NHibernate ISession Flush: Where and when to use it, and why?

Asked 16 years, 3 months ago Modified 9 years, 1 month ago

Viewed 85k times

190

One of the things that get me thoroughly confused is the use of `session.Flush`,in conjunction with `session.Commit`, and `session.Close`.

Sometimes `session.Close` works, e.g., it commits all the changes that I need. I know I need to use commit when I have a transaction, or a unit of work with several creates/updates/deletes, so that I can choose to rollback if an error occurs.

But sometimes I really get stymied by the logic behind `session.Flush`. I have seen examples where you have a `session.SaveOrUpdate()` followed by a flush, but when I remove Flush it works fine anyway. Sometimes I run into errors on the Flush statement saying that the session timed out, and removing it made sure that I didn't run into that error.

Does anyone have a good guideline as to where or when to use a Flush? I've checked out the NHibernate documentation for this, but I still can't find a straightforward answer.

.net session nhibernate flush

Share

Improve this question

Follow

edited Sep 10, 2015 at 8:39

geothachankary

1,082 ● 1 ● 18 ● 31

asked Sep 4, 2008 at 8:04

Jon Limjap

95.3k ● 15 ● 103 ● 153

4 Answers

Sorted by: Highest score (default)

240

Briefly:

1. Always use transactions
2. Don't use `Close()`, instead wrap your calls on an `ISession` inside a `using` statement or **manage the lifecycle of your ISession somewhere else**.

From the documentation:

> From time to time the `ISession` will execute the SQL statements needed to synchronize the ADO.NET connection's state with the state of objects held in memory. This process, flush, occurs by default at the following points
>
> - from some invocations of `Find()` or `Enumerable()`

> - from `NHibernate.ITransaction.Commit()`
> - from `ISession.Flush()`
>
> The SQL statements are issued in the following order
>
> 1. all entity insertions, in the same order the corresponding objects were saved using `ISession.Save()`
> 2. all entity updates
> 3. all collection deletions
> 4. all collection element deletions, updates and insertions
> 5. all collection insertions
> 6. all entity deletions, in the same order the corresponding objects were deleted using `ISession.Delete()`
>
> (An exception is that objects using native ID generation are inserted when they are saved.)
>
> **Except when you explicity `Flush()`, there are absolutely no guarantees about when the Session executes the ADO.NET calls, only the order in which they are executed**. However, NHibernate does guarantee that the `ISession.Find(..)` methods will never return stale data; nor will they return the wrong data.

> It is possible to change the default behavior so that flush occurs less frequently. The `FlushMode` class defines three different modes: only flush at commit time (and only when the NHibernate `ITransaction` API is used), flush automatically using the explained routine, or never flush unless `Flush()` is called explicitly. The last mode is useful for long running units of work, where an `ISession` is kept open and disconnected for a long time.

...

Also refer to this section:

> Ending a session involves four distinct phases:
>
> - flush the session
> - commit the transaction
> - close the session
> - handle exceptions
>
> ## Flushing the Session
>
> If you happen to be using the `ITransaction` API, you don't need to worry about this step. It will be performed implicitly when the transaction is committed. Otherwise you should call

`ISession.Flush()` to ensure that all changes are synchronized with the database.

Committing the database transaction

If you are using the NHibernate ITransaction API, this looks like:

```
tx.Commit(); // flush the session and commit the t
```

If you are managing ADO.NET transactions yourself you should manually `Commit()` the ADO.NET transaction.

```
sess.Flush();
currentTransaction.Commit();
```

If you decide not to commit your changes:

```
tx.Rollback();  // rollback the transaction
```

or:

```
currentTransaction.Rollback();
```

If you rollback the transaction you should immediately close and discard the current

session to ensure that NHibernate's internal state is consistent.

Closing the ISession

A call to `ISession.Close()` marks the end of a session. The main implication of Close() is that the ADO.NET connection will be relinquished by the session.

```
tx.Commit();
sess.Close();

sess.Flush();
currentTransaction.Commit();
sess.Close();
```

If you provided your own connection, `Close()` returns a reference to it, so you can manually close it or return it to the pool. Otherwise `Close()` returns it to the pool.

Share Improve this answer

Follow

edited Jun 20, 2020 at 9:12

Community Bot

1 ● 1

answered Sep 4, 2008 at 11:58

Matt Hinze

13.7k ● 3 ● 36 ● 40

3 for me, this line was key: "The main implication of Close() is that the ADO.NET connection will be relinquished by the

session." if you don't call ISession.Close(), your connections get filled up until you get db timeouts. :o – Dave Thieben Nov 12, 2010 at 15:49

We usually: open session session.BeginTransaction() work... session.Transaction.Commit() session.BeginTransaction() work... session.Transaction.Commit() session.BeginTransaction() work.. session.Transaction.Commit() dispose session. – Agile Jedi May 9, 2013 at 14:52

Brilliant write-up and +1 and etc - however I think an edit might be required because you say at the top "Never use close" and then later "If you rollback the transaction you should immediately close and discard the current session" – csharpnumpty Apr 14, 2014 at 10:22

Can the order of the SQL statements be changed. I mean I need to perform update over an entity object and than insert because I have a constraint in the corresponding table. – bob_saginowski Apr 25, 2017 at 7:46

15

Starting in NHibernate 2.0, transactions are required for DB operations. Therefore, the `ITransaction.Commit()` call will handle any necessary flushing. If for some reason you aren't using NHibernate transactions, then there will be no auto-flushing of the session.

Share Improve this answer

Follow

answered Sep 4, 2008 at 17:17

Sean Carpenter

7,711 ● 3 ● 38 ● 39

From time to time the ISession will execute the SQL statements needed to synchronize the ADO.NET

1

connection's state with the state of objects held in memory.

And always use

```
using (var transaction = session.BeginTransaction())
{
    transaction.Commit();
}
```

after the changes are committed than this changes to save into database we use transaction.Commit();

Share Improve this answer

Follow

edited Sep 22, 2014 at 15:59

ganders

7,423 ● 18 ● 72 ● 116

answered Mar 5, 2014 at 9:25

user3364059

0

Here are two examples of my code where it would fail without session.Flush():

http://www.lucidcoding.blogspot.co.uk/2012/05/changing-type-of-entity-persistence.html

at the end of this, you can see a section of code where I set identity insert on, save the entity then flush, then set identity insert off. Without this flush it seemed to be setting identity insert on and off then saving the entity.

The use of Flush() gave me more control over what was going on.

Here is another example:

[Sending NServiceBus message inside TransactionScope]

I don't fully understand why on this one, but Flush() prevented my error from happening.

Share Improve this answer

Follow

edited May 23, 2017 at 12:25

Community Bot

1 ● 1

answered Nov 6, 2012 at 13:38

Paul T Davies

2,573 ● 2 ● 22 ● 39