Relation Model and Document model

Sự ra đời của NoQuery

Bây giờ, vào những năm 2010, NoQuery là nỗ lực mới nhất nhằm lật đổ sự thống trị của mô hình quan hệ. Cái tên không có gì đáng tin cậy vì nó không thực sự đề cập đến bất kỳ công nghệ cụ thể nào, nó ban đầu chỉ đơn giản là một hashtag hấp dẫn cho một cuộc gặp gỡ trên cơ sở dữ liệu mở, phân tán, không liên quan trong năm 2009 [Tuy nhiên, thuật ngữ này gây ra sự căng thẳng và nhanh chóng lan truyền thông qua cộng đồng khởi nghiệp web và hơn thế nữa. Một số hệ thống cơ sở dữ liệu thú vị hiện được liên kết với hashtag #NoQuery và nó đã được giải thích lại theo cách hồi tố là Không chỉ SQL [4].

Có một số động lực đằng sau việc áp dụng cơ sở dữ liệu NoQuery, bao gồm:

• Nhu cầu về khả năng mở rộng lớn hơn cơ sở dữ liệu quan hệ có thể dễ dàng đạt được, bao gồm các bộ dữ liệu rất lớn hoặc thông lượng ghi rất cao

• Ưu tiên rộng rãi cho phần mềm miễn phí và nguồn mở đối với các sản phẩm cơ sở dữ liệu thương mại

• Các hoạt động truy vấn chuyên biệt không được mô hình quan hệ hỗ trợ tốt

• Thất vọng với sự hạn chế của các lược đồ quan hệ và mong muốn một mô hình dữ liệu năng động và biểu cảm hơn.

Các ứng dụng khác nhau có các yêu cầu khác nhau và sự lựa chọn công nghệ tốt nhất cho một trường hợp sử dụng cũng có thể khác với lựa chọn tốt nhất cho trường hợp sử dụng khác. Do đó, dường như trong tương lai gần, các cơ sở dữ liệu quan hệ sẽ tiếp tục được sử dụng cùng với nhiều kho dữ liệu không liên quan khác nhau, một ý tưởng đôi khi được gọi là sự tồn tại của polyglot [3].

Sự không phù hợp đối tượng

Hầu hết việc phát triển ứng dụng ngày nay được thực hiện bằng các ngôn ngữ lập trình hướng đối tượng, dẫn đến sự chỉ trích chung về mô hình dữ liệu SQL: nếu dữ liệu được lưu trữ trong các bảng quan hệ, thì cần có một lớp dịch vụng giữa các đối tượng trong mã ứng dụng và mô hình cơ sở dữ liệu của bảng, hàng và cột. Việc ngắt kết nối giữa các mô hình đôi khi được gọi là không khớp trở kháng

Các khung ánh xạ quan hệ đối tượng (ORM) như ActiveRecord và Hibernate giảm số lượng mã soạn sẵn cần thiết cho lớp dịch này, nhưng chúng không thể che giấu hoàn toàn sự khác biệt giữa hai mô hình.

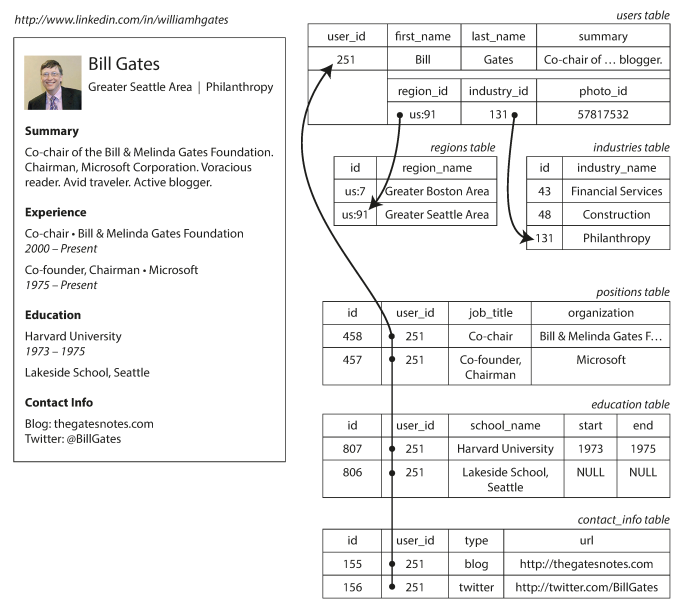
Ví dụ, Hình 2-1 minh họa cách một sơ yếu lý lịch (hồ sơ LinkedIn) có thể được thể hiện trong một lược đồ quan hệ. Toàn bộ hồ sơ có thể được xác định bởi một định danh duy nhất, user\_id. Các trường như First\_name và last\_name xuất hiện chính xác một lần cho mỗi người dùng, vì vậy chúng có thể được mô hình hóa dưới dạng cột trên bảng của người dùng. Tuy nhiên, hầu hết mọi người đã có nhiều hơn một công việc trong sự nghiệp (vị trí) và mọi người có thể thay đổi số lượng thời gian giáo dục và bất kỳ số lượng thông tin liên lạc.

Có một mối quan hệ một-nhiều từ người dùng đến các mục này, có thể được thể hiện theo nhiều cách khác nhau:

• Trong mô hình SQL truyền thống (trước SQL: 1999), biểu diễn chuẩn hóa phổ biến nhất là đặt các vị trí, giáo dục và thông tin liên hệ vào các bảng riêng biệt, với tham chiếu khóa ngoài tới bảng của người dùng, như trong Hình 2-1.

• Các phiên bản sau của tiêu chuẩn SQL đã thêm hỗ trợ cho các kiểu dữ liệu có cấu trúc và dữ liệu XML; điều này cho phép dữ liệu đa giá trị được lưu trữ trong một hàng đơn, với sự hỗ trợ cho truy vấn và lập chỉ mục bên trong các tài liệu đó. Các tính năng này được hỗ trợ ở các mức độ khác nhau của Oracle, IBM DB2, MS SQL Server và PostgreQuery [6, 7]. Một kiểu dữ liệu JSON cũng được hỗ trợ bởi một số cơ sở dữ liệu, bao gồm IBM DB2, MySQL và PostgreQuery [8].

• Tùy chọn thứ ba là mã hóa các công việc, giáo dục và thông tin liên hệ dưới dạng tài liệu JSON hoặc XML, lưu trữ nó trên một cột văn bản trong cơ sở dữ liệu và để ứng dụng diễn giải cấu trúc và nội dung của nó. Trong thiết lập này, bạn thường không thể sử dụng cơ sở dữ liệu để truy vấn các giá trị bên trong cột được mã hóa đó.



Hình 2-1. Đại diện cho một hồ sơ LinkedIn bằng cách sử dụng một lược đồ quan hệ. Hình ảnh của Bill Gates với sự giúp đỡ của Wikimedia Commons, Ricardo Stuckert, Agência Brasil.

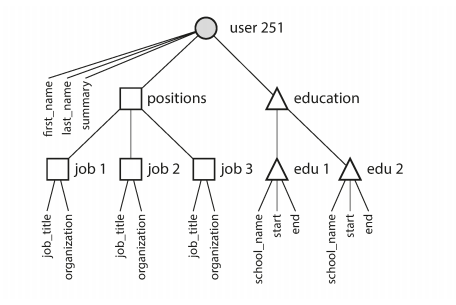
Đối với cấu trúc dữ liệu như sơ yếu lý lịch, phần lớn là tài liệu độc lập, cách trình bày JSON có thể khá phù hợp: xem Ví dụ 2-1. JSON có sức hấp dẫn là đơn giản hơn nhiều so với XML. Các cơ sở dữ liệu hướng tài liệu như MongoDB [9], RethinkDB [10], CouchDB [11] và Espresso [12] hỗ trợ mô hình dữ liệu này.

Example 2-1. Representing a LinkedIn profile as a JSON document{  
"user\_id": 251,  
"first\_name": "Bill",  
"last\_name": "Gates",  
"summary": "Co-chair of the Bill & Melinda Gates... Active blogger.",  
"region\_id": "us:91",  
"industry\_id": 131,  
"photo\_url": "/p/7/000/253/05b/308dd6e.jpg",  
"positions": [  
{"job\_title": "Co-chair", "organization": "Bill & Melinda Gates Foundation"},  
{"job\_title": "Co-founder, Chairman", "organization": "Microsoft"}  
],  
"education": [  
{"school\_name": "Harvard University", "start": 1973, "end": 1975},  
{"school\_name": "Lakeside School, Seattle", "start": null, "end": null}  
],  
"contact\_info": {  
"blog": "http://thegatesnotes.com",  
"twitter": "http://twitter.com/BillGates"  
}  
}

Một số nhà phát triển cảm thấy rằng mô hình JSON làm giảm sự không phù hợp trở kháng giữa mã ứng dụng và lớp lưu trữ. Tuy nhiên, như chúng ta sẽ thấy trong Chương 4, cũng có vấn đề với JSON là định dạng mã hóa dữ liệu. Việc thiếu một lược đồ là thường được trích dẫn là một lợi thế; chúng tôi sẽ thảo luận về tính linh hoạt của Schema trong mô hình tài liệu trên trang 39.

Biểu diễn JSON có một địa phương tốt hơn so với lược đồ nhiều bảng trong Hình 2-1. Nếu bạn muốn tìm nạp một hồ sơ trong ví dụ quan hệ, bạn cần thực hiện nhiều truy vấn (truy vấn từng bảng theo user\_id) hoặc thực hiện nối nhiều đường lộn xộn giữa bảng của người dùng và các bảng phụ của nó. Trong biểu diễn JSON, tất cả các thông tin liên quan đều ở một nơi và một truy vấn là đủ.

Các mối quan hệ một-nhiều từ hồ sơ người dùng đến các vị trí người dùng, lịch sử giáo dục và thông tin liên hệ ngụ ý cấu trúc cây trong dữ liệu và biểu diễn JSON làm cho cấu trúc cây này rõ ràng (xem Hình 2-2).



Many-to-One and Many-to-Many Relationships

Trong ví dụ 2-1 trong phần trước, area\_id và industrial\_id được cung cấp dưới dạng ID, không phải là chuỗi văn bản đơn giản "Khu vực Greater Seattle" và "Từ thiện". Tại sao? Nếu giao diện người dùng có các trường văn bản tự do để vào khu vực và ngành, thì việc lưu trữ chúng dưới dạng các chuỗi văn bản đơn giản. Nhưng có những lợi thế khi có danh sách tiêu chuẩn của các khu vực và ngành địa lý và cho phép người dùng chọn từ danh sách thả xuống hoặc tự động hoàn thành:

• Phong cách nhất quán và chính tả trên các hồ sơ

• Tránh sự mơ hồ (ví dụ: nếu có một số thành phố có cùng tên)

• Dễ cập nhật, tên chỉ được lưu trữ ở một nơi, do đó rất dễ cập nhật trên bảng nếu cần thay đổi (ví dụ: thay đổi tên thành phố do các sự kiện chính trị)

• Hỗ trợ bản địa hóa khi trang web được dịch sang các ngôn ngữ khác, các danh sách được tiêu chuẩn hóa có thể được bản địa hóa, do đó, khu vực và ngành có thể được hiển thị bằng ngôn ngữ của người xem

• Tìm kiếm tốt hơn, ví dụ, một tìm kiếm cho các nhà hảo tâm ở tiểu bang Washington có thể phù hợp với hồ sơ này vì danh sách các khu vực có thể mã hóa thực tế rằng Seattle đang ở Washington (không rõ ràng từ chuỗi "Khu vực Greater Seattle")

Cho dù bạn lưu trữ một ID hoặc một chuỗi văn bản là một câu hỏi về sự trùng lặp. Khi bạn sử dụng ID, thông tin có ý nghĩa đối với con người (chẳng hạn như từ Philanthropy) chỉ được lưu trữ ở một nơi và mọi thông tin liên quan đến nó đều sử dụng ID (chỉ có ý nghĩa trong cơ sở dữ liệu). Khi bạn lưu trữ văn bản trực tiếp, bạn đang sao chép

thông tin có ý nghĩa con người trong mỗi hồ sơ sử dụng nó.

Ưu điểm của việc sử dụng ID là vì nó không có ý nghĩa đối với con người, nên nó không bao giờ cần thay đổi: ID có thể giữ nguyên, ngay cả khi thông tin mà nó xác định

thay đổi. Bất cứ điều gì có ý nghĩa đối với con người có thể cần phải thay đổi đôi khi trong tương lai và nếu thông tin đó bị trùng lặp, tất cả các bản sao dự phòng cần phải được cập nhật. Điều đó phát sinh chi phí viết và rủi ro không nhất quán (trong đó một số bản sao của thông tin được cập nhật nhưng một số khác lại không được). Loại bỏ sự trùng lặp như vậy là ý tưởng chính đằng sau việc chuẩn hóa trong cơ sở dữ liệu.

Thật không may, việc bình thường hóa dữ liệu này đòi hỏi các mối quan hệ nhiều-một (nhiều người sống trong một khu vực cụ thể, nhiều người làm việc trong một ngành cụ thể), điều này không phù hợp với mô hình tài liệu. Trong các cơ sở dữ liệu quan hệ, nó rất bình thường khi tham chiếu các hàng trong các bảng khác bằng ID, vì các phép nối rất dễ dàng. Trong cơ sở dữ liệu tài liệu, các phép nối không cần thiết cho các cấu trúc cây một-nhiều và sự hỗ trợ cho các phép nối thường yếu.

Nếu cơ sở dữ liệu không hỗ trợ các phép nối, bạn phải mô phỏng phép nối mã ứng dụng bằng cách thực hiện nhiều truy vấn tới cơ sở dữ liệu. (Trong trường hợp này, danh sách các vùng và ngành có thể nhỏ và thay đổi chậm đến mức ứng dụng có thể chỉ cần giữ chúng trong bộ nhớ. Tuy nhiên, công việc tạo liên kết bị thay đổi

từ cơ sở dữ liệu đến mã ứng dụng.)

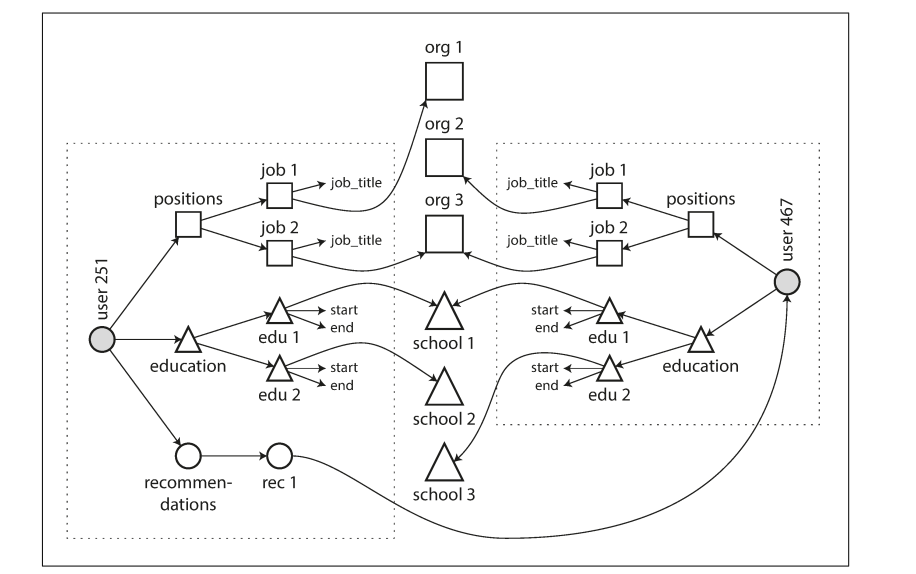
Ngoài ra, ngay cả khi phiên bản ban đầu của ứng dụng phù hợp với mô hình tài liệu không tham gia, dữ liệu có xu hướng trở nên kết nối với nhau hơn khi các tính năng được thêm vào ứng dụng. Ví dụ, hãy xem xét một số thay đổi mà chúng tôi có thể thực hiện đối với ví dụ sơ yếu lý lịch:

Các tổ chức và trường học là các thực thể Trong mô tả trước đó, tổ chức (công ty nơi người dùng làm việc) và school\_name (nơi họ nghiên cứu) chỉ là các chuỗi. Có lẽ họ nên được tham chiếu đến các thực thể thay thế? Sau đó, mỗi tổ chức, trường học hoặc trường đại học có thể có trang web riêng (có logo, nguồn cấp tin tức, v.v.); mỗi sơ yếu lý lịch có thể liên kết với các tổ chức và trường học mà nó đề cập, và bao gồm logo và thông tin khác của họ (xem Hình 2-3 để biết ví dụ từ LinkedIn).

khuyến nghị

Giả sử bạn muốn thêm một tính năng mới: một người dùng có thể viết đề xuất cho người dùng khác. Khuyến nghị được hiển thị trên sơ yếu lý lịch của người dùng được đề xuất, cùng với tên và ảnh của người dùng đưa ra đề xuất. Nếu người giới thiệu cập nhật ảnh của họ, bất kỳ đề xuất nào họ

có văn bản cần phản ánh bức ảnh mới. Do đó, khuyến nghị nên có một tham chiếu đến hồ sơ của tác giả.



Hình 2-4 minh họa cách các tính năng mới này yêu cầu các mối quan hệ nhiều-nhiều. Dữ liệu trong mỗi hình chữ nhật chấm có thể được nhóm thành một tài liệu, nhưng các tham chiếu đến các tổ chức, trường học và người dùng khác cần được thể hiện dưới dạng tham chiếu và yêu cầu tham gia khi được truy vấn.

Là cơ sở dữ liệu tài liệu lặp lại lịch sử?

Trong khi nhiều mối quan hệ và nhiều mối quan hệ được sử dụng thường xuyên trong cơ sở dữ liệu quan hệ, cơ sở dữ liệu tài liệu và NoQuery đã mở lại cuộc tranh luận về cách tốt nhất để thể hiện các mối quan hệ đó trong cơ sở dữ liệu. Cuộc tranh luận này lâu đời hơn nhiều so với NoQuery, trên thực tế, nó quay trở lại các hệ thống cơ sở dữ liệu được vi tính hóa sớm nhất. Cơ sở dữ liệu phổ biến nhất để xử lý dữ liệu kinh doanh trong những năm 1970 là Hệ thống quản lý thông tin IBM (IMS), ban đầu được phát triển để lưu trữ trong chương trình không gian Apollo và được phát hành thương mại lần đầu tiên vào năm 1968 [13]. Nó vẫn đang được sử dụng và duy trì cho đến ngày nay, chạy trên OS / 390 trên máy tính lớn của IBM [14].

Thiết kế của IMS đã sử dụng một mô hình dữ liệu khá đơn giản gọi là mô hình phân cấp, có một số điểm tương đồng đáng chú ý với mô hình JSON được sử dụng bởi cơ sở dữ liệu tài liệu [2]. Nó đại diện cho tất cả dữ liệu dưới dạng một cây các bản ghi được lồng trong các bản ghi, giống như cấu trúc JSON của Hình 2-2.

Giống như cơ sở dữ liệu tài liệu, IMS hoạt động tốt cho các mối quan hệ một-nhiều, nhưng nó làm cho các mối quan hệ nhiều-nhiều trở nên khó khăn và nó đã không hỗ trợ. Các nhà phát triển phải quyết định có nên sao chép dữ liệu (không chuẩn hóa) hoặc giải quyết thủ công các tham chiếu từ bản ghi này sang bản ghi khác. Những vấn đề của thập niên 1960 và 70 của 70 rất giống với những vấn đề mà các nhà phát triển đang gặp phải với cơ sở dữ liệu tài liệu ngày nay [15].

Các giải pháp khác nhau đã được đề xuất để giải quyết các hạn chế của mô hình phân cấp. Hai mô hình nổi bật nhất là mô hình quan hệ (đã trở thành SQL và chiếm lĩnh thế giới) và mô hình mạng (ban đầu có số lượng người theo dõi lớn nhưng cuối cùng bị mờ dần). Cuộc tranh luận lớn của người Viking giữa hai trại này kéo dài trong suốt những năm 1970 [2].

Do vấn đề mà hai mô hình đang giải quyết vẫn còn rất phù hợp cho đến ngày hôm nay, nên nó đáng để xem xét lại cuộc tranh luận này trong ánh sáng ngày hôm nay.

Mô hình mạng

Mô hình mạng được chuẩn hóa bởi một ủy ban gọi là Hội nghị về Ngôn ngữ hệ thống dữ liệu (CODASYL) và được thực hiện bởi một số nhà cung cấp cơ sở dữ liệu khác nhau; nó còn được gọi là mô hình CODASYL [16].

Mô hình CODASYL là một khái quát của mô hình phân cấp. Trong cấu trúc cây của mô hình phân cấp, mỗi bản ghi có chính xác một cha mẹ; trong mô hình mạng, một bản ghi có thể có nhiều cha mẹ. Ví dụ: có thể có một bản ghi cho khu vực "Khu vực Greater Seattle" và mọi người dùng sống ở khu vực đó đều có thể được liên kết với khu vực đó. Điều này cho phép các mối quan hệ nhiều-một và nhiều-nhiều được mô hình hóa.

Các liên kết giữa các bản ghi trong mô hình mạng không phải là khóa ngoại, mà giống như các con trỏ trong ngôn ngữ lập trình (trong khi vẫn được lưu trữ trên đĩa). Cách duy nhất để truy cập một bản ghi là đi theo một đường dẫn từ một bản ghi gốc dọc theo các chuỗi liên kết này. Điều này được gọi là một đường dẫn truy cập. Trong trường hợp đơn giản nhất, một đường dẫn truy cập có thể giống như duyệt qua danh sách được liên kết: bắt đầu ở đầu danh sách và xem xét một bản ghi tại một thời điểm cho đến khi bạn tìm thấy bản ghi mình muốn. Nhưng trong một thế giới của nhiều mối quan hệ nhiều-nhiều, nhiều con đường khác nhau có thể dẫn đến

cùng một bản ghi và một lập trình viên làm việc với mô hình mạng phải theo dõi các đường dẫn truy cập khác nhau này trong đầu.

Một truy vấn trong CODASYL được thực hiện bằng cách di chuyển con trỏ qua cơ sở dữ liệu bằng cách lặp qua danh sách các bản ghi và theo các đường dẫn truy cập. Nếu một bản ghi có nhiều cha mẹ (nghĩa là, nhiều con trỏ đến từ các bản ghi khác), mã ứng dụng phải theo dõi tất cả các mối quan hệ khác nhau. Ngay cả các thành viên ủy ban CODASYL cũng thừa nhận rằng điều này giống như điều hướng xung quanh một không gian dữ liệu n chiều [17].

Mặc dù lựa chọn đường dẫn truy cập thủ công có thể sử dụng hiệu quả nhất các khả năng phần cứng rất hạn chế trong những năm 1970 (chẳng hạn như ổ đĩa băng, tìm kiếm rất chậm), vấn đề là họ đã tạo ra mã để truy vấn và cập nhật cơ sở dữ liệu phức tạp và không linh hoạt. Với cả mô hình phân cấp và mô hình mạng, nếu bạn không có đường dẫn đến dữ liệu bạn muốn, bạn sẽ ở trong một tình huống khó khăn. Bạn có thể thay đổi các đường dẫn truy cập, nhưng sau đó bạn phải trải qua rất nhiều mã truy vấn cơ sở dữ liệu viết tay và viết lại để xử lý các đường dẫn truy cập mới. Rất khó để thực hiện các thay đổi đối với mô hình dữ liệu ứng dụng.

Mô hình quan hệ

Ngược lại, những gì mô hình quan hệ đã làm là bố trí tất cả dữ liệu trong mở: một mối quan hệ (bảng) chỉ đơn giản là một tập hợp các bộ dữ liệu (các hàng) và đó là nó. Không có cấu trúc lồng nhau mê cung, không có đường dẫn truy cập phức tạp để theo nếu bạn muốn xem dữ liệu. Bạn có thể đọc bất kỳ hoặc tất cả các hàng trong bảng, chọn những hàng khớp với điều kiện tùy ý. Bạn có thể đọc một hàng cụ thể bằng cách chỉ định một số cột làm khóa và khớp với các cột đó. Bạn có thể chèn một hàng mới vào bất kỳ bảng nào mà không phải lo lắng về các mối quan hệ khóa ngoài đến và từ các bảng khác.

Trong cơ sở dữ liệu quan hệ, trình tối ưu hóa truy vấn sẽ tự động quyết định phần nào của truy vấn sẽ thực hiện theo thứ tự nào và chỉ mục nào sẽ sử dụng. Những lựa chọn đó thực sự là đường dẫn truy cập, một sự khác biệt lớn là chúng được thực hiện tự động bởi trình tối ưu hóa truy vấn chứ không phải bởi nhà phát triển ứng dụng, vì vậy chúng ta hiếm khi cần phải suy nghĩ về chúng. Nếu bạn muốn truy vấn dữ liệu của mình theo những cách mới, bạn chỉ cần khai báo một chỉ mục mới và các truy vấn sẽ tự động sử dụng bất kỳ chỉ mục nào là phù hợp nhất. Bạn không cần phải thay đổi các truy vấn của mình để tận dụng một chỉ mục mới. (Xem thêm Ngôn ngữ truy vấn của dữ liệu đối với dữ liệu trên trang 42.) Mô hình quan hệ do đó giúp dễ dàng thêm các tính năng mới vào các ứng dụng. Tối ưu hóa truy vấn cho cơ sở dữ liệu quan hệ là những con thú phức tạp và chúng đã tiêu tốn nhiều năm nỗ lực nghiên cứu và phát triển [18]. Nhưng một cái nhìn sâu sắc quan trọng của

mô hình quan hệ là thế này: bạn chỉ cần xây dựng trình tối ưu hóa truy vấn một lần và sau đó tất cả các ứng dụng sử dụng cơ sở dữ liệu có thể được hưởng lợi từ nó. Nếu bạn không có trình tối ưu hóa truy vấn, thì nó dễ dàng mã hóa các đường dẫn truy cập cho một truy vấn cụ thể hơn là viết một trình tối ưu hóa cho mục đích chung nhưng giải pháp cho mục đích chung sẽ chiến thắng trong thời gian dài.

So sánh với cơ sở dữ liệu tài liệu

Cơ sở dữ liệu tài liệu trở lại mô hình phân cấp theo một khía cạnh: lưu trữ các bản ghi lồng nhau (mối quan hệ một-nhiều, như vị trí, giáo dục và contact\_info trong Hình 2-1) trong hồ sơ gốc của chúng thay vì trong một bảng riêng biệt.

Tuy nhiên, khi nói đến việc đại diện cho mối quan hệ nhiều-một và nhiều-nhiều, cơ sở dữ liệu quan hệ và tài liệu không khác nhau về cơ bản: trong cả hai trường hợp, mục liên quan được tham chiếu bởi một định danh duy nhất, được gọi là khóa ngoại mô hình quan hệ và tham chiếu tài liệu trong mô hình tài liệu [9].

Mã định danh đó được giải quyết tại thời điểm đọc bằng cách sử dụng truy vấn nối hoặc theo dõi. Đến nay, cơ sở dữ liệu tài liệu đã không đi theo con đường của CODASYL.

Cơ sở dữ liệu tài liệu Versus ngày hôm nay

Có nhiều sự khác biệt cần xem xét khi so sánh cơ sở dữ liệu quan hệ với cơ sở dữ liệu tài liệu, bao gồm các thuộc tính chịu lỗi của chúng (xem Chương 5) và xử lý đồng thời (xem Chương 7). Trong chương này, chúng tôi sẽ chỉ tập trung vào sự khác biệt trong mô hình dữ liệu.

Các đối số chính có lợi cho mô hình dữ liệu tài liệu là tính linh hoạt của lược đồ, hiệu suất tốt hơn do tính cục bộ và đối với một số ứng dụng, nó gần với cấu trúc dữ liệu được sử dụng bởi ứng dụng. Bộ đếm mô hình quan hệ bằng cách cung cấp hỗ trợ tốt hơn cho các phép nối, và các mối quan hệ nhiều-một và nhiều-nhiều.

Mô hình dữ liệu nào dẫn đến một mã ứng dụng đơn giản hơn?

Nếu dữ liệu trong ứng dụng của bạn có cấu trúc giống như tài liệu (nghĩa là, một cây có nhiều mối quan hệ, trong đó thông thường toàn bộ cây được tải cùng một lúc), thì đó có lẽ là một ý tưởng tốt để sử dụng mô hình tài liệu. Kỹ thuật quan hệ chia nhỏ cấu trúc giống như tài liệu thành nhiều bảng (như vị trí, giáo dục và contact\_info trong Hình 2-1).

Mô hình tài liệu có các hạn chế: ví dụ: bạn không thể tham chiếu trực tiếp đến một mục được lồng trong tài liệu, nhưng thay vào đó, bạn cần nói điều gì đó giống như mục mục thứ hai trong danh sách các vị trí cho người dùng 251, (giống như một đường dẫn truy cập trong mô hình phân cấp). Tuy nhiên, miễn là tài liệu không được lồng quá sâu, đó thường không phải là vấn đề. Sự hỗ trợ kém cho các phép nối trong cơ sở dữ liệu tài liệu có thể hoặc không thể là một vấn đề, tùy thuộc vào ứng dụng. Ví dụ: các mối quan hệ nhiều-nhiều có thể không bao giờ cần thiết trong một ứng dụng phân tích sử dụng cơ sở dữ liệu tài liệu để ghi lại những sự kiện xảy ra vào thời điểm nào [19].

Tuy nhiên, nếu ứng dụng của bạn sử dụng nhiều mối quan hệ nhiều-nhiều, mô hình tài liệu sẽ trở nên ít hấp dẫn hơn. Nó có thể giảm nhu cầu tham gia bằng cách không chuẩn hóa, nhưng sau đó mã ứng dụng cần thực hiện thêm công việc để giữ cho dữ liệu không chuẩn hóa nhất quán. Các phép nối có thể được mô phỏng trong mã ứng dụng bằng cách thực hiện nhiều yêu cầu tới cơ sở dữ liệu, nhưng điều đó cũng chuyển sự phức tạp vào ứng dụng và thường chậm hơn so với phép nối được thực hiện bởi mã chuyên dụng bên trong cơ sở dữ liệu.

Trong những trường hợp như vậy, sử dụng mô hình tài liệu có thể dẫn đến mã ứng dụng phức tạp hơn đáng kể và hiệu năng kém hơn [15].

Nói chung, không thể nói mô hình dữ liệu nào dẫn đến mã ứng dụng đơn giản hơn; nó phụ thuộc vào các loại mối quan hệ tồn tại giữa các mục dữ liệu. Đối với dữ liệu được kết nối với nhau, mô hình tài liệu rất khó xử, mô hình quan hệ có thể chấp nhận được và các mô hình đồ thị (xem Mô hình dữ liệu giống như đồ thị trên trang 49) là tự nhiên nhất.

Lược đồ linh hoạt trong mô hình tài liệu

Hầu hết các cơ sở dữ liệu tài liệu và hỗ trợ JSON trong cơ sở dữ liệu quan hệ không thực thi bất kỳ lược đồ nào trên dữ liệu trong tài liệu. Hỗ trợ XML trong cơ sở dữ liệu quan hệ thường đi kèm với xác thực lược đồ tùy chọn. Không có lược đồ nào có nghĩa là các khóa và giá trị tùy ý có thể được thêm vào tài liệu và khi đọc, khách hàng không có gì đảm bảo về các trường mà tài liệu có thể chứa.

Cơ sở dữ liệu tài liệu đôi khi được gọi là schemaless, nhưng điều đó gây hiểu lầm, vì mã đọc dữ liệu thường giả định một loại cấu trúc nào đó, tức là có một lược đồ ngầm, nhưng nó không được cơ sở dữ liệu thực thi [20]. Một thuật ngữ chính xác hơn là lược đồ khi đọc (cấu trúc của dữ liệu là ẩn và chỉ được giải thích khi dữ liệu được đọc), ngược lại với lược đồ ghi (cách tiếp cận truyền thống của cơ sở dữ liệu quan hệ, trong đó lược đồ là rõ ràng và cơ sở dữ liệu đảm bảo tất cả dữ liệu bằng văn bản phù hợp với nó) [21].

Schema-on-read tương tự như kiểm tra kiểu động (thời gian chạy) trong các ngôn ngữ lập trình, trong khi lược đồ ghi lại tương tự như kiểm tra kiểu tĩnh (thời gian biên dịch). Giống như những người ủng hộ việc kiểm tra kiểu tĩnh và động có những cuộc tranh luận lớn về giá trị tương đối của chúng [22], việc thực thi các lược đồ trong cơ sở dữ liệu là một chủ đề gây tranh cãi và nói chung, không có câu trả lời đúng hay sai. Sự khác biệt giữa các cách tiếp cận đặc biệt đáng chú ý trong các tình huống trong đó một ứng dụng muốn thay đổi định dạng dữ liệu của nó. Ví dụ: giả sử bạn hiện đang lưu trữ mỗi tên người dùng đầy đủ trong một trường và thay vào đó bạn muốn lưu trữ tên và họ riêng biệt [23]. Trong cơ sở dữ liệu tài liệu, bạn sẽ chỉ bắt đầu viết tài liệu mới với các trường mới và có mã trong ứng dụng xử lý trường hợp khi đọc tài liệu cũ. Ví dụ:

Thay đổi lược đồ có tiếng xấu là chậm và yêu cầu thời gian chết. Danh tiếng này không hoàn toàn xứng đáng: hầu hết các hệ thống cơ sở dữ liệu quan hệ thực thi câu lệnh ALTER TABLE trong vài mili giây. MySQL là một ngoại lệ đáng chú ý, nó sao chép toàn bộ bảng trên ALTER TABLE, có thể có nghĩa là vài phút hoặc thậm chí hàng giờ ngừng hoạt động khi thay đổi một bảng lớn mặc dù các công cụ khác nhau tồn tại để giải quyết giới hạn này [24, 25, 26].

Chạy câu lệnh CẬP NHẬT trên một bảng lớn có thể bị chậm trên bất kỳ cơ sở dữ liệu nào vì mỗi hàng cần phải được viết lại. Nếu điều đó không được chấp nhận, ứng dụng có thể để First\_name được đặt thành mặc định của NULL và điền vào lúc đọc, giống như với cơ sở dữ liệu tài liệu.

Cách tiếp cận lược đồ đọc là thuận lợi nếu các mục trong bộ sưu tập không có cùng cấu trúc vì một lý do nào đó (ví dụ: dữ liệu không đồng nhất) Ví dụ: vì:

• Có nhiều loại đối tượng khác nhau và việc đặt từng loại đối tượng vào bảng riêng của nó là không thực tế.

Có nhiều loại đối tượng khác nhau và việc đặt từng loại đối tượng vào bảng riêng của nó là không thực tế.

Cấu trúc của dữ liệu được xác định bởi các hệ thống bên ngoài mà bạn không có quyền kiểm soát và có thể thay đổi bất cứ lúc nào.

Trong những tình huống như thế này, một lược đồ có thể gây tổn hại nhiều hơn nó và các tài liệu schemaless có thể là một mô hình dữ liệu tự nhiên hơn nhiều. Nhưng trong trường hợp tất cả các bản ghi được dự kiến ​​có cùng cấu trúc, các lược đồ là một cơ chế hữu ích để ghi lại và thực thi cấu trúc đó. Chúng ta sẽ thảo luận về lược đồ và sự phát triển lược đồ chi tiết hơn trong Chương 4

Địa phương dữ liệu cho các truy vấn

Một tài liệu thường được lưu trữ dưới dạng một chuỗi liên tục duy nhất, được mã hóa dưới dạng JSON, XML hoặc một biến thể nhị phân của nó (như MongoDB Bash BSON). Nếu ứng dụng của bạn thường cần truy cập vào toàn bộ tài liệu (ví dụ: để hiển thị nó trên trang web), có một lợi thế về hiệu suất đối với địa phương lưu trữ này. Nếu dữ liệu được chia thành nhiều bảng, như trong Hình 2-1, cần phải tra cứu nhiều chỉ mục để truy xuất tất cả, điều này có thể cần nhiều lần tìm kiếm đĩa hơn và mất nhiều thời gian hơn.

Lợi thế địa phương chỉ áp dụng nếu bạn cần các phần lớn của tài liệu cùng một lúc. Cơ sở dữ liệu thường cần tải toàn bộ tài liệu, ngay cả khi bạn chỉ truy cập một phần nhỏ của tài liệu, điều này có thể gây lãng phí cho các tài liệu lớn. Khi cập nhật tài liệu, toàn bộ tài liệu thường cần được viết lại chỉ sửa đổi mà không thay đổi kích thước được mã hóa của tài liệu có thể dễ dàng được thay đổi

hình thành tại chỗ [19]. Vì những lý do này, thông thường bạn nên giữ tài liệu khá nhỏ và tránh viết làm tăng kích thước của tài liệu [9].

Những hạn chế về hiệu suất này làm giảm đáng kể tập hợp các tình huống trong đó cơ sở dữ liệu tài liệu là hữu ích.

Nó có giá trị chỉ ra rằng ý tưởng nhóm các dữ liệu liên quan với nhau cho địa phương không bị giới hạn trong mô hình tài liệu. Ví dụ: cơ sở dữ liệu Google West Spanner cung cấp các thuộc tính cục bộ tương tự trong mô hình dữ liệu quan hệ, bằng cách cho phép lược đồ khai báo rằng các hàng bảng bảng phải được xen kẽ (lồng nhau) trong bảng cha [27].

Oracle cho phép tương tự, sử dụng một tính năng gọi là bảng cụm chỉ mục nhiều bảng [28].

Khái niệm họ cột trong mô hình dữ liệu Bigtable (được sử dụng trong Cassandra và HBase) có mục đích tương tự là quản lý địa phương [29].

Chúng ta cũng sẽ thấy nhiều hơn về địa phương trong Chương 3

Sự hội tụ của tài liệu và cơ sở dữ liệu quan hệ

Hầu hết các hệ thống cơ sở dữ liệu quan hệ (trừ MySQL) đã hỗ trợ XML từ giữa những năm 2000. Điều này bao gồm các hàm để thực hiện sửa đổi cục bộ đối với tài liệu XML và khả năng lập chỉ mục và truy vấn bên trong tài liệu XML, cho phép các ứng dụng sử dụng các mô hình dữ liệu rất giống với những gì chúng sẽ làm khi sử dụng cơ sở dữ liệu tài liệu.

PostgreSQL kể từ phiên bản 9.3 [8], MySQL kể từ phiên bản 5.7 và IBM DB2 kể từ phiên bản 10.5 [30] cũng có mức hỗ trợ tương tự cho các tài liệu JSON. Với sự phổ biến của JSON cho API web, có thể các cơ sở dữ liệu quan hệ khác sẽ đi theo bước chân của họ và thêm hỗ trợ JSON.

Về phía cơ sở dữ liệu tài liệu, RethinkDB hỗ trợ các phép nối giống như quan hệ trong ngôn ngữ truy vấn của nó và một số trình điều khiển MongoDB tự động giải quyết các tham chiếu cơ sở dữ liệu (thực hiện hiệu quả một phép nối phía máy khách, mặc dù điều này có thể chậm hơn so với phép nối được thực hiện trong cơ sở dữ liệu vì nó yêu cầu các chuyến đi khứ hồi mạng bổ sung và ít được tối ưu hóa hơn).

Dường như cơ sở dữ liệu quan hệ và tài liệu đang trở nên giống nhau hơn theo thời gian và đó là một điều tốt: các mô hình dữ liệu bổ sung cho nhau.v Nếu cơ sở dữ liệu có thể xử lý dữ liệu giống như tài liệu và cũng thực hiện các truy vấn quan hệ trên đó, các ứng dụng có thể sử dụng kết hợp các tính năng phù hợp nhất với nhu cầu của họ.

Sự kết hợp giữa các mô hình quan hệ và tài liệu là một lộ trình tốt cho cơ sở dữ liệu trong tương lai.

MapReduce là một mô hình lập trình để xử lý số lượng lớn dữ liệu hàng loạt trên nhiều máy, được phổ biến bởi Google [33]. Một dạng MapReduce giới hạn được hỗ trợ bởi một số kho lưu trữ dữ liệu NoQuery, bao gồm MongoDB và CouchDB, như một cơ chế để thực hiện các truy vấn chỉ đọc trên nhiều tài liệu. MapReduce, nói chung, được mô tả chi tiết hơn trong Chương 10. Hiện tại, chúng tôi sẽ chỉ thảo luận ngắn gọn về việc sử dụng mô hình MongoDB.

MapReduce không phải là ngôn ngữ truy vấn khai báo cũng không phải là API truy vấn hoàn toàn bắt buộc, nhưng ở đâu đó ở giữa: logic của truy vấn được thể hiện bằng các đoạn mã, được gọi bởi khung xử lý. Nó dựa trên bản đồ (cũng

được gọi là hàm coll) và hàm rút gọn (còn được gọi là hàm gấp hoặc hàm) tồn tại trong nhiều ngôn ngữ lập trình hàm.

Để đưa ra một ví dụ, hãy tưởng tượng bạn là một nhà sinh vật học biển và bạn thêm một hồ sơ quan sát vào cơ sở dữ liệu của bạn mỗi khi bạn nhìn thấy động vật trong đại dương. Bây giờ bạn muốn tạo một báo cáo cho biết bạn đã nhìn thấy bao nhiêu con cá mập mỗi tháng.

Google là một công ty dẫn đầu về phần mềm xử lý Big Data. Hầu hết các phần mềm xử lý dữ liệu như Hadoop đều có nguồn gốc ý tưởng từ Google. Tuy vậy Google hầu như không opensource các phần mềm do họ viết ra mà chỉ viết bài báo công bố về sản phẩm của minh sau một thời gian nhất định. Đến khi cộng đồng mã nguồn mở clone lại thành công, thì Google thường tuyên bố rằng [đó là công nghệ đã cũ và họ đã không còn dùng vài năm rồi](http://www.datacenterknowledge.com/archives/2014/06/25/google-dumps-mapreduce-favor-new-hyper-scale-analytics-system/) :D.

Tuy rằng Google không còn sử dụng nhưng không có nghĩa là mọi người sẽ không sử dụng nữa vì đơn giản các công ty hoàn toàn có thể hưởng lợi từ các sản phẩm clone này (do không có nhiều dữ liệu như Google).

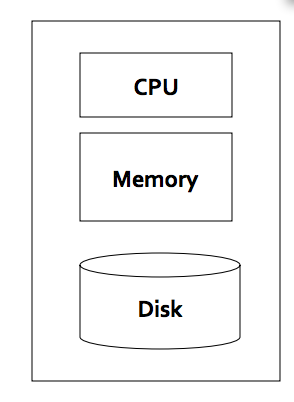
**Giới thiệu**

Gần đây thấy bản thân chém gió rất nhiều về MapReduce, Hadoop v.v nhưng chưa thấy có bài viết nào tổng hợp + giải thích cụ thể về MapReduce vì vậy mình quyết định viết một bài **Chém gió** về Map-Reduce. Bài viết này sẽ giới thiệu 3 vấn đề sau:

* Distributed File-System - DFS (Hệ thống file phân tán)
* Mô hình tính toán MapReduce
* Lập lịch và luồng dữ liệu

**Vấn đề với năng lực xử lý của một máy tính**

Máy tính mà chúng ta vẫn sử dụng để xử lý dữ liệu hàng ngày có thể được mô tả một cách trừu tượng hoá nhất bằng hình vẽ dưới đây.

[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/single%20machine.png_gji8d6mtu9)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/single%20machine.png_gji8d6mtu9)

Theo như hình vẽ chúng ta có: CPU, Memory và đĩa cứng. Theo công nghệ hiện tại mình biết thì mỗi module có các thông số tối đa như sau (dòng máy chủ phổ thông):

* CPU: với máy tính thông thường thì trên 1 board chỉ có 1 đến 2 CPUs. Mỗi CPU có thể có tối đa 18 cores (theo [công nghệ hiện tại mình biết](http://ark.intel.com/products/84689))
* Memory: trên 1 mainboard có thể cắm tối đa 16 thanh RAM. Mỗi thanh có dung lượng tối từ 8G. Một máy tính có thể có khoảng 8\* 16 = 128GB RAM)
* Đĩa cứng: dung lượng tối đa của HDD theo [công nghệ hiện tại mình biết là 8TB](https://en.wikipedia.org/wiki/Hard_disk_drive)

Với máy tính với cấu hình dù rất khoẻ như ở trên thì ta vẫn thấy giới hạn xử lý dữ liệu của nó. Cụ thể máy tính không thể lưu trữ được quá 8TB dữ liệu, không thể đồng thời xử lý được dữ liệu lớn hơn 128GB (kích thước của RAM) và không thể đồng thời xử lý được lớn hơn 18 luồng chương trình.

Trong khi đấy thực tế lại dữ liệu lại đến với số lượng nhanh và nhiều hơn dung lượng có thể xử lý bởi một máy tính rất nhiều. Ví dụ điển hình nhất là Google (dù rằng sẽ có người tranh luận rằng chả ai có dữ liệu nhiều như Google). Theo Google họ [có rất nhiều dữ liệu](http://static.googleusercontent.com/media/research.google.com/en/people/jeff/Stanford-DL-Nov-2010.pdf):

* **10 tỉ** trang web
* Trung bình kích thước 1 trang web: 20KB → 10 tỉ \* 20KB == 200TB
* Tốc độ đọc của đĩa cứng là **50MB/giây** → thời gian để đọc 200TB là 4 triệu giây ~ **46+ ngày**.

Do vậy chưa tính đến việc xử lý dữ liệu, chỉ riêng việc đọc dữ liệu đã vượt qua năng lực xử lý của 1 máy tính. Điều này đòi hỏi một mô hình truy vấn dữ liệu mới cho phép xử lý lượng dữ liệu trên. Mô hình tính toán Cluster và Distributed File System ra đời do đòi hỏi thực tế này.

**Giải quyết bài toán một máy tính thế nào?**

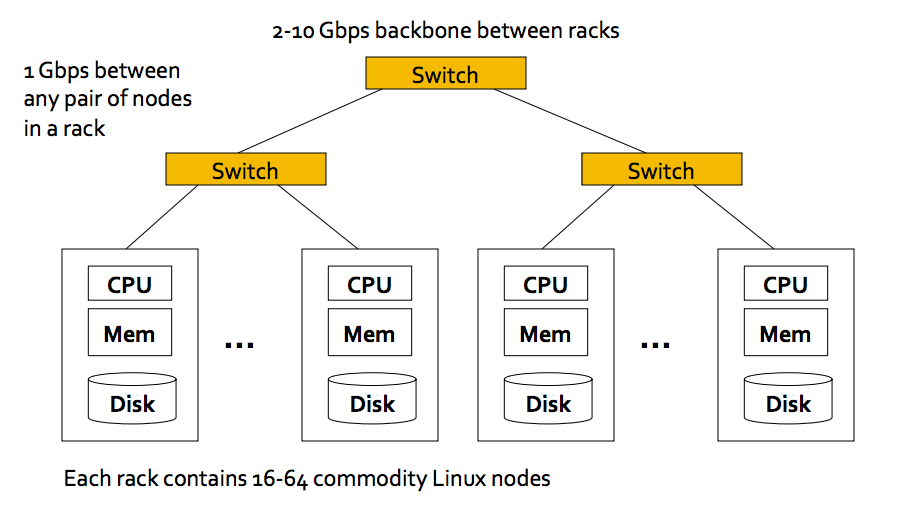
Như đã tính toán thử ở trên, một máy tính không thể nào xử lý được số lượng dữ liệu lớn như vậy. Điều này dẫn đến 2 đòi hỏi:

* **Đấu nối nhiều máy tính** để cùng xử lý
* Tìm ra **cách để cho các máy tính cùng xử lý**

**Đấu nối (GFS)**

Theo như [bài viết trên trang DataCenterKnowledge](http://www.datacenterknowledge.com/archives/2011/08/01/report-google-uses-about-900000-servers/) thì năm 2011 Google có hơn 1 triệu máy tính! Vậy họ giải quyết vấn đề đấu nối máy tính kiểu gì?

Dưới đây là một phương pháp:

[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/Cluster.png_p79wu8iti6)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/Cluster.png_p79wu8iti6)

* Mỗi rack trong Data Center có một cơ số máy tính (từ 16 → 64 máy tính với cấu hình phổ thông)
* Các máy tính được kết nối với nhau bằng 1Gbps Switch
* Các Rack được kết nối với nhau bằng Switch 2-10Gbps

Nhờ kiến trúc này ta có thể đấu nối được nhiều máy chủ với nhau. Bài toán đấu nối được giải quyết. Tuy vậy, số lượng máy chủ nhiều lên kéo theo rất nhiều vấn đề phát sinh như:

* Hỏng hóc
* Dữ liệu không còn thống nhất sau khi máy chủ hỏng
* Chương trình sẽ ra sao khi máy chủ hỏng hóc.
* Tắc nghẽn mạng: số lượng máy chủ nhiều → các máy tính nói chuyện với nhau nhiều → tắc nghẽn mạng.

→ **Tính toán với nhiều máy tính không hề dễ**

**Cách để cho các máy tính cùng xử lý (Map-Reduce)**

Khi đã tìm được cách đấu nói, vấn đề tiếp theo là lập ra được mô hình xử lý dữ liệu trên cluster hiện tại. Map-Reduce là một giải pháp! [Map-Reduce](http://research.google.com/archive/mapreduce.html) được phát minh bởi các kỹ sư Google để giải quyết bài toán của họ từ hơn 10 năm trước! Map-Reduce giải quyết các vấn đề liệt kê ở trên bằng cách:

* Phân chia dữ liệu thành nhiều block và chia cho nhiều máy tính lưu trữ (đảm bảo tính toàn vẹn và tính sẵn sàng của dữ liệu).
* Chuyển tính toán về nơi có dữ liệu. Ý tưởng chuyển tính toán đến nơi có dữ liệu thực sự là một sự đột phá và đã được đề xuất trong [một bài báo](http://research.microsoft.com/pubs/70001/tr-2003-24.pdf) của nhà khoa học máy tính [Jim Gray](https://en.wikipedia.org/wiki/Jim_Gray_(computer_scientist)) từ năm 2003.
* Đưa ra mô hình và giao diện tính toán đơn giản.

**Chia dữ liệu thành nhiều Block**

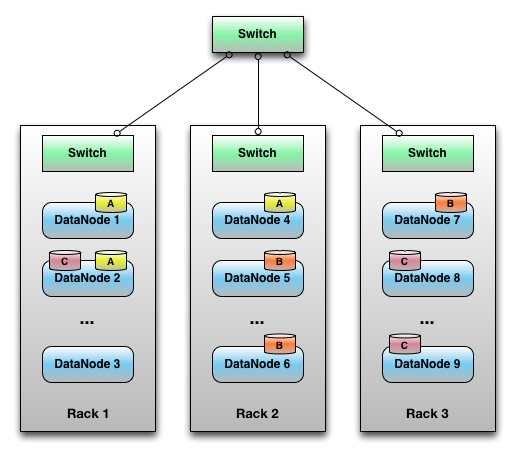
Các kỹ sư Google giải quyết bài toán này bằng [giải pháp GFS](https://en.wikipedia.org/wiki/Google_File_System). GFS là một hệ thống quản lý File phân tán với các chức năng giống như hệ thống File bình thường của Linux như: không gian tên (namespace), tính thừa thãi (redundancy), và tính sẵn sàng (availability). Như đã giới thiệu ở bài viết [Các giải pháp BigData](http://kipalog.com/posts/Cac-giai-phap-BigData--so-sanh-tuong-quan-voi-Google), HDFS cũng là một một hệ thống quản lý File phân tán.

Đặc điểm của hệ thống File phân tán này là:

* Dùng để quản lý các File có kích thước lớn: kích thước từ trăm GB đến TB
* Dữ liệu ít khi bị cập nhật ở giữa file (kiểu mở file, đến dữa file, cập nhật) mà thường được đọc học ghi vào cuối File (append).

GFS được thiết kế bao gốm 2 module: **Chunk Server** và **Master Server**.

**Chunk Server** làm nhiệu vụ lưu trữ các chunk (hay block) dữ liệu. Mỗi chunk có thể của 1 File hay của file khác. Các Chuck dữ liệu được sao chép giữa các máy chủ khác nhau.

[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/chunk.png_rf04u0vo5f)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/chunk.png_rf04u0vo5f)

Chunk Server có thể được mình hoạ bởi hình ở trên. Ví dụ ta có file A gồm 3 Chunks, File B gồm 3 Chunk và File C gồm 3 Chunk. Các Chunk sẽ được lưu trữ ở các Chunk Server (DataNode) khác nhau nằm ở các Rack khác nhau.

Mỗi Chunk Server vừa làm nhiệm vụ lưu trữ dữ liệu vừa thực hiện tính toán. Thông thường 1 File sẽ được chia làm các chunks có kích thường từ (64MB đến 128MB) và thường mỗi chunk sẽ được chép 3 bản và lưu ở các máy chủ khác nhau.

**Master Server** (hay NameNode trong HDFS) làm nhiệm vụ quản lý metadata cho hệ thống File. Cụ thể Master Server sẽ lưu giữ thông tin như File này có bao nhiêu Chunk và mỗi Chunk sẽ được lưu ở máy chủ nào. Dựa vào thông tin metadata này mà thuật toán tính toán sẽ chuyển tính toán về máy chủ có chunk (sẽ trình bày kỹ hơn)

**Client** truy cập 1 file trên GFS bằng quy trình sau:

* Hỏi master server thông tin File (số lượng Chunk, "vị trí" máy chủ lưu trữ)
* Hỏi máy chủ lưu trữ thông tin của File.

**Mô hình tính toán (MapReduce)**

Bài toán word-count (đếm từ) là bài toán dễ hiểu nhất minh hoạ cho MapReduce (MR). Bài toán có những đặc điểm sau:

* File cần đếm rất lớn (quá lớn để có thể được tải lên bộ nhớ chính của 1 máy)
* Mỗi cặp <từ ngữ, số lượng> quá lớn cho bộ nhớ.

MR chia làm 3 thao tác: Map và Reduce

**Map**: quét file đầu vào và ghi lại từng bản ghi

**Group by Key**: sắp xếp và trộn dữ liệu cho mỗi bản ghi sinh ra từ Map

**Reduce**: tổng hợp, thay đổi hay lọc dữ liệu từ thao tác trước và ghi kết quả ra File.

Hình minh hoạ sau làm rõ các bước và nội dung thực hiện ở mỗi bước.

[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/MapReduce_Work_Structure.png_2b168e8bkx)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/MapReduce_Work_Structure.png_2b168e8bkx)

Về mặt định nghĩa thuật toán, ta có thể mô tả MR như sau:

* Input: dữ liệu dưới dạng Key → Value
* Lập trình viên viết 2 thủ tục:
  + Map(k, v) → <k', v'>\*
  + Reduce(k', <v'>\*) → <k', v''>\*

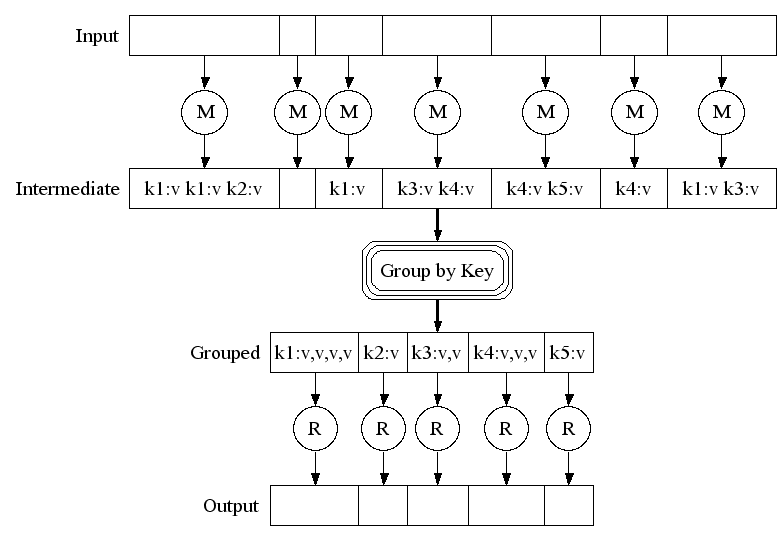
**Map** biến mỗi key k thu được bằng thành cặp <k', v'>. **Reduce** nhận đầu vào là khoá k' và danh sách cách giá trị v' và trả về kết quả là cặp <k', v''>.

Ví dụ với hình mô tả ở trên thì Map trả về danh sách: <Bear, 1>, <Bear, 1> còn Reduce nhận kết quả trên và trả về <Bear, 2>.

Mô hình này nhìn có vẻ đơn giản nhưng thật sự rất mạnh và có thể giải quyết được rất nhiều bài toán (có dịp mình sẽ cố viết)

**Lập lịch và dòng dữ liệu**

Sau khi đã có cách đấu nối và phương pháp tính toán, vấn đề tiếp theo cần bàn là tính thế nào, khi nào và ra sao. Map-Reduce có một đặc điểm thú vị là chỉ cần phân chia các File thành các vùng độc lập thì các thủ tục Map không hoàn toàn liên quan đến nhau có thể thực hiện song song.

[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/google_schema.gif_o7lnpux3be)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/google_schema.gif_o7lnpux3be)

Một File Input có thể được xử lý bởi nhiều Map/Reduce

Như đã viết ở trên Map-Reduce sẽ cố gắng cung cấp giao diện lập trình đơn giản trong khi che dấu những xử lý phức tạp đi. Các xử lý chi tiết phức tạp bao gồm:

* Phân chia dữ liệu
* lập lịch chạy các thủ tục Map/Reduce trên các máy tính
* Thực hiện thủ tục Groupby
* Quản lý hỏng hóc (ví dụ tự động khởi động các thủ tục M/R đang chạy dở thì máy hỏng, quản lý dữ liệu khi máy hỏng)
* Quản lý giao tiếp giữa các máy tính.

Vậy dữ liệu sẽ được lưu truyển thế nào trong 1 quá trình M/R. Về cơ bản Framework lập trình M/R sẽ cố gắng lập lịch cho các thủ tục M/R diễn ra ở các Chunk server có dữ liệu mà thủ tục cần (Chuyển tính toán đến nơi có dữ liệu). Các dữ liệu trung gian trong quá trình M/R sẽ được lưu ở **Local FS** của các máy chủ đang chạy M/R. Dữ liệu Output từ các M/R sẽ là output cho các M/R khác tổng hợp dữ liệu.

Trong quá trình chạy, một máy tính gọi là **Master node** phải chịu trách nhiệm quản lý tiến trình cũng như trạng thái các M/R. Khi Map kết thúc, thủ tục này sẽ thông báo trạng thái cũng như đường dẫn file kết quả cho Master để Master biết khởi động Reduce. Node master cũng làm nhiệm vụ **ping** các máy chủ workers định kỳ để phát hiện hỏng hóc có thể có.

Khi xảy ra lỗi, master sẽ khởi động và tái lập lịch lại các Map/Reduce. Khi Master bị lỗi, toàn bộ hệ thống sẽ lỗi.

**Kết luận**

Bài viết trình bày 3 vấn đề cơ bản về Map-Reduce

* Distributed File-System - DFS (Hệ thống file phân tán)
* Mô hình tính toán MapReduce
* Lập lịch và luồng dữ liệu