card", qty: 15 } )
Insert a Single Document	db.collection.insertOne( <document>, { writeConcern: <document> } )	Thêm một thông tin trong document  Nếu Collection chưa tồn tại document, sẽ tạo một document tương ứng.  Nếu document không tạo trường _id, mongod sẽ thêm trường _id vào	try { db.products.insertOne( { item: "card", qty: 15 } ); } catch (e) { print (e); };
Insert Multiple Documents	db.collection.insertMany( [ <document 1> , <document 2> , ... ], { writeConcern: <document>, ordered: <boolean> })	Thêm nhiều document vào collection.  Nếu ordered được cài đặt là false, document được thêm vào sẽ là định dạng không có thứ tự và có thể được sắp xếp lại theo	try { db.products.insertMany([ { item: "card", qty: 15}, {item: "envelope", qty: 20}, { item: "stamps" , qty: 30}]); } catch (e) {

	<u>Note:</u> dòng ordered của hàm có kiểu boolean (1,-1) để chỉ định dữ liệu đó có sắp xếp theo thứ tự hay không. Mặc định là true ( có sắp xếp)	mongodb để tăng hiệu suất.	print (e);}
--	--	----------------------------	-------------

### 3.2. Thao tác Sửa document

Các loại Update	Cú pháp	Chức năng	Ví dụ
db.collection.update()	<pre>db.collection.update(   &lt;query&gt;,   &lt;update&gt;,   {     upsert: &lt;boolean&gt;,     multi: &lt;boolean&gt;,     writeConcern:     &lt;document&gt;,     collation: &lt;document&gt;,     arrayFilters:     [ &lt;filterdocument1&gt;, ... ],     hint: &lt;document string&gt;   })</pre> <p><u>Note:</u> upsert có hai giá trị là true và false. Upset là true có nghĩa là khi tìm trong câu truy vấn không có, hệ thống sẽ tự thêm mới với giá trị là kết quả của câu truy vấn. Ngược lại là false</p> <p>Multi có hai giá trị là true và false. Multi là true thì nó sẽ tiến hành thêm vào nhiều document thỏa điều kiện truy</p>	Cập nhật một hoặc nhiều document theo điều kiện truy vấn	<pre>db.members.update(   {user name: "AAA"},   {\$set: {age: 25}},   {     Upsert: false,     Multi: false   } )</pre>

	vấn. False là không thêm vào nhiều document		
db.collection.updateOne()	<pre>db.collection.updateOne(     &lt;filter&gt;,     &lt;update&gt;,     {         upsert: &lt;boolean&gt;,         writeConcern:             &lt;document&gt;,         collation: &lt;document&gt;,         arrayFilters:             [ &lt;filterdocument1&gt;, ... ],         hint: &lt;document string&gt;         // Available starting in         MongoDB 4.2.1     } )</pre>	Cập nhật một document theo điều kiện truy vấn	<pre>db.restaurant.update One(     { "name" :         "Central Perk Cafe" },     { \$set:         { "violations" : 3 } } )</pre>
db.collection.updateMany()	<pre>db.collection.updateMany(     &lt;filter&gt;,     &lt;update&gt;,     {         upsert: &lt;boolean&gt;,         writeConcern:             &lt;document&gt;,         collation: &lt;document&gt;,         arrayFilters:             [ &lt;filterdocument1&gt;, ... ],         hint: &lt;document string&gt;         // Available starting in         MongoDB 4.2.1     } )</pre>	Cập nhật nhiều document theo điều kiện truy vấn	<pre>db.restaurant.update Many(     { violations: { \$gt:         4 } },     { \$set: { "Review" :         true } } )</pre>

### 3.3. Thao tác xóa document/field

Các loại xóa	Cú pháp	Chức năng	Ví dụ
db.collection.remove() (xóa trường)	<pre>db.collection.remove(   &lt;query&gt;,   { justOne:&lt;boolean&gt;,     writeConcern:&lt;document&gt;,     collation:&lt;document&gt;   }) <u>Note:</u> justOne gồm true và false.Nếu true thì chỉ xóa một trường thỏa điều kiện truy vấn. Ngược lại là false</pre>	Xóa các trường theo điều kiện truy vấn	db.products.remove( { qty: { \$gt: 20 } } )
db.collection.deleteOne()	<pre>db.collection.deleteOne(   &lt;filter&gt;,   {writeConcern:&lt;document&gt;,    collation:&lt;document&gt;   } )</pre>	Xóa một document theo điều kiện truy vấn	db.orders.deleteOne( { "_id" : ObjectId("563237a41a4d68582c2509da") } )
db.collection.deleteMany()	<pre>db.collection.deleteMany(   &lt;filter&gt;,   {     writeConcern:&lt;document&gt;   } )</pre>	Xóa nhiều document theo điều kiện truy vấn	db.orders.deleteMany( { "client" : "Crude Traders Inc." } )

	<pre>         collation:         &lt;document&gt;     } ) </pre>		
--	--	--	--

### 3.4. Thao tác truy vấn

Các loại truy vấn	Cú pháp	Chức năng	Ví dụ
<pre>db.collection.find(query, projection)</pre> <p><u>Note:</u> Để làm cho kết quả truy vấn đẹp và dễ nhìn hơn thì thêm lệnh db.collection.find(query, projection).pretty()</p>	<pre> db.collection.find(     { field1: &lt;value&gt;         , field2: &lt;value&gt; ... }) </pre> <p><u>Note:</u> với value là giá trị 1 hoặc 0. 1 là trường có tồn tại giá trị, 0 là không tồn tại</p>	Tìm kiếm thông tin của document theo điều kiện truy vấn	<pre> db.bios.find(     { name: { first: "Yukihiro",         last: "Matsumoto" } } ) </pre>
<pre>db.collection.findOne(query, projection)</pre>	<pre> db.collection.findOne(     {&lt;query&gt;,         field1: &lt;value&gt;         , field2: &lt;value&gt; ... }) </pre>	Tìm kiếm thông tin của document theo điều kiện truy vấn và trả về kết quả chỉ có tham số đầu vào	<pre> db.bios.findOne(     { contribs: 'OOP' },     { _id: 0, 'name.first': 0, birth: 0 } ) </pre>
<pre>db.collection.findOneAndDelete(filter, options)</pre>	<pre> db.collection.findOneAndDelete(     &lt;filter&gt;,     {         projection:         &lt;document&gt;,         sort: &lt;document&gt;,         maxTimeMS:         &lt;number&gt;,         collation:         &lt;document&gt;     } ) </pre>	Tìm kiếm thông tin của một trường trong document và xóa nó	<p>Có bản ghi: { _id: 6305, name : "A. MacDyver", "assignment" : 5, "points" : 24 }, { _id: 6312, name : "M. Tagnum", "assignment" : 5, "points" : 30 },</p> <p>Lệnh xóa: db.scores.findOneAndDelete( { "name" : "M. Tagnum" } )</p> <p>Kết quả bị xóa: { _id: 6312, name: "M. Tagnum", "assignment" : 5, "points" : 30 }</p>
<pre>db.collection.findOneAndReplace(filter,</pre>	<pre> db.collection.findOneAndReplace(     &lt;filter&gt;, </pre>	Tìm kiếm thông tin của một field trong	Bản ghi: { "_id" : 2231, "team" : "Tactful Mooses", "score" : 23500 },

replacement, options)	<pre> &lt;replacement&gt;, {   projection: &lt;document&gt;,   sort: &lt;document&gt;,   maxTimeMS: &lt;number&gt;,   upsert: &lt;boolean&gt;,   returnNewDocument: &lt;boolean&gt;,   collation: &lt;document&gt; } ) </pre>	document và thay thế bằng điều kiện <replacement>	{ "_id" : 4511, "team" : "Aquatic Ponies", "score" : 19250 }, Lệnh thực hiện: db.scores. findOneAndReplace( { "score" : { \$lt : 20000 } }, { "team" : "Observant Badgers", "score" : 20000 } ) Dòng bị thay thế là: { "_id" : 2512, "team" : "Aquatic Ponies", "score" : 19250 }
db.collection. findOneAndUpdate (filter, update, options)	<pre> db.collection.  findOneAndUpdate(    &lt;filter&gt;,    &lt;update document or  aggregation pipeline&gt;,  {   projection: &lt;document&gt;,   sort: &lt;document&gt;,   maxTimeMS: &lt;number&gt;,   upsert: &lt;boolean&gt;,   returnNewDocument: &lt;boolean&gt;,   collation: &lt;document&gt;,   arrayFilters: [ &lt;filterdocument1&gt;, ... ] } ) </pre>	Tìm kiếm thông tin của một field trong document và cập nhật bằng điều kiện <update>	Bản ghi: { _id: 6322, name : "A. MacDyver", "assignment" : 2, "points" : 14 }, { _id: 6234, name : "R. Stiles", "assignment" : 1, "points" : 10 } Lệnh thực hiện: db.grades.findOneAndUpdate( { "name" : "R. Stiles" }, { \$inc: { "points" : 5 } } ) Dòng bị cập nhật: { _id: 6319, name: "R. Stiles", "assignment" : 2, "points" : 12 }

### **3.5. Câu lệnh truy vấn khác:** db.collection.findAndModify(document) (Tìm kiếm và sửa đổi)

#### a. Định nghĩa:

Lệnh truy vấn này sẽ sửa đổi và trả về một document. Theo mặc định, document trả về không bao gồm sự thay đổi được cập nhật. Để trả về document với sự thay đổi đã được cập nhật, có thể sử dụng tùy chọn mới.

Lệnh truy vấn findAndModify() được định nghĩa theo mẫu:

```
db.collection.findAndModify({  
    query: <document>,  
    sort: <document>,  
    remove: <boolean>,  
    update: <document or aggregation pipeline>, // Changed in MongoDB 4.2  
    new: <boolean>,  
    fields: <document>,  
    upsert: <boolean>,  
    bypassDocumentValidation: <boolean>,  
    writeConcern: <document>,  
    collation: <document>,  
    arrayFilters: [ <filterdocument1>, ... ]  
});
```

#### b. Data trả về:

- Với thao tác xóa, nếu điều kiện của câu truy vấn khớp với document, lệnh findAndModify() sẽ trả về document bị xóa bỏ. Nếu điều kiện không khớp, findAndModify() trả về null
- Với thao tác update, findAndModify() sẽ trả về một trong những kiểu :
  - Nếu tham số đầu vào không được thêm vào hay là false :
    - Document trước sự thay đổi nếu điều kiện truy vấn khớp với document trước đó
    - Trường hợp khác thì trả về null
  - Nếu tham số đầu vào là true :
    - Document thay đổi nếu điều kiện truy vấn trả về trùng khớp
    - Document được thêm vào nếu upsert : true và không có document khớp với điều kiện truy vấn

- o Trường hợp khác trả về null
- c. Cơ chế hoạt động: Upsert and Unique Index

Khi `findAndModify()` có `upsert : true` and trường trong câu truy vấn không là khóa duy nhất, nó có thể chèn một document nhiều lần trong hoàn cảnh nhất định.

Ví dụ: Có câu lệnh sau (Trong document trên không có tên Andy)

```
db.people.findAndModify({  
    query: { name: "Andy" },  
    sort: { rating: 1 },  
    update: { $inc: { score: 1 } },  
    upsert: true  
})
```

Nếu sử dụng `findAndModify()` sẽ chạy câu truy vấn trước mọi sự thay đổi. Do không khóa duy nhất trên trường `name`, hệ thống sẽ chạy lệnh `upsert` (có giá trị là `true` hoặc `false`). Nếu `true`, document sẽ được tự động thêm vào theo tham số `upsert` nếu không thỏa điều kiện truy vấn), tạo ra nhiều document trùng lặp.

Để ngăn chặn sự trùng lặp dữ liệu, nên tạo một khóa duy nhất trên trường `name`. Với khóa duy nhất này, các phương thức sẽ được thực hiện theo thứ tự:

- `findAndModify()` sẽ thêm thành công 1 document mới
- Không hoặc nhiều phương thức `findAndModify()` sẽ cập nhật document được thêm vào mới nhất
- Không hoặc nhiều phương thức `findAndModify()` sẽ không chèn được những tài liệu có cùng tên.

#### 4. Function (chỉ có trong MongoDB Stitch)

##### 4.1. Giới thiệu

Stitch function cho phép xác định và thực thi logic phía máy chủ cho application. Ta có thể gọi các function từ client applications của mình cũng như từ các function khác và trong các biểu thức JSON trong suốt Stitch.

Các function được viết bằng modern JavaScript (ES6+) và thực thi theo cách thức không máy chủ. Khi bạn gọi một function, bạn có thể truy cập động các thành phần của application hiện tại cũng như thông tin về yêu cầu để thực thi function và đăng nhập cho người dùng đã gửi yêu cầu.

###### a. Trường hợp sử dụng:

Bạn có thể sử dụng các function để xử lý low-latency, short-running connection logic và các tương tác phía máy chủ khác. Function đặc biệt hữu ích khi ta muốn làm việc với nhiều dịch vụ, hoạt động linh hoạt dựa trên người dùng hiện tại hoặc trừu tượng hóa các chi tiết triển khai từ các client application.

Stitch cũng sử dụng các function nội bộ cho các webhooks và triggers. Các function mà bạn tạo cho các thành phần này là các regular Stitch function có các đối số cụ thể tùy thuộc vào dịch vụ.

b. Cơ chế hoạt động

Stitch tự động mã hóa các giá trị được trả về từ các function dưới dạng Extended JSON.

Các function không tiếp tục execute sau khi return. Bạn không thể sử dụng các function với các callbacks không đồng bộ hoặc trình nghe sự kiện (event listeners). Nếu bạn cần thực hiện công việc không đồng bộ trong function, hãy sử dụng promise.

c. Ràng buộc

- Thời gian chạy (runtime) của function được giới hạn trong 60 giây.
- Memory usage giới hạn ở 256MB.
- Function hiện không hỗ trợ các tính năng ES6 + sau:
- Module Imports (i.e. import và yêu cầu).
- Các loại đối tượng toàn cầu mới (VD: Bản đồ, WeakMap, Set, Weakset, Symbol, Proxy).
- Các API toán học, số, chuỗi, mảng và đối tượng mới (VD: Array.prototype.fill, Array.prototype.includes, Object.assign).

## 4.2. Các khái niệm

a. Active User

Active user của function là tài khoản người dùng mà Stitch thực thi function. User này cũng là active user cho bất kỳ function hoặc hành động dịch vụ nào khác được gọi từ bên trong function.

Ta có thể tham chiếu động active user của function trong function's source code với context.user cũng như trong các quy tắc hành động dịch vụ với %%user expansion. Stitch giải quyết các tham chiếu này đến đối tượng user của active user.

b. System Functions

System functions là một function trong đó active user là system user thay vì application user. System functions có quyền truy cập đầy đủ vào MongoDB CRUD

và Aggregation APIs và không bị ảnh hưởng bởi bất kỳ rules, roles, hay permissions nào.

Ta có thể định cấu hình function để chạy với tư cách là system user bằng cách bật tùy chọn cấu hình Run As System. Ta có thể xác định xem một function có đang thực thi như một system user trong thời gian chạy hay không bằng cách gọi phương thức context.rastyAsSystem () .

### 4.3. **Configure functions**

Ta có thể tạo và định cấu hình các function trong ứng dụng bất cứ lúc nào từ Stitch UI hoặc bằng cách nhập tệp cấu hình chức năng.

#### a. Stitch UI

- Bước 1: Tạo một function mới

Để xác định function phía máy chủ mới từ Stitch UI:

- + Nhấp vào Function trong điều hướng bên trái.
- + Nhấp vào New Function ở trên cùng bên phải của trang Function.

- Bước 2: Đặt tên cho New Function

Nhập một tên duy nhất, xác định cho function trong Name field. Tên này phải khác biệt với tất cả các function khác trong application.

- Bước 3: Định cấu hình xác thực người dùng

Functions trong Stitch luôn thực thi trong ngữ cảnh của một application user cụ thể hoặc là một system user bỏ qua các rules. Để định cấu hình người dùng thực thi function, hãy chỉ định loại authentication mà Stitch nên sử dụng (Application Authentication, System, User Id, Script).

#### Authentication

This is where you can choose the authentication method for your function.

Application Authentication 

System 

User Id 

Script 

- Bước 4: Cấu hình Logs thực thi chức năng

Theo mặc định, Stitch bao gồm các đối số mà một function nhận được trong mục logs cho mỗi lần execute. Nếu muốn ngăn Stitch logging các đối số, hãy vô hiệu hóa Log Function Arguments.

#### Log Function Arguments

Save and view arguments to this function within [logs](#)



- Bước 5: Chỉ định một biểu thức ủy quyền

Ta có thể ủy quyền động các yêu cầu dựa trên nội dung của từng yêu cầu bằng cách xác định Can Evaluate JSON expression. Stitch đánh giá biểu thức bất cứ khi nào function được gọi. Nếu bạn không chỉ định biểu thức thì Stitch sẽ tự động ủy quyền cho tất cả các yêu cầu đến được xác thực.

Biểu thức có thể mở rộng các biến biểu thức tiêu chuẩn, bao gồm cả yêu cầu %%request và %%user.

**Can Evaluate**

This is a JSON expression that must evaluate to TRUE before the function may run. If this field is blank, it will evaluate to TRUE. This expression is evaluated before service-specific rules.

```

1 %
2 %%request.remoteIPAddress: {
3   "%nin": [
4     "248.88.57.58",
5     "19.241.23.116",
6     "147.64.232.1"
7   ]
8 }
9

```

- Bước 6: Cấu hình cấp độ riêng tư của function

Theo mặc định, bạn có thể gọi một function từ các client application cũng như các function khác trong cùng một application. Ta có thể ngăn các client application nhìn thấy hoặc gọi một function bằng cách đặt Private thành true.

Ta vẫn có thể gọi một function riêng từ JSON expressions và các function khác, bao gồm các incoming webhooks and triggers.

#### Private

If private, this function may be called from incoming webhooks, rules, and other functions defined in the Stitch console. Private functions may not be called from Stitch client application.



- Bước 7: Viết function code

Sau khi đã tạo và cấu hình chức năng mới, ta viết javascript code thực tế chạy khi gọi function. Ta có thể viết code trực tiếp trong Stitch UI bằng cách dùng function editor.

**Chú ý:** Bạn có thể sử dụng hầu hết các tính năng modern JavaScript (ES6 +) trong các function, bao gồm async/await, destructuring và template literals.

Từ trang function's Setting:

- + Nhập vào tab Function Editor.
- + Thêm javascript code vào function. Tối thiểu, code phải gán một function cho exports biến toàn cục. VD:

```
+ exports = function() {
+   return "Hello, world!";
+ };
• Bước 8: Lưu function
```

Sau khi viết function code, bấm Save từ Function Editor hoặc Settings tab. Sau khi lưu function, ta có thể bắt đầu sử dụng ngay lập tức.

#### b. Import/Export

- Bước 1: Export Your Stitch Application

```
| stitch-cli export --app-id=<App ID>
• Bước 2: Thêm một Function Configuration Directory
```

```
| mkdir -p functions/<function name>
• Bước 3: Thêm một Function Configuration File
```

```
{
  "name": "<Unique Function Name>",
  "run_as_authed_user": <boolean>,
  "run_as_user_id": "<Stitch User ID>",
  "run_as_user_id_script_source": "<Stringified Function>",
  "disable_arg_logs": <boolean>,
  "can_evaluate": <JSON Expression>,
  "options": <document>
}
```

- Bước 4: Đặt tên Webhook mới

```
{
  "name": "<Unique Webhook Name>"
}
```

- Bước 5: Configure User Authentication

+ Application authentication :

```
{
  "run_as_authed_user": true,
  "run_as_user_id": "",
  "run_as_user_id_script_source": ""
}
```

+ System :

```
{
  "run_as_authed_user": false,
```

```

        "run_as_user_id": "",
        "run_as_user_id_script_source": "",
    }
+ User ID :

{
    "run_as_authed_user": false,
    "run_as_user_id": "<Stitch User ID>",
    "run_as_user_id_script_source": "",
}
+ Script :

{
    "run_as_authed_user": false,
    "run_as_user_id": "",
    "run_as_user_id_script_source": "<Stringified Function>",
}

```

- Bước 6: Configure Function Execution Logs
  - Bước 7: Specify an Authorization Expression
- VD:

```
{
    "%request.remoteIPAddress": {
        "%nin": [
            "248.88.57.58",
            "19.241.23.116",
            "147.64.232.1"
        ]
    }
}
```

- Bước 8: Configure the Function's Privacy Level
- Bước 9: Viết code Function
- Bước 10: Import the Function

```

stitch-cli login --username=<MongoDB Cloud Username> --api-key=
<MongoDB Cloud API Key>
stitch-cli import

```

#### 4.4. Work with functions

##### a. Gọi hàm

Ví dụ dưới đây thể hiện việc gọi một hàm đơn giản có tên là sum có hai đối số, cộng lại và trả về kết quả:

```
// sum: cộng hai số
exports = function(a, b) {
    return a + b;
};
```

- Call từ một function khác:

```
// difference: lấy a trừ b bằng hàm sum
exports = function(a, b) {
    return context.functions.execute("sum", a, -1 * b);
};
```

- Call từ một biểu thức JSON:

```
{
  "numGamesPlayed": {
    "%function": {
      "name": "sum",
      "arguments": [
        "%root.numWins",
        "%root.numLosses"
      ]
    }
  }
}
```

- Call từ Client Application:
  - JavaScript SDK:

```
const client = Stitch.defaultAppClient;
client.callFunction("sum", [3, 4]).then(result => {
  console.log(result) // Output: 7
});
```

- Android SDK:

```
StitchAppClient client = Stitch.getDefaultAppClient()
client.callFunction("sum", Arrays.asList(3, 4), BsonValue.class)
    .addOnCompleteListener(new OnCompleteListener<BsonValue>() {
        @Override
        public void onComplete(@NonNull final Task<BsonValue> task)
```

```

        if (task.isSuccessful()) {
            Log.d("stitch", String.format("%s", task.getResult()));
        // Output: 7
    } else {
        Log.e("stitch", "Error calling function:", task.getException());
    }
}
);

```

- o IOS SDK:

```

let client = Stitch.defaultAppClient!
client.callFunction(withName: "sum", withArgs: [3, 4]) { (result: StitchResult<String>) in
    switch result {
        case .success(let sum):
            print(sum)
        case .failure(let error):
            print("Failed to calculate sum: \(String(describing: error))")
    }
}

```

*b. Access function context*

Application Logic :

- Call a function :

```

// difference: lấy a trừ b bằng hàm sum
exports = function(a, b) {
    return context.functions.execute("sum", a, -1 * b);
};

```

- Kết nối tới External Service :

```

exports = function() {
    // Instantiate a service client for the HTTP Service named "myHttpService"
    const http = context.services.get("myHttpService");
    // Call the HTTP service's `get` action (if the service rules allow it)
    return http.get({ url: "https://www.mongodb.com" });
}

```

```
};
```

- Access Global Values :

```
exports = function() {
    // Get a global value (or `undefined` if no value has the specified name)
    const theme = context.values.get("theme");
    console.log(theme.colors)      // Output: { red: "#ee1111", blue: "#1111ee" }
    console.log(theme.colors.red)   // Output: "#ee1111"
};
```

#### Execution Information

- Lấy thông tin của user :

```
exports = function() {
    const currentUser = context.user
    console.log(currentUser.id)
}
```

- Nhận thông tin yêu cầu :

```
exports = function() {
    const request = context.request;
    if (request.httpReferrer == "https://www.example.com/") {
        applyPercentageDiscount(10);
    }
}
```

- Xem một Request's Headers :

```
exports = function() {
    return context.request.requestHeaders;
    // e.g. {
    //     "Content-Type": ["application/json"],
    //     "Cookie": [
    //         "someCookie=someValue",
    //         "anotherCookie=anotherValue"
    //     ]
    // }
}
```

- Xem một tham số truy vấn yêu cầu (Request's Query Parameters) :

```
exports = function(payload, response) {
```

```

const rawQuery = context.request.rawQueryString;
// e.g. "?someParam=foo&anotherParam=42"
const query = payload.query
// e.g. { someParam: "foo", anotherParam: 42 }
}

```

- ❖ Webhook query parameters: Các trường trong `payload.query` không được đảm bảo theo thứ tự như đã được chỉ định trong yêu cầu. Nếu bạn cần giữ nguyên thứ tự của các tham số truy vấn, hãy sử dụng `context.request.rawQueryString`.
- Kiểm tra function có chạy như system user hay không?

```

const isSystemUser = context.runningAsSystem()
if(isSystemUser) {
    // Do some work that bypasses rules
} else {
    // Do some work in the context of the user that called the function.
}

```

### Incoming Webhook Metadata

- Nhận Current Incoming Webhook URL

```

exports = function(payload, response) {
    // Returns the webhook URL as a string
    return context.request.webhookUrl
}

```

- Nhận Current Incoming Webhook HTTP Method

```

exports = function(payload, response) {
    switch(context.request.httpMethod) {
        case "GET": { /* Handle GET requests */ }
        case "POST": { /* Handle POST requests */ }
        case "PUT": { /* Handle PUT requests */ }
        case "DELETE": { /* Handle DELETE requests */ }
        case "PATCH": { /* Handle PATCH requests */ }
        default: {}
    }
}

```

## 4.5. References

### a. Function Context

Các hàm MongoDB Stitch có thể tương tác với các dịch vụ được kết nối, thông tin người dùng, các giá trị được xác định trước và các chức năng khác thông qua các modules được gắn với biến global context.

Biến context chứa các modules sau:

- Context Modules :

- `context.services.get(serviceName)`
- `context.functions.execute(functionName, args...)`
- `context.values.get(valueName)`
- `context.http`
  - `context.http.get()`

```
exports = async function() {  
  const response = await context.http.get({ url: "https://www.example.com/users" })  
  // The response body is a BSON.Binary object. Parse it and return.  
  return EJSON.parse(response.body.text());  
};  
  ◦ context.http.post()  
  
exports = async function() {  
  const response = await context.http.post({  
    url: "https://www.example.com/messages",  
    body: { msg: "This is in the body of a POST request!" },  
    encodeBodyAsJSON: true  
  })  
  // The response body is a BSON.Binary object. Parse it and return.  
  return EJSON.parse(response.body.text());  
};  
  ◦ context.http.put()  
  
exports = async function() {  
  const response = await context.http.put({  
    url: "https://www.example.com/messages",  
    body: { msg: "This is in the body of a PUT request!" },  
    encodeBodyAsJSON: true  
  })  
  // The response body is a BSON.Binary object. Parse it and return.  
  return EJSON.parse(response.body.text());  
};  
  ◦ context.http.patch()
```

```

exports = async function() {
  const response = await context.http.patch({
    url: "https://www.example.com/diff.txt",
    body: { msg: "This is in the body of a PATCH request!" },
    encodeBodyAsJSON: true
  })
  // The response body is a BSON.Binary object. Parse it and return.
  return EJSON.parse(response.body.text());
};

○ context.http.delete()

exports = async function() {
  const response = await context.http.delete({ url: "https://www.example.com/user/8675309" })
};

○ context.http.head()

exports = async function() {
  const response = await context.http.head({ url: "https://www.example.com/users" })
  // The response body is a BSON.Binary object. Parse it and return.
  EJSON.parse(response.body.text());
};

```

- Context Properties :

- context.user

VD: context.user document phản ánh Email / Password của user được liên kết với một khóa User API.

```
{
  "id": "5cbf68583025b12840664682",
  "type": "normal",
  "data": {
    "email": "someone@example.com",
    "name": "myApiKeyName"
  },
  "identities": [
    {
      "id": "5cbf68583025b12880667681",
      "provider_type": "local-userpass"
    },

```

```

        {
          "id": "5cbf6c6a922616045a388c71",
          "provider_type": "api-key"
        }
      ]
    }
  • context.request

```

VD: Tài liệu context.request sau đây phản ánh lệnh gọi function được phát hành từ <https://myapp.example.com/> bởi người dùng duyệt bằng Chrome 73 trên macOS High Sierra:

```

{
  "remoteIPAddress": "54.173.82.137",
  "httpReferrer": "https://myapp.example.com/",
  "httpUserAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 Safari/537.36"
}

```

#### - Context Method

- context.runningAsSystem()

```

const isSystemUser = context.runningAsSystem()
if(isSystemUser) {
  // Do some work that bypasses rules
} else {
  // Do some work in the context of the user that called the function.
}

```

#### b. Utility Packages

Để thuận tiện cho nhà phát triển, các MongoDB Stitch function có quyền truy cập vào các gói tiện ích được cung cấp thông qua các biến toàn cầu khác nhau.

Các global utility packages sau đây có sẵn trong tất cả các function:

- Console
  - console.log(**arg...**)
  - console.warn(**arg...**)
  - console.error(**arg...**)
- JSON
  - JSON.parse(**<json-string>**)
  - JSON.stringify(**<json-object>**)

- EJSON

- EJSON.parse(<ejson-string>)
- EJSON.stringify(<ejson-object>)

- BSON

Để tránh sự nhầm lẫn với \$regex EJSON type, các truy vấn với toán tử \$regex phải sử dụng BSON.BSONRegExp.

- BSON.Binary

- BSON.Binary.fromHex(hexString, subType)

Returns: BSON.Binary

- BSON.Binary.toHex()

Returns: String

- BSON.Binary.fromBase64(base64String, subType)

Returns: String

- BSON.Binary.toBase64()

Returns: String

- BSON.Binary.text()

Returns: String

- BSON.Decimal128

- BSON.Decimal128.fromString(decimalString)

Returns: BSON.Decimal128

- BSON.Long

- BSON.Long(low32, high32)

Returns: BSON.Long

VD:

```
exports = function () {
  return {
    a: BSON.Long(10),
    b: BSON.ObjectId("5a14179d01236a9fc1086df6"),
    c: BSON.Code("var x = 1"),
    d: new Date()
  };
};
```

- utils.jwt

- utils.jwt.encode(signingMethod, payload, secret)

Returns: Web JSON string encoded.

VD:

```
{  
  "sub": "1234567890",
```

```

    "name": "Joe Example",
    "iat": 1565721223187
}

exports = function() {
  const signingMethod = "HS512";
  const payload = {
    "sub": "1234567890",
    "name": "Joe Example",
    "iat": 1565721223187
  };
  const secret = "SuperSecret";
  return utils.jwt.encode(signingMethod, payload, secret);
}

```

- `utils.jwt.decode(jwtString, key)`

Returns: Decoded EJSON payload.

VD:

```

exports = function(jwtString) {
  const signingMethod = "HS512";
  const key = "SuperSecret";
  return utils.jwt.decode(jwtString, key);
}

{
  "sub": "1234567890",
  "name": "Joe Example",
  "iat": { "$numberDouble": 1565721223187 }
}

```

#### - `Utils.crypto`

- `utils.crypto.encrypt(encryptionType, message, key)`

Returns: BSON Binary object chứa chuỗi văn bản được mã hóa với loại và khóa mã hóa được chỉ định.

- `utils.crypto.decrypt(encryptionType, encryptedMessage, key)`  
Returns: BSON Binary object chứa thông điệp được giải mã.
- `utils.crypto.sign(encryptionType, message, privateKey, signatureScheme)`  
Returns: BSON.Binary cryptographic signature.
- `utils.crypto.verify(encryptionType, message, publicKey, signature, signatureScheme)`

Returns: Boolean.

- `utils.crypto.hmac(input, secret, hash_function, output_format)`  
Returns: signature của input, theo định dạng được chỉ định bởi output\_format.
- `utils.crypto.hash(hash_function, input)`  
Returns: Giá trị băm cho input được tạo bởi hàm băm được chỉ định.

VD:

```
exports = function() {
    const cur_unix_time = Math.floor(Date.now() / 1000);
    const api_url = "https://api.coinbase.com";
    const request_path = "/v2/accounts/primary/transactions";

    const message = cur_unix_time.toString() + "GET" + request_path;
    const signature = utils.crypto.hmac(message, context.values.get("coinbase_secret"), "sha256", "hex");

    const coinbase = context.services.get("myCoinbaseService")
    const response = await coinbase.get({
        url: api_url + request_path,
        headers: {
            "CB-ACCESS-SIGN": [signature],
            "CB-ACCESS-TIMESTAMP": [cur_unix_time.toString()],
            "CB-ACCESS-KEY": [context.values.get("coinbase_api_key")]
        }
    });

    return EJSON.parse(response.body.text());
};
```

## 5. Trigger (chỉ có trong MongoDB Stitch)

MongoDB Stitch trigger cho phép execute application và database logic tự động, để đáp ứng với các sự kiện hoặc dựa trên lịch trình được xác định trước. Stitch hỗ trợ ba loại trigger:

- *Database trigger:* Có thể tự động phản hồi khi các tài liệu được added, updated hoặc removed trong linked MongoDB Collection.
- *Authentication trigger:* thực thi logic bổ sung ở phía máy chủ khi user được created, authenticated hoặc deleted.
- *Scheduled trigger:* execute functions theo các khoảng thời gian đều đặn theo một lịch trình được xác định trước.

Stitch giới hạn việc thực thi các trigger function ở tốc độ 1000 lần thực hiện mỗi giây trên tất cả các trigger trong một application. Nếu các kích hoạt bổ sung kích hoạt vượt quá ngưỡng này, Stitch sẽ thêm các function liên quan của chúng vào hàng đợi và execute function khi có thể.

References:

- Trigger Snippets
- CRON Expressions

## 6. Một số phương thức có sẵn trong MongoDB

Tên phương thức	Cú pháp	Chức năng	Ví dụ
Limit()	>db.Collection.name .find().limit(NUMBER)	Giới hạn bản ghi. Phương thức nhận một tham số dạng số, là số document sẽ hiển thị	Có bản ghi:  <pre>{ "_id" : ObjectId("5983548781331adf45ec5"),   "title": "MongoDB Overview"}   { "_id" : ObjectId("5983548781331adf45ec6"),   "title": "NoSQL Overview"}   { "_id" : ObjectId("5983548781331adf45ec7"),   "title": "Tutorials Point Overview"}   &gt;db.mycol.find({},{title:1,_id:0}).limit(2)   ⇒ Kết quả: {"title": "MongoDB Overview"}   {"title": "NoSQL Overview"}</pre>
Skip()	>db.collectionname .find().limit(NUMBER).skip(NUMBER)	Nhận một tham số ở dạng số và được sử dụng để nhảy qua số Document đã xác định	Có bản ghi:  <pre>{ "_id" : ObjectId("5983548781331adf45ec5"),   "title": "MongoDB Overview"}   { "_id" : ObjectId("5983548781331adf45ec6"),   "title": "NoSQL Overview"}</pre>

			<pre>{   "_id": ObjectId("5983548781331adf45ec7"),   "title": "Tutorials Point Overview" }  &gt;db.mycol.find({}, {"title": 1, "_id": 0}).limit(2).skip(1)  ⇒ Kết quả: {"title": "MongoDB Overview"} {"title": "NoSQL Overview"}</pre>
Sort()	<pre>&gt;db.COLLECTION_NAME.find().sort({&lt;field&gt;:&lt;type&gt;})</pre> <p>(với type là 1 hoặc -1)</p>	Sắp xếp data theo thứ tự. 1 là tăng dần, -1 là giảm dần	<p>Có bản ghi:</p> <pre>{   "_id": ObjectId("5983548781331adf45ec5"),   "title": "MongoDB Overview" }  {   "_id": ObjectId("5983548781331adf45ec6"),   "title": "NoSQL Overview" }  {   "_id": ObjectId("5983548781331adf45ec7"),   "title": "Tutorials Point Overview" }  &gt;db.mycol.find({}, {"title": 1, "_id": 0}).sort({"title": -1})  ⇒ Kết quả: {"title": "Tutorials Point Overview"} {"title": "NoSQL Overview"} {"title": "MongoDB Overview"}</pre>

## 7. Chi tiết các hàm chính/hàm bổ trợ/toán tử có trong MongoDB

Tên Operator	Loại Operator	Tên	Chức năng	Ví dụ
Truy vấn và phép chiếu	Toán tử so sánh	\$eq	Lấy những giá trị = nhau	db.inventory.find( { qty: { \$eq: 20 } } )
		\$gt	Lấy những giá trị lớn hơn	db.inventory.find( { qty: { \$gt: 20 } } )
		\$gte	Lấy những giá trị ≥	db.inventory.find( { qty: { \$gte: 20 } } )

<i>Toán tử logic</i>	\$in	Lấy giá trị của những phần tử trong mảng	db.inventory.find( { qty: { \$in: [ 5, 15 ] } } )
	\$lt	Lấy những giá trị bé hơn	db.inventory.find( { qty: { \$lt: 20 } } )
	\$lte	Lấy những giá trị ≤	db.inventory.find( { qty: { \$lte: 20 } } )
	\$ne	Lấy những giá trị ≠	db.inventory.find( { qty: { \$ne: 20 } } )
	\$nin	Lấy giá trị của những phần tử không có trong mảng	db.inventory.find( { qty: { \$nin: [ 5, 15 ] } } ) Câu truy vấn sẽ lấy những trường có giá trị ≠. Nói cách khác kết quả sẽ là những document không có giá trị của trường qty
	\$and	Kết những mệnh đề truy vấn với toán tử AND và trả về documents có giá trị bằng với giá trị ở tất cả các mệnh đề	db.inventory.find( { \$and: [ { price: { \$ne: 1.99 } }, { price: { \$exists: true } } ] } ) Câu truy vấn sẽ lấy những trường có giá trị ≠1.99 và những trường có tồn tại
	\$or	Kết những mệnh đề truy vấn với toán tử OR, trả về documents có giá trị bằng với giá trị ở điều kiện của mệnh đề này hoặc mệnh đề khác	db.inventory.find( { price: { \$not: { \$gt: 1.99 } } } ) Câu truy vấn sẽ lấy những trường có giá trị ≤1.99 hoặc những trường không tồn tại

	\$nor	Trả về kết quả không thỏa mãn các điều kiện truyền vào, khác với not là nor có <b>param là 1 array và lấy field làm param.</b>	db.inventory.find( { \$nor: [ { price: 1.99 }, { qty: { \$lt: 20 } }, { sale: true } ] } )  Câu truy vấn sẽ:  Chứa những trường price có giá trị ≠ 1.99 và Chứa những giá trị của qty ≥ 20 và Chứa những trường sale có giá trị khác với giá trị sale trong document  Những documents không chứa các trường được gọi trong câu truy vấn
	\$not	Trả về kết quả không thỏa mãn các điều kiện truyền vào, lưu ý <b>param ở đây chỉ là 1 field</b>	db.inventory.find( { price: { \$not: { \$gt: 1.99 } } } )  Câu truy vấn sẽ lấy kq:  Chứa những trường price có giá trị < 1.99 hoặc  Chứa những giá trị price không tồn tại
<i>Toán tử truy vấn thành phần</i>	\$exists <u>Syntax:</u> {field : {\$exists: <boolean>}}	Lấy những giá trị tồn tại được xác định	db.inventory.find( { qty: { \$exists: true, \$nin: [ 5, 15 ] } } )  Câu truy vấn sẽ lấy tất cả documents mà trường qty tồn tại và giá trị của chúng ≠ 5 hoặc 15
	\$type <u>Syntax:</u> {field : { \$type: <BSON type> }}	Lấy những documents nếu như trường của loại được xác định có trong documents	db.addressBook.find( { "zipCode" : { \$type : "string" } } );  Câu truy vấn sẽ lấy tất cả documents mà trường zipcode có kiểu dữ liệu là string
	\$expr	Cho phép sử dụng các biểu thức tổng	db.monthlyBudget.find( { \$expr: { \$gt: [ "\$spent" , "\$budget" ] } } )

			hợp trong ngôn ngữ truy vấn	Câu truy vấn có kết quả là giá trị $\text{spent} \geq \text{budget}$
		\$jsonSchema	Xác thực các tài liệu theo Lược đồ JSON đã cho.	<pre>let myschema = {   required: [ "item", "qty", "instock" ],   properties: {     item: { bsonType: "string" },     qty: { bsonType: "int" },     size: {       bsonType: "object",       required: [ "uom" ]}}</pre> <p>Câu truy vấn sẽ tìm tất cả các documents trong collection thỏa mãn kiểu schema trên</p>
	Toán tử truy vấn đánh giá	\$mod	lấy ra tập các dữ liệu mà giá trị của các trường chia hết bởi <b>số chia</b> và <b>số còn lại</b>	<pre>db.inventory.find( { qty: { \$mod: [ 4, 0 ] } } )</pre> <p>Câu truy vấn sẽ tìm tất cả các documents chia 4 dư 0</p>
		\$regex	tìm kiếm so khớp theo mẫu pattern(tìm kiếm theo mẫu định dạng)	<pre>db.inventory.find( { item: { \$not: { \$regex: "^p.*" } } } )</pre> <p>Câu truy vấn sẽ tìm tất cả các documents có mẫu pattern như trên</p>
		\$text	Thực hiện tìm kiếm văn bản.	<pre>db.articles.insert([   { _id: 1, subject: "coffee", author: "xyz", views: 50 },   { _id: 2, subject: "Coffee Shopping", author: "efg", views: 5 },</pre>

				<pre>{ _id: 3, subject: "Baking a cake", author: "abc", views: 90 }])</pre> <pre>db.articles.find( { \$text: { \$search: "coffee" } } )</pre> <p>Câu truy vấn sẽ tìm tất cả các documents có coffee</p>
		\$where	Điều kiện truy vấn, toán tử where giúp ta liên tưởng tới toán tử where trong SQL	
		\$currentDate	Lấy giá trị ngày hiện tại cho trường	
Toán tử cập nhật	Toán tử miễn	\$inc	Tăng thêm giá trị cho 1 trường nếu là số dương, giảm đi nếu là số âm	<p>Có documents:{ _id: 1, sku: "abc123", quantity: 10, metrics: { orders: 2, }}</p> <p>Cập nhật: db.products.update( { sku: "abc123" }, { \$inc: { quantity: -2, "metrics.orders": 1 } })</p> <p>Kết quả:</p> <pre>{"_id" : 1, "sku" : "abc123", "quantity" : 8, "metrics" : { "orders" : 3, "ratings" : 3.5}</pre>

	\$min	Cập nhật 1 trường nếu giá trị của trường đó là nhỏ hơn trường hiện tại	Có bảng score như sau: { _id: 1, highScore: 800, lowScore: 200 } Cập nhật với min: db.scores.update( { _id: 1 }, { \$min: { lowScore: 199 } } ) Kết quả đổi lowScore = 199
	\$max	Cập nhật 1 trường nếu giá trị của trường đó là lớn hơn trường hiện tại	Có bảng score như sau: { _id: 1, highScore: 800, lowScore: 200 } Cập nhật với max: db.scores.update( { _id: 1 }, { \$max: { highScore: 900 } } ) Kết quả đổi highScore = 900
	\$mul	Thay đổi giá trị bằng cách nhân giá trị chỉ định trong scope \$mul với giá trị hiện tại (Chú ý: nếu field đặt trong scope \$mul này không tồn tại => )	Có bảng:{ "_id" : 1, "item" : "ABC", "price" : NumberDecimal("10.99"), "qty" : 25 } Cập nhật: db.products.update( { _id: 1 }, { \$mul: { price: NumberDecimal("1.25"), qty: 2 } } ) Kết quả:{ "_id" : 1, "item" : "ABC", "price" : NumberDecimal("13.7375"), "qty" : 50 }
	\$rename	Thay đổi tên field thành 1 field khác	
	\$set	Chỉ thay đổi giá trị truyền vào từ set, đặc điểm là nếu thuộc tính trong set tồn tại thì nó sẽ thay đổi giống như đã query, ngược lại nếu ko tồn tại thì nó	{_id: 100, sku: "abc123", quantity: 250, instock: true,} db.products.update(

		<b>sẽ tự động thêm như là 1 field mới</b>	<pre>{ _id: 100 }, { \$set:   {     quantity: 500   } }</pre> <p>Kết quả:</p> <pre>{_id: 100, sku: "abc123", quantity: 500, instock: true,}</pre>
	\$setOnInsert	Lấy giá trị của trường nếu kết quả cập nhật có thêm vào documents. Không xảy ra ảnh hưởng gì trên hoạt động cập nhật tài liệu khi nó tài liệu không có sự thay đổi gì	<pre>db.products.update(   { _id: 1 },   { \$set: { item: "apple" },     \$setOnInsert: { defaultQty: 100 },     { upsert: true } )</pre> <p>Kết quả:</p> <pre>{ "_id" : 1, "item" : "apple", "defaultQty" : 100 }</pre>
	\$unset	Hủy bỏ 1 thuộc tính cụ thể được chỉ định trong 1 row	
<i>Toán tử cập nhật mảng</i>	\$	Cập nhật phần tử đầu tiên hợp với điều kiện truy vấn	<pre>db.students.insert([   { "_id" : 1, "grades" : [ 85, 80, 80 ] } ]) db.students.updateOne(   { _id: 1, grades: 80 },   { \$set: { "grades.\$" : 82 } }) </pre> <p>Kết quả:</p>

			<pre>{ "_id" : 1, "grades" : [ 85, 82, 80 ] }</pre>
	\$[]	Cập nhật tất cả phần tử trong mảng của documents hợp với điều kiện truy vấn	<pre>db.collection.update(   { myArray: [ 5, 8 ] },   { \$set: { "myArray.\$[]": 10 } }, { upsert: true })</pre> <p>Kết quả:</p> <pre>{ "_id" : ObjectId(...), "myArray" : [ 10, 10 ] }</pre>
	\$[<identifier>]	Cập nhật tất cả phần tử trong mảng của documents hợp với điều kiện truy vấn của mảng	<pre>db.collection.update(   { myArray: [ 0, 1 ] },   { \$set: { "myArray.\$[element]": 2 } },   { arrayFilters: [ { element: 0 } ],     upsert: true } )</pre> <p>Kết quả:</p> <pre>{ "_id" : ObjectId(...), "myArray" : [ 2, 1 ] }</pre>
	\$addToSet	Thêm phần tử cho mảng nếu phần tử đó không tồn tại trong tập hợp	<pre>{ _id: 1, letters: ["a", "b"] }</pre> <p><u>Cập nhật:</u> db.test.update( { _id: 1 }, { \$addToSet: { letters: [ "c", "d" ] } } )</p> <p>Kết quả:</p> <pre>{ _id: 1, letters: [ "a", "b", [ "c", "d" ] ] }</pre> <p><u>Note:</u> Nếu giá trị là một mảng, toán tử sẽ xem toàn mảng như một phần tử riêng biệt</p>
	\$pop	Xóa phần tử đầu hoặc cuối của mảng	<pre>{ _id: 1, scores: [ 8, 9, 10 ] }</pre>

			<p><u>Cập nhật:</u> db.students.update( { _id: 1 }, { \$pop: { scores: -1 } } )</p> <p>Kết quả:</p> <pre>{ _id: 1, scores: [ 9, 10 ] }</pre> <p><u>Note:</u> + \$pop sẽ không được thực hiện nếu đối tượng không là 1 mảng + Nếu \$pop xóa phần tử cuối cùng của trường, trường đó sẽ là một mảng rỗng</p>
	\$pull	Xóa các phần tử trong mảng theo điều kiện truy vấn	<pre>{ _id: 1,   fruits: [ "apples", "pears", "oranges",   "grapes", "bananas" ], vegetables:   [ "carrots", "celery", "squash",   "carrots" ]}</pre> <p><u>Cập nhật:</u> db.stores.update( { }, { \$pull: { fruits: { \$in: [ "apples", "oranges" ] } }, vegetables: "carrots" } }, { multi: true } )</p> <p>Kết quả:</p> <pre>{"_id" : 1,"fruits" : [ "pears", "grapes", "bananas" ],"vegetables" : [ "celery", "squash" ]}</pre>
	\$push	Thêm phần tử vào mảng	<pre>db.students.update(   { _id: 1 },   { \$push: { scores: 89 } } )</pre> <p>Kết quả: giá trị 89 sẽ được thêm vào mảng</p>
	\$pullAll	Xóa tất cả các giá trị khỏi mảng nếu hợp với điều kiện	<pre>{ _id: 1, scores: [ 0, 2, 5, 5, 1, 0 ] }</pre> <p><u>Cập nhật:</u> db.survey.update( { _id: 1 }, { \$pullAll: { scores: [ 0, 5 ] } } )</p> <p>Kết quả:</p>

			{ "_id" : 1, "scores" : [2, 1] }
	\$each	Hỗ trợ toán tử \$push và \$addToSet để bổ sung thêm nhiều mục cho việc cập nhật mảng	<p><u>Hỗ trợ cho \$push:</u></p> <pre>db.students.update(   { name: "joe" },   { \$push: { scores: { \$each: [ 90, 92, 85 ] } } })</pre> <p>Kết quả: 90,92,85 sẽ được thêm vào scores của người tên joe</p> <p><u>Hỗ trợ cho \$addToSet:</u></p> <pre>db.inventory.update(   { _id: 2 },   { \$addToSet: { tags: { \$each: [ "camera", "electronics", "accessories" ] } } })</pre> <p>Kết quả: "camera", "electronics", "accessories" sẽ được thêm vào thuộc tính tags của id: 2</p>
	\$position	Sửa đổi toán tử \$push để xác định vị trí trong mảng cần thêm các phần tử  <u>Note:</u> chỉ số của \$position nếu là số dương thì phần tử được thêm vào kế những phần tử bên trái của mảng, số âm thì thêm bên phải	<p>Có bản ghi:{ "_id" : 1, "scores" : [ 100 ] }</p> <p>Cập nhật: db.students.update({ _id: 1},  { \$push: {scores: {  \$each: [ 50, 60, 70 ],  \$position: 0  }}})</p> <p>Kết quả: { "_id" : 1, "scores" : [ 50, 60, 70, 100 ] }</p> <p>Có bản ghi:{ "_id" : 1, "scores" : [ 20,50,70,80,100 ] }</p> <p>Cập nhật: db.students.update({ _id: 1},</p>

				<pre>{\$push: {scores: {     \$each: [ 10, 60 ],     \$position: -2 }}})</pre> <p>Kết quả: { "_id" : 1, "scores" : [ 20, 50, 70, 10, 60, 80, 100 ] }</p>
	\$slice	<p>Giới hạn số phần tử trong mảng được cập nhật bởi toán tử \$push</p> <p><u>Note:</u> chỉ số của \$slice nếu là số dương thì là sắp xếp tăng dần và ngược lại với chỉ số âm</p>		<p>Có bản ghi: { "_id" : 2, "scores" : [ 89, 90 ] }</p> <p>Cập nhật: db.students.update({ _id: 2 }, {\$push: {scores: {     \$each: [ 100, 20 ],     \$slice: 3 }}})</p> <p>Kết quả: { "_id" : 2, "scores" : [ 89, 90, 100 ] }</p>
	\$sort	<p>Sắp xếp các phần tử trong mảng được cập nhật bởi toán tử \$push</p>		<p>Có bản ghi:</p> <pre>{ "_id" : 2, "tests" : [ 89, 70, 89, 50 ] }</pre> <p>Cập nhật:</p> <pre>db.students.update(     { _id: 2 },     { \$push: { tests: { \$each: [ 40, 60 ], \$sort: 1 } } })</pre> <p>Kết quả: { "_id" : 2, "tests" : [ 40, 50, 60, 70, 89, 89 ] }</p>
<i>Aggregation Pipeline Stages</i>		<p><u>Syntax:</u> db.collection.aggregate( [ { &lt;stage&gt; }, ... ] )</p>		
		\$addFields	Thêm trường	<p><u>Syntax:</u></p> <pre>{ \$addFields: { &lt;newField&gt;: &lt;expression&gt;, ... } }</pre> <p><u>Ví dụ:</u></p>

			<pre>db.scores.aggregate( [   {     \$addFields: {       totalHomework: { \$sum: "\$homework" },       totalQuiz: { \$sum: "\$quiz" }     }   }) }</pre>
	\$bucket	<p>Phân loại document thành các nhóm, dựa trên một biểu thức xác định</p>	<p><u>Syntax:</u> {</p> <pre>\$bucket: {   groupBy: &lt;expression&gt;,   boundaries: [ &lt;lowerbound1&gt;, &lt;lowerbound2&gt;, ... ],   default: &lt;literal&gt;,   output: {     &lt;output1&gt;: { &lt;\$accumulator expression&gt; },     ...     &lt;outputN&gt;: { &lt;\$accumulator expression&gt; }   } } }</pre> <p><u>Ví dụ:</u></p> <pre>db.artwork.aggregate( [   {     \$bucket: {       groupBy: "\$price",       boundaries: [ 0, 200, 400 ],       default: "Other",     }   } ])</pre>

```

        output: {
            "count": { $sum: 1 },
            "titles" : { $push: "$title" }
        }
    }
])

```

Phân loại tự động document thành các nhóm, dựa trên một biểu thức xác định

\$bucketAuto

Syntax:

```
{
    $bucketAuto: {
        groupBy: <expression>,
        buckets: <number>,
        output: {
            <output1>: { <$accumulator expression> },
            ...
        }
    }
}
```

granularity: <string>

}

Ví dụ:

```
db.artwork.aggregate([
{
    $bucketAuto: {
        groupBy: "$price",
        buckets: 4
    }
}
])
```

	\$collStats	Trả về số lượng thống kê của collection hoặc view	<u>Syntax:</u> <pre>{   \$collStats:   {     latencyStats: { histograms: &lt;boolean&gt; },     storageStats: { scale: &lt;number&gt; },     count: {}   } }</pre> <u>Ví dụ:</u> <pre>db.matrices.aggregate( [ { \$collStats: { latencyStats: { histograms: true } } } ] )</pre>
	\$count	Trả về số lượng của document với chỉ định ban đầu <u>Note:</u> \$count là sự kết hợp của \$group và \$sum	<u>Syntax:</u> <pre>{ \$count: &lt;string&gt; }</pre> <u>Ví dụ:</u> <pre>db.scores.aggregate( [{\$match: {score: {\$gt: 80}}}, {   \$count: "passing_scores"}])</pre>
	\$facet	Xử lý nhiều Aggregation Pipeline Stages của document đầu vào.	<u>Syntax :</u> <pre>{ \$facet:   {     &lt;outputField1&gt;: [ &lt;stage1&gt;, &lt;stage2&gt;, ... ],     &lt;outputField2&gt;: [ &lt;stage1&gt;, &lt;stage2&gt;, ... ],     ...   } }</pre>

		Gom nhóm document đầu vào	<u>Syntax:</u> <pre>{   \$group:   {     _id: &lt;expression&gt;, // Group By Expression     &lt;field1&gt;: { &lt;accumulator1&gt; : &lt;expression1&gt; },     ...   } }</pre> <u>Ví dụ :</u> <pre>db.sales.aggregate( [   {     \$group: {       _id: null,       count: { \$sum: 1 }     }   } ])</pre>
	\$indexStats	Trả về số lượng thống kê của index sử dụng trong collection	<u>Syntax:</u> <pre>{ \$indexStats: { } }</pre> <u>Ví dụ :</u> <pre>db.orders.aggregate( [ { \$indexStats: { } } ] )</pre>
	\$limit	Giới hạn của document đầu vào	<u>Syntax:</u> <pre>{ \$limit: &lt;positive integer&gt; }</pre> <u>Ví dụ :</u> <pre>db.article.aggregate(   { \$limit : 5 } );</pre>

	\$listSessions	Liệt kê tất cả các Session đang hoạt động lâu dài của hệ thống	<u>Syntax:</u> { \$listSessions: <document> } <u>Ví dụ :</u> use config db.system.sessions.aggregate( [ { \$listSessions: { allUsers: true } } ] )
	\$lookup	Thực hiện kết trái giữa các collection trong cùng CSDL để lọc kết quả	<u>Syntax:</u> {\$lookup: { from: <collection to join>, localField: <field from the input documents>, foreignField: <field from the documents of the "from" collection>, as: <output array field> }} <u>Ví dụ :</u> db.collection.aggregate([ { \$lookup: { from: "fromCollection", ... } }])
	\$match	Lọc document cho phép các tài liệu khớp với những tài liệu không sửa đổi sang giai đoạn pineline tiếp theo  Với mỗi tài liệu input, output sẽ gồm một hoặc không document	<u>Syntax:</u> { \$match: { <query> } } <u>Ví dụ :</u> db.articles.aggregate( [ { \$match : { author : "dave" } } ] );
	\$merge	Ghi kết quả những document của tiến trình tổng hợp pineline vào	<u>Syntax:</u> { \$merge: { }

		<p>collection. Sau đó gộp những document là kết quả của tiến trình và đưa ra output</p> <p>Để sử dụng \$merge, bắt buộc nó phải nằm trong tiến trình cuối cùng của pineline</p>	<pre>into: &lt;collection&gt; -or- { db: &lt;db&gt;, coll: &lt;collection&gt; }, on: &lt;identifier field&gt; -or- [ &lt;identifier field1&gt;, ...], // Optional  let: &lt;variables&gt;, // Optional  whenMatched: &lt;replace keepExisting merge fail pipeline&gt;, // Optional  whenNotMatched: &lt;insert discard fail&gt; // Optional</pre> <p>} } <u>Ví dụ :</u></p> <pre>{ \$merge: { into: "myOutput", on: "_id", whenMatched: "replace", whenNotMatched: "insert" } }</pre>
	\$out	<p>Ghi những document kết quả của tiến trình tổng hợp Pineline vào collection</p> <p>Để sử dụng \$out, nó phải nằm trong tiến trình cuối cùng của pineline</p>	<p><u>Syntax:</u></p> <pre>{ \$out: "&lt;output-collection&gt;" } <u>Ví dụ :</u> db.books.aggregate( [ { \$group : { _id : "\$author", books: { \$push: "\$title" } } }, { \$out : "authors" } ] )</pre>
	\$planCache Stats	Trả về thông tin của plan Cache cho collection	<p><u>Syntax:</u></p> <pre>{ \$planCacheStats: { } }</pre> <p><u>Ví dụ :</u></p> <pre>db.orders.aggregate( [ { \$planCacheStats: { } } ] )</pre>
	\$project	Định hình lại mỗi document trong luồng, chẳng hạn như thêm trường	<p><u>Syntax:</u></p> <pre>{ \$project: { &lt;specification(s)&gt; } }</pre> <p><u>Ví dụ :</u></p>

		<p>hay xóa trường đã tồn tại.</p> <p>Cho mỗi document input, có một output tương ứng</p>	<pre>db.books.aggregate( [ { \$project : { title : 1 , author : 1 } } ] )</pre>
	\$redact	<p>Định hình lại mỗi document trong luồng bằng cách giới hạn nội dung cho từng tài liệu dựa trên thông tin đã được lưu trữ trên document đó</p> <p>Cho mỗi document input, có một hoặc không có output nào</p>	<u>Syntax:</u> <pre>{ \$redact: &lt;expression&gt; }</pre>
	\$replaceRoot	<p>Thay thế document bằng document nhúng đã được chỉ định. Hành động này thay thế tất cả những trường đã tồn tại trong document input, bao gồm cả id</p>	<u>Syntax:</u> <pre>{ \$replaceRoot: { newRoot: &lt;replacementDocument&gt; } }</pre> <u>Ví dụ :</u> <pre>db.collection.aggregate([   { \$replaceRoot: { newRoot: "\$name" } }])</pre>
	\$replaceWith	<p>Thay thế document bằng document nhúng được chỉ định. Hành động này thay thế tất cả các trường của document input, bao gồm cả id</p>	<u>Syntax:</u> <pre>{ \$replaceWith: &lt;replacementDocument&gt; }</pre> <u>Ví dụ :</u> <pre>db.collection.aggregate([   { \$replaceWith: "\$name" } ])</pre>
	\$sample	<p>Lựa chọn ngẫu nhiên số lượng được chỉ định từ document input</p>	<u>Syntax:</u> <pre>{ \$sample: { size: &lt;positive integer&gt; } }</pre> <u>Ví dụ :</u>

			db.users.aggregate( [ { \$sample: { size: 3 } } ] )
	\$set	Thêm trường mới vào document	<u>Syntax:</u> { \$set: { <newField>: <expression>, ... } } <u>Ví dụ :</u> db.vehicles.aggregate( [ { \$set: { "specs.fuel_type": "unleaded" } } ] )
	\$skip	Lượt bỏ n document đầu tiên được chỉ định.	<u>Syntax:</u> { \$skip: <positive integer> } <u>Ví dụ :</u> db.article.aggregate( { \$skip : 5 } );
	\$sort	Sắp xếp document	<u>Syntax:</u> { \$sort: { <field1>: <sort order>, <field2>: <sort order> ... } } <u>Ví dụ :</u> db.users.aggregate( [ { \$sort : { age : -1, posts: 1 } } ])
	\$sortByCount	Sắp xếp document và đếm chúng (không đếm những document trùng) <u>Note:</u> \$sortByCount là sự kết hợp giữa \$group và \$sort	<u>Syntax:</u> { \$sortByCount: <expression>} <u>Ví dụ :</u> db.exhibits.aggregate( [ { \$unwind: "\$tags" }, { \$sortByCount: "\$tags" } ] )

		\$unset	Xóa những trường khỏi document	<u>Syntax:</u> { \$unset: [ "<field1>", "<field2>", ... ] }
				<u>Ví dụ :</u> db.books.aggregate([ { \$unset: "copies" } ])
<u>Syntax:</u> { <operator>: [ <argument1>, <argument2> ... ] }				
Aggregation Pipeline Operators	\$sum	Tổng giá trị được xác định từ tất cả Document trong Collection đó		db.mycol.aggregate([{\$group :{_id : "\$by_user", numTutorial : {\$sum : "\$likes"}}}])
	\$avg	Tính trung bình của tất cả giá trị đã cho từ tất cả Document trong Collection đó		db.mycol.aggregate([{\$group :{_id : "\$by_user", numTutorial : {\$avg : "\$likes"}}}])
	\$min	Lấy giá trị nhỏ nhất của các giá trị từ tất cả Document trong Collection đó		db.mycol.aggregate([{\$group :{_id : "\$by_user", numTutorial : {\$min : "\$likes"}}}])
	\$max	Lấy giá trị lớn nhất của các giá trị từ tất cả Document trong Collection đó		db.mycol.aggregate([{\$group :{_id : "\$by_user", numTutorial : {\$max : "\$likes"}}}])
	\$push	Chèn giá trị vào trong một mảng trong Document kết quả		db.mycol.aggregate([{\$group :{_id : "\$by_user", url : {\$push: "\$url"}}}])
	\$addToSet	Chèn giá trị tới một mảng trong Document kết quả, nhưng không tạo các bản sao		db.mycol.aggregate([{\$group :{_id : "\$by_user", url : {\$addToSet : "\$url"}}}])
	\$first	Lấy Document đầu tiên từ Source Document theo nhóm		db.mycol.aggregate([{\$group :{_id : "\$by_user", firstUrl : {\$first: "\$url"}}}])

		\$last	Lấy Document cuối cùng từ Source Document theo nhóm	db.mycol.aggregate([{\$group : {_id : "\$by_user", last_url : {\$last : "\$url"}}}])
<p><b>Note:</b> Ngoài ra còn rất nhiều toán tử khác. Có thể xem thêm trong trang web của nhà phát triển:  <a href="https://docs.mongodb.com/manual/reference/operator/aggregation/">https://docs.mongodb.com/manual/reference/operator/aggregation/</a></p>				
Query Modifier		\$comment	Thêm chú thích cho câu truy vấn	<u>Sử dụng \$comment theo các cách:</u> <pre>&gt; db.collection.find( { &lt;query&gt; } )._addSpecial( "\$comment", &lt;comment&gt; )  &gt; db.collection.find( { &lt;query&gt; } ).comment( &lt;comment&gt; )  &gt; db.collection.find( { \$query: { &lt;query&gt; }, \$comment: &lt;comment&gt; } )</pre>
		\$explain	Yêu cầu MongoDB báo cáo về Query Execution Plan	<u>Sử dụng \$explain theo các cách:</u> <pre>&gt; db.collection.find()._addSpecial( "\$explain", 1 )  &gt; db.collection.find( { \$query: {}, \$explain: 1 } )</pre>
		\$hint	Yêu cầu MongoDB sử dụng index đã chỉ định	<u>Sử dụng \$hint theo các cách:</u> <pre>db.users.find()._addSpecial( "\$hint", { age : 1 } )  db.users.find( { \$query: {}, \$hint: { age : 1 } } )</pre>
		\$max	Chỉ định giới hạn trên cho chỉ mục sử dụng trong câu truy vấn	<u>Sử dụng \$max theo các cách:</u> <pre>db.collection.find( { &lt;query&gt; } ).max( { field1: &lt;max value&gt;, ... fieldN: &lt;max valueN&gt; } )  db.collection.find( { &lt;query&gt; } )._addSpecial( "\$max", { field1: &lt;max value1&gt;, ... fieldN: &lt;max valueN&gt; } )  db.collection.find( { \$query: { &lt;query&gt; }, \$max: { field1: &lt;max value1&gt;, ... fieldN: &lt;max valueN&gt; } } )</pre>

	\$min	Chỉ định giới hạn dưới cho chỉ mục trong câu truy vấn	<p><u>Sử dụng \$min theo các cách:</u></p> <pre>db.collection.find( { &lt;query&gt; } ).min( { field1: &lt;min value&gt;, ... fieldN: &lt;min valueN&gt;} )</pre> <pre>db.collection.find( { &lt;query&gt; } )._addSpecial( "\$min", { field1: &lt;min value1&gt;, ... fieldN: &lt;min valueN&gt;} )</pre> <pre>db.collection.find( { \$query: { &lt;query&gt; }, \$min: { field1: &lt;min value1&gt;, ... fieldN: &lt;min valueN&gt;} } )</pre>
	\$orderby	Trả về con trả với những tài liệu đã được sắp xếp theo chỉ định	<p><u>Sử dụng \$orderby theo các cách:</u></p> <pre>db.collection.find().sort( { age: -1 } )</pre> <pre>db.collection.find()._addSpecial( "orderby", { age : -1 } )</pre> <pre>db.collection.find( { \$query: {}, \$orderby: { age : -1 } } )</pre>
	\$returnKey	Yêu cầu con trả chỉ trả về trường có index	<p><u>Sử dụng \$returnKey theo các cách:</u></p> <pre>&gt; db.collection.find( { &lt;query&gt; } )._addSpecial( "\$returnKey", true )</pre> <pre>db.collection.find( { \$query: { &lt;query&gt; }, \$returnKey: true } )</pre>
	\$showDiskLoc	Sửa đổi các tài liệu trả về để tham chiếu đến ổ đĩa chứa tài liệu	<p><u>Thêm diskLoc:</u></p> <pre>"\$diskLoc": {     "file": &lt;int&gt;,     "offset": &lt;int&gt; }</pre> <p><u>Sử dụng \$showDiskLoc theo các cách:</u></p> <pre>db.collection.find().showDiskLoc()</pre> <pre>db.collection.find( { &lt;query&gt; } )._addSpecial("showDiskLoc" , true)</pre> <pre>db.collection.find( { \$query: { &lt;query&gt; }, \$showDiskLoc: true } )</pre>

		\$natural	Một thứ tự sắp xếp đặc biệt yêu cầu các tài liệu sử dụng thứ tự các tài liệu trên đĩa.	Sử dụng \$natural theo các cách:
--	--	-----------	--	----------------------------------

## 8. Chỉ mục

### 8.1. Tạo chỉ mục bằng MongoDB Compass:

Các bước tạo chỉ mục:

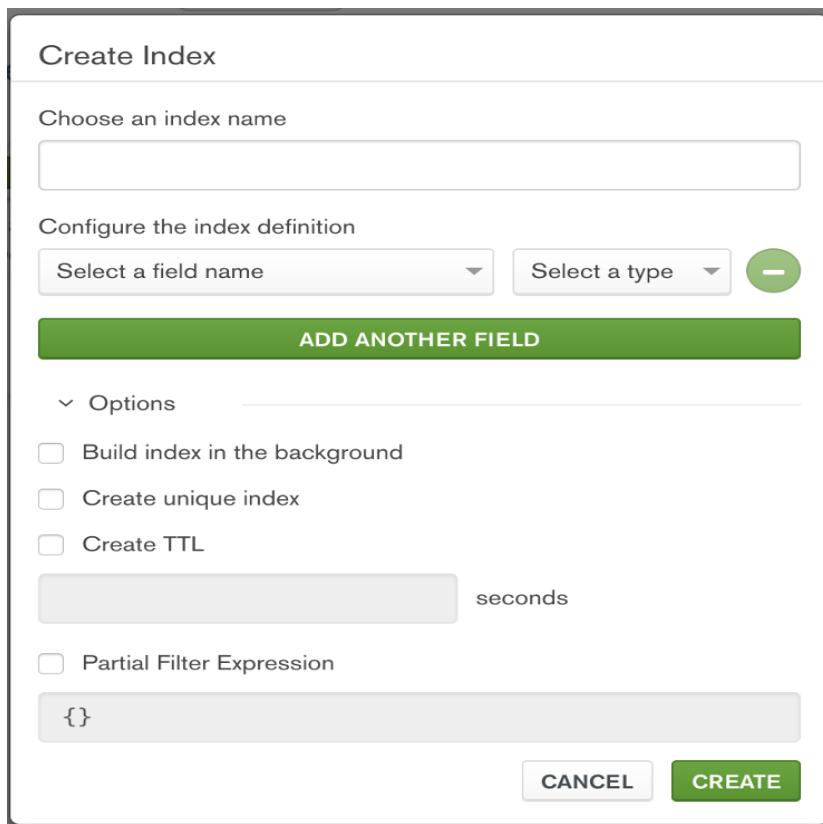
- Bước 1: Tìm đến collection trong CSDL mà bạn muốn tạo chỉ mục
- Bước 2: Chọn vào tab Indexes

Name and Definition	Type	Size	Usage	Properties	Drop
_id _id ↑	REGULAR ⓘ	36.0 KB	0 since Wed Jan 03 2018	UNIQUE ⓘ	

- Bước 3: Chọn vào nút Create Index

Name and Definition	Type	Size	Usage	Properties	Drop
_id _id ↑	REGULAR ⓘ	36.0 KB	0 since Wed Jan 03 2018	UNIQUE ⓘ	

Một hộp giao diện mới sẽ hiện ra:



- *Bước 4: (tùy chọn) Đặt tên cho Index. Có thể bỏ trống vì MongoDB Compass sẽ đặt tên mặc định cho chỉ mục*
- *Bước 5: Thêm trường cho Index. Nếu muốn thêm nhiều trường thì chọn Add another field*
- *Bước 6: (tùy chọn) Chỉ định các tùy chọn cho chỉ mục. Chỉ mục có thể được chỉ định thành các loại:*
  - Unique Index
  - TTL
  - Biểu thức lọc
- *Bước 7: Bấm Create => Tạo Index thành công*

## **8.2. *Tạo chỉ mục bằng MongoDB Shell:***

### *a. Chỉ mục mặc định:*

MongoDB tạo một chỉ mục đơn trên trường \_id trong suốt quá trình tạo collection. Chỉ mục \_id ngăn khách hàng chèn 2 documents với cùng giá trị cho trường \_id. Người dùng không thể xóa chỉ mục trên trường \_id này.

**Note:** Trong trường hợp sharded clusters, nếu người dùng không sử dụng trường \_id như khóa phân đoạn, thì ứng dụng đó phải đảm bảo tính duy nhất của giá trị trong trường \_id để ngăn lỗi. Điều này thường được thực hiện bằng cách sử dụng ObjectId – một phương thức tạo ID tự động theo tiêu chuẩn

### *b. Chỉ mục đơn:*

Chỉ mục là một cấu trúc dữ liệu, thu thập thông tin về giá trị của các trường trong các văn bản của một bộ sưu tập. Cấu trúc dữ liệu này được sử dụng trong tối ưu truy vấn MongoDB để xắp xếp nhanh các văn bản trong một bộ sưu tập.

Chúng ta có thể khởi tạo chỉ mục bằng cách gọi hàm `createIndex()` và cung cấp một văn bản với một hoặc nhiều khóa để đánh chỉ mục.

- Cú pháp: `db.collection.createIndex({ <field>:<options> })` với `<options>` là 1 hoặc -1 (1 là dữ liệu tăng dần, -1 là dữ liệu giảm dần)
- Tên chỉ mục: Tên mặc định cho một chỉ mục là ghép các khóa được lập chỉ mục và mỗi hướng khóa chính trong chỉ mục (tức là 1 hoặc -1) bằng cách sử dụng dấu gạch dưới để nối chúng lại

Ví dụ: `db.products.createIndex( { item: 1, quantity: -1 } )`

Tên chỉ mục là `item_1_quantity_-1`

Người dùng có thể tự đặt tên cho chỉ mục, có thể xem ví dụ:

Ví dụ: `db.products.createIndex(`

```
{ item: 1, quantity: -1 },  
{ name: "query for inventory" })
```

Tên chỉ mục là `queru for inventory`

Hàm `createIndex()` chỉ khởi tạo chỉ mục nếu nó chưa tồn tại, Để kiểm tra việc tồn tại chỉ mục trên collection `user` hay chưa, ta có thể sử dụng hàm:  
`db.user.getIndexes()`

c. Chỉ mục trên mảng:

Giả sử chúng ta muốn tìm kiếm tài liệu của người dùng dựa trên nhãn của họ. Để thực hiện việc này, chúng ta sẽ tạo ra chỉ mục trên nhãn của mảng trong collection

Tạo chỉ mục trên mảng là tạo chỉ mục trên từng trường của nó. Vì vậy trong trường này ta sẽ tạo chỉ mục trên nhãn của mảng.

Để tạo chỉ mục trên nhãn của mảng, thực hiện lệnh

```
>db.users.ensureIndex( { "tags":1 } )
```

Sau khi tạo chỉ mục, chúng ta có thể tìm kiếm nhãn của collection như thế này

```
>db.users.find( { tags: "cricket" } )
```

Để kiểm chứng chỉ mục thích hợp, sử dụng hàm `explain`

```
>db.users.find( { tags: "cricket" } ).explain()
```

d. Chỉ mục trên trường của tài liệu:

Giả sử ta muốn tìm kiếm document dựa trên thành phố/quận và trường mã pin. Vì những trường này là một phần của trường địa chỉ, chúng ta sẽ tạo index trên những trường nhỏ này.

Để tạo chỉ mục cho tất cả 3 trường nhỏ của trường lớn này, ta sử dụng câu lệnh

```
>db.users.ensureIndex ({"address.city":1,"address.state":1,"address.pincode":1})
```

Khi chỉ mục được tạo, ta có thể tìm kiếm mọi trường có nhiều trường nhỏ như câu lệnh sau:

```
>db.users.find ({"address.city":"Los Angeles","address.state":"California"})
```

Phương thức `ensureIndex()` cũng chấp nhận danh sách các tùy chọn tùy ý, được liệt kê dưới đây:

Tham số	Kiểu	Miêu tả
background	Boolean	Xây dựng chỉ mục trong Background để mà nó không gây trở ngại các hoạt động cơ sở dữ liệu khác. Xác định true để xây dựng trong Background. Giá trị mặc định là false
unique	Boolean	Tạo một unique index để mà Collection đó sẽ không chấp nhận việc chèn các Document có key kết nối với một giá trị đang tồn tại trong chỉ mục. Xác định là true để tạo unique index. Giá trị mặc định là false
name	Chuỗi	Tên của chỉ mục. Nếu không được xác định, MongoDB tạo một tên chỉ mục bằng cách nối chuỗi các tên của các trường đã được lập chỉ mục và sắp xếp thứ tự
dropDups	Boolean	Tạo một dropDups index trên một trường mà có thể có các bản sao. MongoDB chỉ lập chỉ mục choc ho lần xuất hiện đầu tiên của key và xóa tất cả Document từ Collection mà chứa lần xuất hiện tiếp theo của key đó. Xác định true để tạo dropDups index. Giá trị mặc định là false
sparse	Boolean	Nếu true, chỉ mục chỉ tham chiếu tới các Document với trường đã xác định. Các chỉ mục này sử dụng ít không gian hơn, nhưng vận hành theo cách khác nhau trong

		một số tình huống (cụ thể với sắp xếp). Giá trị mặc định là false
expireAfterSeconds	Số nguyên	Xác định một giá trị (bằng giây), dưới dạng một TTL để điều khiển thời gian bao lâu MongoDB duy trì các Document trong Collection này
v	Phiên bản index	Số phiên bản của chỉ mục. Phiên bản mặc định phụ thuộc vào phiên bản của MongoDB đang chạy khi tạo chỉ mục
weights	document	Là một số trong dãy từ 1 tới 99999 và biểu thị ý nghĩa của trường quan hệ tới các trường được lập chỉ mục khác về mặt score
default_language	Chuỗi	Với một text index, đây là ngôn ngữ xác định các qui tắc của chỉ mục. Giá trị mặc định là english
language_override	Chuỗi	Với một text index, xác định tên của trường được chứa trong Document, ngôn ngữ để nạp chồng ngôn ngữ mặc định. Giá trị mặc định là language

e. Tổng hợp các câu lệnh thao tác với index của MongoDB so với SQL:

	SQL	MongoDB
Tạo chỉ mục	CREATE INDEX myindexname ON users(name)	db.users.ensureIndex({name:1})
	CREATE INDEX myindexname ON users(name,ts DESC)	db.users.ensureIndex({name:1,ts:-1})
Xóa chỉ mục	DROP INDEX ten_index;	db.collection.dropIndexes({ a: 1, b: 1})

## 9. Sharding

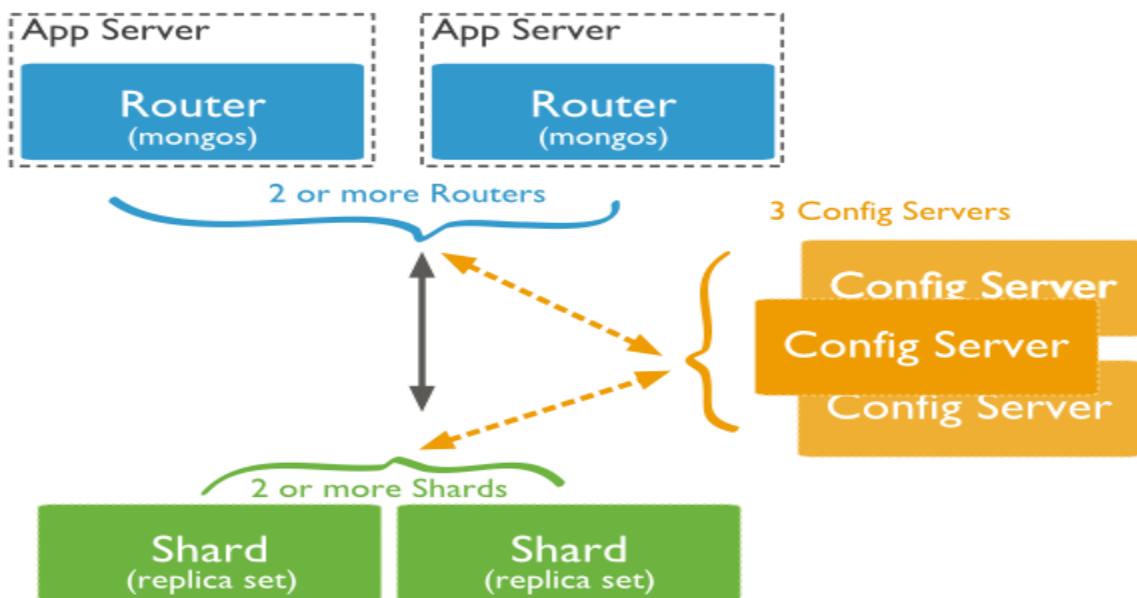
Sharding là một tiến trình lưu giữ các bản ghi dữ liệu qua nhiều thiết bị và nó là một phương pháp của MongoDB để đáp ứng yêu cầu về sự gia tăng dữ liệu. Khi kích cỡ của dữ liệu tăng lên, một thiết bị đơn không thể đủ để lưu giữ dữ liệu. Sharding giải quyết vấn đề này với việc mở rộng phạm vi theo bề ngang (horizontal scaling). Với

Sharding, bạn bổ sung thêm nhiều thiết bị để hỗ trợ cho việc tăng dữ liệu và các yêu cầu của các hoạt động đọc và ghi.

### **Tại sao sử dụng Sharding?**

- Trong Replication, tất cả hoạt động ghi thực hiện ở node sơ cấp.
- Các truy vấn tiềm tàng vẫn đến node sơ cấp.
- Một Replica Set đơn có giới hạn là 12 node.
- Bộ nhớ không thể đủ lớn khi tập dữ liệu hoạt động là lớn.
- Local Disk là không đủ lớn.
- Việc mở rộng phạm vi theo chiều dọc (vertical scaling) là quá tốn kém.

Dưới đây là sơ đồ minh họa Sharding trong MongoDB sử dụng Sharded Cluster:



Trong sơ đồ trên, có ba thành phần chính:

- **Shards:** được sử dụng để lưu giữ dữ liệu. Chúng cung cấp tính khả dụng cao và dữ liệu có tính đồng nhất. Trong môi trường tạo lập, mỗi Shard là một Replica Set riêng biệt.
- **Config Servers:** lưu giữ metadata của Cluster. Dữ liệu này chứa một ánh xạ của tập dữ liệu của Cluster tới Shards. Query Router sử dụng metadata này để hướng các hoạt động tới Shards cụ thể. Trong môi trường tạo lập, sharded clusters có chính xác 3 Config Servers.
- **Query Routers:** về cơ bản nó là mongo instance, giao diện với Ứng dụng Client và hướng các hoạt động tới Shard phù hợp. Query Router xử lý và hướng các hoạt động tới Shard và sau đó trả kết quả về Clients. Một Sharded Cluster có thể chứa nhiều hơn một Query Router để phân chia việc tải yêu cầu từ Client. Một Client gửi các yêu cầu tới một Query Router. Nói chung, một Sharded Cluster có nhiều Query Routers.

## 10. Map Reduce

Trong MongoDB Documentation, Map-Reduce là một hệ xử lý dữ liệu để cô đọng một khối lượng lớn dữ liệu thành các kết quả tổng thể có ích. MongoDB sử dụng lệnh mapReduce cho hoạt động Map-Reduce. Nói chung, Map Reduce được sử dụng để xử lý các tập dữ liệu lớn.

Cú pháp lệnh cơ bản:

```
>db.collection.mapReduce(  
  function() {emit(key,value);}, //map function  
  function(key,values) {return reduceFunction}, //reduce function  
  {  
    out: collection,  
    query: document,  
    sort: document,  
    limit: number  
  }  
)
```

Đầu tiên, hàm (function) của Map Reduce truy vấn Collection, sau đó ánh xạ các Document kết quả để phát xạ (Emit) các cặp key-value mà sau đó bị rút gọn dựa trên các key mà có nhiều value.

Trong cú pháp trên:

- **map** là một hàm JavaScript mà ánh xạ một value với một key và phát xạ một cặp key-value.
- **reduce** là một hàm JavaScript mà rút gọn hoặc nhóm tất cả Document có cùng key.
- **out** xác định vị trí của kết quả truy vấn Map-Reduce.
- **query** xác định tiêu chuẩn chọn tùy ý để lựa chọn các Document.
- **sort** xác định tiêu chuẩn sắp xếp tùy ý.
- **limit** xác định số lượng Document tối đa tùy ý để được trả về.

## 11. Replication

Giống như hệ quản trị CSDL quan hệ, MongoDB hỗ trợ replication của database theo thời gian thực hoặc gần với thời gian thực. Replication trong MongoDB rất đơn giản để cài đặt và sử dụng. Đó cũng là một trong những điểm đặc biệt của hệ quản trị cơ sở dữ liệu thế hệ mới, thế hệ cho nền tảng web 2.0 và công nghệ lưu trữ điện toán đám mây. Có nhiều kịch bản đặt ra cho việc sử dụng Replication. MongoDB hỗ trợ đầy đủ các trường hợp Replication mà thực tế đặt ra. Hiện nay MongoDB hỗ trợ các kịch bản Replication sau:

- Single master/single slave replication
- Single master/multiple slave replication
- Multiple master/single slave replication

- Cascade replication • Master/master replication
- Interleaved replication • Replica pairs (legacy MongoDB clustering)
- Replica sets (advanced MongoDB clustering)

### **11.1. Khi nào thì sử dụng Replication?**

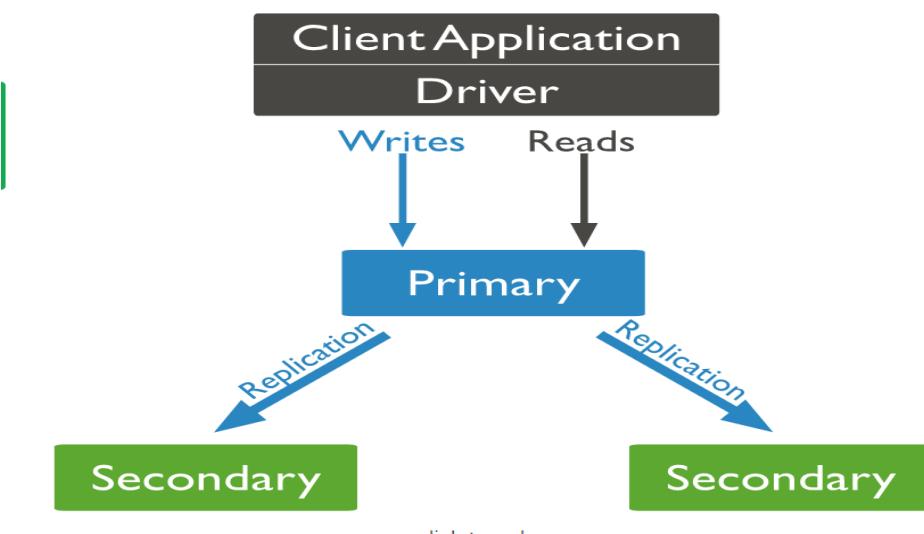
Replication là giải pháp được ứng dụng cho môi trường phân phối dữ liệu trên nhiều Server, chính vì vậy mà sử dụng chúng khi:

- Sao chép và phân phối dữ liệu trên nhiều Server khác nhau
- Phân phối bản sao dữ liệu theo lịch trình nhất định.
- Phân phối dữ liệu vừa thay đổi trên nhiều Server khác nhau.
- Cho phép nhiều người dùng và nhiều Server kết hợp dữ liệu khác nhau một cách thống nhất mà không sợ mất dữ liệu.
- Xây dựng CSDL sử dụng cho những ứng dụng trực tuyến hay ngoại tuyến
- Xây dựng ứng dụng Web khi người dùng cần trình bày một số lượng lớn dữ liệu.

### **11.2. Replica Set**

Replica set là một hệ replication trong MongoDB. Tập data sẽ được nhân bản trên nhiều server thay vì tập trung trên một single server. Nhờ vậy, replica set cung cấp tính năng high availability và dự phòng. Nó cũng scale read request cho mongodb. Mô hình của replica set trong mongodb gần giống replication trong mysql.

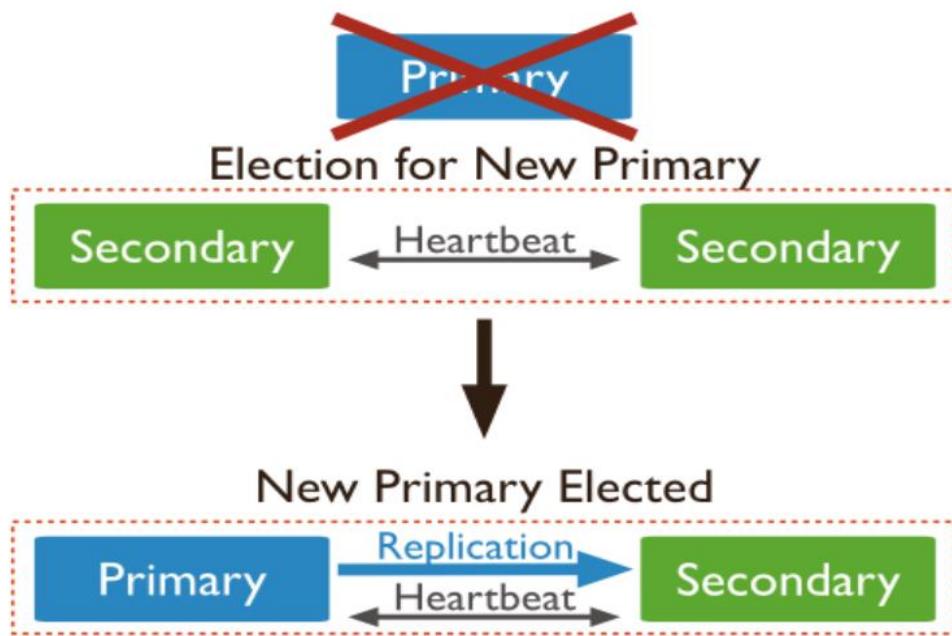
Tổng quan:



Một replica set chỉ có duy nhất một primary. Primary member sẽ nhận các write request. Primary ghi các thay đổi của nó vào oplog - một file có vai trò như binlog trên mysqld. Các secondary sẽ apply từ primary nên có chung data set với primary. Read request có thể scale trên primary và tất cả các secondary. Một replica set có thể có tối đa là 50 member. Nếu lớn hơn 50 member thì bạn phải dùng giải pháp khác.

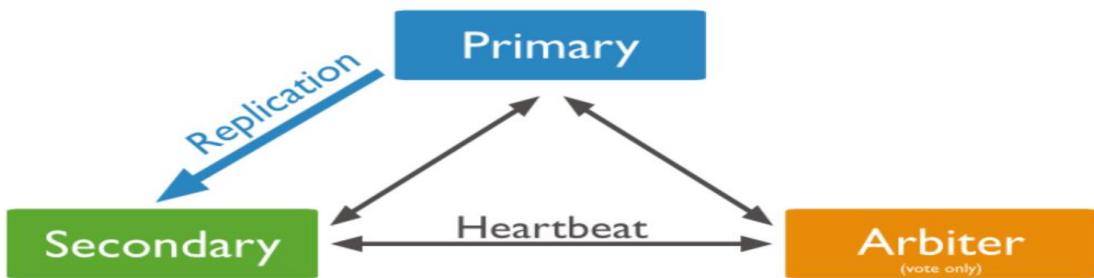
Mongodb có đề nghị giải pháp master-slave replication cho môi trường lớn hơn 50 member nhưng tôi chưa tìm hiểu về giải pháp này.

Giữa các member trong replica set luôn duy trì kết nối heartbeat nên khi một member nào đó down các member còn lại sẽ nhận ra luôn và tự động tiến hành failover. Đây là điểm khác biệt so với mysql. Mysql replication thiếu cơ chế để tự động failover (Trong các phiên bản mysql từ 5.6 hỗ trợ GTID thì mysql bắt đầu có thể failover tự động nhưng để thực hiện được, bạn cần sử dụng một node giám sát như mysqlfailover còn trong mongodb thì các member tự giám sát và tự failover )



Cơ chế thực hiện failover của replica set là dựa trên voting. Một secondary sẽ được bầu lên làm primary của cả replica set. Để voting thành công thì số member trong một replica set phải là số lẻ nếu không sẽ xảy ra trường hợp hai ứng viên đều nhận được số phiếu bầu bằng nhau rốt cục chẳng ai làm primary cả hoặc có thể dẫn đến tình huống có hai member đều tự nhận là primary nếu network partition xảy ra.

Tuy vậy, để giảm chi phí đầu tư, bạn có thể chỉ cần hai member trong một replica set. Thành viên thứ ba sẽ là một arbiter. Arbiter là một member đặc biệt. Nó không apply các oplog từ primary nên nó không lưu data gì cả. Nhiệm vụ của arbiter là giám sát hệ replica set qua các đường liên kết heartbeat và bầu chọn một secondary lên primary khi failover. Arbiter hoạt động không có gì nặng nề, bạn không cần server riêng cho arbiter nhưng arbiter không nên được deploy trên server dùng làm primary hay secondary trong replica set. Arbiter thì sẽ mãi là arbiter không như primary có thể down rồi trở thành secondary khi join lại vào replica set hay secondary có thể trở thành primary khi failover xảy ra.



Replication từ primary về secondary là bất đồng bộ (asynchronous) do đó writeset có trên primary sẽ không thể ngay lập tức có trên secondary. Hệ quả là data set trên secondary sẽ không phản ánh trạng thái mới nhất của data set trên primary. Thực ra đây là nhược điểm đều có trong mọi hệ thống mà dựa trên replication.

### 11.3. Read Preference

Mặc định, driver mongodb sẽ luôn thực hiện read request đến master. Read request chỉ từ primary gọi là strict consistency với ý nghĩa application sẽ luôn lấy data state mới nhất. Nếu application không yêu cầu luôn trả về phiên bản mới nhất của data thì bạn có thể điều chỉnh read preference của mongodb để scale out read. Read từ secondary gọi là eventual consistency với ý nghĩa rồi cuối cùng thì thế nào data state trên secondary cũng đồng nhất với primary. Read preference có 5 mode tất cả:

<http://docs.mongodb.org/manual/core/read-preference/#read-preference-modes>

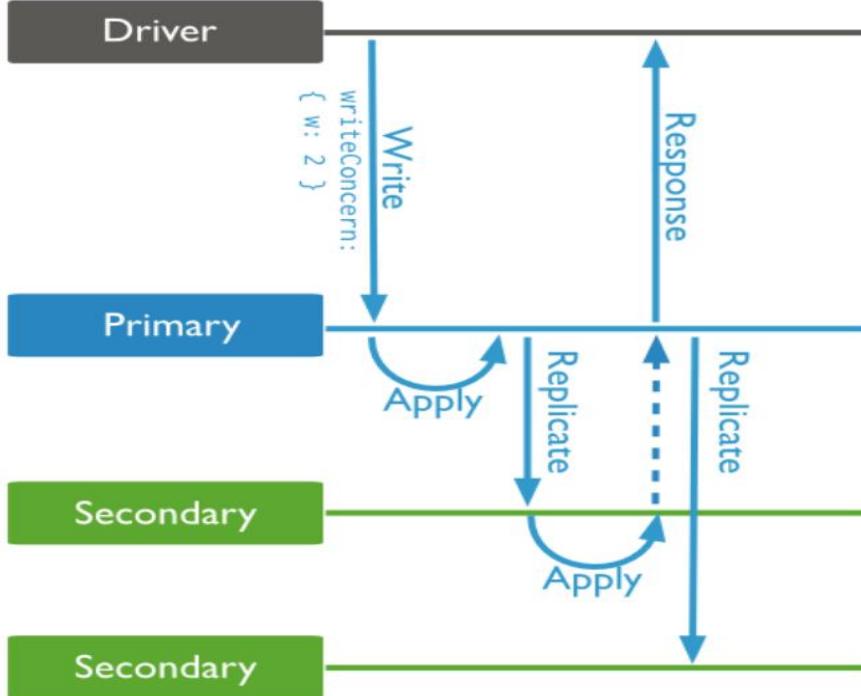
- **primary** là mặc định, mọi read request sẽ chỉ đi đến primary
- **primaryPreferred**: mọi read request sẽ đi đến primary nhưng nếu primary down nó sẽ đi đến secondary
- **secondary**: mọi read request chỉ đi đến các secondary.
- **secondaryPreferred**: mọi read request sẽ đi đến secondary nhưng nếu tất cả các secondary down nó sẽ đi đến primary.
- **nearest**: read request sẽ đến member có network latency thấp nhất không phân biệt member đó là primary hay secondary.

Tham số read preference bạn sẽ khai báo từ application. Chi tiết tham khảo ở đây:

<https://docs.mongodb.org/manual/reference/connection-string/>

### 11.4. Write Concern

Write concern sẽ yêu cầu MongoDB xác nhận một write request có thành công hay không. Với replica set, mặc định write concern sẽ chỉ yêu cầu mongodb xác nhận write request trên primary member. Ở chế độ này, application chỉ có thể biết được write request có thành công hay không trên primary. Nó không thể biết được liệu write request đó đã được replicate thành công đến các secondary member hay chưa.



Mongodb cho phép bạn thay đổi hành vi mặc định của write concern. Application có thể lựa chọn thay đổi write concern ngay trong write request. Hành vi write concern thay đổi chỉ áp dụng cho write request này:

```
- db.products.insert(
-   { item: "envelopes", qty : 100, type: "Clasp" },
-   { writeConcern: { w: 2, wtimeout: 5000 } }
- )
```

Write request insert một item vào collection products sẽ được trả về sau khi write request thực hiện trên primary và ít nhất một secondary (w: 2) hoặc nếu không có kết quả trả về thì kết thúc trong vòng 5s (wtimeout: 5000). Giá trị wtimeout để tránh cho write request bị block lại quá lâu nếu như không có đủ số members mà write concern cần.

Bạn cũng có thể thay đổi hành vi mặc định của write concern trên phạm vi toàn bộ các write request thay vì chỉ từng request như trên:

```
- cfg = rs.conf()
- cfg.settings = {}
- cfg.settings.getLastErrorDefaults = { w: "majority", wtimeout: 5000 }
- rs.reconfig(cfg)
```

Cấu hình trên sẽ yêu cầu mongodb trả về kết quả sau khi write request được hoàn thành trên majority member hoặc trả về sau 5s. Majority members là phần member chiếm đa số trong một replica set. Ví dụ một replica set có 5 node thì majority

member sẽ bao gồm 1 primary và ít nhất 2 secondary ( Tổng số sẽ là 3 trên 5 ). Majority trong write concern đề cập sẽ luôn có primary vì write request luôn đập vào primary member trước tiên.

Từ vị trí application, nếu bạn không áp dụng write concern đủ thì có thể dẫn đến write set bị mất sau khi failover: Giả sử application dùng write concern mặc định nên kết quả trả về ngay khi primary xử lý xong write set nhưng có thể vì lý do nào đó, write set đó chưa được replicate sang secondary. Application không biết điều này. Không may, tại thời điểm đó, primary down, failover được thực hiện tự động, một secondary lên làm primary nhưng secondary này sẽ không có write set đó. Đó là cách dữ liệu bị mất khi dùng write concern không đầy đủ.

## 12. Export/Import dữ liệu

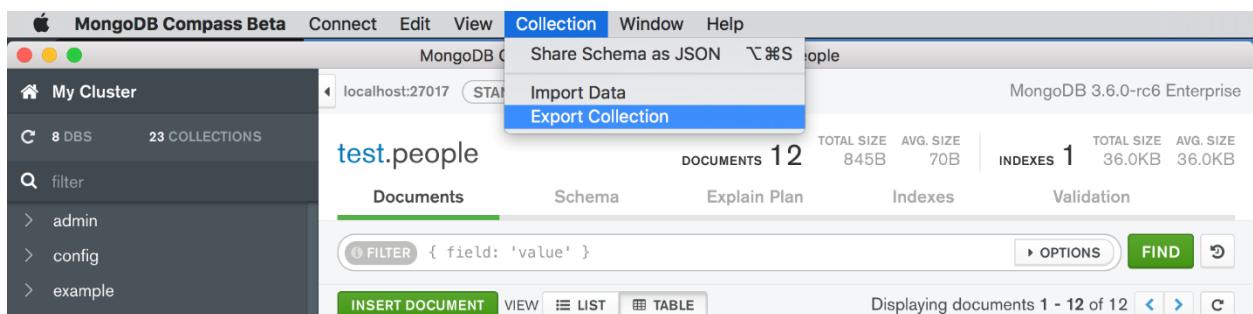
### 12.1. Thực hiện trên MongoDB Compass

#### a. Export dữ liệu:

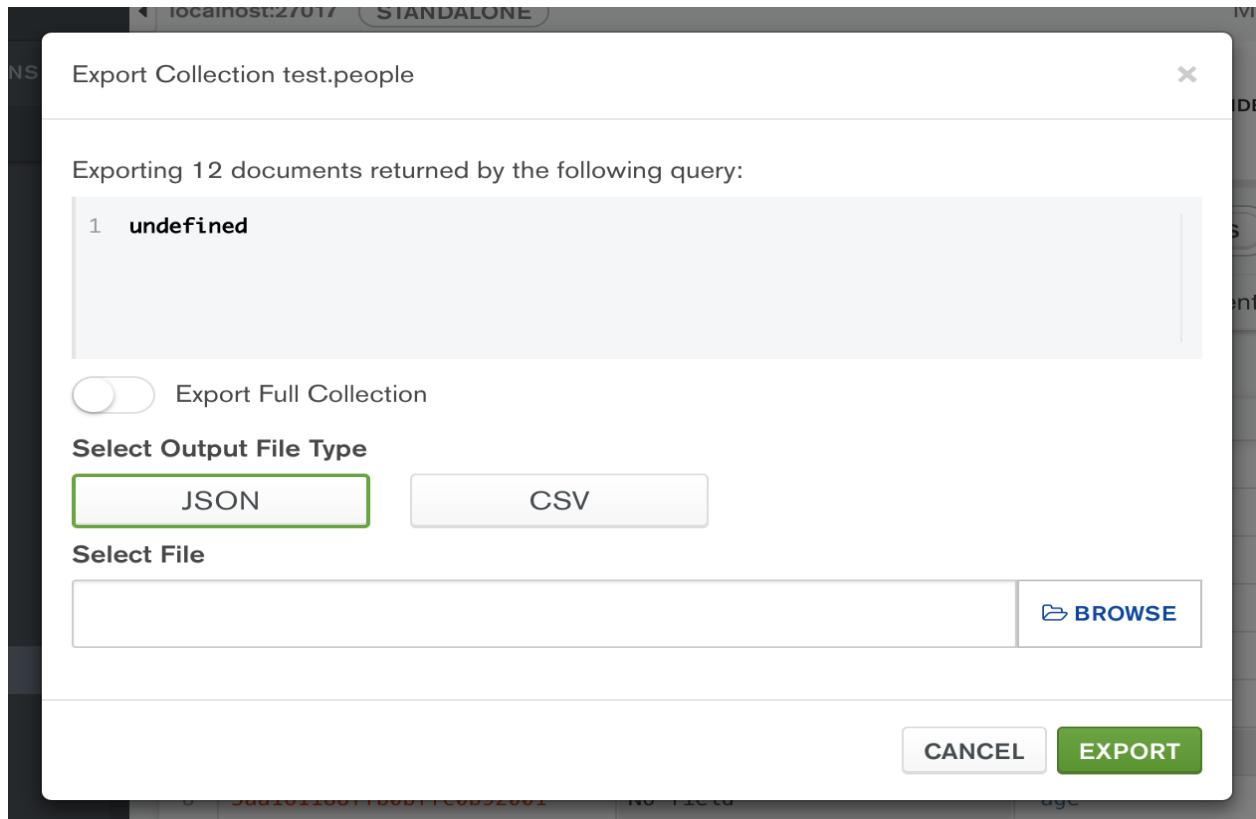
MongoDB Compass có thể export data từ Collection thành các file JSON và CSV.

Nếu người dùng chỉ định loại file xuất ra thì Compass chỉ có thể xuất tài liệu phù hợp với truy vấn đã chỉ định.

#### Các bước thực hiện theo hình ảnh:



- Màn hình sẽ xuất hiện tùy chọn:



- Chọn Export và bạn đã thực hiện xuất file thành công
- b. Import dữ liệu:

MongoDB Compass chỉ có thể import data từ file JSON hay CSV

❖ **Import data bằng file JSON:**

Khi Import data từ file JSON, phải chắc chắn mỗi tài liệu phải xuất hiện trên đúng hàng của nó. Không được sử dụng dấu câu (dấu phẩy) ở dòng cuối cùng để phân chia tài liệu.

EXAMPLE: The following .json file imports three documents:

```
{ "type": "home", "number": "212-555-1234" }
{ "type": "cell", "number": "646-555-4567" }
{ "type": "office", "number": "202-555-0182" }
```

❖ **Import data bằng file CSV:**

Khi Import data từ file CSV, dòng đầu tiên của file phải có dấu phẩy phân chia các tên trường. Các dòng sau đó phải có dấu phẩy ngăn cách giữa các giá trị của trường để có thể xác định được các giá trị tương ứng.

EXAMPLE: The following .csv file imports three documents:

```
name,age,fav_color,pet
```

```
Jeff,25,green,Bongo
Alice,20,purple,Hazel
Tim,32,red,Lassie
```

### Các bước thực hiện theo hình ảnh:

The screenshot shows the MongoDB Compass Beta interface. The top menu bar has 'Collection' highlighted. A dropdown menu is open under 'Collection' with 'Import Data' selected. The main window shows a cluster named 'My Cluster' with 8 DBs and 23 collections. A collection named 'test.people' is selected, displaying 12 documents. The interface includes tabs for 'Documents', 'Schema', 'Explain Plan', 'Indexes', and 'Validation'. Below the tabs are buttons for 'FILTER', 'OPTIONS', 'FIND', and 'INSERT DOCUMENT'. At the bottom, it says 'Displaying documents 1 - 12 of 12'.

- Màn hình sẽ xuất hiện tùy chọn:

The screenshot shows a modal dialog box titled 'Import To Collection test.people'. It has a title bar with 'localhost:27017 STANDALONE'. Inside, there's a section 'Select Input File Type' with 'JSON' selected and 'CSV' as an option. Below that is a 'Select File' section with a file input field and a 'BROWSE' button. At the bottom right are 'CANCEL' and 'IMPORT' buttons.

- Chọn Import và bạn đã thực hiện xuất file thành công

### 12.2. Thực hiện trên MongoDB Shell:

Để xem các tập tin mà MongoDB hỗ trợ thì nhập lệnh:

```
C:\Program Files\MongoDB\Server\4.2\bin>dir
```

```
C:\Program Files\MongoDB\Server\4.2\bin>dir
Volume in drive C is OS
Volume Serial Number is BCCA-D00B

Directory of C:\Program Files\MongoDB\Server\4.2\bin

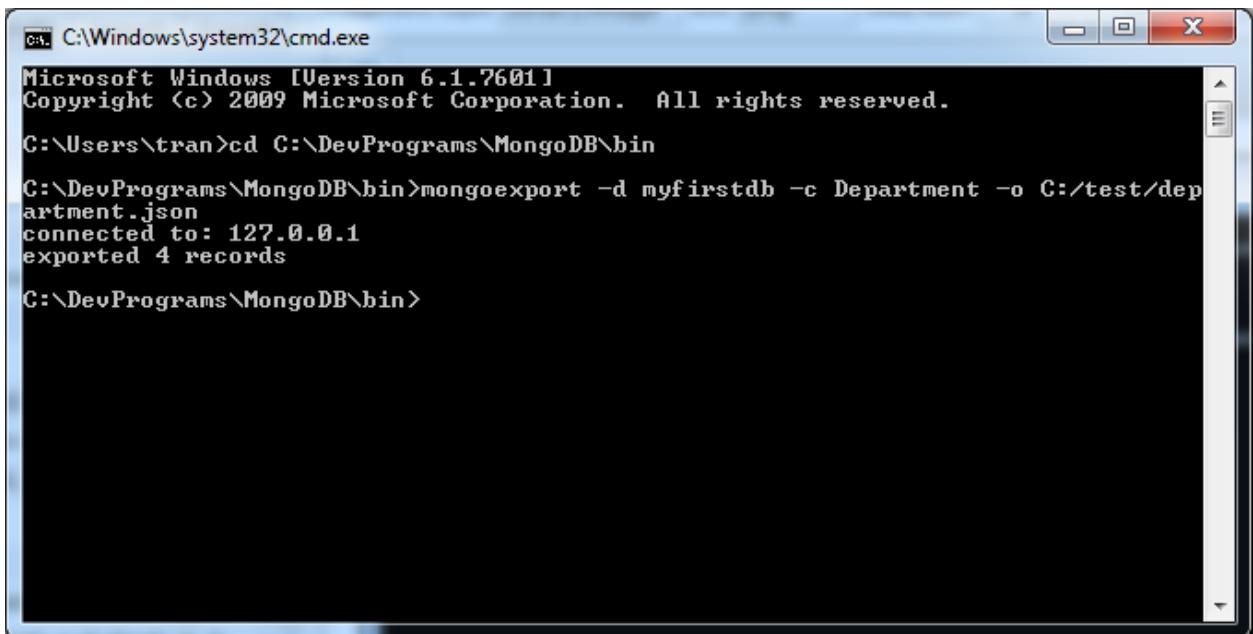
11/04/2019  06:10 PM    <DIR>          .
11/04/2019  06:10 PM    <DIR>          ..
10/16/2019  01:58 AM           9,469,952 bsondump.exe
10/16/2019  02:25 AM           1,568 InstallCompass.ps1
10/16/2019  02:12 AM           21,366,272 mongo.exe
11/04/2019  06:11 PM           616 mongod.cfg
10/16/2019  02:29 AM           35,551,232 mongod.exe
10/16/2019  02:29 AM           453,439,488 mongod.pdb
10/16/2019  01:58 AM           14,441,472 mongodump.exe
10/16/2019  01:58 AM           14,183,424 mongoexport.exe
10/16/2019  01:58 AM           14,138,880 mongofiles.exe
10/16/2019  01:58 AM           14,355,456 mongoimport.exe
10/16/2019  01:58 AM           14,760,960 mongorestore.exe
10/16/2019  02:17 AM           18,285,056 mongos.exe
10/16/2019  02:17 AM           234,115,072 mongos.pdb
10/16/2019  01:58 AM           13,880,832 mongostat.exe
10/16/2019  01:59 AM           13,602,304 mongotop.exe
               15 File(s)   871,592,584 bytes
                2 Dir(s)  91,682,635,776 bytes free
```

Các tập tin mà MongoDB hỗ trợ cho việc Import/Export

a. Export dữ liệu:

❖ **Export ra file JSON:**

**Cú pháp:** mongoexport -d database\_name -c collection\_name -o outfile.json



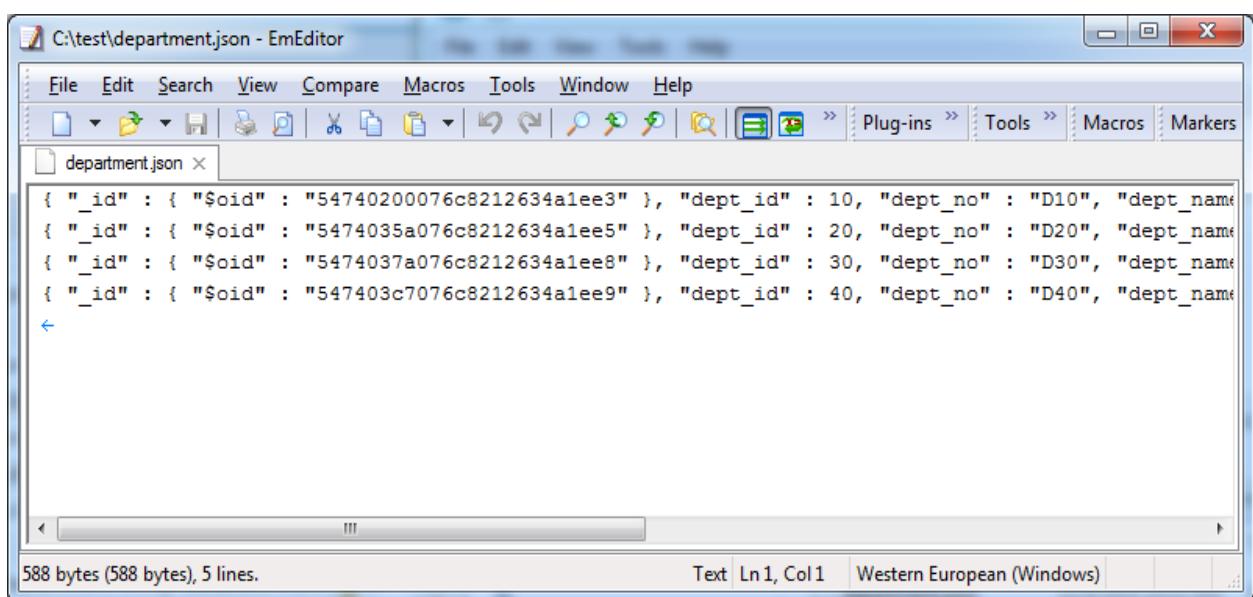
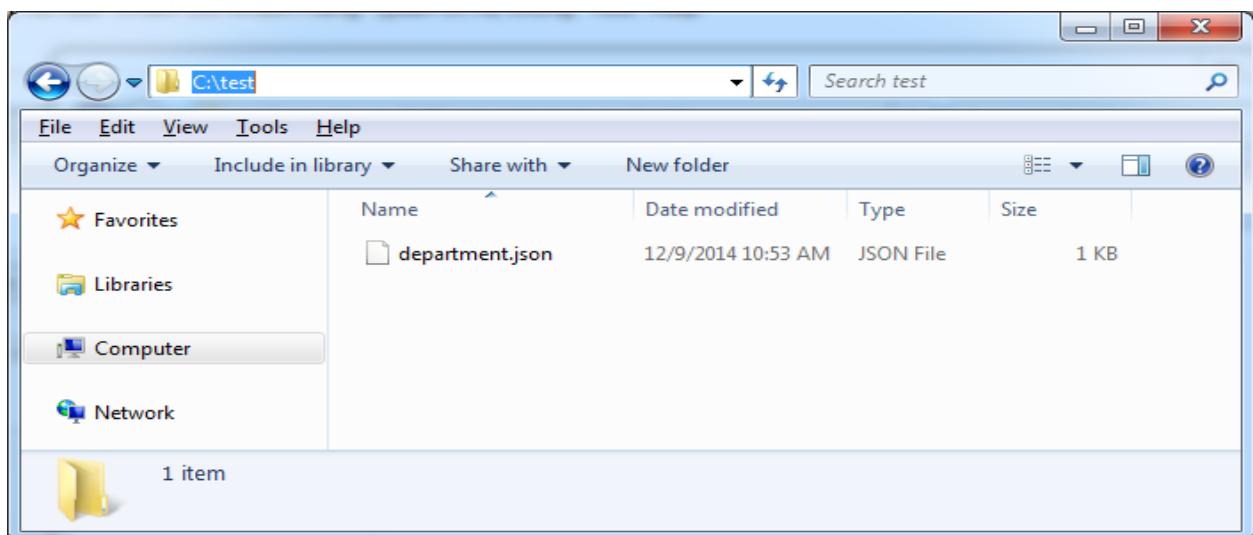
```
cmd C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\tran>cd C:\DevPrograms\MongoDB\bin

C:\DevPrograms\MongoDB\bin>mongoexport -d myfirstdb -c Department -o C:/test/department.json
connected to: 127.0.0.1
exported 4 records

C:\DevPrograms\MongoDB\bin>
```

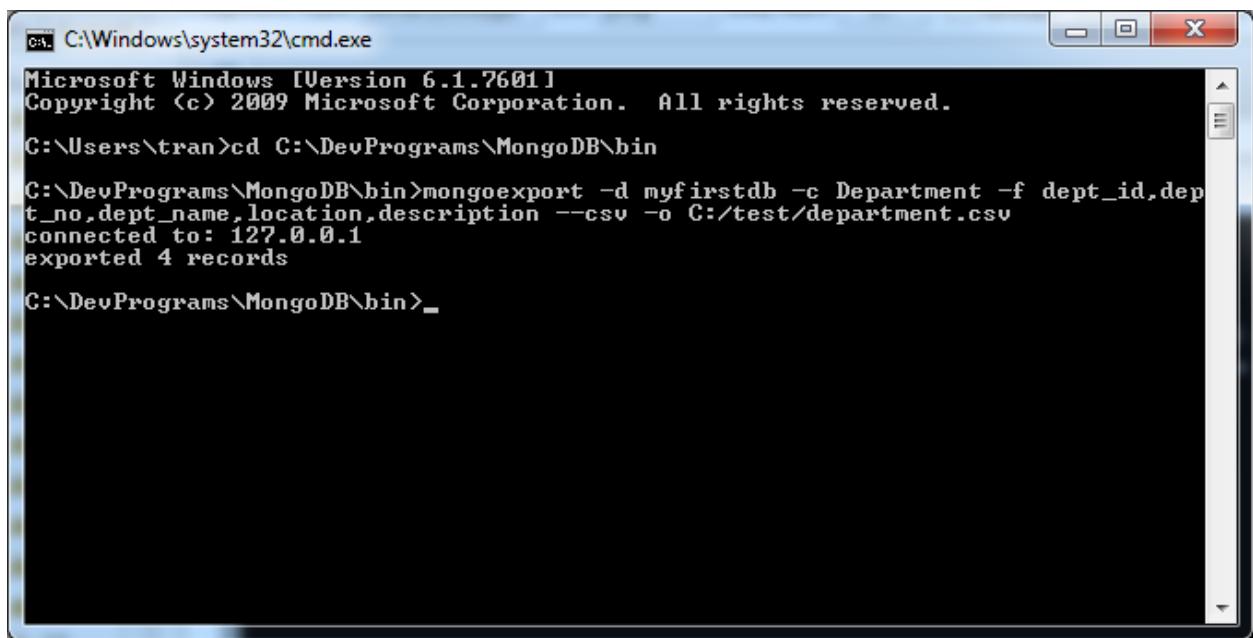
Kết quả sau khi export:



❖ **Export từ file CSV (Excel):**

**Cú pháp:**

```
mongoexport -d database_name -c collection_name -f  
column_1,column_2,column_3 --csv -o outfile.csv
```



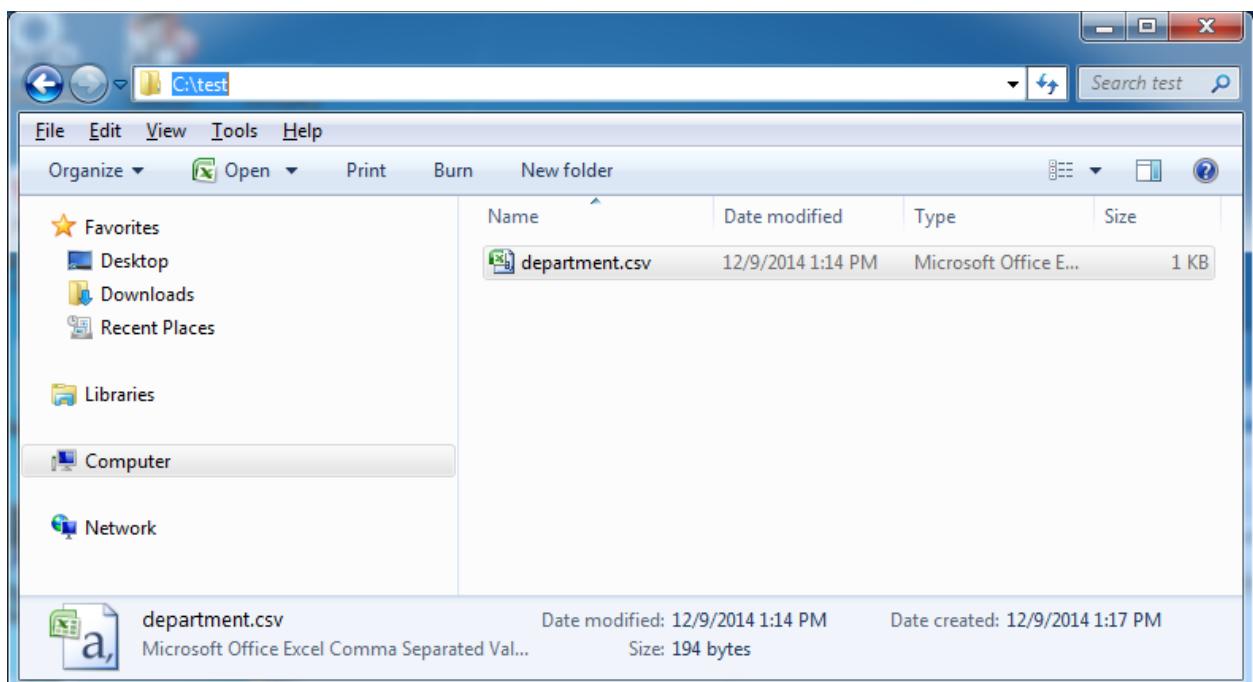
```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\tran>cd C:\DevPrograms\MongoDB\bin

C:\DevPrograms\MongoDB\bin>mongoexport -d myfirstdb -c Department -f dept_id,dept_no,dept_name,location,description --csv -o C:/test/department.csv
connected to: 127.0.0.1
exported 4 records

C:\DevPrograms\MongoDB\bin>_
```

Kết quả sau khi export:



	A	B	C	D	E	F	G
1	dept_id	dept_no	dept_name	location	description		
2	10	D10	ACCOUNTING	NEW YORK			
3	20	D20	RESEARCH	DALLAS	First department		
4	30	D30	SALES	CHICAGO			
5	40	D40	OPERATIONS	BOSTON			
6							
7							
8							
9							
10							
11							
12							

❖ **Export ra file (đuôi BSON và JSON):**

**Cú pháp:** mongodump -d database\_name -o outfile

❖ **Export ra file zip:**

**Cú pháp:** mongodump -d database\_name -o outfile.json – gzip

b. Import dữ liệu:

❖ **Import từ file JSON:**

**Cú pháp:** mongoimport -d database\_name -c collection\_name outfile.json

Kết quả:

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\tran>cd C:\DevPrograms\MongoDB\bin

C:\DevPrograms\MongoDB\bin>mongoimport -d myfirstdb -c Department2 C:/test/department.json
connected to: 127.0.0.1
2014-12-09T15:32:54.328+0700 imported 4 objects

C:\DevPrograms\MongoDB\bin>
```

❖ ***Import từ file CSV (Excel):***

**Cú pháp:** mongoimport -d database\_name -c collection\_name --type csv --file locations.csv --headerline

Kết quả:

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\tran>cd C:\DevPrograms\MongoDB\bin

C:\DevPrograms\MongoDB\bin>mongoimport -d myfirstdb -c Department3 --type csv --file C:/test/department.csv --headerline
connected to: 127.0.0.1
2014-12-09T15:50:13.300+0700 imported 4 objects

C:\DevPrograms\MongoDB\bin>
```

c. *Các lựa chọn khác:*

Trong trường hợp tổng quát bạn có các tùy chọn (option) để import/export liệt kê trong bảng dưới đây:

Option	Meaning
--help	produce help message

-v [ --verbose ]	be more verbose (include multiple times for more verbosity e.g. -vvvv)
-h [ --host ] arg	mongo host to connect to ("left,right" for pairs)
--port arg	server port. (Can also use --host hostname:port)
--sslCAFile=<filename>	Enables connection to a mongod or mongos that has TLS/SSL support enabled
--sslPEMKeyFile=<filename>	Specifies the .pem file that contains both the TLS/SSL certificate and key. Specify the file name of the .pem file using relative or absolute paths.
--sslPEMKeyPassword=<value>	Specifies the password to de-crypt the certificate-key file (i.e. --sslPEMKeyFile). Use the --sslPEMKeyPassword option only if the certificate-key file is encrypted. In all cases, the <b>mongoexport</b> will redact the password from all logging and reporting output.
--sslAllowInvalidCertificates	Bypasses the validation checks for server certificates and allows the use of invalid certificates. When using the allowInvalidCertificates setting, MongoDB logs as a warning the use of the invalid certificate.
--sslAllowInvalidHostnames	Disables the validation of the hostnames in TLS/SSL certificates. Allows <b>mongoexport</b> to connect to MongoDB instances even if the hostname in their certificates do not match the specified hostname.
-u [ --username ] arg	<p>Username</p> <p>NOTE</p> <p>You cannot specify both --username and --uri.</p>
-p [ --password ] arg	<p>Password</p> <p>NOTE</p>

	You cannot specify both --password and --uri.
--authenticationDatabase=<dbname>	Specifies the authentication database where the specified --username has been created. See Authentication Database.
--authenticationMechanism=<name>	<p><i>Default:</i> SCRAM-SHA-1</p> <p>Specifies the authentication mechanism the <b>mongoexport</b> instance uses to authenticate to the mongod or mongos.</p> <p><b>NOTE</b></p> <p>You cannot specify both --authenticationMechanism and --uri.</p>
--gssapiServiceName=<serviceName>	<p>Specify the name of the service using GSSAPI/Kerberos. Only required if the service does not use the default name of mongodb.</p> <p>This option is available only in MongoDB Enterprise.</p>
--gssapiHostName=<hostname>	<p>Specify the hostname of a service using GSSAPI/Kerberos. <i>Only</i> required if the hostname of a machine does not match the hostname resolved by DNS.</p> <p>This option is available only in MongoDB Enterprise.</p>
--dbpath arg	directly access mongod data files in the given path, instead of connecting to a mongod instance - needs to lock the data directory, so cannot be used if a mongod is currently accessing the same path
--directoryperdb	if dbpath specified, each db is in a separate directory
-d [ --db ] arg	database to use
-c [ --collection ] arg	collection to use (some commands)

--fieldFile arg	file with fields names - 1 per line
--query=<JSON>, -q=<JSON>	<p>Provides a query as a JSON document (enclosed in quotes) to return matching documents in the export.</p> <p>You must enclose the query document in single quotes ('{ ... }') to ensure that it does not interact with your shell environment.</p>
--out=<file>, -o=<file>	Specifies a file to write the export to. If you do not specify a file name, the mongoexport writes data to standard output (e.g. stdout).
--jsonFormat=<canonical relaxed>	<p><i>Default:</i> relaxed</p> <p>Modifies the output to use either canonical or relaxed mode of the MongoDB Extended JSON (v2) format.</p>
--pretty	<p><i>New in version 3.0.0.</i></p> <p>Outputs documents in a pretty-printed format JSON.</p>
--noHeaderLine	<p>By default, <b>mongoexport</b> includes the exported field names as the first line in a CSV output. --noHeaderLine directs <b>mongoexport</b> to export the data without the list of field names. --noHeaderLine is <b>only valid</b> with the --type option with value csv.</p>
--readPreference=<string document>	<p><i>Default:</i> primary</p> <p>Specifies the read preference for <b>mongoexport</b>. The --readPreference option can take</p>
--forceTableScan	<p>Forces mongoexport to scan the data store directly instead of traversing the _id field index. Use --forceTableScan to skip the index. Typically there are two cases where this behavior is preferable to the default:</p> <p>If you have key sizes over 800 bytes that would not be present in the _id index.</p> <p>Your database uses a custom _id field.</p>

	When you run with --forceTableScan, mongoexport may return a document more than once if a write operation interleaves with the operation to cause the document to move.
--skip=<number>	Use --skip to control where mongoexport begins exporting documents. See skip() for information about the underlying operation.
--limit=<number>	Specifies a maximum number of documents to include in the export. See limit() for information about the underlying operation.
--sort=<JSON>	Specifies an ordering for exported results. If an index does <b>not</b> exist that can support the sort operation, the results must be <i>less than</i> 32 megabytes.
--ignoreBlanks	if given, empty fields in csv and tsv will be ignored
--type arg	type of file to import. default: json (json,csv,tsv)
--file arg	file to import from; if not specified stdin is used
--drop	drop collection first
--headerline	CSV,TSV only - use first line as headers
--upsert	insert or update objects that already exist
--upsertFields arg	comma-separated fields for the query part of the upsert. You should make sure this is indexed.
--stopOnError	stop importing at the first error rather than continuing
--jsonArray	load a json array, not one item per line. Currently limited to 4MB.

### 13. Điều khiển truy xuất đồng thời

MongoDB cho phép nhiều người dùng có thể đọc và ghi dữ liệu đồng thời. Để đảm bảo tính nhất quán, nó sử dụng kỹ thuật khóa và biện pháp kiểm soát đồng thời để ngăn chặn người dùng sửa đổi cùng các dữ liệu. Cùng với nhau, các cơ chế đảm bảo

rằng tất cả thao tác ghi trên từng document riêng lẻ sẽ được cùng thực hiện hoặc không thể thực hiện và người dùng sẽ không thấy được chế độ không nhất quán của dữ liệu.

### **13.1. Các loại khóa được dùng trong MongoDB:**

MongoDB sử dụng khóa multi-granularity cho phép các hoạt động khóa ở cấp độ toàn cầu, cơ sở dữ liệu hoặc collection và cho phép các công cụ lưu trữ riêng lẻ thực hiện kiểm soát đồng thời của riêng mình dưới mức thu thập (ví dụ: ở cấp độ tài liệu trong WiredTiger)

MongoDB sử dụng khóa đọc-ghi cho phép người đọc đồng thời chia sẻ quyền truy cập đến tài nguyên.

Khóa (S) sẽ cấp cho việc đọc và khóa độc quyền (X) sẽ được cấp cho việc ghi, khóa intended lock (IS) và khóa intended exclusive (IX) cho ý định đọc hoặc ghi tài nguyên sử dụng khóa chi tiết. Khi khóa ở mức chi tiết nhất định, tất cả các cấp độ cao hơn bị khóa bằng khóa IS.

Ví dụ: Khi khóa collection cho thao tác ghi (sử dụng khóa X), toàn bộ khóa CSDL tương ứng và khóa toàn cục sẽ bị khóa lại ở chế độ intent exclusive (IX). Một CSDL đơn lẻ có thể đồng thời bị khóa trong chế độ IS và IX, nhưng khóa độc quyền (X) không thể cùng tồn tại với bất kỳ chế độ khóa nào, khóa chia sẻ (S) có thể cùng tồn tại với khóa intent shared (IS).

Các khóa cho dù ghi hay đọc đều phải xếp hàng theo thứ tự. Tuy nhiên, để tối ưu hóa, khi một yêu cầu cấp khóa được chấp nhận, tất cả các yêu cầu tương thích được cấp quyền cùng lúc, phải giải phóng trước khi yêu cầu xung đột.

Ví dụ: xem xét trường hợp khóa X vừa được cấp, và hàng đợi xung đột chứa các mục sau: IS → IS → X → X → S → IS

Theo thứ tự, chỉ có 2 khóa IS được cấp. Thay vào đó, MongoDB sẽ cấp quyền cho tất cả các khóa IS và S, khi chúng hoàn toàn giải phóng thì khóa X sẽ được cấp quyền, cho dù khóa IS hay S mới yêu cầu trong cùng thời gian.

❖ Tóm tắt các loại khóa:

<b>Lock Mode</b>	<b>Description</b>
R	Represents Shared (S) lock.
W	Represents Exclusive (X) lock.
r	Represents Intent Shared (IS) lock.
w	Represents Intent Exclusive (IX) lock.

### **13.2. Làm cách nào để xin quyền cấp khóa trên MongoDB?**

Cho việc đọc và ghi, WiredTiger sử dụng cơ chế điều khiển đồng thời tự động. WiredTiger chỉ sử dụng khóa intent ở cấp độ toàn cầu, database và collection. Khi công cụ lưu trữ phát hiện ra sự xung đột giữa hai giao tác, một sẽ dừng lại thao tác ghi và sẽ thực hiện lại sau

Ở một vài hoạt động cấp toàn cầu, điển hình là các hoạt động ngắn hạn liên quan đến nhiều databases, nó vẫn yêu cầu khóa toàn cầu “tổng bộ đối tượng”. Trong các hoạt động khác, như xóa một collection, nó vẫn yêu cầu một khóa độc quyền trên database

### **13.3. Làm cách nào để xem trạng thái của các khóa?**

Sử dụng các phương thức sau để xem:

- db.serverStatus(),
- db.currentOp(),
- mongotop,
- mongostat, and/or
- the MongoDB Cloud Manager

### **13.4. Một thao tác đọc/ghi đều cần cấp khóa?**

Trong một vài trường hợp, đọc và ghi có thể mang khóa.

Các hoạt động đọc/ghi lâu dài, như truy vấn, cập nhật, xóa với nhiều điều kiện. MongoDB có thể cấp khóa cho những việc sửa đổi tài liệu ảnh hưởng đến nhiều document như việc update() với nhiều tham số đầu vào.

Với công cụ hỗ trợ lưu trữ hỗ trợ truy xuất đồng thời, như WiredTiger, việc cấp khóa là không cần thiết khi truy cập vào bộ lưu trữ vì khóa intent locks đã được tạo ở cấp toàn cầu, database và collection, không chặn các thao tác đọc hay ghi khác. Tuy nhiên các hoạt động cần cấp khóa, chẳng hạn như:

- Để tránh lưu trữ các giao tác thực hiện lâu dài bởi vì chúng yêu cầu giữ một lượng lớn dữ liệu trong bộ nhớ
- Để phục vụ như các điểm gián đoạn để có thể bỏ các giao tác lâu dài
- Cho phép các hoạt động yêu cầu tiến trình độc quyền cho collection như tạo index, xóa và tạo collection

### **13.5. Những khóa nào người dùng có thể tạo?**

<b>Operation</b>	<b>Database</b>	<b>Collection</b>
Issue a query	r (Intent Shared)	r (Intent Shared)
Insert data	w (Intent Exclusive)	w (Intent Exclusive)
Remove data	w (Intent Exclusive)	w (Intent Exclusive)
Update data	w (Intent Exclusive)	w (Intent Exclusive)
Perform Aggregation	r (Intent Shared)	r (Intent Shared)
Create an index (Foreground)	W (Exclusive)	
Create an index (Background)	w (Intent Exclusive)	w (Intent Exclusive)
List collections	r (Intent Shared) <i>Changed in version 4.0.</i>	
Map-reduce	W (Exclusive) and R (Shared)	w (Intent Exclusive) and r (Intent Shared)

### **13.6. Những cấp người dùng nào có thể khóa CSDL?**

Các hoạt động quản trị sau đây cần có khóa độc quyền trên CSDL trong thời gian dài:

- cloneCollectionAsCapped
- collMod
- compact
- convertToCapped

Các hoạt động quản trị sau có thể tạo khóa CSDL nhưng chỉ giữ khóa được trong thời gian ngắn:

<b>Commands</b>	<b>Methods</b>
authenticate	db.auth()
createUser	db.createUser()
getLastError	db.getLastError()
isMaster	db.isMaster()
replsetgetStatus	rs.Status()
ServerStatus	db.serverStatus()

### **13.7. Hoạt động quản trị nào có thể tạo khóa trên collection?**

Các hoạt động quản trị sau đây yêu cầu tạo khóa độc quyền ở cấp độ collection:

<b>Commands</b>	<b>Methods</b>
create	db.createCollection() db.createView()
createIndexes	db.collection.createIndex() db.collection.createIndexes()
drop	db.collection.drop()

dropIndexes	db.collection.dropIndex() db.collection.dropIndexes()
renameCollection	db.collection.renameCollection()

### 13.8. **Hoạt động của MongoDB có khóa nhiều hơn một CSDL?**

Những hoạt động của MongoDB giữ khóa độc quyền toàn cầu:

- db.copyDatabase() và db.collection.reIndex() chứa khóa độc quyền toàn cầu (W) và sẽ chặn tất cả các hoạt động khác cho đến khi nó hoàn thành.
- Người dùng xác thực yêu cầu khóa đọc trên CSDL của quản trị viên. Khi đang triển khai lược đồ, khóa xác thực sẽ khóa CSDL của quản trị viên và tất cả CSDL đang chạy của người dùng.
- Các tài khoản chuyển đổi thành viên cũng có khóa độc quyền toàn cầu.

## 14. Xử lý giao tác

### 14.1. **Tổng quan:**

MongoDB cung cấp các **Atomic Operation (hoạt động nguyên tử)** trên một Document đơn. Vì người dùng có thể sử dụng các tài liệu và mảng nhúng để nắm bắt các mối quan hệ giữa dữ liệu trong cấu trúc document riêng lẻ, không cần quan tâm đến các mối quan hệ với các document khác.

Vì thế, nếu một Document có hàng trăm trường, thì lệnh update sẽ hoặc cập nhật tất cả các trường đó hoặc không cập nhật bất cứ trường nào, vì thế duy trì tính Atomicity tại cấp độ Document.

Hiện tại MongoDB cũng đã cung cấp Atomic Operation trên nhiều Document và cho phép thực hiện giao tác trên nhiều database, collection, document.

### 14.2. **Cấu trúc:**

```
// Prereq: Create collections. CRUD operations in transactions must be
on existing collections.

db.getSiblingDB("mydb1").foo.insert( {abc: 0}, { writeConcern: { w: "ma
jority", wtimeout: 2000 } } );
db.getSiblingDB("mydb2").bar.insert( {xyz: 0}, { writeConcern: { w: "ma
jority", wtimeout: 2000 } } );

// Start a session.
```

```

session = db.getMongo().startSession( { readPreference: { mode: "primary" } } );

coll1 = session.getDatabase("mydb1").foo;
coll2 = session.getDatabase("mydb2").bar;

// Start a transaction
session.startTransaction( { readConcern: { level: "local" }, writeConcern: { w: "majority" } } );

// Operations inside the transaction
try {
    coll1.insertOne( { abc: 1 } );
    coll2.insertOne( { xyz: 999 } );
} catch (error) {
    // Abort transaction on error
    session.abortTransaction();
    throw error;
}

// Commit the transaction using write concern set at transaction start
session.commitTransaction();

session.endSession();

```

### **13.3. Tính ACID của giao tác:**

a. Giao tác trên nhiều document có tính nguyên tố: **Atomicity**

Khi một giao tác được thực hiện, tất cả các thay đổi trên giao tác được lưu và được thực hiện trên document. Điều đó có nghĩa, một giao tác sẽ không được thực hiện và hiển thị ra bên ngoài nếu có một thao tác không thỏa điều kiện.

Cho đến khi giao tác được thực hiện, dữ liệu thay đổi trong giao tác sẽ không được hiển thị bên ngoài.

Tuy nhiên, khi một giao tác ghi trên nhiều phân đoạn, không phải tất cả các hoạt động đọc bên ngoài phải chờ đợi kết quả thực thi của giao tác trong mỗi phân đoạn.

Ví dụ: Nếu giao tác được thực hiện và ghi 1 được hiển thị trên phân đoạn A nhưng giao tác ghi 2 chưa được hiển thị trên phân đoạn B, bên ngoài giao tác sẽ đọc được kết quả của ghi 1 mà không đọc của ghi 2

Khi một giao tác bị hủy bỏ, tất cả sự thay đổi dữ liệu được thực hiện bởi giao tác sẽ không được hiển thị

Ví dụ: Nếu hoạt động nào của giao tác thất bại, cả giao tác sẽ bị từ chối và tất cả sự thay đổi sẽ bị bỏ đi và không được hiển thị

Cho các giao tác cần tính nguyên tố trong việc đọc/ghi trên nhiều document (trong một hay nhiều collection), MongoDB hỗ trợ các giao tác trên nhiều document:

- Trong phiên bản 4.0, MongoDB hỗ trợ cho việc tạo giao tác trên nhiều document trên các bộ bản sao.
- Trong phiên bản 4.2, MongoDB cung cấp giao tác phân tán, bổ sung hỗ trợ cho các giao tác đa tài liệu trên các cụm bị phân tách và kết hợp hỗ trợ với giao tác đa tài liệu trên các bản sao. Để sử dụng các giao tác này, người dùng phải sử dụng MongoDB driver cho phiên bản MongoDB 4.2

b. Giao tác trên nhiều document có tính nhất quán: Consistency

MongoDB không đáp ứng được tính chất này, do khi một giao tác ghi trên nhiều phân đoạn, không phải tất cả các hoạt động đọc bên ngoài phải chờ đợi kết quả thực thi của giao tác trong mỗi phân đoạn.

c. Giao tác trên nhiều document có tính độc lập: Isolation

Đảm bảo rằng một giao dịch có khả năng làm việc độc lập và không liên quan đến nhau.

d. Giao tác trên nhiều document có tính bền vững: Durability

Đảm bảo rằng cơ sở dữ liệu sẽ theo dõi các thay đổi cấp phát trong một cách mà các máy chủ có thể phục hồi từ một sự kết thúc bất thường. Tính chất này đảm bảo rằng trong trường hợp thất bại hay dịch vụ khởi động lại các dữ liệu có sẵn trong trước khi gặp lỗi.

## 15. Truy cập hệ thống/quản lý tài khoản

Mặc định, authentication bị disable nên bạn có thể truy cập đến mongodb từ localhost lẫn remote host và có toàn bộ mọi quyền thao tác mà không có cản trở nào cả. Khi authentication được enable thì vẫn còn cơ chế localhost exception: mongodb cho phép connection đến từ localhost và có toàn bộ quyền thao tác mà không cần authentication gì cả. Ngoại lệ này áp dụng với các truy cập từ localhost và khi không có user nào được định nghĩa trong mongodb. Có thể loại bỏ ngoại lệ này

bằng cách set enabledLocalhostAuthBypass về giá trị 0. Mặc định tham số này có giá trị 1

Để enable authentication bạn thêm dòng config sau:

```
security:  
    authorization: enabled
```

Lúc này những truy cập từ remote host vẫn hoạt động được nhưng lại không có quyền thực hiện bất cứ hành động gì trên db:

```
> show dbs  
listDatabases failed:{  
    "ok" : 0,  
    "errmsg" : "not authorized on admin to execute command { listDatabases: 1.  
0 }",  
    "code" : 13  
} at src/mongo/shell/mongo.js:47
```

Dựa vào localhost exception, bạn có thể truy cập mongodb từ localhost với toàn quyền để tạo user

Đầu tiên, bạn cần tạo một system user admin

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

Login thử vào user siteUserAdmin:

```
mongo -u siteUserAdmin -p password admin  
MongoDB shell version: 2.6.10  
connecting to: admin  
>
```

Sử dụng siteUserAdmin, bạn có thể tạo thêm các user khác nữa:

```
use development

db.createUser(
{
    user: "developer",
    pwd: "12345678",
    roles: [
        { role: "readWrite", db: "test" },
        { role: "read", db: "product" },
        { role: "readWrite", db: "development" }
    ]
}
)
```

Login thử vào user developer đó:

```
mongo -u developer -p 12345678 development
MongoDB shell version: 2.6.10
connecting to: development
>
```

Sau khi tạo user xong, bạn cần loại bỏ localhost exception bằng cách thêm dòng config sau:

```
setParameter:
enableLocalhostAuthBypass: 0
```

Bạn có thể kiểm tra thông tin xác thực của user qua db.auth(). Ví dụ:

```
use development;
db.auth("developer", "12345678");
```

Hoặc có thể lấy ra thông tin về user hiện tại

```
db.runCommand({connectionStatus : 1})
{
    "authInfo" : {
        "authenticatedUsers" : [
            {
                "user" : "developer",
```

```

        "db" : "development"
    }
]
},
"ok" : 1
}

```

- ❖ Tổng hợp các lệnh được hỗ trợ:

Name	Description	Syntax
db.auth()	Chứng thực user cho database	db.auth( <username>, passwordPrompt() ) // Or db.auth( <username>, <password> )
db.changeUserPassword()	Thay đổi mật khẩu cho user	db.changeUserPassword(username, password)
db.createUser()	Tạo user mới	db.createUser({     user: "<name>",     pwd: passwordPrompt(), // Or "<cleartext password>"     customData: { <any information> },     roles: [         { role: "<role>", db: "<database>" }   "<role>",         ...     ] })

db..dropUser()	Xóa user	db.dropUser(username, writeConcern)
db..dropAllUser()	Xóa tất cả các user có trong database	db.dropAllUsers(writeConcern)
db.getUser()	Trả về thông tin của user được chỉ định	db.getUser( "<username>", { showCredentials: <Boolean>, showPrivileges: <Boolean>, showAuthenticationRestrictions: <Boolean>, filter: <document> } )
db.getUsers()	Trả về thông tin của toàn bộ user trong database	db.getUsers( { showCredentials: <Boolean>, filter: <document> } )
db.grantRolesToUser()	Gán quyền thực hiện một role cho user	db.grantRolesToUser( "<username>", [ <roles> ], { <writeConcern> } )
db.removeUser()	Xóa user khỏi database nếu user đó không dùng nữa	db.removeUser(username)
db.revokeRolesFromUser()	Xóa quyền thực hiện role của user	db.revokeRolesFromUser( "<username>", [ <roles> ], { <writeConcern> } )

		<pre>db.updateUser(     "&lt;username&gt;",     {         customData : { &lt;any information&gt; },         roles : [             { role: "&lt;role&gt;", db:                 "&lt;database&gt;" }   "&lt;role&gt;",             ...         ],         pwd: passwordPrompt(),         // Or "&lt;cleartext password&gt;"         authenticationRestrictions: [             {                 clientSource: ["&lt;IP&gt;"                       "&lt;CIDR range&gt;", ...],                 serverAddress: ["&lt;IP&gt;",                     "&lt;CIDR range&gt;", ...]             },             ...         ],         mechanisms: [ "&lt;SCRAM-             SHA-1 SCRAM-SHA-256&gt;", ... ],         passwordDigestor:         "&lt;server client&gt;"     },     writeConcern: { &lt;write         concern&gt; } )</pre>
db.updateUser()	Cập nhật dữ liệu của user	

passwordPrompt()	Nhắc nhở mật khẩu thay thế cho việc chỉ định mật khẩu xác thực	db.createUser( { user:"user123", pwd: passwordPrompt(), // Instead of specifying the password in cleartext roles:[ "readWrite" ] } )
------------------	--	--

## 16. Tạo Role, phân quyền

MongoDB hỗ trợ các lệnh thao tác các role:

Name	Description	Syntax
db.createRole()	Tạo role và chỉ định quyền cho nó	db.createRole( {role: "<name>", privileges: [ { resource: { <resource> }, actions: [ "<action>", ... ] } ], ... } )
db.dropRole()	Xóa một role của người dùng	db.dropRole(rolename, writeConcern)
db.dropAllRoles()	Xóa toàn bộ các role liên kết với database  <u>Note:</u> lệnh này sẽ xóa toàn bộ các role của người dùng khỏi database	db.dropAllRoles(writeConcern)
db.getRole()	Trả về thông tin của một role được chỉ định	db.getRole(rolename, args)
db.getRoles()	Trả về thông tin của tất cả các role trong database	db.getRoles(<field: type>)

db.grantPrivileges ToRole()	Gán quyền cho người dùng đã xác định role	db.grantPrivilegesToRole( "< rolename >", [ { resource: { <resource> }, actions: [ "<action>", ... ] }, ... ], { < writeConcern > } )
db.revokePrivileges FromRole()	Thu hồi quyền của người dùng đã xác định role	db.revokePrivilegesFromRole( "<rolename>", [ { resource: { <resource> }, actions: [ "<action>", ... ] }, ... ], { <writeConcern> } )
db.grantRoles ToRole()	Chỉ định các role cho role mà người dùng đã xác định thực hiện	db.grantRolesToRole( "<rolename>", [ <roles> ], { <writeConcern> } )
db.revokeRoles FromRole()	Thu hồi các role khỏi các role mà người dùng đã xác định thực hiện	db.revokeRolesFromRole( "<rolename>", [ <roles> ], { <writeConcern> } )
db.updateRole()	Cập nhật role của người dùng	db.updateRole( "<rolename>", { privileges: [

```

    { resource: { <resource> },
  actions: [ "<action>", ... ] },

  ...
  ],
  roles:
  [
    {
      { role: "<role>", db:
        "<database>" } | "<role>",

      ...
    ],
    authenticationRestrictions:
    [
      {
        clientSource: ["<IP>" |
          "<CIDR range>", ...],
        serverAddress: ["<IP>", |
          "<CIDR range>", ...]

      },
      ...
    ]
  },
  { <writeConcern> }
)

```

## 17. Lưu trữ, phục hồi dữ liệu

### 17.1. Định nghĩa

- **Storage engine** là phương thức quản lý dữ liệu chính của MongoDB. MongoDB cung cấp các storage engines rất đa dạng, cho phép bạn lựa chọn phương thức phù hợp.
- **The Journal** là một bản ghi giúp phục hồi database trong trường hợp tắt máy phần cứng. Có một số tùy chọn cấu hình cho phép The Journal đạt được

sự cân bằng giữa hiệu suất và độ tin cậy phù hợp với công việc của người dùng.

- **GridFS** là hệ thống lưu trữ đa năng phù hợp với việc quản lý một lượng lớn file, có thể lưu trữ document có kích thước 16MB.

### ***Storage engine:***

Đây là phương thức quản lý lưu trữ dữ liệu, trên cả phần cứng lẫn phần mềm. MongoDB hỗ trợ rất nhiều Storage engine, mỗi engine sẽ đưa đến hiệu suất làm việc tốt nhất cho từng loại công việc.

Có 2 loại Storage engine chính là :

- a. WiredTiger Storage Engine (mặc định):

Phiên bản: WiredTiger Storage Engine được đưa làm mặc định kể từ phiên bản MongoDB 3.2.

### **Khả năng truy xuất đồng thời:**

- WiredTiger Storage Engine cho phép truy xuất đồng thời các tài liệu cho các hoạt động ghi. Do đó nhiều người dùng có thể cùng lúc sửa đổi các document khác nhau trong một collection cùng một lúc
- Đối với hoạt động đọc và ghi, WiredTiger sử dụng điều khiển đồng thời tự động. Nó chỉ sử dụng khóa mục đích ở cấp độ toàn cầu, database và collection. Khi công cụ lưu trữ phát hiện sự xung đột giữa hai hoạt động, một sẽ phát sinh xung đột ghi và hành động đó sẽ được thực hiện sau (ta sẽ nói về truy xuất đồng thời rõ hơn ở phần xử lý truy xuất đồng thời)
- Với hoạt động toàn cầu, thường là các hoạt động tồn tại trong thời gian ngắn liên quan đến nhiều CSDL, vẫn sẽ yêu cầu khóa toàn cầu. Một số hoạt động khác, chẳng hạn như việc xóa collection, vẫn yêu cầu khóa ghi (Exclusive Lock)

### **Snapshots and Checkpoints:**

- WiredTiger sử dụng điều khiển đồng thời MultiVersion (MVCC). Khi bắt đầu một hoạt động, WiredTiger cung cấp ảnh chụp nhanh (Snapshot) của dữ liệu tại ngay thời điểm thực hiện hoạt động. Ảnh chụp nhanh (Snapshot) thể hiện chế độ xem phù hợp của dữ liệu trong bộ nhớ.
- Khi ghi trên đĩa cứng, WiredTiger ghi tất cả dữ liệu trong ảnh chụp nhanh vào đĩa theo cách nhất quán trên tất cả các file. Dữ liệu bền vững (dữ liệu vẫn được ghi vào máy chủ khi có sự cố) sẽ hoạt động như một điểm kiểm tra (checkpoint) trong các file data. Checkpoint đảm bảo các file data phù hợp bao gồm cả checkpoint cuối cùng; tức là checkpoint có thể đóng vai trò là checkpoint khôi phục.

- Kể từ phiên bản 3.6, MongoDB cấu hình WiredTiger tạo ra checkpoint (ghi ảnh chụp nhanh dữ liệu vào ổ đĩa) trong khoảng 60s. Trong phiên bản cũ hơn, MongoDB cài đặt checkpoint trong WiredTiger trên dữ liệu của người dùng trong khoảng 60s hoặc 2GB của dữ liệu hành trình (dữ liệu hiện thời) đang ghi, miễn là cái nào xuất hiện trước
- Trong suốt quá trình ghi của Checkpoint mới, Checkpoint cũ vẫn còn hiệu lực. Như vậy, kể cả khi MongoDB thay thế hoặc gặp một lỗi nào đó trong quá trình ghi Checkpoint mới, nó vẫn có thể khôi phục nhờ vào checkpoint trước đó.
- Checkpoint mới sẽ có thể được truy cập và sử dụng vĩnh viễn khi bảng WiredTiger's metadata được cập nhật độc lập tham chiếu đến Checkpoint mới. Khi Checkpoint mới được truy cập, WiredTiger sẽ giải phóng các trang khỏi Checkpoint cũ.
- Khi sử dụng WiredTiger, kể cả khi không có Journaling, MongoDB có thể phục hồi Checkpoint trước đó; tuy nhiên, để phục hồi sự thay đổi đã thực hiện sau Checkpoint trước, chạy nó cùng với Journaling.

#### **Journal:**

- WiredTiger sử dụng Journal kết hợp với các Checkpoint để đảm bảo độ bền dữ liệu.
- The WiredTiger journal vẫn giữ lại toàn bộ sự thay đổi dữ liệu giữa các Checkpoints. Nếu MongoDB thoát giữa các Checkpoint, nó sẽ sử dụng Journal để phát lại các dữ liệu đã được sửa đổi kể từ lần Checkpoint trước đó.
- WiredTiger journal được nén bởi thư viện snappy. Để chỉ định thư viện nén khác hoặc không nén, sử dụng chế độ storage.wiredTiger.engineConfig.journalCompressor
- **Note:** Nếu một bản ghi nhỏ hơn hoặc bằng 128 byte (kích thước bản ghi tối thiểu cho WiredTiger), WiredTiger không nén bản ghi đó.
- Người dùng có thể vô hiệu hóa Journal cho các trường hợp độc lập bằng chế độ storage.journal.enabled là false, điều này có thể giảm thiểu chi phí cho Journal. Trong trường hợp độc lập, nghĩa là không sử dụng Journal, khi thoát khỏi hệ thống một cách đột ngột, dữ liệu sửa đổi trong Checkpoint trước có thể mất bất cứ lúc nào.
- ❖ **Note:** Trong phiên bản 4.0, người dùng không thể chỉ định –nojournal hoặc storage.journal.enabled: false khi sử dụng WiredTiger để lưu trữ

#### **Nén tập tin:**

- Với WiredTiger, MongoDB hỗ trợ nén cho tất cả collection và index. Việc nén tập tin giúp giảm thiểu sử dụng tài nguyên lưu trữ của CPU
- Theo mặc định, WiredTiger sử dụng nén khối với thư viện nén snappy cho collections và thư viện nén prefix cho index

- Với collection, thư viện nén có thể nén ở các loại:
  - Zlib
  - Zstd
- Để chỉ định các thuật toán nén khác hoặc không nén, sử dụng chế độ storage.wiredTiger.collectionConfig.blockCompressor
- Với index, để vô hiệu hóa nén prefix, sử dụng chế độ storage.wiredTiger.indexConfig.prefixCompression

### **Sử dụng bộ nhớ:**

- Với WiredTiger, MongoDB sử dụng cả bộ đệm trong của WiredTiger và bộ đệm của hệ thống tập tin.
  - Kể từ phiên bản 3.4, kích thước bộ đệm trong của WiredTiger mặc định là lớn hơn một trong hai:
    - 50% của (RAM - 1GB)
    - 256 MB
  - ❖ **Ví dụ:** trong hệ thống RAM chứa 4GB thì bộ nhớ đệm của WiredTiger sẽ chiếm 1.5GB ( $0.5 * (4GB - 1GB) = 1.5GB$ ) Ngược lại, một hệ thống có tổng cộng 1,25 GB RAM sẽ phân bổ 256 MB cho bộ đệm WiredTiger vì đó là hơn một nửa tổng số RAM trừ đi một GB ( $0.5 * (1.25 GB - 1GB = 128 MB < 256 MB)$ )
  - ❖ **Note:**
    - Để xem giới hạn của bộ nhớ, vào hostInfo.system.memLimitMB.
    - Để điều chỉnh kích thước của bộ đệm trong WiredTiger storage.wiredTiger.engineConfig.cacheSizeGB và --wiredTigerCacheSizeGB
- b. In-Memory Storage Engine (xem thêm trong phiên bản Enterprise): GridFS

### **Định nghĩa:**

- GridFS là một đặc tả để lưu trữ và truy xuất các tệp vượt quá giới hạn kích thước tài liệu BSON là 16 MB
- Thay vì lưu trữ một file là một tài liệu riêng lẻ, GridFS chia các file thành nhiều phần hoặc các đoạn, và lưu trữ từng đoạn dưới dạng từng tài liệu riêng biệt.

### **GridFS collection:**

- Lưu trữ dữ liệu ở hai collections:
  - Chunks lưu trữ các đoạn nhị phân
  - Files lưu trữ các metadata của file
- GridFS đặt các collection vào một nhóm chung bằng cách thêm tiền tố vào mỗi nhóm với tên nhóm. Theo mặc định, GridFS sử dụng hai collection với nhóm có tên fs:
  - fs.files
  - fs.chunks

- The Chunk collection:

Cấu trúc của một chunk collection:

```
{
  "_id" : <ObjectId>,
  "files_id" : <ObjectId>,
  "n" : <num>,
  "data" : <binary>
}
```

- The files collection:

Cấu trúc của một file collection:

```
{
  "_id" : <ObjectId>,
  "length" : <num>,
  "chunkSize" : <num>,
  "uploadDate" : <timestamp>,
  "md5" : <hash>,
  "filename" : <string>,
  "contentType" : <string>,
  "aliases" : <string array>,
  "metadata" : <any>,
}
```

### GridFS Indexes:

GridFS sử dụng index trên mỗi chunks và files collection để đạt hiệu quả. Các trình điều khiển thiết lập với đặc tả GridFS sẽ tự động tạo các chỉ mục này để thuận tiện. Bạn cũng có thể tạo bất kỳ chỉ mục bổ sung nào theo ý muốn cho phù hợp với nhu cầu ứng dụng của mình.

- The chunks Index: GridFS sử dụng một chỉ mục hỗn hợp duy nhất trên bộ sưu tập chunk bằng cách sử dụng các trường files\_id và n. Điều này cho phép truy xuất các chunk hiệu quả, như được minh họa trong ví dụ sau:

```
db.fs.chunks.find( { files_id: myFileID } ).sort( { n: 1 } )
```

- The files Index: GridFS sử dụng một chỉ mục trên collection của file bằng cách sử dụng các trường tên file và uploadDate. Chỉ mục này cho phép truy xuất các tệp hiệu quả, như trong ví dụ này:

```
db.fs.files.find( { filename: myFileName } ).sort( { uploadDate: 1 } )
```

## 18. Cơ chế bảo mật

MongoDB cung cấp rất nhiều cơ chế bảo mật để bảo vệ thông tin dữ liệu của người dùng, sau đây là các cơ chế mà MongoDB hỗ trợ:

Authentication	Authorization	TLS/SSL
<a href="#">Authentication</a>	<a href="#">Role-Based Access Control</a>	<a href="#">TLS/SSL (Transport Encryption)</a>
<a href="#">SCRAM</a>	<a href="#">Enable Access Control</a>	<a href="#">Configure mongod and mongos for TLS/SSL</a>
<a href="#">x.509</a>	<a href="#">Manage Users and Roles</a>	<a href="#">TLS/SSL Configuration for Clients</a>
 <b>Enterprise Only</b>	 <b>Encryption</b>	
<a href="#">Kerberos Authentication</a>	<a href="#">Client-Side Field Level Encryption</a>	
<a href="#">LDAP Proxy Authentication</a>		
<a href="#">Encryption at Rest</a>		
<a href="#">Auditing</a>		

- ❖ Chi tiết về các biện pháp bảo mật:

a. [Enable Access Control and Enforce Authentication](#)

Cho phép kiểm soát truy cập và chỉ định cơ chế xác thực. Người dùng có thể sử dụng cơ chế xác thực mặc định của MongoDB hoặc framwork bên ngoài hiện có. Tính xác thực yêu cầu tất cả khách hàng và server cung cấp thông tin xác thực hợp lệ trước khi kết nối với hệ thống. Trong các triển khai phân cụm, cho phép xác thực cho từng server MongoDB khác nhau.

b. [Configure Role-Based Access Control](#)

Tạo tài khoản quản trị viên trước, sau đó tạo các tài khoản khác của user. Mỗi người dùng có một tài khoản duy nhất để kết nối với hệ thống.

Tạo các role cho các user để xác định được chức năng mà họ có thể thực hiện được. Sau đó tạo user và chỉ định từng role cho họ. Người dùng có thể là người hoặc một ứng dụng khách.

c. [Encrypt Communication \(TLS/SSL\)](#)

Cấu hình MongoDB để sử dụng TLS/SSL cho các kết nối đến và đi. Sử dụng TLS/SSL để mã hóa giao tiếp giữa các thành phần của MongoDB là mongod và mongos cũng như giữa tất cả các ứng dụng khác với MongoDB.

<b>Windows</b>	<b>Secure Channel (Schannel)</b>
----------------	----------------------------------

Linux/BSD	OpenSSL
macOS	Secure Transport

d. Encrypt and Protect Data

Mở cơ chế với MongoDB Enterprise 3.2, công cụ lưu trữ WiredTiger là Encryption at Rest có thể được cấu hình để mã hóa thông tin trong lớp lưu trữ

Nếu bạn chưa sử dụng công cụ Encryption at Rest, dữ liệu của MongoDB nên được mã hóa trên mỗi file hệ thống của máy chủ, thiết bị, hay mã hóa vật lý. Để bảo vệ dữ liệu bằng hệ thống tệp. Dữ liệu MongoDB bao gồm tệp dữ liệu, tệp cấu hình, nhật ký kiểm toán và các tệp chính.

e. Limit Network Exposure

Đảm bảo MongoDB chạy trên môi trường mạng bền vững và giới hạn các phương thức kết nối với MongoDB. Chỉ cho phép khách hàng đáng tin cậy truy cập vào các giao thức mạng và cổng mà các phiên bản MongoDB có sẵn.

f. Audit System Activity

Theo dõi truy cập và thay đổi cấu hình của CSDL và dữ liệu. MongoDB Enterprise bao gồm một hệ thống thống kê có thể ghi lại các sự kiện của hệ thống (hoạt động của người dùng, các sự kiện kết nối). Những kết quả thống kê này cho phép phân tích nghiệp vụ và quản trị viên có thể kiểm soát.

g. Run MongoDB with a Dedicated User

Chạy MongoDB với tài khoản người dùng. Đảm bảo rằng tài khoản phải có quyền truy cập vào dữ liệu và không có những quyền không cần thiết.

Run MongoDB with Secure Configuration Options

MongoDB hỗ trợ thực thi mã JavaScript cho các hoạt động bên máy chủ: mapReduce và toán tử \$where. Nếu người dùng không thực hiện các hành động này, hãy vô hiệu hóa kịch bản máy chủ bằng cách sử dụng tùy chọn --noscripting trên Command line.

h. Run MongoDB with Secure Configuration Options

Giữ kích hoạt xác nhận đầu vào. MongodB cho phép xác thực đầu vào bằng mặc định thông qua cài đặt wireObjectCheck. Việc này đảm bảo tất cả document được lưu trữ trên mongod bằng BSON là hợp lệ.

i. [Request a Security Technical Implementation Guide \(where applicable\)](#)

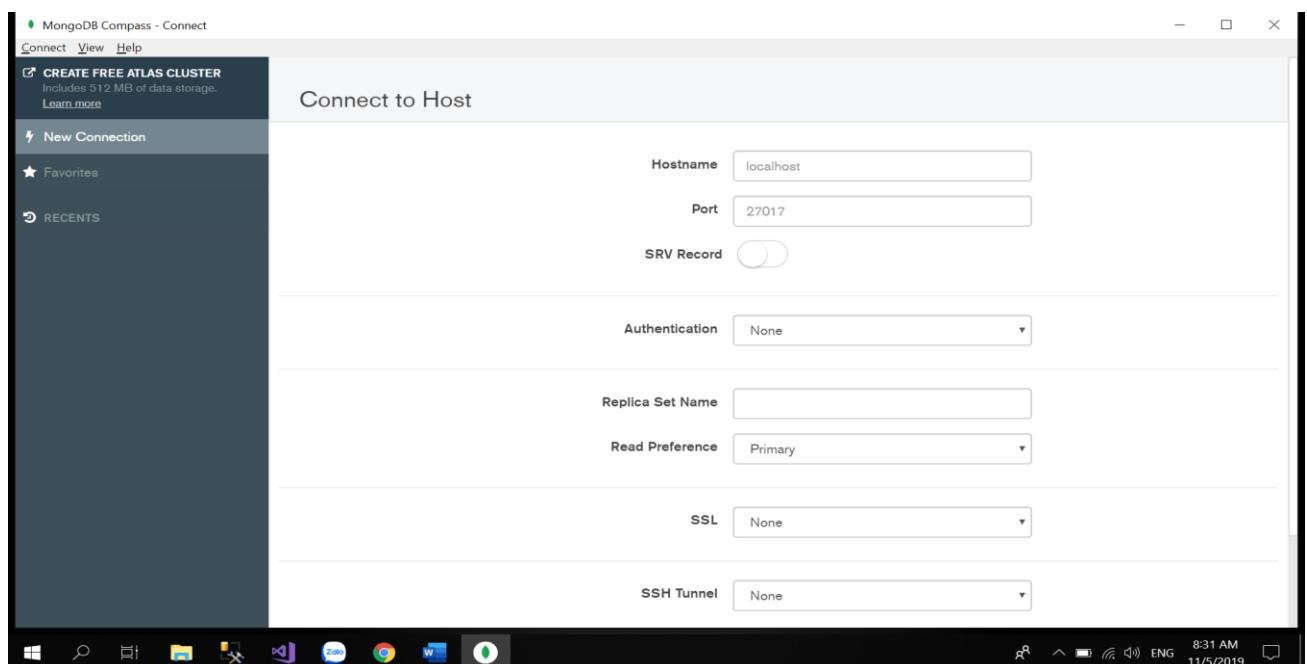
Hướng dẫn triển khai kỹ thuật bảo mật (STIG) chứa các hướng dẫn bảo mật cho việc triển khai bảo mật. MongoDB Inc. cung cấp STIG dựa vào yêu cầu, trong các trường hợp bắt buộc của chính quyền nhà nước.

j. [Consider Security Standards Compliance](#)

Đối với các ứng dụng yêu cầu tuân thủ HIPAA hoặc PCI-DSS, vui lòng tham khảo mục MongoDB Security Reference Architecture để tìm hiểu thêm cách để sử dụng chức năng bảo mật quan trọng để xây dựng ứng dụng phù hợp.

## VI. Cài đặt một CSDL minh họa

### 1. Cài đặt CSDL bằng MongoDB Compass



Giao diện của MongoDB Compass

The screenshot shows the MongoDB Compass interface. On the left, the sidebar displays 'My Cluster' with 'HOST localhost:27017', 'CLUSTER Standalone', and 'EDITION MongoDB 4.2.1 Community'. Below this is a search bar and a list of databases: 'admin', 'config', and 'local'. The main area is titled 'Databases' and shows a table with columns: Database Name, Storage Size, Collections, and Indexes. The table contains three rows corresponding to the databases listed in the sidebar. A green 'CREATE DATABASE' button is located at the top right of the table area.

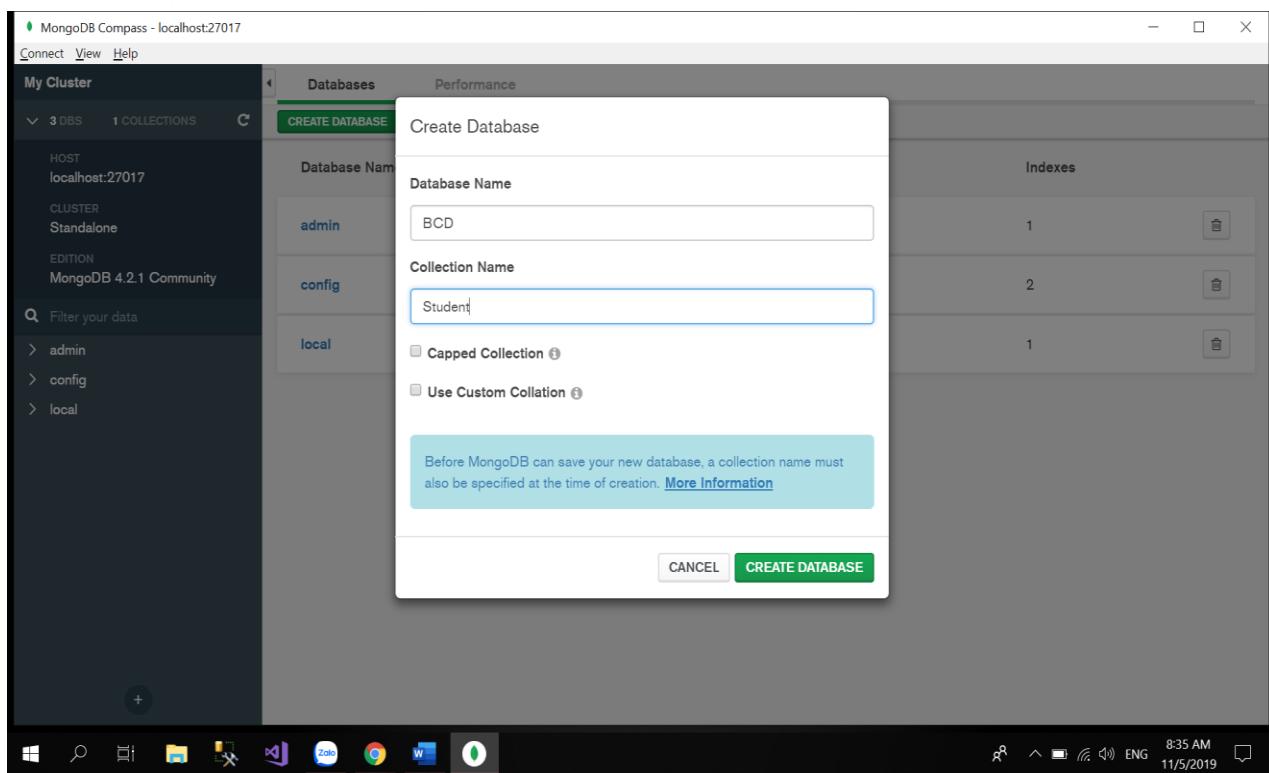
Database Name	Storage Size	Collections	Indexes
admin	20.5KB	0	1
config	24.6KB	0	2
local	36.9KB	1	1

### Các Database có sẵn của hệ thống

Để tạo 1 CSDL thì ta chọn nút Create Database màu xanh trên cùng bên trái

This screenshot is identical to the one above, but the 'CREATE DATABASE' button is highlighted with a red box. This visual cue indicates where the user should click to start creating a new database.

Đặt tên cho CSDL và tên cho Collection (Collection tương đương với Table trong SQL)



Database Name	Storage Size	Collections	Indexes
BCD	4.1KB	1	1
admin	20.5KB	0	1
config	24.6KB	0	2
local	36.9KB	1	1

### Kết quả sau khi tạo Database

Để tạo 1 Document (gồm các dòng dữ liệu tương đương với các thuộc tính của bảng). Ta vào Collection với tên vừa tạo (Student) và chọn Insert Document.

Id sẽ được tạo tự động, và việc còn lại là người dùng nhập thông tin

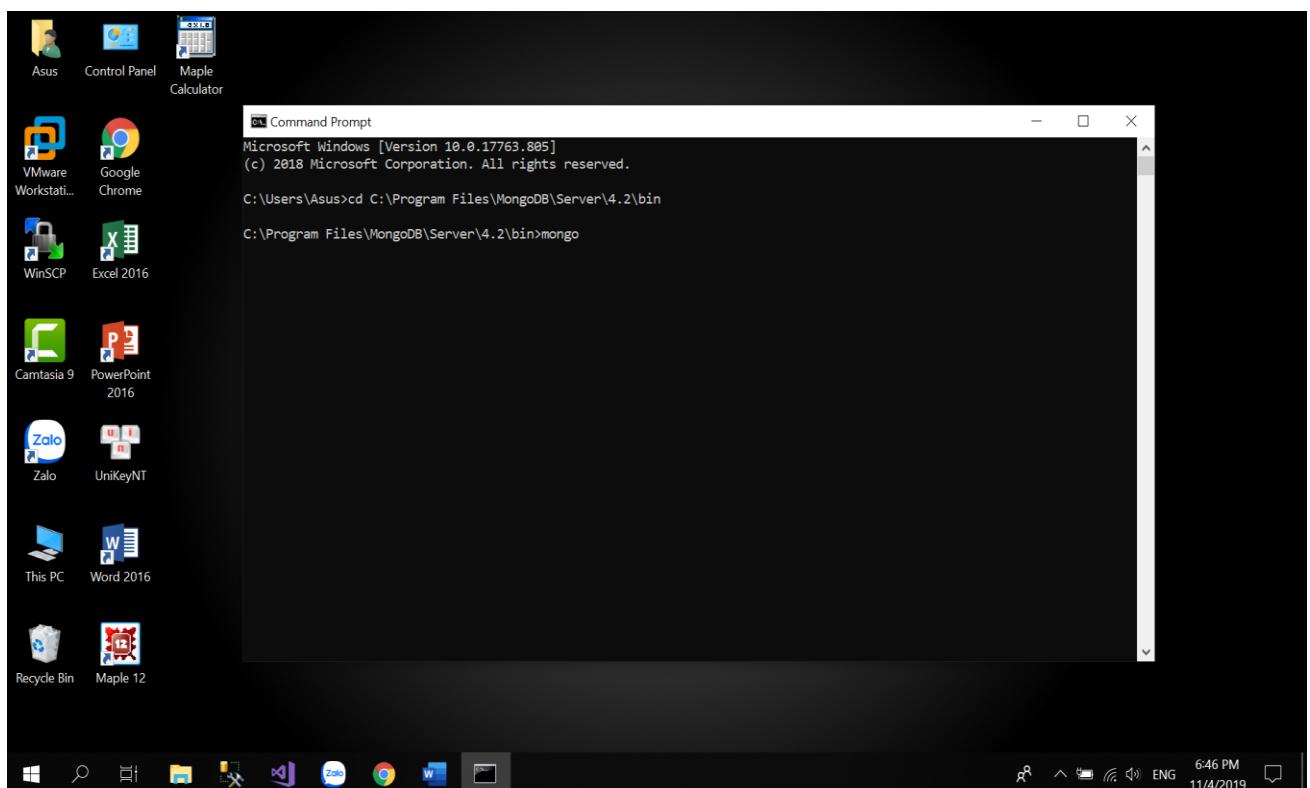
Kết quả sau khi nhập

## 2. Cài đặt CSDL bằng MongoDB Shell

Vào Command Prompt copy đường dẫn:

C:\Program Files\MongoDB\Server\4.2\bin

C:\Program Files\MongoDB\Server\4.2\bin>mongo



- Bước 1: Nhập lệnh show dbs để xem tất cả các Database có trong hệ thống.

```
Command Prompt - mongo
> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
>
```

A screenshot of a Command Prompt window titled "Command Prompt - mongo". The window displays the output of the "show dbs" command, which lists three databases: admin, config, and local, each with 0.000GB of space.

- Bước 2: Muốn tạo một Database mới trên hệ thống thì nhập lệnh **use + Tên Database muốn đặt**. Ở đây dùng tên là acme. Điều khác biệt ở MongoDB và các hệ quản trị CSDL SQL là người dùng không cần lệnh tạo CSDL mà chỉ cần nhập lệnh use, lệnh này cho phép vừa tạo ra CSDL vừa sử dụng trực tiếp trên nó.

### Command Prompt - mongo

```
> use acme
switched to db acme
> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
>
```

- Bước 3: Sử dụng lệnh **db.createCollection('tên collection')** để tạo một collection trong Database. Ở đây tên của Collection minh họa là posts

**Lưu ý:** Một collection ở MongoDB tương đương với một table ở SQL Server

```
> db.createCollection('posts')
{ "ok" : 1 }
>
```

- Bước 4: Để xem lại danh sách các collections đã tạo, nhập lệnh: **show collections**

### Command Prompt - mongo

```
> show collections
posts
>
```

- Bước 5: Ta nhập lệnh **db.nameCollections.insert({})** để thêm Documents vào Collections. Một Document ở MongoDB tương đương với một table ở SQL Server.

**Lưu ý:** Mỗi thông tin thêm thì cần phải để trong dấu nháy đơn ''

```

> show collections
posts
> db.posts.insert({
...     title: 'Post One',
...     body: 'Body of post one',
...     category: 'News',
...     like: 4,
...     tags: ['news', 'events'],
...     user: {
...         name: 'John Doe',
...         status: 'author'
...     },
...     date: Date()
... })
WriteResult({ "nInserted" : 1 })
>

```

- Có dòng lệnh WriteResult ({ "nInserted" : 1 }) => Insert thành công
- Bước 5.1: Để thêm đồng thời nhiều Documents (một record trong SQL) vào trong Collections, ta sử dụng lệnh ***db.nameCollections.insertMany([{}])***.

```

> db.posts.insertMany([
...   {
...     title: 'Post Two',
...     body: 'Body of post two',
...     category: 'Technology',
...     date: Date ()
...   },
...   {
...     title: 'Post Three',
...     body: 'Body of post three',
...     category: 'News',
...     date: Date ()
...   },
...   {
...     title: 'Post Four',
...     body: 'Body of post four',
...     category: 'Entertainment',
...     date: Date ()
... }
... ])

```

- ❖ Đây là kết quả sau khi nhập lệnh:

```

{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5de277412bc30b2ded4e7f80"),
    ObjectId("5de277412bc30b2ded4e7f81"),
    ObjectId("5de277412bc30b2ded4e7f82")
  ]
}
>

```

- Khi Insert thành công thì chương trình sẽ tạo ra 3 id tự động tương ứng với 3 Documents vừa tạo.
- Bước 6: Nhập lệnh ***db.nameCollections.find()*** để truy vấn tìm kiếm tất cả các documents có trong collections vừa tạo.

```
> db.posts.find()
{ "_id" : ObjectId("5de275712bc30b2ded4e7f7f"), "title" : "Post One", "body" : "Body of post one", "category" : "News", "like" : 4, "tags" : [ "news", "events" ], "user" : { "name" : "John Doe", "status" : "author" }, "date" : "Sat Nov 30 2019 20:58:09 GMT+0700 (SE Asia Standard Time)" }
{ "_id" : ObjectId("5de277412bc30b2ded4e7f80"), "title" : "Post Two", "body" : "Body of post two", "category" : "Technology", "date" : "Sat Nov 30 2019 21:05:53 GMT+0700 (SE Asia Standard Time)" }
{ "_id" : ObjectId("5de277412bc30b2ded4e7f81"), "title" : "Post Three", "body" : "Body of post three", "category" : "News", "date" : "Sat Nov 30 2019 21:05:53 GMT+0700 (SE Asia Standard Time)" }
{ "_id" : ObjectId("5de277412bc30b2ded4e7f82"), "title" : "Post Four", "body" : "Body of post four", "category" : "Entertainment", "date" : "Sat Nov 30 2019 21:05:53 GMT+0700 (SE Asia Standard Time)" }
>
```

- Bước 6.1:
- + Để các Documents được sắp xếp hợp lý và dễ nhìn hơn, ta thực hiện lệnh ***db.nameCollections.find().pretty()***

```
> db.posts.find().pretty()
{
  "_id" : ObjectId("5de275712bc30b2ded4e7f7f"),
  "title" : "Post One",
  "body" : "Body of post one",
  "category" : "News",
  "like" : 4,
  "tags" : [
    "news",
    "events"
  ],
  "user" : {
    "name" : "John Doe",
    "status" : "author"
  },
  "date" : "Sat Nov 30 2019 20:58:09 GMT+0700 (SE Asia Standard Time)"
}

{
  "_id" : ObjectId("5de277412bc30b2ded4e7f80"),
  "title" : "Post Two",
  "body" : "Body of post two",
  "category" : "Technology",
  "date" : "Sat Nov 30 2019 21:05:53 GMT+0700 (SE Asia Standard Time)"
}

{
  "_id" : ObjectId("5de277412bc30b2ded4e7f81"),
  "title" : "Post Three",
  "body" : "Body of post three",
  "category" : "News",
  "date" : "Sat Nov 30 2019 21:05:53 GMT+0700 (SE Asia Standard Time)"
}

{
  "_id" : ObjectId("5de277412bc30b2ded4e7f82"),
  "title" : "Post Four",
  "body" : "Body of post four",
  "category" : "Entertainment",
  "date" : "Sat Nov 30 2019 21:05:53 GMT+0700 (SE Asia Standard Time)"
}>
```

- + Để truy vấn với các điều kiện tương ứng (tương ứng với câu truy vấn có điều kiện where trong SQL) ta dùng lệnh ***db.nameCollections.find({điều***

**kiện truy vấn}).** Ở đây ta tìm những documents có điều kiện category là News. Để cho kết quả được sắp xếp gọn gàng, dùng thêm lệnh *pretty()*.

```
> db.posts.find({ category: 'News'}).pretty()
{
  "_id" : ObjectId("5de275712bc30b2ded4e7f7f"),
  "title" : "Post One",
  "body" : "Body of post one",
  "category" : "News",
  "like" : 4,
  "tags" : [
    "news",
    "events"
  ],
  "user" : {
    "name" : "John Doe",
    "status" : "author"
  },
  "date" : "Sat Nov 30 2019 20:58:09 GMT+0700 (SE Asia Standard Time)"
}
{
  "_id" : ObjectId("5de277412bc30b2ded4e7f81"),
  "title" : "Post Three",
  "body" : "Body of post three",
  "category" : "News",
  "date" : "Sat Nov 30 2019 21:05:53 GMT+0700 (SE Asia Standard Time)"
}
>
```

- **Bước 6.2:** Để truy vấn có sắp xếp theo thuộc tính ta dùng lệnh ***db.nameCollections.find({điều kiện truy vấn}).sort({thuộc tính cần sắp xếp: 1 hoặc -1})***. Với 1 là tăng dần, -1 là giảm dần. Ở đây ta sắp xếp các documents với title tăng dần. Để cho kết quả được sắp xếp gọn gàng, dùng thêm lệnh *pretty()*.

```

> db.posts.find().sort({ title: 1 }).pretty()
{
    "_id" : ObjectId("5de277412bc30b2ded4e7f82"),
    "title" : "Post Four",
    "body" : "Body of post four",
    "category" : "Entertainment",
    "date" : "Sat Nov 30 2019 21:05:53 GMT+0700 (SE Asia Standard Time)"
}
{
    "_id" : ObjectId("5de275712bc30b2ded4e7f7f"),
    "title" : "Post One",
    "body" : "Body of post one",
    "category" : "News",
    "like" : 4,
    "tags" : [
        "news",
        "events"
    ],
    "user" : {
        "name" : "John Doe",
        "status" : "author"
    },
    "date" : "Sat Nov 30 2019 20:58:09 GMT+0700 (SE Asia Standard Time)"
}
{
    "_id" : ObjectId("5de277412bc30b2ded4e7f81"),
    "title" : "Post Three",
    "body" : "Body of post three",
    "category" : "News",
    "date" : "Sat Nov 30 2019 21:05:53 GMT+0700 (SE Asia Standard Time)"
}
{
    "_id" : ObjectId("5de277412bc30b2ded4e7f80"),
    "title" : "Post Two",
    "body" : "Body of post two",
    "category" : "Technology",
    "date" : "Sat Nov 30 2019 21:05:53 GMT+0700 (SE Asia Standard Time)"
}
>

```

**Lưu ý:** Kết quả trên đã sắp xếp tăng dần theo chữ cái.

- **Bước 6.3:** Để truy vấn số lượng của thuộc tính thỏa điều kiện ta sử dụng lệnh **db.nameCollections.find({điều kiện truy vấn}).count()**.

```

> db.posts.find({ category: 'News' }).count()
2
>

```

- **Bước 6.4:** Để truy vấn thuộc tính thỏa điều kiện nhưng với giới hạn bản ghi ta thực hiện lệnh **db.nameCollections.find({điều kiện truy vấn}).limit(số lượng giới hạn).pretty()**.

```

> db.posts.find().limit(2).pretty()
{
    "_id" : ObjectId("5de275712bc30b2ded4e7f7f"),
    "title" : "Post One",
    "body" : "Body of post one",
    "category" : "News",
    "like" : 4,
    "tags" : [
        "news",
        "events"
    ],
    "user" : {
        "name" : "John Doe",
        "status" : "author"
    },
    "date" : "Sat Nov 30 2019 20:58:09 GMT+0700 (SE Asia Standard Time)"
}
{
    "_id" : ObjectId("5de277412bc30b2ded4e7f80"),
    "title" : "Post Two",
    "body" : "Body of post two",
    "category" : "Technology",
    "date" : "Sat Nov 30 2019 21:05:53 GMT+0700 (SE Asia Standard Time)"
}
>

```

- Bước 6.5: Để truy vấn một document với điều kiện cho trước, ta thực hiện lệnh ***db.nameCollections.findOne({điều kiện truy vấn})***.

```

> db.posts.findOne({ category: 'News' })
{
    "_id" : ObjectId("5de275712bc30b2ded4e7f7f"),
    "title" : "Post One",
    "body" : "Body of post one",
    "category" : "News",
    "like" : 4,
    "tags" : [
        "news",
        "events"
    ],
    "user" : {
        "name" : "John Doe",
        "status" : "author"
    },
    "date" : "Sat Nov 30 2019 20:58:09 GMT+0700 (SE Asia Standard Time)"
}
>

```

- Bước 6.6: Để truy vấn documents với điều kiện thuộc tính cho trước lớn hơn hoặc bằng 3, ta dùng toán tử ***\$gt: số lượng***.

```

> db.posts.find({ view : { $gt: 3} }).pretty()
{
    "_id" : ObjectId("5de275712bc30b2ded4e7f7f"),
    "title" : "Post One",
    "body" : "Body of post one",
    "category" : "News",
    "tags" : [
        "news",
        "events"
    ],
    "user" : {
        "name" : "John Doe",
        "status" : "author"
    },
    "date" : "Sat Nov 30 2019 20:58:09 GMT+0700 (SE Asia Standard Time)",
    "view" : 6
}
>

```

- Bước 7: Để cập nhật 1 document đã có sẵn ta thực hiện lệnh **`db.nameCollections.update({thuộc tính địa chỉ}, {những thuộc tính cần sửa đổi})`**. Ở đây ta sử dụng lệnh `upsert: true` có nghĩa là nếu thuộc tính địa chỉ tồn tại thì lệnh update sẽ được cập nhật lên trên document đã được chỉ định, với hành động thêm vào tất cả các thuộc tính cần sửa đổi trong câu lệnh vào document.

```
> db.posts.update( { title: 'Post Two' },
...   {
...     title: 'Post Two',
...     body: 'New post 2 body',
...     date: Date ()
...   },
...   {
...     upsert: true
...   }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

- ❖ Kết quả sau khi cập nhật:

```
> db.posts.find({ title: 'Post Two'}).pretty()
{
  "_id" : ObjectId("5de277412bc30b2ded4e7f80"),
  "title" : "Post Two",
  "body" : "New post 2 body",
  "date" : "Sat Nov 30 2019 22:34:18 GMT+0700 (SE Asia Standard Time)"
}
>
```

- Bước 7.1: Để cập nhật documents nhưng vẫn giữ nguyên những thuộc tính không cập nhật thì ta dùng toán tử **`$set`**.

```
> db.posts.update( { title: 'Post Two' },
...   {
...     $set: {
...       title: 'Post Two',
...       body: 'Body of post 2',
...       category: 'Technology'
...     }
...   }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

- ❖ Kết quả sau khi cập nhật:

```
> db.posts.find({ title: 'Post Two'}).pretty()
{
    "_id" : ObjectId("5de277412bc30b2ded4e7f80"),
    "title" : "Post Two",
    "body" : "Body of post 2",
    "date" : "Sat Nov 30 2019 22:34:18 GMT+0700 (SE Asia Standard Time)",
    "category" : "Technology"
}
>
```

- Bước 7.2: Để cập nhật documents với việc thực hiện tăng số lượng của thuộc tính thì dùng toán tử **\$inc: { thuộc tính: số lượng cần tăng}**.

```
> db.posts.update({ title: 'Post One'}, { $inc: {like: 2} })
writeResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

❖ Kết quả sau khi cập nhật:

```
> db.posts.find({ title: 'Post One'}).pretty()
{
    "_id" : ObjectId("5de275712bc30b2ded4e7f7f"),
    "title" : "Post One",
    "body" : "Body of post one",
    "category" : "News",
    "like" : 6,
    "tags" : [
        "news",
        "events"
    ],
    "user" : {
        "name" : "John Doe",
        "status" : "author"
    },
    "date" : "Sat Nov 30 2019 20:58:09 GMT+0700 (SE Asia Standard Time)"
}
>
```

- Thuộc tính like ban đầu là 4,sau đó đã được cập nhật lên 6.
- Bước 7.3: Để cập nhật documents với việc thực hiện thay đổi thuộc tính, sử dụng toán tử **\$rename: { tên thuộc tính: 'tên muốn sửa'}**.

```
> db.posts.update({ title: 'Post One'}, { $rename: {like: 'view'} })
writeResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

❖ Kết quả sau khi cập nhật:

```

> db.posts.update({ title: 'Post One' }, { $rename: {like: 'view'} })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.posts.find({ title: 'Post One'}).pretty()
{
    "_id" : ObjectId("5de275712bc30b2ded4e7f7f"),
    "title" : "Post One",
    "body" : "Body of post one",
    "category" : "News",
    "tags" : [
        "news",
        "events"
    ],
    "user" : {
        "name" : "John Doe",
        "status" : "author"
    },
    "date" : "Sat Nov 30 2019 20:58:09 GMT+0700 (SE Asia Standard Time)",
    "view" : 6
}
>

```

- Bước 8: Để xóa bất kỳ documents nào trong collections, ta thực hiện lệnh **`db.nameCollections.remove({thuộc tính: giá trị thuộc tính})`**.

```

> db.posts.remove({ title : 'Post Four' })
WriteResult({ "nRemoved" : 1 })
>

```

❖ Kết quả sau khi cập nhật:

```

> db.posts.find().pretty()
{
    "_id" : ObjectId("5de275712bc30b2ded4e7f7f"),
    "title" : "Post One",
    "body" : "Body of post one",
    "category" : "News",
    "tags" : [
        "news",
        "events"
    ],
    "user" : {
        "name" : "John Doe",
        "status" : "author"
    },
    "date" : "Sat Nov 30 2019 20:58:09 GMT+0700 (SE Asia Standard Time)",
    "view" : 6
}
{
    "_id" : ObjectId("5de277412bc30b2ded4e7f80"),
    "title" : "Post Two",
    "body" : "Body of post 2",
    "date" : "Sat Nov 30 2019 22:34:18 GMT+0700 (SE Asia Standard Time)",
    "category" : "Technology"
}
{
    "_id" : ObjectId("5de277412bc30b2ded4e7f81"),
    "title" : "Post Three",
    "body" : "Body of post three",
    "category" : "News",
    "date" : "Sat Nov 30 2019 21:05:53 GMT+0700 (SE Asia Standard Time)"
}
>

```

## VII. Đánh giá các tính năng của MongoDB

### 1. Các tính năng hỗ trợ

Tính năng hỗ trợ	Đặc trưng
Lưu trữ hướng văn bản	Văn bản theo phong cách JSON với những lược đồ động đơn giản.
Hỗ trợ chỉ mục đầy đủ	Chỉ mục trên bất kỳ các thuộc tính.
Tính sao lặp và tính sẵn sàng cao	Mở rộng.
Auto-sharding	Mở rộng theo chiều ngang mà không ảnh hưởng đến chức năng.
Truy vấn	Đa dạng, truy vấn dựa trên văn bản.
GridFS	Lưu trữ file với bất kỳ kích cỡ nào mà không làm phức tạp ngăn xếp.
Hỗ trợ thương mại	Hỗ trợ doanh nghiệp, đào tạo, tư vấn khái niệm cơ bản trong MongoDB.
Ít Schema hơn	MongoDB là một cơ sở dữ liệu dựa trên Document. Số trường, nội dung và kích cỡ của Document này có thể khác với Document khác.
Cấu trúc	Cấu trúc của một đối tượng là rõ ràng.
Không có các Join phức tạp và các transaction	Giúp truy vấn nhanh hơn và đơn giản hơn.
Khả năng truy vấn sâu hơn	MongoDB hỗ trợ các truy vấn động trên các Document bởi sử dụng một ngôn ngữ truy vấn dựa trên Document mà mạnh mẽ như SQL.
Dễ dàng để mở rộng	MongoDB có thể được mở rộng theo chiều rộng và theo chiều ngang.
Mở rộng theo chiều ngang	Phương pháp tăng cường khả năng lưu trữ và xử lý là dùng nhiều máy tính phân tán.
Mở rộng theo chiều dọc	Giúp tăng cấu hình server.
Chuyển đổi/ánh xạ của các đối tượng ứng dụng đến các đối tượng cơ sở dữ liệu	Không cần thiết.

Sử dụng bộ nhớ nội tại để lưu giữ phần công việc	Giúp truy cập dữ liệu nhanh hơn. Khi có quá nhiều dữ liệu (giả sử 10 triệu records) thì nên dùng MongoDB do có khả năng tìm kiếm thông tin liên quan nhanh.
Tăng tốc độ truy xuất dữ liệu	Phù hợp cho các ứng dụng cần tốc độ phản hồi nhanh (realtime).
Cơ chế ghi với tốc độ cao và an toàn	Nếu website ở dạng realtime nhiều (nhiều người thao tác với ứng dụng). Nếu trong quá trình load bị lỗi tại một điểm nào đó thì MongoDB sẽ bỏ qua phần đó nên sẽ an toàn.
Không có tính ràng buộc	Rất cần sự cẩn thận khi thao tác trên các collection có quan hệ dữ liệu với nhau để tránh các lỗi mất hoặc sai dữ liệu.
Đẩy trách nhiệm thao tác Database cho tầng ứng dụng	Sẽ làm tốn tài nguyên.
Dùng nhiều máy tính phân tán để lưu trữ dữ liệu	Chi phí sẽ rẻ hơn MySQL (do MySQL sử dụng những máy chủ hàng khủng, độc quyền nên sẽ đắt đỏ hơn).
Đảm bảo hiệu suất của ứng dụng mặc định	Khi có yêu cầu thêm/sửa/xóa bản ghi, MongoDB sẽ chưa cập nhật xuống ổ cứng ngay, mà sau 60 giây MongoDB mới thực hiện ghi toàn bộ dữ liệu thay đổi từ RAM xuống ổ cứng.

Các điểm lưu ý khi lựa chọn MongoDB:

- Ứng dụng của có tính chất INSERT cao, bởi vì mặc định MongoDB có sẵn cơ chế ghi với tốc độ cao và an toàn.
- Ứng dụng ở dạng thời gian thực nhiều, nghĩa là nhiều người thao tác với ứng dụng. Nếu trong quá trình load bị lỗi tại một điểm nào đó thì nó sẽ bỏ qua phần đó nên sẽ an toàn.
- Ứng dụng bạn có nhiều dữ liệu quá. Ví dụ, giả sử web có đến 10 triệu records thì đó là cơn ác mộng với MySQL. Bởi vì MongoDB có khả năng tìm kiếm thông tin liên quan cũng khá nhanh nên trường hợp này nên dùng nó.
- Máy chủ không có hệ quản trị CSDL, trường hợp này thường bạn sẽ sử dụng SQL LIFE hoặc là MongoDB.

## 2. Ưu điểm và khuyết điểm

### 2.1. Ưu điểm:

- Là mã nguồn mở: không phải mất chi phí và có xu hướng tin cậy, an ninh và nhanh hơn để triển khai so với các hệ quản trị cơ sở dữ liệu độc quyền.
- Linh hoạt trong việc mở rộng và phát triển: Về phía công ty quản lý thì giúp dễ dàng mở rộng máy chủ khi dữ liệu càng ngày càng lớn hoặc lượng truy cập, tải dữ liệu quá lớn, thay vì thuê một máy chủ lớn hơn để thế máy chủ trước thì công ty chỉ cần thuê thêm một máy chủ khác. Về phía người lập trình thì giúp dễ dàng thêm Collection (Bảng) hoặc Field (Cột). Dễ dàng trong việc thống kê, truy vấn nhanh.
- Áp dụng được công nghệ điện toán đám mây: dễ dàng mở rộng phạm vi được theo yêu cầu có sử dụng một dịch vụ như là Amazon EC2. Giống như tất cả công nghệ đám mây, EC2 dựa vào ảo hóa. Liên kết yếu của ảo hóa là sự thực thi của I/O, với bộ nhớ và CPU các kết nối mạnh. MongoDB lưu trữ dữ liệu thường được mở rộng phạm vi theo chiều ngang tận dụng được sự cung cấp mềm dẻo của đám mây. Giúp mở rộng dữ liệu dễ dàng hơn.
- Được các hãng lớn sử dụng như: Amazon, BBC, Facebook và Google. Làm nền tảng, cơ sở cho các công ty lớn áp dụng cũng như có sự tin cậy cao về hệ quản trị NoSQL.
- MongoDB không có khái niệm table và relationship nên dữ liệu rất thoải mái.
- MongoDB là một rich query language tức là nó có sẵn các method để thực hiện create, read, update, delete dữ liệu (CRUD).
- MongoDB có hỗ trợ replica set nhằm đảm bảo việc sao lưu và khôi phục dữ liệu.
- Trong MongoDB có khái niệm cluster là cụm các node chứa dữ liệu giao tiếp với nhau, khi muốn mở rộng hệ thống ta chỉ cần thêm một node vào cluster.
- Dữ liệu được caching (ghi đệm) lên RAM, hạn chế truy cập vào ổ cứng nên tốc độ đọc và ghi cao.
- Linh hoạt: Cung cấp các sơ đồ linh hoạt giúp công đoạn phát triển nhanh hơn và có khả năng lặp lại cao hơn. Mô hình dữ liệu linh hoạt biến MongoDB thành một trong những lựa chọn lý tưởng cho dữ liệu không được tổ chức thành cấu trúc hoặc có cấu trúc chưa hoàn chỉnh. MongoDB là document database, dữ liệu lưu dưới dạng JSON, không bị bó buộc về số lượng field, kiểu dữ liệu... ta có thể insert thoải mái dữ liệu mà mình muốn. MongoDB có cấu trúc Schema động, nó không cần phải định nghĩa một mô hình với những mối quan hệ phải được định sẵn khi thiết kế.
- Khả năng thay đổi quy mô: Được thiết kế để tăng quy mô bằng cách sử dụng các cụm phần cứng được phân phối thay vì tăng quy mô bằng cách bổ sung máy chủ mạnh và tốn kém. Một số nhà cung cấp dịch vụ đám mây xử lý các

hoạt động này một cách không công khai dưới dạng dịch vụ được quản lý đầy đủ.

- Hiệu năng cao: Được tối ưu hóa theo mô hình dữ liệu (như document, key-value, graph) và các mẫu truy cập giúp tăng hiệu năng cao hơn so với việc cố gắng đạt được mức độ chức năng tương tự bằng cơ sở dữ liệu quan hệ.
- Cực kỳ thiết thực: Cung cấp các API và kiểu dữ liệu cực kỳ thiết thực được xây dựng riêng cho từng mô hình dữ liệu tương ứng.
- Có hiệu năng cao: MongoDB lưu dữ liệu dạng JSON, khi insert nhiều đối tượng thì nó sẽ là insert một mảng JSON gần như với trường hợp insert 1 đối tượng. Dữ liệu trong MongoDB không có sự ràng buộc lẫn nhau như trong RDBMS, khi insert, xóa hay update nên không cần phải mất thời gian kiểm tra xem có thỏa mãn các bảng liên quan như trong RDBMS. Dữ liệu trong MongoDB được đánh index nên khi truy vấn tốc độ tìm rất nhanh. Khi thực hiện insert, find... MongoDB sẽ khóa các thao tác khác lại. Ví dụ: khi thực hiện find(), trong quá trình find mà có thêm thao tác insert, update thì sẽ dừng hết lại để chờ find() thực hiện xong.

## 2.2. **Nhược điểm:**

- Hỗ trợ không đồng đều cho các doanh nghiệp. Các doanh nghiệp vừa được sự hỗ trợ tốt nhất từ nhà cung cấp RMBMS như Oracle, MySQL, SQL Server,... (vì được phát triển trước NoSQL, cụ thể là MongoDB một thời gian khá dài) còn các doanh nghiệp nhỏ thì thường sử dụng các mã nguồn mở thì không được sự hỗ trợ tốt nhất.
- MongoDB chưa được sử dụng rộng rãi vì NoSQL vẫn chưa nhận được sự tin cậy với nhiều doanh nghiệp, một phần dữ liệu đã được xây dựng từ lâu, nên việc chuyển đổi cũng là vấn đề khó với nhiều doanh nghiệp và chưa được hỗ trợ tốt về chức năng cũng như sự ổn định như RMBMS.
- Còn mới lạ với một số lập trình viên. Chưa được sử dụng để đào tạo rộng rãi. Chưa có tool hỗ trợ giao diện tương tác cũng như các phương thức tốt nhất. Dẫn đến hạn chế về tri thức nghiệp vụ.
- MongoDB có giao diện lập trình ứng dụng API riêng của mình. Sự thiếu hụt các tiêu chuẩn có nghĩa là nó không có khả năng để chuyển một cách đơn giản từ một nhà cung cấp này sang một nhà cung cấp khác.
- Không ràng buộc, toàn vẹn, không có các tính chất ràng buộc như trong RDBMS nên không ứng dụng được cho các mô hình giao dịch yêu cầu độ chính xác cao và dữ liệu dễ bị làm sai.
- Không có cơ chế transaction (giao dịch) để phục vụ các ứng dụng ngân hàng.
- Dữ liệu được caching, lấy RAM làm trọng tâm hoạt động vì vậy khi hoạt động yêu cầu một bộ nhớ RAM lớn.

- Mọi thay đổi về dữ liệu mặc định đều chưa được ghi xuống ổ cứng ngay lập tức vì vậy khả năng bị mất dữ liệu từ nguyên nhân mất điện đột xuất là rất cao.
- Không hỗ trợ join giống như RDBMS nên khi viết function join trong code ta phải làm bằng tay khiến cho tốc độ truy vấn bị giảm.
- Sử dụng nhiều bộ nhớ: do dữ liệu lưu dưới dạng key-value, các collection chỉ khác về value do đó key sẽ bị lặp lại. Không hỗ trợ join nên sẽ bị dữ thừa dữ liệu (trong RDBMS thì ta chỉ cần lưu 1 bản ghi rồi các bản ghi khác tham chiếu tới còn trong MongoDB thì không).
- Bị giới hạn kích thước bản ghi: mỗi document không được có kích thước quá 16Mb và mức độ các document con trong 1 document không được lớn hơn 100.

### 3. So sánh MongoDB và SQL Server

	<b>MongoDB</b>	<b>SQL Server</b>
Bản quyền	Open Source.	Commercial.
Kiểu	Hướng tài liệu (Document).	RDBMS.
Công ty	MongoDB.Inc	Microsoft.
Năm phát hành phiên bản đầu tiên	1989.	2009.
Hệ điều hành	Windows, Linux, OS X.	Windows.
Ngôn ngữ truy vấn	JSON	SQL
Drivers	Dart, Delphi, Erlang, Go, Groovy, Haskell, Java, JavaScript, Lisp, Lua, Matlab, Perl, PHP, PowerShell, Prolog, Python, R, Ruby, Scala, Smalltalk.	.NET, Java, PHP, Python, Ruby, Visual Basic.
Server-side scripts	JavaScript's.	Transaction and .NET languages.
Quan hệ	Không cần các quan hệ, ràng buộc giữa các collection. ⇒ Tốc độ thực hiện thao tác nhanh.	Giữa các bảng thường có các quan hệ, ràng buộc. ⇒ Thực hiện thao tác chính xác, giảm thiểu sai sót

Trường	Không quy định số trường, số thuộc tính trong collection.	Số lượng các trường, thuộc tính trong cùng 1 bảng phải giống nhau.
Câu lệnh truy vấn	Ngắn gọn.	Dài hơn so với MongoDB nhưng được thể hiện tường minh.
	Phân biệt hoa thường.	Không phân biệt hoa thường.
Độ tiện lợi	<p>Tất cả tính năng không được cô đọng trong 1 ứng dụng thống nhất, mà phân thành các ứng dụng nhỏ với các tính năng riêng.</p> <ul style="list-style-type: none"> <li>• MongoDB compass – thao tác bằng giao diện (dành cho người dùng phổ thông).</li> <li>• MongoDB shell – thao tác trên code (dành cho lập trình viên).</li> <li>• Robo Mongo – quản lý cơ sở dữ liệu được tạo trên MongoDB shell.</li> </ul> <p>⇒ Giúp tiết kiệm dung lượng trong một số trường hợp</p>	<p>Tất cả tính năng được cô đọng trong cùng 1 ứng dụng SQL Server</p> <p>⇒ Dễ sử dụng.</p>
Tốc độ truy vấn (Insert, Update, Remove/Delete, Find>Select)	<p>Nhanh hơn SQL Server</p> <ul style="list-style-type: none"> <li>• Tốc độ nhanh hơn khoảng 30-50 lần.</li> <li>• Hiệu suất nhanh hơn khoảng 3 lần.</li> </ul>	Chậm hơn MongoDB.
Khóa ngoại	Không cần khóa ngoại.	Có khóa ngoại.
Joins	Không hỗ trợ.	Hỗ trợ.
Transaction	Không hỗ trợ.	ACID
Trigger	Chỉ hỗ trợ trong MongoDB Stitch.	Hỗ trợ.
Lược đồ dữ liệu	Dynamic.	Fixed.

Khả năng mở rộng	Theo chiều ngang.	Theo chiều dọc.
Map reduce	Hỗ trợ.	Không hỗ trợ.
Index thứ hai	Hỗ trợ.	Không hỗ trợ.
Agile practices	Hỗ trợ.	Không hỗ trợ.
Tính đồng thời	Không.	Có.
XML support	Không.	Có.

Dưới đây là bảng xếp hạng về mức độ phổ biến của các DBMS vào tháng 10 và 11/2019 so với tháng 11/2018.

Rank			DBMS	Database Model	Score		
Nov 2019	Oct 2019	Nov 2018			Nov 2019	Oct 2019	Nov 2018
1.	1.	1.	Oracle	Relational, Multi-model	1336.07	-19.81	+34.96
2.	2.	2.	MySQL	Relational, Multi-model	1266.28	-16.78	+106.39
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1081.91	-12.81	+30.36
4.	4.	4.	PostgreSQL	Relational, Multi-model	491.07	+7.16	+50.83
5.	5.	5.	MongoDB	Document, Multi-model	413.18	+1.09	+43.70
6.	6.	6.	IBM Db2	Relational, Multi-model	172.60	+1.83	-7.27
7.	7.	8.	Elasticsearch	Search engine, Multi-model	148.40	-1.77	+4.94
8.	8.	7.	Redis	Key-value, Multi-model	145.24	+2.32	+1.06
9.	9.	9.	Microsoft Access	Relational	130.07	-1.10	-8.36
10.	10.	11.	Cassandra	Wide column	123.23	+0.01	+1.48
11.	11.	10.	SQLite	Relational	121.03	-1.60	-1.68
12.	12.	12.	Splunk	Search engine	89.06	+2.23	+8.69
13.	13.	14.	MariaDB	Relational, Multi-model	85.57	-1.20	+12.32
14.	14.	15.	Hive	Relational	84.22	-0.52	+19.65
15.	15.	13.	Teradata	Relational, Multi-model	80.35	+1.61	+1.04
16.	16.	21.	Amazon DynamoDB	Multi-model	61.37	+1.19	+7.56
17.	17.	16.	Solr	Search engine	57.78	+0.21	-3.10
18.	18.	20.	FileMaker	Relational	55.73	-0.94	-0.02
19.	19.	18.	SAP Adaptive Server	Relational	55.29	-0.54	-1.27
20.	20.	19.	SAP HANA	Relational, Multi-model	55.11	-0.24	-0.76

## C- TÀI LIỆU THAM KHẢO

<https://docs.mongodb.com/>

<https://viblo.asia/p/tim-hieu-ve-mongodb-4P856ajGIY3>

<https://viblo.asia/p/mongodb-co-ban-phan-1-l5XRBVN3RqPe>

<https://trustradius.com/compare-products/mongodb-vs-sql-server>

[https://www.udemy.com/course/mongo-db/?utm\\_source=adwords&utm\\_medium=udemyads&utm\\_campaign=MongoDB\\_v.PRO\\_F\\_la.EN\\_cc.ROW\\_t.6804&utm\\_content=deal4584&utm\\_term=.ag\\_85479005194\\_.a](https://www.udemy.com/course/mongo-db/?utm_source=adwords&utm_medium=udemyads&utm_campaign=MongoDB_v.PRO_F_la.EN_cc.ROW_t.6804&utm_content=deal4584&utm_term=.ag_85479005194_.a)

[d\\_397920588696\\_.kw\\_.de\\_c\\_.dm\\_.pl\\_.ti\\_dsa-774930035689\\_.li\\_9074084\\_.pd\\_.&matchtype=b&gclid=Cj0KCQiAoIPvBRDgARIsAHsCw0\\_-cgOZAMuobKIGFAEvnCBOPrCBJuWFO\\_rypl6VBZOpXRmdUT\\_j-XwaAh7tEALw\\_wcB](https://www.google.com/search?q=d_397920588696_.kw_.de_c_.dm_.pl_.ti_dsa-774930035689_.li_9074084_.pd_.&matchtype=b&gclid=Cj0KCQiAoIPvBRDgARIsAHsCw0_-cgOZAMuobKIGFAEvnCBOPrCBJuWFO_rypl6VBZOpXRmdUT_j-XwaAh7tEALw_wcB)

<https://www.tutorialspoint.com/mongodb/index.htm>

<https://en.wikipedia.org/wiki/MongoDB>