BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TPHCM

NGUYỄN VĂN HÒA

NGHIÊN CỨU VỀ CHUYỂN ĐỔI LƯỢC ĐỒ CƠ SỞ DỮ LIỆU QUAN HỆ SANG CƠ SỞ DỮ LIỆU NOSQL

LUẬN VĂN THẠC SĨ

Chuyên ngành: Công nghệ thông tin

Mã ngành: 60480201

TP. HCM, tháng 06/2015

BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TPHCM

NGUYỄN VĂN HÒA

NGHIÊN CỨU VỀ CHUYỂN ĐỔI LƯỢC ĐỒ CƠ SỞ DỮ LIỆU QUAN HỆ SANG CƠ SỞ DỮ LIỆU NOSQL

LUẬN VĂN THẠC SĨ

Chuyên ngành: Công nghệ thông tin

Mã ngành: 60480201

HƯỚNG DẪN KHOA HỌC: TS.NGUYỄN ĐÌNH THUÂN

TP. HCM, tháng 06/2015

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH TẠI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP. HCM

Cán bộ hướng dẫn khoa học: **TS. Nguyễn Đình Thuân**

(Ghi rõ họ, tên, học hàm, học vị và chữ ký)

Luận vănThạc sĩ được bảo vệ tại Trường Đại học Công nghệ TP. HCM ngày… tháng …năm…

Thành phần Hội đồng đánh giá Luận văn Thạc sĩ gồm:

(Ghi rõ họ, tên, học hàm, học vị của Hội đồng chấm bảo vệ Luận văn Thạc sĩ)

TT	Họ và tên	Chức danh Hội đồng
1	PGS. TSKH. Nguyễn Xuân Huy	Chủ tịch
2	TS. Cao Tùng Anh	Phản biện 1
3	TS. Trần Đức Khánh	Phản biện 2
4	PGS. TS. Lê Hoài Bắc	Ủy viên
5	TS. Võ Đình Bảy	Ủy viên, Thư ký

Xác nhận của Chủ tịch Hội đồng đánh giá Luận sau khi Luận văn đã được sửa chữa (nếu có).

Chủ tịch Hội đồng đánh giá LV

TRƯỜNG ĐH CÔNG NGHỆ TP. HCM
PHÒNG QLKH – ĐTSĐH

CỘNG HÕA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

TP. HCM, ngày 08 tháng 06 năm 2015

NHIỆM VỤ LUẬN VĂN THẠC SĨ

Họtênhọc viên: Nguyễn Văn Hòa Giới tính: Nam

Ngày, tháng, năm sinh: 28/09/1977 Nơi sinh: Kiên Giang

Chuyên ngành: Công nghệ thông tin MSHV: 1341860005

I-Tên đề tài:

NGHIÊN CỨU VỀ CHUYỂN ĐỔI LƯỢC ĐỒ CƠ SỞ DỮ LIỆU QUAN HỆ SANG CƠ SỞ DỮ LIỆU NOSQL

II- Nhiệm vụ và nội dung:

- Nghiên cứu về cơ sở dữ liệu NoSQL: tìm hiểu những điều căn bản về dữ liệu NoSQLvà các thao tác, nguyên lý hệ thống của nó, các server hỗ trợ NoSQL hiện nay, cụ thể sử dụng MongoDB, tại sao lại sử dụng MongoDB.
- Chuyển đổi lược đồ cơ sở dữ liệu quan hệ sang cơ sở dữ liệu NoSQL: tìm hiểu tương quan các thành phần trong lược đồ SQL và NoSQL.
- So sánh tốc độ thực thi một số câu truy vấn của SQL và NoSQL: ưu và khuyết điểm của SQL và NoSQL, ứng dụng của NoSQL hiện nay ngoài thực tiễn, hướng phát triển trong tương lai.

III- Ngày giao nhiệm vụ: 15/09/2014

IV- Ngày hoàn thành nhiệm vụ: 08/06/2015

V- Cán bộ hướng dẫn: TS**. Nguyễn Đình Thuân**

CÁN BỘ HƯỚNG DẪN **KHOA QUẢN LÝ CHUYÊN NGÀNH**

(Họ tên và chữ ký)

TS. NGUYỄN ĐÌNH THUÂN

LỜI CAM ĐOAN

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi. Các số liệu, kết quả nêu trong Luận văn là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác.

Tôi xin cam đoan rằng mọi sự giúp đỡ cho việc thực hiện Luận văn này đã được cảm ơn và các thông tin trích dẫn trong Luận văn đã được chỉ rõ nguồn gốc.

Học viên thực hiện Luận văn

Nguyễn Văn Hòa

LỜI CẢM ƠN

Em xin gửi lời cảm ơn sâu sắc đến thầy Nguyễn Đình Thuân, đã giúp đỡ, tạo điều kiện cho em hoàn thành tốt luận văn tốt nghiệp này. Thầy đã tận tình hướng dẫn và đưa ra những nhận xét vô cùng quý giá để đề tài ngày càng hoàn thiện hơn. Những góp ý của thầy giúp cho em tiếp cận, hiểu rõ và giải quyết vấn đề dễ dàng hơn.

Đồng thời, em cũng xin bày tỏ lòng biết ơn đến quý thầy, cô Trường Đại Học Công Nghệ Thành Phố Hồ Chí Minh, đặc biệt là các thầy, cô khoa Công nghệ thông tin đã tận tình truyền đạt kiến thức, kinh nghiệm cho em từ những ngày học tập tại trường. Sự nhiệt tình của các thầy, cô đã giúp cho em có kiến thức nền tảng vững chắc cũng như kinh nghiệm thực tiễn quý báu để chúng em có thể hoàn thành tốt các nhiệm vụ học tập, làm việc và nghiên cứu.

Bên cạnh đó, em cũng gửi lời cảm ơn đến gia đình, các anh, chị, bạn bè đã động viên, giúp đỡ chúng em rất nhiều trong quá trình học tập cũng như trong cuộc sống.

Thành phố Hồ Chí Minh, ngày 08 tháng 06 năm 2015

Học viên thực hiện

Nguyễn Văn Hòa

MỤC LỤC

DANH MỤC CÁC TỪ VIẾT TẮT

STT	Từ viết tắt/ ký hiệu	Từ đầy đủ
1.	CSDL	Cơ sở dữ liệu.
2.	DOM	Document Object Model (Mô hình Đối tượng Tài liệu).
3.	NoSQL	None-Relational SQL / Not-Only SQL.
4.	MSSQL	Hệ quản trị cơ sở dữ liệu Microsoft SQL Server.
5.	RDBMs	Relational database management system (Hệ quản trị cơ sở dữ liệu quan hệ).
6.	SQL	Structured Query Language (Ngôn ngữ truy vấn có cấu trúc).
7.	URL	Uniform Resource Locator (Định vị Tài nguyên thống nhất).
8.	ACID	Atomicity, consistency, Isolation, và Durability (Tính toàn vẹn dữ liệu)

DANH MỤC BẢNG

DANH MỤC HÌNH

DANH MỤC LƯỢC ĐỒ

TÓM TẮT KHÓA LUẬN

Hiện nay, lĩnh vực xử lý dữ liệu ngày càng phát triển, đặc biệt là đối với dữ liệu lớn (big data). Tùy vào lĩnh vực khai thác, các dữ liệu ngày càng không còn chú trọng về tính toàn vẹn của dữ liệu mà thay vào đó là những yêu cầu cao về tốc độ xử lý. Trong khi đó, các hệ dữ liệu quan hệ RDBMs dù được hỗ trợ rất chặt chẽ về tính toàn vẹn dữ liệu, tuy nhiên chính vì điều này, mà nó ngày càng bộc lộ những nhược điểm về tốc độ xử lý. Vì vậy, dữ liệu NoSQL ra đời nhằm giải quyết những nhược điểm mà các RDBMs mắc phải, những đặc điểm của nó sẽ bổ sung cho những điều mà RDBMs không thể thực hiện được. Chính vì thế em đã quyết định chọn đề tài "Tìm hiểu dữ liệu NoSQL, cách chuyển đổi giữa lược đồ RDBMs sang lược đồ NoSQL". Cụ thể trong đề tài này, em sẽ đi tìm hiểu và nghiên cứu chuyển đổi với hệ quản trị dữ liệu MongoDB, đây là một hệ quản trị sử dụng dữ liệu NoSQL phổ biến.

Để có thể chuyển đổi lược đồ dữ liệu từ RDBMs sang dữ liệu NoSQL, bước đầu tiên chúng ta cần phải hiểu được những điều căn bản về dữ liệu NoSQLvà các thao tác, nguyên lý hệ thống của nó, cách để tối ưu lược đồ để tổ chức dữ liệu trở nên hiệu quả hơn. Trong đề tài này, em sẽ đề cập đến những vấn đề trên.

LỜI MỞ ĐẦU

Ngày nay, đối với các công ty, doanh nghiệp, việc quản lý tốt, hiệu quả dữ liệu của riêng công ty cũng như dữ liệu khách hàng, đối tác là một trong những bài toán được ưu tiên hàng đầu và đang không ngừng gây khó khăn cho họ. Để có thể quản lý được nguồn dữ liệu đó, ban đầu các doanh nghiệp phải đầu tư, tính toán rất hiều loại chi phí như chi phí cho phần cứng, phần mềm, mạng, chi phí cho quản trị viên, chi phí bảo trì, sửa chữa, … Ngoài ra họ còn phải tính toán khả năng mở rộng, nâng cấp thiết bị; phải kiểm soát việc bảo mật dữ liệu cũng như tính sẵn sàng cao của dữ liệu.

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.

NoSQLvừa mang lại một giải pháp tốt hơn vừa tiết kiệm chi phí hơn do NoSQL có hiệu suất làm việc tốt hơn về tốc độ thực thi, khả năng lưu trữ, phân tán dữ liệu 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. Vì vậy NoSQL chính là sự lựa chọn tốt.

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 không thích ràng buộc thì hãy nghĩ đến NoSQL.

CHƯƠNG I. TÌM HIỂU NOSQL

1.1. Lý do chọn NoSQL

1.1.1. Yếu điểm của RDBMs

Ngày nay, Big Data, Big Users và Cloud Computing là các xu hướng đang phát triển cực kì mạnh mẽ đã dẫn tới sự phát triển theo của công nghệ NoSQL là sự tất yếu. NoSQL ngày càng được xem là một sự bổ sung cho RDBMs.

Các RDBMs hiện tại bộc lộ nhiều yếu điểm như đánh chỉ mục cho một lượng lớn dữ liệu, phân trang, phân phối luồng dữ liệu media, gặp khó khăn khi xử lý một lượng dữ liệu cực lớn và cập nhật liên tục do số lượng người dùng quá nhiều ở một thời điểm, gặp vấn đề về hiệu suất với những bài toán lớn về hệ thống thông tin, phân tán hay lưu trữ dữ liệu.

Ngoài ra, với chi phí triển khai cũng như phát triển các ứng dụng sử dụng CSDL quan hệ tốn kém đặc biệt khi truy vấn một lượng bản ghi lớn trong thời gian dài. Hơn thế nữa, với sự phát triển mạnh của những thiết bị cầm tay như smartphone thì CSDL quan hệ bộc lộ rõ khuyết điểm vì bộ nhớ của thiết bị cũng như khả năng xử lý thấp.

1.1.2. Đặc điểm nổi bật của NoSQL

- Khả năng mở rộng (Scalability): gần như không có giới hạn cho dữ liệu và người dùng hệ thống.
- Tính sẵn sàng (High Availability): Vì chấp nhận sự trùng lặp trong lưu trữ dữ liệu nên nếu một node bị chết sẽ không ảnh hưởng tới toàn bộ hệ thống.
- Tính nguyên tố (Atomicity): Độc lập trạng thái dữ liệu trong các thao tác.
- Tính nhất quán (Consistency): Chấp nhận tính nhất quán yếu, cập nhật mới không đảm bảo các truy xuất sau đó thấy được sự thay đổi. Sau một khoảng thời gian lan truyền thì tính nhất quán cuối cùng của dữ liệu mới được đảm bảo.
- Tính bền vững (Durability): Dữ liệu có thể tồn tại trong bộ nhớ máy tính, nhưng cũng đồng thời được lưu trữ tại đĩa cứng.

- Triển khai linh hoạt (Deployment Flexibility): Hệ thống sẽ tự động nhận biết việc bổ sung thêm hay loại bỏ các node. Hệ thống không đòi hỏi cấu hình phần cứng mạnh, đồng nhất.
- Mô hình hóa linh hoạt (Modeling flexibility): cặp dữ liệu key-value, dữ liệu cấu trúc (Hierarchical data), Graphs.
- Truy vấn linh hoạt (Query Flexibility): Multi-Gets, load một tập giá trị dựa vào tập khóa (Range queries)
- Khả năng mở rộng theo chiều ngang (Horizontal scalable): Bình thường, với các hệ quản trị cơ sở dữ liệu quan hệ, khi mà dữ liệu quá lớn phương pháp tăng khả năng lưu trữ là sẽ phải mở rộng (nâng cấp máy chủ), còn đối với NoSQL thì chỉ cần bổ sung thêm máy chủ khác vì hệ thống hỗ trợ lưu trữ phân tán trên nhiều máy.

1.2. So sánh NoSQL và RDBMs

Cả NoSQL và RDBMs đều là những công nghệ tuyệt vời, mỗi công nghệ đều hay ở một lĩnh vực khác, biết sử dụng linh hoạt 2 công nghệ này sẽ giúp ích và phát huy tối đa lợi ích mà nó đem lại.

1.2.1. Mặt tích cực của RDBMs

- RDBMs được hỗ trợ tốt với tính ACID đặc trưng.
  - Tính nguyên tố (Atomicity). Một giao dịch có nhiều thao tác khác biệt thì hoặc là toàn bộ các thao tác hoặc là không một thao tác nào được hoàn thành. Chẳng hạn việc chuyển tiền có thể thành công hay trục trặc vì nhiều lý do nhưng tính nguyên tố bảo đảm rằng một tài khoản sẽ không bị trừ tiền nếu như tài khoản kia chưa được cộng số tiền tương ứng.
  - Tính nhất quán (Consistency). Một giao dịch hoặc là sẽ tạo ra một trạng thái mới và hợp lệ cho dữ liệu, hoặc trong trường hợp có lỗi sẽ chuyển toàn bộ dữ liệu về trạng thái trước khi thực thi giao dịch.
  - Tính tách biệt (Isolation). Một giao dịch đang thực thi và chưa được xác nhận phải bảo đảm tách biệt khỏi các giao dịch khác.

- Tính bền vững (Durability). Dữ liệu được xác nhận sẽ được hệ thống lưu lại sao cho ngay cả trong trường hợp hỏng hóc hoặc có lỗi hệ thống, dữ liệu vẫn đảm bảo trong trạng thái chuẩn xác.

- Xử lý dễ dàng với độ chính xác cao trong việc giải quyết những thao tác lớn.

1.2.2. Mặt tích cực của NoSQL

- **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 thứ như: 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 hổ trợ rất tốt việc này đồng thời hiệu suất khi phân trang không hề giảm.
- **NoSQL là 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. Những lợi ích khác: 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.
- **Việc mở rộng phạm vi là mềm dẻo**: NoSQL giúp 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.

1.2.3. Các loại NOSQL phổ biến

- **RavenDB**: được viết trên C# bởi Hibernating Rhinos với giấy phép GNU AGPL v3.0. RavenDB là một giải pháp NoSQL trên nền tảng .NET được xây dựng dựa trên kiến trúc client-server. Dữ liệu được lưu trữ trên một thực thể máy chủ và những yêu cầu dữ liệu được gửi tới máy chủ này từ một hoặc nhiều máy người dùng khác nhau.
- **Hadoop**: là một framework nguồn mở viết bằng Java cho phép phát triển các ứng dụng phân tán có cường độ dữ liệu lớn một cách miễn phí. Nó cho phép các ứng dụ ng có thể làm việc với hàng ngàn node khác nhau và hàng petabyte dữ liệu. Hadoop lấy được phát triển dựa trên ý tưởng từ các công bố của Google về mô hình MapReduce và hệ thống file phân tán Google File System (GFS).
- **Cassandra:** là một hệ quản trị cơ sở dữ liệu nguồn mở, được viết bằng Java với mục tiêu chính là trở thành "Best of BigTable". Cassandra được thiết kế với khả năng xử lý một khối dữ liệu cực lớn được trải ra trên rất nhiều máy chủ trong khi cung cấp một dịch vụ có tính sẵn sàng cao và không hỏng. Nó là một giải pháp NoSQL bước đầu được phát triển bởi Facebook.
- **MongoDB**: là một cơ sở dữ liệu NoSQL nguồn mở, hiệu năng cao, có tính mở rộng cao. Được viết bằng C++ . Dùng cách lưu trữ BSON (Json được biên dịch) với giấy phép AGPL. Thay vì lưu trữ dữ liệu theo các bảng như các cơ sở dữ liệu cổ điển. MongoDB lưu cấu trúc dữ liệu thành các văn bản dựa JSON với mô hình động (gọi đó là BSON), khiến cho việc tích hợp dữ liệu cho các ứng dụng trở nên dễ dàng và nhanh hơn. Với mục tiêu là kết hợp các điểm mạnh của mô hình key-values (nhanh mà tính mở rộng cao) với mô hình dữ liệu quan hệ.

MongoDB được sử dụng tốt nhất với nhu cầu cần truy vấn động, nếu bạn muốn định nghĩa chỉ mục mà không cần các hàm map/reduce. Đặc biệt nếu bạn cần tốc độ nhanh cho một cơ sở dữ liệu lớn vì MongoDB ngoài tốc độ đọc nhanh ra thì tốc độ ghi của nó rất nhanh.

- **CouchDB**: được viết bằng Erlang với mục tiêu là tạo ra một cơ sở dữ liệu bền vững, chịu lỗi cao, dễ dàng trong việc sử dụng. Dùng cách lưu trữ thông thường là JSON với giấy phép Apache 2.0. Với CouchDB thì mỗi một cơ sở dữ liệu là một tập các văn bản riêng biệt. Trên mỗi văn bản còn bao gồm các thông tin về phiên bản, khiến cho việc dễ dàng đồng bộ các dữ liệu với nhau khi cơ sở dữ liệu bị mất kết nối một thời gian giữa các thiết bị...

1.2.4. Sự khác nhau giữa RDBMs và NoSQL

	RDBMs	**NoSQL**
Cấu trúc	Cấu trúc dựa trên các bảng	Cấu trúc phân làm 4 loại chính: • Document base • Key-value pair base • Graph database • Wide-column Store
Lược đồ	Lược đồ được định nghĩa để lưu trữ dữ liệu có cấu trúc	Không có một định nghĩa trước nào cho lược đồ, mà lược đồ linh hoạt dựa theo thành phần dữ liệu
Khả năng mở rộng	Được mở rộng theo chiều dọc, nếu chúng ta muốn mở rộng cơ sở dữ liệu thì phải nâng cấp phần cứng, điều này làm hạn chế khả năng mở rộng của CSDL quan hệ	Mở rộng theo chiều ngang, nghĩa là nếu muốn mở rộng cơ sở dữ liệu, chúng ta chỉ cần thêm các node và tạo ra mạng lưới phân phối dựa trên yêu cầu mà ta đưa ra, đây là cách giảm tải trên cơ sở dữ liệu quan hệ.
Các hệ cơ sở dữ liệu tiêu biểu	• Oracle • MySql • Postsgres • MS-SQL	• MongoDB • BigTable • Redis • RavenDb • Cassendra

		• Hbase • Neo4j • CouchDb
Phân loại	Phân làm hai loại chính là CSDL quan hệ mã nguồn mở và CSDL mã nguồn đóng.	Phân chia thành 5 loại dựa trên cách lưu trữ data: • Key-value pair store database • Graph database • Document Store • Column Store • XML Store
Khái niệm cơ bản	Dựa trên những quy luật và tính ACID	Dựa trên 3 tính chất: nhất quán, sẵn sàng và phân mảnh.

Bảng 1-1: Sự khác nhau giữa RDBMs và MongoDB

1.3. Ưu nhược điểm khi chuyển từ RDBMs sang NoSQL.

1.3.1. Ưu điểm

Với những ưu điểm của MongoDB được xem là những nhược điểm của các RDBMs:

- Cải thiện tốc độ xử lý dữ liệu lên nhiều lần.
- Tập trung vào các truy vấn có tần suất sử dụng cao giúp nâng cao hiệu suất.
- Mỗi dữ liệu được lưu thể hiện được sự trực quan.
- Khả năng mở rộng dữ liệu theo chiều ngang cao.
- Khả năng làm việc trực tiếp với Javascript.
- Dữ liệu trở nên linh hoạt nhờ cấu trúc linh hoạt JSON.

1.3.2. Nhược điểm

Dù có những ưu điểm khác nổi trội, tuy nhiên, việc chuyển đổi từ RDBMs sang NoSQL cũng gặp những mặt hạn chế sau:

- Không đảm bảo toàn vẹn dữ liệu (các tính chất ACID của dữ liệu RDBMs)
- Không có các ràng buộc khóa, tham chiếu giữa các thực thể.
- Không hỗ trợ view,transaction, procedure, trigger…

- Không hỗ trợ xử lý song song các truy vấn trên cùng một thể hiện.
- Đang trong quá trình phát triển và còn mới mẻ nên tính ổn định kém hơn các RDBMS

1.4. Các trường hợp nên sử dụng mô hình NoSQL

NoSQL sử dụng đến các mô hình dữ liệu quan hệ, vì thế ở những dữ liệu đòi hỏi sự toàn vẹn của dữ liệu cao như ngân hàng, chứng khoán, các hình thức giao dịch thương mại điện tử … , ở những dạng dữ liệu trên không nên sử dụng NoSQL vì nó không đảm bảo tính toàn vẹn của dữ liệu.

Ngược lại, ở những dữ liệu, mà tốc độ xử lý được ưu tiên cao hơn, ví dụ như quản lý các nội dung của trang mạng xã hội, quản lý thông tin sản phẩm … thì NoSQL sẽ là lựa chọn hàng đầu.

Vì thế, để tối ưu nhất, tùy theo từng ứng dụng cụ thể mà ta có thể sử dụng một trong hai hoặc kết hợp sử dụng cả RDBMs và NoSQL.

CHƯƠNG II. TÌM HIỂU MONGODB

MongoDB là một cơ sở dữ liệu (CSDL) mã nguồn mở, nó có cơ chế hoạt động với hiệu suất cao, dễ dàng cấu hình và dễ dàng mở rộng vì được xây dựng hoàn toàn trên ngôn ngữ C/C++.

MongoDB không có các lược đồ quan hệ giống như hệ cơ sở dữ liệu quan hệ (CSDLQH). Đặc biệt, MongoDB là một "Cơ sở dữ liệu hướng tài liệu".

MongoDB quản lý các dữ liệu thành các tập dạng BSON (Binary JSON), BSON là kiểu dữ đã được mã hóa nhị phân của dạng dữ liệu JSON. Các dữ liệu này có thể được lồng vào nhau, hình thành nên một cấu trúc phức tạp, nhưng vẫn dễ dàng truy xuất được nhờ được đánh các chỉ mục. Điều này cho phép dữ liệu được thiết kế tự nhiên và dễ dàng phù hợp cho ứng dụng.

2.1. Tóm tắt lịch sử

- Năm 2007, dự án MongoDB được thành lập bởi 10gen, tại New York, Mỹ.
- Năm 2009, MongoDB đã chính thức trở thành mã nguồn mở.
- Trong tháng 3 năm 2011, từ phiên bản 1.4, MongoDB đã được hoàn thiện phiên bản đầu tiên sẵn sàng cho các ứng dụng.
- Tới tháng 9 năm 2014, phiên bản 2.6.6 (12/9/2014) là phiên bản mới nhất.

2.2. Các khái niệm cơ bản trong MongoDB

2.2.1.Văn bản (Document)

MongoDB lưu trữ tất cả dữ liệu ở dạng văn bản, theo cấu trúc lưu trữ JSON có các trường và giá trị tương ứng.

Nó được xem tương đương với một dòng dữ liệu trong cơ sở dữ liệu quan hệ.

```
{"item": "pencil", "qty": 500, "type": "no.2"}
```

2.2.1.1.Định dạng văn bản (Document Format)

MongoDB lưu trữ các văn bản ở dạng BSON theo một cách tuần tự. BSON là cách biểu diễn nhị phân của các cấu trúc JSON.

2.2.1.2.Cấu trúc văn bản (Document Structure)

Văn bản MongoDB có cấu trúc bao gồm các trường (field) và các giá trị (value) tương ứng, theo cấu trúc sau:

```
{
field1: value1,
field2: value2,
field3: value3,
...
fieldN: valueN
}
```

Ví dụ:

```
{“greeting”: “Hello, world!”}
```

Văn bản trên gồm 1 trường là “greeting” với giá trị tương ứng là “Hello, world!”

Ví dụ:

```
{“greeting”: “Hello, world!”, “foo”: 3}
```

Các giá trị có thể ở bất cứ kiểu dữ liệu nào, có thể bao gồm các văn bản khác, dạng mảng, hoặc là các mảng văn bản.

```
varmydoc={
_id: ObjectId(“5099803df3f4948bd2f98391”),
name: {first: “Alan”,last: “Turing”},
birth: newDate(“Jun 23, 1912”),
death: newDate(“Jun 07, 1954”),
contribs: [“Turing machine”,“Turing test”,“Turingery”],
views: NumberLong(1250000)
}
```

Ví dụ văn bản trên có:

_id: đối tượng ObjectId.

name: có một văn bản con gồm các trường là “first” và “last” cùng các giá trị tương ứng.

birth và death: giá trị có kiểu dữ liệu dạng Date.

Contribs: giá trị có kiểu dữ liệu mảng của chuỗi.

Views: giá trị có kiểu dữ liệu Long interger.

2.2.1.3.Trường có một số quy luật phải được tuân thủ:

- Tên trường là một chuỗi ký tự.
- Trường có nội dung "_id" được dành riêng cho các khóa chính của văn bản, và cần được tuân thủ các điều kiện của khóa chính.
- Trường không thể được bắt đầu bằng ký tự "$".
- Trường không thể chứa dấu chấm ".".
- Trường phải là một chuỗi kí tự khác rỗng "null"
- Trong một văn bản, không thể chứa các trường giống nhau, ví dụ trường hợp sau là không hợp lệ:

```
{"greeting": "Hello, world!", "greeting": "Hello, MongoDB!"}
```

2.2.1.4.Trường khóa (_id) có một số ràng buộc sau

- Mặc định, MongoDB sẽ tự động tạo ra trường "_id" mỗi khi có một văn bản được tạo.
- Trường khóa luôn luôn là trường đầu tiên của văn bản. Nếu như hệ thống nhận một văn bản có trường "_id" không phải đứng đầu, thì hệ thống sẽ tự động chuyển trường "_id" lên đầu.

2.2.2. Bộ sưu tập (Collection)

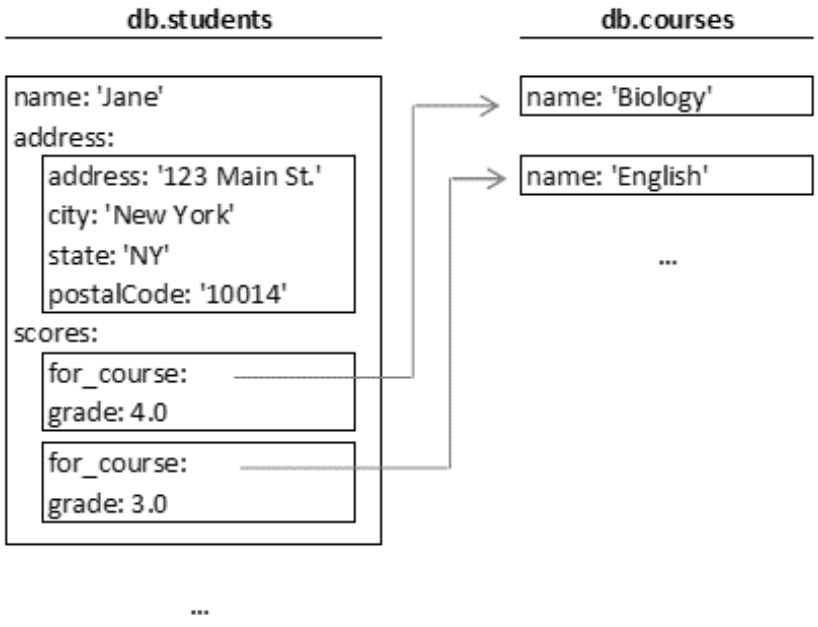
Bộ sưu tập là một nhóm các văn bản, nó được xem là tương đương với các dòng dữ liệu của một bảng trong cơ sở dữ liệu quan hệ.

Một bộ sưu tập được xem là một Schema-Free, nghĩa là các văn bản phải có cấu trúc giống nhau mới có thể lưu vào chung trong một bộ sưu tập.

Ví dụ các văn bản sau có thể được dùng để lưu trong một bộ sưu tập:

```
{"greeting": "Hello, world!"}
{"foo": 5}
```

Bộ sưu tập được xác định bởi tên của nó là một chuỗi UTF-8



Hình 2-1: Hình minh họa có 2 bộ sưu tậpstudents và courses.

Các văn bản student được nhúng văn bản address và văn bản score. Trong đó, văn bản Score được tham chiếu đến Courses. So sánh với lược đồ quan hệ: ta cần lưu Score vào bảng riêng và dùng khóa ngoài liên kết với Student.

2.2.3. Chỉ mục (Index)

Chỉ mục trong MongoDB hỗ trợ thực hiện các truy vấn một các hiệu quả. Nếu không có chỉ mục, MongoDB sẽ phải duyệt tất cả các tài liệu để lựa chọn ra những văn bản phù hợp nhất cho lệnh truy vấn. Cách này thường không hiệu quả vì nó xử lý một khối dữ liệu lớn mà không cần thiết.

Chỉ số là cầu trúc đặc biệt dùng để lưu trữ một phần nhỏ dữ liệu của bộ sưu tập. Chỉ số lưu trữ giá trị của một trường cụ thể hoặc được thiết lập, sắp xếp theo các giá trị của các trường.

Cơ bản, chỉ số trong MongoDB tương tự như trong các hệ thống dữ liệu khác. MongoDB định nghĩa các chỉ mục vào một bộ sưu tập cấp độ và hỗ trợ các trường hoặc các trường con trong văn bản trong bộ sưu tập dữ liệu MongoDB.

Nếu các chỉ số được sử dụng một cách thích hợp trong truy vấn, MongoDB có thể dùng các chỉ số để giới hạn lại số lượng các văn bản cần được kiểm tra. Trong một số trường hợp, MongoDB có thể dùng dữ liệu từ các chỉ số để xác định văn bản nào phù hợp với lệnh truy vấn.

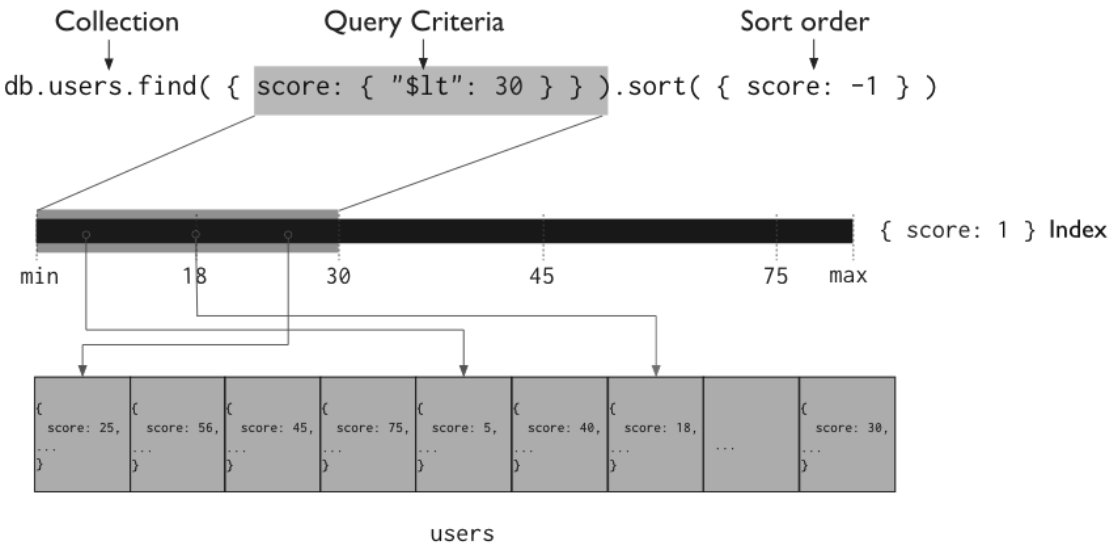
Sơ đồ dưới đây minh họa một truy vấn chọn tài liệu sử dụng một chỉ số.

Hình 2-2: Sơ đồ của một truy vấn văn bản sử dụng một chỉ số.

MongoDB thu hẹp phạm vi tìm kiếm văn bản bằng các duyệt tất cả văn bản có giá trị score nhỏ hơn 30.

2.2.3.1.Sắp xếp kết quả

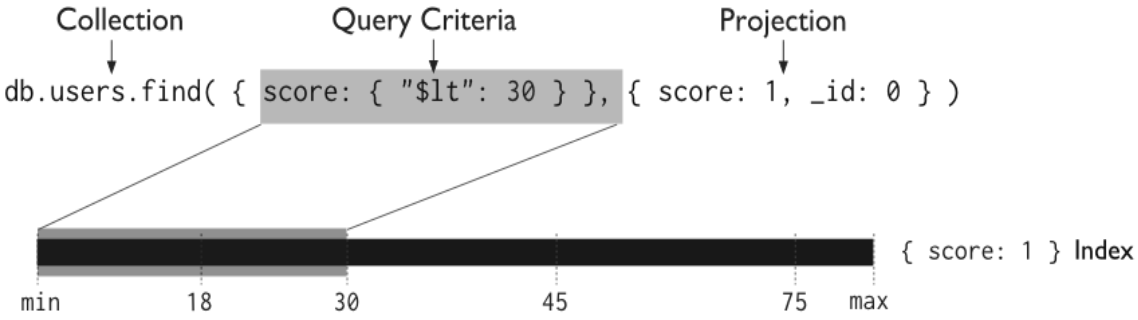
MongoDB có thể dùng các chỉ số để trả về các văn bản đã được sắp xếp bằng các chỉ mục một các trực tiếp mà không cần phải thêm một bước sắp xếp trung gian.



Hình 2-3: Sơ đồ sử dụng chỉ số để truy vấn và sắp xếp tăng dần của "score". MongoDB có thể dùng các chỉ số để duyệt theo thứ tự tăng dần hoặc giảm dần và trả về kết quả đã được sắp xếp.

2.2.3.2.Tối ưu kết quả truy xuất

Khi câu truy vấn hợp lệ với yêu cầu một chỉ số cụ thể, MongoDB sẽ trả về trực tiếp kết quả tương ứng với chỉ số mà không cần phải duyệt hay ghi các văn bản vào bộ nhớ. Điều này giúp cho hệ thống hoạt động hiệu quả hơn.



Hình 2-4: Sơ đồ của truy vấn chỉ sử dụng chỉ số để truy vấn.

MongoDB không cần phải kiểm tra dữ liệu bên ngoài của các chỉ số để thực hiện các truy vấn.

2.3.Các thao tác cơ bản với MongoDB

2.3.1.Thao tác thêm văn bản[i]

Phương thức thêm document vào trong collection là thao tác cơ bản trong MongoDB.

Để thực hiện thêm, ta thực hiện theo cú pháp sau:

```
[tên_CSDL].[tên_collection].save(x);
```

Hoặc

```
[tên_CSDL].[tên_collection].insert(x);
```

Trong đó: x là một document hay là một mảng bao gồm nhiều document

Ví dụ:

```
db.foo.insert({"bar": "baz"})
```

Hệ thống sẽ tự thêm trường "_id" vào document (nếu như nó không được khai báo).

2.3.2.Thao tác xóa document, collection

Để xóa tất cả các document trong 1 collection

Ta sử dụng cú pháp sau để xóa:

```
[tên_CSDL].[tên_collection].remove(x);
```

Trong đó, x là tham số của nó là một document hay là một cú pháp được trả về là document, thì phương thức sẽ xóa các document đó trong collection.

Nếu x = {}: (Rỗng), thì collection sẽ xóa hết tất cả các document bên trong nó.

Ví dụ:

```
db.mailing.remove({"opt-out": true})
```

Khi dữ liệu đã được xóa, sẽ không có cách nào để phục hồi lại dữ liệu đã bị xóa.

2.3.3.Thao tác cập nhật

Một document đã được lưu trong collection, nó vẫn có thể thay đổi giá trị thông qua phương thức

```
[tên_CSDL].[tên_collection].update(x,y);
```

Phương thức gồm có 2 tham số:

x là tham số để xác định document cần thay đổi

y là tham số cập nhật là document đó.

Nếu như 2 phương thức cùng đến trong khoảng thời gian tương tự nhau. Cái nào đến trước sẽ được thực thi trước, đến sau sẽ được thực thi sau.

2.3.4.Thao tác truy vấn

Phương thức find() là loại truy vấn phổ biến nhất trong MongoDB. Nó trả về tập hợp document trong collection. Cách sử dụng như sau:

```
> db.collection.find(x)
```

X: là chuỗi các biểu thức xác định phương thức trả về document.

Những document được xác định trả về phụ thuộc vào tham số của phương thức.

Nếu như x được để trống, thì nó sẽ trả về tất cả những gì có trong collection, mặc định của tham số x sẽ là {} (nghĩa là rỗng), ví dụ như:

```
> db.c.find({})
```

Hoặc

```
> db.c.find()
```

Hai cách biểu diễn phương thức find() trên hoàn toàn tương tự nhau, nó sẽ trả về mọi thứ trong collection c.

Khi muốn truy vấn dữ liệu với một giá trị cần tìm kiếm. Ví dụ, khi cần tìm một Users có giá trị “age” là 27, chúng ta có thể điền điều kiện này vào trong tham số của câu truy vấn.

```
> db.Users.find({“age”: 27})
```

Nếu như muốn truy vấn dựa trên một giá trị dạng chuỗi. Ví dụ, tìm một Users có trường "Username" với giá trị là "joe", ta làm như sau:

```
> db.Users.find({"Username": "joe"})
```

Nếu muốn truy vấn document dựa trên nhiều giá trị trên các trường khác nhau, ta chỉ việc đặt các giá trị thành thứ tự các tham số của phương thức. Ví dụ, ta cần tìm Users có trường "age" là 27 và "Username" là "joe", ta làm như sau:

```
> db.Users.find({"Username": "joe", "age": 27})
```

2.3.5.Làm việc với chỉ số (Index)

Để tăng tốc độ xử lý của các câu lệnh, người ta thường đánh thêm chỉ số index vào trong các trường, Tuy nhiên, với một dữ liệu lớn thì việc đánh chỉ số cho document cho toàn bộ collection tốn khoảng vài phút.

Giả sử ta câu lệnh cho 1 trường của document như sau:

```
> db.people.find({"Username": "mark"})
```

Khi chỉ có một trường được sử dụng trong câu lệnh, trường này có thể được đánh chỉ số index để tăng tốc độ xử lý của câu lệnh. Trong trường hợp này, chúng ta có thể tạo index cho trường "Username". Để làm được điều này, ta thực hiện như sau:

```
> db.people.ensureIndex({"Username": 1})
```

Chỉ số của một trường chỉ thêm được một lần cho collection. Nếu chung ta tạo một lần nữa với 1 index tương tự, sẽ không có điều gì xảy ra nữa.

Chỉ số Index của một trường sẽ khiến cho câu lệnh của nó chạy nhanh hơn, tuy nhiên, không phải tất cả các câu lệnh đều nhanh hơn, mặc dù nó có chứa chỉ số Index. Ví dụ, câu lệnh dưới đây không nhanh hơn với chỉ số Index được tạo phía trên.

```
> db.people.find({"date": date1}).sort({"date": 1, "Username": 1})
```

Với câu lệnh này, mặc dù trường "Username" đã sở hữu index, nhưng "Username" lại nằm ở phương thức "sort" cùng với trường "date" nên index không phát huy được tác dụng của nó. Đầu tiên, server cần phải duyệt tất cả các document có trong collection để tìm thấy điều kiện so khớp của "date" trong phương thức "find", vì trường "date" không có Index, điều này sẽ được thực khi rất chậm nếu số lượng document trong collection lớn.

Để giải quyết được tình trạng trên, chúng ta nên tạo nên những chỉ số index bao gồm tất cả các trường trong câu lệnh. Trong trường hợp này, để tối ưu câu truy xuất trên, chúng ta cần tạo index bao gồm cả "date" và "Username" như sau:

```
> db.ensureIndex({"date": 1, "Username": 1})
```

Các đối số truyền vào phương thức tạo index ensureIndex() cũng tương tự như đối số truyền vào phương thức sort(), giá trị của mỗi trường là 1 hoặc -1, tùy vào chiều tăng giảm của các giá trị trường mà thiết đặt, 1 là tăng dần, -1 là giảm dần.

Nếu như chúng ta có nhiều hơn 1 trường thể đánh chỉ mục index, ta cần nghĩ đến thứ tự ưu tiên sắp xếp, ví dụ ta có 1 collection như sau:

```
{"_id": ..., "Username": "smith", "age": 48, "Users_id": 0}
{"_id": ..., "Username": "smith", "age": 30, "Users_id": 1}
{"_id": ..., "Username": "john", "age": 36, "Users_id": 2}
{"_id": ..., "Username": "john", "age": 18, "Users_id": 3}
{"_id": ..., "Username": "joe", "age": 36, "Users_id": 4}
{"_id": ..., "Username": "john", "age": 7, "Users_id": 5}
{"_id": ..., "Username": "simon", "age": 3, "Users_id": 6}
{"_id": ..., "Username": "joe", "age": 27, "Users_id": 7}
{"_id": ..., "Username": "jacob", "age": 17, "Users_id": 8}
{"_id": ..., "Username": "sally", "age": 52, "Users_id": 9}
{"_id": ..., "Username": "simon", "age": 59, "Users_id": 10}
```

Nếu ta thiết lập index với các thông số như sau {"Username": 1, "age": -1}. MongoDB sẽ tổ chức lại cho chúng ta như sau:

```
{"_id": ..., "Username": "jacob", "age": 17, "Users_id": 8}
{"_id": ..., "Username": "joe", "age": 36, "Users_id": 4}
{"_id": ..., "Username": "joe", "age": 27, "Users_id": 7}
{"_id": ..., "Username": "john", "age": 36, "Users_id": 2}
{"_id": ..., "Username": "john", "age": 18, "Users_id": 3}
```

```
{“_id”: ..., “Username”: “john”, “age”: 7, “Users_id”: 5}
{“_id”: ..., “Username”: “sally”, “age”: 52, “Users_id”: 9}
{“_id”: ..., “Username”: “simon”, “age”: 59, “Users_id”: 10}
{“_id”: ..., “Username”: “simon”, “age”: 3, “Users_id”: 6}
{“_id”: ..., “Username”: “smith”, “age”: 48, “Users_id”: 0}
{“_id”: ..., “Username”: “smith”, “age”: 30, “Users_id”: 1}
```

Trường “Username” được sắp xếp theo thứ tự alphabet tăng dần, nếu như có một nhóm tên trùng nhau, thì “age” sẽ được sắp xếp giảm dần. Sự sắp xếp tăng dần hay giảm dần phụ thuộc vào thông số được thiết lập trong câu lệnh tạo Index.

Chỉ số index cho 2 trường “Username” và “age” cũng làm cho câu lệnh của một trường “Username” nhanh hơn.

Theo nguyên tắc, nếu chỉ số index có N trường, nó sẽ giúp cho các cậu lệnh truy vấn có các tiền tố của index chạy nhanh hơn. Ví dụ, nếu như ta có chỉ số index có các trường {“a”: 1, “b”: 1, “c”: 1, ..., “z”: 1}, có cũng có chức năng như các index {“a”: 1}, {“a”: 1, “b”: 1}, {“a”: 1, “b”: 1, “c”: 1}, hoặc tương tự là tiền tố của index được tạo. Nhưng các câu lệnh có các index có các trường không theo đúng thứ tự {“b”: 1}, {“a”: 1, “c”: 1}, hoặc tương tự sẽ không có tác dụng (không là tiền tố của index được tạo). Chỉ có các index là tiền tố của index được tạo mới có tác dụng.

Nhược điểm của việc sử dụng index là sẽ phát sinh chi phí trong các trình insert, update, remove. Điều này xảy ra là do database không chỉ thực hiện các câu lệnh trên mà còn phải thực hiện cập nhật lại index sau khi thực thi các câu lệnh trong collection.

2.3.5.1.Chỉ số index cho các document con

Chỉ số index có thể được tạo cho các trường của các document con, cách tạo cũng tương tự đối với những trường thông thường. Ví dụ, nếu chúng ta muốn tìm những bình luận trong document blog theo ngày tháng, ta co thể tạo chỉ số index cho trường “date” trong document con của document “comments”, ta thực hiện như sau:

```
> db.blog.ensureIndex({“comments.date”: 1})
```

Chỉ số index của trường trong document được xếp ngang hàng với index của doccument cha.

2.3.5.2.Khai báo tên cho Index

Với mỗi index trong document đều có những chuỗi ký tự “name” lưu tên, và những chuỗi ký tự không trùng nhau. Nó được server dùng để xác định chính xác index cần xử lý. Theo mặc định, chỉ số index được sử dụng tên như sau:

```
keyname1_dir1_keyname2_dir2_..._keynameN_dirN
```

trong đó keynameX là tên của trường và dirX là chiều chỉ số index của trường tương ứng (1 hoặc -1). Chỉ số index với chuỗi tên mặc định như vậy, vì thế, chúng ta có thể thiết đặt thông số này trong đối số của hàm ensureIndex như sau:

```
> db.foo.ensureIndex({“a”: 1, “b”: 1, “c”: 1, ..., “z”: 1},
{“name”: “alphabet”})
```

Với tên được người sử dụng đặt, sẽ dễ dàng thực thi hơn thông qua tên của Index do chính người dùng đặt.

2.3.5.3.Giá trị duy nhất (Unique Index)

Unique đảm bảo rằng với mỗi giá trị của trường được thêm, với mỗi document trong collection thì giá trị đó là duy nhất. Ví dụ, nếu chúng ta muốn đảm bảo rằng, không có giá trị trùng lặp trong trường “Username” trong các documents trong collection “people”, ta thực hiện tạo unique index như sau:

```
> db.people.ensureIndex({“Username”: 1}, {“unique”: true})
```

Theo mặc định, khi thêm 1 document, MongoDB kiểm tra chỉ cần câu lệnh hợp lệ cú pháp, nó sẽ được thêm vào. Tuy nhiên sau khi thêm unique index, trước mỗi khi thêm 1 document, MongoDB sẽ kiểm tra xem trên những giá trị trên unique index có bị trùng lắp hay không, nếu có thì báo lỗi cho người sử dụng, điều này hoạt động tương tự với trường "_id" collection.

2.3.5.4.Index loại bỏ trùng lắp (Dropping Duplicate)

Khi tạo index cho một collection đã tồn tại trước đó, có thể có những giá trị bị trùng lắp. Nếu có bất kì trường trong unique index được tạo có giá trị bị trùng lắp, câu lệnh sẽ báo lỗi, khi đó, có thể bạn muốn loại bỏ đi tất cả các document có trường giá trị bị trùng lắp, việc làm này có thể được thực hiện một cách tự động. Thông số "dropDups" trong câu lệnh "ensureIndex" có thể giúp chúng ta làm việc này, nó sẽ giữ lại document đầu tiên hợp lý, sau đó, nó sẽ loại bỏ bất kỳ các document có giá trị trường bị trùng lắp trước đó:

```
> db.people.ensureIndex({"Username": 1}, {"unique": true, "dropDups": true})
```

Cách này có thể gây thất thoát dữ liệu, vì vậy cần xem xét kỹ nếu xử lý trên các dữ liệu quan trọng.

CHƯƠNG III. CHUYỂN ĐỔI LƯỢC ĐỒ RDBMS SANG NOSQL

3.1.Tổng quan mô hình quan hệ cho dữ liệu

Điều cơ bản nhất trong việc chuyển đổi dữ liệu quan hệ trang lược đồ NoSQL là mô hình hóa lược đồ cho dữ liệu.

Lược đồ dữ liệu của 2 kiểu dữ liệu là khác nhau, tuy nhiên giữa chúng cũng có một số điểm chung mà chúng ta cũng có thể xem xét để áp dụng vào nhau.

Bảng dữ liệu dưới đây cung cấp cho chúng ta một số các tham chiếu thuật ngữ giữa các thành phần RBDMs và NoSQL.

RBDMS	**MONGODB**
Database	Database
Table	Collection
Row	Document
Index	Index
Join	Embedded Document hoặc Reference

Bảng 3-1: Bảng ánh xạ các thành phần giữa RDBMs và NoSQL

Việc mô hình hóa khi chuyển từ RDBMs sang MongoDB đòi hỏi chúng ta cần phải biết áp dụng những điều kiện thực thể để thay đổi cấu trúc của lược đồ phù hợp.

Có 2 cách phương pháp để tổ chức mô hình lược đồ từ RDBMs sang MongoDB:

- Tận dụng lại các cấu trúc lược đồ dữ liệu quan hệ và cách tổ chức dữ liệu của MongoDB hướng kiểu 2-d (2-dimensional) của các hàng và cột trong RDBMs.
- Tổ chức dữ liệu theo hướng nhúng văn bản đơn hoặc trên mảng (embedded sub-documents and arrays).

3.1.1.Sự linh hoạt của dữ liệu JSON/BSON.

Hầu hết các dữ liệu có các thành phần phức tạp hiện nay đều cần phải được mô hình và biểu diễn một cách có hiệu quả bằng việc lưu trữ dưới dịnh dạng JSON (JavaScript Object Notation), điều này tốt hơn so với lưu trữ theo dạng bảng.

MongoDB lưu trữ các tài liệu JSON này theo dạng nhị phân được gọi là BSON (Binary JSON). BSON mã hóa tất cả các kiểu dữ liệu mà JSON lưu trữ.

Với các document, các document con và các mảng dữ liệu, JSON sẽ sắp xếp theo cấu trúc của đối tượng theo các cấp bậc, việc này sẽ giúp cho người khai thác dữ liệu dễ dàng ánh xạ đến dữ liệu và tổ chức dữ liệu trong ứng dụng.

Ngược lại, việc tổ chức dữ liệu theo dạng như các bảng trong RDBMs không thuận lợi cho các nhà lập trình. Việc thêm các đối tượng ORMs (Object Relational Mappers) có thể khiến cho dữ liệu trở nên phức tạp hơn và giảm sự linh hoạt trong việc triển khai các lược đồ, cũng như các câu lệnh khai thác dữ liệu theo yêu cầu của ứng dụng.

Công việc đầu tiên nên được bắt đầu bằng việc thiết kế nên các hướng xử lý dữ liệu dựa theo yêu cầu của ứng dụng. Nó nên được ưu tiên thiết kế để tận dụng nhưng ưu điểm linh hoạt trong MongoDB. Trong việc chuyển đổi, công việc này có thể dễ dàng hơn bằng việc gióng theo mô hình lược đồ dữ liệu quan hệ có sẵn qua cấu trúc của document MongoDB. Tuy nhiên, việc làm này sẽ không khai thác được các ưu điểm bởi vì chúng ta sẽ có rất nhiều các cấu trúc document con khi gióng từ RBDMs sang MongoDB. Ví dụ, trong RDBMs, một dữ liệu có thể có ràng buộc với 2 bảng khác nhau, khi gióng sang MongoDB, nó sẽ sinh ra 2 đối tượng con giống nhau nhưng ở 2 field khác nhau trong cùng một đối tượng cha.Liên quan đến vấn đề này, ta có thể áp dụng cách thức tương như như RBDMs đã làm, đó là cách thức tham chiếu trong NoSQL.

Ta có ví dụ dưới đây:

PERSON

Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rome

CAR

Car_ID	Model	Year	Value	Pers_ID
101	Bently	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

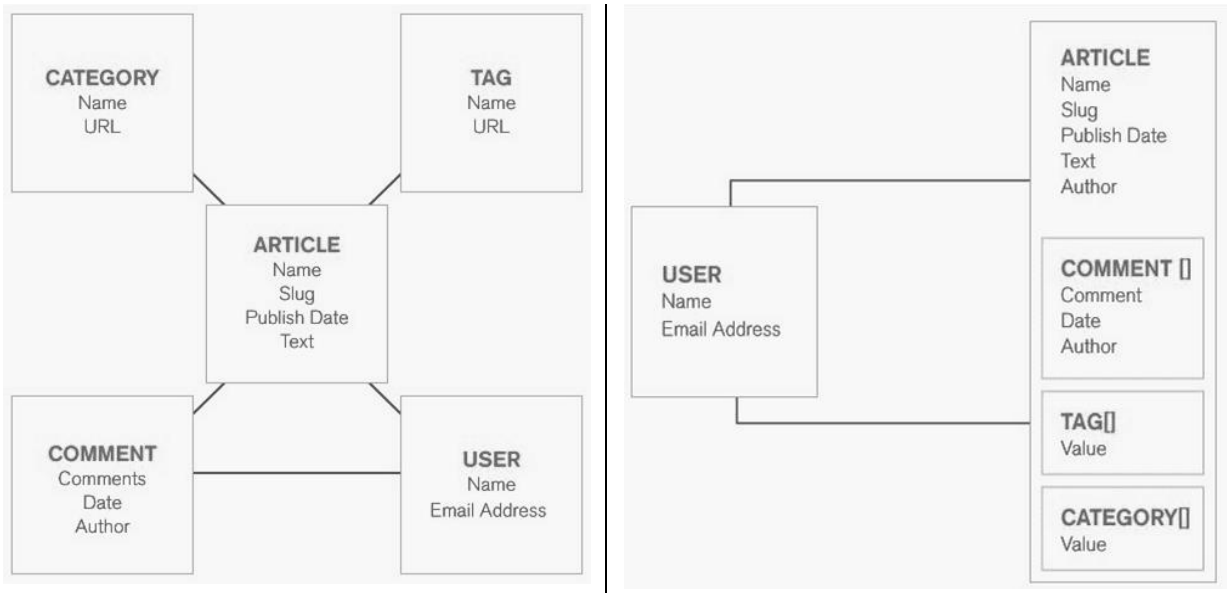
Trong ví dụ này, ta có một số dữ liệu mẫu của RDBMs, bảng "CAR" dùng giá trị của trường "Pers_ID" để tham chiếu JOIN với bảng "PERSON". Trong trường hợp này, khi chuyển sang lược đồ của MongoDB, phương pháp nhúng các document con vào trong một mảng sẽ phát huy tác dụng, ta cần tổ chức các dữ kiện document liên quan với nhau vào một cấu trúc mảng. Các dữ liệu hàng và cột trong RDBMs được phân bố rời rạc, nhưng khi được chuyển sang MongoDB sẽ có một cấu trúc trong 1 document nhất định.

Dưới đây là tổ chức dữ liệu của MongoDB sau khi được chuyển đổi.

```
{
first_name: “Paul”,
surname: “Miller”,
city: “London”,
location: [45.123,47.232],
cars: [
{ model: “Bentley”,
year: 1973,
value: 100000, ….},
{ model: “Rolls Royce”,
year: 1965,
value: 330000, ….},
]
}
```

Việc tổ chức dữ liệu như trên sẽ giúp cho dữ liệu được hiển thị một cách trực quan và tự nhiên, nhà lập trình cũng sẽ dễ dàng khai thác hơn.

Để hiểu rõ hơn về sự khác nhau, giữa lược đồ quan hệ và lược đồ MongoDB, ta cùng đi so sánh ví dụ dưới đây về các tích tổ chức dữ liệu trên 2 lược đồ của 2 nền tảng dữ liệu khác nhau:



RDBMs MongoDB

Trong ví dụ này, ta thấy rằng, ở RDBMs đã tổ chức dữ liệu ở 5 bảng rời rạc khác nhau để xây dựng nên một hệ thống dữ liệu. Trong khi đó, với MongoDB, dữ liệu được tổ chức kết hợp lại với nhau một document, và sử dụng tham chiếu đến Users trong trường "Author" trong document "Article" và "Comment".

3.1.2.Những ưu điểm khác của mô hình nhúng văn bản

Ngoài những ưu thế về biểu diễn dữ liệu được tự nhiên ở các cấp độ khác nhau, việc mô hình như liệu mô hình nhúng văn bản còn giúp tăng hiệu suất, và tính linh động cao:

- Việc kết hợp các dữ liệu lại với nhau trong một document, việc này giúp thay thế cho công đoạn JOIN nhiều bảng lại với nhau khi khai thác dữ liệu. Document trong MongoDB sẽ tập hợp tất cả các dữ liệu cần thiết, và những Dữ liệu này chỉ được khai thác trong 1 lần đọc từ bộ nhớ. Trong khi đó, cơ chế JOIN của RDBMs cần phải đọc dữ liệu từ nhiều bảng, và sau đó JOIN chúng lại với nhau thông qua các khóa ngoại, việc này tương tương đòi hỏi phải thực thi nhiều lần đọc khác nhau.
- Document trong MongoDB với khả năng lưu trữ độc lập, có khả năng phân tán thành các node nhỏ hơn (kỹ thuật dữ liệu phân tán). Chúng ta có thể dễ dàng co giãn theo chiều ngang để tiện lợi cho việc phục vụ các yêu cầu riêng của ứng dụng. Người phát triên DBA sẽ không còn lo lắng đến các lỗi khi thực thi với các phương thức JOIN (những lỗi điều này có thể gặp khi làm việc trên RDBMs).

3.2.Biểu diễn quan hệ trong MongoDB [1]

Các loại mô hình trong quan hệ của MongoDB đó là nhúng (Embed) và tham chiếu (Reference).

Việc quyết định mô hình bằng nhúng văn bản hay tạo những liên kết ở những document rời rạc khác nhau cần phải có sự cân nhắc kỹ lưỡng và dựa vào từng yêu cầu của ứng dụng.

Ngoài 2 phương pháp biểu diễn quan hệ cơ bản là nhúng và tham chiếu, nếu kết hợp giữa 2 phương pháp này ta sẽ có thêm 1 số phương pháp con khác. (Được trình bày trong phần 0)

3.2.1.Nhúng văn bản con (embed sub-document)

Hình 3-1: Ví dụcấu trúc nhúng văn bản

Dữ liệu với các quan hệ số lượng 1: 1 hoặc 1: n được xem là phù hợp để biểu diễn dữ liệu theo dạng nhúng trong cùng một document. Những dữ liệu thuộc dạng độc nhất hay thể hiện tính chất riêng của document cũng có thể áp dụng dạng dữ liệu kiểu nhúng. Ví dụ, trong dữ liệu của một document người dùng sau đây, tên người dùng –các thông tin cá nhân của người dùng - lịch sử làm việc - ..., những thông tin này thích hợp để được lưu trữ dưới dạng nhúng văn bản.

Tuy nhiên, không phải tất cả các quan hệ 1: 1 đều thích hợp để sự dụng văn bản nhúng. Khi xảy ra nên dùng phương pháp tham chiếu trong những trường hợp sau:

Document luôn được truy xuất thường xuyên, tuy nhiên văn bản nhúng lại hiếm hoặc ít khi được truy xuất và sử dụng đến. Việc này sẽ làm cho bộ nhớ yêu cầu cần thiết để đọc dữ liệu từ collection trong vòng 1 lần sẽ tăng lên. Ví dụ, trong document “Customer” nhúng 1 document con về 1 văn bản báo cáo hàng năm (có nghĩa trong vòng 1 năm mới truy xuất và sử dụng đến nó).

- Một phần nào đó của document luôn được cập nhật thường xuyên và dung lượng của document luôn tăng đều, trong khi đó một phần còn lại của document là không thay đổi.

- Khi kích thước của một document trong MongoDB vượt quá giới hạn 16MB (hoặc có nhiều hơn 100 cấp của document).

Đối với các bất kỳ phương pháp nào có sử dụng nhúng văn bản con thì đều có những ưu và nhược điểm sau đây.

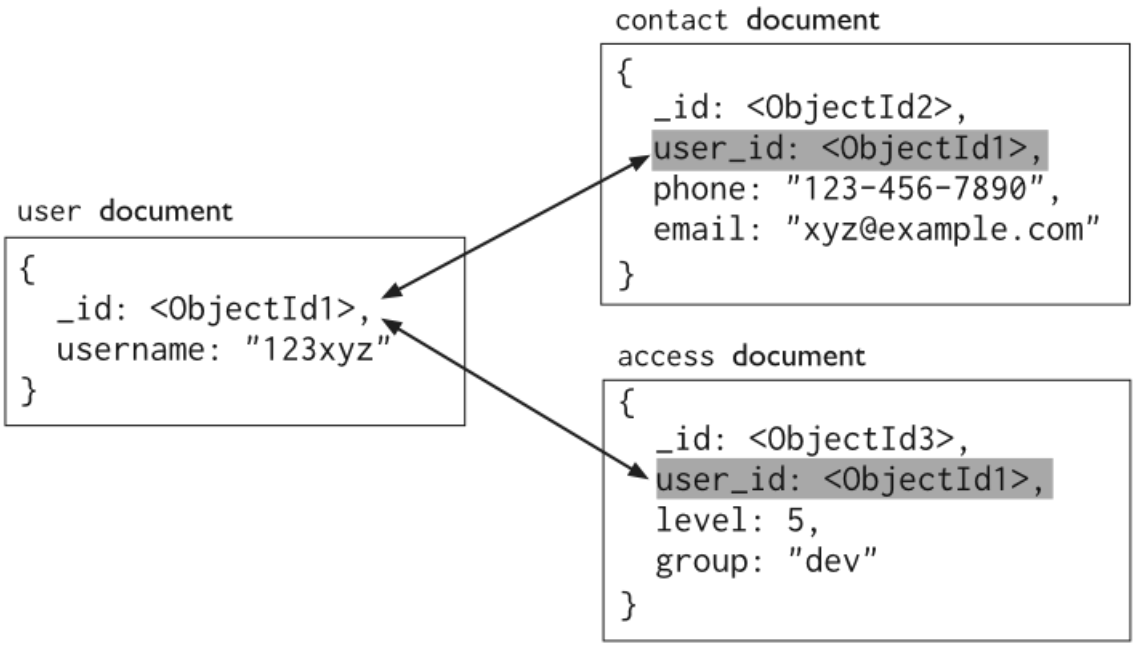
Ưu điểm

- Dữ liệu trở nên trực quan, dễ quản lý.
- Tất cả dữ liệu cần thiết đều được trả về trong 1 truy vấn duy nhất.

Nhược điểm

- Tốn bộ nhớ.
- Các văn bản nhúng phải được truy xuất thông qua văn bản cha.
- Tốc độ truy vấn sẽ bị giới hạn khi điều kiện truy vấn thuộc văn bản nhúng.

3.2.2.Tham chiếu document (Referencing)



Hình 3-2: Ví dụ cấu trúc tham chiếu văn bản

Kiểu dữ liệu tham chiếu là kiểu dữ liệu chuẩn trong RDBMs, tuy nhiên nó vẫn có thể triển khai được trong Mongodb. Nó cũng có một số đặc điểm giúp cho dữ liệu trở nên linh hoạt hơn kiểu dữ liệu nhúng. Tuy nhiên, nhược điểm của nó là khi sử dụng những câu lệnh để truy vấn, câu lệnh sẽ trở nên phức tạp, đồng thời yêu cầu nhiều yêu cầu truy vấn đến server, làm ảnh hưởng đến tốc độ xử lý của hệ thống.

Thông thường, trong MongoDB, kiểu dữ liệu tham chiếu sử dụng giá trị trong trường _id trong document được tham chiếu, và document tham chiếu cũng sẽ có 1 trường lưu giá trị này để thực hiện tham chiếu. Khi muốn truy xuất đối tượng ra cần phải thực hiện truy vấn lần để tham chiếu các dữ liệu lại với nhau.

Ngoài những trường hợp đã được nêu ở phần trên, kiểu dữ liệu tham chiếu còn nên được sử dụng khi:

- Khi một document được tham chiếu đến nhiều document khác nhau.
- Để giải quyết các mối quan hệ phức tạp n: n, n: 1.
- Khi kiểu dữ liệu nhúng không thể đáp ứng được yêu cầu của bộ nhớ, khi kích cỡ của document quá lớn có thể ảnh hưởng đến hiệu suất.

Đối với phương pháp tham chiếu sẽ có những ưu điểm và nhược điểm sau.

Ưu điểm

- Dữ liệu được tổ chức riêng lẻ, ta có thể truy cập trực tiếp từng thành phần.
- Tiết kiệm bộ nhớ.

Nhược điểm

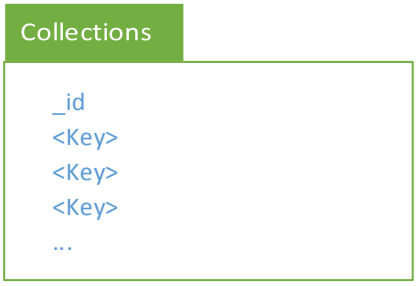
- Các dữ liệu cần phải thực hiện nhiều truy vấn để lấy được các dữ liệu tham chiếu với nhau.
- Các dữ liệu được tổ chức rời rạc và không trực quan, khó quản lý.

3.3.Cách biểu diễn lược đồ trong MongoDB

3.3.1.Biểu diễn thực thể, mối kết hợp trên lược đồ

3.3.1.1.Cấu trúc thực thể cơ bản trên lược đồ

Lược đồ thực thể cơ bản trong MongoDB về cơ bản cũng giống như lược đồ thực thể trong RDBMs hay còn gọi là bảng (table), lược đồ thực thể MongoDB hay còn gọi là bộ sưu tập (collection) có cấu trúc chung như sau:



Lược đồ 3-1: Cấu trúc lược đồ cơ bản thực thể MongoDB

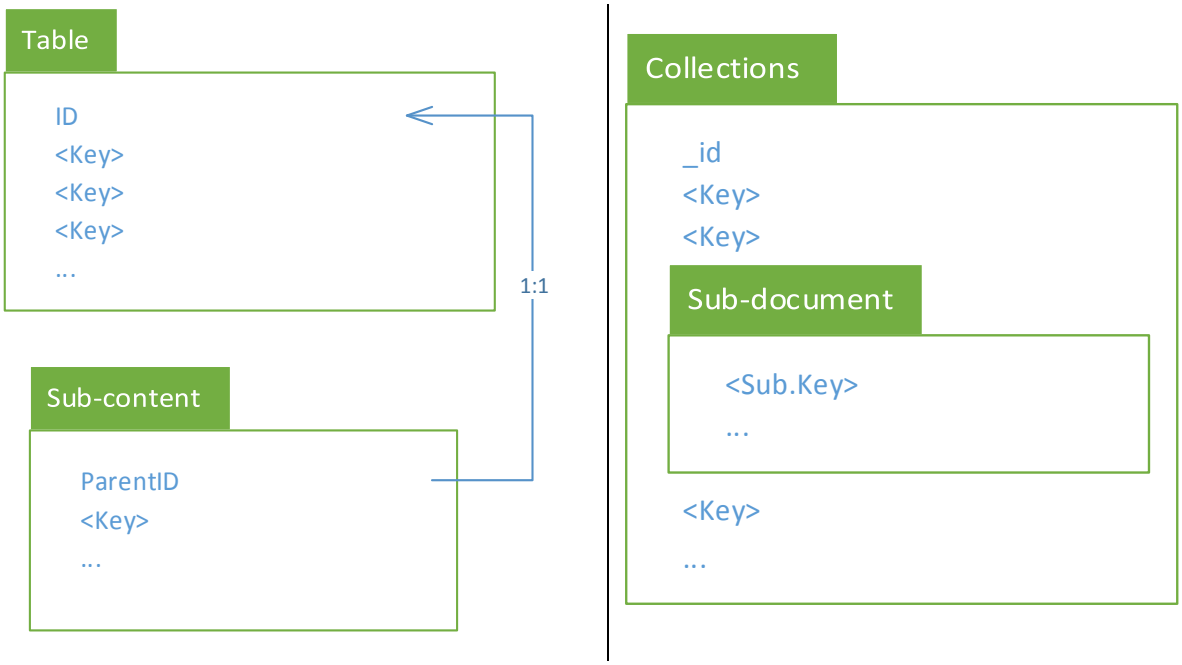
_id sẽ luôn là thuộc tính đứng đầu tiên của thực thể collection.

<Key> là thuộc tính của thực thể.

Khi được lưu trong 2.2.1.Văn bản (Document), MongoDB sẽ tổ chức dữ liệu này theo cấu trúc JSON.

3.3.1.2.Cấu trúc lược đồ nhúng văn bản (đơn)

Với các văn bản con được nhúng vào một văn bản cha, lược đồ được thể hiện như sau:

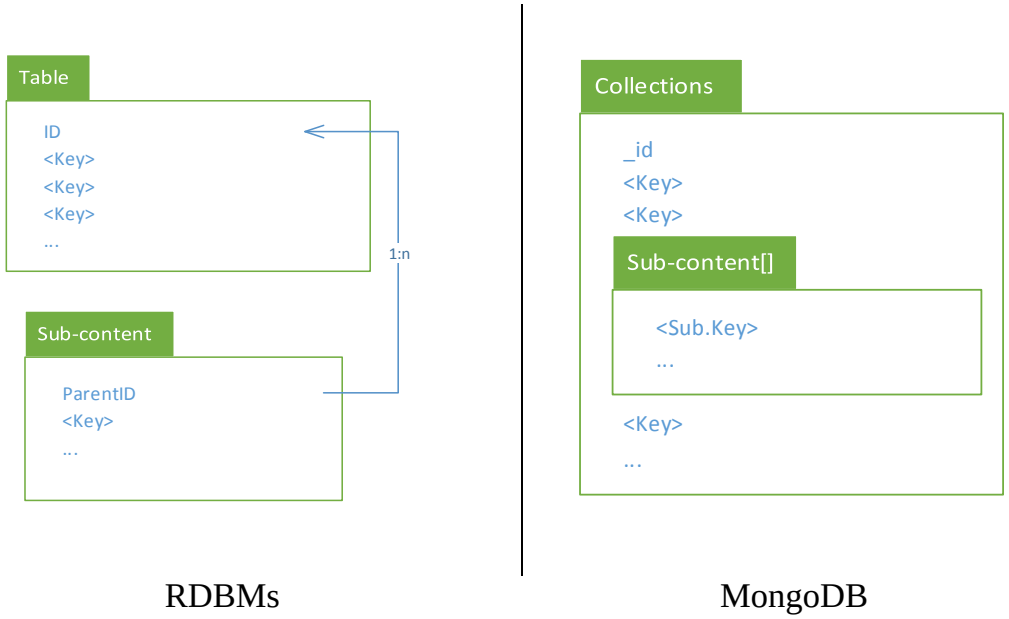


Lược đồ 3-2: Cấu trúc lược đồ văn bản nhúng trong MongoDB

Văn bản con (Sub-Document) được nhúng vào ngang cấp với các trường <Key>, cấu trúc của document con được nhúng vào sẽ là có trường trong <Sub.key> …, các thuộc tính của văn bản con sẽ chỉ được truy xuất thông qua văn bản con tương ứng. Phương pháp này đặc biệt chỉ dùng với các quan hệ số lượng 1:1.

3.3.1.3.Cấu trúc lược đồ nhúng mảng văn bản

Với văn bản con là tập các trường giá trị, kiểu nhúng mảng được thể hiện trên lược đồ như sau:

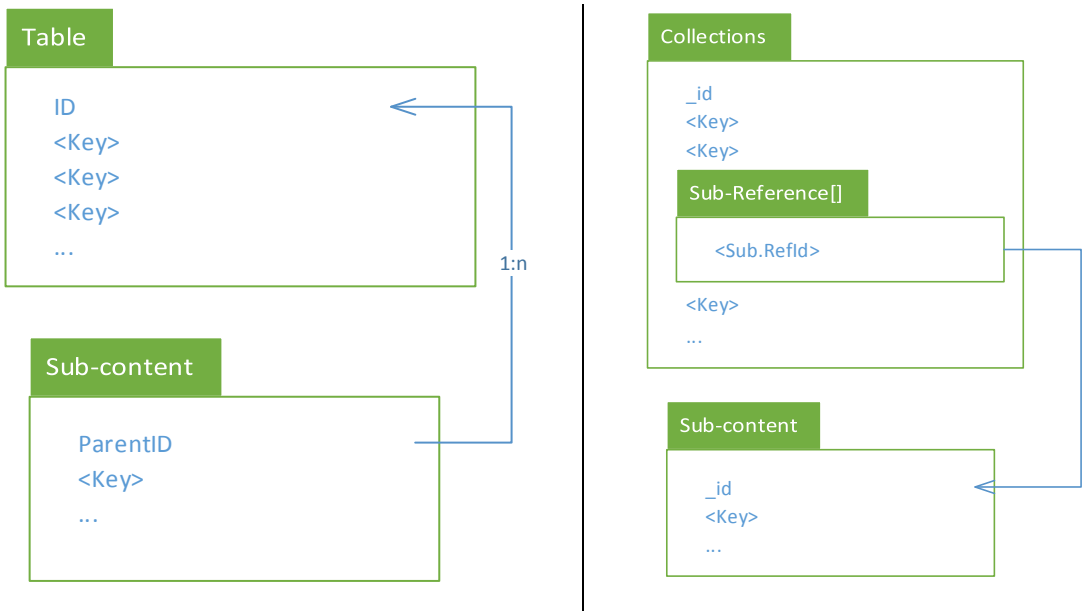


Lược đồ 3-3: Cấu trúc lược đồ nhúng mảng văn bản trong MongoDB

Tương tự với cấu trúc nhúng văn bản, cấu trúc nhúng mảng văn bản vào văn bản cha, tuy nhiên cấu trúc của văn bản con được nhúng vào sẽ là một mảng, mỗi phần tử trong mảng sẽ có cấu trúc riêng biệt.

3.3.1.4.Cấu trúc lược đồ nhúng mảng tham chiếu

Trong một số trường hợp, ta phải dùng đến kiểu dữ liệu nhúng mảng các tham chiếu, lược đồ có cấu trúc như sau:



Lược đồ 3-4: Cấu trúc lược đồ nhúng mảng tham chiếu

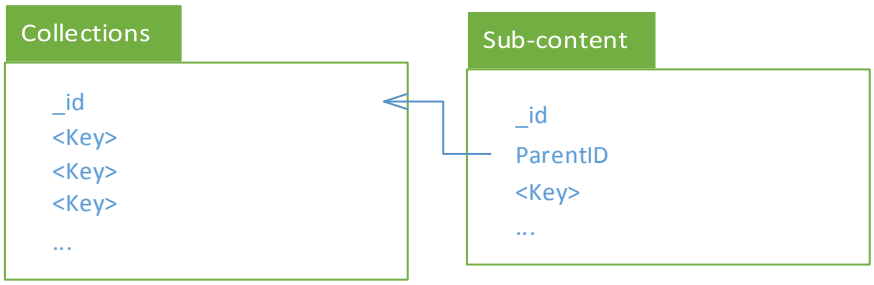
Cặp dấu ngoặc vuông "[]" được thể hiện cho kiểu dữ liệu mảng của văn bản con.

Văn bản nhúng tham chiếu sẽ là một mảng các giá trị id được dùng để tham chiếu đến dữ liệu trong bộ sưu tập khác.

Lưu ý: Ở tất cả các trường hợp sử dụng các phương pháp nhúng, ta đều cần phải lược bỏ đi thuộc tính tham chiếu ở thực thể con.

3.3.1.5.Cấu trúc lược đồ tham chiếu thực thể cha.

Cách này là cách tham chiếu truyền thống trong lược đồ quan hệ RDBMs, cách này cũng có thể được áp dụng trong MongoDB.



Lược đồ 3-5: Cấu trúc lược đồ tham chiếu thực thể cha

Cách này sẽ khiến dữ liệu trở nên rời rạc nhau, một tiêu chí mà MongoDB không hướng đến.

3.3.2.Cách sử dụng các biểu diễn mối kết hợp 1: n [2]

Dù có 2 loại mô hình quan hệ chính trong MongoDB đó là nhúng văn bản và tham chiếu văn bản, tuy nhiên khi phối hợp 2 phương pháp này lại, ta sẽ có được khá nhiều phương pháp khác nhau, mỗi phương pháp lại có một cách sử dụng thích hợp.

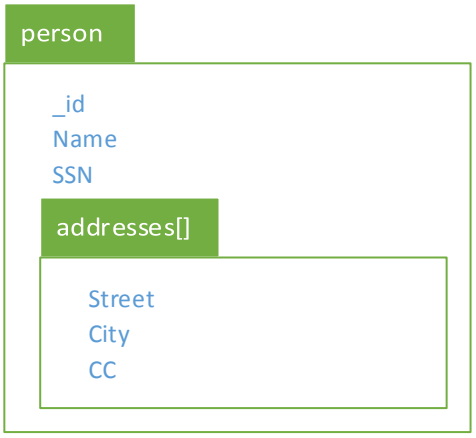
Như đã để cập trước đó, quan hệ số lượng 1: 1 sẽ được sử dụng phương pháp nhúng văn bản đơn. Tuy nhiên khi gặp quan hệ số lượng là 1: n, việc quyết định sẽ trở nên phức tạp hơn, vì nó sẽ còn phụ thuộc vào số lượng "n" quan hệ số lượng này.

3.3.2.1.n nhỏ

Ví dụ một quan hệ 1: n (n nhỏ) là về địa chỉ của một người. Đây sẽ là một lựa chọn tốt cho việc thể hiện dữ liệu, ta có thể biểu diễn các địa chỉ của một người theo dạng mảng, sau đó nhúng vào văn bản "person" với dạng một văn bản con "addresses".

```
> db.person.findOne()
{
  _id : ObjectID('AAAA'),
  name: 'Kate Monster',
  ssn: '123-456-7890',
  addresses : [
     { street: '123 Sesame St', city: 'Anytown', cc: 'USA' },
     { street: '123 Avenue Q', city: 'New York', cc: 'USA' }
  ]
}
```

Dữ liệu này được biểu diễn trên lược đồ như sau:



Lược đồ 3-6: Ví dụ mô hình phương pháp nhúng mảng văn bản

Ưu điểm chính của phương pháp này là ta không cần phải thực hiện nhiều câu truy vấn riêng lẻ để lấy các giá trị văn bản nhúng.

Nhược điểm chính của phương pháp này là ta không thể truy cập vào từng văn bản một cách riêng lẻ, mà ta chỉ có một con đường duy nhất là truy cập thông qua văn bản cha.

Document được nhúng sẽ là một mảng bao gồm cấu trúc của một đối tượng JSON, số lượng phần tử của mảng trong trường hợp này thông thường là khá nhỏ.

3.3.2.2.n trung bình

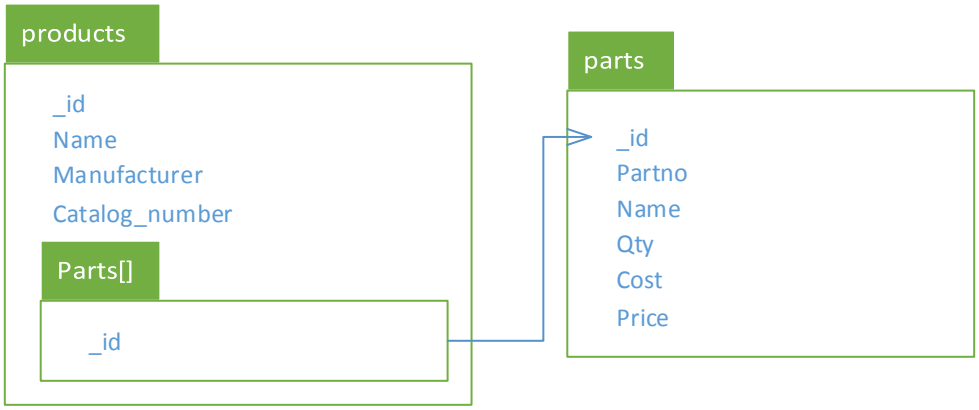
Ví dụ về quan hệ 1: n (n trung bình) có thể là các thành phần linh kiện trong một sản phẩm. Một sản phẩm có thể đi kèm với hàng trăm linh kiện khác nhau (không vượt quá hàng ngàn). Đây sẽ là trường hợp tốt để sử dụng cách biểu diễn này – Chúng ta cũng sẽ nhúng một mảng giá trị tương tự như cách n nhỏ, tuy nhiên một phần tử của mảng sẽ giá trị của trường _id của một văn bản tham chiếu tới, ví dụ trong trường hợp này, ta có một văn bản của một linh kiện của sản phẩm như sau:

```
> db.parts.findOne()
{
    _id : ObjectID('AAAA'),
    partno : '123-aff-456',
    name : '#4 grommet',
    qty: 94,
    cost: 0.94,
    price: 3.99
}
```

Trong văn bản của sản phẩm, ta sẽ liệt kê ra các linh kiện của nó thông qua _id của linh kiện, ví dụ như sau:

```
> db.products.findOne()
{    _id : ObjectID('…'),
    name : 'left-handed smoke shifter',
    manufacturer : 'Acme Corp',
    catalog_number: 1234,
    parts : [     // array of references to Part documents
        ObjectID('AAAA'),      // reference to the #4 grommet
above
        ObjectID('F17C'),    // reference to a different Part
        ObjectID('D2AA'),
        // etc
    ] }
```

Dữ liệu được biểu diễn trên lược đồ như sau:



Lược đồ 3-7: Ví dụ mô hình phương pháp nhúng mảng tham chiếu

Ưu điểm của cách này đó là, mỗi phần của dữ liệu đều độc lập với nhau, nó có thể được dễ dàng tìm kiếm truy vấn, cập nhật, một cách độc lập mà không cần phải thông qua document cha.

Nhược điểm của cách này đó là, để lấy được chi tiết những thành phần, MongoDB phải thực hiện 2 truy vấn, việc này ít nhiều cũng sẽ ảnh hưởng đến hiệu suất của MongoDB.

Điểm mở rộng của mô hình này đó là, nếu như quan hệ 1: n được chuyển thành quan hệ n: n, mô hình này vẫn có thể đáp ứng được yêu cầu đó mà không cần phải thực hiện bất kỳ sự thay đổi nào.

3.3.2.3.n lớn

Đây là phương pháp mà RDBMs thường dùng để lưu trữ dữ liệu với quan hệ 1-n, tuy nhiên phương pháp này ít khi được sử dụng trong MongoDB, ví nó chỉ được dùng khi 2 phương pháp trên không đáp ứng được. Ví dụ, trong trường hợp host lưu lại các thông báo "log" từ các máy khác nhau. Dữ liệu này thông thường sẽ rất lớn, và khi thực hiện nhúng vào 1 văn bản sẽ vượt quá dung lượng tối đa là 16MB, kể cả khi nhúng mảng các tham chiếu. Để giải quyết vấn đề này phương pháp tham chiếu đến thực thể cha sẽ sự lựa chọn cuối cùng.

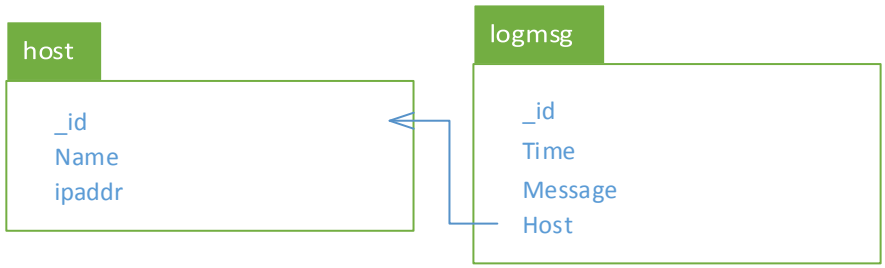
Nếu ta có một văn bản để lưu thông tin của các host, cùng với giá trị trường khóa _id.

Sau đó, với mỗi trường giá trị thông báo “log”, ta sẽ lưu một trường có giá trị _id của host để có thể thực hiện tham chiếu tới.

```
> db.hosts.findOne()
{
    _id : ObjectID('AAAB'),
    name : 'goofy.example.com',
    ipaddr : '127.66.66.66'
}

>db.logmsg.findOne()
{
    _id : ObjectID('…'),
    time : ISODate("2014-03-28T09: 42: 41.382Z"),
    message : 'cpu is on fire!',
    host: ObjectID('AAAB')/ Reference to the Host document
}
```

Dữ liệu được biểu diễn trên lược đồ như sau:

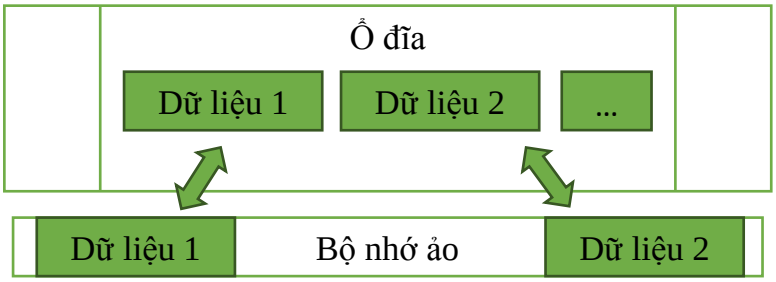


Lược đồ 3-8: Ví dụ mô hình phương pháp tham chiếu thực thể cha

3.4.Cơ chế xử lý dữ liệu trong MongoDB và bài toán tối ưu lược đồ [3]

MongoDB sử dụng cơ chế ánh xạ dữ liệu trên bộ nhớ ảo để lưu trữ, bộ nhớ này sẽ được hệ điều hành cấp phát khi khởi động MongoDB, dung lượng của bộ nhớ cũng sẽ tùy thuộc vào cấu hình của máy server.

Mỗi dữ liệu được ánh xạ sẽ là một phần trong bộ nhớ ảo, mỗi khi có yêu cầu lưu trữ dữ liệu nó sẽ ánh xạ từng byte đến bộ nhớ lưu trữ (ổ đĩa).



Hình 3-3: Mô phỏng cơ chế lưu dữ liệu trên MongoDB

Nhờ cơ chế ánh xạ dữ liệu, MongoDB có thể "phó thác" việc quản lý dữ liệu trong bộ nhớ ảo cho hệ điều hành thay vì tự MongoDB quản lý. Dữ liệu được ánh xạ sẽ được lưu trong bộ nhớ ảo cho đến khi bộ nhớ này đầy.

Khi vùng nhớ ảo đã được nạp đầy dữ liệu và ứng dụng lại yêu cầu một dữ liệu khác không có trong bộ nhớ RAM (Random Access Memory), dữ liệu sẽ được lấy từ trong ổ đĩa, đồng thời, hệ điều hành sẽ tạo ra một vùng nhớ trống để thực hiện lưu dữ liệu của yêu cầu mới hơn vào. Các dữ liệu trong bộ nhớ ảo sẽ được thay thế thông qua thuật LRU (Least Recently Used)để có thể loại bỏ đi dữ liệu ít sử dụng nhất và thêm dữ liệu mới vào.

Dữ liệu ứng dụng khai thác từ ứng yêu cầu xác định. Nếu như cơ chế xử lý dữ liệu có thể thực thi hoàn toàn trên bộ nhớ RAM, thì hệ điều hành sẽ không phải ánh xạ đến ổ đĩa mỗi khi khai thác, nhờ vậy tốc độ xử lý sẽ được ở mức cao nhất. Tuy nhiên, thực thế không như vậy, bộ nhớ ảo trên RAM sẽ không đủ để lưu tất cả các dữ liệu khai thác, nó chỉ lưu được một phần các dữ liệu được truy cập gần nhất đến khi nào đầy bộ nhớ, những dữ liệu không còn được lưu trữ trên bộ nhớ ảo, buộc phải ánh xạ đến dữ liệu trong ổ đĩa.

Vì vậy để tối ưu cho MongoDB, dữ liệu của MongoDB phải đáp ứng yêu cầu dữ liệu lưu trên RAM đều được sử dụng tối đa, tránh trường hợp lưu những dữ liệu không cần thiết làm tốn bộ nhớ của bộ nhớ ảo.

Cơ chế thay thế dữ liệu trong bộ nhớ ảo:

LRU (Least Recently Used), phương pháp này sẽ lưu lại dữ liệu được sử dụng gần nhất, đồng thời, khi có trang nhớ đầy, nó sẽ loại bỏ các dữ liệu được sử dụng xa nhất để thay cho dữ liệu gần nhất lưu vào.

Giả sử ví dụ ta có thứ tự xử lý dữ liệu các văn bản sau từ ứng dụng: 0 2 8 0 2 3 0 4 0 2

Với số trang nhớ là 3.

0	2	8	0	2	3	0	4	0	2
0^-	0	0	0^+	0	0	0^+	0	0^+	0
	2^-	2	2	2^+	2	2	4^-	4	4
		8^-	8	8	3^-	3	3	3	2^-

Bảng 3-2: Mô tả phương pháp thay thế bộ nhớ LRU

– : Cơ chế nạp dữ liệu thông qua ổ đĩa

+ : Cơ chế nạp dữ liệu từ bộ nhớ có sẵn trong RAM.

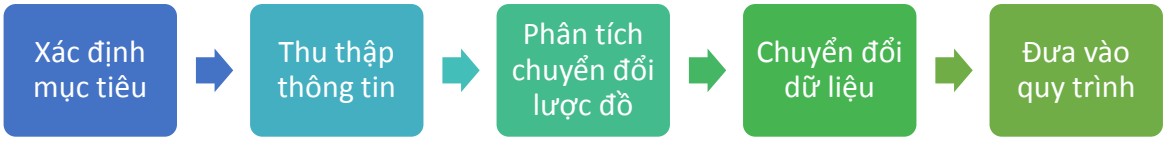
Bằng phương pháp này, trong ví dụ trên, ngoài 3 lần nạp đầu tiên từ ổ đĩa, trong quá trình xử lý, sẽ có 3/7 lần dữ liệu phải nạp từ ổ đĩa và 4/7 lần dữ liệu được nạp trong RAM.

Cơ chế này sẽ giúp cho các dữ liệu được sử dụng gần nhất được lưu trong bộ nhớ RAM, giảm tối đa chi phí ánh xạ đến dữ liệu trong ổ đĩa.

Vì vậy, để dữ liệu được lưu trên bộ nhớ RAM một cách hiệu quả, ta cần tiến hành phân mảnh, và chọn lọc các phép kết hợp hợp lý, để dữ liệu được lưu trên RAM luôn được khai thác triệt để.

3.5.Các bước chuyển đổi

Quy trình chuyển đổi lược đồ từ RDBMs sang MongoDB gồm 5 thao tác chính sau:



Hình 3-4: Mô hình các bước chuyển đổi[1]

Để thực hiện chuyển đổi từ dữ liệu RDBMs sang dữ liệu MongoDB, ta có các bước chính như sau:

3.5.1.Xác định mục tiêu

Ở bước đầu tiên, ta đã có lược đồ quan hệ RDBMs, từ lược đồ này, ta xác định mục tiêu, làm tôn chỉ cho việc chuyển đổi, tránh việc chuyển đổi không đáp ứng mục tiêu ban đầu đề ra, xác định tổng quan về mục đích chính ứng dụng.

3.5.2.Thu thập thông tin

Ở bước này, dựa vào lược đồ RDBMs có sẵn, ta thực hiện các công việc sau:

- Xác định các thực thể dữ liệu.
- Xác định các Index của dữ liệu thực thể (nếu có).
- Phân loại các nhóm thực thể chính dựa theo tính chất của thực thể và mục đích ứng dụng.
- Xác định cụ thể các quan hệ số lượng của các thực thể (nếu lược đồ chưa có).
- Xác định yêu cầu chức năng của ứng dụng (dựa vào đặc tả chức năng), cụ thể như sau:
    - Ta tiến hành liệt kê các chức năng của ứng dụng.
    - Xác định độ ưu tiên "p"của từng chức năng cho ứng dụng.
    - Liệt kê các dữ liệu cần thiết cho mỗi chức năng (dữ liệu đầu vào, đầu ra).

Với mỗi chức năng ta thể hiện như sau:

Chức năng[i]: tên chức năng.

Độ ưu tiên: $p_{Chức\ năng[i]}$ (%)

Thực thể	Trường	Dữ liệu đầu vào	Dữ liệu đầu ra
Bảng[ ]	Thuộc tính bảng[ ]	<x, 1, 2, …>	<x, 1, 2, …>
	…	…	…

Bảng 3-3: Xác định các dữ liệu cần thiết cho chức năng

"Độ ưu tiên chức năng p": hệ số này có thể được xác định trên nhiều cách khác nhau, có thể do khách hàng xác định dựa trên tiêu chí xác định các tính năng chính và phụ của ứng dụng. Hoặc cũng có thể được xác định bằng tỉ lệ sử dụng các chức năng trên ứng dụng.

3.5.3.Phân tích chuyển đổi lược đồ

Từ những điều đã xác định được ở bước trước đó, ta tiến hành xây dựng mô hình lược đồ NoSQL.

3.5.3.1. Xác định nhúng văn bản thực thể

Dựa vào các thông số tần suất giữa các thực thể dữ liệu với nhau, ta xác định được giải pháp biểu diễn dữ liệu phù hợp, cụ thể như sau:

Bước 1:

Với mỗi nhóm thực thể chính, ta xác định ra các thực thể con có các quan hệ số lượng 1: 1 hoặc 1: n.

Đây là các thực thể, ta sẽ thực hiện nhúng dữ liệu:

- 1:1: nhúng văn bản đơn.
- 1:n (n nhỏ): thực hiện nhúng mảng văn bản.
- 1:n (n lớn): thực hiện nhúng mảng tham chiếu.

Bước 2:

Với mỗi nhóm thực thể chính, ta biểu diễn theo bảng sau:

Thực thể	Chức năng [i]	…	Tần suất sử dụng ưu tiên (p)
Bảng chính (c)	<0,1>	…	$p(c)$
Bảng phụ [k]	<0,1>	…	$p[k]$
	$q[i][k]$		$p[k](c)$
…	…	…	…

Bảng 3-4: Phân tích thực hiện nhúng thực thể

Với mỗi giá trị của cột chức năng [i], ta đánh 0 nếu như không có dữ liệu của bảng trong dữ liệu đầu ra/vào của chức năng, ngược lại nếu nó có ta đánh 1.

Bước 3:

Ứng với mỗi bảng phụ [k], ta xác định tần suất giữa thực thể chính – thực thể phụ [k].

Gọi $q[i,k]$ số thể hiện dữ liệu được sử dụng của cả thực thể chính (c) – thực thể phụ [k] trên chức năng [i]

- 0 nếu có 1 trong 2 không có dữ liệu trong chức năng [i]
- 1 nếu có 2 dữ liệu trong 2 bảng đều là dữ liệu trong chức năng [i]

$$q[i][k] = (\text{Chức năng[i, c]} \cap \text{Chức năng[i, k]})$$

Bước 4:

Gọi $p[k](c)$ là tần suất sử dụng ưu tiên của dữ liệu cùng có trong thực thể chính (c) – thực thể phụ [k] trên chức năng [i]:

$$p[k](c) = \sum_{i=1}^{n} q[i][k] \times p_{Chức\ năng[i]}$$

Ta xác định tỉ lệ sử dụng T(%) của tần suất sử dụng chung $p[k](c)$ đối với từng thực thể chính (c) và thực thể phụ [k]:

$$T(c) = \frac{p[k](c)}{p(c)}(\%)$$

$$T[k] = \frac{p[k](c)}{p[k]}(\%)$$

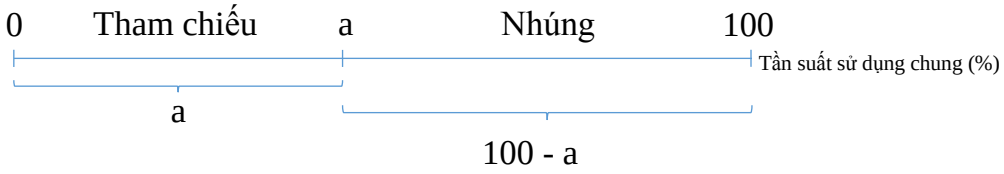
Bước 5:

Giả sử gọi:

x là chi phí xử lý dữ liệu trung bình bằng phương pháp nhúng.

y là chi phí xử lý dữ liệu trung bình bằng phương pháp tham chiếu.

x, y có thể được xác định bằng phương pháp ước lượng hay đo đạt cụ thể.



Hình 3-5: Xác định tần suất thực hiện nhúng

Giả sử gọi a là tỉ lệ tần suất để xác định thực hiện nhúng, theo (mục 0: 3.2.Biểu diễn quan hệ trong MongoDB) ta có:

- Tần suất sử dụng chung thấp (từ 0% → a) thực hiện tham chiếu.
- Tần suất sử dụng chung cao (từ a → 100%) thực hiện nhúng.

Để cân bằng chi phí xử lý dữ liệu giữa 2 phương pháp ta có công thức sau:

$$x \times (100 - a) = y \times a$$

→

$$a = \frac{x}{x + y} \times 100\ (\%)$$

(Nói cách khác, a cũng là tỉ lệ chi phí xử lý của phương pháp nhúng.)

Tiêu chuẩn để xác định nhúng cho văn một thực thể phụ [k] với thực thể chính (c) ta thực hiện như sau:

- ⇨ Quan hệ giữa thực thể chính (c) và thực thể [k] là 1: 1 hoặc 1: n.
- ⇨ $MIN(T(c), T[k]) > a$
- ⇨ Số lượng trường của một document nhỏ.
- ⇨ Số lượng document của thực thể [k] khi nhúng vào thực thể (c) nhỏ.
  (Đảm bảo kích thước document không vướt quá 16MB)

3.5.3.2.Xác định phân mảnh dữ liệu

Để phân mảnh được hiệu quả, thực thể đó cần phải đảm bảo các điều kiện sau đây:

- Số lượng trường của thực thể lớn.
- Tần suất được sử dụng của các trường là chênh lệch nhau.
- Dữ liệu khi được phân mảnh vẫn đảm bảo được tính trực quan của dữ liệu.

Dựa vào các thông số tần suất sử dụng các trường thuộc tính với nhau của các trường dữ liệu, ta tiến hành phân mảnh các thực thể, cụ thể như sau

Bước 1:

Với các thực thể, ta biểu diễn như sau:

Thực thể	Trường / Thuộc tính	Chức năng [i]	…	Tần suất sử dụng ưu tiên (p)
Bảng[k]	Bảng[k].thuộc tính[ ]	<0,1>	…	$p[j]$
	…	…	…	
…	…	…	…	
Trung bình				$\bar{p}$

Bảng 3-5: Phân tích phân mảnh thực thể

Giả sử, ta có:

n: tổng số chức năng.

u: tổng số thuộc tính trong mỗi bảng[k].

m: tổng số bảng.

p: Hệ số ưu tiên chức năng (%).

Với mỗi thuộc tính của từng bảng[k] ta đánh giá trị của từng cột chức năng[i] là 0 nếu thuộc tính đó không tồn tại thuộc tính đó trong bảng dữ liệu đầu ra / vào của chức năng [i], ngược lại nếu có tồn tại, ta đánh 1.

Bước 2:

Sau đó, "Tần suất sử dụng ưu tiên p" cho mỗi thuộc tính trong bảng [k] được xác định bằng cách:

$$\text{Giá trị } p[j] = \sum_{i=1}^{n} \text{Chức năng[i, j]}.\, p_{Chức\ năng[i]}$$

Bước 3:

Tính giá trị $\bar{p}$:

$$\bar{p} = \frac{\sum_{j=1}^{m} p[j]}{Count(p[j])} \text{ Với } (p[j] \neq 0)$$

Bước 4:

So sánh,với mỗi thuộc tính của bảng[k], ta so sánh giá trị $\bar{p}$ và $p[j]$ tương ứng.

Nếu $p[j] \geq \bar{p}$: Ta chọn trường tương ứng để phân mảnh.

Sau khi duyệt hết các phần tử thuộc tính của bảng[k]:

- Nếu số phần tử được tách lớn hơn 50% của u. Ta không tiến hành phân mảnh bảng[k].
- Ngược lại, ta tiến hành phân mảnh những phần tử được chọn với 2 nhóm thuộc tính được chọn và không được chọn trong bảng[k]. Trong đó, thực thể chính là nhóm các thuộc tính được chọn, thực thể phụ là nhóm những thuộc tính không được chọn.
- Thực thể phụ sẽ thực hiện tham chiếu đến thực thể chính thông qua _id của bảng với quan hệ số lượng 1:1.

Tuy nhiên, việc xác định phân mảnh cho thực thể còn tùy thuộc vào số lượng thuộc tính u lớn, và tính chất của dữ liệu, tránh trường hợp dữ liệu được tách ra sẽ trở nên khó hiểu và không có ý nghĩa.

3.5.3.3.Xác định lược đồ sau cùng

Kết hợp 2 kết quả phân tích được:

- Dựa vào các tính chất tự nhiên của dữ liệu, tinh chỉnh cho phù hợp (nếu có).
- Tiếp theo, ta tiến hành biểu diễn trên lược đồ.

3.5.4.Chuyển đổi dữ liệu

Dựa vào lược đồ ta đã xây dựng được ở bước trước đó:

- Tiến hành mô phỏng một số dữ liệu mẫu và thêm vào MongoDB.
- Kiểm tra hoạt động của dữ liệu mẫu.
- Tiến hành xây dựng công cụ chuyển đổi dữ liệu ETL (Extract, transform, load)

3.5.5.Đưa vào quy trình

Dữ liệu liệu sau khi được chuyển đổi bằng công cụ ETL, sẽ được tiến hành tiếp các bước cuối cùng:

- Quản lý và kiểm định chất lượng.
- Thực hiện các thao các bản phục hồi dữ liệu để tránh sự cố xảy ra ngoài ý muốn.
- Tiến hành phân tán dữ liệu (nếu cần) (High Availability - Horizontal Scaling).
- Xây dựng các phương pháp bảo mật phù hợp.

CHƯƠNG IV. CÀI ĐẶT

4.1.Tiến hành chuyển đổi lược đồ ứng dụng stackoverflow

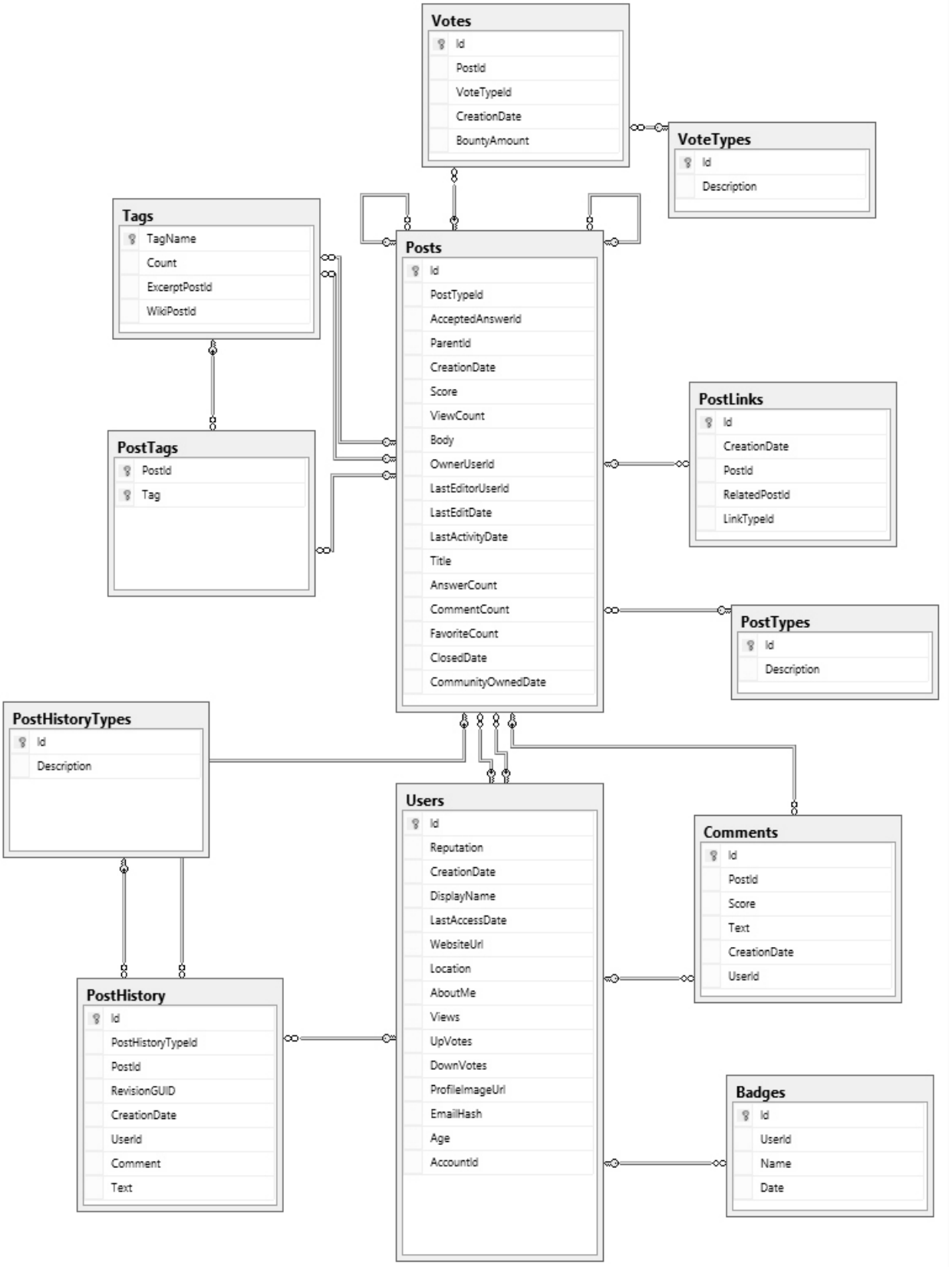
Để hiểu rõ ràng hơn về cách chuyển đổi lược đồ từ RDBMs sang lược đồ MongoDB, đề tài khóa luận này sẽ thực hiện mẫu cách chuyển đổi của một lược đồ RDBMs có sẵn sang lược đồ của MongoDB.

Trên nguồn dữ liệu Internet hiện nay, ta có thể dễ dàng tìm được các CSDL chuẩn như NorthWind, Pubs, hay AdventureWorks. Tuy nhiên, để phù hợp với yêu cầu của một ứng dụng thực tế, nên tôi đã chọn CSDL của ứng dụng StackoverFlow.com, vì đây là một ứng dụng thực tế, và khá gần gũi đối với những người lập trình viên.

4.1.1.Xác định mục tiêu

StackoverFlow.com là một cộng đồng hỏi đáp dành cho cộng đồng CNTT, không phân biệt ngôn ngữ lập trình, kỹ thuật, mạng… và được duy trì bởi cộng đồng thông qua các công cụ hỏi đáp, vote, điểm… cộng đồng chủ yếu sử dụng các chức năng đăng các vấn đề mà mình gặp phải, và sẽ được cộng đồng hỗ trợ thông qua các phản hồi. Vì vậy, việc chuyển đổi sẽ phục vụ chức năng chính là đăng tải các bài viết, quản lý các thông số thống kê và các hỗ trợ khác cho bài viết.

Ta có mô hình lược đồ RDBMs của StackoverFlow như sau:



Hình 4-1: Mô hình lược đồ use-case RDBMs ứng dụng stackoverflow.com

Nhìn vào lược đồ, dựa vào ý nghĩa và tác dụng của từng thực thể, ta tiến hành phân chia thành phần chính/phụ, đồng thời xác định các mối phụ thuộc nhau của các thực thể của lược đồ gồm:

- Thực thể Users
  - Thực thể Badges (1: n)
- Thực thể Posts
  - Thực thể Comments (1: n)
  - Thực thể PostLinks (1: 1)
  - Thực thể PostTags(1: n)
    - Thực thể Tag (n: 1)
  - Thực thể PostTypes (n: 1)
  - Thực thể PostHistory (1: n)
    - Thực thể PostHistoryTypes (n: 1)
  - Thực thể Vote (1: n)
    - Thực thể VoteType (n: 1)

Ở đây, ta có 2 thực thể chính đó là Users và Posts.

4.1.2.Thu thập thông tin

Với mỗi nhóm thực thể chính, ta liệt kê các trường trong RDBMs, cụ thể như sau:

Thực thể	Trường
Users	Users.Id
	Users.Reputation
	Users.CreationDate
	Users.DisplayName
	Users.LastAccessDate
	Users.WebsiteUrl
	Users.Location
	Users.AboutMe

		Users.Views
		Users.UpVotes
		Users.DownVotes
		Users.ProfileImageUrl
		Users.EmailHash
		Users.Age
		Users.AccountId
1: n	Badges	Badges.Id
		Badges.UserId
		Badges.Name
		Badges.Date

Bảng 4-1: Mô tả thành phần thực thể User

Thực thể	Trường
Posts	Posts.Id
	Posts.PostTypeId
	Posts.AcceptedAnswerId
	Posts.ParentId
	Posts.CreationDate
	Posts.Score
	Posts.ViewCount
	Posts.Body
	Posts.OwnerUserId
	Posts.LastEditorUserId
	Posts.LastEditDate
	Posts.LastAct4ityDate
	Posts.Title
	Posts.AnswerCount
	Posts.CommentCount
	Posts.FavoriteCount

			Posts.ClosedDate
			Posts.CommunityOwnedDate
1: n	Comments		Comments.Id
			Comments.PostId
			Comments.Score
			Comments.Text
			Comments.CreationDate
			Comments.UserId
1: 1	PostLinks		PostLinks.Id
			PostLinks.CreationDate
			PostLinks.PostId
			PostLinks.RelatedPostId
			PostLinks.LinkTypeID
1: n	PostTags		PostTags.PostId
			PostTags.Tag
	n: 1	Tags	Tags.TagName
			Tags.Count
			Tags.ExcerptPostId
			Tags.WikiPostId
			PostType.Id
1: n	PostType		PostType.Description
			PostHistory.Id
1: n	PostHistory		PostHistory.PostHistoryTypesId
			PostHistory.PostId
			PostHistory.RevisionGUID
			PostHistory.CreationDate
			PostHistory.UserId
			PostHistory.Comment
			PostHistory.Text

			PostHistoryTypes.Id
	n: 1	PostHistoryTypes	PostHistoryTypes.Description
			Votes.Id
1: n	Votes		Votes.PostId
			Votes.VoteTypeId
			Votes.CreationDate
			Votes.BountyAmount
	n: 1	VoteTypes	VoteTypes.Id
			VoteTypes.Description

Bảng 4IV-2: Mô tả thực thể Post

Tiếp theo, dựa theo yêu cầu của ứng dụng, ta xác định tần suất các dữ liệu được khai thác chung với nhau. Do không thu thập được yêu cầu chi tiết đặc tả chức năng của ứng dụng website stackoverflow.com nên em đã quyết định khai thác thông tin này dựa trên các thông tin được sử dụng trên các tính năng đặc trưng chính trong các nhánh khác nhau của ứng dụng.

4.1.2.1.Chức năng liệt kê danh sách câu hỏi

Error! Reference source not found., ta thấy các dữ liệu xử lý chính của chức năng như sau:

Thực thể	Trường	Dữ liệu đầu vào	Dữ liệu đầu ra
Posts	Posts.Id	x	
	Posts.OwnerUserId	x	
	Posts.PostTypeId	x	
	Posts.FavoriteCount		1
	Posts.AnswerCount		1
	Posts.ViewCount		1
	Posts.Title		2
	Posts.Body		3
	Posts.CreationDate		5
	Posts.LastEditDate		5

PostTags	PostTags.PostId	x	
	PostTags.Tag		4
Tags	Tags.TagName	x	
	Tags.Count	x	
	Tags.ExcerptPostId	x	
PostLinks	PostLinks.PostId	x	2
	PostLinks.RelatedPostId	x	2
Users	Users.Id	x	
	Users.DisplayName		5
	Users.Reputation		5
	Users.ProfileImageUrl		5
Badges	Badges.Id	x	5
	Badges.UserId	x	

Bảng 4-3: Dữ liệu chức năng liệt kê sanh sách câu hỏi

4.1.2.2.Chức năng hiển thị chi tiết bài đăng

Error! Reference source not found., ta thấy các dữ liệu xử lý chính của chức năng như sau:

Thực thể	Trường	Dữ liệu đầu vào	Dữ liệu đầu ra
Posts	Posts.Id	x	
	Posts.OwnerUserId	x	
	Posts.LastEditorUserId	x	
	Posts.ParentId	x	
	Posts.PostTypeId	x	
	Posts.Title		1
	Posts.CreationDate		2,8
	Posts.ViewCount	x	2
	Posts.LastAct4ityDate		2
	Posts.Score		3
	Posts.FavoriteCount		4
	Posts.Body		5

	Posts.LastEditDate		7
	Posts.AnswerCount		17
	Posts.AcceptedAnswerId		18
PostTags	PostTags.PostId	x	
	PostTags.Tag		6
Tags	Tags.TagName	x	
	Tags.Count	x	
	Tags.ExcerptPostId	x	
PostType	PostType.Id	x	
	PostType.Description	x	
PostLinks	PostLinks.PostId	x	
	PostLinks.RelatedPostId	x	
Comments	Comments.Id	x	
	Comments.PostId	x	
	Comments.UserId	x	
	Comments.Score	x	
	Comments.Text		14
	Comments.CreationDate		16
Users	Users.Id	x	
	Users.DisplayName		9, 15
	Users.Reputation		10
	Users.ProfileImageUrl		8
Badges	Badges.Id		11, 12,13
	Badges.UserId	x	

Bảng 4-4: Dữ liệu chức năng hiển thị chi tiết bài đăng

4.1.2.3.Chức năng đăng câu hỏi

Error! Reference source not found., ta thấy các dữ liệu xử lý chính của chức năng như sau:

Thực thể	Trường	Dữ liệu đầu vào	Dữ liệu đầu ra
Users	Users.Id	x	

Posts	Posts.Id	x	
	Posts.PostTypeId	x	
	Posts.AcceptedAnswerId	x	
	Posts.ParentId	x	
	Posts.CreationDate	x	
	Posts.Score	x	
	Posts.ViewCount	x	
	Posts.Body	x	
	Posts.OwnerUserId	x	
	Posts.LastEditorUserId	x	
	Posts.Title	x	
PostTags	PostTags.PostId	x	
	PostTags.Tag	x	
PostLinks	PostLinks.Id	x	
	PostLinks.CreationDate	x	
	PostLinks.PostId	x	
	PostLinks.RelatedPostId	x	

Bảng 4-5: Dữ liệu chức năng đăng câu hỏi

4.1.2.4.Chức năng lịch sử bài đăng

Error! Reference source not found.Error! Reference source not found., ta thấy các dữ liệu xử lý chính của chức năng như sau:

Thực thể	Trường	Dữ liệu đầu vào	Dữ liệu đầu ra
PostHistoryTypes	PostHistoryTypes.Id	x	
	PostHistoryTypes.Description		1
PostHistory	PostHistory.Id	x	
	PostHistory.PostId	x	
	PostHistory.UserId	x	

	PostHistory.PostHistoryTypesId	x	
	PostHistory.Text		3,7
Posts	Posts.Id	x	
	Posts.PostTypeId	x	
	Posts.OwnerUserId	x	
	Posts.LastEditDate		2
	Posts.CreationDate		6
	Posts.Title		5
	Posts.Body		4
PostTags	PostTags.PostId	x	
	PostTags.Tag		8
Tags	Tags.TagName	x	
	Tags.Count	x	
	Tags.ExcerptPostId	x	
PostType	PostType.Id	x	
	PostType.Description	x	
PostLinks	PostLinks.PostId	x	
	PostLinks.RelatedPostId	x	
Users	Users.Id	x	
	Users.DisplayName		2,6
	Users.Reputation		2,6
	Users.ProfileImageUrl		2,6
Badges	Badges.Id		2,6
	Badges.UserId	x	

Bảng 4-6: Dữ liệu chức năng lịch sử bài đăng

4.1.2.5.Chức năng liệt kê Tags

Error! Reference source not found.Error! Reference source not found., ta thấy các dữ liệu xử lý chính của chức năng như sau:

Thực thể	Trường	Dữ liệu đầu vào	Dữ liệu đầu ra
Tags	Tags.TagName		1
	Tags.Count		2
	Tags.ExcerptPostId	x	
Posts	Posts.Id	x	
	Posts.Body		3
	Posts.CreationDate	x	5
PostTags	PostTags.PostId	x	
	PostTags.Tag	x	4

Bảng 4-7: Dữ liệu chức năng liệt kê Tags

4.1.2.6.Chức năng liệt kê bài viết theo Tags

Error! Reference source not found.Error! Reference source not found., ta thấy các dữ liệu xử lý chính của chức năng như sau:

Thực thể	Trường	Dữ liệu đầu vào	Dữ liệu đầu ra
Tags	Tags.WikiPostId		1
Error! Reference source not found.			2

Bảng 4-8: Dữ liệu chức năng liệt kê bài viết theo Tags

4.1.2.7.Chức năng liệt kê người dùng

Error! Reference source not found.. Error! Reference source not found., ta thấy các dữ liệu xử lý chính của chức năng như sau:

Thực thể	Trường	Dữ liệu đầu vào	Dữ liệu đầu ra
Users	Users.Id	x	
	Users.ProfileImageUrl		1
	Users.DisplayName		2
	Users.Location		3
	Users.Reputation		4
	Users.CreationDate	x	6
	Users.UpVotes	x	7

	Users.DownVotes	x	7
Posts	Posts.OwnerUserId	x	
	Posts.Score	x	
	Posts.CreationDate	x	
PostTags	PostTags.PostId	x	
	PostTags.Tag		5

Bảng 4-9: Dữ liệu chức năng liệt kê người dùng

4.1.2.8.Chức năng xem chi tiết thông tin người dùng

Error! Reference source not found.Error! Reference source not found., ta thấy các dữ liệu xử lý chính của chức năng như sau:

Thực thể	Trường	Dữ liệu đầu vào	Dữ liệu đầu ra
Users	Users.Id	x	
	Users.DisplayName		1
	Users.WebsiteUrl		2
	Users.Location		2
	Users.Age		2
	Users.CreationDate		2
	Users.LastAccessDate		2
	Users.Views		2
	Users.AboutMe		3
	Users.ProfileImageUrl		4
	Users.Reputation		5
Badges	Badges.Id	x	6
	Badges.UserId	x	6

Bảng 4-10: Dữ liệu chức năng xem chi tiết thông tin người dùng

Trong trường hợp chức năng này, ta chỉ xét các thông tin của người dùng là chức năng chính, và bỏ qua các chức năng tổng hợpliệt kê chi tiết quá trình hoạt động, thống kê, vì nó có liên quan đến hầu hết các trường của các thực thể trong database.

Cụ thể, trong trường hợp này, chúng ta có thể bỏ qua các thông tin được liệt kê trong các tab "Summary, answers, questions, tags, …"

4.1.2.9.Chức năng liệt kê các Badges

Error! Reference source not found.Error! Reference source not found., ta thấy các dữ liệu xử lý chính của chức năng như sau:

Thực thể	Trường	Dữ liệu đầu vào	Dữ liệu đầu ra
Badges	Badges.Id	x	
	Badges.UserId	x	2
	Badges.Date	x	
	Badges.Name		1,3
Users	Users.Id	x	
	Users.DisplayName		4

Bảng 4-11: Dữ liệu chức năng liệt kê các Badges

4.1.2.10.Chức năng xem chi tiết Badges

Error! Reference source not found..Error! Reference source not found., ta thấy các dữ liệu xử lý chính của chức năng như sau:

Thực thể	Trường	Dữ liệu đầu vào	Dữ liệu đầu ra
Badges	Badges.Id	x	
	Badges.UserId	x	
	Badges.Date	x	2
	Badges.Name		1
Users	Users.Id	x	
	Users.ProfileImageUrl		3
	Users.DisplayName		4
	Users.Reputation		5

Bảng 4-12: Dữ liệu chức năng xem chi tiết Badges

4.1.2.11.Xác định độ ưu tiên p

Sau khi tiến hành so khớp dữ liệu: Ta có bảng sau:

Bằng việc đánh chỉ số tỉ lệ % độ ưu tiên ở các chức năng:

- Chức năng hiển thị chi tiết bài đăng 35%

- Chức năng liệt kê sanh sách câu hỏi 25%
- Chức năng đăng câu hỏi..... 20%
- Chức năng lịch sử bài đăng..... 2%
- Chức năng liệt kê Tag 4%
- Chức năng liệt kê bài viết theo tag 4%
- Chức năng liệt kê người dùng..... 2%
- Chức năng xem chi tiết thông tin người dùng 4%
- Chức năng liệt kê các Badges 2%
- Chức năng xem chi tiết badges 2%

4.1.3.Phân tích chuyển đổi lược đồ

4.1.3.1.Xác định nhúng văn bản

Ta có quan hệ của các thực thể như sau:

- Users – Badges: ..... 1 – n
- Posts – Comments: ..... 1 – n
- Posts – PostLinks: ..... 1 – 1
- Posts – PostTags: ..... 1 – n
- Posts – PostHistory: ..... 1 – n
- Posts – Votes: ..... 1 – n
- Posts – PostType: ..... n – 1
- PostTags – Tags: ..... n – 1
- Votes – VoteType: ..... n – 1
- PostHistory – PostHistoryType: ..... n – 1

Ta sẽ sét tần suất sử dụng ưu tiên cho các quan hệ số lượng 1 – 1 và 1 – n để thực hiện nhúng.

Các quan hệ số lượng n – 1 ta sẽ thực hiện tham chiếu.

Thực thể	Tần suất sử dụng ưu tiên (p)	$T(c)$	$T[k]$
Users	96%		

1: n	Badges	74%		
	$Users \cap Badges$	74%	77%	100%

Bảng 4-13: Bảng phân tích tần suất sử dụng thực thể Users

Thực thể		Tần suất sử dụng ưu tiên (p)	$T(c)$	$T[k]$
Posts		92%		
1: n	Comment	35%		
	$Posts \cap Comment$	35%	38%	100%
1: 1	PostLinks	86%		
	$Posts \cap PostLinks$	86%	93%	100%
1: n	PostTags	92%		
	$Posts \cap PostTags$	92%	100%	100%
1: n	PostHistory	2%		
	$Posts \cap PostHistory$	2%	2%	100%
1: n	Votes	0%		
	$Posts \cap Votes$	0%	0%	0%

Bảng 4-14: Bảng phân tích tần suất sử dụng thực thể Posts

Gọi x, y lần lượt là chi phí thực hiện phương pháp nhúng và tham chiếu.

Bằng phương pháp ước lượng, giả sử ta ước lượng: x = 1; y = 1.2.

a là tỉ lệ tần suất để xác định thực hiện nhúng, ta có:

$$x \times (100 - a) = y \times a$$

➔

$$a = \frac{x}{x + y} \times 100\ (\%)$$

Vậy $a = 45.45(\%)$.

Xét tần suất giữa 2 thực thể Users và Badges:

- Quan hệ số lượng giữa Users và Badges là 1: n
- $MIN(T(c), T[k]) = 77\% > a$

- Dung lượng của một document nhỏ.
- Số lượng nhúng Badges trong một Users nhỏ.

➔Vậy có thể thực hiện nhúng thực thể Badges vào Users.

Xét tần suất giữa 2 thực thể Posts và Comments:

- Quan hệ số lượng giữa Posts và Comments là 1: n
- $MIN(T(c), T[k]) = 38\% < a$

➔Vậy ta không thực hiện nhúng Comments vào Post.

Xét tần suất giữa 2 thực thể Posts và PostLinks:

- Quan hệ số lượng giữa Posts và PostLinks là 1: 1
- $MIN(T(c), T[k]) = 93\% > a$
- Dung lượng của một document nhỏ.

➔Vậy có thể thực hiện nhúng thực thể PostLinks vào Posts.

Xét tần suất giữa 2 thực thể Posts và PostTags:

- Quan hệ số lượng giữa Posts và PostTags là 1: n
- $MIN(T(c), T[k]) = 100\% > a$
- Dung lượng của một document nhỏ.
- Số lượng nhúng PostTags trong một Posts nhỏ.

➔Vậy có thể thực hiện nhúng thực thể PostTags vào Posts.

Xét tần suất giữa 2 thực thể Posts và PostHistory:

- Quan hệ số lượng giữa Posts và PostHistory là 1: n
- $MIN(T(c), T[k]) = 2\% < a$

➔Vậy ta không thực hiện nhúng PostHistory vào Posts.

Xét tần suất giữa 2 thực thể Posts và Votes:

- Quan hệ số lượng giữa Posts và Votes là 1: n
- $MIN(T(c), T[k]) = 0\% < a$

➔Vậy ta không thực hiện nhúng PostHistory vào Posts.

Vậy có thể thực hiện nhúng thực thể PostLinks vào Posts.

Sau khi phân tích nhúng văn bản ta được kết quả như sau:

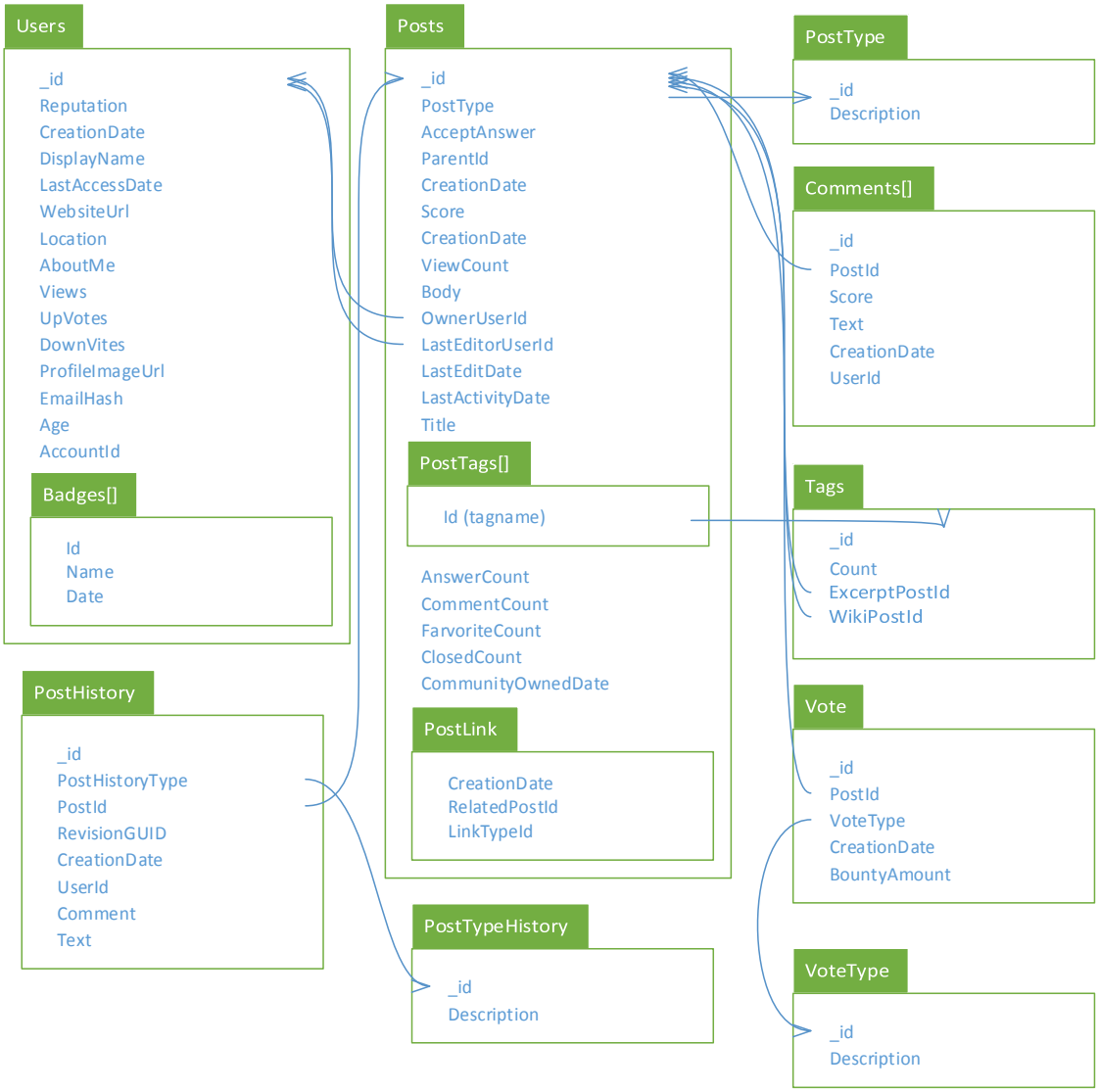
Các quan hệ thực hiện nhúng bao gồm:

- Users và Badges
- Posts và PostLinks
- Posts và PostTags

Các quan hệ thực hiện tham chiếu bao gồm

- Posts và Comments
- Posts và PostHistory
- Posts và Votes
- Các quan hệ n – 1: Posts – PostType, PostTags – Tags, Votes – VoteType, PostHistory – PostHistoryType

Dựa vào các kết luận trên, ta có thể thực hiện phác thảo lược đồ như sau:



Lược đồ 4-1: Kết quả sau khi phân tích nhúng văn bản

4.1.3.2.Xác định phân mảnh dữ liệu

Ta tiến hành thực hiện xác bước phân tích như ở phần 0, ta được biểu đồ sau:

Thực thể	Cột	
	Trường	Tần suất sử dụng ưu tiên (p)

Users		Users.Id	96%
		Users.Reputation	74%
		Users.CreationDate	6%
		Users.DisplayName	76%
		Users.LastAccessDate	4%
		Users.WebsiteUrl	4%
		Users.Location	6%
		Users.AboutMe	4%
		Users.Views	4%
		Users.UpVotes	2%
		Users.DownVotes	2%
		Users.ProfileImageUrl	74%
		Users.EmailHash	0%
		Users.Age	4%
		Users.AccountId	0%
1: n	Badges	Badges.Id	74%
		Badges.UserId	74%
		Badges.Name	4%
		Badges.Date	4%
Posts		Posts.Id	90%
		Posts.PostTypeId	86%
		Posts.AcceptedAnswerId	55%
		Posts.ParentId	55%
		Posts.CreationDate	92%
		Posts.Score	57%
		Posts.ViewCount	84%
		Posts.Body	90%
		Posts.OwnerUserId	88%
		Posts.LastEditorUserId	55%

			Posts.LastEditDate	66%
			Posts.LastAct4ityDate	35%
			Posts.Title	86%
			Posts.AnswerCount	64%
			Posts.CommentCount	0%
			Posts.FavoriteCount	64%
			Posts.ClosedDate	0%
			Posts.CommunityOwnedDate	0%
1: n	Comments		Comments.Id	35%
			Comments.PostId	35%
			Comments.Score	35%
			Comments.Text	35%
			Comments.CreationDate	35%
			Comments.UserId	35%
1: 1	PostLinks		PostLinks.Id	20%
			PostLinks.CreationDate	20%
			PostLinks.PostId	86%
			PostLinks.RelatedPostId	86%
			PostLinks.LinkTypeID	0%
1: n	PostTags		PostTags.PostId	92%
			PostTags.Tag	92%
	n: 1	Tags	Tags.TagName	70%
			Tags.Count	70%
			Tags.ExcerptPostId	70%
			Tags.WikiPostId	4%
1: n	PostType		PostType.Id	37%
			PostType.Description	37%
			PostHistory.Id	2%
1: n	PostHistory		PostHistory.PostHistoryTypesId	2%

			PostHistory.PostId	2%
			PostHistory.RevisionGUID	0%
			PostHistory.CreationDate	0%
			PostHistory.UserId	2%
			PostHistory.Comment	0%
			PostHistory.Text	2%
	n: 1	PostHistoryTypes	PostHistoryTypes.Id	2%
			PostHistoryTypes.Description	2%
1: n	Votes		Votes.Id	0%
			Votes.PostId	0%
			Votes.VoteTypeId	0%
			Votes.CreationDate	0%
			Votes.BountyAmount	0%
	n: 1	VoteTypes	VoteTypes.Id	0%
			VoteTypes.Description	0%

Bảng 4-15: Bảng tính tần suất sử dụng ưu tiên

Theo như lược đồ, ta tính được:

$$\bar{p} = 43.63\%$$

Nhìn vào bảng dữ liệu thống kê được, ta thấy số thuộc tính của 2 thực thể Posts và Users là tương đối nhiều, vì thế, ta chỉ xét phân mảnh cho 2 thực thể này.

Xét thực thể Users, ta thấy:

- Tổng số thuộc tính là 15.
- Số thuộc tính được chọn để phân mảnh là 4 ($p > \bar{p}$), chiếm tỉ lệ 26.67% (< 50%).
- Các thuộc tính được chọn và không được chọn khi tách ra vẫn thể hiện được tính trực quan của dữ liệu.

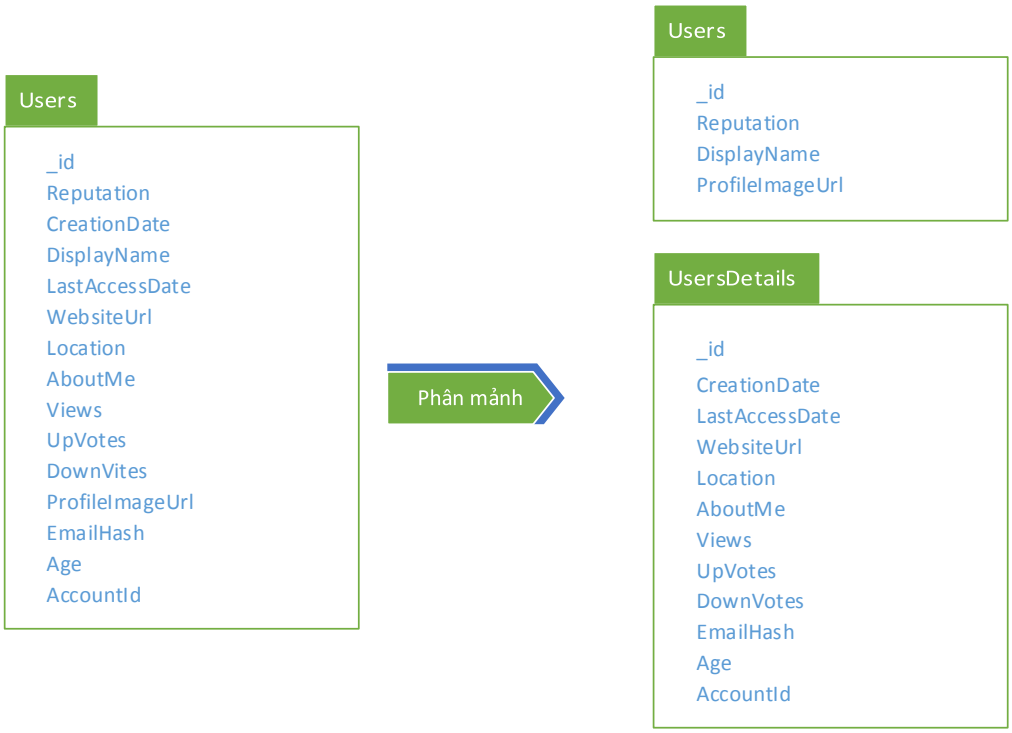
➔Vậy, thực thể Users có thể tiến hành phân mảnh.

Xét thực thể Posts, ta thấy:

- Tổng số thuộc tính là 18.
- Số thuộc tính được chọn để phân mảnh là 14 ($p > \bar{p}$), chiếm tỉ lệ 77.78% (> 50%).
- Các thuộc tính được chọn và không được chọn khi tách ra không thể hiện được tính trực quan của dữ liệu.

➔Vậy, thực thể Posts không được tiến hành phân mảnh.

Sau khi phân tích phân mảnh ta được kết quả như sau:



Lược đồ 4-2: Kết quả sau khi phân tích phân mảnh thực thể Users

4.1.3.3.Xác định lược đồ sau cùng

Để thấy được sự linh hoạt của MongoDB.

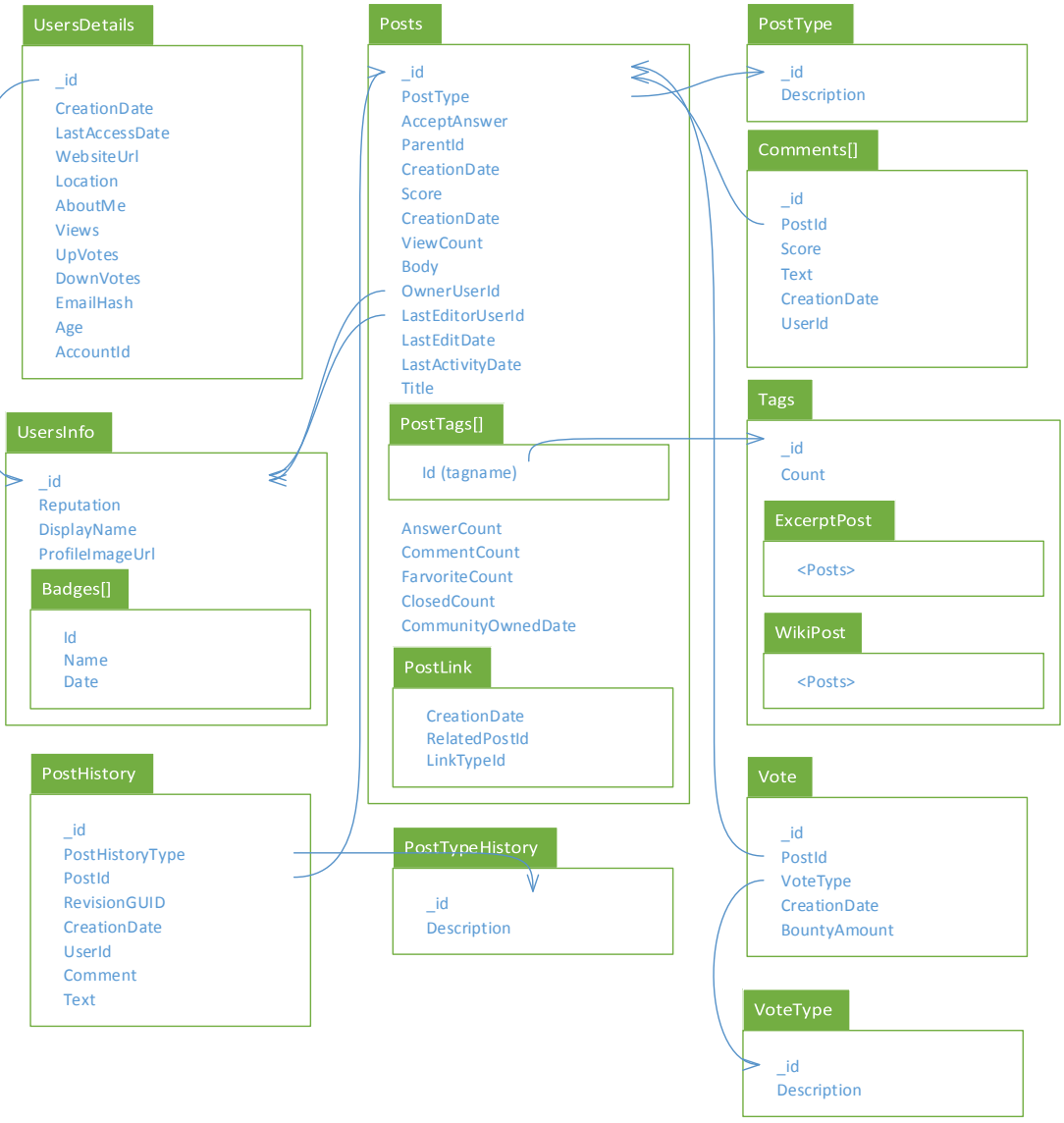
Ta xét trong thực thể Tags, có 2 trường là ExcerptPostId và WikiPostId, hai trường này được tham chiếu tới PostID trong thực thể Posts, nó có tác dụng như là một bài viết để trình bày khái niệm về Tag, giả sử các bài viết này không có các chức năng Vote, Favorite, Answer, Tag.

- Mỗi một thực thể Tag sẽ chỉ có duy nhất một bài viết ExcerptPost và WikiPost.
- Mỗi ExcerptPost và WikiPost chỉ được sử dụng trong một Tag duy nhất

Vậy, quan hệ giữa Tag – ExcerptPost và Tag – ExcerptPost đều là quan hệ 1–1.

Ở trường hợp nàyta có thể sử dụng phương pháp nhúng, thay thế cho phương pháp tham chiếu, ta có thể nhúng 1 document có cấu trúc của một thực thể Post.

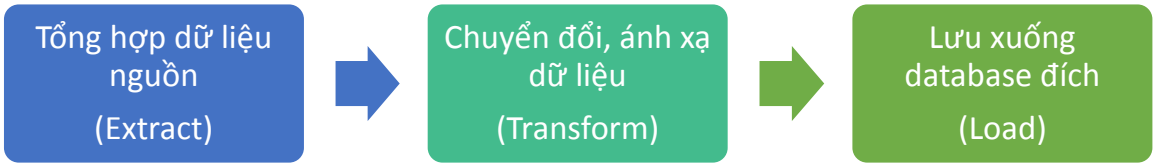
Khi này lược đồ cuối cùng sau khi chuyển đổi như sau:



Lược đồ 4-3: Kết quả lược đồ cuối cùng

4.1.4.Thực hiện chuyển đổi

Ta tiến hành thực hiện xây dựng công cụ chuyển đổi ETL (Extract, Transform, Load), công cụ này bao gồm 3 chức năng chính đó là:



Hình 4-2: Sơ đồ xử lý của công cụ ETL

- **Tổng hợp dữ liệu nguồn**

  Tại bước này sẽ dùng các lớp mô phỏng các đối tượng tương ứng với dữ liệu trên các cơ sở dữ liệu SQL để phục vụ việc chuyển đổi, ánh xạ ở bước sau.

  Để thuận tiện và nhanh chóng, ở đây dùng đối tượng ADO .NET Entity FW6.0 để mô phỏng và hỗ trợ thực hiện truy xuất dữ liệu SQL.

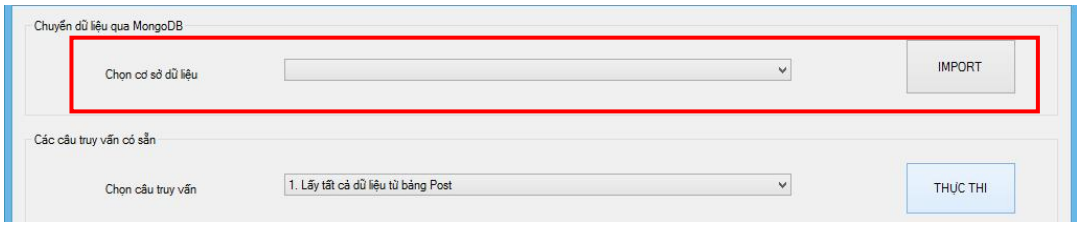
- **Chuyển đổi, ánh xạ dữ liệu**

  Tại bước này, công cụ sẽ chuyển đổi, ánh xạ bộ dữ liệu cũ sang bộ dữ liệu mới với cấu trúc của các BSON.

  Ta thực hiện viết các phương thức để hỗ trợ chuyển đổi bộ dữ liệu cũ sang bộ dữ liệu mới tương ứng với lược đồ CSDL MongoDB mà ta đã chuyển đổi được trước đó

- **Lưu xuống database đích**

  Khi có các bộ dữ liệu mới, công cụ thực hiện lưu dữ liệu xuống database MongoDB.



Hình 4-3: Giao diện chính công cụ chuyển đổi cơ sở dữ liệu.

Chi tiết màn hình:

Chọn cơ sở dữ liệu SQL Server chuyển sang cơ sở dữ liệu Mongo

Lưu ý:

- Trước khi thực hiện các bước trên, phải chắc chắn là Server của MongoDB đã chạy trên máy.

4.2.Đánh giá

4.2.1.So sánh tốc độ xử lý truy vấn

Môi trường thực hiện:

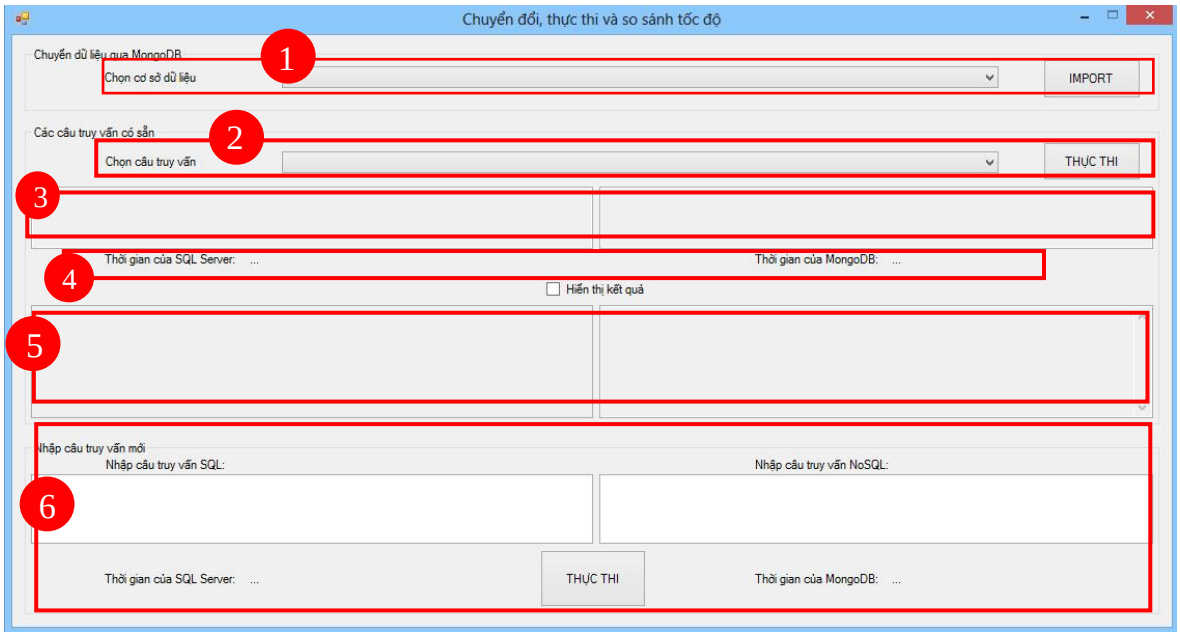
- Máy tính dùng Core i3

- Hệ điều hành Windows 8
- RAM 4GB
- SQL Server 2012 EXPRESS
- MongoDB 2.6.4

Để thuận tiện theo dõi tốc độ xử lý của SQL Server và MongoDB, em đã viết chương trình đơn giản nhằm thể hiện thời gian thực hiện các câu truy vấn giữa SQL Server và MongoDB. Tất cả các lệnh truy vấn đều chạy trên cùng một process để đảm bảo tính khách quan.

Chi tiết màn hình:

- Khung số 1: Chọn cơ sở dữ liệu SQL Server chuyển sang cơ sở dữ liệu MongoDB.
- Khung số 2: Chọn mẫu câu truy vấn sẵn có để so sánh.
- Khung số 3: Câu lệnh truy vấn SQL và câu lệnh truy vấn MongoDB.
- Khung số 4: Thời gian thực hiện truy vấn trên SQL và MongoDB.
- Khung số 5: Hiển thị dữ liệu truy vấn SQL và MongoDB.
- Khung số 6: Truy vấn tự nhập.



Các bước so sánh

1. Chọn mẫu câu truy vấn ở Combobox “Chọn câu truy vấn”.
2. Nhấn nút “Thực thi” để thực hiện truy vấn.
3. Xem kết quả truy vấn ở khung số 4 và khung số 5.

Lưu ý:

Trước khi thực hiện các bước trên, phải chắc chắn là Server của MongoDB đã chạy trên máy.

Các câu lệnh truy vấn mẫu

- ***Truy vấn 1:****Lấy tất cả dữ liệu bảng Posts*
  - *SQL Server*

```
SELECT *
FROM Posts
```

  - *MongoDB*

```
db.Posts.find()
```

- ***Truy vấn 2:*** *Chọn ra tất cả các bản ghi Post có AnswerCount<>10*
  - *SQL Server*

```
SELECT *
FROM Posts
WHERE AnswerCount!=6
```

  - *MongoDB*

```
db.Posts.find( {AnswerCount:{$ne:6}})
```

- ***Truy vấn 3:****Chọn ra tất cả các bản ghi Post mới nhất cóTitle ký tự đầu là Why*
  - *SQL Server*

```
SELECT *
FROM Posts
WHERE  Title LIKE '%Why%'
```

  - *MongoDB*

```
db.Posts.find({Title: /Why/})
```

- ***Truy vấn 4:*** *Chọn ra tất cả các bản ghi PostHistorycóPostHistoryTypeId>7*
  - *SQL Server*

```
SELECT *
FROM PostHistory
WHERE PostHistoryTypeId >7
```

  - *MongoDB*

```
db.PostHistory.find({PostHistoryTypeId :{$gt:7}})
```

- ***Truy vấn 5:*** *Chọn ra tất cả các bản ghi PostHistorycóText ký tự đầu là Why*
  - *SQL Server*

```
SELECT *
FROM PostHistory
WHERE  Text LIKE '%Why%'
```

- *MongoDB*

```
db.PostHistory.find({Text: /Why/})
```

Kết quả:

Để đảm bảo tính chính xác, kết quả được thực hiện với các điều kiện sau:

- SQL Server và MongoDB đều không có bất kỳ chỉ số Index nào.
- Để giảm thiểu sai số, mỗi truy vấn được thực hiện 5 đợt và tính giá trị trung bình.
- Trước mỗi câu truy vấn, bộ nhớ được giải phóng, đảm bảo các dữ liệu truy vấn đều được truy cập trực tiếp đến ổ đĩa.

		Lần 1	Lần 2	Lần 3	Lần 4	Lần 5	TB
Truy vấn 1	SQL Server	59.819	46.990	46.775	47.790	53.714	51.018
	MongoDB	0.040	0.039	0.051	0.041	0.055	0.045
Truy vấn 2	SQL Server	41.868	41.769	37.354	36.455	36.816	38.852
	MongoDB	0.034	0.012	0.091	0.080	0.091	0.062
Truy vấn 3	SQL Server	31.655	23.083	23.398	23.354	22.499	24.798
	MongoDB	0.076	0.093	0.081	0.106	0.077	0.087
Truy vấn 4	SQL Server	26.469	28.351	27.156	27.797	26.781	27.311
	MongoDB	1.493	1.570	1.534	1.560	1.525	1.536
Truy vấn 5	SQL Server	78.488	72.433	73.399	64.177	68.183	71.336
	MongoDB	0.082	0.021	0.101	0.075	0.076	0.071

4.2.2.Đánh giá

4.2.2.1.Những điểm tích cực khi sử dụng MongoDB

- Tốc độ: Một điểm mạnh nhất và quan trọng của MongoDB đó là tốc độ đọc ghi rất ấn tượng, do bỏ qua khâu kiểm tra các ràng buộc toàn vẹn nên MongoDB đọc và ghi nhanh hơn nhiều lần so với RDBMS (cụ thể xem ở phần kết quả so sánh tốc độ).
- Chi phí thấp hơn: Chi phí bỏ ra cho việc xử lý dữ liệu sẽ giảm đáng kể, các yêu cầu được đáp ứng nhanh chóng, làm tăng trải nghiệm của người sử dụng.
- Lược đồ đơn giản hơn: Khi sử dụng MongoDB, mô hình dữ liệu thường được kiểm soát bởi các lập trình viên mà không cần tới những quản trị viên cơ sở dữ liệu, lược đồ cũng được tối ưu, thân thiện với ứng dụng.

4.2.2.2.Những điểm hạn chế khi sử dụng MongoDB

- Do cơ chế của MongoDB không hỗ trợ xử lý song song các truy vấn cùng một lúc, tất cả các truy vấn được đưa vào hàng đợi, và server xử lý từng query một. Để khắc phục điều này, ta tạo nhiều thể hiện của MongoDB trên một máy và tạo ra nhiều cơ sở dữ liệu cho các khóa tranh chấp. Vì vậy việc quản lý các thể hiện và cơ sở dữ liệu cũng phức tạp.
- Ưu điểm cũng chính là nhược điểm của MongoDB khi không kiểm tra các ràng buộc toàn vẹn, chính điều này đôi khi làm cho cơ sở dữ liệu không nhất quán và xảy ra xung đột nếu lập trình viên không xử lý và bao quát hết các trường hợp.
- Do không có một mô hình chuẩn nào được định nghĩa trên cơ sở dữ liệu MongoDB nên khó kiểm soát
- Do còn mới mẻ và đang trong quá trình phát triển nên tính ổn định chưa bằng các hệ quản trị cơ sở dữ liệu quan hệ.

TÀI LIỆU THAM KHẢO

[1] RDBMS to MongoDB Migration Guide, New York • Palo Alto • Washington, D.C. • London • Dublin • Barcelona • Sydney • Tel Av4, 2014, p. 2.

[2] "6 Rules of Thumb for MongoDB Schema Design," 29 5 2014. [Online]. Available: http://blog.mongodb.org/post/87200945828/6-rules-of-thumb-for-mongodb-schema-design-part-1.

[3] "MongoDB Storage," [Online]. Available: http://learnmongodbthehardway.com/schema/chapter3/.

[4] "MongoDB: The Definit4e Guide," in *MongoDB: The Definit4e Guide*.

[5] Extending a Methodology for Migration of the Database Layer to the Cloud Considering Relational Database Schema Migration to NoSQL. ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/MSTR-3460/MSTR-3460.pdf. Access on July, 2014.