Index là gì:

* Raven không cho phép truy vấn mà không sử dụng index. Do đó, tất cả truy vấn chúng ta thực hiện đều phải dùng index. Điều này nghe có vẻ lạ? Đoạn code bên dưới chạy được:

|  |
| --- |
| **//listing 5.1** - querying RavenDB  var ayendeBlog = session.Query<Blog>()  .Where(blog => blog.Title == "Ayende @ Rahien")  .First(); |

* Cách thức làm việc như thế nào. Khi chúng ta tạo câu truy vấn tới RavenDB, bộ tối ưu hóa truy vấn của RavenDB sẽ tìm đến những index thích hợp để truy vấn dữ liệu. Nhưng chuyện gì sẽ xảy ra nếu không có index khớp với yêu cầu truy vấn. RavenDB sẽ tạo ra một index tạm thời dùng cho truy vấn này. Chúng ta vẫn chưa có khái niệm chính xác về index. RavenDB tạo ra index trên máy chủ như sau:

|  |
| --- |
| **//listing 5.2** - the auto generated index created by RavenDB  from blogItem in docs.Blogs  select new { blogItem.Title } |

* Đoạn code trên trông giống truy vấn Linq, và không giống như bất cứ kiểu index nào trước đây chúng ta đã từng thấy, vậy chuyện gì đang diễn ra. Câu trả lời là đoạn code trên là định nghĩa hàm index (index definition function), cái mà RavenDB sử dụng để lấy thông tin được đánh dấu từ những document. Giả sử trên server có chưa document sau:

|  |
| --- |
| **// listing 5.3** - sample documents  { // blogs/1234  "Title": "Ayende @ Rahien",  "Author": "...",  "StartedAt": "..."  }  { // blogs/1235  "Title": "Raven’s Flight",  "Author": "...",  "StartedAt": "..."  } |

* Kết quả xuất ra khi sử dụng hàm index trong listing 5.2 vào những document trong listing 5.3 được như bên dưới:

|  |
| --- |
| **// listing 5.4** - the ouput of the indexing function over the sample documents  { "Title": "Ayende @ Rahien", "\_\_document\_id": "blogs/1234" }  { "Title": "Raven’s Flight", "\_\_document\_id": "blogs/1235" } |

* Những kết quả này sẽ được lưu trữ trong những persisten index cho phép chúng ta thực hiện những truy vấn tốn ít chi phí dựa trên những giá trị được lưu trữ trong index.
* Lưu ý: “\_\_document\_id” từ đâu mà có, chúng không xuất hiện trong hàm định nghĩa index.
* Giá trị này được RavenDB thêm vào trong những kết quả của hàm định nghĩa index, nó là một trong những giá trị được tự động thêm vào bởi RavenDB (một giá trị khác nữa là \_\_reduce\_key nhưng giá trị này có trong map/reduce index).
* Sau khi RavenDB chắc chắn rằng index đã tồn tại, nó có thể truy vấn index. Phần trước chúng ta đã nói về cách thức RavenDB xây dựng indexes bên dưới nền và khái niệm staleness. Bởi vì RavenDB không phải đợi cho quá trình indexing hoàn thành, nó có thể đưa ra những kết quả mà không cần phải đợi ngay cả khi có nhiều tiến trình indexing chạy đồng thời.
* Tất cả những điều đó cho chúng ta biết được vì sao RavenDB lại truy vấn nhanh đến như vậy. Tất cả truy vấn đều được chạy trên những kết quả tính toán trước trong indexes, và những truy vấn này không phải đợi.
* Định dạng lưu trữ index dưới dạng Lucene index (phần sau sẽ nói rõ)

----------------------------

**Map/reduce index**

* Map/reduce đơn giản là cách nói khác của group by. Trước đây chúng ta đã quen với khái niệm group by trong cơ sở dữ liệu quan hệ. Với group by thì chúng ta đã quen thuộc nhưng nghe đến map/reduce thì giống như một cái gì đó mới lạ và khó sử dụng.
* Minh họa bằng một ví dụ cho dễ hiểu, chúng ta sẽ xem việc đếm số lượng bình luận (comments) trong mỗi blog là một ví dụ. Bên dưới là một blog post document:

|  |
| --- |
| **// Listing 5.**1 - A sample blog post  { // Document id: posts/1923  "Name": "Raven’s Map/Reduce functionality",  "BlogId": "blogs/1234",  "Comments": [  {  "Author": "Martin",  "Text": "..."  }  ]  } |

* Để trả lời cho câu hỏi có bao nhiêu comments trong mỗi bài viết (post), chúng ta phải tập hợp dữ liệu từ nhiều documents. Sử dụng Linq, chúng ta làm việc đó rất dễ dàng, như đoạn code bên dưới thực hiện:

|  |
| --- |
| **// Listing 5.2** - A Linq query to aggregate all the comment count per blog  from post in docs.Posts  group post by post.BlogId into g  select new  {  BlogId = post.BlogId,  CommentCount = g.Sum(x=>x.Comments.Length)  }; |

* Có thể chúng ta đã thấy đoạn code trên vài lần trước đây, đoạn code ngắn hầu như không có vấn đề gì, nó giả sử rằng nó có thể truy cập được hết tất cả dữ liệu. Nhưng chuyện gì sẽ xảy ra nếu dữ liệu quá lớn không vừa với bộ nhớ, hoặc là dữ liệu quá lớn để lưu trữ trên một máy tính đơn.
* Đây cũng chính là lúc map/reduce xuất hiện. Map/reduce chỉ là phương tiện để làm việc với group by trên quy mô lớn có tính mở rộng cao. Nhưng khái niệm thì vẫn như cũ. Thật ra nó chỉ là chúng ta cần phải chia group by ra thành nhiều bước nhỏ hơn có thể chạy trên những máy khác nhau.

**Quá trình thực hiện map/reduce**

* Chúng ta sẽ chia thao tác trong listing 5.3 thành nhiều thao tác riêng biệt. Bắt đầu với việc nhóm tất cả bài viết theo BlogId và chúng ta chọn 2 thuộc tính BlogId và Comment.Length. Với việc làm này chúng ta chỉ chọn ra những thông tin nào thực sự cần từ bài viết, đó là 2 thuộc tính BlogId và Comment.Length. Vì vậy chúng ta định nghĩa truy vấn Linq làm việc này. Code bên dưới như sau:

|  |
| --- |
| **// Listing 5.3** - Projecting just the required fields from the posts  from post in docs.Posts  select new  {  post.BlogId,  CommentCount = Comments.Length  } |

* Hữu ích của việc này là, thay vì phải làm việc với toàn bộ nội dung post documents, chúng ta chỉ thao tác với một một số ít các thuộc tính dữ liệu từ đối tượng post document ban đầu. Chúng ta đã giảm thiểu số lượng dữ liệu mà chúng ta cần làm việc và chúng ta có được một tập hợp document thông qua truy vấn Linq trong Listing 5.3, chúng ta sẽ có được kết quả như bên dưới:

|  |
| --- |
| **// Listing 5.4** - The results of the query in liting 5.4  { BlogId: "blogs/1234", CommentCount: 4 }  { BlogId: "blogs/9313", CommentCount: 2 }  { BlogId: "blogs/1234", CommentCount: 3 }  { BlogId: "blogs/2394", CommentCount: 1 }  { BlogId: "blogs/9313", CommentCount: 0 } |

* Như ta thấy thì có sự khác biệt khá lớn giữa kích cỡ dữ liệu của kết quả truy vấn và dữ liệu ban đầu. Và bây giờ chúng ta sẽ làm tiếp bước thứ 2 trong quá trình này là nhóm kết quả và thực hiện kết hợp thực sự. Đoạn code sau làm điều đó:

|  |
| --- |
| **// Listing 5.5** - Grouping the results to find the final result  from result in results  group result by result.BlogId into g  select new  {  BlogId = g.Key,  CommentCount = g.Sum(x=>x.CommentCount)  } |

* Truy vấn này trên dữ liệu Listing 5.4 có vẻ hợp lý, nhìn tương tự như kết quả trong Listing 5.2:

|  |
| --- |
| **// listing 5.6:**  { BlogId: "blogs/1234", CommentCount: 7 }  { BlogId: "blogs/9313", CommentCount: 2 }  { BlogId: "blogs/2394", CommentCount: 1 } |

* Nãy giờ thì chúng ta thực hiện không có gì thực sự đặc biệt nhưng chúng ta đã thực hiện một việc mà việc này có thể làm cho bạn ngạc nhiên. Chúng ta đã định nghĩa một cặp hàm map/reduce:
  + Listing 5.3 là hàm map
  + Listing 5.5 là hàm reduce
* Chúng ta không làm phức tạp truy vấn trong Listing 5.2 bằng cách tách ra làm 2 truy vấn riêng biệt mà không có lý do. Giả sử chúng ta có một tập hợp dữ liệu trên một máy khác. Dữ liệu này được thể hiện bên dưới:

|  |
| --- |
| **// listing 5.7:**  { BlogId: "blogs/1234", CommentCount: 5 }  { BlogId: "blogs/7269", CommentCount: 2 }  { BlogId: "blogs/1234", CommentCount: 4 }  { BlogId: "blogs/9313", CommentCount: 2 } |

* Chúng ta muốn lấy **tất cả** kết quả của blogs, không chỉ những bài viết trên một máy riêng biệt (truy vấn ở listing 5.2 chỉ làm việc tốt trên một máy tính riêng biệt). Những gì chúng làm tiếp theo là chạy tất cả dữ liệu trong listing 5.7 thông qua truy vấn 5.3 và cho chúng ta dữ liệu như sau:

|  |
| --- |
| **// listing 5.8:**  { BlogId: "blogs/1234", CommentCount: 9 }  { BlogId: "blogs/7269", CommentCount: 2 }  { BlogId: "blogs/9313", CommentCount: 2 } |

* Phần thú vị bây giờ mới bắt đầu, bởi vì hàm reduce chạy đệ quy nên nhiệm vụ của chúng ta là thực thi truy vấn ở listing 5.5 trên cả 2 dữ liệu ở listing 5.6 và 5.8 (đơn giản là chúng ta kết hợp 2 tập dữ liệu và thực thi truy vấn trên tất cả dữ liệu một lần). Kết quả như bên dưới:

|  |
| --- |
| **// listing 5.9**  { BlogId: "blogs/1234", CommentCount: 16 }  { BlogId: "blogs/7269", CommentCount: 2 }  { BlogId: "blogs/9313", CommentCount: 4 }  { BlogId: "blogs/2394", CommentCount: 1 } |

* Thật sự thì đó là toàn bộ bí mật của hàm map/reduce. Chúng ta có thể lấy 2 tập dữ liệu từ 2 node khác nhau (được lưu trữ trên 2 máy khác nhau) và áp dụng thuật toán map/reduce, chúng ta có thể nhận được kết quả cuối cùng cho một tập hợp mà ranh giới của các máy trải rộng.

Map/reduce là gì?

* Map/reduce đơn giản là cách chia khái niệm group by thành nhiều bước. Bằng cách chia nhỏ group by thành nhiều bước, chúng ta có thể thực thi một thao tác group by trên tập hợp nhiều máy tính, cho phép chúng ta thực hiện nhiều thao tác trên nhiều tập dữ liệu mà chúng quá lớn để lưu trữ trên một máy đơn. Map/reduce gồm 2 bước:
  + Bước đầu tiên là map. Map chỉ là một hàm (hoặc một truy vấn Linq) được thực hiện trên một tập dữ liệu. Nhiệm vụ của hàm map là lọc dữ liệu (mệnh đề where Linq) từ dữ liệu ban đầu và lấy ra những dữ liệu cần thiết từ dữ liệu đầu vào (mệnh đề Select).
  + Bước thứ hai trong quá trình map/reduce là hàm reduce (hoặc truy vấn Linq). Hàm recude có chức năng lấy kết quả đầu ra của hàm map và reduce (làm giảm) tập giá trị. Trên thực tế, hàm reduce luôn sử dụng mệnh đề group by để tập hợp tập dữ liệu đầu vào dựa trên khóa chung.
* Map/reduce phân tán dựa vào trình thực thi có thể thực thi hàm map, sau đó hàm reduce sẽ thực thi dựa trên kết quả của hàm map. Nếu nhiều nodes được sử dụng, trình thực thi sẽ kết hợp dữ liệu đã được reduce (làm nhỏ đi) từ nhiều nodes và thực thi reduce một lần nữa trên kết quả đã được kết hợp.

**-------------------------------------------**

**Quy luật cho thao tác map/reduce**

* RavenDB sử dụng truy vấn Linq để định nghĩa hàm map và hàm reduce.Truy vấn Linq có xu hướng tự nhiên phù hợp với với quy luật của hàm map/reduce, và một số điều quan trọng cần nhận ra một số quy luật là:
  + Hàm reduce phải có khả năng xử lý tốt kết quả đầu ra của hàm map cũng như kết quả đầu ra của chính nó. Đây là yêu cầu bắt buộc bởi vì hàm reduce có thể áp dụng đệ quy để xử lý kết quả đầu ra của chính nó. Trên thực tế, điều này có nghĩa là kết quả đầu ra của hàm map phải cùng kiểu với kiểu đầu ra của hàm reduce. Bởi vì cùng kiểu nên hàm reduce có thể chạy trên kết quả đầu ra của chính nó. Listing 5.10 là một ví dụ của cặp map/reduce trả về cùng kiểu dữ liệu:

|  |
| --- |
| **// Listing 5.10** - Map/reduce pair returning the same type.  // map  from post in docs.Posts  select new { post.BlogId, CommentCount = post.Comments.Length }  // reduce  from result in results  group result by result.BlogId into g  select new {  BlogId = g.Key,  CommentCount = g.Sum(x=>x.CommentCount)  } |

Và listing 5.11 là một ví dụ về cặp map/reduce không hợp lệ:

|  |
| --- |
| // Listing 5.11 - Map/reduce pair returning different types  // map  from post in docs.Posts  select new { post.BlogId, CommentCount = post.Comments.Length }  // reduce  from result in results  group result by result.BlogId into g  select new {  BlogId = g.Key,  TotalComments = g.Sum(x=>x.CommentCount)  } |

Giả sử chúng ta gởi kết quả đầu ra của hàm reduce trong listing 5.11 trở lại vào trong chính hàm đó thì chúng ta sẽ gặp lỗi bởi vì không có thuộc tính CountComment trong kết quả đầu ra của hàm reduce

* + Hàm map và reduce phải là những hàm thuần khiết. Hàm thuần khiết là những hàm thỏa mãn những điều kiện sau:
    - Đưa vào cùng kết quả đầu vào thì sẽ trả về cùng kết quả đầu ra. Ví dụ: [map(doc) == map(doc), cho bất kỳ document nào]. Điều này có nghĩa là chúng ta không thể dựa vào bất kỳ đầu vào bên ngoài nào, chỉ có thể chấp nhận những đầu vào hợp lệ.
    - Đánh giá chức năng sẽ không có tác dụng phụ. Điều này có nghĩa là trên thực tế chúng ta không thể thực hiên bất kỳ lời gọi nào bên ngoài hàm map/reduce. Đó không phải là một yêu cầu khó, bởi vì thông thường chúng ta không có cách nào thực hiện lời gọi từ bên ngoài.

**Áp dụng map/reduce**

* Như đã đề cập, map/reduce chỉ là cách tôn vinh việc sử dụng group by. Điều thú vị là nó hữu ích nhiều như thế nào. Một kết quả rõ ràng của map/reduce là chạy các tập hợp đơn giản:
  + Count
  + Sum
  + Distinct
  + Average

Và nhiều thứ khác tương tự như thế. Nhưng bạn cũng có thể sử dụng map/reduce để thực hiện join.

**Map/reduce làm việc trong RavenDB như thế nào**

* RavenDB sử dụng map/reduce cho phép chúng ta thực hiện phép kết hợp trên nhiều documents. Một điều quan trọng cần lưu ý là từ đầu tới giờ chúng ta chưa thấy RavenDB áp dụng map/reduce phân tán, nó chỉ chạy map/reduce trên máy đơn. Điều này đặt ra câu hỏi: nếu chúng ta sử dụng map/reduce chỉ trên máy tính đơn thì tại sao chúng ta phải bận tâm có nên dùng map/reduce trong khi chúng ta vẫn có thể thực hiện một truy vấn Linq với mệnh đề group by.
* Về lý thuyết, chúng ta có thể thực hiện điều đó(sử dụng truy vấn Linq với mệnh đề group by). Nhưng trong khi RavenDB không sử dụng map/reduce phân tán thì nó được dùng tránh những tính toán và nhập xuất không cần thiết. Bởi vì quá trình map/reduce có thể thay thế, có nghĩa là chúng ta có thể lưu trữ một cách hiệu quả và phân chia công việc khi cần thiết. Khi một document được đánh chỉ mục (indexed) bởi map/reduce index bị thay đổi, chúng ta sẽ chạy hàm map đối với document này, và sau đó reduce document cùng với kết quả reduce của tất cả các documents khác mà cùng chia sẻ chung reduce key.
* Listing 5.2 cho ta thấy hàm reduce:

|  |
| --- |
| **//Listing 5.12** - A sample reduce function  // reduce  from result in results  group result by result.BlogId into g  select new { BlogId = g.Key, CommentCount = g.Sum(x=>x.CommentCount) } |

* Reduce key trong danh sách 5.12 là giá trị của result.BlogId. RavenDB sử dụng key này để tối ưu giá trị mà nó gửi vào trong hàm reduce( thực sự thì group by được thực hiện bởi RavenDB mà không phải là bởi truy vấn Linq). Điều này dẫn đến chi phí sẽ ít hơn rất nhiều cho việc lập chỉ mục (indexing) cho map/reduce indexes so với việc chạy một truy vấn duy nhất với “group by” trên tất cả documents với cùng reduce key.
* Lưu ý: RavenDB chưa thực hiện việc re-reduce

**RavenDB lưu trữ kết quả của map/reduce indexes như thế nào**

* Ở phần trước chúng ta đã thảo luận RavenDB làm việc với kết quả của những indexes đơn giản như thế nào( chỉ bao gồm một hàm map). Map/reduce indexes thực sự tạo ra 2 điểm dữ liệu khác nhau. Điểm đầu tiên là kết quả đầu ra của hàm map, trong nội bộ những giá trị này được gọi là kết quả được ánh xạ (mapped results) bên trong RavenDB và không bao giờ trình bày ra bên ngoài, nhưng đó là những thứ cho phép RavenDB thực hiện những cập nhật index cục bộ.
* Kết quả đầu ra thứ hai là kết quả đầu ra của hàm reduce, đây là kết quả đầu ra bên ngoài có thể nhìn thấy được từ map/reduce index. Và giống như những indexes đơn giản, dữ liệu cũng được lưu trữ bên trong Lucene index. Lưu trữ dữ liệu trong Lucene index cho phép truy vấn hiệu quả và đầy đủ tính năng (full text searching…)
* Không giống như những index đơn giản (giả sử hầu hết là muốn tìm kiếm trên index, nhưng kết quả là những documents thực sự), map/reduce indexes không chỉ dùng như index, nhưng thực sự lưu trữ dữ liệu mà chúng ta cần như kết quả truy vấn. Ví dụ, nếu ta truy vấn index mà chúng ta định nghĩa ở listing 5.3 và listing 5.4 (và những kết quả đầu ra trong listing 5.9) cho kết quả blog blogs/9313 chúng ta nhận được:

{ BlogId: "blogs/9313", CommentCount: 4 }

Giá trị này được lưu trữ trong chính index, và nó sẽ được load lên trực tiếp từ đây. Điều này có nghĩa là chúng ta sẽ không đụng đến bất kỳ documents nào khi chúng ta truy vấn map/reduce index. Tất cả việc đó được quản lý bởi RavenDB và thực hiện bên dưới nền. Và giống như những index đơn giản, khi truy vấn map/reduce index sẽ cho kết quả cũ (stale results). Ta quản lý việc này giống với việc quản lý stale index của những index đơnn giản.

**Tạo map/reduce index đầu tiên**

* Sử dụng ví dụ về shopping cart, chúng ta muốn biết được số lượng của từng sản phẩm được bán ra. Listing 5.13 cho biết cấu trúc của shopping cart:

|  |
| --- |
| //listing 5.13 - a shopping cart document  { // shoppingcarts/1342  "Products": [  { "Id": "products/31", "Quantity":3 },  { "Id": "products/25", "Quantity":1 },  ]  } |

* Trước khi viết map/reduce index, chúng ta sẽ viết câu truy vấn Linq đầy đủ để làm việc này. Điều này sẽ giúp chúng ta dễ dàng viết map/reduce index hơn. Truy vấn Linq được viết bên dưới:

|  |
| --- |
| **// listing 5.14** - a linq query to calculate the count of products across all shopping carts  from shoppingCart from docs.ShoppingCarts  from product in shoppingCart.Products  group product by product.Id into g  select new { ProductId = g.Key, Count = g.Sum(x=>x.Count) } |

* Bước tiếp theo là chia truy vấn trong listing 5.14 thành nhiều bước nhỏ và tạo ra index. Sử dụng AbstractIndexCreationTask để tạo index:

|  |
| --- |
| **// listing 5.15** - The products count index  public class Products\_ByCountInShoppingCart : AbstractIndexCreationTask<ShoppingCart, ProductByCountProjection  {  public Products\_ByCountInShoppingCart()  {  Map = carts => from cart in carts  from product in cart.Products  select new { ProductId = product.Id, Count = product.Count };  Reduce = results => from result in results  group result by result.ProductId into g  select new { ProductId = g.Key, Count = g.Sum(x=>x.Count) };  }  } |

* Hàm map trong index này sẽ lấy số lượng của mỗi sản phẩm từ tất cả shopping cart, và hàm reduce sẽ tập hợp kết quả lại theo productId thành kết quả cuối cùng.

**Truy vấn map/reduce index**

* Giống như index bình thường, chúng ta có thể truy vấn map/reduce index bằng cách sử dụng session API. Listing 5.16 chỉ ra cách load lên số lượng của từng sản phẩm được bán ra:

|  |
| --- |
| **// listing 5.16** - querying a map / reduce index  var results = session.Query<ProductByCountProjection,Products\_ByCountInShoppingCart>()  .Where(x => x.ProductId == "products/31")  .ToList(); |

* Tham số đầu tiên của truy vấn là kiểu dữ liệu trả về, tham số thứ hai chính là index chúng ta sử dụng để truy vấn. Không giống như những index chuẩn(cũng còn được gọi là index đơn giản hay index chỉ có hàm map), kết quả trả về của map/reduce luôn luôn là một phép chiếu và không bao giờ là document gốc.

**Nên sử dụng map/reduce index khi nào**

* Map/reduce index rất hữu ích trong việc tổng hợp dữ liệu, nhưng không nên nhầm lẫn với giải pháp lập báo cáo tốt nhất. Chắc chắn là chúng ta có thể sử dụng map/reduce để thực hiện việc báo cáo, nhưng trong nhiều trường hợp một báo cáo yêu cầu nhiều hơn một map/reduce index có thể cung cấp(ví dụ như map/reduce index không thể hỗ trợ nhóm tùy ý)
* Map/reduce rất hữu ích khi chúng ta muốn xem dữ liệu dưới định dạng duy nhất. Một trong những trường hợp hay sử dụng là một phần khung nhìn của một trang chủ (homepage) hay trang điều khiển (dashboard). Một lợi thế quan trọng của map/reduce index trong RavenDB là nó được tính toán trước, có nghĩa là truy vấn tốn chi phí rất thấp (giống với index chuẩn).
* Điều đó làm cho nó thật lý tưởng trong việc tổng hợp một số lượng lớn dữ liệu mà được xem nhiều.