**Báo cáo thực hành NHibernate**

Mục lục

[1) Thực hiện Insert: 2](#_Toc16320595)

[a) CSDL: 2](#_Toc16320596)

[b) Sử dụng ISession: insert 5tr record 3](#_Toc16320597)

[c) Sử dụng ISession: insert 5tr record 3](#_Toc16320598)

[d) Sử dụng SQL BulkCopy 4](#_Toc16320599)

[2) Tối ưu các câu truy vấn 5](#_Toc16320600)

[a) Insert, delete, update 5](#_Toc16320601)

[b) Get dữ liệu: 5](#_Toc16320602)

[3) Thực hiện mutil task 6](#_Toc16320603)

[4) Test hiệu năng 7](#_Toc16320604)

[a) Sử dụng SQL: 7](#_Toc16320605)

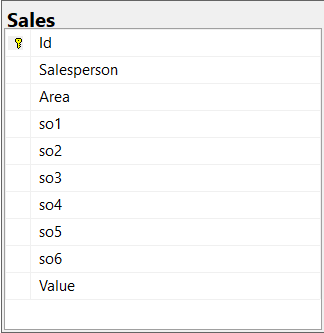
[b) Sử dụng StatelessSession: 7](#_Toc16320606)

[c) Thực hiện Mutil task 8](#_Toc16320607)

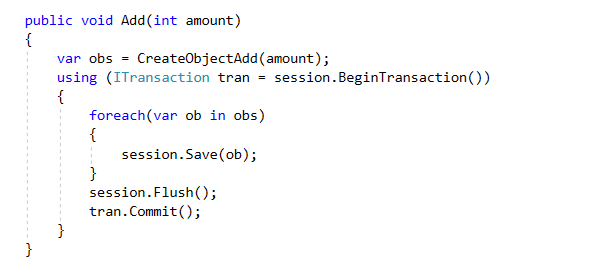
[d) 1 số giải pháp đã thử 8](#_Toc16320608)

# Thực hiện Insert:

## **CSDL**:



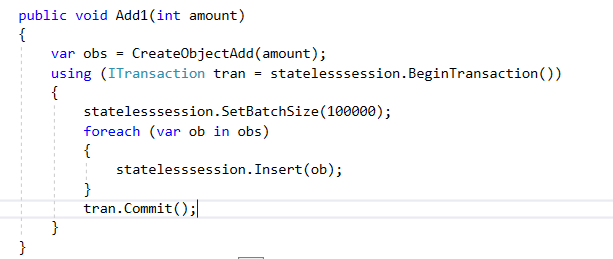
## **Sử dụng ISession:** insert 5tr record



**1) Sủ dụng ISession**

* Thời gian: 00:03:15.6623342
* 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.

## **Sử dụng IStatelessSession:** insert 5tr record

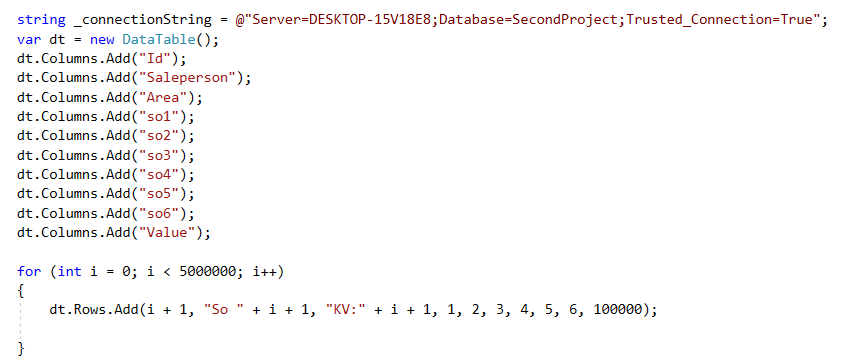


**2) Thêm dữ liệu thông qua phương thức Insert**

* Thời gian: 00:02:32.5964607
* 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.

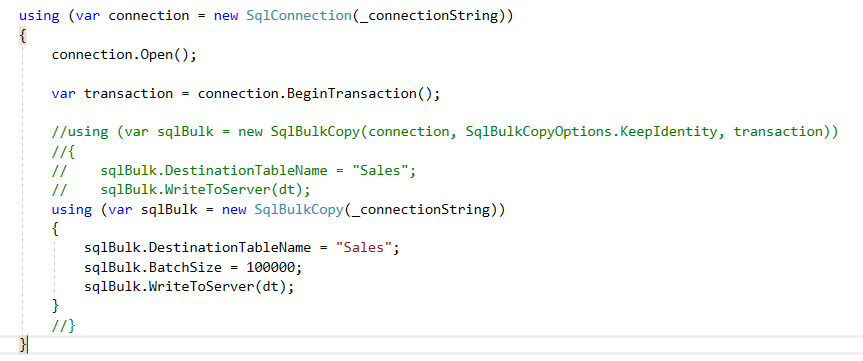
## Sử dụng SQL BulkCopy:

* Thời gian: 00:00:22.2164573
* Nhận xét:
* Cơ chế của sql BulbCopy: Đầu tiên tạo 1 DataTable phải chắc rằng Column của DataTable phải giống với bảng đích.



**3) Tạo chuỗi kết nối và DataTable**

* Tiến hành kết nối->Mở kết nối->Tạo 1 biến-> sau đó truyền 1 chuỗi kết nối vào SQL bulkcopy



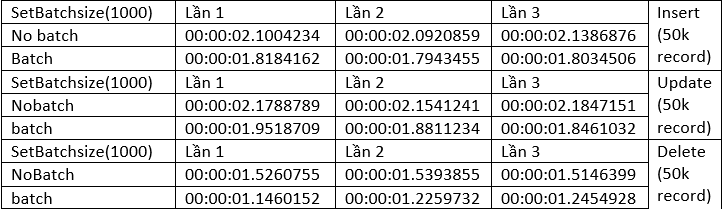
**4) Kết nối sử dụng Sql bulkCopy**

* Theo mặc định, tất cả các record trong nguồn sẽ được ghi vào bảng đích trong 1 đợt. Tức nghĩa là khi bản ghi trong nguồn tăng lên, bộ nhớ sử dụng bởi sqlBulkCopy sẽ tăng lên. Sử dụng BatchSize để giảm số lượng record trong mỗi batch và sẽ tiêu thụ ít bộ nhớ hơn
* Việc sử dụng SqlBulkCopyOntions.KeepIdentity phải chắc chắn rằng chúng ta chưa set khoá. Nếu sử dụng mà đã set primary key rồi sẽ gây ra 1 exception: 'Violation of PRIMARY KEY constraint 'PK\_\_Sales\_\_3214EC07847E9CA5'. Cannot insert duplicate key in object 'dbo.Sales'. The duplicate key value is (1). The statement has been terminated. Nó thông báo đã có key rồi. Việc ta cần làm là bỏ key trong bảng hoặc là bỏ câu lệnh trên.

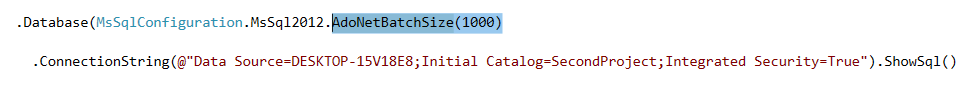
# Tối ưu các câu truy vấn

## Insert, delete, update

* **Thời gian truy vấn:**

****

* **Nhận xét:** Khi setBatchSize thời gian truy vấn sẽ nhanh hơn
* **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.
* Config Batch\_size:

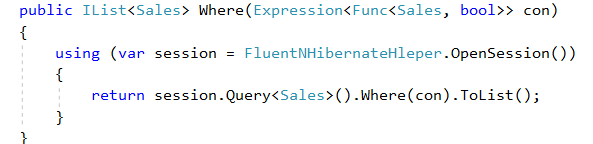


**5) Thêm batch\_size trong class FluentNHibernatehelper**

* + Batch\_size không hổ trợ mutilcriteria hoặc mutilquery của nhibernate vì vậy lệnh này không được áp dụng cho việc Get dữ liệu.
* **Exception**: NHibernate.StaleStateException: 'Batch update returned unexpected row count from update; actual row count: 0; expected: 1'. Xử lý: Lỗi này xảy ra khi update hoặc delete nhiều dòng do số dòng trả ra là 0 nhưng đáng lẽ ra là 1. Khi hệ thống update hoặc delete cùng 1 Id hoặc Id không tồn tại, để giải quyết vấn đề này ta tạo 1 try catch và các trường hợp exception tiến hành roll back lại.

## Get dữ liệu:

* Hàm sử dụng query với biểu thức lmbda:

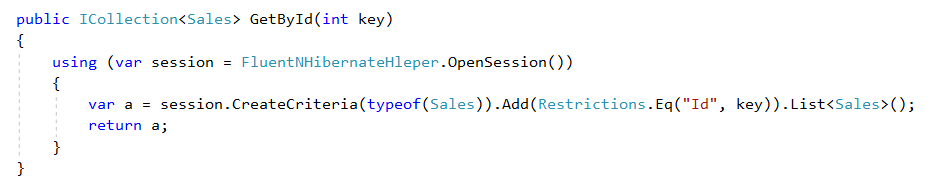
****

**6) Linq Query**

* Thời gian truy vấn:

****

* Hàm sử dụng Criteria:

****

**7) Criterial**

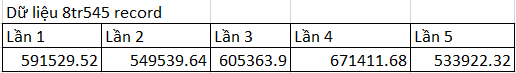
* Thời gian truy vấn:

****

* Nhận xét:
* Nhìn chung: Đối với Get dữ liệu khi sử dụng Criteria sẽ nhanh hơn khi sử dụng query.
* Giải thích: Criteria là một API nhằm đơn giản hoá việc lấy về các entities nhờ việc thêm các tiêu chí tìm kiếm vào đối tượng object. Criteria sẽ thích hợp cho các câu lệnh truy vấn tìm kiếm hơn. Với lại, Linq Query do thư viện Linq hỗ trợ, còn Criterial do thư viện nhibernate hỗ trợ.
* **?** Em chỉ biết là Cái Criteria nó được hỗ trợ bởi Nhibernate, cái query được hỗ trợ bởi Linq nhưng khi đo thời gian thì thấy là cái criteria nó nhanh hơn bên linq query, vào trong SQL Profilter thì khi insert cùng 1 Id á nó chạy giống nhau cái StarTime và EndTime của mỗi cái trừ ra chênh lệch thời gian cũng giống nhau luôn. Vậy do cơ chế nào mà criteria nhanh hơn vậy ạ?

# Thực hiện mutil task

* Thực hiện Get theo Id liên tục trong 100 Task. Id random từ 1->10tr
* Thời gian TB:

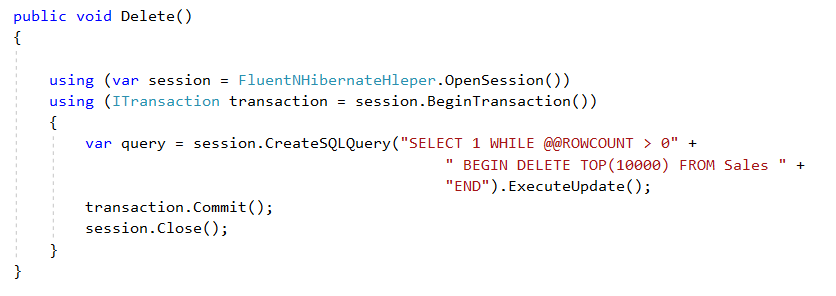
****

* Nhận xét: Các task thực hiện đan xen lẫn nhau khi task này chưa hoàn thành thì task kia đã bắt đầu. Việc thực hiện mutil task cho quá trình xử lý nhanh sau khi gọi get dữ liệu thì task không đợi get xong mà ngay lập tức sẽ quay lại phương thức main và bắt đầu task tiếp theo. Điều này sẽ nhanh hơn rất nhiều so với tuần tự.

**?** Khi e cho 2 task mỗi task chạy 1 tác vụ khác nhau. Task 1 Add 5 lần, task 2 Get 1 lần. Khi mà chạy cùng lúc thì nó cũng chen vào nhưng mà Task 1 lại Add có 2 lần còn Task 2 thì get tận 2,3 lần. Khi nó chen vào thì nó chạy tác vụ của nó luôn ha sao anh. Có cách nào cho nó thực hiện đầy đủ các tác vụ không?

# Test hiệu năng

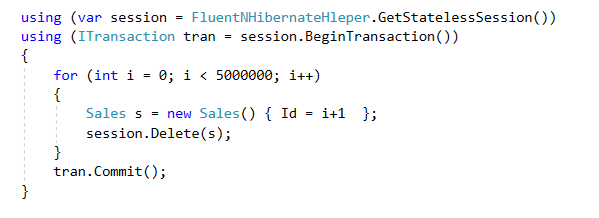
## **Sử dụng SQL:** Xoá 5tr record

****

**8) Lệnh xoá sử dụng SQL**

* Thời gian truy vấn là: 
* Nhận xét: Đây là phương thức do nhibernate hỗ trợ truy vấn SQL, cho nên nó trực tiếp truy vấn bằng câu lệnh trong SQL, dựa theo vòng lặp mỗi lần sẽ xoá 10000 record nên thời gian thực hiện rất nhanh.

## **Sử dụng StatelessSession:** xoá 5tr record:



**9) Delete sử dụng StatelessSession**

* Thời gian: 00:01:08.2726137
* Nhận xét: Tương tự như Insert, StatelessSession không làm gì cả mà giải phóng các State session để thực hiện load dữ liệu lớn nên thời gian và hiệu suất sẽ được tăng lên.

## Thực hiện Mutil task

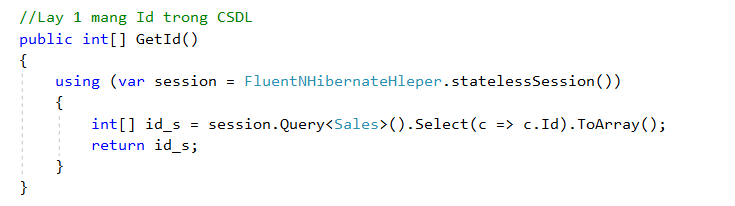
* Tạo 100 task Get random theo Id tất cả các trường, Delete &Update các record%2==0, Get các record%4==0, còn lại Insert.
* **Thời gian**:

****

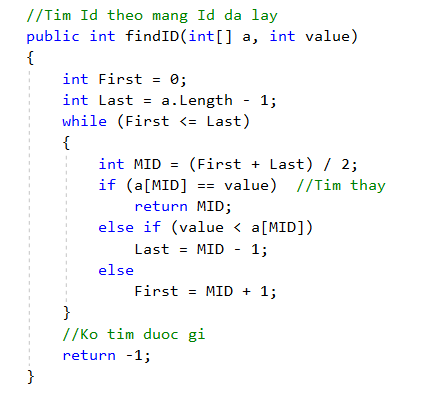
* **Exception**: SqlException: Violation of PRIMARY KEY constraint 'PK\_\_Sales\_\_3214EC07458D0011'. Cannot insert duplicate key in object 'dbo.Sales'. The duplicate key value is (5010004).The statement has been terminated. Khi random ngẫu nhiên nó bị trùng 1 Id đã có rồi trong CSDL nên khi thêm sẽ báo lỗi. Để giải quyết ta tiến hành thực hiện try catch rollback lại hàm Add, khi gặp Id trùng transaction sẽ tiến hành rollback.
* **Nhận xét**: Thực hiện mutil task cho quá trình xử lý nhanh, cải thiện hiệu năng hơn khi thực hiện tuần tự.

## **1 số giải pháp đã thử** (ngoài việc rollback dữ liệu):

* Xây dựng hàm kiểm tra tồn tại của Id trong CSDL:

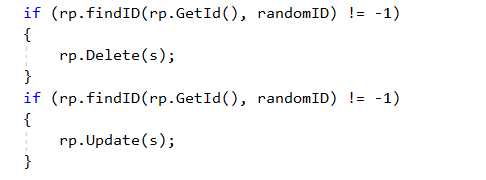


**10) Hàm lấy 1 mảng Id trong CSDL**



**11) Giải thuật tìm kiếm nhị phân.**

* **Ý đồ**: Trước hết xây dựng 1 hàm lấy mảng 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 nếu không thì cho tiếp tục tìm kiếm.



**12) Sử dụng hàm kiểm tra Id cho Delete và Update**

* **Thời gian thực hiện**: Khi visual chạy đến 11:05 minus thì bị exception.
* **Exception:** OutOfMemoryException: Exception of type 'System.OutOfMemoryException' was thrown. Thường xảy ra khi thời gian chạy ngôn ngữ chung không thể phân bổ đủ bộ nhớ cần thiết để thực hiện thao tác hiện tại.
* **Nhận xét**: Đối với phương pháp này, khi mà muốn tìm kiếm một Id nó sẽ duyệt tất cả 5tr record. Sau đó, dùng giải thuật tìm kiếm nhị phân phân bổ 1 nửa mảng ra để tìm kiếm. Điều đáng nói là mỗi lần tìm kiếm nó lại duyệt hết 5tr record. Như vậy, sẽ tốn rất nhiều thời gian thậm chí không đủ bộ nhớ để thực hiện. Phương pháp này không khả thi.

**?** Ở SQL e có xem 1 cái tối ưu trong DB sủ dụng index chỉ khi không có Primary key nó mới thực hiện việc tra cứu bằng cách scan cả table, khi có Primary key nó sẽ tạo 1 Clustered index cho cái PK đó. Vậy khi Table có PK rồi việc sử dụng nó là không cần thiết à a?