**BÁO CÁO THỰC HÀNH NHIBERNATE**

Họ và tên: Phan Quốc Phong

Mục lục

[I. Chương trình 3](#_Toc15656622)

[II. Báo cáo 4](#_Toc15656623)

[a. Tạo cơ sở dữ liệu với 5 triệu record 4](#_Toc15656624)

[b. Tối ưu Query 5](#_Toc15656625)

[c. Tối ưu Query cho Multi task 8](#_Toc15656626)

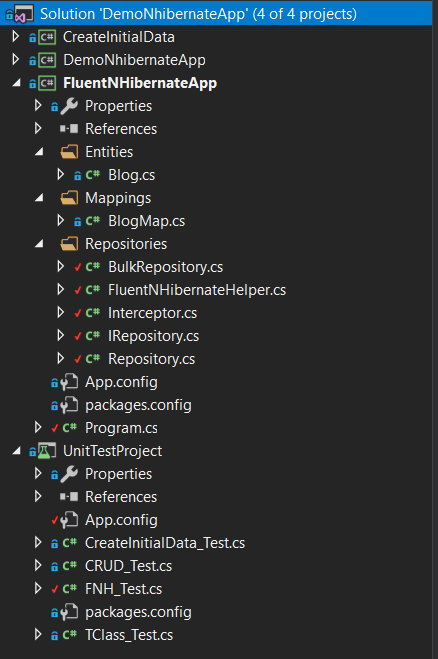
[d. Test hiệu năng 10](#_Toc15656627)

[e. Khác 11](#_Toc15656628)

1. Chương trình

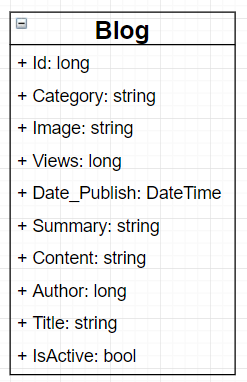
Solution **DemoNhibernateApp**, chỉ sử dụng 2 project sau

* 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
     + Repostitories: thư mục chứa lớp Helper, lớp Interceptor, Interface và lớp Implement tương ứng.
  2. **UnitTestProject**: Project UnitTest, chỉ sử dụng lớp sau
     + **FNH\_Test**: Đặt các phương thức test được sử dụng.



Link github:

1. Báo cáo
   1. Tạo cơ sở dữ liệu với 5 triệu record
   * Sơ đồ class:



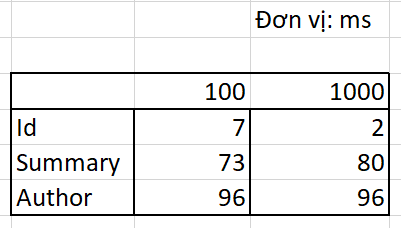
* + Sơ đồ database:



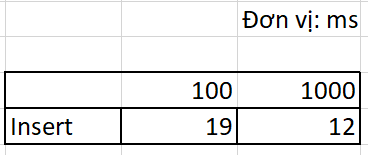
* + Sử dụng SqlBulkCopy để Insert số lượng lớn Record
    - Thời gian Insert 5 triệu record trung bình 40 s
    - Để Insert bằng SqlBulkCopy, ta cần có được tên table, tên các column (bản chất của SqlBulkCopy là thực hiện Copy từ một table này sang table khác); ta phải làm các bước từ xác định tên table, tên column tới Copy, tạo đối tượng DataTable, tạo các column, các row chứa dữ liệu ...
  + Table có số lượng record lớn sẽ khó khăn cho việc quản lý, lưu trữ, truy vấn, cập nhật, ...
  + Thông thường ta không cần phải truy vấn và cập nhật trên toàn bộ bảng mà chỉ thực hiện trên một vùng.
  + Việc chia vùng còn giúp cho việc truy vấn thống kê hiệu quả hơn.
  + Sử dụng Partition để chia bảng, chọn trường Author (long – bigint) làm partition key, chia bảng thành 5 vùng ( partition) tương ứng với giá trị của trường Author như sau:

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

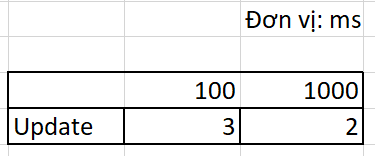
* 1. Tối ưu Query
* Chưa chia partition, khoá chính là trường Id (clustered index duy nhất), các trường khác không có index.
  + Get:
    - Khi chạy tuần tự 100 tác vụ, mỗi tác vụ lấy ra 100 record thì thời gian thực thi trung bình của tác vụ là :
      * Query theo Id (Primary key): 7 ms
      * Query theo Summary (chuỗi): 73 ms
      * Query theo Author: 96 ms
    - Khi chạy tuần tự 1000 tác vụ,:
      * Query theo Id (Primary key): 2 ms
      * Query theo Summary (chuỗi): 80 ms
      * Query theo Author: 100 ms



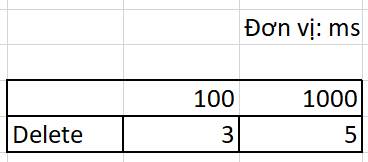
* + Insert:
    - Chạy tuần tự các tác vụ, mỗi tác vụ tạo ngẫu nhiên một mảng object mới.
    - Mỗi object tương ứng sẽ insert một record vào cơ sở dữ liệu.
    - Khi chạy tuần tự 100, thời gian thực thi trung bình là 19 ms.
    - Khi chạy tuần tự 1000, thời gian thực thi trung bình là 12 ms



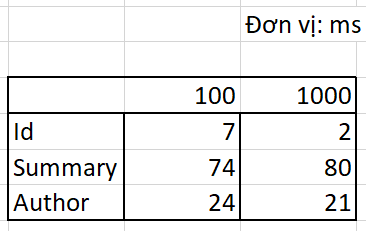
* + Update:
    - Chạy tuần tự các tác vụ, mỗi lần lấy ra một 50 record lưu vào mảng object tương ứng.
    - Cập nhật lại giá trị trường trong mảng tương ứng.
    - Thời gian thực thi Update tính từ khi gọi API đến khi kết thúc.
    - Đối với 100 tác vụ, thời gian cập nhật mỗi mỗi tác vụ trung bình là 3 ms
    - Đối với 1000 tác vụ, thời gian cập nhật mỗi mỗi tác vụ trung bình là 2 ms



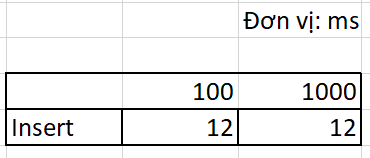
* + Delete:
    - Chạy tuần tự các tác vụ, mỗi tác vụ lấy ra 50 record rồi xoá đi 50 record thông qua API.
    - Thời gian thực thi tính từ khi gọi API xoá.
    - Khi chạy tuần tự 100 tác vụ, thời gian xoá trung bình là 3 ms
    - Khi chạy tuần tự 1000 tác vụ, thời gian xoá trung bình là 5 ms



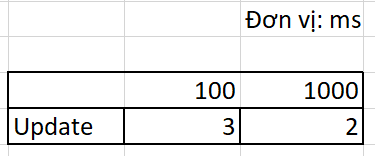
* Chia 5 partition, clustered index sau khi tạo lại gồm Id (duy trì tính duy nhất) và Author (partition key).
  + Get:
    - Khi chạy tuần tự 100 tác vụ,:
      * Query theo Id (Primary key): 6 ms
      * Query theo Summary (chuỗi): 74 ms
      * Query theo Author: 24 ms
    - Khi chạy tuần tự 1000 tác vụ,:
      * Query theo Id (Primary key): 2 ms
      * Query theo Summary (chuỗi): 80 ms
      * Query theo Author: 21 ms



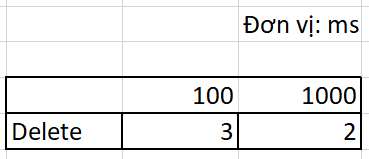
* + Insert:
    - Khi chạy tuần tự 100, thời gian thực thi trung bình là 12 ms.
    - Khi chạy tuần tự 1000, thời gian thực thi trung bình là 12 ms



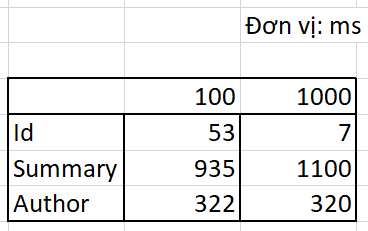
* + Update:
    - Đối với 100 tác vụ, thời gian cập nhật mỗi mỗi tác vụ trung bình là 3 ms
    - Đối với 1000 tác vụ, thời gian cập nhật mỗi mỗi tác vụ trung bình là 2 ms



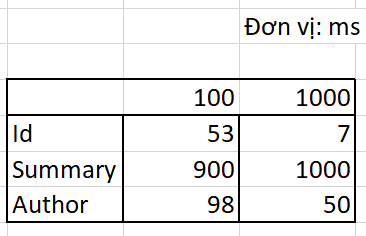
* + Delete:
    - Khi chạy tuần tự 100 tác vụ, thời gian xoá trung bình là 3 ms
    - Khi chạy tuần tự 1000 tác vụ, thời gian xoá trung bình là 2 ms



* Nhận xét:
  + Việc chia bảng (partition table) không giúp cải thiện tốc độ truy vấn trên các trường không được đánh index (partition key phải được đánh index).
  + Với trường được đánh index (Id), tốc độ không cải thiện.
  + Truy vấn trên trường được tạo index tốt hơn trên các trường không được tạo index.
  + Truy vấn trên trên trường là partition key cải thiện tốc độ gấp 4 lần. Thay vì quét trên toàn bộ bảng, ta chỉ làm việc trong một hoặc một số vùng của bảng.
  + 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 thực hiện query get 100 record (có thể trùng nhau giữa các task).
* Để thời gian thực hiện truy vấn tốt hơn, mỗi lần Nhibernate phân tích câu lệnh thành các lệnh Select của Sql, ta thêm hint “With (Nolock)” vào câu truy vấn.
* Chưa chia partition, khoá chính là trường Id (clustered index duy nhất), các trường khác không có index.
  + Chạy song song 100 task:
    - Query theo Id : 53 ms
    - Query theo Summary (chuỗi, không có index): 935 ms
    - Query theo Author: 322 ms
  + Chạy song song 1000 task:
    - Query theo Id (Primary key): 7 ms
    - Query theo Summary (chuỗi, không có index): 1100 ms
    - Query theo Author: 320 ms



* Chia 5 partition, clustered index sau khi tạo lại gồm Id (duy trì tính duy nhất) và Author (partition key).
  + Chạy song song 100 task:
    - Query theo Id : 53 ms
    - Query theo Summary (chuỗi, không có index): 900 ms
    - Query theo Author: 98 ms
  + Chạy song song 1000 task:
    - Query theo Id (Primary key): 7 ms
    - Query theo Summary (chuỗi): 1000 ms
    - Query theo Author: 50 ms



* Nhận xét:
  + Việc chia bảng không giúp cải thiện tốc độ truy vấn trên các trường không được đánh index hoặc đã được đánh index từ trước.
  + Tốc độ truy vấn trên trường được tạo index tốt hơn các trường không được tạo index.
  + Truy vấn trên trên trường là partition key cải thiện tốc độ rõ rệt. Ta chỉ còn làm việc trong một hoặc một số vùng của bảng thay vì toàn bộ bảng.
  + 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. Test hiệu năng
* Cơ sở dữ liệu đang có 5 triệu record
* Tiến hành chạy song song 50 Task:
  + 25 Task Insert, mỗi task insert 500 record khác nhau.
  + 25 Task còn lại mỗi task Get 500 record, sau đó Delete (có thể trùng nhau).
* Khi chưa chia partition, khoá chính là trường Id (clustered index duy nhất), các trường khác không có index. Thời gian thực thi trung bình mỗi Task là:
  + - Insert: 160 ms
    - Get: 170 ms ( theo Id)
    - Delete: 220 ms
* Chia 5 partition, clustered index sau khi tạo lại gồm Id (duy trì tính duy nhất) và Author (partition key).
  + - Insert: 150 ms
    - Get: 150 ms (theo Id)
    - Delete: 230 ms
* Nhận xét:
  + Khi tiến hành chạy song song các chỉ các câu câu lệnh Insert, Update, Select, sẽ có trường hợp deadlock xảy ra (Không xảy ra nếu chỉ chạy xong xong Select).
  + Đối với các trường tạo index, biểu thức điều kiện nếu phức tạp sẽ không sử dụng được index.
  + Số lượng task Insert, Update, Delete (sẽ lock bảng khi thực thi trên cơ sở dữ liệu) trên cùng một nơi được giảm xuống, cải thiện tốc độ thực thi.
  + Nếu mệnh đề truy vấn không sử dụng đến partition key thì việc chia vùng không có tác dụng gì trong câu truy vấn, thậm chí có thể làm chậm nếu dữ liệu cần trả về nằm ở các vùng khác nhau.
  + Việc chia bảng không giúp cải thiện tốc độ truy vấn trên các trường không được đánh index hoặc đã được đánh index từ trước.
* Giải pháp cải tiến:
  + Mỗi ứng dụng sử dụng duy nhất một Session, mỗi thao tác nghiệp vụ đầy đủ chỉ diễn ra trong một transaction.
  + Đối với câu lệnh Select, thêm hint “With (Nolock)” để có thể truy vấn kể cả khi cơ sở dữ liệu bị lock (đổi lại có thể dữ liệu trả về sẽ không chính xác).
  + Đối với các thao tác Insert, Update, Delete nhiều phần tử một lúc (mảng), ta chỉ nên commit một lần trước khi kết thúc transaction thay vì commit đối với một phần tử. Việc này giúp hạn chế tình trạng cở sở dữ liệu bị lock nhiều lần và kéo dài.
  + Hạn chế dùng các hàm, các tính toán phức tạp trong điều kiện truy vấn nếu không sẽ không thể sử dụng chức năng tìm kiếm theo index.
  + Chỉ chia vùng khi dữ liệu lưu trữ quá lớn, cần chia ra để bảo quản hoặc việc truy vấn thường xuyên xảy ra ở một vùng nhỏ trong cơ sở dữ liệu.
  + Đưa vào điều kiện truy vấn trường làm partition key.
  + Tuỳ theo nghiệp vụ, chọn trường có tính phân loại nhất làm partition key.
  + Các trường được tạo index phải là các trường thường xuyên được sử dụng để truy vấn và không nên chứa các các giá trị giống nhau.
  1. Khác
* Partition table
  + Lợi ích:
    - Việc sao lưu, phục hồi, cập nhật chỉ thực hiện ở mức vùng chứ không phải trên toàn bộ bảng giúp giảm đáng kể thời gian thực thi.
    - Truy vấn ở vùng nào thì chỉ truy cập vào vùng đó thay vì toàn bộ bảng.
    - Giảm tranh chấp giữa các câu lệnh khi chạy bất đồng bộ.
    - Các đoạn dữ liệu có thể được lưu trữ ở các ổ cứng khác nhau.
    - Chèn hoặc xoá một lượng lớn record vào database nhanh (sử dụng switch in – switch out).
  + Hạn chế:
    - Nếu mệnh đề truy vấn không sử dụng đến trường là partition key thì việc chia vùng không có tác dùng gì.
    - Nếu các record ta cần truy vấn thuộc nhiều vùng khác nhau sẽ khiến cho thời gian thực thi bị chậm.
  + Giải pháp:
    - Chỉ chia vùng khi thật sự cần thiết. Tuỳ theo nghiệp vụ, xác định cột mang tính phân loại nhất để chọn làm partition key.
    - Tạo index từ nhiều cột thường được query thay vì mỗi cột một index.
    - Khi query, tận dụng tối đa khả năng tìm kiếm theo index của SQL.
    - Chú ý thứ tự trong mệnh đề where để tránh mấy index của cột khi tìm kiếm.
    - Đưa vào mệnh đề where thuộc tính là partition key.
* Index
  + Clustered Index:
    - Dữ liệu của bảng sẽ được lưu trữ vật lý theo thứ tự sắp xếp dựa vào trường index.
    - Bảng trở thành một cây index (thường là B-cây).
    - Trong bảng chỉ có duy nhất một clustered index (record chỉ có thể lưu trữ vật lý theo một thứ tự duy nhất).
    - Việc truy vấn sẽ nhanh hơn nếu sử dụng trường làm clustered index (thường là khoá chính) trong mệnh đề truy vấn.
    - Trong nhiều trường hợp, việc Insert các record có thể chậm hơn do phải sắp xếp lại thứ tự lưu trữ dữ liệu.
  + Nonclustered Index:
    - Bao gồm một index key chứa con trở tới vị trí record tương ứng của key.
    - Được lưu trữ tách biệt với bảng cơ sở.
    - Một bảng có thể có nhiều nonclustered index.
    - Việc truy vấn dữ liệu dựa trên trường là nonclustered index sẽ nhanh hơn (nhưng vẫn thua clustered index).
    - Thao tác Insert và update nhanh hơn so với Clustered index vì không phải sắp xếp lại thứ tự lưu trữ vật lý.