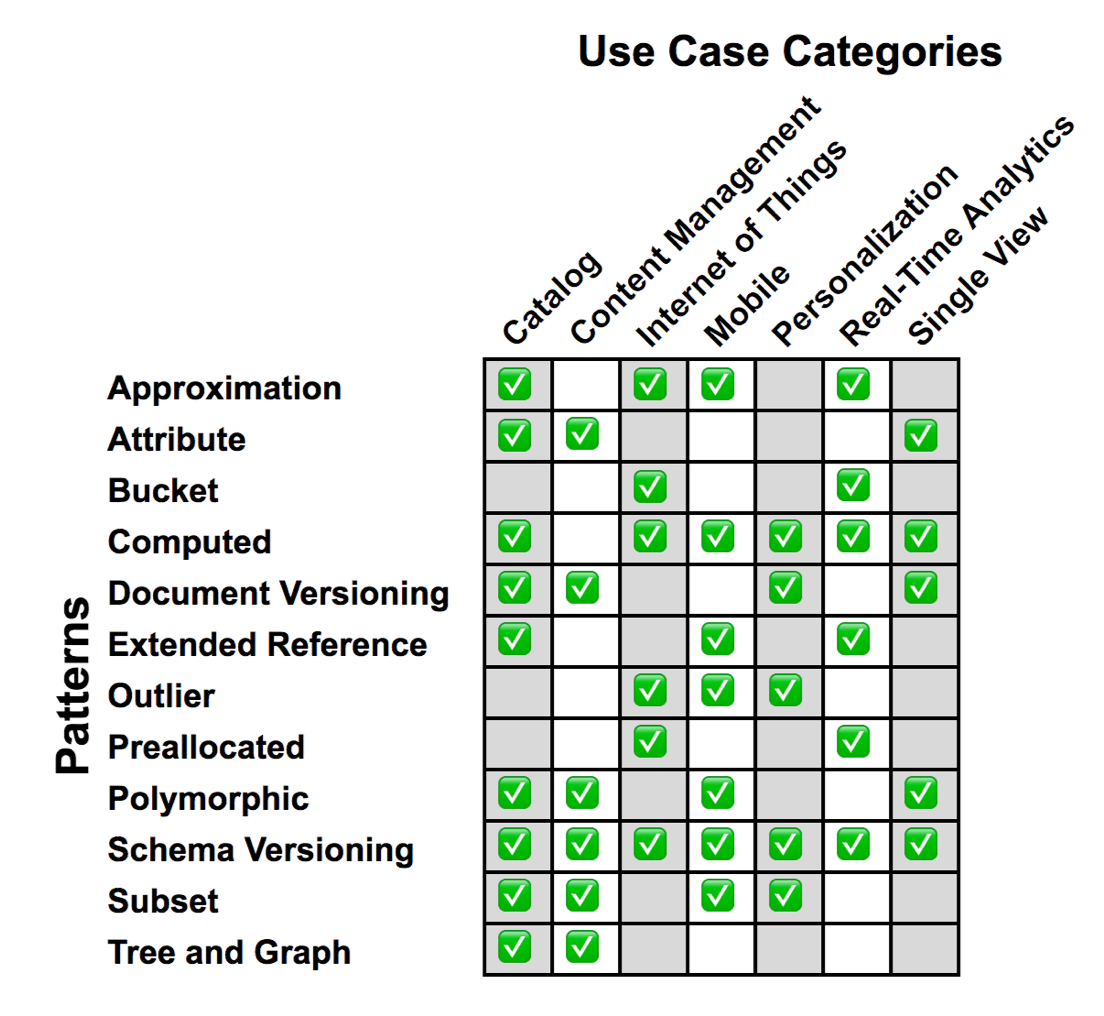
**LÝ THUYẾT: CÁC LOẠI PATTERN**

* **Attribute pattern:** Có 1 subdocument có trường key và value
* **Extended Reference patter** : Nhúng **1 vài trường** ,**toàn bộ doc** của col này vào col kia
* **Subset pattern:** Nhúng **toàn bộ trường** nhưng chỉ lấy **một vài document**
* **Computed Pattern:** lưu dữ liệu vào 1 collection riêng, có một trường được tính từ việc tính tổng hoặc gì đó từ colectiion đó, cập nhật dữ liệu lại sau mỗi phút/ngày/giờ
* **Bucket:**  Thay vì lưu từng doc riêng có thời gian, chúng ta sẽ gom nhóm dựa theo thời gian, lưu vào cùng một mảng, khi có dữ liệu phát sinh thì push vào mảng. Khi làm việc với dữ liệu chuỗi thời gian, sử dụng Bucket Pattern trong MongoDB là một lựa chọn tuyệt vời. Nó làm giảm tổng số tài liệu trong một bộ sưu tập, cải thiện hiệu suất lập chỉ mục và bằng cách tận dụng tính năng tổng hợp trước, nó có thể đơn giản hóa việc truy cập dữ liệu.
* **Tree:** có một trường parent là thằng chứa nó
* **A screenshot of a computer code

  Description automatically generatedSchema version:**

**Các trường hợp sử dụng**



**VÍ DỤ**

* **Attribute Pattern**

Hãy nghĩ về một bộ sưu tập các bộ phim. Các tài liệu có thể sẽ có các trường tương tự liên quan đến tất cả các tài liệu: tiêu đề, đạo diễn, nhà sản xuất, dàn diễn viên, v.v. Giả sử chúng ta muốn tìm kiếm vào ngày phát hành. Một thách thức mà chúng tôi phải đối mặt khi thực hiện đó là ngày phát hành nào ? Phim thường được phát hành vào những ngày khác nhau ở các quốc gia khác nhau.

**{**

**title: "Star Wars",**

**director: "George Lucas",**

**...**

**release\_US: ISODate("1977-05-20T01:00:00+01:00"),**

**release\_France: ISODate("1977-10-19T01:00:00+01:00"),**

**release\_Italy: ISODate("1977-10-20T01:00:00+01:00"),**

**release\_UK: ISODate("1977-12-27T01:00:00+01:00"),**

**...**

**}**

Việc tìm kiếm ngày phát hành sẽ yêu cầu xem xét nhiều lĩnh vực cùng một lúc. Để nhanh chóng tìm kiếm ngày phát hành, chúng tôi cần một số chỉ mục trên bộ sưu tập phim của mình:

**{release\_US: 1}**

**{release\_France: 1}**

**{release\_Italy: 1}**

Bằng cách sử dụng Mẫu thuộc tính, chúng ta có thể di chuyển tập hợp thông tin con này vào một mảng và giảm nhu cầu lập chỉ mục. Chúng tôi biến thông tin này thành một mảng các cặp khóa-giá trị:

**{**

**title: "Star Wars",**

**director: "George Lucas",**

**…**

**releases: [**

**{**

**location: "USA",**

**date: ISODate("1977-05-20T01:00:00+01:00")**

**},**

**{**

**location: "France",**

**date: ISODate("1977-10-19T01:00:00+01:00")**

**},**

**{**

**location: "Italy",**

**date: ISODate("1977-10-20T01:00:00+01:00")**

**},**

**{**

**location: "UK",**

**date: ISODate("1977-12-27T01:00:00+01:00")**

**},**

**…**

**],**

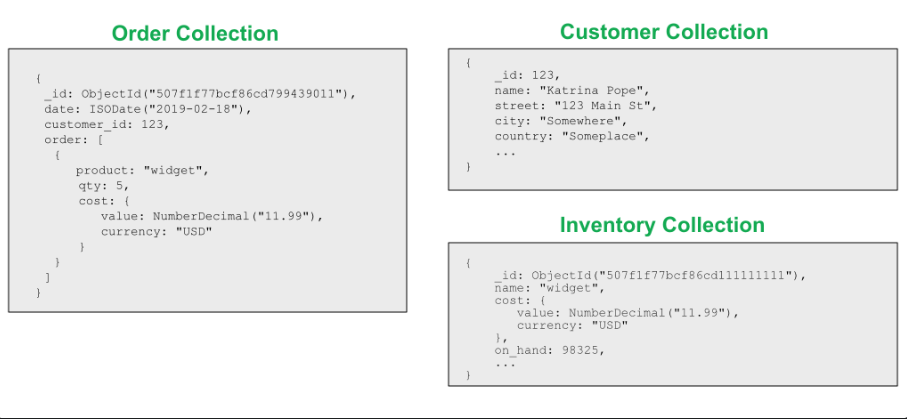
**…**

**}**

Việc lập chỉ mục trở nên dễ quản lý hơn nhiều bằng cách tạo một chỉ mục trên các phần tử trong mảng:

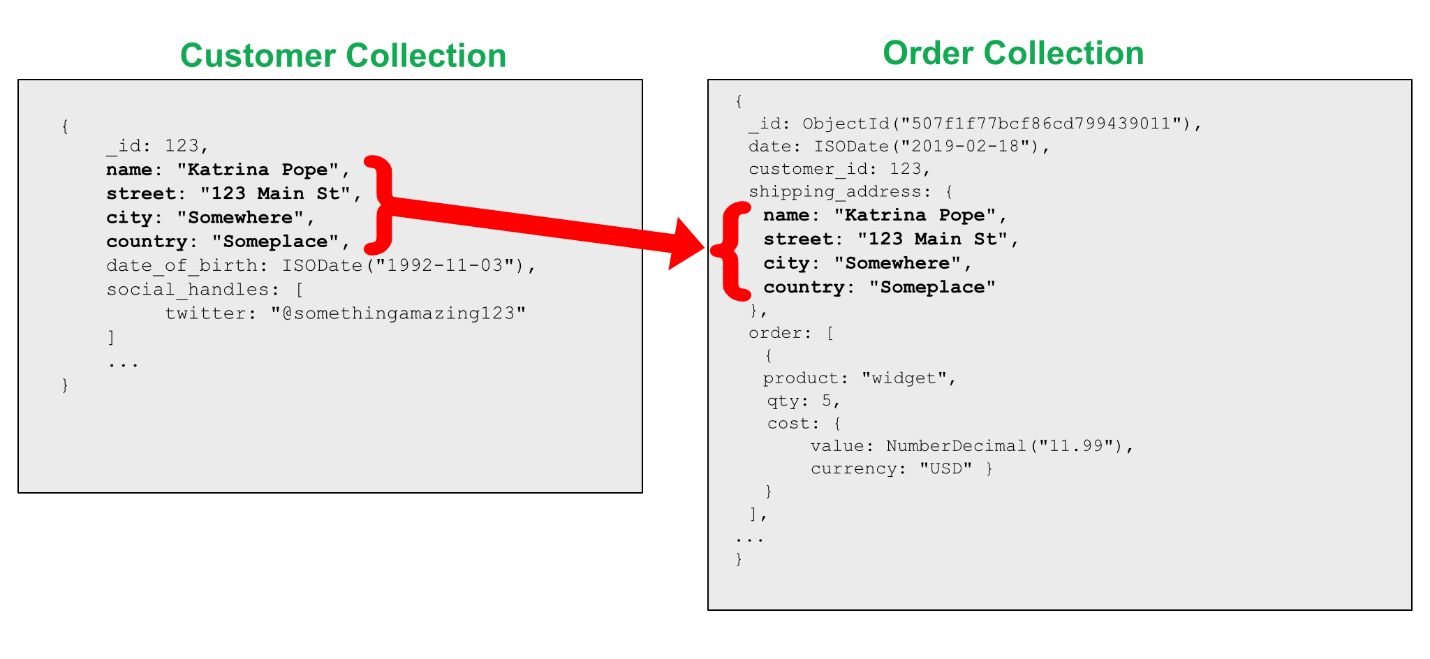
**{ "releases.location": 1, "releases.date": 1}**

* **Extented Reference Pattern**

****

Từ quan điểm hiệu suất, điều này trở thành vấn đề khi chúng ta cần ghép các phần thông tin lại với nhau theo một thứ tự cụ thể. Một khách hàng có thể có N đơn hàng, tạo nên mối quan hệ 1-N. Từ quan điểm đặt hàng , nếu chúng ta lật lại vấn đề đó, họ có mối quan hệ N-1 với khách hàng. Việc nhúng tất cả thông tin về khách hàng cho mỗi đơn hàng chỉ để giảm thao tác lookup dẫn đến nhiều thông tin trùng lặp. Ngoài ra, không phải tất cả thông tin khách hàng đều có thể cần thiết cho một đơn đặt hàng.

Mẫu Tham chiếu Mở rộng cung cấp một cách tuyệt vời để xử lý những tình huống này. Thay vì sao chép toàn bộ thông tin về khách hàng, chúng tôi chỉ sao chép những trường mà chúng tôi thường xuyên truy cập. Thay vì nhúng tất cả thông tin hoặc bao gồm tham chiếu đến lookup thông tin, chúng tôi chỉ nhúng các trường có mức độ ưu tiên cao nhất và được truy cập thường xuyên nhất, chẳng hạn như tên và địa chỉ.



* **Subset Pattern**

Mẫu này giải quyết các vấn đề liên quan đến bộ công việc vượt quá RAM, dẫn đến thông tin bị xóa khỏi bộ nhớ. Điều này thường xảy ra do các tài liệu lớn có nhiều dữ liệu không được ứng dụng thực sự sử dụng. Chính xác thì ý tôi là gì?

Hãy tưởng tượng một trang web thương mại điện tử có danh sách đánh giá về một sản phẩm. Khi truy cập dữ liệu của sản phẩm đó, rất có thể chúng tôi chỉ cần khoảng mười bài đánh giá gần đây nhất. Việc thu thập toàn bộ dữ liệu sản phẩm cùng với tất cả các đánh giá có thể dễ dàng khiến nhóm làm việc được mở rộng.

A screenshot of a computer code

Description automatically generated

Thay vì lưu trữ tất cả các đánh giá về sản phẩm, chúng ta có thể chia bộ sưu tập thành hai bộ sưu tập. Một bộ sưu tập sẽ có dữ liệu được sử dụng thường xuyên nhất, ví dụ: các đánh giá hiện tại và bộ sưu tập khác sẽ có dữ liệu được sử dụng ít thường xuyên hơn, ví dụ: các đánh giá cũ, lịch sử sản phẩm, v.v. Chúng tôi có thể sao chép một phần của mối quan hệ 1-N hoặc N-N được sử dụng bởi mặt được sử dụng nhiều nhất của mối quan hệ.



* **Bucket Pattern**

Mẫu này đặc biệt hiệu quả khi làm việc với dữ liệu Internet of Things (IoT), Phân tích thời gian thực hoặc Chuỗi thời gian nói chung. Bằng cách gộp dữ liệu lại với nhau, chúng tôi giúp việc tổ chức các nhóm dữ liệu cụ thể trở nên dễ dàng hơn, tăng khả năng khám phá các xu hướng lịch sử hoặc đưa ra dự báo trong tương lai và tối ưu hóa việc sử dụng bộ nhớ.

Ví dụ chúng ta có một cảm biến đo nhiệt độ và lưu nó vào cơ sở dữ liệu mỗi phút, luồng dữ liệu của chúng ta có thể trông giống như sau:

**{**

**sensor\_id: 12345,**

**timestamp: ISODate("2019-01-31T10:00:00.000Z"),**

**temperature: 40**

**}**

**{**

**sensor\_id: 12345,**

**timestamp: ISODate("2019-01-31T10:01:00.000Z"),**

**temperature: 40**

**}**

**{**

**sensor\_id: 12345,**

**timestamp: ISODate("2019-01-31T10:02:00.000Z"),**

**temperature: 41**

**}**

Điều này có thể gây ra một số vấn đề khi ứng dụng mở rộng về mặt dữ liệu và kích thước chỉ mục. Ví dụ cuối cùng chúng ta có thể phải lập chỉ mục cho trường **sensor\_id** và trường **timestamp** để có thể truy cập nhanh chóng với chi phí RAM thấp. Tuy nhiên, bằng cách tận dụng mô hình dữ liệu tài liệu, chúng ta có thể "gộp" dữ liệu này vào các tài liệu chứa các phép đo trong một khoảng thời gian cụ thể theo thời gian và bằng cách áp dụng **Bucket Pattern** cho mô hình dữ liệu của mình, chúng ta nhận được một số lợi ích về mặt tiết kiệm kích thước chỉ mục, đơn giản hóa truy vấn và khả năng sử dụng dữ liệu được tổng hợp trước đó trong tài liệu của chúng ta ví dụ:

**{**

**sensor\_id: 12345,**

**start\_date: ISODate("2019-01-31T10:00:00.000Z"),**

**end\_date: ISODate("2019-01-31T10:59:59.000Z"),**

**measurements: [**

**{**

**timestamp: ISODate("2019-01-31T10:00:00.000Z"),**

**temperature: 40**

**},**

**{**

**timestamp: ISODate("2019-01-31T10:01:00.000Z"),**

**temperature: 40**

**},**

**…**

**{**

**timestamp: ISODate("2019-01-31T10:42:00.000Z"),**

**temperature: 42**

**}**

**],**

**transaction\_count: 42,**

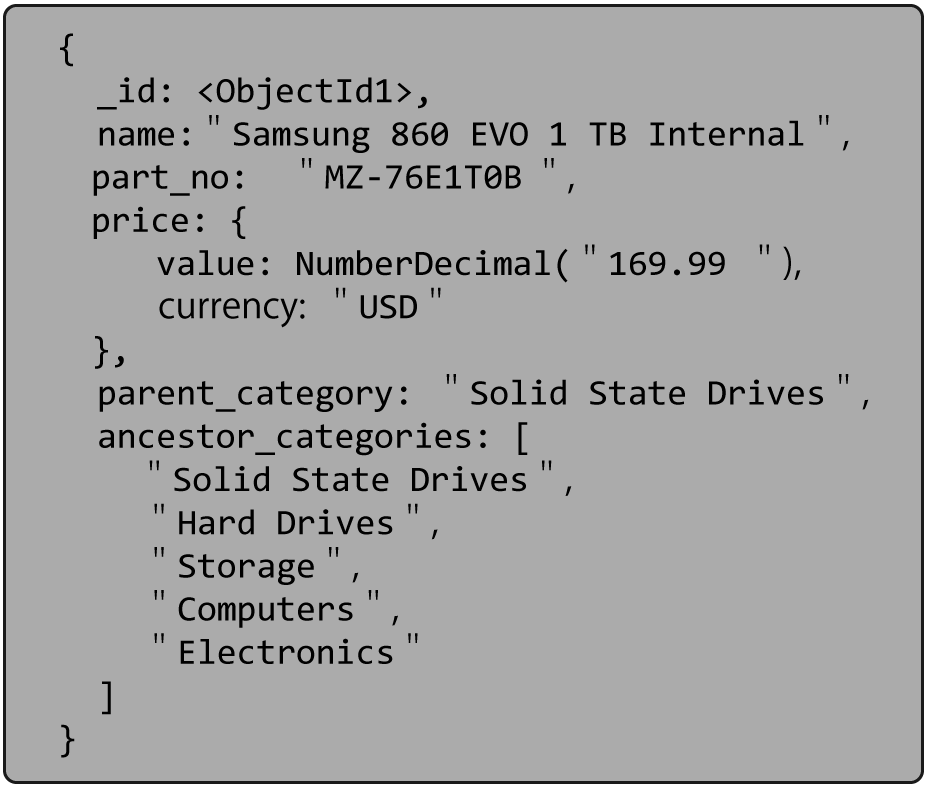
**sum\_temperature: 2413**

**}**

Bằng cách sử dụng **Bucket Pattern**, trong trường hợp này, chúng ta đã "gộp" dữ liệu của mình vào nhóm một giờ. Luồng dữ liệu cụ thể này sẽ vẫn phát triển vì hiện tại nó chỉ có 42 phép đo; vẫn còn nhiều số đo khác cho giờ đó sẽ được thêm vào "nhóm". Khi chúng được thêm vào **measurements** mảng giá trị này **transaction\_count** sẽ tăng lên và **sum\_temperature** cũng sẽ được cập nhật. Khi đó việc tính nhiệt độ trung bình cũng dễ dàng hơn (**sum\_temperature / transaction\_count**)

* **Tree Pattern**

Danh mục sản phẩm là một ví dụ rất hay khác về việc sử dụng **Tree Pattern**. Thông thường các sản phẩm thuộc về các danh mục, là một phần của các danh mục khác. Ví dụ: Ổ đĩa thể rắn có thể nằm trong Ổ đĩa cứng , bên dưới Bộ lưu trữ , bên dưới Bộ phận máy tính . Đôi khi việc tổ chức các hạng mục có thể thay đổi nhưng không quá thường xuyên.



Lưu ý trong document phía trên ancestor\_categories là trường theo dõi toàn bộ hệ thống phân cấp. Cũng có trường parent\_category. Sao chép cấp độ gốc trực tiếp trong hai trường này là phương pháp hay nhất mà mongoDB đã phát triển sau khi làm việc với nhiều khách hàng bằng cách sử dụng Tree Pattern. Việc bao gồm trường "parent" thường rất hữu ích, đặc biệt nếu bạn cần duy trì khả năng sử dụng $graphLookup trên tài liệu của mình.

Giữ tổ tiên trong một mảng cung cấp khả năng tạo multi-key index trên các giá trị đó. Nó cho phép dễ dàng tìm thấy tất cả children của một danh mục nhất định. Đối với các con trực tiếp, chúng có thể được truy cập bằng cách xem các tài liệu có danh mục nhất định của chúng tôi là "cha mẹ" trực tiếp của nó. Chúng tôi vừa nói với bạn rằng lĩnh vực này sẽ rất hữu ích.

**BÀI TẬP CHƯƠNG 4**

**Bài 1: Sách và nhà xuất bản (Books and publishers)**

Chúng tôi muốn thiết kế một CSDL MongoDB để lưu trữ dữ liệu về sách (Books) và các nhà xuất bản (Publishers).Mỗi nhà xuất bản được đặc trưng bởi tên (name), năm thành lập (year) và quốc gia (country). Một dánh sách các phương tiện để liên hệ. Các phương tiện liên hệ có thể là email, điện thoại, website. Ứng với mỗi một phương tiện liên hệ chỉ lưu một thông tin liên hệ duy nhất. Các phương tiện liên hệ có thể được thêm vào trong tương lai.

* Mỗi cuốn sách được xác định bằng mã ISBN và được đặc trưng bởi tiêu đề sách (title), tiêu đề phụ (subtitle), số trang (number of page), ngôn ngữ (language) và ngày xuất bản (publication date). Mỗi cuốn sách được xuất bản bởi duy nhất một nhà xuất bản duy nhất. Một nhà xuất bản xuất bản nhiều sách khác nhau.
* Thiết kế lược đồ lưu trữ phải được tối ưu hóa để hiển thị cho mỗi cuốn sách với tất cả thông tin của nó và tên nhà xuất bản.

Yêu cầu: Thiết kế lược đồ lưu trữ dữ liệu để đáp ứng các yêu cầu trên (chỉ ra các collection trong CSDL, cấu trúc Document, mối quan hệ, các chiến lược (các pattern) được sử dụng để thiết kế và nêu lý do tại sao dùng chiến lược đó).

**Bài làm**

1. **Các Collection.**

* Ở đây chúng ta sẽ có 2 collection: Sách (Books), Nhà Xuất Bản (Publishers)

1. **Cấu trúc Document, mối quan hệ, và pattern.**

* Cấu trúc document.

A computer code with text

Description automatically generated with medium confidence

* Books: có tiêu đề sách (title), tiêu đề phụ (subtitle), số trang (number of page), ngôn ngữ (language) và ngày xuất bản (publication date) và sách được xuất bản bởi duy nhất một nhà xuất bản cho nên sẽ có một object publisher gồm mã (id) của publisher và trường nữa là tên nhà xuất bản (name).
* Publisher: có tên (name), năm thành lập (year) và quốc gia (country), và 1 trường nữa là liên hệ (contact). Ở đây em lưu dưới dạng mảng object theo dạng attribute parrtern gồm key và value, mảng liên lạc (contacts) ở đây phải lưu tối thiểu 1 liên lạc để đảm bảo nhà xuất bản này uy tín, hay có trường rủi ro gì đó phải cần đến nhà xuất bản giải quyết. Trung bình trường liên lạc là 3, thông thường nhà xuất bản cần cho người sử dụng sách muốn biết thêm về nhà xuất bản này thì chỉ cần số điện thoại, gmail, website là đủ có thể tương tác qua lại. Tối đa trường liên lạc ở đây 5, gồm 3 phần tử số điện thoại, gmail và website, thì 2 phần tử còn lại sẽ có thể là đường link và phần tử dự phòng, đường link mà em nói đến là mạng xã hội, thì mạng xã hội ở đây có thể link với nhiều nền tảng mạng xã hội khác, nhưng chúng ta chỉ cần 1 đường link thôi thì các link nền tảng mạng xã hội khác cũng sẽ link theo. Ví dụ như là đường link facebook thì trong đó có các nền tảng khác như (fahasa, instagram,...)

1. **Mối quan hệ:**

* Một nhà xuất bản sẽ xuất bản nhiều cuốn sách.
* Một quyển sách chỉ thuộc về duy nhất 1 nhà xuất bản.

1. **Pattern + 5. Giải thích:**

* Ở đây em dùng Extends Reference Pattern cho bảng Books, cách này giúp người truy vấn về toàn bộ thông tin cuốn sách và tên nhà xuất bản. Bằng cách này, chúng ta tránh được việc truy vấn toàn bộ thông tin của nhà xuất bản một cách không cần thiết.
* Ở bảng nhà xuất bản (Publisher) có 1 object tên là liên hệ (Contact) em sẽ sử dụng Attribute Pattern, để lưu các thông tin liên lạc gồm key là loại liên lạc và value là thông tin của loại liên lạc đó

**Bài 2: Blog Website**

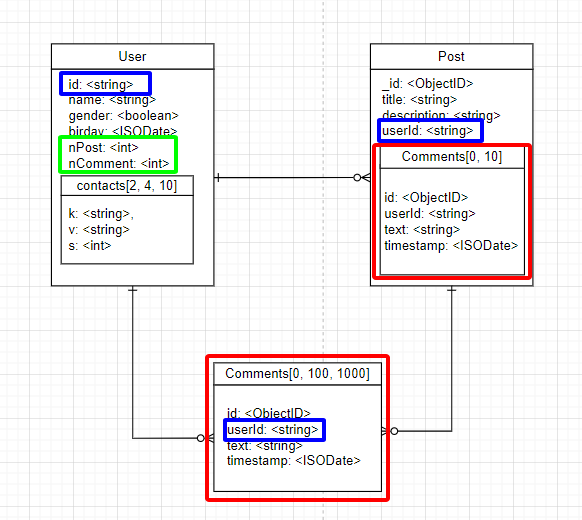
Chúng tôi muốn thiết kế CSDL MongoDB để quản lý blog.

* Mỗi người dùng (user) đã đăng ký blog được xác định bằng một tên người dùng (username). Chúng tôi cũng muốn ghi nhận lại phương thức liên hệ của người dùng, chẳng hạn như email, điện thoại, tài khoản Twitter, v.v. Các phương thức liên hệ được ghi nhận lại có thể thay đổi theo thời gian. Một phương thức liên hệ có thể có nhiều hơn một thông tin liên hệ. Mỗi phương thức liên hệ có thể có một trạng thái, ví dụ: "hoạt động", "không hoạt động", "đã xác nhận", v.v.
* Các bài đăng trên blog được đặc trưng bởi tiêu đề (title), mô tả (description) và tác giả (tức là username). Các bài viết nhận được bình luận (comment) từ các người dùng. Mỗi bình luận được mô tả bởi nội dung (text), ai đã bình luận (user) và bình luận lúc nào (timestamp)?
* Ứng dụng sẽ chỉ hiển thị cho mỗi bài đăng 10 bình luận gần đây nhất. Hồ sơ người dùng cũng phải bao gồm tổng số bài đăng và tổng số bài mà người dùng đã bình luận.

Yêu cầu: Thiết kế lược đồ lưu trữ dữ liệu để đáp ứng các yêu câu trên (chỉ ra các collection trong CSDL, cấu trúc Document, mối quan hệ, các chiến lược (design pattern) được sử dụng để thiết kế và nêu lý do tại sao dùng chiến lược đó).

**Bài làm**

1. **Các Collection.**



* User sử dụng computed pattern: để lưu tổng số bình luận và tổng số bài đăng, số lần bình luận và đăng bài sẽ được lưu vào collection riêng, và sẽ được cập nhật vào User sau mỗi giờ, sử dụng pattern này sẽ lấy được số bình luận và bài đăng mà không cần tham chiếu tới những collection khác.
* Contacts sử dụng attributed pattern: trường k đại diện cho phương thức liên hệ (ví dụ: facebook, instagram, số điện thoại, email), trường v đại diện cho giá trị tương ứng (đường link dẫn đến các trang mạng xã hội của chủ sở hữu, tài khoản gmail, số điện thoại),, trường key đại diện cho trạng thái (hoạt động, không hoạt động, đã xác nhận), do các phương thức liên hệ có thể thay đổi theo thời gian nên cần sử dụng attributed pattern.
* Comments sử dụng subset pattern: vì mỗi bài đăng cần hiển thị 10 bình luận gần nhất, nên chúng ta nhúng 10 bình luận mới nhất vào bài đăng để việc truy vấn và load dữ liệu nhanh chóng, và lấy hết tất cả các trường của của collection Comments bởi vì không trường nào là không cần thiết cả, vì thế chúng ta sử dụng subset pattern.

**Bài 3: Breadcrumbs**

Chúng tôi muốn thiết kế CSDL để quản lý sơ đồ trang web.

* Mỗi trang (page) của webisite được đặc trưng bởi đường dẫn tương đối (path), tiêu đề (title), nội dung (body), thời gian tạo (creation timestamp), thời gian cập nhật cuối cùng (last update timestamp) và danh sách các thẻ tag (list of tags).
* Các trang được sắp xếp theo hình cây. Trang chủ (home page) là gốc. Tất cả các trang khác nằm trong các đường dẫn phụ từ gốc.
* Ứng dụng phải xây dựng mã breadcrumb cho mỗi trang. Breadcrumb là một tập hợp các liên kết giúp cho người dùng xác định được vị trí hiện tại của mình trên cấu trúc website, là một danh sách có thứ tự các liên kết chứa tất cả các tổ tiên (ancestors) của trang hiện tại quay trở lại trang chủ, tức là thư mục gốc.

Yêu cầu: Thiết kế lược đồ lưu trữ dữ liệu để đáp ứng các yêu câu trên (chỉ ra các collection trong CSDL, cấu trúc Document, mối quan hệ, các chiến lược (design pattern) được sử dụng để thiết kế và nêu lý do tại sao dùng chiến lược đó).

**Bài làm**

1. **Đưa ra đủ và đúng các collection cần thiết (2đ)**

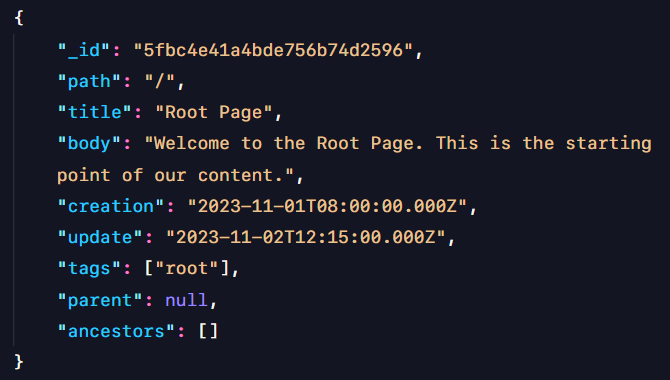
* Hệ thống này chỉ cần lưu trữ 1 tại collection Pages là đủ để thực hiện quản lý được sơ đồ website

1. **Đưa ra cấu trúc document: đủ các trường, đúng kiểu dữ liệu (2đ)**

**A screenshot of a computer

Description automatically generated**

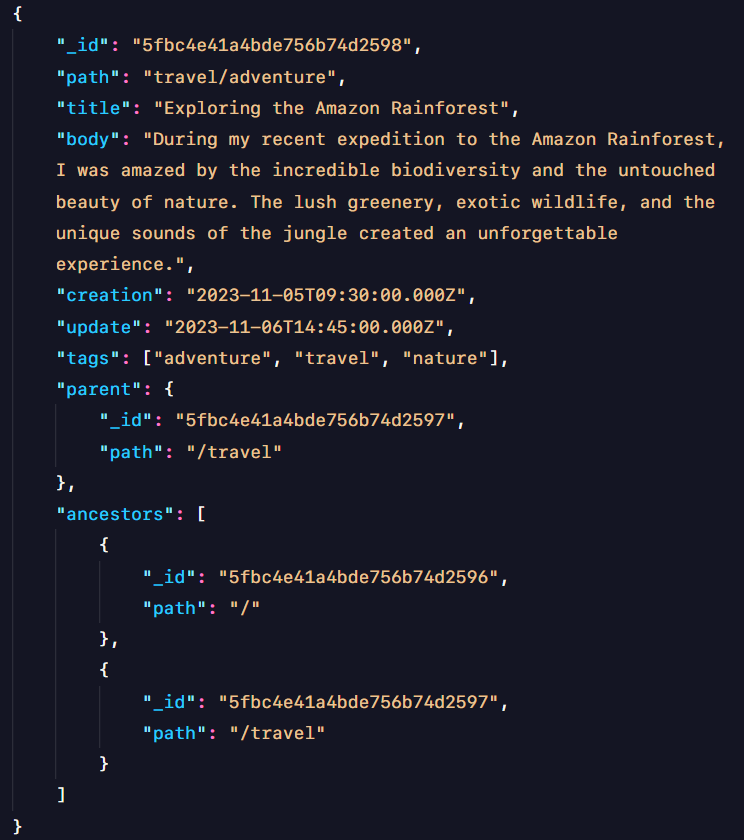
1. **Mối quan hệ: dùng embedding (2đ)**



**Page Root**



**Page Travel**



**Page Adventure**

1. **Áp dụng pattern: đưa ra được cấu trúc lưu trữ có sử dụng pattern (2đ)**

**A screenshot of a computer

Description automatically generated**

1. **Nêu lý do áp dụng pattern: Giải thích lợi ích khi áp dụng pattern (2đ)**

* Sử dụng Tree Pattern bằng cách thực hiện việc nhúng \_id và path của các document tổ tiên vào trong mảng ancestors của document con. Phần tử đầu tiên của mảng này sẽ là phần tử gốc, tiếp theo là các phần tử tổ tiên ở các cấp sau, và phần tử cuối cùng sẽ là trang cha của trang đó. Nếu mảng này rỗng, có nghĩa là trang đó là trang root.
* Để hỗ trợ việc truy vấn nhanh hơn, chúng ta sẽ thiết kế thuộc tính parent để lưu trữ id và path của trang cha.
* Khi ta thực hiện lưu trữ bằng cách này giúp chúng ta dễ dàng truy cập cha của trang hiện tại và tìm kiếm các phần tử con của một trang cụ thể mà không cần phải lặp qua toàn bộ cấu trúc cây.
* Khi sử dụng pattern này góp phần giúp truy vấn dữ liệu nhanh hơn dễ dàng hơn giúp việc quản lý thuận tiện nhất.

**Bài 4 – chuyển phát bưu kiện (Parcel delivery)**

Thiết kế cơ sở dữ liệu MongoDB để quản lý việc chuyển phát bưu kiện theo các yêu cầu sau.

* Khách hàng (customers) của dịch vụ chuyển phát bưu kiện là công dân được xác định bằng số an sinh xã hội (social security number) của họ. Họ có thể là người gửi hoặc người nhận bưu kiện đã chuyển. Họ có đặc trưng bởi tên (name), họ (surname), địa chỉ thư điện tử (email), số điện thoại (telephone number) và các địa chỉ (address) khác nhau, mỗi địa thuộc một loại địa chỉ khác nhau như một địa chỉ thanh toán (billing address), địa chỉ nhà (home address), địa chỉ công sở (work address), …. Mỗi địa chỉ bao gồm tên đường (street name, số nhà (street number), mã bưu chính bưu (postal code), thành phố (city), tỉnh (province), và quốc gia (country).
* Bưu kiện được đặc trưng bởi một mã vạch (barcode) duy nhất và kích thước vật lý (physical dimensions) của chúng như chiều rộng (width), chiều cao (height), độ sâu (depth) và trọng lượng (weight). Chiều rộng, chiều cao và chiều sâu luôn được tính bằng mét. Trọng lượng là luôn luôn được tính bằng kilôgam.
* Thông tin người nhận và người gửi được yêu cầu để chuyển từng bưu kiện phải luôn có sẵn khi truy cập dữ liệu của một bưu kiện. Thông tin người nhận và người gửi cần thiết để chuyển một bưu kiện bao gồm họ, tên đường, số nhà, mã bưu chính, thành phố, tỉnh, quốc gia.
* Kho bưu kiện được chia thành nhiều khu vực (area) khác nhau. Mỗi khu vực được xác định bởi một mã, ví dụ: 'area\_51' và bao gồm các dòng (line) khác nhau. Mỗi dòng được xác định bởi duy nhất mã, ví dụ: 'line\_12' và tạo từ một giá đỡ (rack). Mỗi giá đỡ được xác định bằng mã duy nhất, ví dụ: 'rack\_33' và được tạo thành từ các kệ (shelf). Mỗi bưu kiện được đặt trên một kệ cụ thể của kho hàng, được xác định bằng một mã duy nhất, ví dụ: 'shelf\_99'. Cơ sở dữ liệu được yêu cầu để theo dõi vị trí của từng bưu kiện trong kho.
* Với một bưu kiện cho trước, cơ sở dữ liệu phải được thiết kế để cung cấp một cách hiệu quả đầy đủ vị trí của nó, từ kệ, đến khu vực, thông qua giá để và dòng.
* Với một khách hàng, cơ sở dữ liệu phải được thiết kế để cung cấp hiệu quả tất cả bưu kiện với tư cách là người gửi và tất cả các bưu kiện của người đó với tư cách là người nhận.

Yêu cầu: Thiết kế lược đồ lưu trữ dữ liệu để đáp ứng các yêu câu trên (chỉ ra các collection trong CSDL, cấu trúc Document, mối quan hệ, các chiến lược (design pattern) được sử dụng để thiết kế và nêu lý do tại sao dùng chiến lược đó).

**Bài làm**

1. **Đưa ra đủ và đúng các collection cần thiết (2đ)**

* Các collection cần thiết để quản lý việc chuyển phát bưu kiện là: Customers và Parcel

1. **Đưa ra cấu trúc document: đủ các trường, đúng kiểu dữ liệu (2đ)**

A close-up of a document

Description automatically generated

1. **Mối quan hệ: dùng embedding hoặc reference (trường tham chiếu); bản số (cũng thể hiện được mối quan hệ) (2đ)**

* Mối quan hệ dùng reference

A screenshot of a computer program

Description automatically generated

A diagram of a computer code

Description automatically generated with medium confidence

1. **Áp dụng pattern: đưa ra được cấu trúc lưu trữ có sử dụng pattern (2đ)**
2. **Nêu lý do áp dụng pattern: Giải thích lợi ích khi áp dụng pattern (2đ)**

* Một khách hàng trong hệ thống có thể có N đơn hàng, tạo nên mối quan hệ 1-N. Từ vị trí khách hàng, nếu suy nghĩ ngược lại ở vị trí hệ thống thì hệ thống có quan hệ N-1 với khách hàng. Do đó việc nhúng tất cả các thông tin khách hàng cho mỗi bưu kiện chỉ để giảm thao tác lookup khi truy vấn dẫn đến nhiều thông tin trùng lặp hơn nữa không phải tất cả các thông tin của khách hàng đều có thể cần thiết cho một bưu kiện. Do đó sử dụng Extended Reference Pattern là một các xử lý tối ưu cho tình huống này. Thay vì sao chép toàn bộ thông tin của khách hàng, chúng ta chỉ sao chép nhưng trường mà chúng ta thường xuyên truy cập ở document Parcel chẳng hạn như địa chỉ nhận hàng, tên khách hàng, số điện thoại.
* Sử dụng tree pattern cho việc quản lí phân cấp khu vực để hàng sẽ tránh việc phải tạo lưu trữ những thông tin đó sang một collection khác làm khó khăn cho việc truy xuất dữ liệu sau này.

**Bài 5: Museum exhibitions**

* Chúng tôi muốn thiết kế cơ sở dữ liệu để quản lý triển lãm bảo tàng theo các yêu cầu. Các bảo tàng (museums) được đặc trưng bởi tên (name), địa chỉ (address), số điện thoại (phone) và trang web (website) nếu có. Địa chỉ bao gồm tọa độ địa lý (coordinates), tên đường (street), số nhà (number), mã bưu điện (postal code) và thành phố (city).
* Các vật phẩm (items) được trưng bày trong bảo tàng được xác định bằng một số lũy tiến và được đặc trưng bởi một tiêu đề (title), mô tả (description) và danh sách tên tác giả (authors). Các vật phẩm được phân loại là phát hiện khảo cổ học, hoặc tranh, hoặc tác phẩm điêu khắc.
* Cơ sở dữ liệu phải ghi lại tất cả các đặc tính chính của vật phẩm, chẳng hạn như kích thước của nó (tùy vào loại vật phẩm, ví dụ chiều rộng, chiều cao, trọng lượng, v.v.). Mỗi đối tượng địa lý có ít nhất một tên và một giá trị, và có thể là một đơn vị đo lường. Ví dụ, vật liệu chính là đặc điểm của một phát hiện khảo cổ học, kích thước hình học là đặc điểm của một bức tranh. Đối với mỗi vật phẩm, bảo tàng mà nó thuộc về phải được ghi lại, với tên bảo tàng thường xuyên được truy cập cùng với đặt điểm chính đó.
* Một số cuộc triển lãm được tổ chức trong mỗi bảo tàng. Triển lãm được đặc trưng bởi một tiêu đề (title), một mô tả (description), danh sách tên người phụ trách (curators). Bạn phải ghi lại tất cả các vật phẩm liên quan đến mỗi cuộc triển lãm; chúng có thể lên tới hàng trăm vật phẩm. Một vật phẩm có thể là một phần của các cuộc triển lãm khác nhau. Hơn nữa, mỗi cuộc triển lãm có thể được tổ chức bởi một số bảo tàng trong các thời kỳ khác nhau. Bạn phải ghi lại ngày bắt đầu và ngày kết thúc của mỗi triển lãm ở mỗi bảo tàng.
* Với một vật phẩm, cơ sở dữ liệu phải được thiết kế để cung cấp một cách hiệu quả tên của bảo tàng sở hữu nó.
* Với một cuộc triển lãm, cơ sở dữ liệu phải được thiết kế để cung cấp một cách hiệu quả tên của bảo tàng và tọa độ địa lý nơi nó đã được tổ chức.
* Hơn nữa, với một cuộc triển lãm, danh sách các vật phẩm có trong cuộc triển lãm và số lượng của chúng phải được đưa ra một cách hiệu quả.

Yêu cầu: Thiết kế lược đồ lưu trữ dữ liệu để đáp ứng các yêu câu trên (chỉ ra các collection trong CSDL, cấu trúc Document, mối quan hệ, các chiến lược (design pattern) được sử dụng để thiết kế và nêu lý do tại sao dùng chiến lược đó).

**Bài làm**

A diagram of a database

Description automatically generated

* Museums có tên (name), địa chỉ (address), số điện thoại (phone) và trang web (website) nếu có. Địa chỉ bao gồm tọa độ địa lý (coordinates), tên đường (street), số nhà (number), mã bưu điện (postal code) và thành phố (city).
* Items được xác định bằng một số lũy tiến và một tiêu đề (title), mô tả (description) và danh sách tên tác giả (authors). Các vật phẩm được phân loại là phát hiện khảo cổ học, hoặc tranh, hoặc tác phẩm điêu khắc. Ghi lại tất cả các đặc tính chính của vật phẩm, chẳng hạn như kích thước của nó (tùy vào loại vật phẩm, ví dụ chiều rộng, chiều cao, trọng lượng, v.v.). [Attibute Pattern: do mỗi vật phẩm sẽ có một tính chất và sẽ có một vài thuộc tính tương tự nhau như kích thước hay chất liệu. Outlier Patternn: vì mỗi vật phẩm là khác nhau nên có thể có những tính chất đặc biệt không giống các loại khác]. Với một vật phẩm, cơ sở dữ liệu phải được thiết kế để cung cấp một cách hiệu quả tên của bảo tàng sở hữu nó. [Extended Reference Pattern: Với một vật phẩm khi được truy xuất thì có thể người truy không cần muốn biết hết trường trong museum mà chỉ cần biết một phần thông tin về museum vật phẩm đang được trưng bày]
* Triển lãm được đặc trưng bởi một tiêu đề (title), một mô tả (description), danh sách tên người phụ trách (curators), phải ghi lại ngày bắt đầu và ngày kết thúc của mỗi triển lãm ở mỗi bảo tàng. [Bucket Pattern: vì một triển lãm có thể tổ chức ở nhiều museum nên cần lưu gom nhóm lại trong một document, bởi vì đây là dữ liệu cập nhật theo thời gian (có thể là ngày hoặc tuần)]. Với một cuộc triển lãm cung cấp tên của bảo tàng và tọa độ địa lý. [Extended Pattern: Extended Reference Pattern: Với một triển lãm khi được truy xuất thì có thể người truy không cần muốn biết hết trường trong museum mà chỉ cần biết một phần thông tin về museum là triển lãm này được diễn ra ở các bảo tàng nào].

**BÀI TẬP CHƯƠNG 5**

**Nội dung 1:** Để tăng hiệu năng cho hệ thống, hãy optimize từng câu truy vấn sau và đề xuất câu tạo chỉ mục tương ứng

**Tạo index dựa theo quy tắc esr: (equals -> sort -> range)**

1. db.users.find({numoflike: {$gte:50000}, gender: 'female', age: 29}).sort(name:1);

**optimize:** db.users.find({age: 29, gender: ‘female’, numoflike: {$gte:50000}).sort(name:1)

**index:** db.users.createIndex({age: 1, name: 1, numoflike: 1,name:1})

1. db.users.find({'Addresses.Country': 'Norfolk'}).skip(10).limit(20).pretty()

**index:** createIndex({Addresses.Country: 1})

1. db.products.find({price: { $gt: 1, $lt: 5 }, in\_stock: true }).sort({ name: 1 })

**optimize:** ({in\_stock: true, price: { $gt: 1, $lt: 5 }}).sort

**index:** ({in\_stock: 1, name: 1, price: 1})

1. db.user.find( { numoflike : { $gte : NumberDecimal(100000.00) }, city: "TP HCM" } ) .sort( { lastName: 1, firstName: 1 } )

**optimize:** ({city: "TP HCM" , numoflike : { $gte : NumberDecimal(100000.00) }})….

**index:** createIndex({city:1, lastName:1, firstName:1, numoflike: 1})

1. db.products.find({'details.manufacturer': 'Acme','pricing.sale': { $lt: 7500}})

**index:** createIndex({'details.manufacturer':1, 'pricing.sale': 1})

1. db.myColl.find( { score: 5, price: { $gt: NumberDecimal( "10" ) } } ).sort( { price: 1 } )

**index:** createIndex({score:1, price:1})

**Nội dung 2:**

Công ty bào hiểm ABC, có bán nhiều sản phẩm bảo hiểm khác nhau cho khách hàng ở toàn quốc (Việt Nam). Do nhiều lý do khác nhau, dữ liệu hệ thống cần được lưu trữ phân tán trên một hệ thống sharding gồm 3 shard node. Theo kết quả phân tích workload, cho thấy người dùng hệ thống thường xuyên truy vấn các hồ sơ bảo hiểm của khách hàng thành phố mà khách hàng trực thuộc.

Bạn hãy đề xuất một kế hoạch cho việc dựng một sharding sao cho tăng được hiệu năng của hệ thống trong khi truy vấn dữ liệu.

**Giải:**

* Xây dựng shard key là trường city vì đây là trường thường xuyên được truy vấn, việc này sẽ giúp truy vấn nhanh hơn
* Phân chia dữ liệu dựa trên city vào 3 shard node. Mỗi node chứa dữ liệu của các thành phố gần nhau đế tối ưu latency khi truy vấn

VD: Node1: Hồ Chí Minh, Bình Dương, Long An

Node2: Hà Nội, Phú Thọ, Vĩnh Phúc

Node3: Đà Nẵng, Quảng Nam, Thừa Thiên Huế

* Sử dụng TTL index trên trường expiration date để xóa các doc bảo hiểm hết hạn và để dữ liệu không bị quá tải. Index này xóa doc hết hạn mà không cần scan toàn bộ dữ liệu.
* Xây dựng hệ thống replicaset ở mỗi shard node để đảm bảo tính toàn vẹn và sẵn sàng của dữ liệu khách hàng.

**Nội dung 3: một số câu hỏi lý thuyết gợi ý**

**Những yếu tố nào ảnh hưởng đến hiệu năng của hệ thống, cho ví dụ và giải thích lý do tại sao**

Yếu tố ảnh hưởng đến hiệu suất của hệ thống MongoDB

MongoDB là một hệ thống cơ sở dữ liệu NoSQL dựa trên tài liệu. Nó sử dụng bộ nhớ để lưu trữ dữ liệu và các index. Hiệu suất của MongoDB có thể bị ảnh hưởng bởi nhiều yếu tố, bao gồm:

**Kích thước bộ nhớ (Memory Size)**

* Kích thước bộ nhớ là yếu tố quan trọng nhất ảnh hưởng đến hiệu suất của MongoDB. MongoDB sử dụng bộ nhớ để lưu trữ dữ liệu và các index. Nếu bộ nhớ không đủ lớn, hệ thống sẽ phải thực hiện truy cập đĩa, làm giảm hiệu suất.
* Ví dụ: Trong trường hợp cơ sở dữ liệu lưu trữ thông tin về hóa đơn và chi tiết sản phẩm, nếu bộ nhớ không đủ lớn để chứa index và dữ liệu gần đây, hệ thống sẽ phải thực hiện truy cập đĩa thường xuyên khi người dùng truy vấn các hóa đơn mới. Điều này dẫn đến trễ do thời gian truy cập đĩa lớn, làm giảm hiệu suất do chi phí I/O tăng. Việc duy trì bộ nhớ đủ lớn để chứa working set là quan trọng để tối ưu hóa hiệu suất.

**Thiết kế schema (Schema)**

* Thiết kế schema ảnh hưởng đến cách dữ liệu được lưu trữ và truy cập. Schema phức tạp có thể làm tăng chi phí của truy vấn và làm giảm hiệu suất.
* Ví dụ: Trong ứng dụng thương mại điện tử, nếu thiết kế schema quá phức tạp, chẳng hạn sử dụng nhiều bảng liên quan khi người dùng xem sản phẩm, việc truy vấn và ghép thông tin này có thể làm giảm hiệu suất. Ngược lại, sử dụng embedded documents giúp giảm số lượng truy vấn và tăng tốc độ truy cập dữ liệu bằng cách giảm việc thực hiện các truy vấn join.

**Chỉ mục (Indexs)**

* Chỉ mục giúp tăng tốc độ truy cập dữ liệu bằng cách giảm số lượng bản ghi cần quét. Tuy nhiên, quá nhiều chỉ mục hoặc sử dụng không đúng cũng có thể làm giảm hiệu suất khi thực hiện các thao tác cập nhật.
* Ví dụ: Trong ứng dụng chia sẻ ảnh, việc tạo chỉ mục trên trường tên người đăng khi người dùng thường xuyên tìm kiếm ảnh theo tên có thể giúp tăng tốc độ truy vấn. Tuy nhiên, quá nhiều chỉ mục không cần thiết có thể làm tăng chi phí lưu trữ và giảm hiệu suất cập nhật do mỗi chỉ mục mới phải được cập nhật khi có sự thay đổi.

**Hiệu suất trên cụm (Cluster)**

* MongoDB có thể được triển khai trên một cụm các máy chủ. Hiệu suất của cụm MongoDB có thể bị ảnh hưởng bởi các yếu tố như:
* Số lượng node trong cụm
* Kích thước của các node trong cụm
* Khả năng kết nối giữa các node trong cụm
* Ví dụ: Trong môi trường cụm MongoDB, nếu một node chứa dữ liệu phổ biến gặp sự cố, các node khác cần đảm bảo rằng dữ liệu có thể được đồng bộ hóa một cách hiệu quả. Việc sử dụng các chiến lược như replica sets và sharding có thể giúp đảm bảo khả năng chịu lỗi và hiệu suất toàn bộ cụm trong trường hợp sự cố.

**Tối ưu hóa CRUD Operations**

* Tối ưu hóa CRUD Operations giúp giảm chi phí của các thao tác I/O và tăng tốc độ thực hiện các thao tác.
* Ví dụ: Trong một ứng dụng quản lý học viên, khi cần cập nhật điểm số của nhiều học viên sau một kỳ thi, thay vì thực hiện các truy vấn cập nhật cho từng học viên, có thể sử dụng một câu truy vấn cập nhật (update query) để áp dụng thay đổi cho tất cả học viên cùng một lúc. Điều này giúp giảm số lượng truy vấn và tối ưu hóa hiệu suất trong quá trình cập nhật điểm số.

**Để nâng cao hiệu năng của hệ thống, một trong những giải pháp là sử dụng RAM hiệu quả. Bạn đề xuất các giải pháp có thể có trong khi thiết kế cấu trúc lưu trữ dữ liệu mà giúp ta dùng RAM hiệu quả, giải thích tại sao?**

Các giải pháp có thể có trong khi thiết kế cấu trúc lưu trữ dữ liệu giúp sử dụng RAM hiệu quả:

**Sử dụng indexes một cách hiệu quả để tăng tốc độ truy cập dữ liệu.**

* Indexes là cấu trúc dữ liệu giúp tăng tốc độ truy vấn bằng cách cung cấp một cách nhanh chóng để tìm dữ liệu dựa trên một hoặc nhiều trường nhất định. Khi indexes được tạo, MongoDB có thể giữ một phần hoặc toàn bộ chỉ mục trong bộ nhớ. Điều này giúp giảm thời gian truy vấn, vì dữ liệu có thể được truy cập nhanh chóng thông qua chỉ mục trong bộ nhớ thay vì phải đọc từ đĩa.
* Để sử dụng indexes hiệu quả, cần lưu ý các nguyên tắc sau:
* Chỉ tạo indexes cho các trường được sử dụng thường xuyên trong các truy vấn.
* Sử dụng các loại indexes phù hợp với kiểu dữ liệu và cách sử dụng của dữ liệu.
* Tối ưu hóa kích thước và cấu trúc của indexes để giảm thiểu không gian sử dụng.

**Thiết kế, cấu hình Working Set phù hợp với kích thước của bộ nhớ ram**

* Working set là tập hợp dữ liệu mà hệ thống thường xuyên truy cập. Nếu thiết kế working set nhỏ hơn lượng dữ liệu cần xử lý sẽ làm thời gian xử lý sẽ bị ảnh hưởng ngược lại nếu working set lớn hơn kích thước bộ nhớ Ram sẽ tác động rất lớn đến hiệu suất của hệ thống, gây tràn RAM có thể dẫn đến treo hệ thống

**Thiết kế cấu trúc dữ liệu sao cho chúng tận dụng tốt bộ nhớ.**

* Sử dụng các cấu trúc dữ liệu phù hợp có thể giúp giảm lượng bộ nhớ được sử dụng và tăng tốc độ truy cập. Ví dụ, sử dụng các arrays để lưu trữ các giá trị giống nhau có thể giúp giảm số lượng không gian bị lãng phí.
* Một số nguyên tắc thiết kế cấu trúc dữ liệu hiệu quả như sau:
* Sử dụng các cấu trúc dữ liệu phù hợp với kiểu dữ liệu và cách sử dụng của dữ liệu.
* Tránh sử dụng các cấu trúc dữ liệu phức tạp khi không cần thiết.
* Sử dụng các kỹ thuật tối ưu hóa bộ nhớ như sử dụng các loại dữ liệu có kích thước nhỏ hơn.

**Sử dụng các cấu trúc dữ liệu có kích thước cố định.**

* Các cấu trúc dữ liệu có kích thước cố định có thể giúp giảm lượng bộ nhớ bị lãng phí. Ví dụ, sử dụng các số nguyên hoặc số float thay vì các strings để lưu trữ các giá trị có kích thước cố định.

**Sử dụng các cấu trúc dữ liệu có thể được phân mảnh.**

* Các cấu trúc dữ liệu có thể được phân mảnh có thể giúp giảm lượng bộ nhớ được sử dụng. Ví dụ, sử dụng biện pháp Sharding và chọn sharding key phù hợp kết hợp với Replicaset để tối ưu việc đọc ghi dữ liệu trong hệ thống và cũng đảm bảo tính toàn vẹn, sẵn sàng của dữ liệu

**Hệ thống có sử dụng replication, hãy cho ví dụ 1 trường hợp là chúng ta nên đọc (Read) dữ liệu từ secondary tốt hơn là từ Primary, giải thích lý do**

**Trường hợp khi node primary đang bị quá tải**

* Trong hệ thống có sử dụng replication, primary node là node chịu trách nhiệm ghi dữ liệu mới vào cơ sở dữ liệu. Các node secondary sẽ được đồng bộ dữ liệu từ primary node theo định kỳ.
* Khi primary node bị quá tải, nó sẽ không thể đáp ứng kịp các yêu cầu đọc và ghi dữ liệu mới. Điều này có thể dẫn đến tình trạng chậm trễ hoặc thậm chí là gây ra lỗi hệ thống. Để giải quyết vấn đề này, chúng ta có thể đọc dữ liệu từ secondary node thay vì primary vì node secondary thường có tốc độ đọc nhanh hơn primary node, vì nó không cần xử lý các yêu cầu ghi dữ liệu mới.
* Ví dụ, trong hệ thống thu thập dữ liệu thời tiết khi lượng dữ liệu mà các sensor gửi về rất lớn cùng lúc nếu khi đó các nhân viên DevOps cũng truy vấn đọc dữ liệu trên primay sẽ làm cho primary node có thể bị quá tải. Trong trường hợp này, chúng ta có thể đọc dữ liệu từ secondary node, thay vì đọc từ primary node. Điều này sẽ giúp giảm tải cho primary node, giúp cải thiện hiệu suất tổng thể của hệ thống.
* Cụ thể, việc đọc dữ liệu từ secondary node sẽ giúp cải thiện hiệu suất hệ thống theo các cách sau:
* Giảm thời gian phản hồi cho các yêu cầu đọc dữ liệu.
* Tăng khả năng chịu tải của hệ thống.
* Giảm thiểu nguy cơ lỗi hệ thống.
* Tuy nhiên, cần lưu ý rằng việc đọc dữ liệu từ secondary node có thể dẫn đến dữ liệu không nhất quán, vì dữ liệu trên secondary node có thể chưa được cập nhật mới nhất. Do đó, chỉ nên đọc dữ liệu từ secondary node trong trường hợp dữ liệu không cần thiết phải được cập nhật ngay lập tức hoặc có thể chấp nhận dữ liệu có độ trễ.

**Những yếu tố nào ảnh hưởng đến tốc độ của câu truy vấn (Find) dữ liệu trong mongoDB, cho ví vụ và nêu lý do tại sao?**

**Những yếu tố ảnh hưởng đến tốc độ của câu truy vấn (Find) dữ liệu trong MongoDB:**

* **Kiểu truy vấn:** Các truy vấn đơn giản, chỉ truy vấn một trường hoặc một số trường nhất định sẽ có tốc độ nhanh hơn các truy vấn phức tạp, truy vấn nhiều trường hoặc sử dụng các toán tử logic phức tạp.

Ví dụ:

**db.collection.find({"name": "John Doe"})**

Truy vấn này sẽ truy vấn tất cả các documents trong collection có trường name có giá trị là John Doe. Truy vấn này sẽ có tốc độ nhanh hơn truy vấn sau:

**db.collection.find({"name": "John Doe", "age": 30})**

Truy vấn này sẽ truy vấn tất cả các documents trong collection có trường name có giá trị là John Doe và trường age có giá trị là 30. Truy vấn này sẽ có tốc độ chậm hơn vì nó cần phải kiểm tra thêm một trường.

* **Kích thước của collection:** Các collection nhỏ sẽ có tốc độ truy vấn nhanh hơn các collection lớn.

Ví dụ:

Một collection có 100 documents sẽ có tốc độ truy vấn nhanh hơn một collection có 1 triệu documents.

* **Kích thước của document:** Các document nhỏ sẽ có tốc độ truy vấn nhanh hơn các document lớn.

Ví dụ:

Một document có 10 trường sẽ có tốc độ truy vấn nhanh hơn một document có 100 trường.

* **Thiết kế indexes:** Các indexes sẽ giúp tăng tốc độ truy vấn bằng cách cung cấp một cách nhanh chóng để tìm dữ liệu dựa trên một hoặc nhiều trường nhất định.

Ví dụ:

Nếu chúng ta tạo một index cho trường name trong một collection, thì khi truy vấn lên trường name sẽ cho tốc độ nhanh hơn nhưng ngược lại nếu có nhiều index không cần thiết sẽ làm ảnh hưởng đến tốc độ

* **Phân mảnh:** Phân mảnh sẽ giúp tăng tốc độ truy vấn bằng cách phân chia dữ liệu trong một collection thành nhiều shards. Ví dụ nếu như khi có dữ liệu khách hàng theo 3 miền Bắc, Trung và Nam thì khi sử dụng Sharding thì khi truy vấn đến khách hàng theo từng khu vực sẽ nhanh hơn giảm tải cho hệ thống

**Đứng dước gốc độ người optimize hiệu năng của hệ thống hãy cho biết ưu điểm và nhược điểm lưu trữ dữ liệu phân tán trên hệ thống sharding.**  
**Ưu điểm của lưu trữ dữ liệu phân tán trên hệ thống sharding:**

* Tăng khả năng chịu tải: Phân mảnh dữ liệu giúp giảm tải cho mỗi shard, giúp hệ thống có thể xử lý nhiều truy vấn cùng lúc mà không bị chậm trễ. Điều này đặc biệt quan trọng đối với các hệ thống có lượng truy cập cao, như các hệ thống thương mại điện tử, mạng xã hội, hoặc các hệ thống xử lý giao dịch trực tuyến.
* Tăng tốc độ truy vấn: Dữ liệu được lưu trữ gần người dùng trên các shard, giúp giảm thời gian truy cập dữ liệu và cải thiện hiệu suất truy vấn. Điều này đặc biệt quan trọng đối với các hệ thống cần phải truy cập dữ liệu thường xuyên, như các hệ thống phân tích dữ liệu, hoặc các hệ thống quản lý kho hàng.
* Tăng khả năng mở rộng: Hệ thống có thể mở rộng linh hoạt bằng cách thêm shard mới khi nhu cầu mở rộng tăng lên. Điều này giúp hệ thống đáp ứng nhu cầu của người dùng khi hệ thống phát triển.

**Nhược điểm của lưu trữ dữ liệu phân tán trên hệ thống sharding:**

* Tăng độ phức tạp: Quản lý shard và đảm bảo tính nhất quán của dữ liệu là một công việc phức tạp, đòi hỏi sự kiểm soát và quản lý kỹ thuật chặt chẽ. Điều này có thể gây khó khăn cho việc triển khai và vận hành hệ thống.
* Tăng chi phí: Thêm máy chủ mới để hỗ trợ các shard mới có thể tăng chi phí triển khai và duy trì hệ thống. Điều này cần được cân nhắc kỹ lưỡng trước khi triển khai hệ thống sharding.
* Đòi hỏi người thiết kế, quản lý và sử dụng hệ thống sharding có nhiều kinh nghiệm lựa chọn shard key hợp lý