**ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH**

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

—————o0o—————


**HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU**

# Document Store (MongoDB, CouchDB, SimpleDB

**Giảng viên hướng dẫn:** Thầy Trần Quang

**Sinh viên thực hiện:**



| Họ và tên | MSSV |
|---|---|
| Trương Ngọc Anh | 1410141 |
| Lê Nguyễn Minh Trí | 1414207 |
| Nguyễn Quốc Kim Hoàng | 141xxxx |
| Nguyễn Khánh Bình | 141xxxx |

# Mục lục

# Danh sách hình vẽ

# Danh sách bảng

# Chương 1

# Giới thiệu NoSQL và MongoDB

## 1.1 Mở đầu

Hiện nay khi nói đến cơ sở dữ liệu, có sự phân chia rõ ràng giữa 2 trường phái kĩ thuật cơ sở dữ liệu là SQL và NoSQL - hay còn gọi cách khác là Cơ sở dữ liệu Quan hệ và Cơ sở dữ liệu Phi Quan hệ. Trong các chương trình giảng dạy ở bậc đại học thì thường chú trọng vào nghiên cứu và tìm hiểu các nguyên lý, cấu tạo của cơ sở dữ liệu SQL mà không khai thác các cơ sở dữ liệu NoSQL. Do đó ở bài báo cáo này, chúng ta sẽ tập trung tìm hiểu và khai thác kiến thức về các cơ sở dữ liệu NoSQL mà đối tượng nghiên cứu cụ thể tiêu biểu là MongoDB.

## 1.2 NoSQL - Non SQL

Đầu tiên, ta sẽ giới thiệu về NoSQL. Bên cạnh đó nêu ra cách phân loại các cơ sở dữ liệu NoSQL và nêu ra các điểm khác nhau nổi bật giữa cơ sở dữ

liệu NoSQL và cơ sở dữ liệu quan hệ truyền thống.

### 1.2.1 Giới thiệu

Tuy NoSQL đã xuất hiện từ thập niên 60 nhưng thời gian gần đây nó mới bắt đầu tìm được vị trí của mình. Như tên gọi của nó, NoSQL cung cấp cơ chế triển khai cơ sở dữ liệu không theo cấu trúc nhất định. SQL đã tận dụng tốt việc khai báo schema chặt chẽ để quản lý, lưu trữ dữ liệu một cách tối ưu và cung cấp cho người dùng phương tiện để truy cập, sử dụng cơ sở dữ liệu hiệu quả nhất, nhờ đó mà nó đã từng là lựa chọn hàng đầu cho việc triển khai cơ sở dữ liệu.

Trong thời đại hiện nay, với sự bùng nổ của các dịch vụ internet kéo theo sự bùng nổ dữ liệu người dùng phi cấu trúc. Chính sự chặt chẽ của cơ sở dữ liệu SQL đã gây khó khăn trong việc quản lý, lưu trữ các dữ liệu phi cấu trúc này do SQL chỉ hoạt động tốt trên dữ liệu có cấu trúc. Điều này đã tạo cơ hội cho NoSQL phát huy điểm mạnh của mình.

### 1.2.2 Phân loại

Cơ sở dữ liệu NoSQL sử dụng khai báo schema động để quản lý cơ sở dữ liệu theo nhiều cách khác nhau, dựa vào đó ta có thể phân chúng thành nhiều loại:

- **Key-value store** mô hình SQL đơn giản nhất, lưu trữ dữ liệu dưới dạng khóa và trị. Tiêu biểu: ArangoDB, Aerospike, Oracle NoSQL Database, Dynamo, Riak, Voldemort.
- **Column store** lưu các bảng dữ liệu theo dạng cột như đơn vị lưu

trữ nhỏ nhất thay vì theo hàng như truyền thống. Tiêu biểu: Amazon DynamoDB, Bigtable, Cassandra, Druid, HBase, Hypertable.

- **Document store** với ý tưởng chính là việc định nghĩa "document", là đơn vị dữ liệu nhỏ nhất chứa dữ liệu của bản thân nó và có mang key để phục vụ việc truy xuất. Ngoài việc truy xuất theo khóa thì các Document-Database này còn cung cấp các API giúp truy xuất document dựa trên nội dung nó chứa. Tiêu biểu: ***MongoDB***, ArangoDB, BaseX, Clusterpoint, Couchbase, CouchDB, DocumentDB, IBM Domino, MarkLogic, Qizx, RethinkDB.

- **Graph** thường được dùng cho dữ liệu mà quan hệ giữa các thực thể có thể được biểu diễn tốt bằng đồ thị. Tiêu biểu: Neo4J, Polyglot.

## 1.3 So sánh SQL và NoSQL

### 1.3.1 Query phức tạp

Nhờ vào cấu trúc chặt chẽ của mình nên SQL có khả năng thực hiện các câu query phức tạp một cách tối ưu nhất so với NoSQL.

### 1.3.2 ACID

Tuân thủ ACID giúp CSDL SQL bảo vệ tính toàn vẹn, đảm bảo tính hợp lý của transaction. Tuy nhiên CSDL NoSQL bỏ qua ACID để có thể trở nên linh động và tăng tốc độ xử lý.

### 1.3.3 Dữ liệu lưu trữ

SQL chỉ có thể lưu trữ có cấu trúc. NoSQL có thể lưu trữ dữ liệu có cấu trúc hoặc phi cấu trúc.

### 1.3.4 Khả năng mở rộng

SQL khả mở rộng độ sâu của CSDL trong khi NoSQL khả mở rộng chiều ngang, giúp cho CSDL có thể được phân tán ở nhiều server qua đó tăng hiệu suất truy xuất dữ liệu, phù hợp cho các CSDL lớn và thay đổi liên tục.

### 1.3.5 Cấu trúc

Trong khi SQL ràng buộc việc khai báo schema rõ ràng trước khi có thể thêm dữ liệu thì với NoSQL ta có thể tạo document mà không cần khai báo schema trước. Mỗi document có cấu trúc, schema khác nhau.

## 1.4 MongoDB

Sau khi đã tìm hiểu về NoSQL nói chung ở phần trước thì phần này ta sẽ đi vào chi tiết cụ thể về một điển hình tiêu biểu của NoSQL là MongoDB. Ở phần 1.4.1 ta sẽ giới thiệu sơ bộ về MongoDB bên cạnh các khái niệm về các đơn vị và cấu trúc cơ bản của một CSDL xây dựng bằng MongoDB. Sau đó ta sẽ liệt kê sơ bộ các loại dữ liệu trong phần 1.4.2 và các lệnh cơ bản nhất trong phần 1.4.3.

### 1.4.1 Giới thiệu

MongoDB là hệ CSDL NoSQL Document store mở. Mỗi record trong MongoDB là một ***document***, gồm có các cặp trường và trị, và mỗi bảng sẽ là ***collection***, tập hợp nhiều document, và ***Database*** là tập hợp của nhiều collection. Do cấu trúc linh động nên bảng không có schema cố định mà schema của mỗi document sẽ tự do document quản lý và quyết định. Document của MongoDB có cấu trúc gần giống với các đối tượng JSON. Các giá trị có thể là document khác hoặc mảng hoặc mảng các document.

### 1.4.2 Datatype

Ngoài việc hỗ trợ các datatype thông dụng trong các ngôn ngữ lập trình, MongoDB còn định nghĩa một số datatype khác nhau:

- **Date**
- **ObjectId** là kiểu dữ liệu được MongoDB dùng mặc định để đánh Id cho các document trong collection

- **Số** MongoDB mặc định lưu trữ các dữ liệu số dưới dạng double 64-bit. Bên cạnh đó còn cung cấm nhiều wrapper để đặc tả các số một cách cụ thể như:
  - **NumberLong** 64-bit integer
  - **NumberInt** 32-bit integer
  - **NumberDecimal** 128-bit decimal-based floating-point numbers

### 1.4.3 Các thao tác cơ bản trong MongoDB sử dụng mongo shell - CRUD

#### 1.4.3.1 Create

- ```
  use DATABASE_NAME
  ```

  sẽ truy cập database có tên *DATABASE_NAME* hoặc tạo nếu chưa tồn tại và truy cập database vừa tạo.

- ```
  db.createCollection(collection_name, options)
  ```

  dùng để tạo một collection tên *collection_name* và có thể khai báo thêm một số option như capped, size, max - Giới hạn kích thước hay số lượng document tối đa của collection - hay autoIndexId - tự động index trường _id của document.

- ```
  db.collection_name.insert(document)


  db.collection_name.insertOne(document)
  ```
  ```

thêm *document* vào collection *collection_name*

- ```
  db.collection_name.insertMany(documents)
  ```

  thêm tất cả các document trong danh sách *documents* vào collection *collection_name*

#### 1.4.3.2 Read

- ```
  show dbs
  ```

  xuất cái database hiện có.

- ```
  show collections
  ```

  xuất danh sách các collection trong database hiện tại.

- ```
  db.collection_name.find()
  ```

  xuất tất cả document trong collection *collection_name*.

- ```
  db.collection_name.find(filter)
  ```

  xuất tất cả document trong collection *collection_name* thỏa *filter*.

#### 1.4.3.3 Update

- ```
  db.colletion_name.updateOne(filter, action)
  ```

  thực hiện update được mô tả trong *action* trên document thỏa *filter* trong *collection_name*. Nếu có nhiều hơn 1 document thỏa filter thì sẽ thực hiện *action* trên document đầu tiền thỏa.

- ```
  db.colletion_name.updateMany(filter, action)
  ```

  thực hiện update được mô tả trong *action* trên tất cả document thỏa *filter* trong *collection_name*.
- ```
  db.colletion_name.replaceOne(filter, document)
  ```

  thay thế document thỏa *filter* trong *collection_name* bằng *document*. Nếu có nhiều hơn 1 document thỏa *filter* trong collection thì thực hiện thay thế trên document đầu tiên thỏa.

#### 1.4.3.4 Delete

- ```
  db.dropDatabase()
  ```

  dùng để xóa database đang truy cập.
- ```
  db.collection_name.drop()
  ```

  dùng để drop collection có tên *collection_name*
- ```
  db.colletion_name.deleteOne(filter)
  ```

  xóa một document thỏa *filter* trong *collection_name*. Nếu có nhiều hơn 1 document thỏa *filter* trong collection thì xóa document đầu tiên thỏa.
- ```
  db.colletion_name.deleteMany(filter)
  ```

  xóa tất cả document thỏa *filter* trong *collection_name*.

## 1.5 Kiến trúc lưu trữ của MongoDB

Cuối cùng, ở phần này ta sẽ đi chi tiết vào kiến trúc của MongoDB. Ta sẽ giới thiệu sơ về 2 định dạng dữ liệu JSON ở 1.5.1 và BSON ở 1.5.2. Sau đó ta sẽ đi vào cốt lõi của việc lưu trữ dữ liệu của MongoDB trong phần 1.5.3.

### 1.5.1 JSON

**JSON** (viết tắt cho *JavaScript Object Notation*) là một định dạng lưu trữ, trao đổi dữ liệu nhẹ, con người có thể dễ dàng đọc và ghi cho cả con người và máy. JSON là định dạng được lưu dưới dạng text và không phụ thuộc vào ngôn ngữ lập trình sử dụng tuy cách kí hiệu gần gũi với các ngôn ngữ họ C như C, C++, C#, Java, Javascript, Perl, Python,...

JSON được xây dựng dựa trên 2 cấu trúc dữ liệu nền tảng là:

- *Tập hợp các cặp key-value (khóa-trị)* được gọi là *object* (đối tượng).
- *Danh sách các giá trị* hay còn được gọi là *array* (mảng).

Các loại dữ liệu (trị) mà JSON có thể nhận dạng là:

- string
- number
- object
- array
- true
- false

- null

### 1.5.2 BSON

MongoDB lưu trữ dữ liệu dưới một phương thức biểu diễn nhị phân gọi là BSON. **BSON** (BinaryJSON) là phương thức mã hóa nhị phân các document với định dạng tương tự JSON. BSON còn mở rộng các kiểu dữ liệu, thêm vào một số kiểu dữ liệu mà JSON chưa có như Date, BinData.

Các thuộc tính chính của BSON:

- **Nhẹ**. Các overhead được tối giảm hết mức có thể.
- **Khả duyệt tuần tự**. BSON được thiết kế có khả năng cho phép duyệt tuần tự dữ liệu.
- **Hiệu quả**. Mã hóa và giải mã file BSON có thể được thực hiện nhanh chóng nhờ vào sử dụng các kiểu dữ liệu của C.

Ví dụ về mã hóa BSON:

```
 {"hello": "world"}
->

  \x16\x00\x00\x00                    // total
     ↪ document size
  \x02                                // 0x02 = type
     ↪ String
  hello\x00                           // field name
```

```
  \x06\x00\x00\x00world\x00           // field value
  \x00                                // 0x00 = type
     ↪ EOO ('end of object')

 {"BSON": ["awesome", 5.05, 1986]}
->
  \x31\x00\x00\x00
  \x04BSON\x00
  \x26\x00\x00\x00
  \x02\x30\x00\x08\x00\x00\x00awesome\x00
  \x01\x31\x00\x33\x33\x33\x33\x33\x33\x14\x40
  \x10\x32\x00\xc2\x07\x00\x00
  \x00
  \x00
```

### 1.5.3 Storage Engine

**Storage Engine** chính là bộ phận giữ vai trò quản lý lưu trữ dữ liệu trên bộ nhớ chính cũng như trên đĩa. MongoDB cho phép người dùng chọn các storage engine khác nhau trong nhiều trường hợp khác nhau. Việc chọn storage engine phù hợp với mục đích sử dụng sẽ giúp cải thiện hiệu năng của hệ thống sử dụng cơ sở dữ liệu.

#### 1.5.3.1 WiredTiger

**WiredTiger** là storage engine mặc định kể từ MongoDB 3.2 trở đi. WiredTiger xuất hiện và thể hiện được sự mạnh mẽ của mình, qua đó trở thành lựa chọn thông dụng trong việc xây dựng CSDL.

WiredTiger kiểm soát song song hóa ở tầng document (**Document-level Concurrency Control**) cho các tác vụ ghi. Việc này có nghĩa là nhiều user có thể ghi đồng thời nhiều document khác nhau trong cùng một collection một cách song song. Phần lớn các tác vụ đọc, ghi được WiredTiger kiểm soát bằng Optimistic concurrency control. Khi phát hiện xung đột giữa 2 tác vụ, một trong 2 sẽ phát lỗi ghi dữ liệu khiến MongoDB sẽ thử thực hiện lại tác vụ đó sau.

Bên cạnh đó WiredTiger còn sử dụng **MultiVersion Concurrency Control** (MVCC). Khi bắt đầu tác vụ, WiredTiger sẽ tạo một **snapshot** - bản sao dữ liệu của cơ sở dữ liệu - thể hiện một phần nhỏ cơ sở dữ liệu vào trong bộ nhớ chính. Khi ghi xuống đĩa, WiredTiger ghi tất cả dữ liệu trong snapshot xuống đĩa. Dữ liệu lúc này đã trở nên bền vững và được xem là một checkpoint. **Checkpoint** có nhiệm vụ đảm bảo tính nhất quán của các file dữ liệu của cơ sở dữ liệu. Từ phiên bản *3.6* thì WiredTiger tạo checkpoint mỗi 60 giây. Trong khi đang ghi checkpoint mới thì checkpoint cũ hợp lệ sẽ vẫn được giữ nên nếu có trường hợp xảy ra lỗi khi đang ghi dữ liệu xuống đĩa thì ta vẫn có thể khôi phục lại đến checkpoint cũ. Checkpoint mới chỉ được xem là bền vững khi bảng metadata của WiredTiger được cập nhật đồng nhất để tham chiếu đến checkpoint mới, khi đó checkpoint cũ mới được xóa.

Tuy với WiredTiger ta có thể khôi phục đến checkpoint cuối cùng nhưng để khôi phục các thay đổi sau checkpoint đó thì ta cần sử dụng thêm ghi log.

WiredTiger sử dụng *write-ahead transaction log* - tức là các thay đổi tạo ra do các transaction được log ghi nhận chỉ được phép ghi xuống đĩa nếu log đó đã được ghi xuống đĩa.

Với WiredTiger, MongoDB hỗ trợ nén dữ liệu cho tất cả collection và index. Nén giữ liệu sử dụng thêm năng suất từ CPU đổi lại giúp giảm dung lượng sử dụng của cơ sở dữ liệu. Mặc định WiredTiger dùng thư viện *snappy* để nén các block dữ liệu và file log. Bên cạnh đó ta có thể chọn không nén dữ liệu hoặc nén bằng thuật toán khác như *zlib*.

#### 1.5.3.2 MMAPv1

**MMAPv1** là storage engine cơ bản của MongoDB dựa trên memory mapped file với các ưu điểm về hoạt động tốt với các tác vụ insert, reads số lượng lớn và in-place update.

Để đảm bảo tính bền vững của cơ sở dữ liệu, MongoDB mặc định ghi log tất cả các thay đổi vào file log (journal) trên đĩa. MongoDB ghi log thường xuyên hơn là ghi dữ liệu xuống đĩa. Mặc định ghi dữ liệu xuống đĩa mỗi 60 giấy còn ghi log mỗi 100 mili giây. Chính việc ghi log giúp MongoDB khôi phục dữ liệu khi xảy ra sự cố.

Các record sẽ được cấp phát bộ nhớ liên tục trên đĩa, dùng để lưu trữ các document. Nếu document trở nên lớn hơn kích thước record đã được cấp phát thì record mới sẽ được cấp phát và document sẽ được dời sang record mới đồng thời cập nhất các tham chiếu index tới document đó.

#### 1.5.3.3 So sánh WiredTiger và MMAPv1

- **Song song**

MMAPv1 sử dụng Collection-level Concurrency Control. Nếu một người dùng thực hiện tác vụ thay đổi một document thì cả collection của document đó sẽ bị lock và các người dùng khác sẽ không thể truy cập collection đó. Ở các phiên bản trước thì MMAPv1 dùng Database-level Concurrency Control.

Còn WiredTiger sử dụng Document-level, Optimistic Concurrency Control.

- **Logging (Journal)**

  MMAPv1 dùng tính năng này để khôi phục.

  WiredTiger sử dụng các checkpoint để khôi phục dữ liệu, có thể dùng thêm logging để khôi phục các thay đổi chưa được tạo checkpoint.

- **Nén dữ liệu**

  MMAPv1 không nén dữ liệu trong khi WiredTiger có thể nén dữ liệu theo giải thuật snappy hay zlib.

#### 1.5.3.4 In-memory Storage Engine

Như tên gọi, **In-memory Storage Engine** lưu trữ toàn bộ dữ liệu, dữ kiện thiết lập, index,... trên bộ nhớ chính. Vì thế giảm thiểu lượng lớn thời gian hao tổn do các tác vụ I/O trên đĩa. In-memory Storage Engine chỉ dành cho phiên bản *MongoDB Enterprise*.

In-memory Storage Engine dùng Document-level Concurrency Control.

Do toàn bộ dữ liệu chỉ được lưu trữ trên bộ nhớ chính nên đổi lại năng suất, tốc độ cao thì CSDL không có tính bền vững.

# Chương 2

# Thiết kế cơ sở dữ liệu với MongoDB

## 2.1 Index trong MongoDB

[2]

### 2.1.1 Index là gì?

Trong cơ sở dữ liệu, ta đã biết Index(chỉ mục) là một file cấu trúc lưu trữ trên ổ cứng tương ứng với một table hoặc một view nào đó trong cơ sở dữ liệu nhằm mục đích cải thiện tốc độ truy xuất dữ liệu từ table hoặc view đó.

Ở mục này, chúng ta sẽ tìm hiểu cách đánh index cũng như các loại index có trong MongoDB.

### 2.1.2 Các loại index

#### 2.1.2.1 Compound index (Chỉ mục hỗn hợp các khóa)

Đây là chỉ mục được đánh trên nhiều trường khác nhau trong database. Loại index nay hữu ích khi chúng ta thực hiện câu truy vấn trên đòi hỏi nhiều trường trên dữ liệu. Ví dụ câu lệnh sau đây giúp tạo chỉ mục hỗn hợp các khóa trên MongoDB.

Trong câu lệnh trên, ta giả sử có một thư mục data có nhiều trường trong

```
> db.users.ensureIndex({"age" : 1, "username" : 1})
```

Hình 2.1: Câu lệnh tạo index hỗn hợp các khóa

đó có trường username và age. Câu lệnh trên sẽ tạo ra chỉ mục đánh trên hai trường username và age đó. Số ứng với mỗi khóa trong câu lệnh chỉ mục trên thể hiện hướng của dữ liệu trên trường đó được sắp xếp như thế nào, với 1 là tăng dần, -1 là giảm dần. Trong ví dụ này thì age và username đều được đánh chỉ mục tăng dần.

Nếu ta có một tập dữ liệu gồm mà mỗi document trong dữ liệu đó bao gồm username có các giá trị như *user01, user02, user03...* và age bao gồm các giá trị nguyên như *1, 2, 3, 4....* Thì chỉ mục sẽ có hình dạng như thế này  Mỗi


Hình 2.2: Kết quả

index entry sẽ chứa lần lượt các giá trị của các trường là age ứng với giá trị nguyên và trường username ứng với các username *user1, user2,....* Và trong file index ấy do câu lệnh tạo index phía trên với trường age đứng trước tiên nên các entry được xếp tăng dần theo trường age và tiếp đến là tăng dần theo trường username có nghĩa là trước tiên chúng sắp xếp tăng dần theo trường age là số tuổi, tiếp đó với các user có cùng số tuổi thì sẽ sắp xếp tăng dần theo username. Bên cạnh đó, mỗi entry còn trỏ đến địa chỉ *(được biểu diễn bằng chuỗi Hex)* của document được lưu trên đĩa.

#### 2.1.2.2 Unique index (Chỉ mục duy nhất)

Với các hệ cơ sở dữ liệu như oracle, mySQL,... ta có primary index là dạng index đối với các trường có các giá trị không trùng nhau(unique). Với mongoDB, ta có Unique index hay còn gọi là chỉ mục duy nhất, các trường được đánh index này không được trùng nhau. Trong collection của mongoDB, trường "id" sẽ được tự động đánh là unique index.

- **Compound unique index:** đây là sự kết hợp chỉ mục duy nhất với chỉ mục hỗn hợp các khóa. Cụ thể là, nếu có hai trường như name và address được đánh compound unique index thì một trong hai trường có thể có giá trị trùng nhau giữa các document nhưng không có document nào mà giống nhau ở cả hai trường đã được đánh index.

- **Các giá trị null:** trong quá trình insert dữ liệu, nếu một trường không có giá trị nào khi insert thi mongoDB sẽ mặc định giá trị đó là null. Nhưng nếu trường nói trên được đánh chỉ mục duy nhất thì nếu có thêm một trường bị gán giá trị null thì sẽ báo lỗi insert do trùng giá trị null.

#### 2.1.2.3 Sparse index (Chỉ mục thưa)

Ở phần trên đề cập đến việc những trường được đánh chỉ mục duy nhất nhưng lại xuất hiện giá trị null. Trong những trường hợp có một trường nào đó trong collection mà có hoặc không cần insert giá trị nhưng bắt buộc trường đó là unique thì khi đó ta tạo chỉ mục duy nhất kết hợp với option sparse(thưa).

Ví dụ dưới đây là câu lệnh thực hiện việc tạo ra chỉ mục duy nhất trên trường email mà trường này không nhất thiết phải có giá trị. Các giá trị cấu hình cho unique và sparse lần lượt là "1" và "true":

```
> db.ensureIndex({"email" : 1}, {"unique" : true, "sparse" : true})
```

Hình 2.3: tạo chỉ mục duy nhất kết hợp với option sparse

## 2.2 Aggregation

[3] Sau khi data đã được lưu và thực hiện tạo index cần thiết trên mongoDB, trong công việc chúng ta có thể cần thực hiện nhiều tác vụ như phân tích theo nhiều cách khác nhau. Chính vì vậy mà mongoDB đã cung cấp một số công cụ để thực hiện các việc đó.

### 2.2.1 Aggregation framework

Aggragation có thể hiểu là sự tập hợp. Các Aggregation operation xử lý các bản dữ liệu và trả về kết quả đã được tính toán thành công. Các phép toán Aggregation nhóm các giá trị từ nhiều Document lại với nhau sau đó thực hiện nhiều phép toán đa dạng khác để trả về một kết quả duy nhất. Nói tới đây chúng ta liên tưởng tới phép toán count() và GROUPBY trong các hệ cơ sở dữ liệu SQL, các phép toán này tương ứng với Aggregation trong mongoDB.

Với Aggregation, mongoDB có hỗ trợ việc sử dụng thực hiện pipeline, có nghĩa là cho phép tạo ra một hoạt động hay tiến trình trên một số input và sử dụng output đó trở thành input cho các lệnh tiếp theo. Có một tập hợp các trạng thái có thể có, mỗi giai đoạn trên lấy một tập các document và tạo kết quả mà kết quả này được sử dụng cho các giai đoạn tiếp theo. **Các giai đoạn trong Aggregation:**

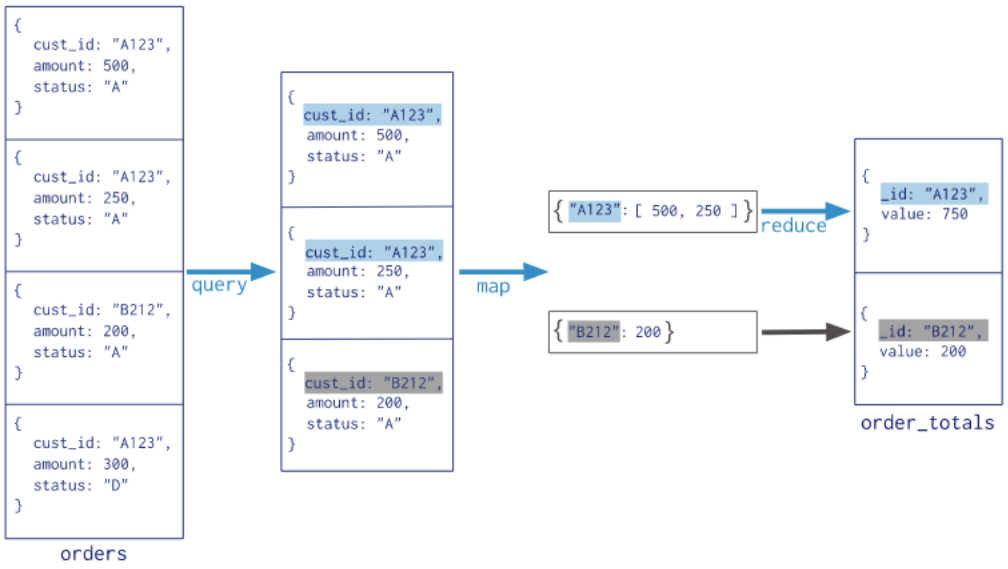
- $match: công dụng như câu lệnh where trong các hệ cơ sở dữ liệu thông thường. Nó là hoạt động lọc giúp giảm số lượng document.
- $project: giúp chọn một số trường cụ thể trong collection.

- $group: thực hiện tập hợp lại các bản dữ liệu như phần đầu mục đã trình bày.
- $sort: sắp xếp các document
- $limit: giới hạn số document
- $skip: Bỏ qua hay nhảy qua số document đã cung cấp.
- $unwind: chia một document thành nhiều document. Tạo ra một số lượng document cho các giai đoạn khác nhau.

### 2.2.2 MapReduce

MapReduce là hệ xử lý dữ liệu để tổng hợp (cô đọng) một khối lượng dữ liệu lớn thành các kết quả tổng thể có ích. Tuy nhiên đây là một công việc tốn kém về thời và không nên sử dụng cho việc phân tích dữ liệu real-time(thơi gian thực). Dưới đây là minh họa cho công việc MapReduce.

Như hình trên, MapReduce gồm 2 bước chính. Thứ nhất là bước map, bước




Hình 2.4: MapReduce

[5]

này mục đích là gộp các giá trị của các document khác nhau nhưng có chung một Key, như hình trên gom các giá trị của các document có key là "A123" lại với nhau và tương tự với key "B212". Bước tiếp theo là Reduce có nghĩa là rút gọn và gom hay tổng các giá trị cùng một Key được map ở bước trên (Hình minh họa)

## 2.3 Các đặc điểm khi thiết kế cơ sở dữ liệu với MongoDB

[4]

### 2.3.1 Normalization và Denormalization(Chuẩn hóa và không chuẩn hóa

**Normalization (chuẩn hóa).** Khi thiết kế một hệ cơ sở dữ liệu nhằm mục đích trình bày cấu trúc của dữ liệu, chuẩn hóa là một trong những yếu tố quan trọng nhất. Chuẩn hóa trong store document là việc chia nhỏ dữ liệu ra nhiều collection và việc tham chiếu lẫn nhau giữa các collection. Vì thế nên có thể nhiều document có thể tham chiếu tới một mẫu dữ liệu nào trong một collection. Do tính chất đó mà khi cần thêm vào hay thay đổi dữ liệu được trỏ tới thì chỉ cần thay đổi tại một nơi. Tuy nhiên trong mongoDB thì việc query để thu thập các document sẽ là một câu query mà chứa nhiều câu query nhỏ.

**Denormalization (Không chuẩn hóa).** Ngược lại với chuẩn hóa, không chuẩn hóa không chia nhỏ dữ liệu hay nhiều document thao khảo thới một collection nào đó mà là việc nhúng một mẫu document này trong hoàn toàn một document khác, điều nay khiến cho việc update hay insert diễn ra khác khó khăn và phức tạp. Tuy nhiên không chuẩn hóa lại giúp việc đọc hay truy xuất dữ liệu nhanh hơn.

Do các yếu tố đặc điểm của store document trên mà việc lựa chọn chuẩn hóa hay không chuẩn hóa dữ liệu là một quyết định khó khăn. Trong những

trường hợp cụ thể mà người thiết kế schema dưới dạng store document cần tính toán sao cho chuẩn hóa và không chuẩn hóa có lợi nhất.

### 2.3.2 Ví dụ về trình bày dữ liệu trong store document

Dưới đây la một ví dụ chuẩn hóa dữ liệu. Có thể thấy đây là một docu-

```
{
    "_id" : ObjectId("512512c1d86041c7dca81915"),
    "studentId" : ObjectId("512512a5d86041c7dca81914"),
    "classes" : [
        ObjectId("512512ced86041c7dca81916"),
        ObjectId("512512dcd86041c7dca81917"),
        ObjectId("512512e6d86041c7dca81918"),
        ObjectId("512512f0d86041c7dca81919")
    ]
}
```

Hình 2.5: data representation 1

ment chứa thông tin một sinh viên, trường id là biểu diễn id cho document đó ta bỏ qua, trường **studentId** chứa giá trị là một chuỗi Hex và có trường **classes** là một mảng chứa các đối tượng lưu những mã Hex khác nhau. Giờ đến với hình minh họa tiếp theo. Với document này, ta chú ý đến trường **_id**

```
{
    "_id" : ObjectId("512512a5d86041c7dca81914"),
    "name" : "John Doe",
    "classes" : [
        ObjectId("512512ced86041c7dca81916"),
        ObjectId("512512dcd86041c7dca81917"),
        ObjectId("512512e6d86041c7dca81918"),
        ObjectId("512512f0d86041c7dca81919")
    ]
}
```

Hình 2.6: data representation 2

giá trị Hex của trường này trùng với giá trị Hex trong trường **studentId** của hình 2.5. Có nghĩa rằng trường **studentId** thao khảo đến document 2.6 la một sinh viên nào đó tên "John Doe". Cứ theo ý nghĩa trên, ta có thể nhận thấy rằng giá giá trị Hex trong mảng **classes** sẽ tham khảo đến một document nào đó mang thông tin lớp học mà có **_id** trùng với giá trị mà nó mang.

Nếu muốn biểu diễn dữ liệu để tối ưu việt đọc truy xuất mà không quan tâm đến chi phí update thì dữ liệu thường được biểu diễn mọi thông tin class hay tên họ của sinh viên sẽ được lưu thẳng trực tiếp vào một document như sau mà không cần sử dụng việc tham khảo như trên. Đây là không chuẩn hóa đã được trình bày lý thuyết ở mục trên.

```
{
    "_id" : ObjectId("512512a5d86041c7dca81914"),
    "name" : "John Doe",
    "classes" : [
        {
            "class" : "Trigonometry",
            "credits" : 3,
            "room" : "204"
        },
        {
            "class" : "Physics",
            "credits" : 3,
            "room" : "159"
        },
        {
            "class" : "Women in Literature",
            "credits" : 3,
            "room" : "14b"
        },
        {
            "class" : "AP European History",
            "credits" : 4,
            "room" : "321"
        }
    ]
}
```

Hình 2.7: dữ liệu không chuẩn hóa

# Chương 3

# Sharding

Sharding là một phương pháp phân phối dữ liệu trên nhiều máy. MongoDB sử dụng sharding để hỗ trợ triển khai với các tập dữ liệu rất lớn và các hoạt động thông lượng cao.

Các hệ thống cơ sở dữ liệu với các tập dữ liệu lớn hoặc các ứng dụng thông lượng cao có thể thách thức khả năng của một máy chủ đơn lẻ. Ví dụ, tỷ lệ truy vấn cao có thể làm cạn kiệt dung lượng CPU của máy chủ. Các kích thước bộ làm việc lớn hơn RAM của hệ thống làm tăng dung lượng I / O của ổ đĩa.

Có hai phương pháp để giải quyết sự tăng trưởng hệ thống: mở rộng theo chiều dọc và chiều ngang.

Chia tỷ lệ theo chiều dọc liên quan đến việc tăng dung lượng của một máy chủ đơn lẻ, chẳng hạn như sử dụng CPU mạnh hơn, tăng thêm RAM hoặc tăng dung lượng bộ nhớ. Các hạn chế trong công nghệ có sẵn có thể hạn chế một máy duy nhất không đủ mạnh cho một khối lượng công việc đã cho. Ngoài ra, các nhà cung cấp dựa trên đám mây có trần cứng dựa trên

các cấu hình phần cứng có sẵn. Kết quả là, có tối đa thực tế cho việc mở rộng theo chiều dọc.

Chia tỷ lệ theo chiều ngang liên quan đến việc chia bộ dữ liệu hệ thống và tải trên nhiều máy chủ, thêm máy chủ bổ sung để tăng dung lượng theo yêu cầu. Mặc dù tốc độ hoặc dung lượng tổng thể của một máy đơn có thể không cao, mỗi máy sẽ xử lý một tập hợp con của khối lượng công việc tổng thể, có khả năng mang lại hiệu quả tốt hơn so với một máy chủ dung lượng cao tốc độ cao duy nhất. Việc mở rộng khả năng triển khai chỉ yêu cầu thêm các máy chủ bổ sung nếu cần, có thể thấp hơn tổng chi phí so với phần cứng cao cấp cho một máy. Cân sự phức tạp trong cơ sở hạ tầng và bảo trì cho việc triển khai.

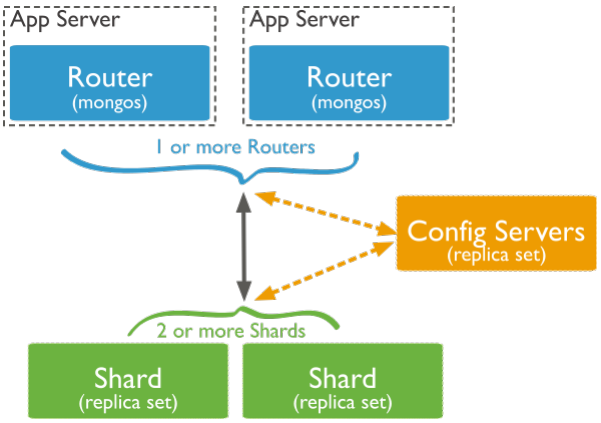
MongoDB hỗ trợ mở rộng theo chiều ngang thông qua sharding.

## 3.1 Sharded Cluster

Một cụm MongoDB shard bao gồm các thành phần sau:

- shard: Mỗi shard chứa một tập con của dữ liệu được shard. Mỗi shard có thể được triển khai như một bộ bản sao.
- mongos: Các mongos hoạt động như một bộ định tuyến truy vấn, cung cấp một giao diện giữa các ứng dụng máy khách và cụm shard.
- máy chủ cấu hình: Máy chủ cấu hình lưu trữ cài đặt siêu dữ liệu và cấu hình cho cụm. Theo MongoDB 3.4, các máy chủ cấu hình phải được triển khai như một bộ bản sao (CSRS).

Đồ họa sau đây mô tả sự tương tác của các thành phần trong một cụm shard:




Hình 3.1: Sơ đồ của một cụm mẫu shard cho mục đích sản xuất. Chứa chính xác 3 máy chủ cấu hình, 2 hoặc nhiều bộ định tuyến truy vấn '*mongos*' và ít nhất 2 mảnh. Các mảnh là bộ bản sao.

MongoDB phân chia dữ liệu ở cấp thu thập, phân phối dữ liệu thu thập trên các shard trong cụm.

## 3.2 Shard Keys

Để phân phối các tài liệu trong một bộ sưu tập, MongoDB phân vùng bộ sưu tập bằng cách sử dụng khóa shard. Khóa shard bao gồm một trường hoặc trường bất biến tồn tại trong mọi tài liệu trong bộ sưu tập đích.

Bạn chọn khoá shard khi sharding một bộ sưu tập. Không thể thay đổi khóa shard sau khi sharding. Bộ sưu tập được phân loại có thể chỉ có một khóa shard.

Để shard một bộ sưu tập không trống, bộ sưu tập phải có chỉ mục bắt đầu bằng khóa shard. Đối với các bộ sưu tập rỗng, MongoDB tạo chỉ mục nếu bộ sưu tập không có chỉ mục thích hợp cho khóa shard đã chỉ định.

Sự lựa chọn của shard key ảnh hưởng đến hiệu suất, hiệu quả và khả năng mở rộng của một cluster bị phân mảnh. Một cụm với phần cứng và cơ sở hạ tầng tốt nhất có thể bị tắc nghẽn bởi sự lựa chọn của khóa shard. Việc lựa chọn mã khóa và chỉ số sao lưu của nó cũng có thể ảnh hưởng đến chiến lược tích trữ mà cụm của bạn có thể sử dụng.

## 3.3 Chunk

Các phân vùng MongoDB phân chia dữ liệu thành các đoạn. Mỗi đoạn có một cận trên và dưới dựa trên khoá shard.

MongoDB di chuyển các đoạn trên các mảnh trong cluster bị phân mảnh bằng cách sử dụng cân bằng cluster bị phân mảnh. Các cân bằng cố gắng để đạt được một sự cân bằng thậm chí của khối trên tất cả các mảnh trong cụm.

## 3.4 Ưu điểm của Sharding

**Đọc / ghi** MongoDB phân phối khối lượng công việc đọc và ghi trên các shard trong cụm shard, cho phép mỗi shard xử lý một tập con của các phép toán cụm. Cả hai khối lượng công việc đọc và ghi có thể được thu nhỏ theo chiều ngang trên cụm bằng cách thêm nhiều shard hơn.

Đối với các truy vấn bao gồm khóa shard hoặc tiền tố của khóa shard phức hợp, mongos có thể nhắm mục tiêu truy vấn tại shard cụ thể hoặc tập hợp các shard. Các hoạt động được nhắm mục tiêu này thường hiệu quả hơn là phát sóng tới mọi shard trong cụm.

**Khả năng lưu trữ** Sharding phân phối dữ liệu trên các shard trong cụm, cho phép mỗi shard chứa một tập hợp con của tổng số dữ liệu cụm. Khi tập dữ liệu phát triển, các shard bổ sung sẽ tăng dung lượng lưu trữ của cụm.

**Tính sẵn sàng cao** Một cụm shard có thể tiếp tục thực hiện một phần hoạt động đọc / ghi ngay cả khi một hoặc nhiều shard không có sẵn. Mặc dù tập hợp con dữ liệu trên các shard không có sẵn không thể truy cập được trong thời gian ngừng hoạt động, nhưng việc đọc hoặc viết trực tiếp tại các shard có sẵn vẫn có thể thành công.

Trong môi trường sản phẩm, shard cá nhân nên được triển khai dưới dạng bản sao, cung cấp khả năng dự phòng và tính khả dụng cao hơn.

## 3.5 Cân nhắc trước khi Sharding

Yêu cầu cơ sở hạ tầng cụm phức tạp và yêu cầu phức tạp đòi hỏi phải lập kế hoạch, thực hiện và bảo trì cẩn thận.

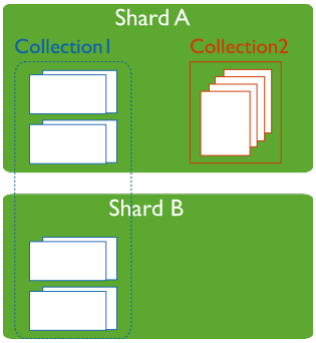
Cân nhắc cẩn thận trong việc chọn khóa shard là cần thiết để đảm bảo hiệu suất và hiệu quả của cụm. Bạn không thể thay đổi khóa shard sau khi sharding, cũng không thể unshard một bộ sưu tập sharded. Xem Chọn một khóa shard.

Sharding có một số yêu cầu và hạn chế hoạt động nhất định. Xem các hạn chế hoạt động trong các cụm đã shard để biết thêm thông tin.

Nếu các truy vấn không bao gồm khóa shard hoặc tiền tố của khóa shard hợp chất, mongos thực hiện một hoạt động phát sóng, truy vấn tất cả các shard trong cụm bị phân mảnh. Các truy vấn phân tán / thu thập này có thể là các hoạt động chạy dài.

## 3.6 Sharded and Non-Sharded Collections

Một cơ sở dữ liệu có thể có một hỗn hợp các bộ sưu tập bị phân mảnh và không bị trầy xước. Các bộ sưu tập được phân chia được shard và phân phối trên các shard trong cụm. Bộ sưu tập chưa được lưu trữ được lưu trữ trên shard chính. Mỗi cơ sở dữ liệu có shard chính của riêng nó.

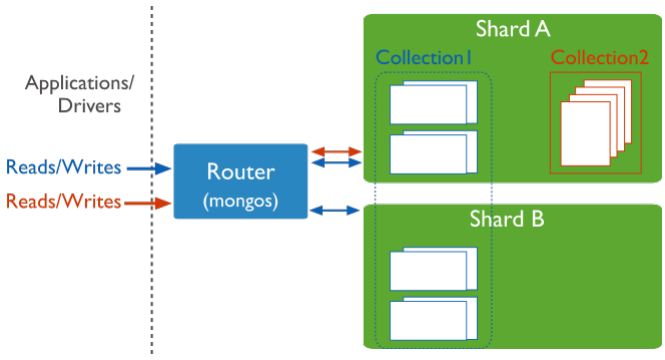



Hình 3.2: Sơ đồ shard chính. Phân đoạn chính chứa các bộ sưu tập không bị shard cũng như các khối tài liệu từ các bộ sưu tập được phân loại. Shard A là shard chính.

## 3.7 Kết nối với một cụm được shard

Bạn phải kết nối với bộ định tuyến mongos để tương tác với bất kỳ bộ sưu tập nào trong cụm được shard. Điều này bao gồm các bộ sưu tập bị phân tán và không bị trói buộc. Khách hàng không bao giờ nên kết nối với một mảnh duy nhất để thực hiện các hoạt động đọc hoặc ghi.

Bạn có thể kết nối với một mongos giống như cách bạn kết nối với một Mongod, chẳng hạn như thông qua trình bao mongo hoặc trình điều khiển MongoDB.



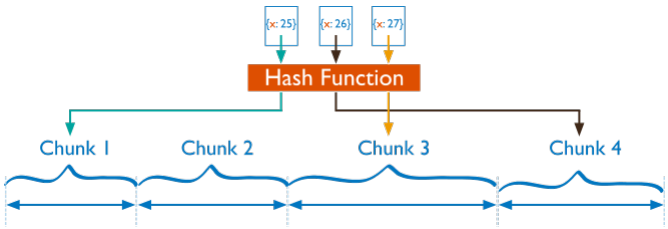

Hình 3.3: Sơ đồ các ứng dụng / trình điều khiển đưa ra các truy vấn tới mongos cho bộ sưu tập không bị quản lý cũng như bộ sưu tập được phân loại. Máy chủ cấu hình không được hiển thị. Bạn có thể kết nối với một mongos giống như cách bạn kết nối với một Mongod, chẳng hạn như thông qua trình bao mongo hoặc trình điều khiển MongoDB.

## 3.8 Sharding Strategy

MongoDB hỗ trợ hai chiến lược sharding để phân phối dữ liệu trên các cụm bị phân mảnh.

**Hashed Sharding** Hashed Sharding liên quan đến việc tính toán giá trị băm của giá trị của trường khóa. Mỗi đoạn sau đó được gán một phạm vi dựa trên các giá trị khóa shard băm.

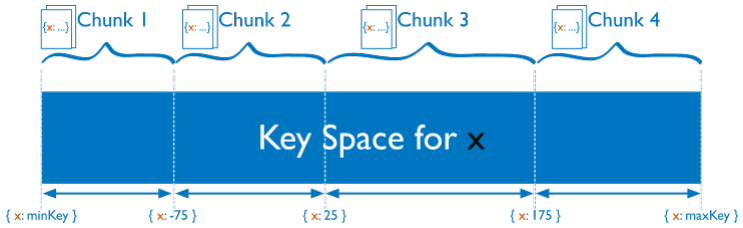
Mặc dù một loạt các khóa shard có thể là "đóng", nhưng các giá trị băm của chúng không thể ở cùng một đoạn. Phân phối dữ liệu dựa trên các giá trị băm tạo điều kiện phân phối dữ liệu thậm chí nhiều hơn, đặc biệt là trong các tập dữ liệu trong đó khóa phân tử thay đổi một cách đơn điệu.




Hình 3.4: Sơ đồ shard dựa trên băm.

Tuy nhiên, phân phối băm có nghĩa là các truy vấn dựa trên phạm vi trên khóa shard ít có khả năng nhắm mục tiêu một shard đơn lẻ, dẫn đến nhiều hoạt động phát rộng hơn theo cụm

**Ranged Sharding** Sự phân rã có phạm vi liên quan đến việc chia dữ liệu thành các phạm vi dựa trên các giá trị khóa shard. Mỗi đoạn sau đó được gán một phạm vi dựa trên các giá trị khóa shard.




Hình 3.5: Sơ đồ không gian giá trị khóa shard được chia thành các phạm vi hoặc khối nhỏ hơn.

Một loạt các khóa shard có giá trị "đóng" có nhiều khả năng cư trú trên cùng một đoạn. Điều này cho phép các hoạt động được nhắm mục tiêu như một mongos có thể định tuyến các hoạt động đến chỉ các mảnh có chứa dữ liệu cần thiết.

Hiệu quả của sharding dao động phụ thuộc vào phím shard được chọn.

Các khóa shard kém được xem là có thể dẫn đến phân phối dữ liệu không đồng đều, có thể phủ nhận một số lợi ích của việc tích trữ hoặc có thể gây ra tắc nghẽn hiệu suất.

## 3.9 Zones in Sharded Clusters

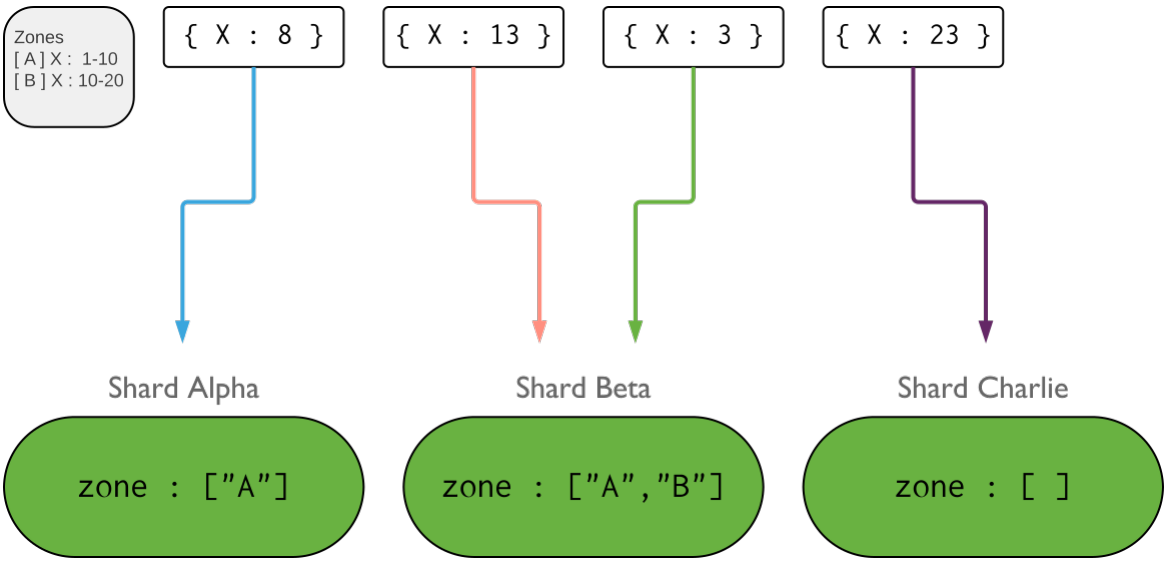
Trong các cụm shard, bạn có thể tạo các vùng dữ liệu được phân loại dựa trên khóa shard. Bạn có thể kết hợp từng vùng với một hoặc nhiều mảnh trong cụm. Phân đoạn có thể kết hợp với bất kỳ số vùng nào. Trong một cụm cân bằng, MongoDB di chuyển các khối được bao phủ bởi một vùng chỉ cho các mảnh vỡ liên kết với vùng đó.

Mỗi vùng bao gồm một hoặc nhiều dải giá trị khóa shard. Mỗi phạm vi một vùng bao gồm luôn luôn bao gồm ranh giới thấp hơn của nó và độc quyền của ranh giới trên của nó.

Bạn phải sử dụng các trường có trong khóa shard khi xác định phạm vi mới cho vùng cần bao gồm. Nếu sử dụng khóa shard hợp chất, phạm vi phải bao gồm tiền tố của khóa shard.

Khi chọn một khóa shard, hãy xem xét cẩn thận khả năng sử dụng sharding khu vực trong tương lai, vì bạn không thể thay đổi khóa shard sau khi sharding bộ sưu tập.

Phổ biến nhất, các khu vực phục vụ để cải thiện địa phương của dữ liệu cho các cụm shard trải rộng trên nhiều trung tâm dữ liệu.




Hình 3.6: Sơ đồ phân bố dữ liệu dựa trên các vùng trong một cụm shard

## 3.10 Collations in Sharding

Sử dụng lệnh shardCollection với đối chiếu: tùy chọn *locale* : "*simple*" để phân tích một bộ sưu tập có collation mặc định. Sharding thành công đòi hỏi rằng:

- Bộ sưu tập phải có chỉ mục có tiền tố là khóa shard
- Chỉ mục phải có collation *locale* : "*simple*"

Khi tạo bộ sưu tập mới với collation, đảm bảo các điều kiện này được đáp ứng trước khi sharding bộ sưu tập.

# Chương 4

# Các cơ sở dữ liệu Document store

## 4.1 Tổng quan vê CouchDB

## 4.2 Tổng quan vê SimpleDB

## 4.3 So sánh

# Tài liệu tham khảo


[1] bsonspec.org. BSON. 2018.

[2] Kristina Chodorow. MongoDB: The Definitive Guide - CHAPTER 5 - Indexing. 2013.

[3] Kristina Chodorow. MongoDB: The Definitive Guide - CHAPTER 7 - Aggregation. 2013.

[4] Kristina Chodorow. MongoDB: The Definitive Guide - CHAPTER 8 - Application Design, Normalization vesus Denormalization. 2013.

[5] https://docs.mongodb.com.

[6] MongoDB Inc. MongoDB. 2018.

[7] json.org. JSON. 2018.

[8] Dinesh Ramalingam. MongoDB MMap vs WiredTiger- A comprehensive comparison. 2015.