**BÁO CÁO THỰC HÀNH NHIBERNATE**

**(Tiếp theo)**

Họ và tên: Phan Quốc Phong

Mục lục

[I. Chương trình 3](#_Toc16536413)

[II. Ý tưởng: 5](#_Toc16536414)

[a. Sql Server: 5](#_Toc16536415)

[b. Source Code: 5](#_Toc16536416)

[III. Báo cáo 6](#_Toc16536417)

[a. Cơ sở dữ liệu 6](#_Toc16536418)

[b. Tối ưu Query 7](#_Toc16536419)

[c. Tối ưu Query cho Multi task 9](#_Toc16536420)

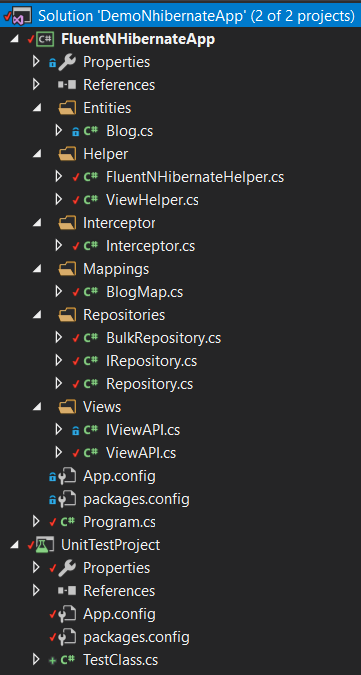
[d. Test hiệu năng 10](#_Toc16536421)

[e. Khác 11](#_Toc16536422)

1. Chương trình

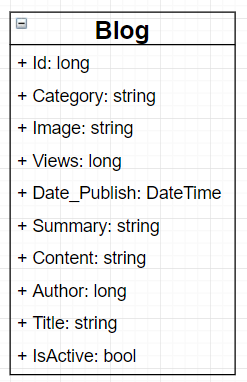
Solution **DemoNhibernateApp**, chỉ sử dụng 2 project sau

* 1. **FluentNHibernateApp**: Source code chương trình chính
     + Entites: thư mục chứa lớp định nghĩa đối tượng
     + Mappings: thư mục chứa lớp định nghĩa ánh xạ
     + Helper: thư mục chứa 2 lớp helper cấu hình SessionFactory.
     + Interceptor: thư mục chứa lớp định nghĩa một Interceptor cho Session.
     + Repostitories: thư mục chứa Interface và Implement tương ứng cho các nghiệp vụ Insert, Update, Delete.
     + Views: thư mục chứa Interface và Implement tương ứng cho nghiệp vụ truy vấn dữ liệu.
  2. **UnitTestProject**: Project UnitTest
     + **TestClass**: Đặt các phương thức test được sử dụng.



Link github: <https://github.com/qphonghcmus/DemoFluentNHibernate>

1. Ý tưởng:
   1. Sql Server:
   * Sử dụng kỹ thuật Replication để tạo ra nhân bản (Subcriber) của cơ sở dữ liệu chính (Publisher) nhằm chia tải khi chạy bất đồng bộ.
   * Cơ sở dữ liệu thứ nhất (Publisher) là đối tượng thao tác của các nghiệp vụ Insert, Update, Delete dữ liệu.
   * Cơ sở dữ liệu thứ hai (Subcriber) là đối tượng thao tác của các nghiệp vụ truy vấn dữ liệu.
   * Thông qua cơ chế của Sql Server (Job) , khi dữ liệu trên Publisher bị thay đổi, Publisher sẽ đồng bộ các thay đổi qua Subcriber, đảm bảo sự đồng nhất dữ liệu giữa Publisher và Subcriber.
   * Tại cả hai cơ sở dữ liệu, tạo Index cho các trường quan trọng để truy vấn trở nên nhanh hơn.
   * Dùng kỹ thuật Partition để chia bảng ra thành nhiều phần, giảm tải khi thự thi truy vấn đồng bộ trên bảng và dễ dàng quản lý.
   * Tại Subcriber, tạo ra các View tương ứng nhu cầu truy vấn thay vì truy vấn toàn bộ trường. Khi truy vấn theo nhu cầu (báo cáo) sẽ truy vấn tới view tương ứng.
   1. Source Code:
   * Lớp Blog định nghĩa đối tượng tương ứng với bảng trong cơ sở dữ liệu
   * Lớp BlogMap định nghĩa các ánh xạ cho đối tượng vào bảng dữ liệu.
   * Lớp FluentNHibernateHelper định nghĩa một SessionFactory tương ứng với Publisher, phương thức trả về một đối tượng Repository.
   * Lớp ViewHelper định nghĩa một SessionFactory tương ứng với Subcriber, phương thức trả về một đối tượng ViewAPI.
   * Đối với các nghiệp vụ Insert, Update, Delete, sử dụng các phương thức được định nghĩa trong interface IRepository và triển khai trong Repository. BulkRepository tương tự nhưng chỉ dùng để Insert một lượng lớn record vào bảng ( thường là hàng triệu record).
   * Đối với các nghiệp vụ truy vấn (Select), sử dụng các phương thức được định nghĩa trong Interface IViewAPI và triển khai trong ViewAPI.
   * Cả Repository và ViewAPI đều sử dụng đối tượng Session của Nhibernate và các API của nó để giao tiếp với cơ sở dữ liệu. Đối tượng session này được truyền từ SessionFactory tương ứng (FluentNHibernateHelper – Repository, ViewHelper – ViewAPI).
   * Khi các lớp Helper truyền các Session tạo từ SessionFactory, truyền vào contructor của Session một đối tượng Interceptor ( định nghĩa trong lớp Interceptor) để thêm các hint vào câu truy vấn sẽ được thực thi trên Sql Server (dùng trong truy vấn bất đồng bộ).
2. Báo cáo
3. Cơ sở dữ liệu
   * Sơ đồ class:



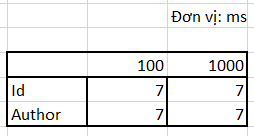
* + Sơ đồ database:



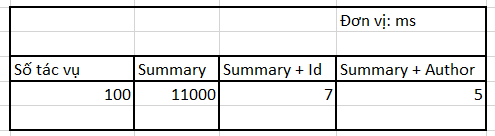
* + Số lượng record ban đầu là 50 triệu.
  + Khoá chính là trường Id, Partition key là Author.
  + Clustered index của bảng Author; non-clustered index là Id.
  + Bảng được partition thành 5 vùng theo trường Author:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Partition** | **Partition 1** | **Partition 2** | **Partition 3** | **Partition 4** | **Partition 5** |
| Dải giá trị | [- ∞ ;10,000,000] | (10,000,000 ; 20,000,000] | (20,000,000 ; 30,000,000] | (30,000,000 ; 40,000,000] | (40,000,000 ; + ∞] |

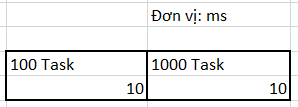
1. Tối ưu Query
   * Get:
     + Truy vấn trên trường Id và Author: mỗi tác vụ lấy ra 100 record, xác định thời gian thực thi trung bình của tác vụ.
       - Khi chạy tuần tự 100 tác vụ:
         * Query theo Id: 7 ms
         * Query theo Author: 7 ms
       - Khi chạy tuần tự 1000 tác vụ,:
         * Query theo Id: 7 ms
         * Query theo Author: 7 ms



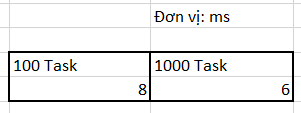
* + - Truy vấn trên trường Summary (chuỗi): lấy ra 1 record có giá trị Summary bằng chuỗi điều kiện.
      * 100 tác vụ tuần tự.
      * Không kết hợp trường Index : 11000 ms
      * Kết hợp trường Id: 7 ms
      * Kết hợp trường Author: 5 ms



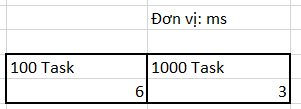
* + Insert:
    - Chạy tuần tự các tác vụ, mỗi tác vụ tạo một mảng 50 object mới.
    - Mỗi object tương ứng sẽ là một record mới cần insert vào bảng.
    - Khi chạy tuần tự 100, thời gian thực thi trung bình là 19 ms.
    - Khi chạy tuần tự 1000, thời gian thực thi trung bình là 12 ms



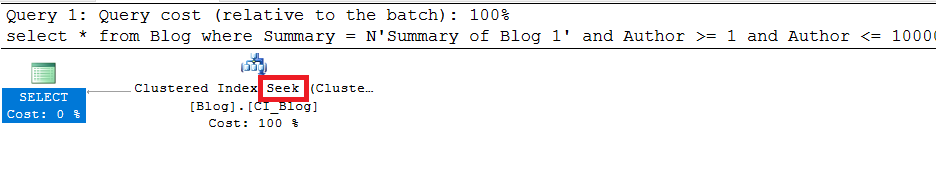
* + Update:
    - Chạy tuần tự các tác vụ, mỗi lần lấy ra một 50 record lưu vào mảng object tương ứng.
    - Cập nhật lại giá trị trường IsActive của các object.
    - Thời gian thực thi Update tính từ khi gọi API đến khi kết thúc.

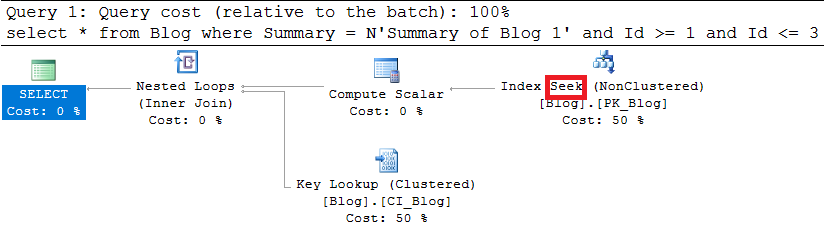


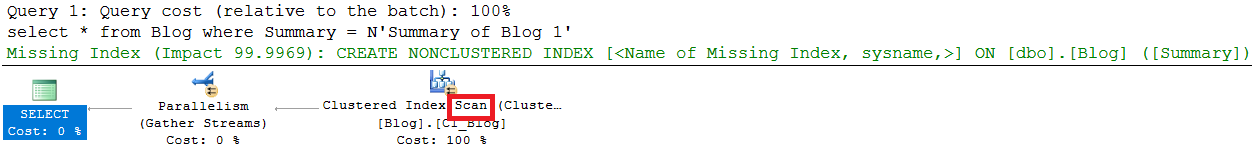
* + Delete:
    - Chạy tuần tự các tác vụ, mỗi tác vụ lấy ra 50 record rồi xoá đi 50 record thông qua API.
    - Thời gian thực thi tính từ khi gọi API xoá.
    - Khi chạy tuần tự 100 tác vụ, thời gian xoá trung bình là 6 ms
    - Khi chạy tuần tự 1000 tác vụ, thời gian xoá trung bình là 3 ms



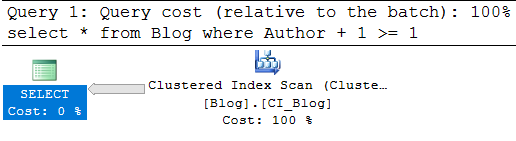
* Nhận xét:
  + Truy vấn có sử dụng các trường được đánh index giúp cải thiện tốc độ rõ rệt.
  + Thay vì phải quét (scan) trên toàn bộ bảng, ta chỉ tìm kiếm (seek) trong một đoạn nhỏ .







* + Nếu truy vấn chỉ sử dụng trường không được đánh index (Summary), tốc độ vẫn rất chậm, không có tác dụng gì.
  + Truy vấn trên trên trường là Author (non-clusterd index ) nhỉnh hơn so với truy vấn trên trường Id (clustered index). Đối với Id, ngoài seek ta phải làm thêm thao tác look up.
  + Với các cột được tạo index, không nên đặt các phép tính toán phức tạp trong mệnh đề where vì sẽ làm mất index của cột, làm cho tốc độ truy vấn chậm hơn.



1. Tối ưu Query cho Multi task

* Chạy bất đồng bộ 100 task, mỗi task liên tục query lấy 100 record trong bảng.
* Sau thời gian 5 phút, kết thúc toàn bộ task.
* Có được thời gian thực thi trung bình của mỗi task ( mỗi task có số lần query khác nhau).
* Tính thời gian thực thi trung bình của 100 task.
  + Chạy song song 100 task:
    - Query theo Id : 90 ms
    - Query theo Author: 80 ms
* Nhận xét:
  + Mỗi task có số lần thực hiện query chênh lệch. Các task đầu tiên thực hiện rất nhiều lần ( hàng ngàn lần), các task sau thực hiện ít hơn ( vài trăm lần).
  + Khi để thời gian thực hiện ít ( 2 phút hoặc 3 phút), nhiều task chưa được thực thi mà chủ yếu là các task trước (chạy rất nhiều lần) => Tăng thời gian đo để các Task thực hiện đầy đủ.
  + Mỗi task khi thực hiện một lần query sẽ lấy toàn bộ các trường trong bảng nên sẽ mất thời gian và không thực tế vì thông thường ta chỉ cần thông tin một số trường => Tạo ra view bao gồm các trường cần thiết để truy vấn nhanh hơn.
  + Thêm hint with nolock vào câu query sau khi Nhibernate parse lệnh thành sql string để truy vấn không bị lock trên bảng (đổi lại tính chính xác của dữ liệu không được đảm bảo).

1. Test hiệu năng

* Tiến hành chạy song song 60 Task:
  + 20 Task Insert, mỗi task liên tục insert 50 record khác nhau.
  + 20 Task Delete, mỗi task liên tục delete 50 record khác nhau.
  + 20 Task Select, mỗi task liên tục truy vấn 50 record khác nhau
* Sau thời gian thực thi 60s, kết thúc toàn bộ task.
* Các object tương ứng với các record mới được phát sinh ngẫu nhiên, vì vậy có thể xảy ra lỗi trùng key (Author). Tạm thời bỏ qua lỗi này để thuận lợi cho việc đo thời gian thực thi.
* Có được thời gian thực thi trung bình (mỗi task) của task Get, Insert, Delete.
* Tính thời gian thực thi trung bình (20 task) của toàn bộ task Get, Insert, Delete.
* Kết quả:
  + - Insert: 70 ms
    - Get: 7 ms (theo Author)
    - Delete: 15 ms
* Nhận xét:
  + Các task Get chỉ thực thi trên csdl ở Subcriber, đã được partitio và truy vấn theo Author nên thời gian thực thi rất nhỏ.
  + Đối với các lệnh Insert hoặc Delete nhiều phần tử, việc đóng gói tất cả thành một transaction duy nhất, tiến hành Commit sau cùng giúp cải thiện tốc độ rõ rệt so với việc mỗi Insert là một transaction.
  + Cơ chế lưu cache của SQL Server cũng giúp ta cải thiện tốc độ truy vấn: càng về sau các câu lệnh của mỗi loại tốn thời gian ít hơn (vẫn có trường hơn ngoại lệ nếu điều kiện truy vấn liên quan đến nhiều bảng nhỏ).
  + Việc thực hiện chạy bất đồng bộ trên cùng một máy và thời gian chênh lệch giữa Insert và Delete tạo ra một tình huống: nhiều lệnh delete được thực hiện hơn so với lệnh insert trong khoảng thời gian đo => Tăng thời gian đo hoặc Insert bổ sung sau cùng.
  + Nhìn chung, việc áp dụng các kỹ thuật của SQL SERVER và cách tổ chức code khoa học so với việc chỉ đơn thuần sử dụng API của Nhibernate giúp nâng cao tốc độ truy vấn, cập nhật trên cơ sở dữ liệu lớn.

1. Khác

* View:
  + Là một bảng ảo, được tạo ra từ một bảng hoặc nhiều bảng với một tập các cột và dòng chứa dữ liệu; nhưng không được xem là cấu trúc lưu trữ dữ liệu tồn tại trong cơ sở dữ liệu.
  + Dữ liệu tự động cập nhật theo bảng chính mà view tham chiếu tới.
  + View giúp đơn giản hoá và tập trung dữ liệu khi cho phép ta chọn một số trường quan trọng trong các bảng.
  + Hạn chế phần nào việc người dùng truy cập trực tiếp dữ liệu, một người dùng thường chỉ chỉ được cho phép truy vấn đến một số trường dữ liệu nhất định.
  + View là tập hợp dữ liệu từ nhiều bảng, thay vì phải dùng các truy vấn phức tạp đến các bảng, người dùng có thể truy vấn thông qua view.
  + Thông thường các truy vấn trên view sẽ được chuyển thành các truy vấn trên bảng chính nên nếu view được định nghĩa bởi một truy vấn phức tạp thì chi phí khi thực hiện truy vấn trên view sẽ lớn.
  + Về lý thuyết, có thể cập nhật bảng chính thông qua view nhưng chỉ hạn chế đối với các view đơn giản. Về mặt logic, view chỉ nên dành cho việc đọc dữ liệu (trích xuất báo cáo).
* Job – Sql Server Agent
  + Sql Server Agent là một module trong Sql Server dùng để thực hiện một công việc nào đó tự động theo thời gian mà người dùng quy định.
  + Thông thường người dùng có nhu cầu trích xuất và tính toán dữ liệu từ các bảng như các bảng báo cáo hàng ngày, hàng tuần; việc tính toán không thể thực hiện thủ công tại lúc truy vấn vì sẽ mất thời gian. Sql Server Agent sẽ giúp ta thực hiện công việc này tự động theo lịch trình.

* Replication:
  + Replication là một kỹ thuật quan trọng trong việc sao chép và phân phối cơ sở dữ liệu giữa các Server và đồng bộ chúng nhằm duy trì tính nhất quán dữ liệu.
  + Replication là một giải pháp scale out, hướng tới khả năng phục vụ nhiều truy vấn đồng thời thông qua việc tăng số lượng Server nhưng luôn đảm bảo tính nhất quán.
  + Ngoài ra Replication còn là một giải pháp tuyệt vời cho bài toán real time backup. Đối với các hệ thống có cở sở dữ liệu lớn, việc backup không thể thực hiện thường xuyên và việc Server sập dù trong khoảng thời gian ngắn cũng sẽ gây thiệt hại lớn. Replication đảm bảo dữ liệu của người dùng luôn an toàn và sẵn sàng khi cần.
* Database mirroring:
  + Là một giải pháp xây dựng cơ sở dữ liệu có tính sẵn sàng cao (gần như luôn hoạt động) trong Sql Server.
  + Trong trường hợp có sự cố xảy ra, database mirror có thể được sử dụng để client truy cập
  + Database mirroring (DM) tăng khả năng bảo vệ cơ sở dữ liệu nhờ duy trì 1 bản sao của CSDL.
  + Mỗi CSDL chính chỉ có một CSDL mirror.
  + Cơ chế hoạt động:
    - DM yêu cầu có 3 Sql Server Instance:
    - Pricipal: quản lý cơ sở dữ liệu chính
    - Mirror: quản lý bản copy của CSDL chính
    - Witness: không bắt buộc có, không bắt buộc chứa cơ sở dữ liệu; kết nối tới Pricipal và Mirror, tự động điều chỉnh Mirror trở thành Pricipal khi gặp sự cố.
  + Khi chuyển cơ chế DM được kích hoạt, các replication sẽ hỏng, cần phải khởi tạo lại.