MỤC LỤC

[MỤC LỤC 1](#_Toc8373063)

[**LỜI CẢM ƠN** 3](#_Toc8373064)

[**PHẦN I: MỞ ĐẦU** 4](#_Toc8373065)

[**PHẦN IV: KẾT QUẢ VÀ THẢO LUẬN** 8](#_Toc8373066)

[**4.1. Tổng quan về cơ sở dữ liệu NoSQL.** 8](#_Toc8373067)

[4.1.2. Khái niệm NoSQL. 8](#_Toc8373068)

[4.1.6. So sánh NoSQL với các loại cơ sở dữ liệu khác. 16](#_Toc8373069)

[4.1.7. Cách triển khai một ứng dụng NoSQL. 24](#_Toc8373070)

[**4.2. Phân loại cơ sở dữ liệu NoSQL.** 26](#_Toc8373071)

[4.2.1. Key-Value Databases 26](#_Toc8373072)

[4.2.2. Document databases 32](#_Toc8373073)

[4.2.3. Column family stores 43](#_Toc8373074)

[4.2.4. Graph Databases 55](#_Toc8373075)

[**4.3.** **Tìm hiểu về MongoDB.** 59](#_Toc8373076)

[**4.3.5.** **Ưu nhược điểm của MongoDB** 61](#_Toc8373077)

[**4.4.1.5.** **Yêu cầu** 73](#_Toc8373078)

[o ***SAEQ002: Chức năng đăng xuất*** 74](#_Toc8373079)

[o ***SAEQ005: Chức năng sửa loại danh mục sản phẩm*** 76](#_Toc8373080)

[o ***SAEQ006: Chức năng thêm mới sản phẩm*** 77](#_Toc8373081)

[o ***SAEQ008: Chức năng sửa thông tin sản phẩm*** 79](#_Toc8373082)

[o ***SAEQ011: Chức năng tìm kiếm sản phẩm (Front - end)*** 82](#_Toc8373083)

[**PHẦN V: KẾT LUẬN VÀ ĐỀ NGHỊ** 84](#_Toc8373084)

[**PHẦN VI: TÀI LIỆU THAM KHẢO** 85](#_Toc8373085)

LỜI CAM ĐOAN

Tôi cam đoan rằng kết quả nghiên cứu trong bài khóa luận này là trung thực và chưa được bảo vệ một học vị nào.

Tôi xin cam đoan rằng mọi sự giúp đỡ cho việc thực hiện khóa luận này đã được cảm ơn và các thông tin trích dẫn trong khóa luận đã đều được ghi rõ nguồn gốc. Đồng thời tôi xin cam đoan rằng trong quá trình thực hiện đề tài này tại địa điểm thực tập tôi luôn chấp hành đúng mọi quy định của địa phương nơi thực hiện đề tài.

*Hà Nội, ngày tháng năm 2019*

**Sinh viên**

**Vũ Thị Hoài Linh**

**LỜI CẢM ƠN**

Lời đầu tiên em xin gửi lời cảm ơn đến Ban Giám Đốc Học viện Nông Nghiệp Việt Nam đã tạo cơ hội cho em được học tập và nghiên cứu trong suốt quá trình học tập tại Học viện. Để thực hiện và hoàn thành khóa luận tốt nghiệp này, ngoài sự cố gắng của bản thân, em còn nhận được sự giúp đỡ tận tình của các thầy cô giáo, các đơn vị, gia đình, bạn bè về cả vật chất và tinh thần để em hoàn thành khóa luận này.

Qua đây em xin bày tỏ lòng biết ơn sâu sắc tới toàn thể các thầy cô giáo trong khoa Công Nghệ Thông Tin cùng các thầy cô giáo trong trường Học viện Nông nghiệp Việt Nam đã dìu dắt, dạy dỗ em trong quá trình học tập tại trường.

Em xin gửi lời cảm ơn chân thành nhất đến các thầy cô trong bộ môn Công Nghệ Phần Mềm, khoa Công Nghệ Thông Tin, Học Viện Nông Nghiệp Việt Nam cùng toàn thể các thầy cô giáo đã giảng dạy, truyền đạt cho em những kiến thức nhận xét góp ý cho em để em hoàn thành khóa luận này.

Em xin bày tỏ lòng kính trọng, lòng biết ơn chân thành và sâu sắc nhất tới cô giáo - Th.S Hoàng Thị Hà, người đã tận tình hướng dẫn, đóng góp ý kiến quý báu giúp đỡ em vượt qua những khó khăn trong quá trình nghiên cứu để hoàn chỉnh bài khóa luận này.

Em xin chân thành cảm ơn gia đình, bạn bè đã quan tâm động viên, giúp đỡ và tạo mọi điều kiện để em học tập, nghiên cứu.

Qua đề tài này em đã tiếp cận được nhiều kiến thức chuyên môn khá hấp dẫn và quan trọng. Khi thực hiện đề tài em đã cố gắng hết khả năng, nhưng do thời gian có hạn nên đề tài không tránh khỏi những sai sót. Vì vậy em rất mong nhận được sự góp ý của các quý thầy cô để đề tài của em ngày càng được hoàn thiện hơn và có thể áp dụng thành công vào trong thực tiễn. Cuối cùng em xin kính chúc quý thầy cô sức khỏe, công tác tốt và thành công trong cuộc sống!

Em xin chân thành cảm ơn!

*Hà Nội, ngày tháng năm 2019*

**Sinh viên thực hiện**

**Vũ Thị Hoài Linh**

# **PHẦN I: MỞ ĐẦU**

* 1. Tên Đề tài

Tìm hiểu NoSQL và xây dựng ứng dụng Demo.

1.2. Đặt vấn đề

Ngày nay trong thời đại công nghệ thông tin bùng nổ, thành công của Internet đã khiến cho số lượng người dùng truy cập vào cùng một hệ thống ngày càng tăng. Điển hình như trung bình mỗi một ngày có khoảng 1,45 tỷ người đăng nhập vào Facebook (Nguồn: Facebook 25/04/2018) thì ta mới hình dung được sự bùng nổ của thông tin là như thế nào. Để giải quyết vấn đề như trên thì chúng ta đã mở rộng các hệ thống máy chủ siêu lớn, phân chia thành nhiều cụm đặt khắp nơi trên thế giới. Nhưng với tốc độ phát triển như hiện nay thì việc tăng số lượng máy chủ vẫn chưa đủ. Ta cần xem xét và nâng cấp các giải pháp lưu trữ cho tương lai. Hệ thống máy chủ cơ sở dữ liệu đòi hỏi phải rất mạnh mẽ nếu không máy chủ sẽ bị quá tải. Với các hệ thống lớn với số lượng lên đến hàng triệu cho đến hàng tỷ thì việc hiệu năng tốt là việc bắt buộc.

Với yêu cầu cho việc lưu trữ ngày càng cao như: lưu trữ nhiều dữ liệu, tốc độ truy xuất nhanh, phân tán dữ liệu trên nhiều máy chủ… Hệ quản trị cơ sở dữ liệu phi quan hệ - RDBMS hiện nay thì vấn đề hiệu năng thường không tốt cho trường hợp này vì lý do quá chặt chẽ, yêu cầu nhất quán dữ liệu đã gây ra sự phức tạp làm giảm hiệu suất hoạt động, nhất là trong trường hợp phải chứa một lượng lớn dữ liệu.

Mọi vấn đề đều có giải pháp. Thật vậy, những năm gần đây đã nổi lên một xu hướng lưu trữ mới có cách thức lưu trữ không dùng mô hình cơ sở dữ liệu quan hệ như đa số các cơ sở dữ liệu hiện tại, mà sử dụng mô hình cơ sở dữ liệu phi quan hệ NoSQL. NoSQL sinh ra để khắc phục các vấn đề mà một cơ sở dữ liệu dạng RDBMS gặp phải. NoSQL sinh ra không phải để cạnh tranh với RDBMS mà là để đảm nhiệm những việc mà RDBMS chưa làm tốt.

Mục đích mà NoSQL hướng đến đó là hiệu suất hoạt động cao với số lượng dữ liệu rất lớn. Tuy nhiên để đạt được điều đó thì NoSQL đã bỏ qua thông dịch và các truy vấn trong SQL. Việc sử dụng các ràng buộc quan hệ cùng truy vấn SQL sẽ thân thiện và thích hợp với nhiều dạng dữ liệu. Tuy nhiên, nếu dữ liệu quá đơn giản, các thủ tục SQL sẽ không cần thiết. Đồng thời NoSQL cũng có cách thiết kế dữ liệu khác với cơ sở dữ liệu truyền thống như: tư tưởng thiết kế dữ liệu phi quan hệ, lưu trữ dữ liệu dạng document, sử dụng tối đa indexes… Trong các đặc tính đó, dữ liệu phi quan hệ là một yếu tố quan trọng góp phần làm nên thành công cho NoSQL. Dữ liệu phi quan hệ tức là không tuân theo các dạng chuẩn hóa mà cơ sở dữ liệu RDBMS đặt ra. Thay vào đó, khi thiết kế một cơ sở dữ kiệu NoSQL ta phải tuân theo một số quy tắc mới mà NoSQL đặt ra để đạt được hiệu suất hoạt động cao.

Như vậy, trong đề tài này sẽ tìm hiểu xem NoSQL đã giải quyết các vấn đề trên như thế nào và áp dụng kiến thức tìm hiểu đó vào việc xây dựng một ứng dụng sử dụng cơ sở dữ liệu dạng NoSQL.

1.3. Mục đích và yêu cầu

1.3.1. Mục đích

Với những vấn đề nêu trên, đề tài này cần đạt được các mục đích như sau:

* Hiểu NoSQL, kiến trúc NoSQL, phân loại và đặc điểm từng loại để có cái nhìn tổng quan về NoSQL đồng thời biết được cách mà NoSQL đã giải quyết được vấn đề hiệu suất cao với lượng dữ liệu lớn như thế nào.
* Hiểu trường hợp áp dụng cơ sở dữ liệu dạng NoSQL, trường hợp nào không phù hợp với NoSQL. Phân tích làm rõ ưu khuyết điểm của việc áp dụng cơ sở dữ liệu NoSQL. So sánh giữa việc sử dụng cơ sở dữ liệu SQLvà cơ sở dữ liệu NoSQL trên cùng một ứng dụng. So sánh hiệu suất giữa một cơ sở dữ liệu dạng NoSQL và cơ sở dữ liệu SQL để làm rõ hiệu suất hoạt động của NoSQL.
* Hiểu MongoDB, CouchDB.
* Vận dụng MongoDB; NoSQL xây dựng ứng dụng Demo Website bán hàng.

1.3.2. Yêu cầu

* Tìm hiểu lý thuyết NoSQL.
* Tìm hiểu MongoDB.
* Tìm hiểu bài toán Demo.
* Xây dựng Demo ứng dụng Web có sử dụng NoSQL, MongoDB.

PHẦN II: TỔNG QUAN TÌNH HÌNH NGHIÊN CỨU TRONG VÀ NGOÀI NƯỚC

2.1. Tình hình nghiên cứu trong nước

Hệ quản trị cơ sở dữ liệu phi quan hệ - NoSQL khá là phát triển ở Việt Nam. Có rất nhiều công ty, tổ chức đã sử dụng NoSQL. Đồng thời cũng có rất nhiều khóa học đã được mở để giúp các lập trình viên tiếp cận NoSQL được nhanh hơn.

2.2. Tình hình nghiên cứu ngoài nước

Hệ quản trị cơ sở dữ liệu phi quan hệ - NoSQL được rất nhiều quan tâm của cộng đồng lập trình viên quốc tế. Có rất nhiều khóa học được mở, tài liệu phong phú và rất đầy đủ. NoSQL được sử dụng ở rất nhiều công ty, tập đoàn lớn, ví dụ như FaceBook sử dụng Cassandra do FaceBook phát triển, Google phát triển và sử dụng BigTable...

PHẦN III: NỘI DUNG VÀ PHƯƠNG PHÁP NGHIÊN CỨU

3.1. Địa điểm và thời gian nghiên cứu (thực tập)

* Địa điểm: Học viện Nông Nghiệp Việt Nam
* Thời gian nghiên cứu 25/12/2018 đến ngày 10/05/2019
* Địa điểm: Công ty cổ phần phần mềm Nara.
* Thời gian nghiên cứu 02/01/2019 đến ngày 22/03/2019

3.2. Nội dung nghiên cứu

* Tìm hiểu lý thuyết NoSQL.
* Tìm hiểu MongoDB.
* Tìm hiểu bài toán Demo.
* Xây dựng Demo ứng dụng Web có sử dụng NoSQL, MongoDB.

3.3. Phương pháp nghiên cứu:

* Phương pháp nghiên cứu lý luận: thu thập, phân tích, tổng hợp, xử lý dữ liệu liên quan đến đề tài nghiên cứu.
* Phương pháp nghiên cứu bằng thực tiễn: tham khảo tài liệu và các ứng dụng đã được phát triển bằng MongoDB.
* Tham khảo ý kiến của thầy cô, kết hợp với sự tìm hiểu của bản thân.

# **PHẦN IV: KẾT QUẢ VÀ THẢO LUẬN**

**4.1. Tổng quan về cơ sở dữ liệu NoSQL.**

4.1.1. Tại sao lại chọn NoSQL.

Cơ sở dữ liệu quan hệ được thiết kế cho những mô hình dữ liệu không quá lớn trong khi các dịch vụ mạng xã hội lại có một lượng lớn dữ liệu và cập nhật liên tục do số lượng người dùng quá nhiều. Do đó cơ sở dữ liệu NoSQL sinh ra với mục tiêu giải quyết các thiếu sót của RDBMS trong các hệ thống phần mềm hiện đại. NoSQL sẽ tập trung giải quyết các vấn đề như tốc độ thực thi, khả năng lưu trữ, các nghiệp vụ phức tạp (phân trang, đánh chỉ mục…). Nhờ vậy giải pháp sử dụng cơ sở dữ liệu NoSQL sẽ hạ thấp chi phí nếu so sánh với RDBMS truyền thống. NoSQL vừa mang lại một giải pháp tốt hơn vừa tiết kiệm chi phí hơn do NoSQL có hiệu suất làm việc tốt hơn và các cơ sở dữ liệu NoSQL thường là miễn phí. Ngoại trừ một số trường hợp đặc biệt, với cùng một chi phí thì giải pháp sử dụng NoSQL sẽ mang lại lợi ích to lớn. Hãy tưởng tượng, với một hệ thống cho bạn đầy đủ quyền kiểm soát (mã nguồn mở), đáp ứng được tốc độ thực thi, khả năng lưu trữ, phân tán dữ liệu… và nhất là chi phí sẽ thấp hơn thì NoSQL chính là sự lựa chọn tuyệt vời.

Mặc khác, thường chúng ta sử dụng rất hạn chế những khả năng mà các cơ sở dữ liệu RDBMS cung cấp nhưng vẫn phải trả phí cho nó. Nếu không cần đến các tính năng cao cấp, không cần các chức năng của SQL hoặc thấy phức tạp khi viết các câu lệnh SQL thì hãy nghĩ đến NoSQL.

**4.1.2. Khái niệm NoSQL.**

Cơ sở dữ liệu NoSQL (tên gốc là "Non SQL" hoặc "non relational" - phi quan hệ) cung cấp một cơ chế để lưu trữ và truy xuất dữ liệu được mô hình hóa khác với các quan hệ bảng được sử dụng trong các cơ sở dữ liệu kiểu quan hệ.

Các cơ sở dữ liệu như vậy đã tồn tại kể từ cuối những năm 1960, nhưng không được gọi là "NoSQL" cho đến khi nổi tiếng đột ngột đầu thế kỷ XXI tạo nên bởi sự cần thiết cho các công ty Web 2.0 như Facebook, Google và Amazon.com. Các cơ sở dữ liệu NoSQL đang được sử dụng ngày càng nhiều trong các ứng dụng dữ liệu lớn và ứng dụng nền web thời gian thực. Các hệ thống NoSQL cũng đôi khi được gọi là "Not only SQL" (không chỉ là SQL) để nhấn mạnh rằng chúng có thể hỗ trợ các ngôn ngữ truy vấn dạng như SQL.

NoSQL được xem như thế hệ database kế tiếp của RDBMS, là một thế hệ cơ sở dữ liệu non-relational (không ràng buộc), distributed (phân tán), open source, horizontal scalable (khả năng mở rộng theo chiều ngang) có thể lưu trữ, xử lý từ một lượng rất nhỏ cho tới hàng petabytes dữ liệu trong hệ thống có độ chịu tải, lỗi cao với những đòi hỏi về tài nguyên phần cứng thấp. Để hiểu thêm về các khái niệm này trong NoSQL, có thể xem chi tiết ở phần 2.5 Một số thuật ngữ liên quan.

Một số đặc điểm nhận dạng cho thế hệ database này bao gồm:

* Lược đồ tự do (Schema-free).
* Hỗ trợ mở rộng dễ dàng.
* API đơn giản.
* Eventual consistency (nhất quán cuối) và transactions hạn chế trên các  
  thành phần dữ liệu đơn lẻ.
* Không giới hạn không gian dữ liệu…

NoSQL storage đặc biệt phổ dụng trong thời kỳ Web 2.0 bùng nổ, nơi các mạng dịch vụ dữ liệu cộng đồng cho phép người dùng tạo rất nhiều nội dung trên web. Do đó, dữ liệu lớn rất nhanh vượt qua giới hạn phần cứng và cần phải giải quyết bằng bài toán phân tán. Nửa đầu năm 2009, người ta đã manh nha thuật ngữ NoSQL đánh dấu sự trưởng thành của thế hệ database mới: distributed (phân tán) + non-relational (không ràng buộc).

Khi làm việc với NoSQL ta sẽ gặp một số khác niệm sau:

* Fields: tương đương với khái niệm Columns trong SQL. Một field tương đương một key trong document thể hiện một thuộc tính của dữ liệu.
* Document: thay thế khái niệm row trong SQL. Đây cũng chính là khái niệm làm nên sự khác biệt giữa NoSQL và SQL, 1 document chứa số cột (fields) không cố định trong khi 1 row thì số cột (columns) là định sẵn trước.
* Collection: tương đương với khái niệm table trong SQL. Một collection là tập hợp các document. Điều đặc biệt là một collection có thể chứa các document hoàn toàn khác nhau.
* Key-value: cặp khóa - giá trị được dùng để lưu trữ dữ liệu trong NoSQL. Trong mô hình này, dữ liệu được biểu diễn như một bộ sưu tập các cặp khóa - giá trị, như vậy mỗi khoá có thể xuất hiện chỉ một lần trong bộ sưu tập. Các ví dụ bao gồm cơ sở dữ liệu Oracle NoSQL, Redis, và dbm.
* Cursor: tạm dịch là con trỏ. Sử dụng cursor để lấy dữ liệu từ database.

Trong các hệ cơ sở dữ liệu quan hệ, các cột được định nghĩa theo bảng còn với hệ cơ sở dữ liệu không ràng buộc các cột được định nghĩa ở mỗi document. Bởi thế, các document quản lý gần như tất cả, các collection không cần quản lý chặt chẽ những gì đang xảy ra trong nó nữa.

4.1.3. Ưu và nhược điểm của NoSQL.

Sự ra đời của NoSQL đã cung cấp nhiều giải pháp hiệu quả cho việc lưu trữ dữ liệu trong xu hướng mới. Tuy nhiên, dù đã thể hiện được những ưu điểm của mình, NoSQL vẫn còn tồn tại không ít các hạn chế.

**Ưu điểm**

* NoSQL là mã nguồn mở: các sản phẩm nguồn mở đưa ra cho những người phát triển với nhiều lợi ích to lớn, trong đó việc sử dụng miễn phí là một lợi ích lớn. Ngoài ra phần mềm nguồn mở có xu hướng sẽ là tin cậy hơn, an ninh hơn và nhanh hơn để triển khai so với các lựa chọn thay thế sở hữu độc quyền. Ví dụ như các hệ quản trị cơ sở dữ liệu (CSDL) NoSQL: Cassandra, CouchDB, Hbase, RavenDB, MongoDB và Redis.
* Khả năng mở rộng linh hoạt, mềm dẻo: NoSQL thay thế câu thần chú cũ của các nhà quản trị CSDL về “mở rộng phạm vi” với một thứ mới: “mở rộng ra ngoài”. Thay vì bổ sung thêm các máy chủ lớn hơn để điều khiển nhiều tải dữ liệu hơn, thì CSDL NoSQL cho phép một công ty phân tán tải qua nhiều máy chủ khi mà tải gia tăng.
* Hiệu suất hoạt động cao: NoSQL có hiệu suất hoạt động cao, lưu trữ lượng lớn dữ liệu để đáp ứng nhu cầu lưu trữ ngày càng tăng hiện nay. Tuy nhiên để đạt được điều này cần loại bỏ đi một số ràng buộc dữ liệu của mô hình quan hệ, tính nhất quán dữ liệu, ngôn ngữ truy vấn SQL. Đồng thời NoSQL có một số cải tiến mới như sử dụng tốt index, khả năng phân tán dễ dàng đã giúp NoSQL có một hiệu suất hoạt động rất cao.
* Khả năng phân trang: phân trang trong cơ sở dữ liệu quan hệ khá khó khăn khi không có một phương pháp chính thống nào để phục vụ cho việc này. Người lập trình phải dùng các phương pháp khác nhau để có thể lấy đúng số item cần lấy. Trong khi NoSQL hỗ trợ rất tốt việc này đồng thời hiệu suất khi phân trang không hề giảm.
* NoSQL phù hợp với công nghệ lưu trữ đám mây: NoSQL và đám mây là một sự trùng khớp tự nhiên. Các máy chủ ngày nay là không đắt và có thể 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 các kết nối mạnh. Các CSDL NoSQL hầu hết sử dụng bộ nhớ qua đĩa như là vị trí ghi đầu tiên - vì thế ngăn ngừa được sự thực thi không ổn định của I/O. Và vì NoSQL lưu trữ dữ liệu thường thúc đẩy được tính mở rộng phạm vi theo chiều ngang thông qua việc ngăn chia, chúng có khả năng tận dụng được việc cung cấp mềm dẻo của đám mây.
* NoSQL được các hãng lớn sử dụng: Các công ty như Amazon, BBC, Facebook và Google dựa vào các CSDL NoSQL.
* Các CSDL NoSQL khác nhau cho những dự án khác nhau:
* MongoDB và Redis là những lựa chọn tốt cho việc lưu trữ các dữ liệu thống kê ít được đọc mà lại được viết thường xuyên, như một số đếm truy cập web chẳng hạn.
* Hadoop, một CSDL dạng tự do, phân tán làm tốt công việc lưu trữ các dữ liệu lớn như các con số thống kê thời tiết hoặc công việc phân tích nghiệp vụ.
* Memcache, một CSDL nhất thời chóng tàn, tuyệt vời trong lưu trữ các phiên làm việc web, các khóa, và các con số thống kê ngắn hạn.
* Cassandra và Riak (các lưu trữ dư thừa, tự động tạo bó cluster) làm tốt trong các môi trường với các ứng dụng có tính sẵn sàng cao, khi thời gian sống tối đa là sống còn

**Nhược điểm**

* Cấu trúc dữ liệu phi quan hệ: với cấu trúc dữ liệu phi quan hệ đã giúp NoSQL giảm đi rất nhiều tính toán không cần thiết. Điều này dẫn đến dữ liệu sẽ không ràng buộc chặt chẽ và ảnh hưởng tính nhất quán dữ liệu. Như vậy với các ứng dụng yêu cầu dữ liệu phải chặt chẽ như ứng dụng về tài chính, ngân hàng với các con số phải rất chính xác thì NoSQL không phải một sự lựa chọn tốt.
* Nguồn mở có thể có nghĩa là sự hỗ trợ không đồng đều cho các doanh nghiệp: trong khi các nhà cung cấp chủ chốt của RMBMs như Oracle, IBM hay Sybase đưa ra sự hỗ trợ tốt nổi tiếng cho các khách hàng doanh nghiệp cỡ vừa, thì các doanh nghiệp nhỏ hơn, thường là các nhà cung cấp nguồn mở mới thành lập không thể mong đợi được cung cấp sự hỗ trợ có thể so sánh được (ngoại trừ một nhóm các khách hàng blue chip). Nhà cung cấp nguồn mở trung bình thiếu sự tiếp cận toàn cầu, các dịch vụ hỗ trợ và sự tin cậy của Oracle hay IBM.
* Những hạn chế về tri thức nghiệp vụ: Có một vài câu hỏi xung quanh những khả năng về tri thức nghiệp vụ (BI) của các CSDL NoSQL. Liệu các CSDL này có thể cung cấp dạng phân tích dữ liệu lớn và mạnh mà các doanh nghiệp đã quen với các RDBMS? Cần bao nhiêu sự tinh thông về lập trình cần có để tiến hành những truy vấn và phân tích hiện đại? Các câu trả lời là không tích cực. Các CSDL NoSQL không có nhiều sự đeo bám tới các công cụ BI thường được sử dụng, trong khi những yêu cầu và phân tích hiện đại đơn giản nhất thì cũng liên quan khác nhiều tới sự tinh thông về lập trình. Tuy vậy, các giải pháp là sẵn sàng. QuestSoftware, ví dụ, đã tạo ra Toad cho các CSDL đám mây, mà nó phân phối các khả năng truy vấn hiện đại tới một số CSDL NoSQL.
* Những vấn đề về tính tương thích: Không giống như các CSDL quan hệ, các CSDL NoSQL chia sẻ ít theo cách thức của các tiêu chuẩn. Mỗi CSDL NoSQL có các giao diện lập trình ứng dụng API riêng của mình, các giao diện truy vấn độc nhất vô nhị, và những sự riêng biệt. 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 nếu bạn không hài lòng với dịch vụ.
* Thiếu sự tinh thông: Tính rất mới mẻ của NoSQL có nghĩa là không có nhiều lập trình viên và người quản trị mà biết công nghệ này - là khó khăn cho các công ty tìm người với sự tinh thông phù hợp. Đổi lại, thế giới của RDBMS có hàng ngàn những người đủ tư cách.

4.1.4. Kiến trúc

Các RDBMS hiện tại đã bộc lộ những yếu kém như việc đánh chỉ mục một lượng lớn dữ liệu, phân trang, hoặc phân phối luồng dữ liệu media (phim, ảnh, nhạc...). Cơ sở dữ liệu quan hệ được thiết kế cho những mô hình dữ liệu nhỏ thường xuyên đọc viết trong khi các Social Network Services lại có một lượng dữ liệu cực lớn và cập nhật liên tục do số lượng người dùng quá nhiều ở một thời điểm. Thiết kế trên Distributed NoSQL giảm thiểu tối đa các phép tính toán, I/O liên quan kết hợp với batch processing đủ đảm bảo được yêu cầu xử lý dữ liệu của các mạng dịch vụ dữ liệu cộng đồng này. Facebook, Amazon là những ví dụ điển hình.

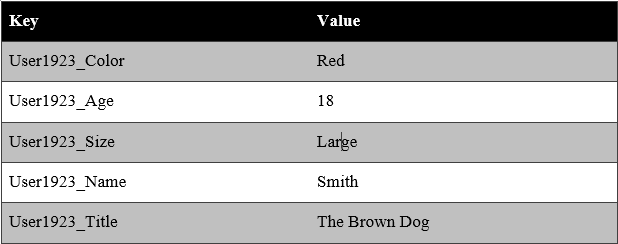
Về cơ bản, các thiết kế của NoSQL lựa chọn mô hình lưu trữ tập dữ liệu theo cặp giá trị key-value. Khái niệm node được sử dụng trong quản lý dữ liệu phân tán.

Figure: ví dụ cơ bản về Key/Value

Với các hệ thống phân tán, việc lưu trữ chấp nhận trùng lặp dữ liệu. Một yêu cầu truy vấn dữ liệu có thể gửi tới nhiều máy cùng lúc, khi một máy nào nó bị chết cũng không ảnh hưởng nhiều tới toàn bộ hệ thống. Để đảm bảo tính thời gian thực trong các hệ thống xử lý lượng lớn dữ liệu, thông thường người ta sẽ tách biệt database ra làm 2 hoặc nhiều database như sơ đồ dưới đây.

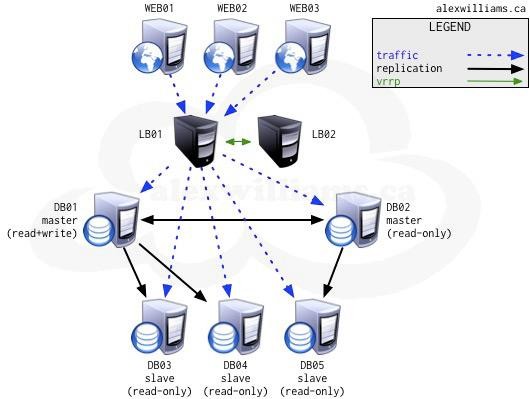


Figure: Sơ đồ thiết kế hệ thống database Master –Slave

4.1.5. Một số thuật ngữ liên quan.

* **Non-relational**:

Relational - ràng buộc, thuật ngữ sử dụng để chỉ ra các mối quan hệ giữa các bảng trong cơ sở dữ liệu quan hệ (RDBMS) sử dụng mô hình khóa gồm 2 loại khóa: khóa chính và khóa phụ (primary key + foreign key) để ràng buộc dữ liệu nhằm thể hiện tính nhất quán dữ liệu từ các bảng khác nhau.

Nonrelational là khái niệm không sử dụng các ràng buộc dữ liệu cho nhất quán dữ liệu ở NoSQL database.

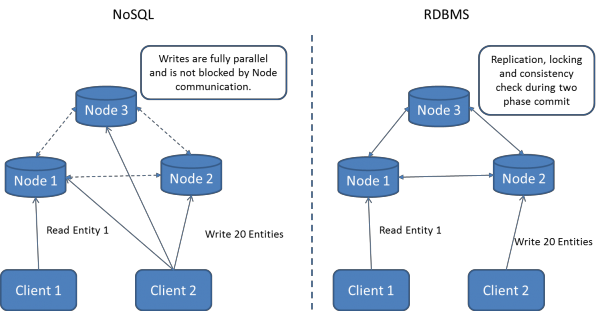


Figure: So sánh cách thiết kế giữa NoSQL và RDBMS

Nhìn vào hình trên ta thấy NoSQL có cách thiết kế lỏng lẻo, không ràng buộc chặt chẽ như RDBMS. Các mối liên kết giữa các Node trong NoSQL chỉ là liên kết ảo, NoSQL không nhìn thấy mối liên kết gì ở đây cả. Tuy nhiên nhờ bỏ qua tính ràng buộc này đã giúp cho NoSQL có khả năng làm việc tốt với lượng dữ liệu lớn.

* **Distributed storage**:

Mô hình lưu trữ phân tán các file hoặc dữ liệu ra nhiều máy tính khác nhau trong mạng LAN hoặc Internet dưới sự kiểm soát của phần mềm.

* **Eventual consistency (nhất quán cuối cùng)**:

Tính nhất quán của dữ liệu không cần phải đảm bảo ngay tức khắc sau mỗi phép write. Một hệ thống phân tán chấp nhận những ảnh hưởng theo phương thức lan truyền và sau một khoảng thời gian (không phải ngay tức khắc) thay đổi sẽ đi đến mọi điểm trong hệ thống, tức là cuối cùng (eventually) dữ liệu trên hệ thống sẽ trở lại trạng thái nhất quán.

* **Vertical scalable (khả năng mở rộng chiều dọc)**:

Khi dữ liệu lớn về lượng, phương pháp tăng cường khả năng lưu trữ và xử lý bằng việc cải thiện phần mềm và cải thiện phần cứng trên một máy tính đơn lẻ được gọi là khả năng mở rộng chiều dọc. Ví dụ việc tăng cường CPUs, cải thiện đĩa cứng, bộ nhớ trong một máy tính... cho RDBMS nằm trong phạm trù này. Khả năng mở rộng chiều dọc còn có một thuật ngữ khác scale up.

* **Horizontal scalable (khả năng mở rộng chiều ngang):**
* Khi dữ liệu lớn về lượng, phương pháp tăng cường khả năng lưu trữ và xử lý là dùng nhiều máy tính phân tán. Phân tán dữ liệu được hỗ trợ bởi phần mềm tức cơ sở dữ liệu.
* Trong khi giá thành phần cứng ngày càng giảm, tốc độ xử lý, bộ nhớ ngày càng tăng thì horizontal scalable là một lựa chọn đúng đắn. Hàng trăm máy tính nhỏ được chập lại tạo thành một hệ thống tính toán mạnh hơn nhiều so với vi xử lý RISC truyền thống đơn lẻ. Mô hình này tiếp tục được hỗ trợ bởi các công nghệ kết nối Myrinet và InfiniBand. Từ đó chúng ta có thể quản lý, bảo trì từ xa, xây dựng batch procession (xử lý đồng loạt tập lệnh) tốt hơn. Do những đòi hỏi về tốc độ xử lý I/O cao, lượng cực lớn dữ liệu,... scale horizontally sẽ thúc đẩy các công nghệ lưu trữ mới phát triển giống như object storage devices (OSD).

## **4.1.6. So sánh NoSQL với các loại cơ sở dữ liệu khác.**

Để thấy sự khác biệt của NoSQL với các phương thức lưu trữ khác, chúng ta sẽ so sánh NoSQL với XML và RDBMS. Lý do lựa chọn XML và RDBMS để so sánh với NoSQL là vì:

* XML là phương thức lưu trữ dữ liệu dạng văn bản tương tự như cách lưu trữ của một số NoSQL sử dụng encoding là XML hoặc JSON.
* RDBMS là hệ quản trị cơ sở dữ liệu đã rất thành công với mô hình dữ liệu quan hệ cho hệ thống vừa và nhỏ.
* **Những điểm khác biệt giữa NoSQL với RDBMS.**

Như đã đề cập ở trên, cơ sở dữ liệu NoSQL sinh ra để giải quyết các thiếu sót của RDBMS trong các hệ thống phần mềm hiện đại. NoSQL sẽ tập trung giải quyết các vấn đề như tốc độ thực thi, khả năng lưu trữ, các nghiệp vụ phức tạp (phân trang, đánh chỉ mục...) và NoSQL đã làm được điều đó.

|  |  |  |
| --- | --- | --- |
| Tham số | SQL | NoSQL |
| Định nghĩa | CSDL SQL chủ yếu được gọi là RDBMS hoặc CSDL quan hệ. | CSDL NoSQL chủ yếu được gọi là CSDL phi quan hệ hoặc phân tán. |
| Design for | RDBMS truyền thống sử dụng cú pháp và truy vấn SQL để phân tích và lấy dữ liệu để có thêm thông tin chi tiết. Được sử dụng cho các hệ thống OLAP. | Hệ thống CSDL NoSQL bao gồm nhiều loại công nghệ CSDL khác nhau. Các CSDL này được phát triển để đáp ứng nhu cầu trình bày cho sự phát triển của ứng dụng hiện đại. |
| Ngôn ngữ Query | Structured query language (SQL) | Không có ngôn ngữ Query |
| Type | SQL databases là cơ sở dữ liệu dựa trên bảng | NoSQL databases có thể dựa trên tài liệu, cặp Key-value, CSDL đồ thị |
| Schema | SQL databases có lược đồ được xác định trước | NoSQL databases sử dụng lược đồ động cho dữ liệu phi cấu trúc. |
| Khả năng mở rộng | SQL databases có thể mở rộng theo chiều dọc, hạn chế về lượng | NoSQL databases có thể mở rộng theo chiều ngang. Hỗ trợ một lượng rất lớn các Node. |
| Ví dụ | Oracle, Postgres, and MS-SQL. | MongoDB, Redis, , Neo4j, Cassandra, Hbase. |
| Thay đổi các Node trong hệ thống | Phải shutdown cả hệ thống.  Việc thay đổi số node phức  tạp | Không cần phải shutdown cả  hệ thống. Việc thay đổi số node đơn giản, không ảnh hưởng đến hệ thống. |
| Hiệu suất | Kém hơn NoSQL do truy vấn giữa các bảng phức tạp hơn | Tốt, bỏ qua các liên kết ràng buộc dữ liệu |
| Hiệu suất đọc - ghi | Kém do thiết kế để đảm bảo sự vào/ra liên tục của dữ liệu | Tốt với mô hình xử lý lô và những tối ưu về đọc - ghi dữ liệu |
| Phù hợp | Là lựa chọn lý tưởng cho môi trường truy vấn phức tạp | Không phù hợp với truy vấn phức tạp |
| Lưu trữ dữ liệu phân cấp | SQL databases không thích hợp cho việc lưu trữ dữ liệu phân cấp. | Phù hợp hơn cho kho lưu trữ dữ liệu phân cấp vì nó hỗ trợ phương thức cặp key-value. |
| Variations | Một loại có biến thể nhỏ | Nhiều loại khác nhau bao gồm các kho key-value, database document và cơ sở dữ liệu đồ thị. |
| Năm phát triển | Được phát triển vào những năm 1970 để giải quyết các vấn đề với lưu trữ tệp phẳng | Được phát triển vào cuối những năm 2000 để khắc phục các vấn đề và hạn chế của SQL databases. |
| Open-source | Một sự kết hợp của mã nguồn mở như Postgres & MySQL, và thương mại như Oracle Database. | Open-source |
| Tính nhất quán | Nó phải được cấu hình cho sự nhất quán chặt chẽ. | Cung cấp tính nhất quán mạnh mẽ như MongoDB, trong khi những cơ sở dữ liệu khác cung cấp sự nhất quán cuối cùng như Cassandra. |
| Được sử dụng tốt nhất cho | RDBMS database là tùy chọn thích hợp để giải quyết các vấn đề về ACID. | NoSQL được sử dụng tốt nhất để giải quyết các vấn đề về tính khả dụng của dữ liệu |
| Tầm quan trọng | Nó nên được sử dụng khi hiệu lực dữ liệu là siêu quan trọng | Sử dụng khi nó quan trọng hơn để có dữ liệu chính xác với tốc độ truy suất dữ liệu nhanh hơn |
| Lựa chọn tốt nhất | Khi cần hỗ trợ truy vấn động | Sử dụng khi cần mở rộng quy mô dựa trên yêu cầu thay đổi |
| Hardware | Specialized DB hardware (Oracle Exadata, etc.) | Commodity hardware |
| Network | Highly available network (Infiniband, Fabric Path, etc.) | Commodity network (Ethernet, etc.) |
| Loại lưu trữ | Highly Available Storage (SAN, RAID, etc.) | Commodity drives storage (standard HDDs, JBOD) |
| Tính năng tốt nhất | Hỗ trợ đa nền tảng, bảo mật và miễn phí | Dễ sử dụng, hiệu suất cao và công cụ linh hoạt. |
| Mô hình ACID và BASE | ACID (Atomicity, nhất quán, cách ly và độ bền) là một chuẩn cho RDBMS | Cơ bản (có sẵn, trạng thái mềm, phù hợp cuối cùng) là một mô hình của nhiều hệ thống NoSQL |
| Performance | SQL hoạt động tốt và nhanh thì việc desgin tốt là cực kì quan trọng và ngược lại. | Nhanh hơn SQL, NoSQL thì denormalized cho phép lấy được tất cả thông tin về một item cụ thể với các codition mà không cần JOIN liên quan hoặc truy vấn SQL phức tạp. |
| Phần cứng | Đòi hỏi cao về phần cứng | Đòi hỏi thấp hơn về giá trị và tính đồng nhất của phần cứng |
| Kết luận | Dự án đã có yêu cầu dữ liệu rõ ràng xác định quan hệ logic có thể được xác định trước. | Phù hợp với những dự án yêu cầu dữ liệu không liên quan, khó xác định, đơn giản mềm dẻo khi đang phát triển |

Table: so sánh SQL và NoSQL

Để thấy hiệu suất mà NoSQL đạt được, hãy xem các biểu đồ dưới đây là kết quả của phép so sánh giữa MongoDB (một cơ sở dữ liệu NoSQL) và MSSQL.

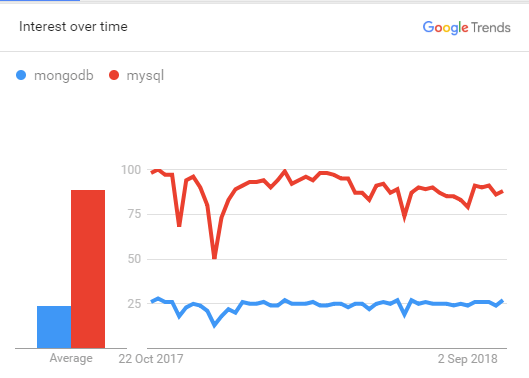


Figure: NoSQL (mongoDB) Vs RDBMS (mysql) Google Trend

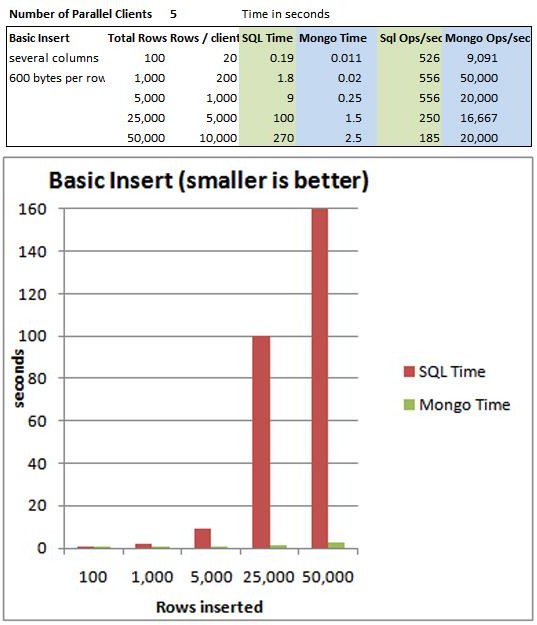
­­­­­

Figure: kết quả Insert data năm 2008

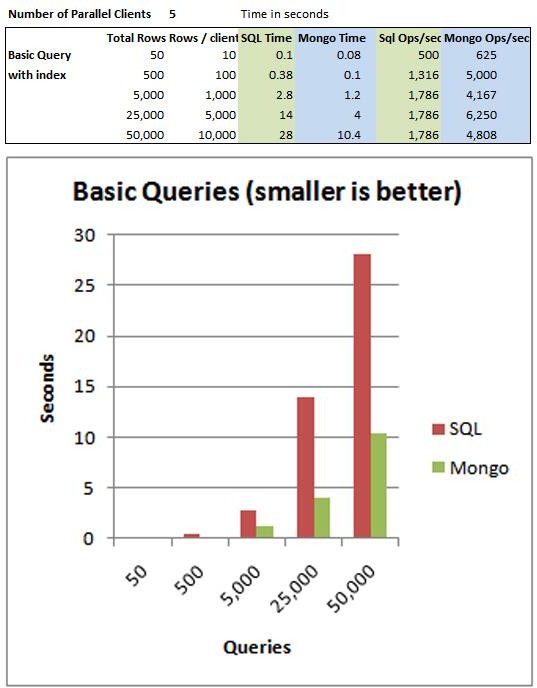


Figure: kết quả truy vấn năm 2008

Như vậy, NoSQL khắc phục được rất nhiều nhược điểm của cơ sở dữ liệu phi quan hệ và mang đến một giải pháp rất tốt cho nhu cầu lưu trữ dữ liệu lớn.

* **So sánh NoSQL với XML.**

Cả NoSQL và XML đều có phương thức lưu trữ tương tự nhau: lưu dạng văn bản. XML dùng để lưu trữ dữ liệu sử dụng các thẻ đánh dấu. Tuy nhiên để sử dụng XML như một cơ sở dữ liệu sẽ có một số thuận lợi và khó khăn như sau:

* Thuận lợi: có thể kiểm soát tất cả, nắm được luồng xử lý của hệ thống.
* Khó khăn: XML chỉ là các file văn bản nên không có một platform cho truy xuất dữ liệu do đó cần phải xây dựng mới hoàn toàn lớp thao tác dữ liệu với XML như insert, delete, update, query như vậy rất tốn chi phí. Ngoài ra việc tự xây dựng nhiều khi mang đến một kết quả không tốt ví dụ như source code chưa được tối ưu, chưa có giải thuật tốt.

Do đó, hiệu suất hoạt động có tốt hay không phụ thuộc rất nhiều vào lớp mới mà người lập trình tạo ra. Nên không có một đảm bảo nào cho việc sử dụng XML có thể cho hiệu suất tốt hơn NoSQL khi mà:

- Cơ sở dữ liệu NoSQL được các nhà lập trình chuyên nghiệp xây dựng ra với nhiều giải thuật, khả năng tối ưu source code cao mang đến một hiệu suất làm việc tuyệt vời.

- Các đặc điểm khác như: khả năng phân tán dữ liệu, đánh số index, phân trang, transaction hoặc các gói hổ trợ nâng cao như bảo mật, mã hoá thông tin… thì khó mà lập trình ra.

Như vậy, NoSQL đã làm tốt nhiệm vụ của nó đồng thời còn là mã nguồn mở thì ta đâu có lý do gì phải xây dựng một hệ thống lưu trữ mới dựa trên các file XML đầy khó khăn.

**4.1.7. Cách triển khai một ứng dụng NoSQL.**

* **Xác định NoSQL có phù hợp**

Khi làm việc với một lượng lớn dữ liệu, chúng ta nên nghĩ đến NoSQL. NoSQL rất thích hợp để làm việc với dữ liệu lớn bằng cách loại bỏ các ràng buộc toàn vẹn dữ liệu, cách thiết kế mô hình phi chuẩn hoá, cách sử dụng index…. Đã giúp NoSQL trở nên mạnh mẽ để làm việc với lượng lớn dữ liệu. Tuy nhiên, có một số tính chất sau đây cần lưu ý khi lựa chọn cơ sở dữ liệu NoSQL.

Như đã đề cập trong mục 4.1.5 Một số thuật ngữ liên quan, tính nhất quán cuối (Eventual consistency) cần phải được ứng dụng chấp nhận. Có nghĩa là ứng dụng không yêu cầu ràng buộc dữ liệu, không yêu cầu dữ liệu phải cập nhật chính xác ngay tức thì. Một số ứng dụng phù hợp như các trang mạng xã hội, các ứng dụng ghi log tự động… Các ứng dụng loại này chấp nhận dữ liệu cũ trong một khoảng thời gian ngắn trước khi được cập nhật mới. Đổi lại chúng ta đạt được những tiêu chuẩn cao về khả năng mở rộng và hiệu quả về chi phí, trong khi phục vụ liên tục hàng triệu khách hàng từ khắp nơi. Đặc biệt chúng ta đạt được một hiệu suất hoạt động cao hơn gấp nhiều lần nhờ vào việc loại bỏ các yêu cầu nhất quán dữ liệu.

Các ứng dụng không phù hợp với cơ sở dữ liệu NoSQL là các ứng dụng yêu cầu tính nhất quán dữ liệu cao. Tính nhất quán dữ liệu được xem như tính sống còn của ứng dụng. Ví dụ như các ứng dụng tài chính, ngân hàng… với các con số luôn được cập nhật và cần được cập nhật tức thì. Sự chậm trễ có thể sẽ phải trả giá rất đắt. Bởi thế nếu các ứng dụng của bạn thuộc loại này thì hãy lựa chọn cơ sở dữ liệu RDBMS với mô hình quan hệ truyền thống. Các yêu cầu phân tích hiện đại (BI - business intelligence) cũng không phù hợp với cơ sở dữ liệu NoSQL này. Bởi vì NoSQL hổ trợ rất ít các câu truy vấn. Tất cả đều phụ thuộc vào sự tinh thông lập trình. Như vậy, với một yêu cầu phân tích đơn giản thì cũng cần đến lập trình trong đó. Trong khi với cơ sở dữ liệu RDBMS sử dụng ngôn ngữ SQL để truy vấn, SQL giúp chúng ta rất nhiều việc trong truy vấn, phân tích.

* **Thiết kế cấu trúc dữ liệu dạng document**

NoSQL lưu trữ dữ liệu không theo một lược đồ cố định, nó có lược đồ tùy ý tùy biến. Nhưng điều đó không có nghĩa rằng chúng ta không nên dành nhiều thời gian để xem xét làm thế nào để thiết kế các document đảm bảo rằng chúng ta có thể truy cập tất cả dữ liệu chúng ta cần để phục vụ các yêu cầu của người dùng một cách hiệu quả, đáng tin cậy và chi phí bảo trì ít nhất có thể.

Lỗi điển hình nhất mà chúng ta mắc phải là cố gắng thiết kế mô hình dữ liệu của document database giống với cách chúng ta thiết kế mô hình dữ liệu trong cơ sở dữ liệu quan hệ. NoSQL lưu trữ dữ liệu phi quan hệ. Nếu cố gắng thiết kế theo mô hình quan hệ thì chúng ta có được nhiều kết quả tốt. Nhưng chúng ta sẽ đạt được kết quả vô cùng to lớn nếu sử dụng những điểm mạnh của document database.

Để thiết kế cấu trúc dữ liệu dạng document cần lưu ý 2 điểm sau:

* Các document có cấu trúc không giống nhau:

Trong RDBMS, một dòng chỉ có thể chứa dữ liệu đơn giản và những cấu trúc dữ liệu phức tạp hơn cần được lưu trữ như là mối quan hệ (có nghĩa là lưu dữ liệu trong nhiều bảng khác nhau và sử dụng khóa ngoại để tham chiếu). Đối với document database, một document có thể lưu một đối tượng phức tạp tùy ý. Đối tượng đó có thể là arrays, dictionaries và trees.

* Thiết kế dữ liệu phi quan hệ:

Tư tưởng thiết kế ở đây là đi ngược lại với thiết kế chuẩn hoá, mục tiêu hạn chế các phép “join”. Ở đây có thể chấp nhận dữ liệu dư thừa và không thống nhất trong 1 khoảng thời gian và sau đó sẽ được cập nhật lại. Bù lại sẽ nhận được một hiệu suất hoạt động mạnh mẽ với lượng lớn dữ liệu.

Vấn đề đặt ra khi ta cần cập nhật dữ liệu. Như ví dụ cần cập nhật: tên của “Customer. Đối với thiết kế chuẩn hoá, ta chỉ cần cập nhật ở 1 nơi là table Customer. Nhưng đối với thiết kế document thì khác tên của “Customer” đặt ở nhiều nơi: trong object Customer và trong các object Order. Việc cập nhật lại tên của Customer là phụ thuộc vào nghiệp vụ của chương trình. Khi xây dựng chương trình, ta phân tích xem tên của Customer có cần được cập nhật ở tất cả các nơi hay chỉ cần ở 1 số nơi. Từ đó ta sẽ viết code cho việc cập nhật này. Tất cả đều do phân tích cho từng chương trình sao cho hiệu suất hoạt động tốt nhất và nghiệp vụ vẫn đúng.

**4.2. Phân loại cơ sở dữ liệu NoSQL.**

* + 1. **Key-Value Databases**
* **Key-Value Database là gì?**

Key-value Database là một bảng băm đơn giản để sử dụng từ phối cảnh API, chủ yếu được sử dụng khi tất cả quyền truy cập vào cơ sở dữ liệu đều thông qua khóa chính. Một bảng trong RDBMS truyền thống có hai cột, chẳng hạn như ID và NAME, cột ID là cột chính và cột NAME để lưu trữ giá trị. Trong RDBMS, cột NAME bị hạn chế để lưu trữ dữ liệu của kiểu String. Ứng dụng có thể cung cấp ID, Value và duy trì cặp. Nếu ID đã tồn tại giá trị hiện tại được ghi đè, nếu không, một mục nhập mới được tạo.

Kiểu dữ liệu này Client có thể lấy được giá trị từ key, đặt giá trị cho một key, hay xóa bỏ key khỏi cơ sở dữ liệu. Chỉ lưu trữ các giá trị (value) mà không quan tâm đến giá trị đó chứa gì bên trong, việc đó dành cho Application. Mặt khác kiểu CSDL này luôn luôn sử dụng truy cập kiểu primary-key nên rất dễ dàng mở rộng cũng như đem lại hiệu năng tốt.

Một số key-value database có thể kể đến:

* Riak, Redis thường được sử dụng như Data Structure server - dữ liệu cấu trúc máy chủ. Chúng ta hãy nhìn vào thuật ngữ so sánh như thế nào trong Oracle và Riak.

|  |  |
| --- | --- |
| **Oracle** | **Riak** |
| Database instance | Riak Cluster |
| Table | Bucket |
| Row | Key-value |
| Rowid | Key |

* Memcached, Berkeley DB, HamsterDB đặc biệt thích hợp với hệ thống nhúng.
* Amazon DynamoDB không phải nguồn mở, mở rộng dựa trên SSD nhanh, nhiều khu vực sẵn có, tích hợp đàn hồi MapReduce.
* Project Voldemort là một nguồn mở để triển khai Amazon DynamoDB.
* Aerospike có quy mô cơ sở dữ liệu nhanh. RAM hoặc SSD. Hiệu suất dự đoán đạt được 2,5 M TPS (đọc và viết), 99% dưới 1 ms. Điều chỉnh nhất quán, được sao chép, cấu hình bằng không, thời gian chết bằng không, tự động phân cụm, nâng cấp cuộn, sao chép dữ liệu chéo. Viết bằng: C. API: C, C #, Erlang, Go, Java, Libevent, Node, Perl, PHP, Python, Ruby. Kết hợp phân mảnh bộ nhớ.

Các key-value database đa phần được thiết kế khá khác nhau. Những đặc điểm này quyết định key-value database nào nên được chọn khi triển khai một giải pháp cho hệ thống. Ví dụ như:

* Dữ liệu trong Memcached là không bền vững trong khi Riak là có.
* Redis tổng hợp được lưu trữ không phải là một miền đối tượng có thể là bất kỳ cấu trúc dữ liệu nào. Redis hỗ trợ lưu trữ danh sách, bộ, băm và có thể thực hiện trên phạm vi, và các hoạt động giao nhau. Các tính năng này cho phép Redis được sử dụng theo nhiều cách khác nhau hơn là một key-value database tiêu chuẩn.

**Riak:**

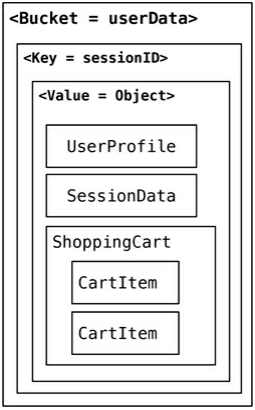
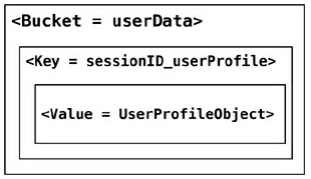
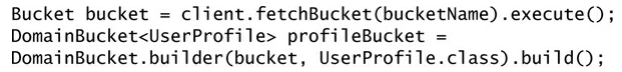
* Có nhiều cơ sở dữ liệu quan trọng hơn và nhiều cơ sở dữ liệu mới đang được thực hiện tại thời điểm này. Nên chúng ta chỉ phân tích rõ hơn về Riak bởi vì Riak cho phép lưu trữ các khóa vào các buckets.
* Muốn lưu trữ dữ liệu phiên người dùng, thông tin giỏ hàng và tùy chọn người dùng trong Riak, chỉ có thể lưu trữ tất cả trong cùng một nhóm với một khóa và một giá trị duy nhất cho tất cả các đối tượng. Trong kịch bản này, chúng ta sẽ có một đối tượng lưu trữ tất cả dữ liệu và được đưa vào một bucket.

Figure: Lưu trữ tất cả dữ liệu trong một nhóm

Nhược điểm của việc lưu trữ tất cả các đối tượng (tập hợp) khác nhau trong một nhóm sẽ là một bucket sẽ lưu trữ các loại tài liệu khác nhau, làm tăng cơ hội xung đột dữ liệu. Một giải pháp đưa ra sẽ là nối tên của đối tượng vào khóa, chẳng hạn như 288790b8a421\_userProfile, để có thể kết nối đến từng đối tượng khi cần thiết.

Figer: Thay đổi thiết kế khóa để phân đoạn dữ liệu trong một nhóm



Sử dụng các tên miền hoặc các nhóm khác nhau cho các đối tượng khác nhau (như UserProfile và ShoppingCart) phân đoạn dữ liệu qua các nhóm khác nhau cho phép bạn chỉ đọc đối tượng bạn cần mà không phải thay đổi thiết kế chính.

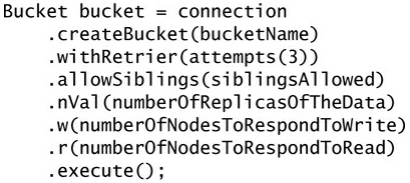
* **Tính năng lưu trữ key-value**

Trong khi sử dụng bất kỳ dữ liệu NoQuery nào, có một nhu cầu không thể tránh khỏi để hiểu cách các tính năng so sánh với các kho lưu trữ dữ liệu RDBMS tiêu chuẩn mà chúng ta đã quá quen thuộc. Lý do chính là hiểu những tính năng nào còn thiếu và kiến ​​trúc ứng dụng cần thay đổi thành như thế nào sử dụng tốt hơn các tính năng của kho lưu trữ dữ liệu key-value. Một số tính năng chúng tôi sẽ thảo luận cho tất cả. Các cửa hàng dữ liệu của NoQuery là tính nhất quán, giao dịch, tính năng truy vấn, cấu trúc của dữ liệu và tỷ lệ.

* **Tính nhất quán:**

Tính nhất quán chỉ được áp dụng cho các hoạt động trên một khóa duy nhất, vì các hoạt động này là một, đặt hoặc xóa trên một phím duy nhất. Rất tốn kém để thực hiện, bởi vì sự thay đổi giá trị không thể được xác định bởi kho lưu trữ dữ liệu.

Trong các triển khai lưu trữ key-value phân tán như Riak tính nhất quán được thực hiện. Vì giá trị có thể đã được sao chép sang các nút khác, nên Riak có hai cách để giải quyết xung đột cập nhật: hoặc là viết mới và viết cũ hơn, hoặc cả hai (tất cả) giá trị được trả về cho phép khách hàng giải quyết xung đột.

Ở Riak, các tùy chọn này có thể được thiết lập trong quá trình tạo bucket. Bucket chỉ là một cách để không gian tên ví dụ như các khóa va chạm có thể được giảm bớt, ví dụ, tất cả các khóa khách hàng có thể nằm trong bucket khách hàng. Khi tạo một nhóm, các giá trị mặc định cho tính nhất quán có thể được cung cấp, cho ví dụ rằng ghi chỉ được coi là tốt khi dữ liệu nhất quán trên tất cả các nút trong đó dữ liệu được lưu trữ.

Nếu chúng ta cần dữ liệu trong mỗi nút để thống nhất, chúng ta có thể tăng numberOfNodesToRonymousToWrite được đặt bởi w giống với nVal. Tất nhiên làm điều đó sẽ giảm hiệu suất ghi của cụm. Để cải thiện việc viết hoặc đọc xung đột, chúng ta có thể thay đổi allowSiblings trong quá trình tạo nhóm.

* **Tính năng giao dịch**

Các sản phẩm khác nhau của loại key-value database có thông số kỹ thuật khác nhau của giao dịch. Nói chung là không có đảm bảo trên các văn bản. Nhiều loại dữ liệu thực hiện giao dịch trong những cách khác. Riak sử dụng khái niệm đại diện được triển khai bằng cách sử dụng giá trị W, yếu tố ứng dụng đơn giản trong cuộc gọi viết API. Giả sử chúng ta có một cụm Riak với hệ số sao chép là 5 và chúng ta cung cấp giá trị W là 3. Khi nào viết, viết được báo cáo là thành công chỉ khi nó được viết và báo cáo là thành công vào lúc ít nhất ba trong số các nút. Điều này cho phép Riak có khả năng chịu đựng viết; trong ví dụ của chúng tôi, với N bằng 5 and với giá trị W là 3, cụm có thể chịu đựng được N - W = 2 nút bị ngừng hoạt động ghi, mặc dù vẫn sẽ mất một số dữ liệu trên các nút đó để đọc.

* **Tính năng truy vấn**

Tất cả key-value database có thể truy vấn bằng khóa chính. Nếu yêu cầu truy vấn bởi sử dụng một số thuộc tính của cột giá trị, nó không thể sử dụng cơ sở dữ liệu.

Truy vấn bằng khóa cũng có tác dụng phụ thú vị. Điều gì sẽ xảy ra nếu không biết khóa, đặc biệt là trong thời gian truy vấn và trong quá trình gỡ lỗi. Hầu hết các dữ liệu sẽ không cung cấp một danh sách tất cả các chìa khóa. Việc lấy danh sách các khóa và sau đó truy vấn giá trị sẽ rất cồng kềnh. Một số cơ sở dữ liệu khóa giá trị khắc phục điều này bằng cách cung cấp khả năng tìm kiếm bên trong giá trị, chẳng hạn như tìm kiếm Riak cho phép bạn truy vấn dữ liệu giống như bạn sẽ truy vấn nó bằng cách sử dụng Chỉ số Lucene.

Trong khi sử dụng các key-value database, rất nhiều suy nghĩ phải được đưa ra cho thiết kế của khóa. Chìa khóa có thể tạo bằng thuật toán nào? Khóa có thể được cung cấp bởi người dùng (ID người dùng, email...) không? Hoặc là bắt nguồn từ dấu thời gian hoặc dữ liệu khác có thể được lấy từ bên ngoài cơ sở dữ liệu?

Các đặc điểm truy vấn này làm cho các key-value database có khả năng để lưu trữ dữ liệu phiên (với ID phiên làm chìa khóa), dữ liệu giỏ hàng, hồ sơ người dùng... Thuộc tính expiry\_secs có thể được sử dụng để hết hạn khóa sau một khoảng thời gian nhất định, đặc biệt đối với các đối tượng giỏ hàng phiên / mua sắm.

* **Cấu trúc dữ liệu**

Cơ sở dữ liệu khóa-giá trị không quan tâm những gì được lưu trữ trong phần giá trị của cặp key-value. Giá trị có thể là một blob, văn bản, JSON, XML, v.v. Ở Riak, chúng ta có thể sử dụng Kiểu nội dung trong yêu cầu POST để chỉ định kiểu dữ liệu.

* **Các trường hợp sử dụng phù hợp**
* **Lưu trữ thông tin phiên:**

Nói chung, mỗi phiên web là duy nhất và được gán một giá trị sessionid duy nhất. Các ứng dụng lưu trữ sessionid trên đĩa hoặc trong RDBMS sẽ được hưởng lợi rất nhiều từ việc chuyển sang kho lưu trữ khóa-giá trị, vì mọi thứ về phiên có thể được lưu trữ bởi một yêu cầu PUT duy nhất hoặc được truy xuất bằng GET. Thao tác yêu cầu này làm cho nó rất nhanh, vì mọi thứ về phiên được lưu trữ trong một vật. Các giải pháp như Memcached được sử dụng bởi nhiều ứng dụng web và có thể sử dụng Riak khi sẵn có là quan trọng.

* **Hồ sơ người dùng:**

Hầu hết mọi người dùng đều có một userId, tên người dùng hoặc một số thuộc tính khác, cũng như các tùy chọn chẳng hạn như ngôn ngữ, màu sắc, múi giờ, những sản phẩm mà người dùng có quyền truy cập... Tất cả có thể là đưa vào một đối tượng, do đó, việc nhận các tùy chọn của người dùng chỉ cần một thao tác GET. Tương tự, sản phẩm hồ sơ có thể được lưu trữ.

* **Dữ liệu giỏ hàng**

Các trang web thương mại điện tử có giỏ mua hàng gắn liền với người dùng. Giỏ hàng được có sẵn mọi lúc, trên các trình duyệt, máy và phiên, tất cả thông tin mua sắm có thể được đưa vào giá trị trong đó khóa là userid. Một cụm Riak sẽ phù hợp nhất cho các loại ứng dụng.

* **Các trường hợp không sử dụng:**

Có những vấn đề mà key-value database không phải là giải pháp tốt nhất.

* **Mối quan hệ giữa các dữ liệu**:

Nếu mối quan hệ giữa các bộ dữ liệu khác nhau hoặc tương quan dữ liệu giữa các bộ khóa khác nhau, các key-value database không phải là giải pháp tốt nhất để sử dụng, mặc dù một số key-value database cung cấp các tính năng liên kết tốt.

* **Giao dịch đa hoạt động:**

Nếu đang lưu nhiều khóa và không thể lưu bất kỳ khóa nào trong số đó và muốn khôi phục phần còn lại của các hoạt động, các key-value database không phải là giải pháp tốt nhất được sử dụng.

* **Truy vấn theo dữ liệu:**

Nếu tìm kiếm các khóa dựa trên thứ được tìm thấy trong phần giá trị của các cặp key-value thì key-value database sẽ không hoạt động tốt. Không có cách nào để kiểm tra giá trị trên phía cơ sở dữ liệu, ngoại trừ một số sản phẩm như Tìm kiếm hoặc công cụ lập chỉ mục như Lucene.

* **Hoạt động theo bộ:**

Vì các hoạt động được giới hạn ở một khóa tại một thời điểm, không có cách nào để hoạt động trên nhiều khóa tại cùng một lúc. Nếu cần thao tác trên nhiều khóa, phải xử lý việc này từ phía máy khách.

* + 1. **Document databases**

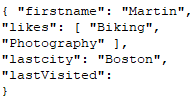
Document là khái niệm chính trong document database. Cơ sở dữ liệu lưu trữ và lấy tài liệu, có thể là XML, JSON, BSON... Các document này là tự mô tả, cây phân cấp cấu trúc dữ liệu có thể bao gồm các bản đồ, bộ sưu tập và giá trị vô hướng. Các document được lưu trữ là tương tự nhau nhưng không nhất thiết phải giống nhau. Cơ sở dữ liệu lưu trữ tài liệu trong phần giá trị của kho lưu trữ key-value. Nghĩ về cơ sở dữ liệu tài liệu như các key-value trong đó giá trị là có thể kiểm tra.

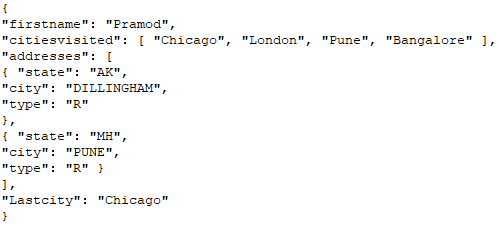
Các thuật ngữ so sánh trong Oracle và MongoDB.

|  |  |
| --- | --- |
| Oracle | MongoDB |
| Database íntance | MongoDB íntance |
| Schema | Database |
| Table | Collection |
| Row | Document |
| Rowid | \_id |
| Join | DBRef |

\_id là một trường đặc biệt được tìm thấy trên tất cả các tài liệu trong MongoDB, giống như ROWID trong Oracle. Trong MongoDB, \_id có thể được chỉ định bởi người dùng, nếu \_id là duy nhất.

* **Khái niệm Document Database**

Ví dụ về một Document Database trong RDBMS truyền thống:

Một ví dụ khác về Document Database:

Nhìn vào các Document, có thể thấy rằng chúng giống nhau, nhưng có sự khác biệt về tên thuộc tính. Điều này được cho phép trong Document Database. Lược đồ của dữ liệu có thể khác nhau giữa các tài liệu, nhưng các tài liệu vẫn có thể thuộc về cùng một bộ sưu tập, không giống như RDBMS nơi mỗi hàng trong bảng phải cùng một lược đồ. Danh sách các citiesvisited được coi là một mảng hoặc một danh sách các addresses như danh sách các tài liệu được nhúng bên trong tài liệu chính. Nhúng tài liệu con làm tiểu dự án, tài liệu bên trong cung cấp cho truy cập dễ dàng và hiệu suất tốt hơn.

Nếu nhìn vào các document, sẽ thấy rằng một số thuộc tính tương tự nhau, chẳng hạn như firstnamehoặc city. Đồng thời, có các thuộc tính trong tài liệu thứ hai không tồn tại trong tài liệu đầu tiên, chẳng hạn như addresses, trong khi likes nằm trong tài liệu đầu tiên nhưng không phải là tài liệu thứ hai.

Cách biểu diễn dữ liệu khác nhau này không giống như trong RDBMS nơi mỗi cột phải có được định nghĩa và nếu nó không có dữ liệu thì nó được đánh dấu là trống hoặc được đặt thành null. Trong tài liệu, không có thuộc tính trống, nếu không tìm thấy thuộc tính đã cho, sẽ giả sử rằng nó không được đặt hoặc không liên quan đến tài liệu. Tài liệu cho phép các thuộc tính mới được tạo mà không cần xác định chúng hoặc có thể thay đổi các tài liệu hiện có.

Một số cơ sở dữ liệu tài liệu phổ biến là MongoDB, CouchDB, Terrastore, OrientDB, RavenDB và nổi tiếng hơn là Lotus Notes sử dụng lưu trữ tài liệu.

* **Tính năng, đặc điểm**

Mặc dù có nhiều cơ sở dữ liệu tài liệu mỗi sản phẩm có một số tính năng có thể không được tìm thấy trong các sản phẩm khác nên sẽ sử dụng MongoDB làm đại diện cho bộ tính năng này. Mỗi phiên bản MongoDB có nhiều cơ sở dữ liệu và mỗi cơ sở dữ liệu có thể có nhiều bộ sưu tập. Khi so sánh với RDBMS, một RDBMS giống như cá thể MongoDB, các lược đồ trong RDBMS tương tự như MongoDB cơ sở dữ liệu và các bảng RDBMS là các bộ sưu tập trong MongoDB. Khi lưu trữ một tài liệu có thể chọn cơ sở dữ liệu và bộ sưu tập.

* **Tính nhất quán:**

Tính nhất quán trong cơ sở dữ liệu MongoDB được cấu hình bằng cách sử dụng các bộ bản sao và chọn chờ ghi để được sao chép cho tất cả hoặc một số kế thừa nhất định. Mỗi lần ghi có thể chỉ định số lượng máy chủ mà ghi phải được truyền tới trước khi nó trở lại thành công. Một lệnh như db.runCommand({ getlasterror : 1 , w : "majority" }) cho biết cơ sở dữ liệu mạnh như thế nào với sự nhất quán mong muốn. Ví dụ: với một máy chủ, chỉ định w là majority, ghi sẽ trở lại ngay lập tức vì chỉ có một nút. Nếu bạn có ba nút trong bộ bản sao và chỉ định w là majority, việc ghi sẽ phải hoàn thành tối thiểu hai nút trước khi nó được báo cáo là thành công. Hoàn toàn có thể tăng giá trị w để có tính nhất quán mạnh hơn nhưng sẽ gây ảnh hưởng bởi hiệu suất ghi, vì bây giờ việc ghi phải hoàn thành ở nhiều nút hơn. Bộ bản sao cũng cho phép tăng hiệu suất đọc bằng cách cho phép đọc từ bản kế thừa bằng cách thiết lập slaveOk. Tham số này có thể được đặt trên kết nối, hoặc cơ sở dữ liệu hoặc bộ sưu tập, hoặc cá nhân cho mỗi hoạt động.

Mongo mongo = new Mongo("localhost:27017");

mongo.slaveOk();

Thiết lập SlaveOk cho mỗi hoạt động, để có thể quyết định thao tác nào có thể hoạt động với dữ liệu từ nút kế thừa.

DBCollection collection = getOrderCollection(); BasicDBObject query = new BasicDBObject(); query.put("name", "Martin");

DBCursor cursor = collection.find(query).slaveOk();

Tương tự như các tùy chọn khác nhau có sẵn để đọc, có thể thay đổi cài đặt để đạt được tính nhất quán ghi mạnh nếu muốn. Theo mặc định, một lượt ghi được báo cáo thành công khi cơ sở dữ liệu nhận được nó và có thể thay đổi điều này để chờ ghi được đồng bộ hóa vào đĩa hoặc truyền sang hai hoặc nhiều bản kế thừa. Điều này được gọi là WriteConcern: để đảm bảo rằng một số lượt ghi nhất định được ghi cho bản gốc và một số kế thừa bằng cách đặt WriteConcern thành REPLICAS\_SAFE. Dưới đây là mã thiết lập WriteConcern cho tất cả ghi vào bộ sưu tập:

DBCollection shopping = database.getCollection("shopping"); shopping.setWriteConcern(REPLICAS\_SAFE);

WriteConcern được đặt cho mỗi thao tác bằng cách chỉ định nó trên lệnh lưu:

WriteResult result = shopping.insert(order, REPLICAS\_SAFE);

* **Tính năng giao dịch**

Các giao dịch theo RDBMS truyền thống, có nghĩa là có thể bắt đầu sửa đổi cơ sở dữ liệu bằng các insert, update hoặc delete trên các bảng khác nhau và sau đó quyết định có muốn giữ các thay đổi hay không bằng cách sử dụng commit hoặc rollback. Các cấu trúc này thường không có sẵn trong các giải pháp NoQuery. Giao dịch ở cấp độ tài liệu đơn được gọi là giao dịch nguyên tử. Các giao dịch liên quan đến nhiều hơn một hoạt động là không thể, mặc dù có những sản phẩm như RavenDB thực hiện các giao dịch hỗ trợ trên nhiều hoạt động.

Kiểm soát tốt hơn đối với mỗi lượt ghi có thể đạt được bằng cách sử dụng tham số WriteConcern. Đảm bảo order được ghi vào nhiều hơn một nút trước khi được báo thành công bằng cách sử dụng WriteConcern.REPLICAS\_SAFE.

Các cấp độ khác nhau của WriteConcern cho phép chọn mức an toàn trong khi ghi. Ví dụ: khi viết các mục nhật ký, có thể sử dụng mức an toàn thấp nhất

WriteConcern.NONE.

final Mongo mongo = new Mongo(mongoURI); mongo.setWriteConcern(REPLICAS\_SAFE);

DBCollection shopping = mongo.getDB(orderDatabase).getCollection(shoppingCollection);

try {

WriteResult result = shopping.insert(order, REPLICAS\_SAFE);

//Writes made it to primary and at least one secondary

} catch (MongoException writeException) {

//Writes did not make it to minimum of two nodes including primary dealWithWriteFailure(order, writeException);

}

* **Tính năng khả dụng**

Định lý CAP chỉ ra rằng chúng ta chỉ có thể có hai tính nhất quán là tính khả dụng và Partition Tolerance. Cơ sở dữ liệu tài liệu cố gắng cải thiện tính khả dụng bằng cách sao chép dữ liệu bằng cách sử dụng cài đặt master-slave. Dữ liệu tương tự có sẵn trên nhiều nút và khách hàng có thể lấy dữ liệu ngay cả khi nút chính bị hỏng. Thông thường, mã ứng dụng không phải xác định xem nút chính có khả dụng hay không. MongoDB thực hiện sao chép, cung cấp tính sẵn sàng cao bằng cách sử dụng các bộ bản sao.

Trong một bộ bản sao, có hai hoặc nhiều nút tham gia vào một bản sao master-slave không đồng bộ. Các nút tự thiết lập bản sao chọn ra nút chính trong tất cả các nút. Giả sử tất cả các nút có quyền biểu quyết như nhau, một số nút có thể được ưu tiên lựa chọn vì gần các máy chủ khác hơn, vì có nhiều RAM hơn... Người dùng có thể tác động đến điều này bằng cách gán mức ưu tiên cho một số từ 0 đến 1000 vào một nút.

Tất cả các yêu cầu đi đến nút chính và dữ liệu được sao chép vào các nút slave. Nếu nút chính bị hỏng, các nút còn lại trong bản sao sẽ tự bỏ phiếu để chọn một nút chính mới. Tất cả các yêu cầu trong tương lai được chuyển đến nút chính mới và các nút slave bắt đầu nhận dữ liệu từ nút chính mới. Khi nút chính hỏng trở lại trực tuyến, nó tham gia như một nút slave và bắt kịp với các nút còn lại bằng cách kéo tất cả dữ liệu cần thiết để có được dòng điện.

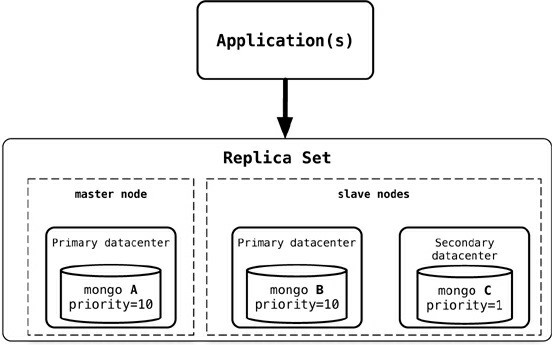


Figure: cấu hình thiết lập bản sao với mức ưu tiên cao hơn được gán cho các nút trong cùng một trung tâm dữ liệu

Xét một cấu hình ví dụ của các bộ bản sao. Hai nút mongo A và mongo B, chạy cơ sở dữ liệu MongoDB trong trung tâm dữ liệu chính và mongo C trong trung tâm dữ liệu thứ cấp. Nếu chúng ta muốn các nút trong trung tâm dữ liệu chính được chọn làm các nút chính, chúng ta có thể gán cho chúng mức ưu tiên cao hơn các nút khác. Nhiều nút có thể được thêm vào các bộ bản sao mà không cần phải đưa chúng ngoại tuyến.

Ứng dụng ghi hoặc đọc từ nút chính. Khi kết nối được thiết lập, ứng dụng chỉ cần kết nối với một nút (không quan trọng có phải là nút chính hay không) trong bộ bản sao và phần còn lại của các nút được phát hiện tự động. Khi nút chính bị hỏng, trình điều khiển sẽ kết nối với nút chính mới được chọn bởi bộ bản sao. Ứng dụng không phải quản lý bất kỳ lỗi kết nối hoặc tiêu chí lựa chọn nút nào. Sử dụng bộ bản sao có khả năng cung cấp một kho lưu trữ dữ liệu tài liệu có sẵn cao.

Các bộ bản sao thường được sử dụng để dự phòng dữ liệu, chuyển đổi dự phòng tự động, đọc tỷ lệ, bảo trì máy chủ mà không có thời gian chết và khắc phục thảm họa. Các thiết lập khả dụng tương tự có thể đạt được với CouchDB, RavenDB, Terrastore và các sản phẩm khác.

* **Tính truy vấn**

Document Database cung cấp các tính năng truy vấn khác nhau. CouchDB cho phép truy vấn thông qua các chế độ xem. Với CouchDB, nếu cần tổng hợp số lượng đánh giá cho một sản phẩm cũng như xếp hạng trung bình, có thể thêm chế độ xem được thực hiện thông qua Materialized Views để trả về số lượng đánh giá và trung bình xếp hạng.

Một trong những tính năng tốt của cơ sở dữ liệu tài liệu, so với các kho lưu trữ key-value, là chúng ta có thể truy vấn dữ liệu bên trong tài liệu mà không phải truy xuất toàn bộ tài liệu bằng khóa của nó và sau đó xem xét tài liệu. Tính năng này mang các cơ sở dữ liệu này đến gần hơn với mô hình truy vấn RDBMS.

MongoDB có ngôn ngữ truy vấn được thể hiện thông qua JSON và có các cấu trúc như $query cho mệnh đề where, $orderby để sắp xếp dữ liệu hoặc $explain để hiển thị kế hoạch thực hiện của truy vấn. Có nhiều cấu trúc như thế này có thể được kết hợp để tạo truy vấn MongoDB.

* **Chia tỷ lệ**

Ý tưởng của việc chia tỷ lệ là thêm các nút hoặc thay đổi lưu trữ dữ liệu mà không cần di chuyển cơ sở dữ liệu sang một hộp lớn hơn. Không phải về việc thực hiện các thay đổi ứng dụng để xử lý tải nhiều hơn thay vào đó quan tâm đến các tính năng trong cơ sở dữ liệu để nó có thể xử lý nhiều tải hơn.

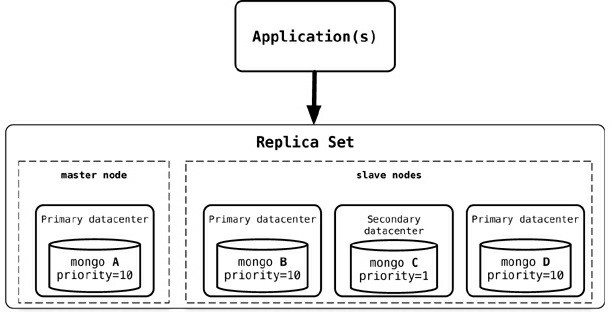
Mở rộng quy mô cho tải nặng có thể đạt được bằng cách thêm nhiều slave đọc hơn, để tất cả các lần đọc có thể được hướng đến tất cả slave. Với một ứng dụng đọc nhiều, với cụm sao chép 3 nút, chúng ta có thể tăng thêm dung lượng đọc cho cụm khi tải đọc tăng chỉ bằng cách thêm nhiều nút slave vào tập sao chép để thực hiện đọc bằng cờ Slave. Đây là tỷ lệ ngang cho đọc.

Figure: Thêm một nút mongo D vào cụm sao chép được thiết lập hiện có

Khi nút mongo D được bắt đầu, nó cần được thêm vào bộ bản sao.

rs.add("mongod:27017");

Khi một nút mới được thêm vào, nó sẽ đồng bộ hóa với các nút hiện có, tham gia bộ bản sao làm nút phụ và bắt đầu phục vụ các yêu cầu đọc. Một lợi thế của thiết lập này là không phải khởi động lại bất kỳ nút nào khác và cũng không có thời gian chết cho ứng dụng. Khi muốn mở rộng quy mô cho việc ghi, có thể bắt đầu shending. Shending tương tự như các phân vùng trong RDBMS nơi chia dữ liệu theo giá trị trong một cột nhất định, chẳng hạn như State hoặc Year. Với RDBMS, các phân vùng thường nằm trên cùng một nút, vì vậy ứng dụng khách không phải truy vấn một phân vùng cụ thể mà có thể tiếp tục truy vấn bảng cơ sở RDBMS đảm nhiệm việc tìm phân vùng phù hợp cho truy vấn và trả về dữ liệu. Trong shending, dữ liệu cũng được phân chia theo trường nhất định, nhưng sau đó được chuyển đến các nút Mongo khác nhau. Dữ liệu được di chuyển linh hoạt giữa các nút để đảm bảo rằng các phân đoạn luôn được cân bằng. Chúng ta có thể thêm nhiều nút vào cụm và tăng số lượng nút có thể ghi, cho phép chia tỷ lệ theo chiều ngang để ghi.

db.runCommand( { shardcollection : "ecommerce.customer",

key : {firstname : 1} } )

Việc phân tách dữ liệu trên tên của khách hàng đảm bảo rằng dữ liệu được cân bằng giữa các phân đoạn để có hiệu suất ghi tối ưu. Hơn nữa, mỗi phân đoạn có thể là một bộ bản sao đảm bảo hiệu suất đọc tốt hơn trong phân đoạn. Khi chúng ta thêm một phân đoạn mới vào cụm phân đoạn hiện có này, dữ liệu sẽ được cân bằng giữa bốn phân đoạn thay vì ba phân đoạn. Vì tất cả các chuyển động dữ liệu và tái cấu trúc cơ sở hạ tầng này đang diễn ra, ứng dụng sẽ không gặp phải bất kỳ thời gian chết nào, mặc dù cụm có thể không hoạt động tối ưu khi một lượng lớn dữ liệu đang được di chuyển để cân bằng lại các phân đoạn.

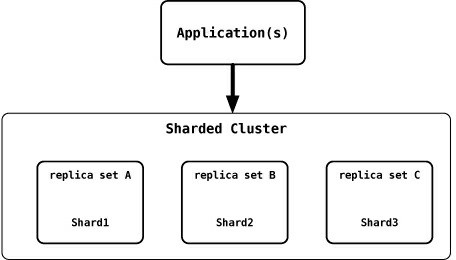


Figure: Phân chia MongoDB trong đó mỗi phân đoạn là một bộ bản sao

* **Các trường hợp nên sử dụng**
* **Ghi nhật ký sự kiện:**

Các ứng dụng có nhu cầu ghi nhật ký sự kiện khác nhau. Trong doanh nghiệp có nhiều ứng dụng khác nhau muốn ghi nhật ký sự kiện. Cơ sở dữ liệu tài liệu có thể lưu trữ tất cả các loại sự kiện khác nhau và có thể hoạt động như một kho lưu trữ dữ liệu trung tâm để lưu trữ sự kiện. Điều này đặc biệt đúng khi loại dữ liệu bị bắt bởi các sự kiện liên tục thay đổi. Các sự kiện có thể được loại bỏ theo tên của ứng dụng nơi sự kiện bắt nguồn hoặc theo loại sự kiện như order\_processed hoặc customer\_logged.

* **Hệ thống quản lý nội dung, nền tảng blog**

Vì Document Database không có lược đồ được xác định trước và thường là tài liệu JSON, chúng hoạt động tốt trong các hệ thống quản lý nội dung hoặc ứng dụng để xuất bản trang web, quản lý nhận xét người dùng, đăng ký người dùng, hồ sơ, tài liệu đối mặt với web.

* **Phân tích trang web hoặc Phân tích thời gian thực**

Cơ sở dữ liệu tài liệu có thể lưu trữ dữ liệu cho các phân tích thời gian thực. Do các phần của tài liệu có thể được cập nhật, nên nó rất dễ dàng để lưu trữ lượt xem trang hoặc khách truy cập duy nhất và có thể dễ dàng thêm các số liệu mới mà không cần thay đổi lược đồ.

* **Ứng dụng thương mại điện tử**

Các ứng dụng thương mại điện tử thường cần có sơ đồ linh hoạt cho các sản phẩm và đơn đặt hàng, cũng như khả năng phát triển các mô hình dữ liệu của chúng mà không cần tái cấu trúc cơ sở dữ liệu đắt tiền hoặc di chuyển dữ liệu.

* **Khi không sử dụng**

Có những không gian vấn đề trong đó cơ sở dữ liệu tài liệu không phải là giải pháp tốt nhất.

* **Giao dịch phức tạp kéo dài các hoạt động khác nhau**

Nếu cần phải có các hoạt động tài liệu chéo nguyên tử, thì cơ sở dữ liệu tài liệu không nên sử dụng. Tuy nhiên, có một số cơ sở dữ liệu tài liệu hỗ trợ các loại hoạt động này chẳng hạn như RavenDB.

* **Truy vấn chống lại cấu trúc tổng hợp khác nhau**

Lược đồ linh hoạt có nghĩa là cơ sở dữ liệu không thực thi bất kỳ hạn chế nào đối với lược đồ. Dữ liệu được lưu dưới dạng các thực thể ứng dụng. Nếu bạn cần truy vấn các thực thể này một cách đặc biệt, các truy vấn của bạn sẽ thay đổi (theo thuật ngữ RDBMS, điều này có nghĩa là khi bạn nối các tiêu chí giữa các bảng, các bảng sẽ tham gia thay đổi). Vì dữ liệu được lưu dưới dạng tổng hợp, nếu thiết kế của tổng hợp thay đổi liên tục, bạn cần lưu các tổng hợp ở mức thấp nhất về mức độ chi tiết, về cơ bản, bạn cần bình thường hóa dữ liệu. Trong trường hợp này, cơ sở dữ liệu tài liệu có thể không hoạt động.

* + 1. **Column family stores**

Column-family database lưu trữ dữ liệu trong một tập hợp các cột, tập hợp này được xác định thông qua một khoá hàng (row key). Các cột trong cùng một hàng lưu trữ các dữ liệu liên quan và thường được truy xuất cùng nhau.



Mỗi column-family có thể coi như một hàng trong cơ sở dữ liệu quan hệ nơi một khóa xác định một hàng và mỗi hàng chứa nhiều cột. Tuy nhiên điểm đặc biệt của CSDL kiểu này đó là mỗi hàng không nhất thiết phải chứa các cột giống nhau, và tại bất kì thời điểm nào ta có thể thêm một cột vào một hàng bất kì mà không phải thêm nó vào các hàng còn lại.

* **Column-family là gì?**

Có nhiều cơ sở dữ liệu column-family, chúng ta sẽ nói về Cassandra nhưng cũng tham khảo các cơ sở dữ liệu column-family khác để thảo luận về các tính năng có thể được quan tâm trong các tình huống cụ thể.Cơ sở dữ liệu column-family lưu trữ dữ liệu trong các nhóm cột dưới dạng các hàng có nhiều cột được liên kết với khóa hàng. Nhóm cột là nhóm các dữ liệu liên quan thường được truy cập cùng nhau. Đối với khách hàng, thường truy cập thông tin hồ sơ cùng một lúc, nhưng không phải Đơn đặt hàng.

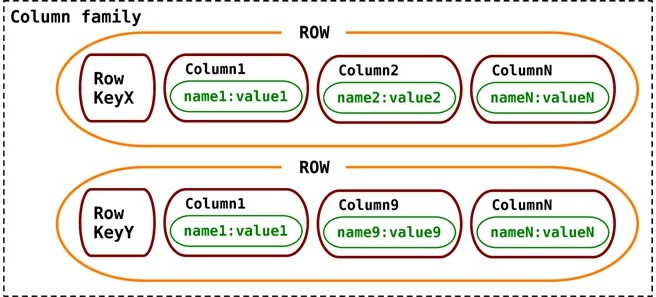


Figure: Mô hình dữ liệu Cassandra với các nhóm cột

* **Tính năng, đặc điểm**

Hãy bắt đầu bằng cách xem xét cách dữ liệu được cấu trúc trong Cassandra. Đơn vị lưu trữ cơ bản trong Cassandra là một cột. Một cột Cassandra bao gồm một cặp tên-giá trị trong đó tên cũng hoạt động như là khóa. Mỗi cặp key-value này là một cột duy nhất và luôn được lưu trữ với giá trị dấu thời gian. Dấu thời gian được sử dụng để hết hạn dữ liệu, giải quyết xung đột ghi, xử lý dữ liệu cũ và làm những việc khác.

Khi dữ liệu cột không còn được sử dụng, không gian có thể được lấy lại sau khi giai đoạn nén hoàn thành.

{

name: "fullName",

value: "Martin Fowler",

timestamp: 12345667890

}

Cột có khóa là FirstName và giá trị của Martin và có dấu thời gian được đính kèm. Một hàng là tập hợp các cột được gắn hoặc liên kết với khóa. Một tập hợp các hàng tương tự làm cho một nhóm cột. Khi các cột trong một nhóm cột là các cột đơn giản, nhóm cột được gọi là nhóm cột tiêu chuẩn.

//column family

{

//row

"pramod-sadalage" : {

firstName: "Pramod",

lastName: "Sadalage",

lastVisit: "2012/12/12"

}

//row

"martin-fowler" : {

firstName: "Martin",

lastName: "Fowler",

location: "Boston"

}

}

Mỗi nhóm cột có thể được so sánh với một thùng chứa các hàng trong bảng RDBMS trong đó khóa xác định hàng và hàng bao gồm nhiều cột. Sự khác biệt là các hàng khác nhau không phải có cùng một cột và các cột có thể được thêm vào bất kỳ hàng nào vào bất kỳ lúc nào mà không cần phải thêm nó vào các hàng khác. Chúng ta có hàng pramod-sadalage và hàng martin-fowler với các cột khác nhau, cả hai hàng là một phần của nhóm cột.

Khi một cột bao gồm một bản đồ các cột, thì chúng ta có một siêu cột. Một siêu cột bao gồm một tên và một giá trị là bản đồ của các cột.

{

name: "book:978-0767905923",

value: {

author: "Mitch Albon",

title: "Tuesdays with Morrie",

isbn: "978-0767905923"

}

}

Khi chúng ta sử dụng các siêu cột để tạo một nhóm cột, chúng ta sẽ có một nhóm các siêu cột.

//super column family

{

//row

name: "billing:martin-fowler",

value: {

address: {

name: "address:default",

value: {

fullName: "Martin Fowler",

street:"100 N. Main Street",

zip: "20145"

}

},

billing: {

name: "billing:default",

value: {

creditcard: "8888-8888-8888-8888",

expDate: "12/2016"

}

}

}

//row

name: "billing:pramod-sadalage",

value: {

address: {

name: "address:default",

value: {

fullName: "Pramod Sadalage",

street:"100 E. State Parkway",

zip: "54130"

}

},

billing: {

name: "billing:default",

value: {

creditcard: "9999-8888-7777-4444",

expDate: "01/2016"

}

}

}

Các nhóm siêu cột rất tốt để giữ các dữ liệu liên quan với nhau, nhưng khi một số cột không cần thiết hầu hết thời gian, các cột vẫn được Cassandra tìm nạp và giải nén, có thể không tối ưu. Cassandra đặt các nhóm cột tiêu chuẩn và siêu lớn vào không gian khóa. Một không gian khóa tương tự như một cơ sở dữ liệu trong RDBMS nơi tất cả các nhóm cột liên quan đến ứng dụng được lưu trữ. Không gian khóa phải được tạo để các nhóm cột có thể được gán cho chúng:

create keyspace ecommerce

* **Tính nhất quán**

Khi Cassandra nhận được một bản ghi, dữ liệu được ghi lại đầu tiên trong nhật ký cam kết, sau đó được ghi vào cấu trúc trong bộ nhớ được gọi là memtable. Một thao tác ghi được coi là thành công khi nó được ghi vào nhật ký cam kết và memtable. Các bài viết được tập hợp trong bộ nhớ và được ghi định kỳ vào các cấu trúc được gọi là SSTable. SSTables không được viết lại sau khi chúng bị xóa. Nếu có thay đổi dữ liệu, SSTable mới sẽ được viết. SSTables không sử dụng được thu hồi bằng cách nén.

Nếu cài đặt nhất quán là ONE làm mặc định cho tất cả các hoạt động đọc, thì khi yêu cầu đọc được thực hiện, Cassandra trả về dữ liệu từ bản sao đầu tiên, ngay cả khi dữ liệu đã cũ. Nếu dữ liệu cũ, các lần đọc tiếp theo sẽ nhận được dữ liệu mới nhất (mới nhất). Quá trình này được gọi là sửa chữa đọc. Mức độ nhất quán thấp rất tốt để sử dụng khi bạn không quan tâm nếu bạn nhận được dữ liệu cũ and/or nếu bạn có yêu cầu hiệu suất đọc cao.

Tương tự, nếu đang thực hiện ghi, Cassandra sẽ ghi vào một nút nhật ký cam kết và trả phản hồi về cho máy khách. Tính nhất quán của ONE là tốt nếu có yêu cầu cần hiệu suất ghi rất cao và cũng không phiền nếu một số ghi bị mất, điều này có thể xảy ra nếu nút bị hỏng trước khi ghi được sao chép sang các nút khác.

quorum = new ConfigurableConsistencyLevel(); quorum.setDefaultReadConsistencyLevel(HConsistencyLevel.QUORUM); quorum.setDefaultWriteConsistencyLevel(HConsistencyLevel.QUORUM);

Sử dụng cài đặt tính nhất quán QUORUM cho cả hoạt động đọc và ghi đảm bảo rằng phần lớn các nút phản hồi lại việc đọc và cột có dấu thời gian mới nhất được trả về cho máy khách, trong khi các bản sao không có dữ liệu mới nhất được sửa chữa thông qua việc đọc hoạt động sửa chữa. Trong các hoạt động ghi, cài đặt tính nhất quán QUORUM có nghĩa là ghi phải truyền đến phần lớn các nút trước khi nó được coi là thành công và máy khách được thông báo.

Sử dụng ALL làm mức độ nhất quán có nghĩa là tất cả các nút sẽ phải phản hồi để đọc hoặc ghi, điều này sẽ làm cho cụm không chịu được các lỗi ngay cả khi một nút bị hỏng, việc ghi hoặc đọc bị chặn và báo cáo là lỗi. Do đó, các nhà thiết kế hệ thống điều chỉnh các mức độ nhất quán khi các yêu cầu ứng dụng thay đổi. Trong cùng một ứng dụng, có thể có các yêu cầu khác nhau về tính nhất quán, cũng có thể thay đổi dựa trên từng thao tác, ví dụ hiển thị nhận xét đánh giá cho sản phẩm có các yêu cầu về tính nhất quán khác nhau so với việc đọc trạng thái của đơn hàng cuối cùng được đặt bởi khách hàng.

Trong quá trình tạo không gian phím, có thể định cấu hình bao nhiêu bản sao dữ liệu mà chúng ta cần lưu trữ. Con số này xác định hệ số sao chép của dữ liệu. Nếu có hệ số sao chép là 3 thì dữ liệu được sao chép vào ba nút. Khi viết và đọc dữ liệu với Cassandra, nếu chỉ định các giá trị nhất quán là 2, bạn sẽ thấy R + W lớn hơn hệ số sao chép (2 + 2> 3) mang lại tính nhất quán tốt hơn trong quá trình viết và đọc.

Chúng ta có thể chạy lệnh sửa chữa nút cho không gian phím và buộc Cassandra so sánh mọi khóa mà nó chịu trách nhiệm với phần còn lại của bản sao. Có thể sửa chữa một nhóm cột cụ thể hoặc một danh sách các nhóm cột:

repair ecommerce

repair ecommerce customerInfo

Trong khi một nút bị hỏng, dữ liệu được cho là được lưu trữ bởi nút đó sẽ được chuyển đến các nút khác. Khi nút quay trở lại trực tuyến, các thay đổi được thực hiện cho dữ liệu được đưa trở lại nút. Kỹ thuật này được gọi là bàn giao gợi ý. Handoffed cho phép khôi phục nhanh hơn các nút bị lỗi.

* **Tính Giao dịch**

Cassandra không có giao dịch theo nghĩa truyền thống, nơi có thể bắt đầu viết nhiều lần và sau đó quyết định xem có muốn thực hiện các thay đổi hay không. Trong Cassandra, một bản ghi là nguyên tử ở cấp hàng, có nghĩa là chèn hoặc cập nhật các cột cho một khóa hàng nhất định sẽ được coi là một lần ghi và sẽ thành công hoặc thất bại. Viết trước tiên được viết để cam kết nhật ký và ghi nhớ, và chỉ được coi là tốt khi ghi để ghi nhật ký và ghi nhớ đã thành công. Nếu một nút bị hỏng, nhật ký cam kết được sử dụng để áp dụng các thay đổi cho nút, giống như nhật ký làm lại trong Oracle.

Có thể sử dụng các thư viện giao dịch bên ngoài, chẳng hạn như ZooKeeper để đồng bộ hóa việc ghi và đọc. Ngoài ra còn có các thư viện như Cages cho phép thực hiện các giao dịch qua ZooKeeper.

* **Tính khả dụng**

Cassandra theo thiết kế có tính sẵn sàng cao, vì không có chủ trong cụm và mỗi nút là một đồng đẳng trong cụm. Tính khả dụng của một cụm có thể được tăng lên bằng cách giảm mức độ nhất quán của các yêu cầu. Tính khả dụng được điều chỉnh bởi công thức (R + W)> N trong đó W là số nút tối thiểu mà ghi phải được ghi thành công, R là số nút tối thiểu phải đáp ứng thành công a đọc và N là số lượng nút tham gia sao chép dữ liệu. Có thể điều chỉnh tính khả dụng bằng cách thay đổi giá trị R và W cho giá trị cố định là N.

Trong cụm Cassandra 10 nút có hệ số sao chép cho không gian khóa được đặt thành 3 (N = 3), nếu đặt R = 2 và W = 2, thì có (2 + 2)> 3. Trong trường hợp này khi một nút đi xuống tính khả dụng không bị ảnh hưởng nhiều, vì dữ liệu có thể được lấy từ hai nút còn lại. Nếu W = 2 và R = 1, khi hai nút cùng xuống thì cụm không có sẵn để ghi nhưng vẫn cho phép đọc.

Tương tự, nếu R = 2 và W = 1, có thể viết nhưng cụm không có sẵn để đọc. Với R + W > N nên đưa ra quyết định có nên thay đổi tính nhất quán. Nên thiết lập các không gian khóa và các thao tác đọc / ghi dựa trên nhu cầu, tính sẵn sàng cao hơn để ghi hoặc sẵn sàng cao hơn để đọc.

* **Tính năng truy vấn**

Khi thiết kế mô hình dữ liệu trong Cassandra, nên làm cho các cột và nhóm cột được tối ưu hóa để đọc dữ liệu, vì nó không có ngôn ngữ truy vấn phong phú. Khi dữ liệu được chèn trong các nhóm cột và dữ liệu trong mỗi hàng được sắp xếp theo tên cột, nếu có một cột được truy xuất thường xuyên hơn nhiều so với các cột khác thì điều đó có hiệu quả tốt hơn khi sử dụng giá trị đó cho khóa hàng thay thế.

**+ Truy vấn cơ bản:**

Các truy vấn cơ bản có thể được chạy bằng ứng dụng khách Cassandra bao gồm GET, SET và DEL. Trước khi bắt đầu truy vấn dữ liệu, khởi tạo lệnh use ecommerce;. Để đảm bảo tất cả các truy vấn được chạy trên không gian khóa khi đưa dữ liệu vào. Trước khi bắt đầu sử dụng nhóm cột trong không gian khóa thì phải xác định nhóm cột.

CREATE COLUMN FAMILY Customer

WITH comparator = UTF8Type

AND key\_validation\_class=UTF8Type

AND column\_metadata = [

{column\_name: city, validation\_class: UTF8Type}

{column\_name: name, validation\_class: UTF8Type}

{column\_name: web, validation\_class: UTF8Type}

];

Xét một nhóm cột có tên là Customer, name,city, web và chèn dữ liệu trong nhóm cột với một máy khách Cassandra.

SET Customer['mfowler']['city']='Boston';

SET Customer['mfowler']['name']='Martin Fowler';

SET Customer['mfowler']['web']='www.martinfowler.com';

Sử dụng Java Hector trên máy khách, có thể chèn cùng một dữ liệu trong nhóm cột.

ColumnFamilyTemplate<String, String> template = cassandra.getColumnFamilyTemplate();

ColumnFamilyUpdater<String, String>updater= template.createUpdater(key);

for (String name :values.keySet()) { updater.setString(name, values.get(name));

}

try {

template.update(updater);

} catch (HectorEception e) {

handleException(e);

}

Có thể đọc lại dữ liệu bằng lệnh GET. Có nhiều cách để lấy dữ liệu như lấy cả column family.

GET Customer['mfowler']['web'];

Chọn một cột cụ thể để truy vấn sẽ hiệu quả hơn, vì chỉ có dữ liệu quan tâm được trả về, giúp tiết kiệm rất nhiều chuyển động dữ liệu, đặc biệt là khi nhóm cột có số lượng cột lớn. Cập nhật dữ liệu giống như sử dụng lệnh SET cho cột cần được đặt thành giá trị mới. Sử dụng lệnh DEL, chúng ta có thể xóa một cột hoặc toàn bộ nhóm cột.

DEL Customer['mfowler']['city'];

DEL Customer['mfowler'];

+ **Truy vấn nâng cao và Indexing**

Cassandra cho phép lập chỉ mục các cột khác với các khóa cho nhóm cột. Có thể định nghĩa một chỉ mục trên cột city.

UPDATE COLUMN FAMILY Customer

WITH comparator = UTF8Type

AND column\_metadata = [{column\_name: city,

validation\_class: UTF8Type,

index\_type: KEYS}];

Bây giờ có thể truy vấn trực tiếp vào cột được lập chỉ mục.

GET Customer WHERE city = 'Boston';

Các chỉ mục này được triển khai dưới dạng các chỉ mục được ánh xạ bit và hoạt động tốt đối với các giá trị cột có số lượng thẻ thấp.

**+ Ngôn ngữ truy vấn Cassandra (CQL)**

Cassandra có ngôn ngữ truy vấn hỗ trợ các lệnh giống như SQL, được gọi là Ngôn ngữ truy vấn Cassandra (CQL). Chúng ta có thể sử dụng các lệnh CQL để tạo một nhóm cột.

CREATE COLUMNFAMILY Customer (

KEY varchar PRIMARY KEY,

name varchar,

city varchar,

web varchar);

Chèn cùng một dữ liệu bằng CQL.

INSERT INTO Customer (KEY,name,city,web)

VALUES ('mfowler',

'Martin Fowler',

'Boston',

'www.martinfowler.com');

Có thể đọc dữ liệu bằng lệnh SELECT. Đọc trên tất cả các cột:

SELECT \* FROM Customer

Hoặc có thể SELECT từng cột:

SELECT name,web FROM Customer

Các cột lập chỉ mục được tạo bằng lệnh CREATE INDEX và sau đó có thể được sử dụng để truy vấn dữ liệu.

SELECT name,web FROM Customer WHERE city='Boston'

CQL có nhiều tính năng hơn để truy vấn dữ liệu, nhưng nó không có tất cả các tính năng mà SQL có. CQL không cho phép tham gia vào các truy vấn phức tạp và điều khoản của nó thường đơn giản.

* **Các trường hợp sử dụng phù hợp**
* **Ghi nhật ký sự kiện**

****Cơ sở dữ liệu Column-family với khả năng lưu trữ bất kỳ cấu trúc dữ liệu nào là một lựa chọn tuyệt vời để lưu trữ thông tin sự kiện, chẳng hạn như trạng thái ứng dụng hoặc lỗi mà ứng dụng gặp phải. Trong doanh nghiệp tất cả các ứng dụng có thể viết các sự kiện của chúng cho Cassandra bằng các cột riêng và mã hàng appname:timestamp. Vì có thể ghi lại tỷ lệ ghi, Cassandra sẽ hoạt động lý tưởng cho một hệ thống ghi nhật ký sự kiện.

**Figure : Ghi nhật ký sự kiện với Cassandra**

* **Hệ thống quản lý nội dung, nền tảng blog**

Sử dụng các nhóm cột có thể lưu trữ các mục blog với các thẻ, danh mục, liên kết và trackback trong các cột khác nhau. Nhận xét có thể được lưu trữ trong cùng một hàng hoặc được chuyển đến một không gian phím khác. Tương tự blog người dùng và blog thực tế có thể được đưa vào các nhóm cột khác nhau.

* **Counters**

Thông thường, trong các ứng dụng web, cần đếm và phân loại khách truy cập của một trang để tính toán phân tích. Có thể sử dụng CounterColumnType trong khi tạo một nhóm cột.

CREATE COLUMN FAMILY visit\_counter

WITH default\_validation\_class=CounterColumnType

AND key\_validation\_class=UTF8Type AND comparator=UTF8Type;

Khi một nhóm cột được tạo, có thể có các cột tùy ý cho mỗi trang được truy cập trong ứng dụng web cho mỗi người dùng.

INCR visit\_counter['mfowler'][home] BY 1;

INCR visit\_counter['mfowler'][products] BY 1;

INCR visit\_counter['mfowler'][contactus] BY 1;

Các counters tăng dần sử dụng CQL:

UPDATE visit\_counter SET home = home + 1 WHERE KEY='mfowler'

* **Hết hạn sử dụng**

Có thể cung cấp quyền truy cập demo cho người dùng hoặc có thể muốn hiển thị các biểu ngữ quảng cáo trên một trang web trong một thời gian cụ thể. Bạn có thể làm điều này bằng cách sử dụng các cột hết hạn: Cassandra cho phép bạn thao tác trên các cột sau một thời gian nhất định các cột này sẽ tự động bị xóa. Thời gian này được gọi là TTL (Thời gian sống) và được xác định bằng giây. Cột bị xóa sau khi hết thời gian. Khi cột không tồn tại, quyền truy cập có thể bị thu hồi hoặc banner có thể bị xóa.

* **Các trường hợp không sử dụng**

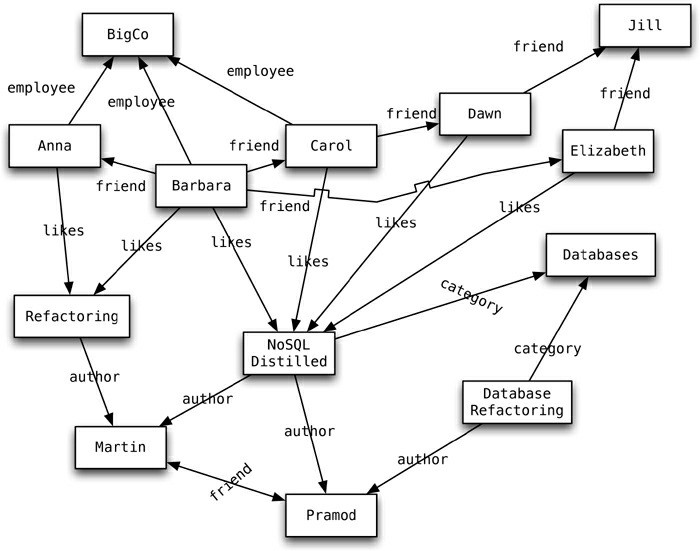
Có những vấn đề mà cơ sở dữ liệu Column-family không phải là giải pháp tốt nhất, chẳng hạn như các hệ thống yêu cầu giao dịch ACID để ghi và đọc. Nếu cần cơ sở dữ liệu để tổng hợp dữ liệu bằng các truy vấn (chẳng hạn như SUM hoặc AVG), phải thực hiện việc này ở phía máy khách bằng cách sử dụng dữ liệu được máy khách truy xuất từ ​​tất cả các hàng. Cassandra không lý tưởng cho các nguyên mẫu ban đầu hoặc các đột biến công nghệ ban đầu. Trong giai đoạn đầu, cách thức các mẫu truy vấn không chắc chắn có thể thay đổi. Nên khi các mẫu truy vấn thay đổi, phải thay đổi thiết kế nhóm cột. Điều này gây ra ma sát cho nhóm đổi mới sản phẩm và làm chậm năng suất của nhà phát triển. RDBMS áp đặt chi phí cao cho thay đổi lược đồ, được đánh đổi bằng chi phí thay đổi truy vấn thấp. Trong Cassandra, chi phí có thể cao hơn cho thay đổi truy vấn so với thay đổi lược đồ.

* + 1. **Graph Databases**

Cơ sở dữ liệu đồ thị cho phép lưu trữ các thực thể và mối quan hệ giữa các thực thể này. Các thực thể còn được gọi là các nút, có các thuộc tính. Một nút như là một ví dụ của một đối tượng trong ứng dụng. Quan hệ được gọi là các cạnh có thể có các thuộc tính. Các cạnh có ý nghĩa định hướng, các nút được tổ chức bởi các mối quan hệ cho phép tìm thấy các mẫu giữa các nút. Việc tổ chức biểu đồ cho phép dữ liệu được lưu trữ một lần và sau đó được diễn giải theo các cách khác nhau dựa trên các mối quan hệ.

* **Graph Database là gì?**

Trong biểu đồ ví dụ dưới đây các nút đều có liên quan với nhau. Các nút là các thực thể có các thuộc tính, chẳng hạn như name. Nút của Martin thực sự là một nút có thuộc tính name được đặt thành Martin.



**Figure: Một ví dụ về cấu trúc đồ thị**

Thấy rằng các cạnh có các loại, chẳng hạn như likes, author... Các thuộc tính này cho phép liên kết các nút. Ví dụ các nút Martin, Pramod có một cạnh kết nối chúng với nhau là mối quan hệ friend. Các cạnh có thể có nhiều thuộc tính. Có thể chỉ định một thuộc tính từ loại quan hệ friend giữa Martin và Pramod. Các loại mối quan hệ có ý nghĩa định hướng, kiểu quan hệ friend là hai chiều nhưng likes thì không.

Khi chúng ta có một biểu đồ gồm các nút và cạnh có thể truy vấn biểu đồ theo nhiều cách, chẳng hạn như nhận được tất cả các nút được sử dụng bởi BigCo, giống như NoSQL Distilled. Một lợi thế của cơ sở dữ liệu đồ thị là có thể thay đổi các yêu cầu di chuyển ngang mà không phải thay đổi các nút hoặc cạnh. Nếu muốn có được tất cả các nút như NoQuery Distilled, thì có thể làm như vậy mà không phải thay đổi dữ liệu hiện có hoặc mô hình của cơ sở dữ liệu, bởi có thể duyệt qua biểu đồ theo bất kỳ cách nào đó theo mong muốn.

Thông thường, khi lưu trữ một cấu trúc giống như đồ thị trong RDBMS, nó sẽ cho một loại mối quan hệ duy nhất. Thêm một mối quan hệ khác thường tạo ra rất nhiều thay đổi lược đồ và làm di chuyển dữ liệu, đó không phải là trường hợp đang sử dụng cơ sở dữ liệu đồ thị. Tương tự, trong các cơ sở dữ liệu quan hệ, khi lập mô hình biểu đồ trước dựa trên Traversal nếu Traversal thay đổi dữ liệu sẽ phải thay đổi.

Trong cơ sở dữ liệu đồ thị, việc duyệt qua các liên kết hoặc các mối quan hệ là rất nhanh. Mối quan hệ giữa các nút không được tính toán tại thời điểm truy vấn nhưng thực sự vẫn tồn tại như một mối quan hệ.

Giữa các nút có thể có các loại mối quan hệ khác nhau, cho phép vừa thể hiện mối quan hệ giữa các thực thể miền vừa thể hiện mối quan hệ thứ cấp ví dụ như thể loại, đường dẫn, cây thời gian... để lập chỉ mục không gian hoặc danh sách được liên kết để truy cập được sắp xếp. Vì không có giới hạn về số lượng và loại mối quan hệ mà một nút có thể có, tất cả đều có thể được biểu diễn trong cùng một cơ sở dữ liệu đồ thị.

* **Tính năng đặc điểm**

Có nhiều cơ sở dữ liệu đồ thị có sẵn, chẳng hạn như Neo4J, Infinite Graph, OrientDB hoặc FlockDB (là trường hợp đặc biệt: cơ sở dữ liệu đồ thị chỉ hỗ trợ các mối quan hệ hoặc độ sâu đơn danh sách, không thể đi sâu hơn một cấp cho các mối quan hệ). Để tìm hiểu kỹ hơn về cơ sở dữ liệu đồ thị sẽ lấy Neo4J làm đại diện cho các giải pháp cơ sở dữ liệu đồ thị để thảo luận về cách chúng hoạt động và cách chúng có thể được sử dụng để giải quyết các vấn đề ứng dụng.

Trong Neo4J, việc tạo một biểu đồ đơn giản như tạo hai nút và sau đó tạo mối quan hệ. Hãy tạo ra hai nút, Martin và Pramod:

Node martin = graphDb.createNode(); martin.setProperty("name", "Martin");

Node pramod = graphDb.createNode(); pramod.setProperty("name", "Pramod");

Gán thuộc tính name của hai nút giá trị của Martin và Pramod. Khi có nhiều hơn một nút thì có thể tạo mối quan hệ cho các nút đó:

martin.createRelationshipTo(pramod, FRIEND); pramod.createRelationshipTo(martin, FRIEND);

Chúng ta phải tạo mối quan hệ giữa các nút theo cả hai hướng của mối quan hệ ví dụ như: user có thể thích một nút product nhưng product không thể thích user. Định hướng này giúp thiết kế một mô hình miền phong phú. Các nút biết về mối quan hệ INCOMING và OUTGOING cả hai hướng.

Mối quan hệ là tốt nhất trong cơ sở dữ liệu đồ thị. Hầu hết giá trị của cơ sở dữ liệu đồ thị có nguồn gốc từ các mối quan hệ. Mối quan hệ không chỉ có một loại tuy chỉ có một nút bắt đầu và một nút kết thúc, nhưng có thể có các thuộc tính của riêng nó. Các thuộc tính trên các mối quan hệ có thể được sử dụng để truy vấn biểu đồ.

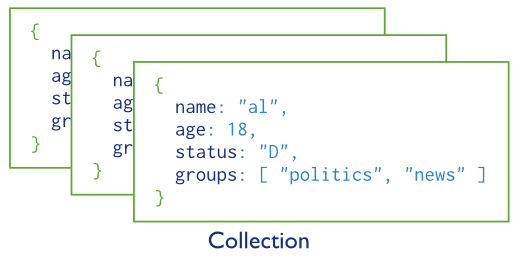
Vì hầu hết sức mạnh từ cơ sở dữ liệu đồ thị đến từ các mối quan hệ và thuộc tính của chúng, nên cần rất nhiều công việc thiết kế và suy nghĩ để mô hình hóa các mối quan hệ trong miền mà chúng tôi đang cố gắng làm việc. Thêm các loại mối quan hệ mới là dễ dàng; thay đổi các nút hiện có và các mối quan hệ của chúng tương tự như di chuyển dữ liệu, vì những thay đổi này sẽ phải được thực hiện trên mỗi nút và từng mối quan hệ trong dữ liệu hiện có.

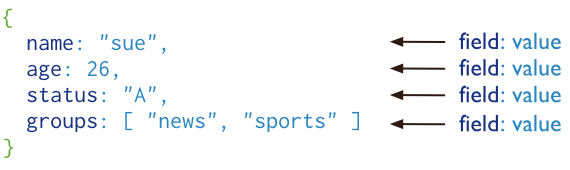
# **Tìm hiểu về MongoDB.**

* + 1. **Khái niệm MongoDB**

MongoDB là một chương trình cơ sở dữ liệu định hướng tài liệu đa nền tảng. Phân loại như một NoSQL chương trình cơ sở dữ liệu, MongoDB sử dụng JSON-like với lược đồ. MongoDB được phát triển bởi MongoDB Inc và được cấp phép theo Giấy phép Công cộng phía Máy chủ (SSPL).

* MongoDB là một cơ sở dữ liệu mã nguồn mở và là cơ sở dữ liệu NoSQL hàng đầu, được hàng triệu người sử dụng. MongoDB được viết bằng C++.
* Ngoài ra, MongoDB hoạt động trên các khái niệm Collection và Document, cung cấp hiệu suất cao, tính khả dụng cao và khả năng mở rộng dễ dàng.
  + 1. **Các thuật ngữ hay sử dụng trong MongoDB**
* **id –** Là trường bắt buộc có trong mỗi document. Trường \_id đại diện cho một giá trị duy nhất trong document MongoDB. Trường \_id cũng có thể được hiểu là khóa chính trong document. Nếu bạn thêm mới một document thì MongoDB sẽ tự động sinh ra một \_id đại diện cho document đó và là duy nhất trong cơ sở dữ liệu MongoDB.
* **Collection**: là nhóm của nhiều document trong MongoDB. Collection có thể được hiểu là một bảng tương ứng trong cơ sở dữ liệu RDBMS (Relational Database Management System). Collection nằm trong một cơ sở dữ liệu duy nhất. Các collection không phải định nghĩa các cột, các hàng hay kiểu dữ liệu trước.



* **Cursor**: đây là một con trỏ đến tập kết quả của một truy vấn. Máy khách có thể lặp qua một con trỏ để lấy kết quả.
* **Database**: nơi chứa các Collection, giống với cơ sở dữ liệu RDMS chúng chứa các bảng. Mỗi Database có một tập tin riêng lưu trữ trên bộ nhớ vật lý. Một mấy chủ MongoDB có thể chứa nhiều Database.
* **Document**: một bản ghi thuộc một Collection thì được gọi là một Document. Các Document lần lượt bao gồm các trường tên và giá trị. Documents có schema động. Schema động có nghĩa là documents trong cùng một collection không cần phải có cùng một nhóm các fields hay cấu trúc giống nhau, và các fields phổ biến trong các documents của collection có thể chứa các loại dữ liệu khác nhau.
* **Field**: là một cặp name – value trong một document. Một document có thể có không hoặc nhiều trường. Các trường giống các cột ở cơ sở dữ liệu quan hệ.
* **JSON**: viết tắt của JavaScript Object Notation. Con người có thể đọc được ở định dạng văn bản đơn giản thể hiện cho các dữ liệu có cấu trúc. Hiện tại JSON đang hỗ trợ rất nhiều ngôn ngữ lập trình.
* **Index**: là những cấu trúc dữ liệu đặc biệt, dùng để chứa một phần nhỏ của các tập dữ liệu một cách dễ dàng để quét. Chỉ số lưu trữ giá trị của một fields cụ thể hoặc thiết lập các fields, sắp xếp theo giá trị của các fields này. Index hỗ trợ độ phân tích một cách hiệu quả các truy vấn. Nếu không có chỉ mục, MongoDB sẽ phải quét tất cả các documents của collection để chọn ra những document phù hợp với câu truy vấn. Quá trình quét này là không hiệu quả và yêu cầu MongoDB để xử lý một khối lượng lớn dữ liệu.

Lưu ý sự khác biệt của các trường và \_id trong một document. Một \_id được dùng để đại diện cho một document và chúng được sinh ra khi thêm một Document vào Collection.

* + 1. **Khi nào nên sử dụng MongoDB**
* Nếu website có tính chất INSERT cao bởi vì mặc định MongoDB có sẵn cơ chế ghi với tốc độ cao và an toàn.Website 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 sẽ bỏ qua phần đó nên sẽ an toàn.
* Website có nhiều dữ liệu quá giả sử web có đến 10 triệu records thì đó là cơn ác mộng với MYSQL. Bởi vì MongoDB có khả năng tìm kiến 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 SQLITE hoặc là MongoDB.
  + 1. **Khi nào không nên sử dụng MongoDB**
* Các ứng dụng cần sử dụng nhiều transaction (như ngân hàng) do Mongodb không có cơ chế transaction (giao dịch) để phục vụ cho các ứng dụng ngân hàng.
* Các ứng dụng cần SQL (sử dụng joins).

### **Ưu nhược điểm của MongoDB**

* **Ưu điểm**
* Schema linh hoạt: do MongoDB sử dụng lưu trữ dữ liệu dưới dạng Document JSON nên mỗi một collection sẽ các các kích cỡ và các document khác nhau.
* Cấu trúc đối tượng rõ ràng: Tuy rằng cấu trúc của dữ liệu là linh hoạt nhưng đối tượng của nó được xác định rất rõ ràng. Sử dụng bộ nhớ nội tại, nên truy vấn sẽ rất nhanh.
* MongoDB rất dễ mở rộng.
* Không có các join: Điều này cũng góp phần tạo nên tốc độ truy vấn cực nhanh trên mongoDB.
* MongoDB phù hợp cho các ứng dụng realtime.
* Ít schema hơn: vì schema được sinh ra là để nhóm các đối tượng vào 1 cụm, dễ quản lý.
* **Nhược điểm**
* Điều đầu tiên phải kể đến ở đây là MongoDB không có các tính chất ràng buộc như trong RDBMS nên khi thao tác với mongoDB thì phải hết sức cẩn thận.
* MongoDB sử dụng sẽ hao tốn tài nguyên của hệ thống nhiều hơn RDBMS. Nhưng đến thời điểm hiện tại thì vấn đề này không còn là điều lo ngại nữa.
* 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.
  + 1. **Các kiểu dữ liệu trong MongoDB**
* Những loại dữ liệu có 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” |

* Mô tả
* 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 Expresion.
  + 1. **Một số câu lệnh thường dùng trong MongoDB**

|  |  |  |
| --- | --- | --- |
| Câu lệnh | SQL | MongoDB |
| Create table | CREATE TABLE people (id MEDIUMINT NOT NULL AUTO\_INCREMENT, user\_id Varchar(30), age Number, status char(1), PRIMARY KEY (id)) | db.people.insertOne({User\_id: “abc123”, Age: 55, Status: “A”}) |
| 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" }) |
| Select | SELECT \*FROM people | db.people.find() |
|  | SELECT id,user\_id, status FROM people | db.people.find( { }, { user\_id: 1, status: 1 } ) |
|  | SELECT \* FROM people WHERE status = "A" | db.people.find( { status: "A" } ) |
|  | SELECT \* FROM people WHERE status = "A" AND age = 50 | db.people.find( { status: "A", age: 50 } ) |
|  | SELECT \* FROM people WHERE status = "A" OR age = 50 | db.people.find( { $or: [ { status: "A" } , { age: 50 } ] } ) |
|  | SELECT \* FROM people WHERE user\_id like "%bc%" | db.people.find( { user\_id: /bc/ } )  db.people.find( { user\_id: { $regex: /bc/ } } ) |
|  | SELECT COUNT(user\_id) FROM people | db.people.count( { user\_id: { $exists: true } } )  db.people.find( { user\_id: { $exists: true } } ).count() |
| Update records | UPDATE people SET status = "C" WHERE age > 25 | db.people.updateMany( { age: { $gt: 25 } }, { $set: { status: "C" } } ) |
|  | UPDATE people SET age = age + 3 WHERE status = "A" | db.people.updateMany( { status: "A" } , { $inc: { age: 3 } } ) |
| Delete Records | DELETE FROM people WHERE status = "D" | db.people.deleteMany( { status: "D" } ) |
|  | DELETE FROM people | db.people.deleteMany({}) |

**Ví dụ:** Hãy cùng xem xét các truy vấn nhất định mà chúng ta có thể làm với MongoDB. Giả sử muốn trả về tất cả các tài liệu trong bộ sưu tập đơn hàng (tất cả các hàng trong bảng đơn hàng).

|  |  |  |
| --- | --- | --- |
|  | SQL | MongoDB |
| Truy vấn | SELECT \* FROM order | db.order.find() |
| Chọn đơn đặt hàng cho một khách hàng “883c2c5b4e5b” | SELECT \* FROM order WHERE customerId = "883c2c5b4e5b" | db.order.find({"customerId":"883c2c5b4e5b"}) |
| chọn orderId và orderDate cho một khách hàng | SELECT orderId,orderDate FROM order WHERE customerId = "883c2c5b4e5b" | db.order.find({customerId:"883c2c5b4e5b"},{orderId:1,orderDate:1}) |
| Truy vấn tất cả các đơn hàng trong đó có một mặt hàng được đặt có tên là: Refactoring. | SELECT \* FROM customerOrder, orderItem, product  WHERE  customerOrder.orderId = orderItem.customerOrderId  AND orderItem.productId = product.productId  AND product.name LIKE '%Refactoring%' | db.orders.find({"items.product.name":/Refactoring/}) |

Truy vấn cho MongoDB đơn giản hơn vì các đối tượng được nhúng bên trong một tài liệu duy nhất và có thể truy vấn dựa trên các tài liệu con được nhúng.

* + 1. **Hoạt động của MongoDB**

MongoDB hoạt động dưới một tiến trình ngầm service, luôn mở một cổng (Cổng mặc định là 27017) để nghe các yêu cầu truy vấn, thao tác từ các ứng dụng gửi vào sau đó mới tiến hành xử lý.

Mỗi một bản ghi của MongoDB được tự động gắn thêm một field có tên “\_id” thuộc kiểu dữ liệu ObjectId mà nó quy định để xác định được tính duy nhất của bản ghi này so với bản ghi khác, cũng như phục vụ các thao tác tìm kiếm và truy vấn thông tin về sau. Trường dữ liệu “\_id” luôn được tự động đánh index (chỉ mục) để tốc độ truy vấn thông tin đạt hiệu suất cao nhất.

Mỗi khi có một truy vấn dữ liệu, bản ghi được ghi đệm (cache) lên bộ nhớ Ram, để phục vụ lượt truy vấn sau diễn ra nhanh hơn mà không cần phải đọc từ ổ cứng.

Khi có yêu cầu thêm/sửa/xóa bản ghi, để đảm bảo hiệu suất của ứng dụng mặc định 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.

* + 1. **Cài đặt MongoDB**
* Yêu cầu hệ thống
* Yêu cầu phần cứng: Hệ thống có kiến trúc 64 bit x86.
* Yêu cầu phần mềm: Windows 7 / Server 2008 R2 hoặc mới hơn.
* Cài đặt MongoDB (phiên bản MongoDB Community Server )
* Tải MongoDB từ link (tải file .msi):

<https://www.mongodb.com/download-center/community>

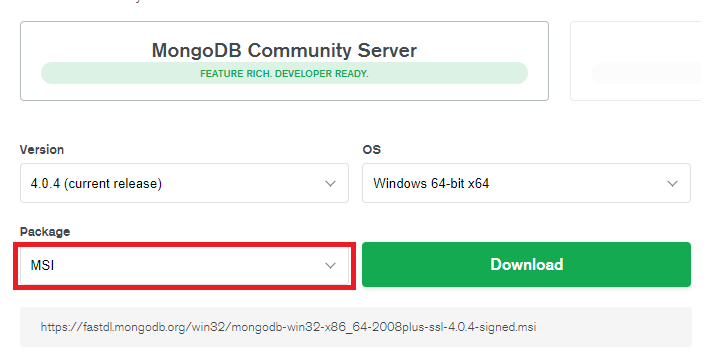


Figure: Download MongoDB

* Chạy file vừa tải về.

Làm theo hướng dẫn cài đặt MongoDB. Trình hướng dẫn sẽ hướng dẫn cài đặt MongoDB và MongoDB Compass. Bạn có thể chọn loại thiết lập Hoàn thành hoặc Tùy chỉnh . Nếu chọn Cài đặt tùy chỉnh, bạn có thể chỉ định thư mục cài đặt. MongoDB không có phụ thuộc hệ thống khác. Có thể cài đặt và chạy MongoDB từ bất kỳ thư mục nào. Bắt đầu từ MongoDB 4.0, có thể cài đặt theo mặc định hoặc cấu hình và khởi động MongoDB như một server trong khi cài đặt, và MongoDB server được bắt đầu khi cài đặt thành công.

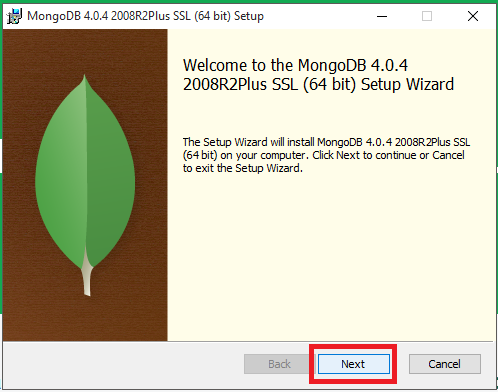


Figure: Cài đặt MongoDB

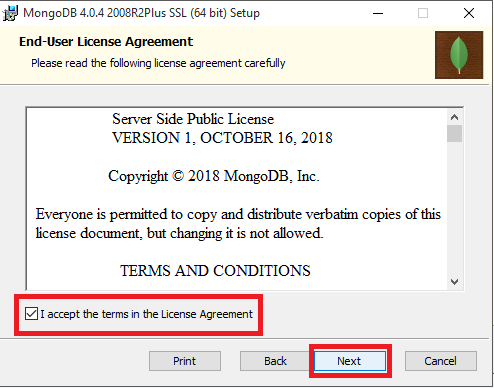


Figure: Đồng ý với thỏa thuận

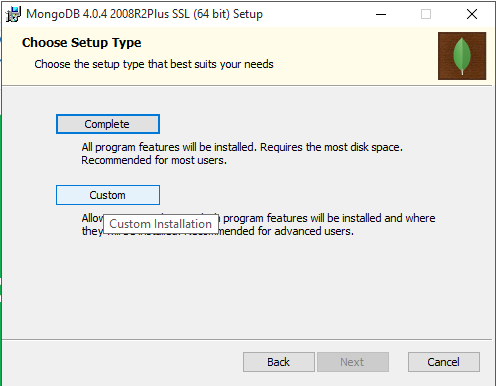


Figure: Chọn kiểu Setup

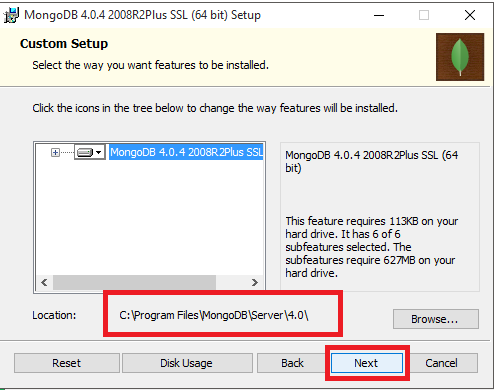


Figure: Chọn đường dẫn lưu trữ

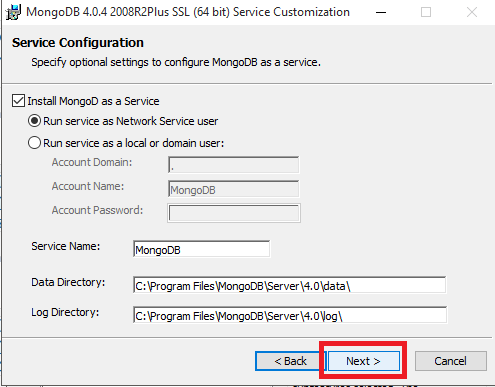


Figure: Thiết lập Service

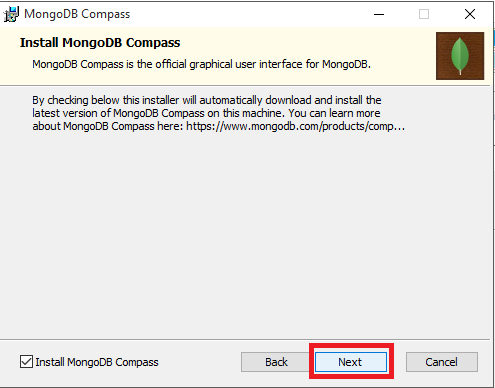


Figure: Cài đặt MongoDB Compass

* 1. **Xây dựng ứng dụng Demo**
     1. **Bài toán**
        1. **Đặt vấn đề**

Xã hội đang trên đà phát triển, chất lượng cuộc sống của người dân nước ta đang ngày càng nâng cao. Từ đó, nhu cầu chăm sóc sức khỏe bản thân càng lớn, mà nguồn thực phẩm chính là 1 phần yếu tố trực tiếp quyết định đến sức khỏe của họ. Tuy nhiên, việc xã hội ngày càng phát triển, thực phẩm đa dạng đa nguồn xuất hiện tràn lan trên thị trường, đồng nghĩa với việc nhiều thực phẩm không rõ nguồn gốc, không đảm bảo vệ sinh an toàn thực phẩm. Điều này tạo sự lo lắng, ảnh hưởng không nhỏ đến sức khỏe mọi người. Không những thế do công việc và cuộc sống bận rộn khiến những bà nội trợ luôn muốn tìm cách tiết kiệm thời gian. Bởi vậy, việc lựa chọn mua thực phẩm online tại những hệ thống uy tín là lựa chọn ưu tiên của các bà nội trợ trong thời đại 4.0. Nó cũng dần trở thành xu hướng mà mọi người muốn hướng đến, nhất là người dân sống tại thành phố lớn, trình độ dân trí cao. Chính vì vậy, người kinh doanh cần phải khai thác tiềm năng thị trường, đáp ứng nhu cầu thiết yếu của người dân. Vấn đề đặt ra là cần phải có một ứng dụng web phục vụ công việc kinh doanh mua bán online.

* + - 1. **Mục tiêu**

Xây dựng một website bán hàng online nhằm:

* Hỗ trợ công việc mua bán
* Khách hàng thuận tiện tìm kiếm, mua sắm.
* Người kinh doanh dễ dàng quản lý công việc bán hàng như thống kê, báo cáo doanh thu...
  + - 1. **Phạm vi hệ thống**

Hệ thống gồm có 2 phân hệ:

* Phân hệ Front-end.
* Phân hệ Back-end.
  + - 1. **Các tác nhân sử dụng hệ thống**

|  |  |  |
| --- | --- | --- |
| **STT** | **Người Sử Dụng** | **Mô Tả** |
| 1 | Admin | Là người quản trị Website có toàn quyền quản trị hệ thống:   * Quản lý nhập hàng: cập nhật thông tin vào cơ sở dữ liệu, update vào hóa đơn nhập hàng, xem chi tiết hóa đơn, cập nhật số lượng hiện tại. * Quản lý bán hàng: tìm kiếm sản phẩm, xem thông tin sản phẩm, quản lý danh mục sản phẩm (Thêm, sửa, xóa). * Thống kê báo cáo: thống kê đơn đặt hàng, thống kê doanh thu (theo ngày, tuần, tháng). * Quản lý người dùng: thông tin khách hàng. |
| 2 | Khách hàng thành viên | Là những khách hàng đã đăng ký tài khoản trên hệ thống có các quyền:   * Xem thông tin sản phẩm. * Tìm kiếm sản phẩm. * Đặt hàng. * Quản lý thông tin tài khoản cá nhân. * Xem lịch sử các giao dịch. |
| 3 | Khách vãng lai | Là những khách hàng chưa đăng ký tài khoản trên hệ thống có các quyền:   * Tìm kiếm sản phẩm. * Xem chi tiết sản phẩm. * Đăng ký tài khoản. |

* + - 1. **Yêu cầu**
* **Đặc tả yêu cầu chức năng**
* ***SAEQ001: Chức năng đăng nhập***
* Tác nhân:

Khách hàng thành viên, quản trị hệ thống (Admin)

* Mô tả nghiệp vụ:

Chức năng đăng nhập cho phép người dùng (Admin, khách thành viên) có thể thực hiện thao tác đăng nhập vào hệ thống.

* Dòng sự kiện chính:
* Tại trang chủ người dùng Click vào button “Đăng nhập” ở menu phía bên phải màn hình. Hệ thống hiển thị màn hình Login yêu cầu nhập thông tin đăng nhập.
* Người dùng nhập: “Địa chỉ Email” và “Mật khẩu”.
* Người dùng Click vào button “Đăng nhập” để đăng nhập.
* Dòng sự kiện rẽ nhánh:
* Khi người dùng nhập thông tin mật khẩu hoặc tài khoản không đúng. Hệ thống thông báo là “Thông tin đăng nhập không chính sác”.
* Khi người dùng Click vào ‘Thoát’ hệ thống thoát khỏi màn hình đăng nhập chuyển đến trang chủ.
* Yêu cầu đặc biệt:

Website chạy và kết nối với cơ sở liệu thành công.

* Điều kiện trước:

Người dùng đã được cấp tài khoản.

* Điều kiện sau:
* Đối với tài khoản là khách thành viên: màn hình chuyển đến trang chủ.
* Đối với tài khoản là quản trị (Admin) hệ thống chuyển đến trang quản trị.
* ***SAEQ002: Chức năng đăng xuất***
* Tác nhân:

Khách hàng thành viên, người quản trị hệ thống (Admin)

* Mô tả nghiệp vụ:

Chức năng này cho phép người dùng đăng xuất tài khoản khỏi hệ thống.

* Dòng sự kiện chính:

Trên menu bên phải màn hình có button “Đăng xuất”(chính là button thay thế button “Đăng nhập” của chức năng SAEQ001 - Đăng nhập). Người dùng Click vào button “Đăng xuất” để đăng xuất tài khoản.

* Dòng sự kiện rẽ nhánh:

Không có.

* Điều kiện đặc biệt:

Không có.

* Điều kiện trước:

Tài khoản đã đăng nhập vào hệ thống thành công.

* Điều kiện sau:

Chuyển đến trang chủ.

* ***SAEQ003: Chức năng đăng ký tài khoản***
* Tác nhân sử dụng:

Khách vãng lai

* Mô tả nghiệp vụ:

Chức năng này cho phép người dùng là khách vãng lai có thể đăng ký tài khoản để có thể thực hiện việc mua hàng trên website.

* Dòng sự kiện chính:
* Trên menu bên phải màn hình có button “Đăng ký” (chính là button bên cạnh button “Đăng nhập” của chức năng SAEQ001 - Đăng nhập).
* Người dùng Click vào button “Đăng ký” để đăng ký tài khoản. Hệ thống hiển thị màn hình đăng ký.
* Trên màn hình đăng ký hệ thống yêu cầu người dùng nhập thông tin đăng ký tài khoản:

+ Chi tiết tài khoản (Họ và tên, Email, Số điện thoại, địa chỉ)

+ Mật khẩu (mật khẩu, xác nhận mật khẩu).

* Người dùng nhập vào thông tin và Click vào “Tiếp tục” để đăng ký.
* Hệ thống thông báo kết quả quá trình nhập thông tin. Nếu thông tin nhập không chính xác thực hiện dòng sự kiện rẽ nhánh. Nếu thông tin nhập vào chính xác hệ thống thông báo đăng ký thành công.
* Dòng sự kiện rẽ nhánh:
* Nếu người dùng nhập thông tin không chính xác hệ thống thông báo yêu cầu người dùng nhập lại.
* Nếu người dùng đồng ý nhập lại thì quay lại bước nhập thông tin. Nếu không đồng ý thì thoát khỏi màn hình đăng ký.
* Điều kiện đặc biệt:

Người dùng chưa đăng ký tài khoản trên hệ thống.

* Điều kiện trước:

Không có

* Điều kiện sau:

Chuyển đến màn hình trang chủ

* ***SAEQ004: Chức năng thêm mới 1 danh mục sản phẩm***
* Tác nhân:

Người quản trị hệ thống (Admin)

* Mô tả nghiệp vụ:

Chức năng này cho phép người dùng là Admin có thể thực hiện công việc thêm mới 1 loại sản phẩm.

* Dòng sự kiện chính:
* Sau khi đăng nhập vào hệ thống, hiển thị màn hình quản trị hệ thống. Trên menu bên trái màn hình có “Danh mục sản phẩm”.
* Người dùng Click vào “Danh mục sản phẩm”
* Hệ thống hiển thị màn hình danh mục sản phẩm. Trên màn hình này cho phép người dùng thực hiện: click button “Thêm loại sản phẩm” để thêm mới 1 loại sản phẩm.
* Người dùng Click vào button “Thêm loại sản phẩm” hệ thống yêu cầu nhập vào text box (không được bỏ trống): “Mã loại sản phẩm”, “Tên loại sản phẩm”.
* Người dùng Click vào “Thêm loại sản phẩm” để lưu dữ liệu vào Database hoặc Click vào button “Quay lại” để hủy quá trình thêm mới hệ thống quay trở về màn hình “Danh mục sản phẩm”.
* Hệ thống thông báo kết quả quá trình nhập thông tin. Nếu thông tin nhập vào không hợp lệ thực hiện dòng sự kiện rẽ nhánh. Nếu thông tin nhập vào hợp lệ hệ thống thông báo “Thêm loại sản phẩm thành công” và quay trở về màn hình “Danh mục sản phẩm”
* Dòng sự kiện rẽ nhánh:
* Nếu người dùng chưa nhập mã loại sản phẩm mà click vào “Thêm mới sản phẩm” hệ thống thông báo: “Chưa nhập mã loại sản phẩm”.
* Nếu người dùng chưa nhập tên loại sản phẩm mà click vào “Thêm mới sản phẩm” hệ thống thông báo: “Chưa nhập tên loại sản phẩm”
* Điều kiện đặc biệt:

Website chạy và kết nối với cơ sở liệu thành công.

* Điều kiện trước:

Người dùng đã được cấp tài khoản quản trị hệ thống (Admin)

* Điều kiện sau:

Chuyển đến màn hình danh mục sản phẩm

* ***SAEQ005: Chức năng sửa loại danh mục sản phẩm***
* Tác nhân:

Người quản trị hệ thống (Admin)

* Mô tả nghiệp vụ:

Chức năng này cho phép người dùng là Admin có thể thực hiện công việc sửa 1 loại sản phẩm.

* Dòng sự kiện chính:
* Sau khi đăng nhập vào hệ thống, hiển thị màn hình quản trị hệ thống. Trên menu bên trái màn hình có “Danh mục sản phẩm”.
* Người dùng Click vào “Danh mục sản phẩm”
* Hệ thống hiển thị màn hình danh mục sản phẩm. Trên màn hình này cho phép người dùng thực hiện: click button “Cập nhật” để sửa tên loại sản phẩm.
* Người dùng Click vào button “Cập nhật” hệ thống hiển thị chi tiết loại sản phẩm mà người dùng chọn sửa.
* Chức năng này chỉ cho phép người dùng sửa tên loại sản phẩm.
* Người dùng Click vào “Cập nhật loại sản phẩm” để lưu dữ liệu vào Database hoặc Click vào button “Quay lại” để hủy quá trình sửa tên loại sản phẩm hệ thống quay trở về màn hình “Danh mục sản phẩm”.
* Hệ thống thông báo kết quả quá trình cập nhật. Nếu thông tin cập nhật vào không hợp lệ thực hiện dòng sự kiện rẽ nhánh. Nếu thông tin cập nhật hợp lệ hệ thống thông báo “Cập nhật sản phẩm thành công” và quay trở về màn hình “Danh mục sản phẩm”
* Dòng sự kiện rẽ nhánh:

Nếu người dùng bỏ trống tên loại sản phẩm mà click vào “Cập nhật loại sản phẩm” hệ thống thông báo: “Chưa nhập tên loại sản phẩm”

* Điều kiện đặc biệt:

Website chạy và kết nối với cơ sở liệu thành công.

* Điều kiện trước:

Người dùng đã được cấp tài khoản quản trị hệ thống (Admin)

* Điều kiện sau:

Chuyển đến màn hình danh mục sản phẩm.

* ***SAEQ006: Chức năng thêm mới sản phẩm***
* Tác nhân:

Người quản trị hệ thống (Admin)

* Mô tả nghiệp vụ:

Chức năng này cho phép người dùng là Admin có thể thực hiện công việc thêm mới 1 sản phẩm.

* Dòng sự kiện chính:
* Sau khi đăng nhập vào hệ thống, hiển thị màn hình quản trị hệ thống. Trên menu bên trái màn hình có “Danh sách sản phẩm”.
* Người dùng Click vào “Danh sách sản phẩm”
* Hệ thống hiển thị màn hình danh sách sản phẩm. Trên màn hình này cho phép người dùng thực hiện: click button “Thêm sản phẩm” để thêm mới 1 sản phẩm.
* Người dùng Click vào button “Thêm sản phẩm” hệ thống hiển thị màn hình thêm sản phẩm yêu cầu:

+ Chọn tên loại sản phẩm có trong Droplist.

+ Nhập vào textbox (không được bỏ trống): “Mã sản phẩm”, “Tên sản phẩm”.

+ Nhập vào textbox: “Số lượng”, “giá”, “giảm giá”, “mô tả”.

+ Chọn ảnh hiển thị (bắt buộc): Click vào button “Chọn tệp” để chọn đủ 2 ảnh đại diện cho sản phẩm là “ảnh hiển thị” và “Ảnh hiển thị chi tiết sản phẩm”.

* Người dùng Click vào “Thêm sản phẩm” để lưu dữ liệu vào Database hoặc Click vào button “Quay lại” để hủy quá trình thêm mới hệ thống quay trở về màn hình “Danh sách sản phẩm”.
* Hệ thống thông báo kết quả quá trình nhập thông tin. Nếu thông tin nhập vào không hợp lệ thực hiện dòng sự kiện rẽ nhánh. Nếu thông tin nhập vào hợp lệ hệ thống thông báo “Thêm sản phẩm thành công” và quay trở về màn hình “Danh sách sản phẩm”.
* Dòng sự kiện rẽ nhánh:
* Nếu người dùng chưa nhập mã sản phẩm mà click vào “Thêm sản phẩm” hệ thống thông báo: “Chưa nhập mã sản phẩm”.
* Nếu người dùng chưa nhập tên sản phẩm mà click vào “Thêm sản phẩm” hệ thống thông báo: “Chưa nhập tên sản phẩm”.
* Nếu người dùng chưa chọn đủ 2 ảnh thì hệ thống thông báo chưa chọn đủ ảnh cho sản phẩm.
* Nếu người dùng chọn ảnh có định dạng khác: jpg, jpeg, png thì ảnh không thể hiển thị. Hệ thống thông báo: “Chọn ảnh theo định dạng jpg, jpeg, png”.
* Điều kiện đặc biệt:

Website chạy và kết nối với cơ sở liệu thành công.

* Điều kiện trước:

Người dùng đã được cấp tài khoản quản trị hệ thống (Admin).

* Điều kiện sau:

Chuyển đến màn hình danh sách sản phẩm.

* ***SAEQ008: Chức năng sửa thông tin sản phẩm***
* Tác nhân:

Người quản trị hệ thống (Admin)

* Mô tả nghiệp vụ:

Chức năng này cho phép người dùng là Admin có thể thực hiện công việc sửa thông tin của 1 sản phẩm.

* Dòng sự kiện chính:
* Sau khi đăng nhập vào hệ thống, hiển thị màn hình quản trị hệ thống. Trên menu bên trái màn hình có “Danh sách sản phẩm”.
* Người dùng Click vào “Danh sách sản phẩm”
* Hệ thống hiển thị màn hình danh sách sản phẩm. Trên màn hình này cho phép người dùng thực hiện: click button “Cập nhật” để sửa thông tin của 1 sản phẩm.
* Người dùng Click vào button “Cập nhật” hệ thống hiển thị thông tin chi tiết sản phẩm yêu cầu:

+ Sửa tên sản phẩm

+ Sửa: “Số lượng”, “giá”, “giảm giá”, “mô tả”.

+ Sửa ảnh hiển thị

* Người dùng Click vào “Cập nhật sản phẩm” để lưu dữ liệu vào Database hoặc Click vào button “Quay lại” để hủy quá trình cập nhật hệ thống quay trở về màn hình “Danh sách sản phẩm”.
* Hệ thống thông báo kết quả quá trình nhập thông tin. Nếu thông tin cập nhật không hợp lệ thực hiện dòng sự kiện rẽ nhánh. Nếu thông tin cập nhật hợp lệ hệ thống thông báo “Cập nhật phẩm thành công” và quay trở về màn hình “Danh sách sản phẩm”.
* Dòng sự kiện rẽ nhánh:

Nếu người dùng bỏ trống tên sản phẩm mà click vào “Cập nhật sản phẩm” hệ thống thông báo: “Chưa nhập tên sản phẩm”

* Điều kiện đặc biệt:

Website chạy và kết nối với cơ sở liệu thành công.

* Điều kiện trước:

Người dùng đã được cấp tài khoản quản trị hệ thống (Admin).

* Điều kiện sau:

Chuyển đến màn hình danh sách sản phẩm.

* ***SAEQ009: Chức năng xóa sản phẩm***
* Tác nhân:

Người quản trị hệ thống (Admin)

* Mô tả nghiệp vụ:

Chức năng này cho phép người dùng là Admin có thể thực hiện công việc xóa 1 sản phẩm.

* Dòng sự kiện chính:
* Sau khi đăng nhập vào hệ thống, hiển thị màn hình quản trị hệ thống. Trên menu bên trái màn hình có “Danh sách sản phẩm”.
* Người dùng Click vào “Danh sách sản phẩm”
* Hệ thống hiển thị màn hình danh sách sản phẩm. Trên màn hình này cho phép người dùng thực hiện: click button “Xóa” để xóa 1 sản phẩm.
* Người dùng Click vào button “Xóa” hệ thống hiển thị Dialog thông báo: “Bạn có muốn xóa sản phẩm không”

+ Người dùng chọn “Có” để xóa sản phẩm.

+ Người dùng chọn “không” hoặc button “thoát” để quay lại.

* Hệ thống thông báo kết quả quá trình xóa sản phẩm. Nếu xóa sản phẩm thành công hệ thống thông báo “Xóa sản phẩm thành công” và quay trở về màn hình “Danh sách sản phẩm”.
* Dòng sự kiện rẽ nhánh:

Không có

* Điều kiện đặc biệt:

Website chạy và kết nối với cơ sở liệu thành công.

* Điều kiện trước:

Người dùng đã được cấp tài khoản quản trị hệ thống (Admin).

* Điều kiện sau:

Chuyển đến màn hình danh sách sản phẩm.

* ***SAEQ010: Chức năng tìm kiếm sản phẩm (Back - end)***
* Tác nhân:

Người quản trị hệ thống (Admin)

* Mô tả nghiệp vụ:

Chức năng này cho phép người dùng là Admin có thể thực hiện công việc tìm kiếm sản phẩm.

* Dòng sự kiện chính:
* Sau khi đăng nhập vào hệ thống.
* Hệ thống hiển thị màn hình danh sách sản phẩm. Trên màn hình này cho phép người dùng thực hiện tìm kiếm sản phẩm. Chức năng này cho phép người dùng tìm kiếm sản phẩm theo loại sản phẩm.
* Người dùng tiến hành tìm kiếm sản phẩm:

+ Người dùng chọn loại sản phẩm trong Droplist.

+ Người dùng nhập từ khóa tìm kiếm vào textbox (Không bắt buộc nhập).

+ Click vào button “tìm kiếm”

* Hệ thống thông báo kết quả quá trình tìm kiếm sản phẩm.
* Dòng sự kiện rẽ nhánh:
* Nếu từ khóa nhập vào có trong CSDL của loại sản phẩm thì hệ thống hiển thị danh sách sản phẩm trùng với kết quả tìm kiếm.
* Nếu người dùng không nhập vào từ khóa tìm kiếm hệ thống hiển thị danh sách tất cả sản phẩm có trong loại sản phẩm.
* Điều kiện đặc biệt:

Website chạy và kết nối với cơ sở liệu thành công.

* Điều kiện trước:

Người dùng đã được cấp tài khoản quản trị hệ thống (Admin).

* Điều kiện sau:

Chuyển đến màn hình danh sách sản phẩm.

* ***SAEQ011: Chức năng tìm kiếm sản phẩm (Front - end)***
* Tác nhân:

Khách hàng vãng lai, khách hàng thành viên.

* Mô tả nghiệp vụ:

Chức năng này cho phép người dùng là khách hàng có thể thực hiện công việc tìm kiếm sản phẩm.

* Dòng sự kiện chính:
* Trên menu có Form “Tìm kiếm” cho phép người dùng thực hiện tìm kiếm sản phẩm.
* Người dùng tiến hành tìm kiếm sản phẩm:

+ Người dùng nhập từ khóa tìm kiếm vào textbox (Không bắt buộc nhập).

+ Click vào button “Tìm kiếm”

* Hệ thống hiển thị kết quả của quá trình tìm kiếm sản phẩm.
* Dòng sự kiện rẽ nhánh:
* Nếu từ khóa nhập vào có trong CSDL của thì hệ thống hiển thị danh sách sản phẩm trùng với kết quả tìm kiếm.
* Nếu người dùng không nhập vào từ khóa tìm kiếm hệ thống hiển thị danh sách tất cả sản phẩm.
* Nếu từ khóa nhập vào không trùng với kết quả tìm kiếm thì hệ thống thông báo: “Không có sản phẩm nào”
* Điều kiện đặc biệt:

Website chạy và kết nối với cơ sở liệu thành công.

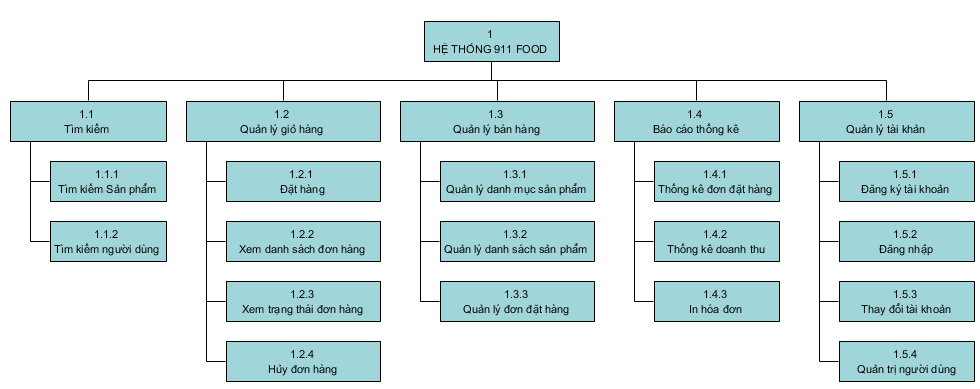
* Điều kiện trước:

Không có

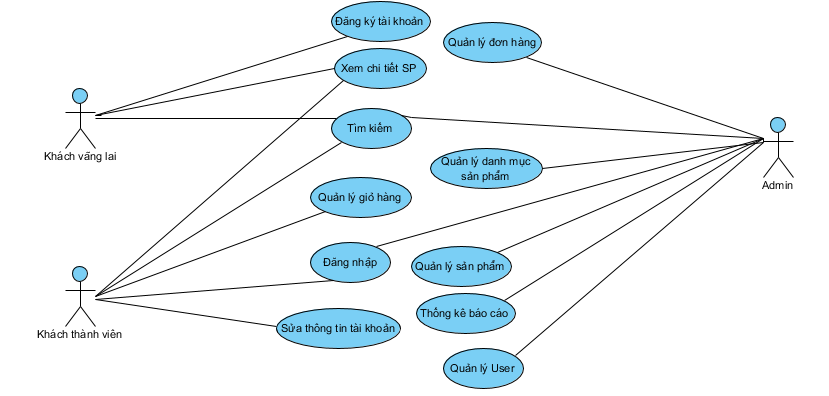
* Điều kiện sau:

Chuyển đến màn hình danh sách sản phẩm.

* + 1. **Phân tích thiết kế hệ thống**
       1. **Các sơ đồ**

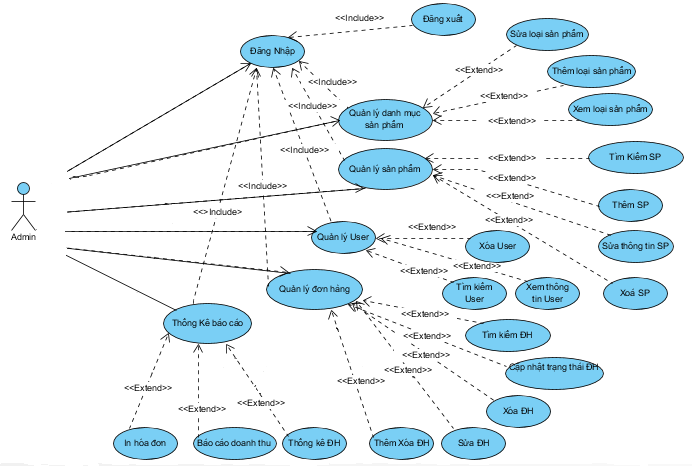
**Sơ đồ phân cấp chức năng**

**Sơ đồ Use Case tổng quát**



**Sơ đồ Use Case chi tiết**

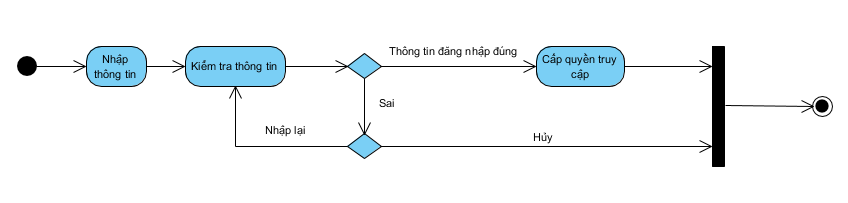
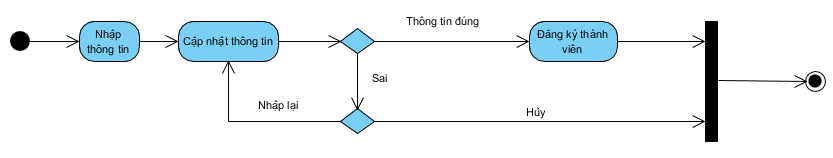
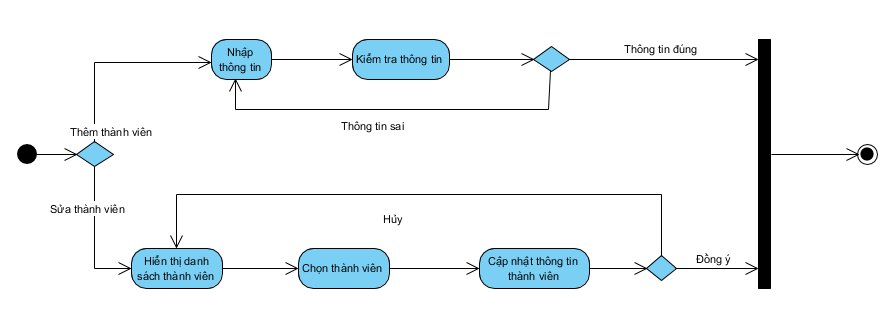
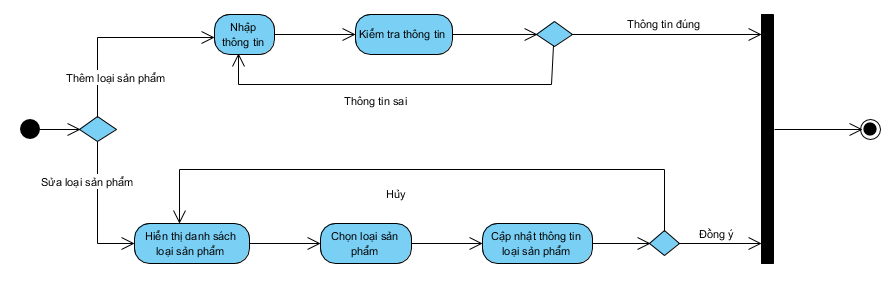
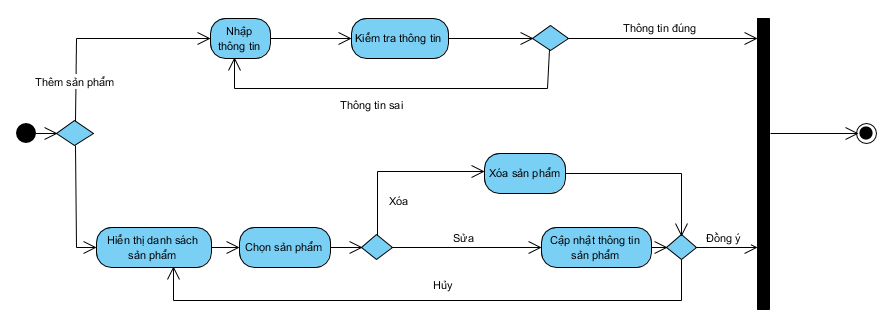
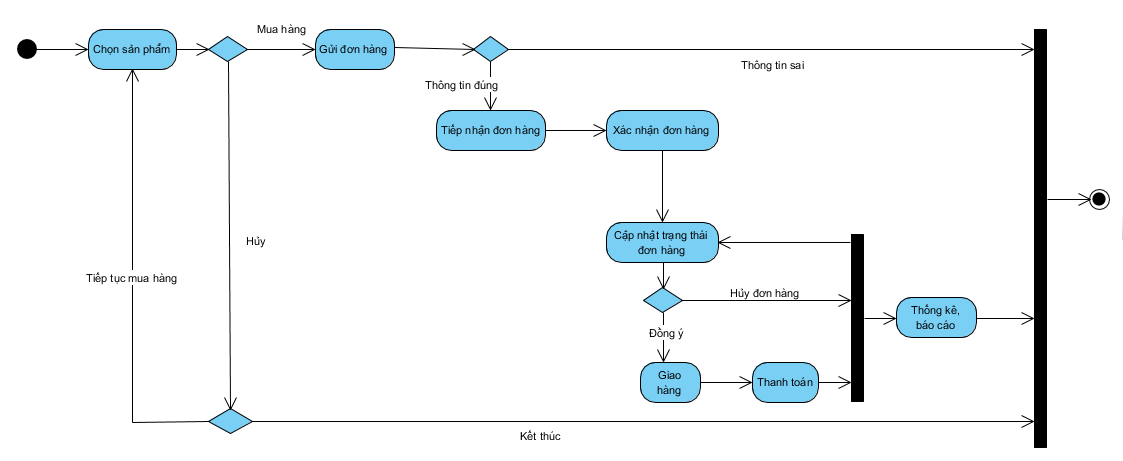
* Use case chi tiết dành cho ADMIN

****

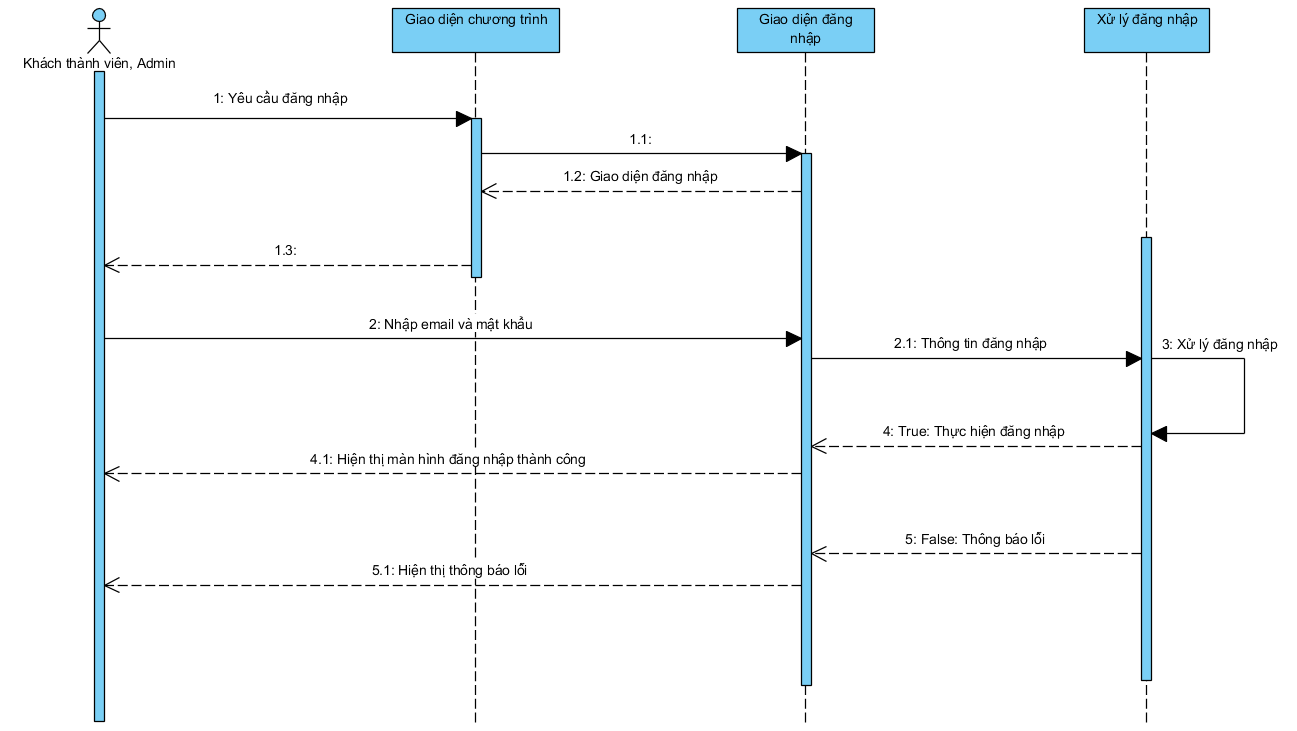
* Use Case chi tiết dành cho khách hàng

****

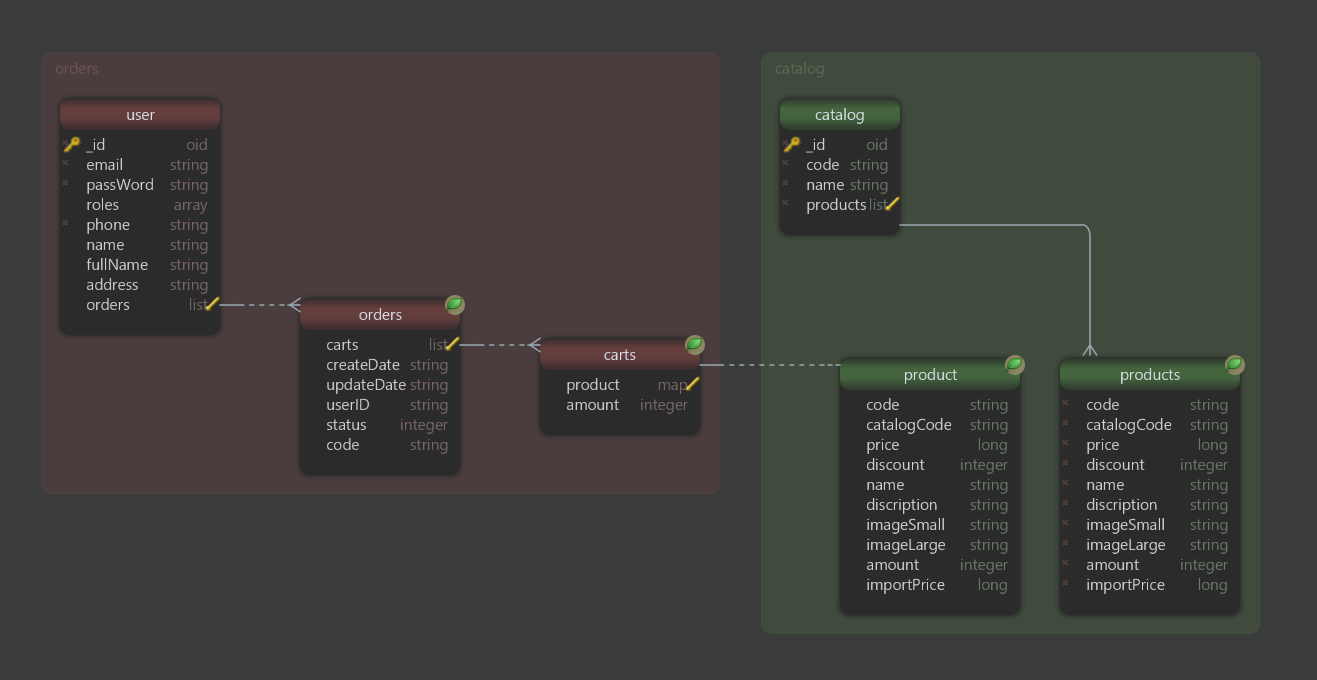
**Biểu đồ hoạt động**

* Chức năng đăng nhập
* Chức năng đăng ký tài khoản
* Chức năng quản lý người dùng
* Chức năng quản lý danh mục sản phẩm
* Chức năng quản lý sản phẩm
* Chức năng mua hàng, xử lý đơn hàng

**Biểu đồ tuần tự**



* 1. **Cơ sở dữ liệu**



# **PHẦN V: KẾT LUẬN VÀ ĐỀ NGHỊ**

5.1. Kết luận

Qua thời gian tìm hiểu và thực hiện đề tài: “Tìm hiểu về NoSQL và ứng dụng”, em đã thu được một số kết quả như sau:

- Về lý thuyết:

+ Tìm hiểu tổng quan về cơ sở dữ liệu NoSQL: đặc điểm, phân loại cùng với những ứng dụng thực tiễn của nó.

+ So sánh cơ sở dữ liệu NoSQL với cơ sở dữ liệu quan hệ và xml.

+ Tìm hiểu tổng quan một số hệ quản trị cơ sở dữ liệu dạng NoSQL: RavenDB, Hadoop, Cassandra, CouchDB.

+ Tìm hiểu về hệ quản trị cơ sở dữ liệu MongoDB và vận dụng kiến thức đã tìm hiểu được vào một ứng dụng thực tế.

- Về thực nghiệm:

Xây dựng được chương trình demo module quản lý User sử dụng hệ quản trị cơ sở dữ liệu MongoDB. Ứng dụng áp dụng được những lý thuyết cơ bản của MongoDB.

5.2. Hướng phát triển

# **PHẦN VI: TÀI LIỆU THAM KHẢO**

1 - NoSQL resources: <http://nosql-database.org/>

2 - NoSQL wiki - <http://en.wikipedia.org/wiki/NoSQL/>

3 - MongoDB vs. SQL Server 2008 Performance Showdown -http://www.michaelckennedy.net/blog/2010/04/29/MongoDBVsSQLServer2008PerformanceShowdown.aspx

4 - NoSQL in the Enterprise - <http://www.infoq.com/articles/nosql-in-the-enterprise>

5 - Những điểm khác biet giữa SQL-RDBMS <https://viblo.asia/p/nhung-diem-khac-biet-giua-sql-va-nosql-gGJ59b4rKX2>