ExecuteReader requires an open and available Connection. The connection's current state is Connecting

Asked 12 years, 9 months ago Modified 2 years, 5 months ago Viewed 229k times

139

When attempting to connect to MSSQL database via ASP.NET online, I will get the following when two or more people connect simultaneously:

> ExecuteReader requires an open and available Connection. The connection's current state is Connecting.

The site works fine on my localhost server.

This is the rough code.

```
public Promotion retrievePromotion()
{
    int promotionID = 0;
    string promotionTitle = "";
    string promotionUrl = "";
    Promotion promotion = null;
    SqlOpenConnection();
    SqlCommand sql = SqlCommandConnection();

    sql.CommandText = "SELECT TOP 1 PromotionID, PromotionTitle, PromotionURL
FROM Promotion";

    SqlDataReader dr = sql.ExecuteReader();
    while (dr.Read())
    {
        promotionID = DB2int(dr["PromotionID"]);
        promotionTitle = DB2string(dr["PromotionTitle"]);
        promotionUrl = DB2string(dr["PromotionURL"]);
        promotion = new Promotion(promotionID, promotionTitle, promotionUrl);
    }
    dr.Dispose();
    sql.Dispose();
    CloseConnection();
    return promotion;
}
```

May I know what might have gone wrong and how do I fix it?

Edit: Not to forget, my connection string and connection are both in static. I believe this is the reason. Please advise.

```
public static string conString =
ConfigurationManager.ConnectionStrings["dbConnection"].ConnectionString;
public static SqlConnection conn = null;
```

c# .net sql-server ado.net database-connection

Share
Improve this question
Follow

edited Apr 18, 2012 at 14:11
abatishchev
100k ●88 ●301 ●442

asked Mar 14, 2012 at 16:09
Guo Hong Lim
1,710 ●3 ●13 ●20

26 Don't use shared/static connections in a multithreading-environment like ASP.NET since you're generating locks or exceptions(too many open connections etc.). Throw your DB-Class to the garbage can and create,open,use,close,dispose ado.net objects where you need them. Have a look at the using-statement as well. – Tim Schmelter Mar 14, 2012 at 16:13

2 can you give me detail about SqlOpenConnection();and sql.ExecuteReader(); functions?.. – ankit rajput Mar 14, 2012 at 16:14

1 private void SqlOpenConnection() { try { conn = new SqlConnection(); conn.ConnectionString = conString; conn.Open(); } catch (SqlException ex) { throw ex; } } – Guo Hong Lim Mar 14, 2012 at 16:41

...and just to make it a known-unknown: Ensuring you also get your database transaction handling / unit-of-work correct is left as an exercise for the reader. – mwardm Mar 10, 2016 at 17:16

2 Answers

Sorted by: Highest score (default)

256

Sorry for only commenting in the first place, but i'm posting almost every day a similar comment since many people think that it would be smart to encapsulate ADO.NET functionality into a DB-Class(me too 10 years ago). Mostly they decide to use static/shared objects since it seems to be faster than to create a new object for any action.

That is neither a good idea in terms of performance nor in terms of fail-safety.

Don't poach on the Connection-Pool's territory

There's a good reason why ADO.NET internally manages the underlying Connections to the DBMS in the ADO-NET Connection-Pool:

> In practice, most applications use only one or a few different configurations for connections. This means that during application execution, many identical

> connections will be repeatedly opened and closed. To minimize the cost of opening connections, ADO.NET uses an optimization technique called connection pooling.
>
> Connection pooling reduces the number of times that new connections must be opened. The pooler maintains ownership of the physical connection. It manages connections by keeping alive a set of active connections for each given connection configuration. Whenever a user calls Open on a connection, the pooler looks for an available connection in the pool. If a pooled connection is available, it returns it to the caller instead of opening a new connection. When the application calls Close on the connection, the pooler returns it to the pooled set of active connections instead of closing it. Once the connection is returned to the pool, it is ready to be reused on the next Open call.

So obviously there's no reason to avoid creating,opening or closing connections since actually they aren't created,opened and closed at all. This is "only" a flag for the connection pool to know when a connection can be reused or not. But it's a very important flag, because if a connection is "in use"(the connection pool assumes), a new physical connection must be opened to the DBMS what is very expensive.

So you're gaining no performance improvement but the opposite. If the maximum pool size specified (100 is the default) is reached, you would even get exceptions(too many open connections ...). So this will not only impact the performance tremendously but also be a source for nasty errors and (without using Transactions) a data-dumping-area.

If you're even using static connections you're creating a lock for every thread trying to access this object. ASP.NET is a multithreading environment by nature. So there's a great chance for these locks which causes performance issues at best. Actually sooner or later you'll get many different exceptions(like your *ExecuteReader requires an open and available Connection*).

Conclusion:

- Don't reuse connections or any ADO.NET objects at all.
- Don't make them static/shared(in VB.NET)
- Always create, open(in case of Connections), use, close and dispose them where you need them(f.e. in a method)
- use the `using-statement` to dispose and close(in case of Connections) implicitly

That's true not only for Connections(although most noticeable). Every object implementing `IDisposable` should be disposed(simplest by `using-statement`), all the more in the `System.Data.SqlClient` namespace.

All the above speaks against a custom DB-Class which encapsulates and reuse all objects. That's the reason why I commented to trash it. That's only a problem source.

Edit: Here's a possible implementation of your `retrievePromotion`-method:

```
public Promotion retrievePromotion(int promotionID)
{
    Promotion promo = null;
    var connectionString = 
System.Configuration.ConfigurationManager.ConnectionStrings["MainConnStr"].Conne
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        var queryString = "SELECT PromotionID, PromotionTitle, PromotionURL 
FROM Promotion WHERE PromotionID=@PromotionID";
        using (var da = new SqlDataAdapter(queryString, connection))
        {
            // you could also use a SqlDataReader instead
            // note that a DataTable does not need to be disposed since it does 
not implement IDisposable
            var tblPromotion = new DataTable();
            // avoid SQL-Injection
            da.SelectCommand.Parameters.Add("@PromotionID", SqlDbType.Int);
            da.SelectCommand.Parameters["@PromotionID"].Value = promotionID;
            try
            {
                connection.Open(); // not necessarily needed in this case 
because DataAdapter.Fill does it otherwise 
                da.Fill(tblPromotion);
                if (tblPromotion.Rows.Count != 0)
                {
                    var promoRow = tblPromotion.Rows[0];
                    promo = new Promotion()
                    {
                        promotionID    = promotionID,
                        promotionTitle = promoRow.Field<String>
("PromotionTitle"),
                        promotionUrl   = promoRow.Field<String>("PromotionURL")
                    };
                }
            }
            catch (Exception ex)
            {
                // log this exception or throw it up the StackTrace
                // we do not need a finally-block to close the connection since 
it will be closed implicitly in an using-statement
                throw;
            }
        }
    }
    return promo;
}
```

edited Jul 8, 2022 at 8:40   answered Mar 14, 2012 at 17:34

Share

Improve this answer

Follow

Dev Leader
357 ● 3 ● 10

Tim Schmelter
460k ● 77 ● 709 ● 968

1 this is really usefull to giving connection work paradigma. Thanks for this explanation. – aminvincent Jul 31, 2018 at 3:22

well written, an explanation for something many people accidentally discover, and i wish more people knew this. (+1) – Andrew Hill Jan 10, 2019 at 23:53

1 Thank you sir, I think that this is the best explanation on this subject that I've ever read, a subject that is very important and that many newbies mistake. I must compliment you for your excellent writing ability. – Sasino Jan 22, 2020 at 20:51

@Tim Schmelter how can I make my queries running on different threads utilize a single transaction for committing/rolling back using your suggested approach? – geeko Mar 12, 2020 at 14:02

2

I caught this error a few days ago.

IN my case it was because I was using a Transaction on a Singleton.

.Net does not work well with Singleton as stated above.

My solution was this:

```
public class DbHelper : DbHelperCore
{
    public DbHelper()
    {
        Connection = null;
        Transaction = null;
    }

    public static DbHelper instance
    {
        get
        {
            if (HttpContext.Current is null)
                return new DbHelper();
            else if (HttpContext.Current.Items["dbh"] == null)
                HttpContext.Current.Items["dbh"] = new DbHelper();

            return (DbHelper)HttpContext.Current.Items["dbh"];
        }
    }

    public override void BeginTransaction()
    {
        Connection = new SqlConnection(Entity.Connection.getCon);
        if (Connection.State == System.Data.ConnectionState.Closed)
            Connection.Open();
        Transaction = Connection.BeginTransaction();
    }
}
```

I used HttpContext.Current.Items for my instance. This class DbHelper and DbHelperCore is my own class

Share Improve this answer Follow

answered Sep 29, 2019 at 0:06

Damon Abdiel

3,453 ● 2 ● 23 ● 15