# Ví dụ 1

* Giả sử một document Article có một tập hợp Tags.
* Câu hỏi đặt ra là: số lượng của mỗi Tag trên tất cả các bài viết?

|  |
| --- |
| public class Content {     public int Id { get; set; }     public string Title { get; set; }      public IEnumerable<Tag> Tags { get; set; } }  public class Tag {     public string Name { get; set; } } |

* Thêm một số dữ liệu mẫu vào cơ sở dữ liệu:

|  |
| --- |
| using (var session = documentStore.OpenSession()) {     session.Store(new Content     {         Title = "Test Title for a Video",         Tags = new List<Tag>         {             new Tag() {Name = "c#"},             new Tag() {Name = "autofac"},             new Tag() {Name = "asp.net"},         }     });     session.Store(new Content     {         Title = "Test Title for an Article",         Tags = new List<Tag>         {             new Tag() {Name = "c#"},             new Tag() {Name = "nhibernate"},             new Tag() {Name = "fluent-nhibernate"},             new Tag() {Name = "mvc"}         }     });     session.Store(new Content     {         Title = "Test Title for an Article",         Tags = new List<Tag>         {             new Tag() {Name = "ravendb"},             new Tag() {Name = "asp.net"},             new Tag() {Name = "autofac"},             new Tag() {Name = "c#"}         }     });      session.SaveChanges(); } |

* Kết quả mong đợi sẽ là:
  + 3 x c#
  + 2 x autufact
  + 2 x asp.net
  + 1 x ravendb
  + 1 x mvc
  + 1 x nhibernate
  + 1 x fluent-nhibernate
* Tạo đối tượng mới TagResult chứa kết quả gồm Count và Name:

|  |
| --- |
| public class TagResult {     public int      Count   { get; set; }     public string   Name    { get; set; } } |

* Tạo index:

|  |
| --- |
| public class All\_Tags : AbstractMultiMapIndexCreationTask<TagResult> {     public All\_Tags()     {     } } |

* Chọn ra thuộc tính Name của Tags, chọn thêm một trường dữ liệu khác nữa gọi là Count với giá trị mặc định là 1. Như thế ta có thể sử dụng lại nó để tính tổng số lần Tags được sử dụng.

|  |
| --- |
| AddMap<Content>(contents => from content in contents                             from tag in content.Tags                             select new                             {                                 Name = tag.Name,                                 Count = 1                             }); |

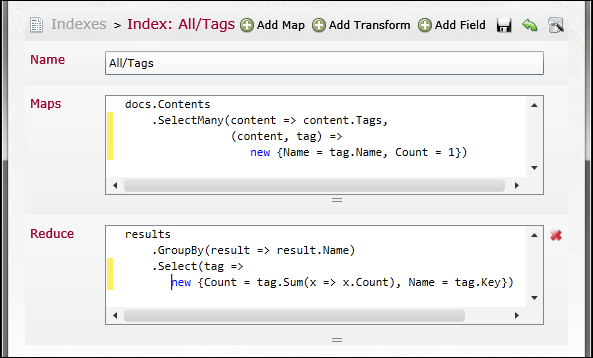
* Kết quả chứa nhiều bản sao của mỗi tags khi sử dụng hàm map trên:

|  |  |
| --- | --- |
| c# | 1 |
| c# | 1 |
| c# | 1 |
| autofac | 1 |
| autofac | 1 |
| asp.net | 1 |
| asp.net | 1 |
| ravendb | 1 |
| mvc | 1 |
| nhibernate | 1 |
| fluent-nhibernate | 1 |

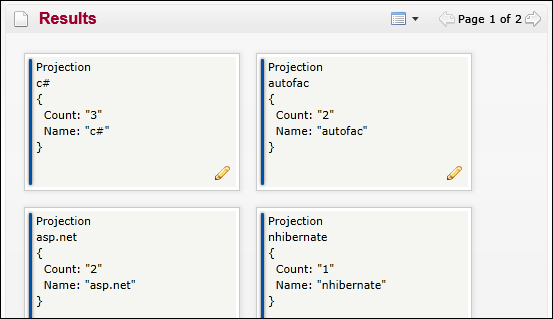
* Trong hàm Reduce, việc cần làm là nhóm các tags lại theo tên (Name)

|  |
| --- |
| Reduce = results => from result in results                     group result by result.Name into tag                     select new                     {                         Count = tag.Sum(x => x.Count),                         Name = tag.Key,                     }; |

* Chúng ta đã nhóm các tags lại theo tên của chúng và tính tổng thuộc tính “Count” để tính số lần mỗi tags được sử dụng.
* Chạy ứng dụng và xem index:



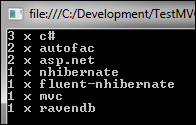
* Truy vấn index



* Trong chương trình, để truy vấn ta dùng class TagResult đã tạo lúc nãy và sử dụng index All\_Tags đã tạo:

|  |
| --- |
| using (var session = documentStore.OpenSession()) {     var result = session.Query<TagResult, All\_Tags>()                         .ToList();      foreach (var tag in result)     {         Console.WriteLine(tag.Count + " x " + tag.Name);     }      session.SaveChanges(); } |

* Chạy chương trình và ta lấy được kết quả:



* Kết quả giống như ta mong đợi. Và đó là Map/Reduce!

# Ví dụ 2

* Về cơ bản, map/reduce chỉ là cách chia một công việc lớn thành những công việc nhỏ riêng biệt mà có thể thực hiện song song.
* Giả sử chúng ta có một tập dữ liệu theo khuôn mẫu sau:

|  |
| --- |
| {  "type": "post",  "name": "Raven's Map/Reduce functionality",  "blog\_id": 1342,  "post\_id": 29293921,  "tags": ["raven", "nosql"],  "post\_content": "<p>...</p>",  "comments": [  {  "source\_ip": '124.2.21.2',  "author": "martin",  "text": "..."  }]  } |

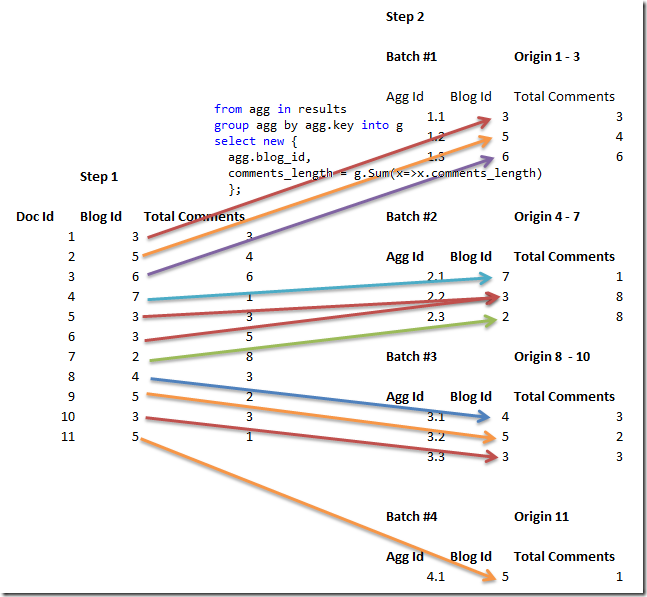
* Và chúng ta muốn trả lời câu hỏi mà cần nhiều documents mới trả lời được. Đó là loại thao tác mà đòi hỏi chúng ta phải sử dụng tổng hợp trên số lượng lớn dữ liệu. Nó được thực hiện tốt nhất bằng cách chia việc gọi là map/reduce.
* Map/reduce chỉ là 2 hàm thao tác trên một danh sách dữ liệu. Trong C#, Linq thực sự cho chúng ta cơ hội lớn để làm việc này theo cách làm cho nó đơn giản để hiểu và làm việc với nó.
* Chúng ta muốn lấy số lượng comments trên mỗi blog. Chúng ta có thể sử dụng truy vấn Map/reduce sau:

|  |
| --- |
| from post in docs.posts  select new {  post.blog\_id,  comments\_length = comments.length  };  from agg in results  group agg by agg.key into g  select new {  agg.blog\_id,  comments\_length = g.Sum(x=>x.comments\_length)  }; |

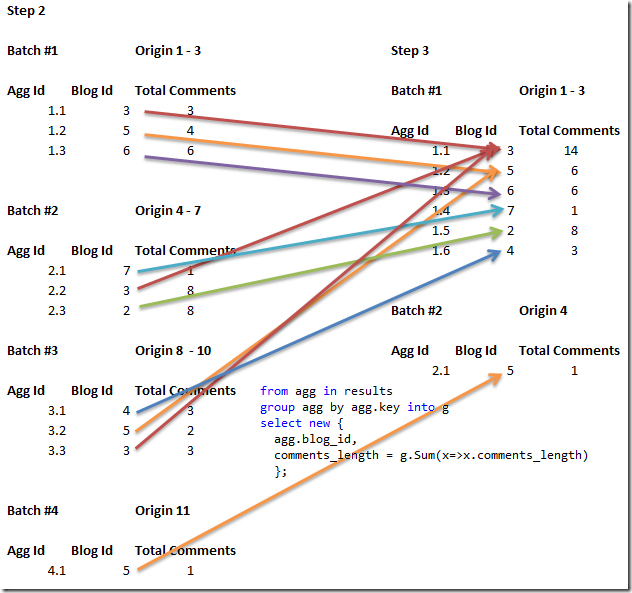
* Ở đây có 2 thứ cần lưu ý:
  + Truy vấn đầu tiên là hàm map, nó ánh xạ dữ liệu document đầu vào thành định dạng kết quả.
  + Truy vấn thứ hai là hàm reduce, nó thao tác trên một tập kết quả và đưa ra kết quả cuối cùng.
  + Lưu ý là: truy vấn reduce phải trả về cùng định dạng mà nó nhận đầu vào.
  + Giá trị đầu tiên trong kết quả là “key”, là thứ chúng ta tập hợp lại (nghĩ tới mệnh đề group by trong SQL)
* Chúng ta cùng xem nó đã làm việc như thế nào:
  + Bắt đầu bằng thực hiện hàm map trên tập dữ liệu mà chúng ta có, và kết quả đầu ra là:



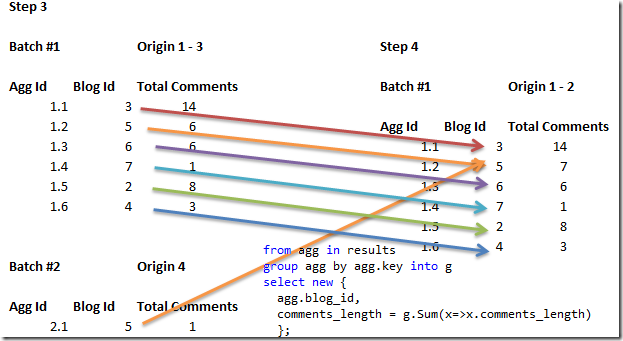
* Bước tiếp theo là reduce (làm giảm) kết quả. Về thuật toán thật sự của Map/Reduce thì sẽ phân chia đầu vào ban đầu, và làm việc cho tới kết quả cuối cùng. Trong trường hợp này, thử tưởng tượng kết quả đầu ra của bước đầu tiên được chia thành 3 nhóm (như vậy có tổng cộng 4 nhóm), và áp dụng hàm reduce lên kết quả này:



* Chúng ta có thể thấy được vì sao gọi là reduce(làm giảm). Với mỗi nhóm nhỏ (batch) chúng ta áp dụng tính tổng theo blog\_id để lấy được giá trị mới của Total Comments. Chúng ta đã làm việc với 11 dòng, và kết thúc với 10 dòng. Đây chính là điều thú vị, bởi vì chúng ta vẫn chưa xong việc, chúng ta vẫn phải tiếp tục reduce dữ liệu nhận được.
* Và đây là thứ chúng ta gọi là bước thứ 3, tiếp tục reduce dữ liệu. Đây cũng là lý do mà kết quả đầu vào và kết quả đầu ra phải giống nhau, chúng ta sẽ lấy kết quả đầu ra của hàm reduce đầu làm kết quả đầu vào cho hàm reduce kế tiếp. có thể thấy là chúng ta đã làm giảm từ 10 dòng xuống còn 7 dòng.



* Bước cuối cùng:



* Và chúng ta đã hoàn thành, chúng ta không thể reduce dữ liệu được nữa vì tất cả các key đã là duy nhất.
* Và còn một tính chất thú vị khác của Map/reduce, giả sử chúng ta vừa thêm một comment vào một bài post, điều đó rõ ràng kết quả truy vấn giờ đã không đúng nữa.
* Nhưng đó không phải là tất cả. Giả sử chúng ta vừa thêm một comment vào bài viết có id là 10, cần làm gì để tính lại kết quả cho đúng:
  + Map Doc #10 again
  + Reduce Step 2, Batch #3 again
  + Reduce Step 3, Batch #1 again
  + Reduce Step 4
* Điều quan trọng là chúng ta không phải làm việc với toàn bộ dữ liệu như trước, nói cách khác tính toán lại theo cách map/reduce làm sẽ ít tốn chi phí hơn rất nhiều.