NoSQL

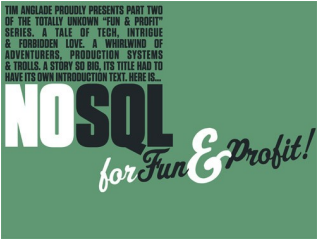
# Giới thiệu NoSQL

## Thuật ngữ NoSQL

NoSQL còn có nghĩa là Non-Relational (NoRel) - không ràng buộc. Tuy nhiên, thuật ngữ đó ít phổ dụng hơn và ngày nay người ta thường dịch NoSQL thành Not Only SQL - Không chỉ SQL. NoSQL ám chỉ đến những cơ sở dữ liệu không dùng mô hình dữ liệu quan hệ để quản lý dữ liệu trong lĩnh vực phần mềm.

## Lịch sử

* Thuật ngữ NoSQL được giới thiệu lần đầu vào năm 1998 sử dụng làm tên gọi chung cho các lightweight open source relational database (cơ sở dữ liệu quan hệ nguồn mở nhỏ) nhưng không sử dụng SQL cho truy vấn.
* Vào năm 2009, Eric Evans, nhân viên của Rackspace giới thiệu lại thuật ngữ NoSQL trong một hội thảo về cơ sở dữ liệu nguồn mở phân tán. Thuật ngữ NoSQL đánh dấu bước phát triển của thế hệ database mới: distributed (phân tán) + non-relational (không ràng buộc).



* Ghi chú: Một mệnh đề khá thú vị về non-relational data store: "select fun, profit from real\_world where relational=false;".

## Định nghĩa

* Thế hệ database kế tiếp là một thế hệ cơ sở dữ liệu non-relational (không ràng buộc), distributed (phân tán), open source, horizontal scalable (khả năng mở rộng theo chiều ngang) có thể lưu trữ, xử lý từ một lượng rất nhỏ cho tới hàng petabytes dữ liệu trong hệ thống có độ chịu tải, lỗi cao với những đòi hỏi về tài nguyên phần cứng thấp.
* Một số đặc điểm nhận dạng cho thế hệ database mới này bao gồm:
  + Schema-free
  + Hỗ trợ mở rộng dễ dàng
  + API đơn giản
  + Eventual consistency (nhất quán cuối) và/hoặc transactions hạn chế trên các thành phần dữ liệu đơn lẻ
  + Không giới hạn không gian dữ liệu
  + ...
* NoSQL storage đặc biệt phổ dụng trong thời kỳ Web 2.0 bùng nổ, nơi các mạng dịch vụ dữ liệu cộng đồng cho phép người dùng tạo hàng tỷ nội dung trên web. Do đó, dữ liệu lớn rất nhanh vượt qua giới hạn phần cứng và cần phải giải quyết bằng bài toán phân tán. Nửa đầu năm 2009, người ta đã manh nha thuật ngữ NoSQL đánh dấu sự trưởng thành của thế hệ database mới trong khi những sản phẩm phần mềm có thể đã được phát triển từ trước đó rất lâu.

## Một số thuật ngữ liên quan

* **Non-relational**: relational - ràng buộc - thuật ngữ sử dụng đến các mối quan hệ giữa các bảng trong cơ sở dữ liệu quan hệ (RDBMs) sử dụng mô hình khóa gồm 2 loại khóa: khóa chính và khóa phụ (primary key + foreign key) để ràng buộc dữ liệu nhằm thể hiện tính nhất quán dữ liệu từ các bảng khác nhau. Non-relational là khái niệm không sử dụng các ràng buộc dữ liệu cho nhất quán dữ liệu ở NoSQL database.
* **Distributed storage**: mô hình lưu trữ phân tán các file hoặc dữ liệu ra nhiều máy tính khác nhau trong mạng LAN hoặc Internet dưới sự kiểm soát của phần mềm.
* **Eventual consistency (nhất quán cuối)**: tính nhất quán của dữ liệu không cần phải đảm bảo ngay tức khắc sau mỗi phép write. Một hệ thống phân tán chấp nhận những ảnh hưởng theo phương thức lan truyền và sau một khoảng thời gian (không phải ngay tức khắc), thay đổi sẽ đi đến mọi điểm trong hệ thống, tức là cuối cùng (eventually) dữ liệu trên hệ thống sẽ trở lại trạng thái nhất quán.
* **Vertical scalable (khả năng mở rộng chiều dọc)**: Khi dữ liệu lớn về lượng, phương pháp tăng cường khả năng lưu trữ và xử lý bằng việc cải tiến phần mềm và cải thiện phần cứng trên một máy tính đơn lẻ được gọi là khả năng mở rộng chiều dọc. Ví dụ việc tăng cường CPUs, cải thiện đĩa cứng, bộ nhớ trong một máy tính,... cho DBMs nằm trong phạm trù này. Khả năng mở rộng chiều dọc còn có một thuật ngữ khác scale up.
* **Horizontal scalable (khả năng mở rộng chiều ngang)**: Khi dữ liệu lớn về lượng, phương pháp tăng cường khả năng lưu trữ và xử lý là dùng nhiều máy tính phân tán. Phân tán dữ liệu được hỗ trợ bởi phần mềm tức cơ sở dữ liệu.
* Trong khi giá thành phần cứng ngày càng giảm, tốc độ xử lý, bộ nhớ ngày càng tăng thì horizontal scalable là một lựa chọn đúng đắn. Hàng trăm máy tính nhỏ được chập lại tạo thành một hệ thống tính toán mạnh hơn nhiều so với vi xử lý RISC truyền thống đơn lẻ. Mô hình này tiếp tục được hỗ trợ bởi các công nghệ kết nối Myrinet và InfiniBand. Từ đó chúng ta có thể quản lý, bảo trì từ xa, xây dựng batch procession (xử lý đồng loạt tập lệnh) tốt hơn. Do những đòi hỏi về tốc độ xử lý I/O cao, lượng cực lớn dữ liệu,... scale horizontally sẽ thúc đẩy các công nghệ lưu trữ mới phát triển giống như object storage devices (OSD).

## Một số khái niệm mới trong NoSQL

* Fields – tương đương với khái niệm Columns trong SQL
* Document – thay thế khái niệm row trong SQL. Đây cũng chính là khái niệm làm nên sự khác biệt giữa NoSQL và SQL, 1 document chứa số cột (fields) không cố định trong khi 1 row thì số cột(columns) là định sẵn trước.
* Collection – tương đương với khái niệm table trong SQL. Một collection là tập hợp các document. Điều đặc biệt là một collection có thể chứa các document hoàn toàn khác nhau.
* Key-value – cặp từ khóa – giá trị được dùng để lưu trữ dữ liệu trong NoSQL
* Cursor – tạm dịch là con trỏ. Chúng ta sẽ sử dụng cursor để lấy dữ liệu từ database.
* Indexes ~ counterparts: Trong các hệ cơ sở dữ liệu quan hệ, các cột được định nghĩa theo bảng còn với hệ cơ sở  dữ liệu không ràng buộc, các cột được định nghĩa ở mỗi document. Bởi thế, các document quản lí gần như tất cả, các collection không cần quản lí chặt chẽ những gì đang xảy ra trong nó nữa .

# Kiến trúc

## Sơ lược

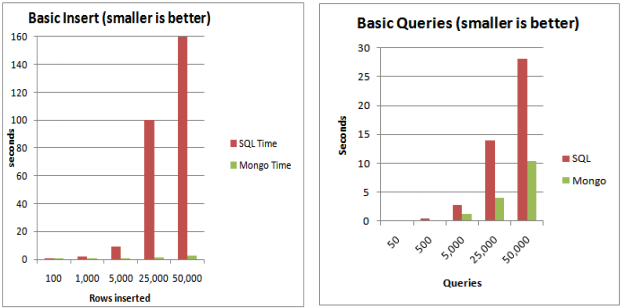
* Các RDBMs hiện tại đã bộc lộ những yếu kém như việc đánh chỉ mục một lượng lớn dữ liệu, phân trang, hoặc phân phối luồng dữ liệu media (phim, ảnh, nhạc, ...). Cơ sở dữ liệu quan hệ được thiết kế cho những mô hình dữ liệu nhỏ thường xuyên đọc viết trong khi các Social Network Services lại có một lượng dữ liệu cực lớn và cập nhật liên tục do số lượng người dùng quá nhiều ở một thời điểm. Thiết kế trên Distributed NoSQL giảm thiểu tối đa các phép tính toán, I/O liên quan kết hợp với batch processing đủ đảm bảo được yêu cầu xử lý dữ liệu của các mạng dịch vụ dữ liệu cộng đồng này. Facebook, Amazon là những ví dụ điểm hình.
* Về cơ bản, các thiết kế của NoSQL lựa chọn mô hình lưu trữ tập dữ liệu theo cặp giá trị key-value. Khái niệm node được sử dụng trong quản lý dữ liệu phân tán. Với các hệ thống phân tán, việc lưu trữ có chấp nhận trùng lặp dữ liệu. Một request truy vấn tới data có thể gửi tới nhiều máy cùng lúc, khi một máy nào nó bị chết cũng không ảnh hưởng nhiều tới toàn bộ hệ thống. Để đảm bảo tính real time trong các hệ thống xử lý lượng lớn, thông thường người ta sẽ tách biệt database ra làm 2 hoặc nhiều database. Một database nhỏ đảm bảo vào ra liên tục, khi đạt tới ngưỡng thời gian hoặc dung lượng, database nhỏ sẽ được gộp (merge) vào database lớn có thiết kế tối ưu cho phép đọc (read operation). Mô hình đó cho phép tăng cường hiệu suất I/O - một trong những nguyên nhân chính khiến performance trở nên kém.

## Một số đặc điểm

* High Scalability: Gần như không có một giới hạn cho dữ liệu và người dùng trên hệ thống.
* High Availability: Do chấp nhận sự trùng lặp trong lưu trữ nên nếu một node (commodity machine) nào đó bị chết cũng không ảnh hưởng tới toàn bộ hệ thống.
* Atomicity: Độc lập data state trong các operation.
* Consistency: chấp nhận tính nhất quán yếu, cập nhật mới không đảm bảo rằng các truy xuất sau đó thấy ngay được sự thay đổi. Sau một khoảng thời gian lan truyền thì tính nhất quán cuối cùng của dữ liệu mới được đảm bảo.
* Durability: dữ liệu có thể tồn tại trong bộ nhớ máy tính nhưng đồng thời cũng được lưu trữ lại đĩa cứng.
* Deployment Flexibility: việc bổ sung thêm/loại bỏ các node, hệ thống sẽ tự động nhận biết để lưu trữ mà không cần phải can thiệp bằng tay. Hệ thống cũng không đòi hỏi cấu hình phần cứng mạnh, đồng nhất.
* Modeling flexibility: Key-Value pairs, Hierarchical data (dữ liệu cấu trúc), Graphs.
* Query Flexibility: Multi-Gets, Range queries (load một tập giá trị dựa vào một dãy các khóa).

## What is NoSQL (technically speaking)?

* Đừng gọi chúng là database. CTO của Amazon, Werner Vogels đề cập đến hệ thống Dynamo của họ đã gọi nó là một "highly available key-value store". Google gọi BigTable để nhấn mạnh đây là "distributed storage system for managing structured data" (hệ thống lưu trữ và quản lý dữ liệu cấu trúc có phân tán).
* Có thể thổi bay một lượng dữ liệu cực lớn. Hypertable, một open source column-based database trên mô hình BigTable được sử dụng cho local search engine của Zvents Inc có thể ghi tới 1 tỷ cell dữ liệu mỗi ngày (theo Doug Judd một kỹ sư của Zvents). Trong khi đó BigTable kết hợp với MapReduce có thể xử lý tới 20 petabytes dữ liệu mỗi ngày.
* Đánh bại performance bottlenecks. Bằng việc bỏ qua thông dịch trong SQL cùng với những truy vấn rườm rà, NoSQL cho ta một kiến trúc tối ưu về tốc độ thực thi (ghi và truy vấn dữ liệu).
* Việc sử dụng các ràng buộc quan hệ cùng truy vấn SQL có vẻ thân thiện và thích hợp với phần đông dữ liệu. Tuy nhiên, nếu dữ liệu quá đơn giản, các thủ tục SQL sẽ không cần thiết (theo Curt Monash - một nhà phân tích cơ sở dữ liệu, một blogger).
* Raffaele Sena, một senior computer scientist ở Adobe Systems Inc. đã nói rằng ConnectNow Web collaboration service của họ sử dụng Java clustering software từ Terracotta thay cho cơ sở dữ liệu quan hệ đã khiến "hệ thống của họ trở nên mạnh hơn, phức tạp hơn so với việc sử dụng cơ sở dữ liệu quan hệ".
* Các thiết kế database có tính đặc thù (như document-oriented database) sẽ lược bỏ được tầng chuyển đổi sang mô hình lưu trữ quan hệ từ interface của nó đồng thời khiến giao tiếp tương tác trở nên tự nhiên hơn.



MongoDB vs. SQL Server 2008 Performance Showdown

* Không quá cần thiết. Đồng ý rằng RDBMs cung cấp một mô hình tuyệt vời để đảm bảo tính toàn vẹn dữ liệu. Tuy nhiên, rất nhiều người lựa chọn NoSQL đã nói rằng chúng không quá cần thiết cho nhu cầu của họ. Như trong dự án ConnectNow của Adobe, dữ liệu người dùng trong một session không cần thiết phải lưu lại, chúng sẽ bị xóa khi người dùng logoff. Vì vậy, một key-value memory storage là đủ dùng.

# Phân loại

## Core NoSQL Systems

### Wide Column Store / Column Families

* Column family databases được biết đến nhiều nhất qua sự triển khai BigTable của Google. Nhìn bên ngoài vào nó giống với cơ sỡ dữ liệu quan hệ nhưng thực sự thì có sự khác biệt rất lớn từ bên trong. Một trong những khác biệt đó chính là việc lưu trữ dữ liệu theo dòng (trong cơ sở dữ liệu quan hệ) so với việc lưu trữ dữ liệu theo cột (trong column family databases). Nhưng sự khác biệt lớn là ở chính khái niệm của nó. Chúng ta không thể áp dụng cùng một giải pháp mà chúng ta sử dụng trong cơ sở dữ liệu quan hệ vào trong cơ sở dữ liệu column family. Đó là bởi vì cơ sở dữ liệu cột (column family database) phi quan hệ. Các khái niệm sau đây rất quan trọng để hiểu được column family database làm việc như thế nào:
  + Column family
  + Super column
  + Column
* Column và super column trong column family database dùng thay thế nhau, có nghĩa là chúng sẽ là 0 byte nếu chúng không có chứa dữ liệu. Không giống như một bảng, thứ duy nhất chúng ta cần xác định trong column family database tên cột và các tùy chọn chính (không có lược đồ cố định).
* Ý tưởng cơ bản:
  + Column families: Một column family là cách thức dữ liệu được lưu trữ trên đĩa. Tất cả dữ liệu trong một cột sẽ được lưu trên cùng một file. Một column family có thể chứa super column hoặc column.
  + Super column: Một super column có thể được dùng như một dictionary(kiểu từ điển). Nó là một column có thể chứa những column khác (mà không phải là super column).
  + Column: Một column là một bộ gồm tên, giá trị và dấu thời gian (thông thường chỉ quan tâm tới key-value).
* Hiểu được lược đồ đượcc thiết kế như thế nào rất quan trọng. Nếu chúng ta thiết kế lược đồ không đúng, chúng ta không thể lấy được dữ liệu. Column family database cung cấp một trong hai cách truy vấn: key hoặc là key range. Điều này có nghĩa là, vì CFDB có thể phân tán nên khóa xác định vị trí vật lý thực sự mà dữ liệu được lưu trữ. Dữ liệu được lưu trữ dựa trên sự sắp xếp của column family và chúng ta không có cách nào để thay đổi sự sắp xếp này (ngoại trừ việc sắp xếp tăng dần hay giảm dần).
* Không giống như trong cơ sở dữ liệu quan hệ, thứ tự sắp xếp không bị ảnh hưởng bởi giá trị cột, nhưng lại bị ảnh hưởng bởi tên cột.
* Giả sử trong column family Users, một dòng với khóa chính là ”@ayende”, một cột name với giá trị “Ayende Rahien” và một cột location với giá trị “Israel”. CFDB sẽ sắp xếp vật lý trong User column family file như sau:

|  |
| --- |
| @ayende/location = "Israel"  @ayende/name = "Ayende Rahien" |

* Bởi vì location đứng trước name nên cột location sẽ được lưu trước cột name. Tương tự cho super column, ví dụ cột Friends, “@ayende” có 2 friend thì trong file Friends column family được lưu trữ vật lý như sau:

|  |
| --- |
| @ayende/friends/arava= 945  @ayende/friends/rose = 14 |

* Điều này rất quan trọng để hiểu được cách làm việc của CFDB. Giả sử chúng ta có mô hình twitter cần lưu trữ: users và tweets. Chúng ta định nghĩa 3 cột:
  + User: sắp xếp theo UTF8
  + Tweets: sắp xếp theo Sequential Guid
  + UsersTweets: super column, sắp xếp theo Sequential Guid
* Giả sử chúng ta muốn tạo một User:

|  |
| --- |
| cfdb.Users.Insert(key: "@ayende", name: "Ayende Rahien", location: "Israel", profession: "Wizard"); |

* Chúng ta có thể xem hình bên dưới để biết được một dòng trông như thế nào, nó không giống với một dòng trong cơ sở dữ liệu quan hệ:



Biểu diễn một dòng trong Column family database

* Giờ chúng ta sẽ tạo một tweet:

|  |
| --- |
| var firstTweetKey = "Tweets/" + SequentialGuid.Create();  cfdb.Tweets.Insert(key: firstTweetKey, application: "TweekDeck", text: "Err, is this on?", private: true;  var secondTweetKey = "Tweets/" + SequentialGuid.Create();  cfdb.Tweets.Insert(key: secondTweetKey, app: "Twhirl", version: "1.2", text: "Well, I guess this is my mandatory hello world”, public: true, version: 1.2); |

Hình 1.3: Biểu diễn 2 tweet trong Column family database

* Giá trị được hiển thị trong hình 1.3. Một vài lưu ý là:
  + Thực sự giá trị của khóa không quan trọng, nhưng nó là những chuỗi liên tiếp cho phép chúng ta sắp xếp sau này,
  + Cả hai dòng chứa dữ liệu khác nhau bởi vì chúng ta không có lược đồ cố định.
  + Chúng ta không có cách nào liên kết giữa một user và tweet
* Trong cơ sở dữ liệu quan hệ, chúng ta sẽ định nghĩa một cột là UserId trong Tweet cho phép chúng ta liên kết với bảng User. Hơn nữa, cơ sở dữ liệu quan hệ còn cho phép chúng ta truy vấn tweets theo UserId. CFDB không làm được điều này, chúng ta không thể truy vấn theo dữ liệu cột. Thay vì thế, điều duy nhất CFDB cho phép chúng ta là truy vấn theo khóa. Chúng ta định nghĩa một index thứ hai, nơi mà cột UsersTweets xuất hiện:

|  |
| --- |
| cfdb.UsersTweets.Insert(key: "@ayende", timeline: { SequentialGuid.Create(): firstTweetKey } );  cfdb.UsersTweets.Insert(key: "@ayende", timeline: { SequentialGuid.Create(): secondTweetKey } ); |

* Hình 1.4 cho thấy dữ liệu trong database như thế nào. Chúng ta thêm dữ liệu vào cột UsersTweets một dòng với khóa là “@ayende”, super column timeline với 2 cột – tên mỗi cột là sequential guid cho phép sắp xếp.

Hình 1.4: biểu diễn index thứ hai, liên kết users và tweets trong Column Family database.

* Lưu ý: Câu hỏi đặt ra là chúng ta có thể tạo ra super column trong cột User để lưu giữ quan hệ. Câu trả lời là: chúng ta có thể làm điều đó, ngoại trừ một column family có thể chứa hoặc column hoặc super column, không thể chứa cả hai.
* Để lấy những tweets của một user, chúng ta cần thực thi:

|  |
| --- |
| var tweetIds =cfdb.UsersTweets.Get("@ayende")  .FetchSuperColumnValues("timeline")  .Take(25)  .OrderByDescending()  .Select(x=>x.Value);  var tweets = cfdb.Tweets.Get(tweetIds); |

* Lưu ý: truy vấn này không phải là API cho .NET, chỉ là ví dụ cho dễ hiểu không phải là API thực sự.
* Về cơ bản chúng ta thực hiện 2 truy vấn:
  + Truy vấn thứ nhất vào cột UsersTweets, yêu cầu cột và giá trị trong timeline super column với khóa là “@ayende”
  + Truy vấn thứ hai là truy vấn vào cột Tweets để lấy giá trị thực sự của Tweet.
* Kiểu này chúng ta thường thấy trong NoSQL. Nó được gọi là ”secondary index”, một cách truy cập dữ liệu nhanh chóng theo khóa dựa trên một giá trị khác entity/row/document. Đây là một ví dụ về truy vấn Tweeys theo User trên dữ liệu chúng ta có. Nếu không tạo ra secondary index thì chúng ta không có cách nào trả lời cho câu hỏi: “cho xem 25 tweets mới nhất của ayende?”
* Bởi vì dữ liệu được sắp xếp theo tên cột và giảm dần, chúng ta có thể lấy được 25 tweets mới nhất của ayende. Và sẽ như thế nào nếu ta muốn truy vấn 25 tweets mới nhất (không theo một user nào)? Rất đơn giản, chỉ cần truy vấn vào cột Tweets sắp xếp theo khóa giảm dần.

Tại sao column family database lại bị giới hạn?

* CFDB khó khăn trong việc nghiên cứu lúc đầu ví nó nhìn bên ngoài thì giống cơ sở dữ liệu quan hệ mà lại bị hạn chế rất nhiều. Không có “join”, không có khả năng truy vấn thực sự(ngoại trừ truy vấn theo khóa chính), không có sự phong phú sự hỗ trợ như cơ sở dữ liệu quan hệ làm được. SQLite hay Access đem lại nhiều lợi ích hơn CFDB. Tại sao CFDB lại hạn chế như vậy?
* Câu trả lời khá đơn giản. CFDB được thiết kế đê chạy trên một số lượng lớn các máy, và lưu trữ một lượng dữ liệu cực lớn. Chúng ta không thể lưu trữ một lượng lớn dữ liệu trong cơ sở dữ liệu quan hệ thậm chí là nhiều máy như Oracle RAC sẽ nhanh chóng bị sụp đổ hoặc là chết rất nhanh về kích thước của dữ liệu và những truy vấn đó được CFDB xử lý một cách dễ dàng. Nhớ rằng CFDB loại bỏ các khái niệm trừu tượng, những thứ làm cho nó cứng nhắt khi chạy trên một cụm máy.
* Lý do mà CFDB không hỗ trợ join là join yêu cầu chúng ta quét toàn bộ tập hợp dữ liệu. Điều đó yêu cầu phải có một nơi có cái nhìn tổng quát về toàn bộ cơ sở dữ liệu(kết quả trong nút cổ chai và điểm duy nhất bị thất bại) hoặc thực hiện một truy vấn thực sự trên tất cả các máy có trong cụm. CFDB không cung cấp cách thức để truy vấn theo cột hay giá trị bởi vì nó sẽ yêu cầu một index trên toàn bộ tập hợp dữ liệu (hay chỉ là một cột duy nhất). Và một lần nữa không thực tế, không thể chạy truy vấn trên tất cả các máy. Bằng cách giới hạn truy vấn theo khóa, CFDB đảm bảo rằng nó biết chính xác về node mà truy vấn có thể thực hiện trên đó. Có nghĩa là mỗi truy vấn được chạy trên một tập nhỏ dữ liệu là cho chi phí rẻ hơn nhiều.
* Nhiều hạn chế, khó sử dụng nhưng CFDB lại có khả năng mở rộng cao. Đây là điều chúng ta cần quan tâm tới.

### Key-Value Store/Tuple store

* Cơ sở dữ liệu NoSQL đơn giản nhất chính là Key/Value stores. Nó đơn giản nhất là vì những API của nó đơn giản, những triển khai thực tế của NoSQL thường rất phức tạp. Hầu hết Key/Value stores thường có những API sau:

|  |
| --- |
| void Put(string key, byte[] data);  byte[] Get(string key);  void Remove(string key); |

* Có rất nhiều biến thể nhưng Key/Value store là cơ sở cho tất cả những biến thể đó. Một Key/Value store cho phép chúng ta lưu trữ giá trị bởi khóa (key-value). Giá trị được lưu dưới dạng BLOB (Binary large object). Đơn giản là lưu trữ dữ liệu mà không quan tâm đến nội dung lưu trữ. Nói cách khác, chúng ta lưu trữ dữ liệu mà không cần xác định lược đồ, nhưng phía client thì có định nghĩa mang tính tham khảo để biết dữ liệu thực sự như thế nào. Lợi ích của phương pháp này là rất đơn giản để xây dựng một key/value store và nó rất dễ dàng mở rộng. Một key/value store có hiệu suất rất tốt bởi vì mô hình truy cập dữ liệu trong key/value store được chú trọng rất nhiều vào việc tối ưu hóa. Nói chúng, hầu hết các thao tác key/value được thực hiện sử dụng O(1) bất kể có bao nhiêu máy dùng để lưu trữ dữ liệu và dữ liệu có độ lớn như thế nào.

### Document Store

#### Giới thiệu

* Document store thực chất là các document-oriented database (cơ sở dữ liệu hướng tài liệu) – một thiết kế riêng biệt cho việc lưu trữ document.
* Nó là một chương trình máy tính được thiết kế dùng để lưu trữ, truy lại và quản lý hướng tài liệu, hoặc dữ liệu bán cấu trúc ,hoặc thông tin.
* Document-oriented databases là một trong những loại chính của cơ sở dữ liệu NoSQL được biết đến với thuật ngữ “document-oriented document” (hay là “document store”). Trái ngược lại với cơ sở dữ liệu quan hệ truyền thống-quan niệm về “Relations” (hay “Tables”), những hệ thống này được thiết kế xung quanh một khái niệm trừu tượng gọi là “document”.

#### Document (tài liệu)

* Khái niệm trung tâm của document-oriented database là khái niệm “document”. Trong khi mỗi loại document-oriented database được triển khai không giống nhau ở phần chi tiết so với định nghĩa thì nói chung tất cả đều giả định các tài liệu đóng gói và mã hóa dữ liệu (thông tin) trong một số định dạng tiêu chuẩn hoặc mã hóa. Một số kiểu mã hóa được sử dụng bao gồm XML, YAML, JSON, và BSON, cũng như các hình thức nhị phân như PDF và các tài liệu Microsoft Office (MS Word, Excel …).
* Các document (tài liệu) bên trong một document-oriented database thì tương tự nhau,nó gần giống với khái niệm “records” hay “row” trong cơ sở dữ liệu quan hệ truyền thống nhưng nó ít cứng nhắc hơn. Documents không bắt buộc phải tuân theo một lược đồ tiêu chuẩn cũng không cần phải có tất cả các phần, chìa khóa, tương tự nhau. Ví dụ đây là một document:
  + *FirstName:"Bob", Address:"5 Oak St.", Hobby:"sailing".*

Một document khác có thể là:

* + *FirstName:"Jonathan", Address:"15 Wanamassa Point Road", Children:[{Name:"Michael",Age:10}, {Name:"Jennifer", Age:8}, {Name:"Samantha", Age:5}, {Name:"Elena", Age:2}].*
* Cả hai document trên có một số thông tin tương tự và một số thông tin khác nhau. Không giống như một cơ sở dữ liệu quan hệ truyền thống, nơi mỗi record(row) có cùng một tập hợp trường dữ liệu (fields hay columns) và các trường dữ liệu này nếu không được sử dụng thì có thể được lưu trữ trống, còn trong một document-oriented database thì không có trường dữ liệu trống trong document. Hệ thống này cho phép thông tin mới được thêm vào và nó không đòi hỏi phải khai báo rõ ràng là khác những mẩu thông tin còn lại.

#### Keys (Khóa)

* Các document được đề cập trong document-oriented database thông qua một khóa duy nhất đại diện cho documnet đó. Thông thường, khóa này là một chuỗi đơn giản. Trong một số trường hợp, chuỗi này có thể là một URI hoặc đường dẫn (path). Bạn có thể sử dụng khóa này để lấy document (tài liệu)từ cơ sở dữ liệu. Thông thường, cơ sở dữ liệu vẫn lưu lại một chỉ số (index)trong khóa của document để document có thể được tìm kiếm nhanh chóng.

#### Retrieval (truy lại hay tìm kiếm thông tin)

* Một trong các đặc điểm khác biệt của một document-oriented database là việc sử dụng key-document (hoặc key-value) để lấy một tài liệu, cơ sở dữ liệu sẽ cung cấp một API hoặc ngôn ngữ truy vấn cho phép bạn lấy các tài liệu dựa trên nội dung . Ví dụ, bạn muốn truy vấn lấy toàn bộ document mà những document đó có tập hợp trường dữ liệu nhất định với những giá trị nhất định. Tập hợp các API truy vấn hoặc ngôn ngữ truy vấn là tính năng có sẵn, cũng như hiệu năng mong đợi của các truy vấn, thay đổi đáng kể từ một trong những việc triển khai.

#### Organization (Tổ chức)

* Các triển khai cung cấp một loạt các cách sắp xếp tài liệu, bao gồm khái niệm về:
  + Collections
  + Tags
  + Non-visible Metadata
  + Directory hierarchies

#### Implementations

| **Name** | **Publisher** | **License** | **Language** | **Notes** | [**RESTful**](http://en.wikipedia.org/wiki/Representational_State_Transfer)**API** |
| --- | --- | --- | --- | --- | --- |
| [eXist](http://en.wikipedia.org/wiki/EXist) | [eXist](http://en.wikipedia.org/wiki/EXist), [[2]](http://exist-db.org/) | [GPL](http://en.wikipedia.org/wiki/GPL) | [XQuery](http://en.wikipedia.org/wiki/XQuery), [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) | XML over REST/HTTP, WebDAV, Lucene Fulltext search, validation, versioning, clustering, triggers, URL rewriting, collections, ACLS, XQuery Update | Yes [[4]](http://en.wikipedia.org/wiki/Document-oriented_database#cite_note-3) |
| [MUMPS](http://en.wikipedia.org/wiki/MUMPS)Database[[7]](http://en.wikipedia.org/wiki/Document-oriented_database#cite_note-6) |  | [Proprietary](http://en.wikipedia.org/wiki/Proprietary_software) and GNU Affero GPL[[8]](http://en.wikipedia.org/wiki/Document-oriented_database#cite_note-7) | [MUMPS](http://en.wikipedia.org/wiki/MUMPS) | Commonly used in health applications. | (unknown) |
| [Lotus Notes](http://en.wikipedia.org/wiki/Lotus_Notes) | IBM | [Proprietary](http://en.wikipedia.org/wiki/Proprietary_software) | [LotusScript](http://en.wikipedia.org/wiki/LotusScript),[Java](http://en.wikipedia.org/wiki/Java_(programming_language)), Lotus @Formula |  | (unknown) |
| [BaseX](http://en.wikipedia.org/wiki/BaseX) | [BaseX Team](http://basex.org/) | [BSD License](http://en.wikipedia.org/wiki/BSD_License) | [Java](http://en.wikipedia.org/wiki/Java_(programming_language)), [XQuery](http://en.wikipedia.org/wiki/XQuery) | Support for XML, JSON and binary formats; client-/server based architecture; concurrent structural and full-text searches and updates; REST APIs. | Yes |
| [OrientDB](http://en.wikipedia.org/wiki/OrientDB) | [Orient Technologies](http://www.orientechnologies.com/) | [Apache License](http://en.wikipedia.org/wiki/Apache_License) | [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) | JSON over HTTP | Yes |
| [Terrastore](http://en.wikipedia.org/wiki/Terrastore) |  | [Apache License](http://en.wikipedia.org/wiki/Apache_License) | [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) | JSON/HTTP | (unknown) |
| [Jackrabbit](http://en.wikipedia.org/wiki/Apache_Jackrabbit) | [Apache Software Foundation](http://en.wikipedia.org/wiki/Apache_Software_Foundation) | [Apache License](http://en.wikipedia.org/wiki/Apache_License) | [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) |  | (unknown) |
| [CouchDB](http://en.wikipedia.org/wiki/CouchDB) | [Couchbase](http://en.wikipedia.org/wiki/Couchbase),[Apache Software Foundation](http://en.wikipedia.org/wiki/Apache_Software_Foundation) | [Apache License](http://en.wikipedia.org/wiki/Apache_License) | [Erlang](http://en.wikipedia.org/wiki/Erlang_(programming_language)) | JSON over REST/HTTP with [Multi-Version Concurrency Control](http://en.wikipedia.org/wiki/Multi-Version_Concurrency_Control) and limited [ACID](http://en.wikipedia.org/wiki/ACID) properties. Uses[map](http://en.wikipedia.org/wiki/Map_(higher-order_function)) and [reduce](http://en.wikipedia.org/wiki/Fold_(higher-order_function)) for views and queries.[[2]](http://en.wikipedia.org/wiki/Document-oriented_database#cite_note-1) | Yes [[3]](http://en.wikipedia.org/wiki/Document-oriented_database#cite_note-2) |
| [FleetDB](http://en.wikipedia.org/wiki/FleetDB) | [FleetDB](http://fleetdb.org/) | [MIT License](http://en.wikipedia.org/wiki/MIT_License) | [Clojure](http://en.wikipedia.org/wiki/Clojure) | A [JSON-based](http://fleetdb.org/docs/protocol.html) schema-free database optimized for agile development. | (unknown) |
| [MarkLogic](http://en.wikipedia.org/wiki/MarkLogic) | MarkLogic Corporation | [Free Express license](http://developer.marklogic.com/licensing) or[Commercial](http://developer.marklogic.com/licensing) | [C++](http://en.wikipedia.org/wiki/C%2B%2B), [XQuery](http://en.wikipedia.org/wiki/XQuery),[XSLT](http://en.wikipedia.org/wiki/XSLT) | Fast, scalable, distributed, enterprise-grade document-oriented database with [Multi-Version Concurrency Control](http://en.wikipedia.org/wiki/Multi-Version_Concurrency_Control), integrated [Full text search](http://en.wikipedia.org/wiki/Full_text_search) and [ACID](http://en.wikipedia.org/wiki/ACID)-compliant transaction semantics | Yes |
| [ThruDB](http://code.google.com/p/thrudb/) |  | [BSD License](http://en.wikipedia.org/wiki/BSD_License) | [C++](http://en.wikipedia.org/wiki/C%2B%2B), [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) | Built on top of [Apache Thrift](http://en.wikipedia.org/wiki/Apache_Thrift) framework that provides indexing and document storage services for building and scaling websites. Alternate implementation is being developed in Java. [Alpha software](http://en.wikipedia.org/wiki/Software_release_life_cycle#Alpha). | (unknown) |
| [Clusterpoint](http://en.wikipedia.org/wiki/Clusterpoint) | [Clusterpoint Ltd.](http://www.clusterpoint.com/) | Free community license /[Commercial](http://en.wikipedia.org/wiki/Commercial_software)[[1]](http://en.wikipedia.org/wiki/Document-oriented_database#cite_note-0) | [C++](http://en.wikipedia.org/wiki/C%2B%2B) | Scalable, high-performance, schema-free, document-oriented [database management system](http://en.wikipedia.org/wiki/Database_management_system)platform with server based data storage, fast [full text search](http://en.wikipedia.org/wiki/Full_text_search) engine functionality, information[ranking](http://en.wikipedia.org/wiki/Ranking) for search relevance and [clustering](http://en.wikipedia.org/wiki/Cluster_computing). | Yes |
| [MongoDB](http://en.wikipedia.org/wiki/MongoDB) | 10gen, Inc | [GNU AGPL v3.0](http://en.wikipedia.org/wiki/Affero_General_Public_License)[[5]](http://en.wikipedia.org/wiki/Document-oriented_database#cite_note-4) | [C++](http://en.wikipedia.org/wiki/C%2B%2B) | Fast, document-oriented database optimized for highly transient data. | Optional [[6]](http://en.wikipedia.org/wiki/Document-oriented_database#cite_note-5) |
| [RavenDB](http://en.wikipedia.org/w/index.php?title=RavenDB&action=edit&redlink=1) | [Hibernating Rhinos](http://hibernatingrhinos.com/) | [AGPL](http://en.wikipedia.org/wiki/Affero_General_Public_License) Free or[Commercial](http://ravendb.net/licensing/) | [C#](http://en.wikipedia.org/wiki/C_Sharp_(programming_language)), [F#](http://en.wikipedia.org/wiki/F_Sharp_(programming_language)),[VB.NET](http://en.wikipedia.org/wiki/VB.NET) | JSON over REST/HTTP with a rich .Net client API. [ACID](http://en.wikipedia.org/wiki/ACID) compliant. Uses [Lucene.net](http://en.wikipedia.org/wiki/Lucene.net) to provide indexes using a [LINQ](http://en.wikipedia.org/wiki/LINQ) interface, optionally with [MapReduce](http://en.wikipedia.org/wiki/Map_reduce) functionality. | Yes [[9]](http://en.wikipedia.org/wiki/Document-oriented_database#cite_note-8) |
| [Redis](http://en.wikipedia.org/wiki/Redis) |  | [BSD License](http://en.wikipedia.org/wiki/BSD_License) | [ANSI C](http://en.wikipedia.org/wiki/ANSI_C) | Key-value store supporting lists and sets with fast, simple and binary-safe protocol. | (unknown) |
| [Rocket U2](http://en.wikipedia.org/wiki/Rocket_U2) | Rocket Software | [Proprietary](http://en.wikipedia.org/wiki/Proprietary_software) |  | UniData, UniVerse | Yes (Beta) |

### Graph Database

* Graph database là một dạng cơ sở dữ liệu được thiết kế riêng cho việc lưu trữ thông tin đồ họa như cạnh, nút, properties. Một số sản phẩm tiêu biểu: Neo4J, Sones, AllegroGraph, Core Data, DEX, FlockDB, InfoGrid, OpenLink Virtuoso,...

| **Name** | **Language** | **Notes** |
| --- | --- | --- |
| [AllegroGraph](http://en.wikipedia.org/wiki/AllegroGraph) | [SPARQL](http://en.wikipedia.org/wiki/SPARQL) | [RDF](http://en.wikipedia.org/wiki/Resource_Description_Framework) GraphStore |
| [DEX](http://en.wikipedia.org/wiki/DEX_(Graph_database)) | [Java](http://en.wikipedia.org/wiki/Java_(programming_language)), [C++](http://en.wikipedia.org/wiki/C%2B%2B) | High-performance [Graph Database](http://en.wikipedia.org/wiki/Graph_Database) |
| [FlockDB](http://en.wikipedia.org/wiki/FlockDB) | [Scala](http://en.wikipedia.org/wiki/Scala_(programming_language)) |  |
| [InfiniteGraph](http://en.wikipedia.org/wiki/InfiniteGraph) | [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) | High-performance, scalable, distributed [Graph Database](http://en.wikipedia.org/wiki/Graph_Database) |
| [Neo4j](http://en.wikipedia.org/wiki/Neo4j) | [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) |  |
| [OpenLink Virtuoso](http://en.wikipedia.org/wiki/Virtuoso_Universal_Server) | [C++](http://en.wikipedia.org/wiki/C%2B%2B), [C#](http://en.wikipedia.org/wiki/C_Sharp_(programming_language)), [Java](http://en.wikipedia.org/wiki/Java_(programming_language)), [SPARQL](http://en.wikipedia.org/wiki/SPARQL) | [middleware](http://en.wikipedia.org/wiki/Middleware) and [database engine](http://en.wikipedia.org/wiki/Database_engine) hybrid |
| [OrientDB](http://en.wikipedia.org/wiki/OrientDB) | [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) |  |
| [Pregel](http://en.wikipedia.org/wiki/Pregel) |  |  |
| [Sones GraphDB](http://en.wikipedia.org/wiki/Sones_GraphDB) | [C#](http://en.wikipedia.org/wiki/C_Sharp_(programming_language)) |  |
| [OWLIM](http://en.wikipedia.org/wiki/Ontotext) | [Java](http://en.wikipedia.org/wiki/Java_(programming_language)), [SPARQL 1.1](http://en.wikipedia.org/wiki/SPARQL) | [RDF](http://en.wikipedia.org/wiki/Resource_Description_Framework) graph store with reasoning |