**Báo cáo thực hành NHibernate**

Mục lục

[1) Thực hiện Insert: 2](#_Toc15983221)

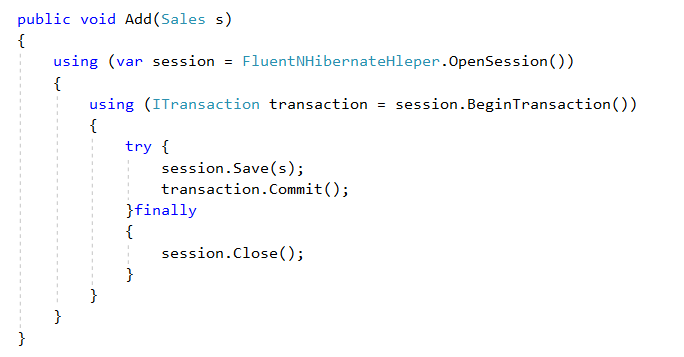
[2) Tối ưu các câu truy vấn 3](#_Toc15983222)

[3) Thực hiện mutil task 4](#_Toc15983223)

[4) Hiệu năng 4](#_Toc15983224)

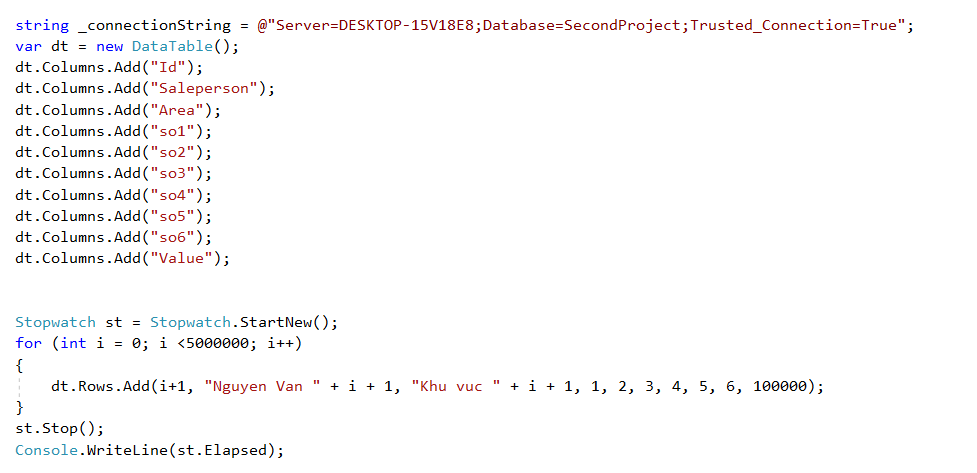
# Thực hiện Insert:

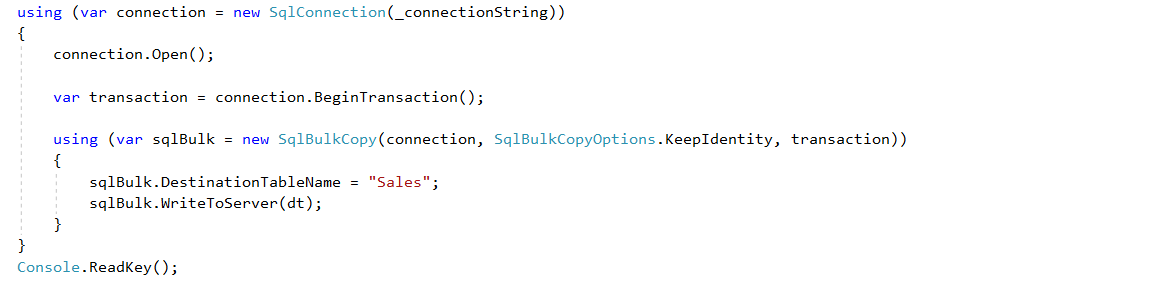
* Sales (Id, Saleperson, Area, so1, so2, so3, so4, so5, so6, Value).
* Id là primary key, set indentity
* **Sử dụng ISession insert 5tr record:**

****

**1**) Sủ dụng ISession

* Thời gian: 00:48:47.9077952
* Nhận xét: Trong cơ chế ISession thực hiện sẽ quản lý cache lv1 và tương tác với cache lv2, cùng với đó phương thức dirty checking được tự động thực hiện. Có nghĩa là khi thêm hoặc thực hiện một số thay đổi trong DB nó sẽ tự động kiểm tra các record update và update lại DB cho phù hợp. Điều này sẽ làm tiêu thụ bộ nhớ và làm cho hiệu suất giảm dần theo thời gian vì nếu thực hiện các thay đổi ngày một nhiều thì số lượng enity mà phương thức này tự động theo dõi sẽ tăng lên.
* **Batch\_size**: Là kích thước Lô thực hiện việc kiểm soát số lượng chèn để đẩy vào CSDL (Ví dụ: batch\_size=100 hoặc một số khác tuỳ vào kích thước của đối tượng, có nghĩ là mỗi lần sẽ thực hiện 50000 chuyến đến DB thay vì 5tr chuyến như ban đầu, mỗi chuyến đến nó sẽ chèn 100 record vào DB). Vì vậy nếu sử dụng thêm thuộc tính Batch\_size sẽ góp phần tăng hiệu suất của chương trình.
* **Sử dụng IStateLessSession insert 5tr record:**
* Thời gian:
* Nhận xét:
* Trong cơ chế IStateLessSession: Cơ chế này không duy trì cache lv 1, không tương tác với cache lv 2, và cũng không thực hiện phương thức dirty checking như ISession. Nói chung là nó chả làm gì cả. Các state session được sử dụng khi cần tải nhiều dữ liệu và thực hiện một số thao tác theo batch. Vì thế, nó tiêu thụ ít bộ nhớ hơn và xử lý sẽ nhanh hơn ISession.
* Vì nó không thực hiện lưu bộ nhớ nên mỗi lần thực hiện insert tất cả các dữ liệu cũ sẽ mất đi.
* **Sử dụng SqlBulkCopy insert 5tr record:**
* Thời gian: 00:00:21.6946393.

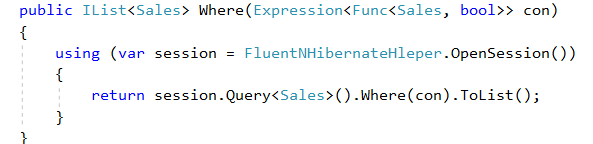




* Đầu tiên, tạo 1 DataTable giống như cấu trúc của bảng trong CSDL. Sau đó tạo vòng lặp ghi vào DataTable này 5tr record.
* Tiếp theo, tạo 1 sqlBulkCopy và sử dụng nó để ghi nội dung của DataTable vào bảng Sales trong CSDL.
* Cơ chế như sau: Tạo kết nối->Mở kết nối->Tạo giao dịch->Chuyển kết nối và các đối tượng giao dịch đến SqlBulkCopy.
* Nên sử dụng lệnh sqlBulk.BatchSize vì khi số lượng record trong nguồn tăng lên, bộ nhớ sử dụng bởi sqlBulkCopy cũng tăng lên do đó nếu sử dụng lệnh này thì có thể giói hạn lại bộ nhớ giảm thiểu sô lượng record ghi được trong mỗi batch do đó sẽ tiêu thụ bộ nhớ ít hơn. Tuỳ thuộc vào hoàn cảnh mà sử dụng các BatchSize khác nhau. Máy hiện đang sử dụng 5000.

# Tối ưu các câu truy vấn

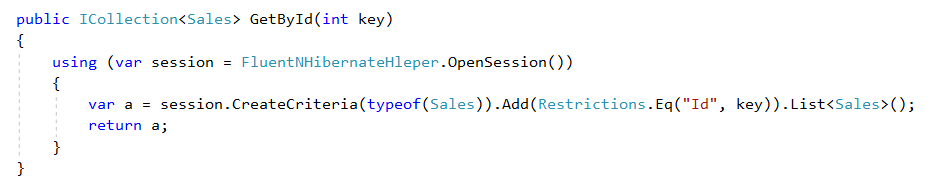
* Việc truy vấn bằng các câu lệnh Insert, Delete, Update sử dụng các phương thức của session interface trong nhibernate. Get dữ liệu thực hiện theo 2 hàm:
* 1 hàm sử dụng query với biểu thức lmbda:

****

* Thời gian truy vấn:

****

* 1 hàm sử dụng Criteria:

****

* Thời gian truy vấn:

****

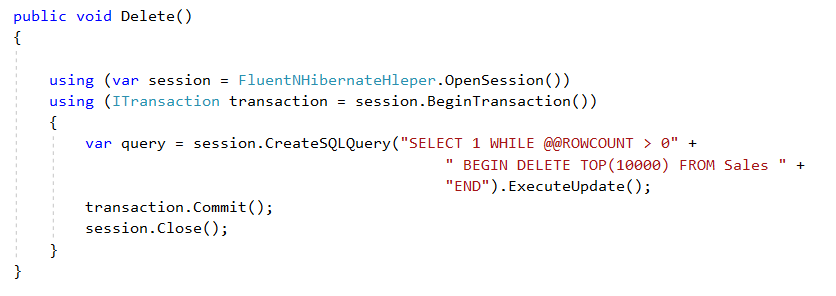
* Nhận xét:
* Cơ chế của các phương thức do session interface là openSession->Thực thi câu lệnh ->commit transaction->Đóng session lại. Tuy nhiên, nếu ta lỡ Insert 1 dòng đã có rồi hoặc Update, Delete những dòng không có nó sẽ ra 1 Exception: trả về 1 dòng nhưng thực tế lại có 0 dòng trả về. Vì vậy ta thực hiện 1 câu lệnh try catch những trường hợp exception như thế ta cần rollback transaction lại.
* Nhìn chung: Đối với lệnh Get khi sử dụng criteria sẽ nhanh hơn khi sử dụng query. Cơ chế criteria được được hibernate cung cấp cho phép xây dựng 1 đối tượng truy vẫn tiêu chuẩn theo yêu cầu của chương trình vì thế sẽ nhanh hơn khi thực hiện query.

# Thực hiện mutil task

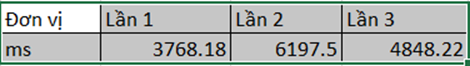
* Thực hiện Get theo Id liên tục trong 100 Task. Id random từ 1->10tr
* Nhận xét: Các task thực hiện đan xen lẫn nhau khi task này chưa hoàn thành thì task kia đã bắt đầu dẫn đến việc khó kiểm soát dữ liệu. Tuy nhiên nếu chỉ Get Id theo 100 Task thì nó không ảnh hưởng gì đến việc truy xuất dữ liệu. Vì các câu lệnh Get này tương ứng với Select trong database nó sẽ thực hiện chế độ share lock cho phép đồng thời đọc cùng 1 dữ liệu.

# Hiệu năng

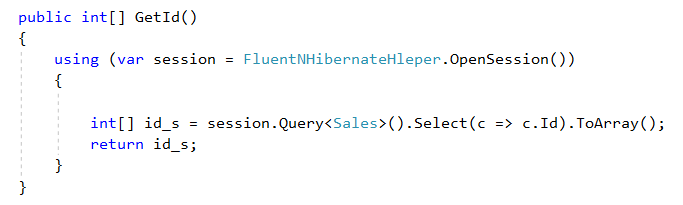
* Giữ mức ổn định 5tr record sử dụng câu lệnh T-SQL để xoá tất cả 5tr record cùng lúc:

****

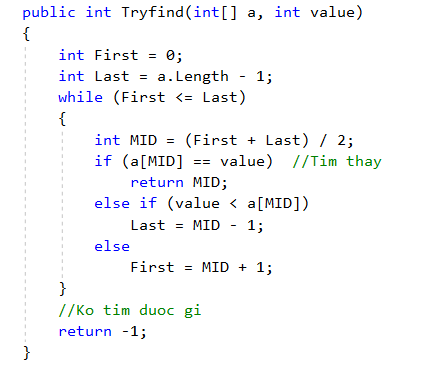
* Thời gian truy vấn là: 
* Tạo 50 task Get random theo Id tất cả các trường, Get các record%2==0 và các record%4==0, Update các record%4==0 còn lại thì Insert.
* Sử dụng hàm GetById.
* Các lệnh insert, delete, update sử dụng các phương thức trong session.
* Thời gian thực hiện:

****

* Nhận xét: Quá trình thực hiện đan xen lẫn nhau diễn ra bình thường, không có lỗi xảy ra vì trước đó đã thực hiện việc rollback dữ liệu.
* 1 số giải pháp đã thử (ngoài việc rollback dữ liệu):
* Sử dụng lock: Đảm bảo rằng các trường cân bằng không thể cập nhật đồng thời giữa 2 luồng. Khi cố gắng gọi các phương thức. Giúp tránh deadlock bằng cách chỉ cho lần lượt từng task vào. Lock yêu cầu truyền vào 1 đối tượng object.
* Cơ chế: Khi khoá được giữ, luồng giữ khoá có thể lấy lại và giải phóng khoá. Bất kỳ 1 luông nào khác đều bị chặn không lấy khoá và đợi cho đến khi khoá được giải phóng
* Nhận xét: Giúp tránh được deadlock. Tuy nhiên, việc thực hiện tuần tự sẽ làm giảm hiệu năng của chương trình. Vì vậy, chỉ thực sự sử dụng khi nghi ngờ rằng có deadlock xảy ra.
* Xây dựng hàm kiểm tra tồn tại của Id trong CSDL:

****

**2**) Hàm lấy danh sách Id trong CSDL

****

**3**) Giải thuật tìm kiếm nhị phân.

* Thực hiện: Trước hết xây dựng 1 hàm lấy danh sách Id trong CSDL sau đó tiến hành thực hiện việc tìm kiếm theo Id đã lấy nếu tìm thấy thành công thì thực hiện việc Get dữ liệu, thêm, sửa, xoá đã nêu trên nếu không thì cho tiếp tục tìm kiếm.
* Nhận xét: Việc thực hiện này cũng có khả năng giải quyết được vấn đề. Tuy nhiên, sẽ tốn rất nhiều thời gian đối với dữ liệu có hàng triệu records.
* Sử dụng HQL.

**Tối ưu hoá database với Index**

* Index là việc cấu trúc dữ liệu, lưu trữ theo 1 cơ chế nào đó để tìm ra các record 1 cách nhanh chóng.
* Sử dụng đối với dữ liệu chưa có primary key, khi table đã có primary key rồi thì trong bảng phần index sẽ tự tạo Clusterd.

a) Clustered Index

Clustered index: thường được tự tạo ra khi bảng có primary key do primary key đã duy trì độ duy nhất dữ liệu của cột, nên có thể nói clustered index chính là unique index. Trong clustered index, các dữ liệu ở cấu trúc bảng trong được sắp xếp một cách vật lý, tức là trong clustered index, dữ liệu bảng trong được sắp xếp đúng theo thư mục cây dựa vào bảng chữ cái Trong một bảng CHỈ được có duy nhất một clustered index

b) Non-Clustered Index

khác với clustered Index, non-Clustered Index không sắp xếp dữ liệu theo một trật tự vật lý như clustered mà là "loạn xà ngầu" trong bảng thông tin, miễn sao nó nằm trong một logic do index qui định. trong một bảng có thể chứa đến 249 non-clustered index. còn cách hoạt động thì tương tự clustered index, có khác là khi truy xuất đến bảng thông tin cuối thì thông tin không được sắp xếp theo trật tự

**Cơ chế:** index trong SQL server được tạo thành từ 1 tập hợp các index node chúng được tổ chức theo 1 cấu trúc gọi là B-tree.

* Khi 1 truy vấn được xây dựng dựa trên các cột tạo index, việc thực thi truy vấn sẽ bắt đầu tại nút gốc và điều hiếu qua các nút trung gian cho đến khi tìm được nút lá

**Ưu điểm:**

* Index đóng vai trò tăng tốc truy xuất dữ liệu.

**Nhược điểm:**

* Index là việc tạo 1 câu trúc bảng trong nên việc này sẽ gây tốn tài nguyên.
* Làm chậm tốc độ insert, delete, update vì Index là sắp sếp dữ liệu nên nếu có sự thay đổi thì index cũng thay đổi theo