SqlCommand Close and Dispose - which to call?

Asked 16 years, 3 months ago Modified 8 months ago

Viewed 162k times

192

Having read the threads Is SqlCommand.Dispose enough? and Closing and Disposing a WCF Service I am wondering for classes such as SqlConnection or one of the several classes inheriting from the Stream class does it matter if I close Dispose rather than Close?

.net

Share

Improve this question

Follow

edited Apr 16 at 12:33

Drew Noakes

310k ● 168 ● 696 ● 761

asked Sep 14, 2008 at 3:17

Richard McGuire

11.1k ● 8 ● 34 ● 34

8 Answers

Sorted by: Highest score (default)

I want to clarify this situation.

234

According to Microsoft guidelines, it's a good practice to provide `Close` method where suitable. Here is a citation from Framework design guidelines

> Consider providing method `Close()`, in addition to the `Dispose()`, if close is standard terminology in the area. When doing so, it is important that you make the `Close` implementation identical to `Dispose` ...

In most of cases `Close` and `Dispose` methods are equivalent. The **main difference** between `Close` and `Dispose` in the case of `SqlConnectionObject` is:

> An application can call `Close` more than one time. No exception is generated.
>
> If you called `Dispose` method `SqlConnection` object state will be reset. If you try to call any method on disposed `SqlConnection` object, you will receive exception.

That said:

- **If you use connection object one time, use `Dispose`. A `using` block will ensure this is called even in the event of an exception.**
- **If connection object must be reused, use `Close` method.**

Share Improve this answer

Follow

edited Dec 12, 2021 at 2:31

Charlieface

70.2k • 8 • 29 • 58

answered Sep 14, 2008 at 6:48

aku

124k • 33 • 176 • 203

6 @Chris, documentation for Close() says "It then releases the connection to the connection pool, or closes the connection if connection pooling is disabled." So Close() should be sufficient to keep the connection pool from overflowing. – David Hammond Dec 15, 2011 at 22:45

4 Does .Dispose() also release the connection back into the pool? – oscilatingcretin Jun 11, 2012 at 16:27

1 This is the best argument I have read on the subject one way or the other in a decade. Excellent point. – Michael Erickson Apr 18, 2017 at 16:31

2 So it works this way **1.** `con.Open() con.Close();` **2** `con.Open(); // reuse` **3.** `con.Dispose(); // use one time con.Open(); // error` – Shaiju T Jun 5, 2018 at 10:59

33

As usual the answer is: it depends. Different classes implement `IDisposable` in different ways, and it's up to you to do the necessary research.

As far as `SqlClient` goes, the recommended practice is to do the following:

```
using (SqlConnection conn = /* Create new instance usi
*/)
{
    conn.Open();
    using (SqlCommand command = /* Create new instance
method */)
    {
        // Do work
    }
    conn.Close(); // Optional
}
```

You **should** be calling `Dispose` (or `Close`*) on the connection! Do **not** wait for the garbage collector to clean up your connection, this will tie up connections in the pool until the next GC cycle (at least). If you call `Dispose`, it is not necessary to call `Close`, and since the `using` construct makes it so easy to handle `Dispose` correctly, there is really no reason to call `Close`.

Connections are automatically pooled, and calling `Dispose`/`Close` on the connection does not physically close the connection (under normal circumstances). Do not attempt to implement your own pooling. `SqlClient` performs cleanup on the connection when it's retrieved from the pool (like restoring the database context and connection options).

*if you are calling `Close`, make sure to do it in an exception-safe way (i.e. in a catch or finally block).

Share Improve this answer

Follow

edited Oct 11, 2017 at 12:34

Xpleria

5,813 ● 5 ● 53 ● 67

answered Sep 14, 2008 at 4:39

Brannon

26.1k ● 5 ● 41 ● 44

When you say, "it's up to you to do the necessary research", what is that research? The only way I know how to say for sure is through Reflection but that has the downside of being "illegal" in most situations. – Storm Jan 21, 2016 at 6:21

12 I wouldn't say: `conn.Close(); // Optional` It's not optional. It's redundant and unnecessary. You are disposing the object twice and this will be marked as a warning by some code analysis tools. – Metalogic Feb 8, 2017 at 18:44

1 @Metalogic I agree it's redundant an unnecessary (and ugly) to call Close with proper using usages. However, nitpicking: calling Close is *not* "disposing" (while Dispose implies Close for an SqlConnection). Compare to `using (var x = ..) { x.Dispose(); }`, in which case `x` really is "disposed twice". – user2864740 Nov 23, 2018 at 1:16

13

For `SqlConnection`, from the perspective of the connection itself, they are equivalent. According to Reflector, `Dispose()` calls `Close()` as well as doing a few additional memory-freeing operations -- mostly by setting members equal to null.

For Stream, they actually are equivalent. `Stream.Dispose()` simply calls Close().

Share Improve this answer

Follow

edited Apr 23, 2017 at 15:49

Gonzalo.-

12.7k ● 5 ● 52 ● 83

answered Sep 14, 2008 at 3:22

Curt Hagenlocher

20.9k ● 8 ● 62 ● 50

1 Are you sure? MSDN says it's inherited from `Component` which doesn't seem to do anything to try and call `Close()`. I can't see anywhere in `DBConnection` or `SqlConnection` that ties into either of those notifications. It does however have a private `DisposeMe()` that isn't referenced anywhere. – Deanna Feb 23, 2015 at 22:07

@Deanna it is overrided here: github.com/dotnet/corefx/blob/… – David Cumps Jan 16, 2019 at 10:48

@DavidCumps It appears it's changed in the 4 years since I wrote that comment. My links are no longer valid. – Deanna Jan 16, 2019 at 10:56

github.com/microsoft/referencesource/blob/master/System.Data/… , i dont see it here – Royi Namir May 13, 2020 at 7:40

11

You DO need to call Dispose()!

Dispose() is for the developer to call, the Garbage Collector calls Finalize(). If you don't call Dispose() on your objects any unmanaged resources that they used won't be disposed until the garbage collector comes around and calls finalize on them (and who knows when that will happen).

This scenario is called Non Deterministic Finalization and is a common trap for .net developers. If you're working with objects that implement IDisposable then call Dispose() on them!

http://www.ondotnet.com/pub/a/oreilly/dotnet/news/programmingCsharp_0801.html?page=last

While there may be many instances (like on SqlConnection) where you call Disposne() on some object and it simply calls Close() on it's connection or closes a file handle, **it's almost always your best bet to call Dispose()! unless you plan on reusing the object in the very near future.**

Share Improve this answer Follow

edited Sep 15, 2008 at 20:15

answered Sep 14, 2008 at 3:44

Tyler

3,200 ● 1 ● 32 ● 44

30 This comment is totally false. The garbage collector never, ever calls `Dispose`. – Stephen Cleary Jul 9, 2010 at 22:39

3 Corollary: You *should* call `Dispose()` *if* you are not using `using()` with a class that implements `IDisposable`. If the class being called implements IDisposable and you have wrapped its usage on the page within `using()`, then you can dispose with the `Dispose()` (pun intended, so shoot me). Using `Close()`, however, is recommended with

anything that explicitly utilizes `Open()`, AFAIK.
– René Kåbis Oct 1, 2016 at 18:05

I'm not sure about other DBMS, but you can NOT do both in **PostgreSql**. Once you `Close` a connection,Postgres automatically sets the connection identifier to `null`. From there on, one can not `Dispose` an sql connection identifier which is already set to `null`. – ssd Dec 19, 2019 at 12:20

This may apply to the PostgreSql implementation of IDbConnection. However, most implementations (e.g. SqlConnection) check for null/valid state inside the Dispose() method, so that state-change methods will never be called against nulled objects. – Erik Midtskogen Apr 14, 2022 at 12:59

6

This would-be quick advice became a long answer. Sorry.

As tyler pointed out in his nice answer, calling `Dispose()` is a great programming practice. This is because this method is supposed to "rally together" all the resource-freeing needed so there are no unneeded open resources. If you wrote some text to a file, for example, and failed to close the file (free the resource), it will remain open and no one else will be able to write to it until the GC comes around and does what you should have done.

Now, in some cases there will be "finalizing" methods more specific to the class you're dealing with, like `StreamWriter.Close()`, which overrides `TextWriter.Close()`. Indeed they are usually more suited to the situation: a StreamWriter's `Close()`, for example,

flushes the stream and the underlying encoder before `Dispose()`ing of the object! Cool!

However, browsing MSDN you'll find that even Microsoft is sometimes confused by the multitude of closers and disposers. In this webpage, for instance, in some examples `Close()` is called before the implicit `Dispose()` (see using statement if you don't understand why it's implicit), and in one in particular they don't bother to. Why would that be? I too was perplexed.

The reason I figured (and, I stress, this is original research and I surely might lose reputation if I'm wrong) is that **`Close()` might fail, yielding an exception whilst leaving resources open, while `Dispose()` would surely free them**. Which is why **a `Dispose()` should always safeguard a `Close()` call** (sorry for the pun).

```
MyResource r = new MyResource();

try {
  r.Write(new Whatever());

  r.Close()
finally {
  r.Dispose();
}
```

And yes, I guess Microsoft slipped on that one example. Perhaps that timestamp would never get flushed to the file.

I'm fixing my old code tomorrow.

Edit: sorry Brannon, I can't comment on your answer, but are you sure it's a good idea to call `Close()` *on a* `finally` *block? I guess an exception from that might ruin the rest of the block, which likely would contain important cleanup code.*

Reply to Brannon's: great, just don't forget to call `Close()` *when it is really needed (e.g. when dealing with streams - don't know much about SQL connections in .NET).*

Share Improve this answer Follow

edited Sep 14, 2008 at 5:10

answered Sep 14, 2008 at 4:42

André Chalella

14.1k ● 10 ● 57 ● 64

Actually, I never call Close(), I just let Dispose() and the 'using' construct *do the right thing*. If you aren't calling Dispose, then you need to be calling Close in an exception-safe manner. It might be a good idea to add exception handling to the finally block. – Brannon Sep 14, 2008 at 4:54

Right, my comments were for SqlClient specifically. The point is, you need to understand the classes you are using. Always calling Dispose isn't necessarily the right answer. – Brannon Sep 14, 2008 at 5:15

Typecast to iDisposable, and call dispose on that. That will invoke whatever method is configured as

2

implementing "iDisposable.Dispose", regardless of what the function is named.

Share Improve this answer

Follow

answered Jul 9, 2010 at 22:42

supercat

80.8k ● 9 ● 174 ● 220

The "function is named" 'Dispose': so we're back to the initial question :} – user2864740 Nov 23, 2018 at 1:21

The function is bound to `IDisposable.Dispose`, but that doesn't mean that's the name. Note that in vb.net, it's possible to have a function be bound to multiple interface members with names that need not be related to that of the function. – supercat Nov 23, 2018 at 2:52

Cast like this: `using (myObj as IDisposable)` – Yousha Aleayoub Nov 1, 2019 at 6:22

1

Generally we are facing issue in Close(), Abort() and Dispose() but let me tell you difference among them.

1) ABORT:- I won't suggest to use this because when abort is called the client will delete the connection without telling the server so server will wait for some amount of time (approximately 1 min). If you having bulk request then you can't use abort() because it may caused time out for your limited connection pool.

2) Close:- Close is very good way to closing the connection because when closing the connection it will

call server and acknowledge the server to close by that side too.

Here, one more thing to look. In some cases, if error generates then it is not a good way to write code in finally that connection.close() because at that time Communication State will be faulted.

3) Dispose :- It is one type of close but after closing the connection you can not open it again.

So try this way ,

```
private void CloseConnection(Client client)
    {
        if (client != null && client.State == Communic
        {
            client.Close();
        }
        else
        {
            client.Abort();
        }
    }
```

Share Improve this answer Follow

edited Jun 19, 2017 at 14:54

answered Jun 19, 2017 at 14:38

Deep Jadia

111 ● 1 ● 9

The check on `client != null` is incorrect/misleading because it does not guard all usages. Also, I'm not sure how

code can get to the state of "this connection is not opened and should be closed". – user2864740 Nov 23, 2018 at 1:19

0

I just had an issue on net6 app using `TransactionScope`: I had multiple consequently created, opened and disposed connections, but was catching `This platform does not support distributed transactions` on completing scope (while clearly there were no distributed transactions).

The issue was resolved by adding `Close` method call. The majority of answers say that `Dispose` and `Close` are identical, but it seems they aren't: from what I gleaned from source code, `Close` method explicitly returns connection to pool, while I could not find identical code for `Dispose`.

Share Improve this answer

Follow

answered Aug 22, 2022 at 9:01

Alexander Kozachenko

935 ● 3 ● 14 ● 29