* Theo bài trước: WrongApproach\_Modeling Documents in a Document Database, bây giờ chúng ta sẽ thiết kế mô hình dữ liệu sử dụng document database đúng cách.
* Một số yêu cầu khi hiển thị dữ liệu trong ứng dụng:
  + Main page: show list of blogs
  + Main page: show list of recent posts
  + Main page: show list of recent comments
  + Main page: show tag cloud for posts
  + Main page: show categories
  + Post page: show post and all comments
  + Post page: add comment to post
  + Tag page: show all posts for tag
  + Categories page: show all posts for category
* Đây là một số dữ liệu ví dụ mà chúng ta sẽ làm việc với chúng:

|  |
| --- |
| var user = new User("ayende");  var blog = new Blog("Ayende @ Rahien", user) { Tags = {".NET", "Architecture", "Databases" } };  var categoryRaven = new Category("Raven");  var categoryNoSQL = new Category("NoSQL");  var post = new Post(blog, "RavenDB", "... content ...")  {  Categories = { categoryRaven, categoryNoSQL },  Tags = {"RavenDB", "Announcements" }  };  var comment = new Comment(post, "Great news");  PersistAll(user, blog, categoryRaven, categoryNoSQL, post, comment); |

* Khi thiết kế mô hình dữ liệu chúng ta sẽ nghĩ đến Aggregates(tập hợp). Nhữn thực thể ở mô hình trên có phải là những Aggregates? User, Blog và Post là những Aggregate vì chúng có thể tồn tại một cách độc lập. Những thực thể còn lại không thể tồn tại riêng biệt được. Tags là Value Objects (những đối tượng chứa giá trị), Categories dùng để liệt kê dữ liệu, Comments thì không thật sự có ý nghĩa nếu tồn tại bên ngoài bài viết của nó (Post). Định dạng được lựa chọn đó là:

|  |  |  |
| --- | --- | --- |
| |  | | --- | | // users/ayende  {  "type": "user",  "name": "ayende"  }  // blogs/1  {  "type": "blog",  "users": ["users/ayende"],  "name": "Ayende @ Rahien",  "tags": [".NET", "Architecture", "Databases"]  } | | // posts/1  {  "blog": "blogs/1",  "title": "RavenDB",  "content": "... content ...",  "categories": ["Raven", "NoSQL"]  "tags" : ["RavenDB", "Announcements"],  "comments": [  { "content": "Great News" }  ]  } | |

* Điều này làm cho mô hình trở nên nhỏ hơn, và làm nhiều thứ như lấy dữ liệu của Post và những Comments của nó ít tốn chi phí. Và chúng ta xem lại một số yêu cầu của ứng dụng vơi mô hình mới này.
* Main page: show list of blogs

|  |
| --- |
| var blogs = docDb.Query<Blog>("DocumentsByType", "type:blog"); |

* Main page: show list of recent posts
  + - Index fuction:

|  |
| --- |
| from doc in docs  where doc.type == "post"  select new {doc.posted\_at} |

* + - Truy vấn dữ liệu sử dụng:

|  |
| --- |
| docDb.Query<Post>("Posts", orderBy:"-posted\_at"); |

* Main page: show list of recent comments:
  + Đây là một điều thú vị vì bây giờ chúng ta không còn khái niệm Comments như một thực thể riêng biệt. Chúng ta định nghĩa index để rút trích thông tin từ bài viết:

|  |
| --- |
| from doc in docs  where doc.type == "post"  from comment in doc.comments  select new {comment.posted\_at, comment.content } |

* + Và truy vấn như sau:

|  |
| --- |
| var recentComments = docDb.Query<Comment>("CommentsByTime", orderBy:"-posted\_at"); |

* Main page: show tag cloud for posts
  + Đây cũng là một điều thú vị khác, chúng ta cần một tập hợp các Tags. Nó được thực hiện bởi thao tác Map/Reduce. Nghe thì có vẻ khủng khiếp nhưng thực sự thì dễ dàng bằng cách sử dụng 2 câu lệnh LINQ:

|  |
| --- |
| from doc in docs  where doc.type == "post"  select tag in doc.tags  select new { tag, count = 1}  from result in results  group result by result.tag into g  select new {tag = g.Key, count = g.Sum(x=>x.count) } |

* + Và chúng ta dùng nó để truy vấn dữ liệu:

|  |
| --- |
| var tagCloud = docDb.Query<TagAndCount>("TagCloud"); |

* Main page: show categories:
  + Tương tự như Tags, chúng ta rút trích thông tin categories từ Posts, và lấy số lượng bài viết trong mỗi categories:

|  |
| --- |
| from doc in docs  where doc.type == "post"  select category in doc.categories  select new { category, count = 1}  from result in results  group result by result.category into g  select new {category= g.Key, count = g.Sum(x=>x.count) } |

* + Và truy vấn dữ liệu:

|  |
| --- |
| var categories = docDb.Query<CategoryAndCount>("CategoriesCount"); |

* Post page: show post and all comments
  + Thật dễ dàng để quản lý đối tượng Post, tất cả những gì chúng ta cần là lấy post document, và chúng ta lấy mọi thứ chúng ta cần để hiển thị trên trang post:

|  |
| --- |
| var post = docDb.Get<Post>("posts/1"); |

* Post page: add comment to post
  + Nên nhớ rằng, comment không thể tồn tại bên ngoài Post được, vì vậy để thêm comment thì chúng ta sẽ lấy Post, thêm Comment và lưu lại:

|  |
| --- |
| var post = docDb.Get<Post>("posts/1");  post.Comments.Add(new Comment(...));  docDb.SaveChanges(); |

* Tag page: show all posts for tag
  + Index

|  |
| --- |
| from doc in docs  where doc.type == "post"  from tag in doc.tags  select new { tag } |

* + Truy vấn như sau

|  |
| --- |
| var posts = docDb.Query<Post>("PostsByTag", "tag:Raven"); |

* Categories page: show all posts for category: tương tự all post for tags
* Một số lưu ý:
  + Để có kỹ thuật thiết kế mô hình tốt thì nghĩ tới Aggregates trong DDD(Domain Driven Design), một Aggregate và tất cả các liên quan tới nó đều được chứa trong một document duy nhất.
  + Nếu chúng ta muốn truy vấn thông tin một thực thể có trong Aggregate thì chúng ta sử dụng index để rút trích thông tin.
  + Chúng ta có thể tạo ra các thực thể ảo (“virtual entities”) như là Tags hay Categories, những thực thể không thực sự tồn tại, nhưng được tạo ra trong index khi chúng ta định nghĩa. Những thực thể ảo hữu dụng khi dùng trong tìm kiếm/tổ chức một Aggregate.
  + Schema không quan trọng bởi vì tìm kiếm dựa trên key những well-known indexs , trả về những định dạng dữ liệu mà chúng ta mong đợi.