Solving “The ObjectContext instance has been disposed and can no longer be used for operations that require a connection” InvalidOperationException

Ask Question

100

19

I am trying to populate a `GridView` using Entity Frameworkm but every time I am getting the following error:

> "Property accessor 'LoanProduct' on object 'COSIS_DAL.MemberLoan' threw the following exception: The ObjectContext instance has been disposed and can no longer be used for operations that require a connection."

My code is:

```
public List<MemberLoan> GetAllMembersForLoan(string keyword)
{
    using (CosisEntities db = new CosisEntities())
    {
        IQueryable<MemberLoan> query = db.MemberLoans.OrderByDescending(m => m.LoanDate);
        if (!string.IsNullOrEmpty(keyword))
        {
            keyword = keyword.ToLower();
            query = query.Where(m =>
                  m.LoanProviderCode.Contains(keyword)
                  || m.MemNo.Contains(keyword)
                  || (!string.IsNullOrEmpty(m.LoanProduct.LoanProductName) &&
m.LoanProduct.LoanProductName.ToLower().Contains(keyword))
                  || m.Membership.MemName.Contains(keyword)
                  || m.GeneralMasterInformation.Description.Contains(keyword)

                  );
        }
        return query.ToList();
    }
```

```
}


protected void btnSearch_Click(object sender, ImageClickEventArgs e)
{
    string keyword = txtKeyword.Text.ToLower();
    LoanController c = new LoanController();
    List<COSIS_DAL.MemberLoan> list = new List<COSIS_DAL.MemberLoan>();
    list = c.GetAllMembersForLoan(keyword);

    if (list.Count <= 0)
    {
        lblMsg.Text = "No Records Found";
        GridView1.DataSourceID = null;
        GridView1.DataSource = null;
        GridView1.DataBind();
    }
    else
    {
        lblMsg.Text = "";
        GridView1.DataSourceID = null;
        GridView1.DataSource = list;
        GridView1.DataBind();
    }
}
```

The error is mentioning the `LoanProductName` column of the `Gridview`. Mentioned: I am using C#, ASP.net, SQL-Server 2008 as back end DB.

I am quite new to Entity Framework. I can't understand why I am getting this error. Can anyone help me please?

c# asp.net entity-framework

edited Dec 12 '18 at 13:46

demo

2,379 3 38 83

asked Aug 23 '13 at 8:34

barsan

946 11 26 49

1 Are you accessing any navigation properties in the gridview. If you do, you need to include those navigation tables in the query as well. Like `query.Include("SomeOtherTable")` – Nilesh Aug 23 '13 at 8:39

Try either creating a proxy class to host your entity or at least return an anonymous object. From my point of view, using ef does require creating proxy classes to implement your logics, use the edmx just as the db access layer not as business. – Gonzix Aug 23 '13 at 8:40

yes in the gridview I am getting another table column also. Which is LoanProviderName. – barsan Aug 23 '13 at 8:41

1 Try `db.MemberLoans.Include("LoanProduct").OrderByDescending()` check the syntax cause I dont have VS in front of me. – Nilesh Aug 23 '13 at 8:48

3 You just need to go on including all the navigation properties that you are accessing outside the context like `db.MemberLoans.Include("LoanProduct").Include("SomeOtherTable")` . Check the answers by @Tragedian and @lazyberezovsky – Nilesh Aug 23 '13 at 8:57

7 Answers

144

By default Entity Framework uses lazy-loading for navigation properties. That's why these properties should be marked as virtual - EF creates proxy class for your entity and overrides navigation properties to allow lazy-loading. E.g. if you have this entity:

```
public class MemberLoan
{
   public string LoandProviderCode { get; set; }
   public virtual Membership Membership { get; set; }
}
```

Entity Framework will return proxy inherited from this entity and provide DbContext instance to this proxy in order to allow lazy loading of membership later:

```
public class MemberLoanProxy : MemberLoan
{
    private CosisEntities db;
    private int membershipId;
    private Membership membership;

    public override Membership Membership
    {
       get
       {
          if (membership == null)
              membership = db.Memberships.Find(membershipId);
          return membership;
       }
       set { membership = value; }
    }
}
```

So, entity has instance of DbContext which was used for loading entity. That's your problem. You have `using` block around CosisEntities usage. Which disposes context before entities are returned. When some code later tries to use lazy-loaded navigation property, it fails, because context is disposed at that moment.

To fix this behavior you can use eager loading of navigation properties which you will need later:

```
IQueryable<MemberLoan> query = db.MemberLoans.Include(m => m.Members
```

That will pre-load all memberships and lazy-loading will not be used. For details see Loading Related Entities article on MSDN.

answered Aug 23 '13 at 8:54

Sergey Berezovskiy
189k 23 321 367

Thanks a lot for your helpful explanation and answer. Actually here I am including three table so I don't know how I can add the three tables with INCLUDE. can you please help me on this please. – barsan Aug 23 '13 at 8:59

8 @barsan just include all navigation properties one by one. E.g.

`db.MemberLoans.Include(m => m.Membership).Include(m => m.LoanProduct).OrderByDescending(m => m.LoanDate);` that will generate JOIN query and return all data at once. – Sergey Berezovskiy Aug 23 '13 at 9:03

1 Thanks a lot lazyberezovsky. I am so grateful to you. You saved me almost a day. From you explanation I am learning more about Entity Framework. Thank you my friend. – barsan Aug 23 '13 at 9:07

Thanks mate, perfect. I had a using statement that was limiting the lazy loading. Great answer. – ncbl Nov 8 '14 at 22:59

@barsan An easy way to prevent accidentally Lazy Loading things is to turn it off with `Configuration.LazyLoadingEnabled = false;` in the constructor of your `DbContext`. Lazy Loading can have serious performance issues in some cases, so I usually turn it off by default. This forces you to explicitly load any navigation properties, as Sergey showed. – Bradley Uffner Jul 13 '17 at 17:12

27

The `CosisEntities` class is your `DbContext`. When you create a context in a `using` block, you're defining the boundaries for your data-oriented operation.

In your code, you're trying to emit the result of a query from a method and then end the context within the method. The operation you pass the result to then tries to access the entities in order to populate the grid view. Somewhere in the process of binding to the grid, a lazy-loaded property is being accessed and Entity Framework is trying to perform a lookup to obtain the values. It fails, because the associated context has already ended.

You have two problems:

1. You're lazy-loading entities when you bind to the grid. This means that you're doing lots of separate query operations to SQL Server, which are going to slow everything down. You can fix this issue by either making the related properties eager-

loaded by default, or asking Entity Framework to include them in the results of this query by using the `Include` extension method.

2. You're ending your context prematurely: a `DbContext` should be available throughout the unit of work being performed, only disposing it when you're done with the work at hand. In the case of ASP.NET, a unit of work is typically the HTTP request being handled.

answered Aug 23 '13 at 8:56

Paul Turner
25.6k 11 80 142

Thank you so much for the useful information and nice explanation of the problem. Actually I am so new in Entity Framework as well as in Linq so this information is really a great lesson for me to learn. – barsan Aug 23 '13 at 9:09

13

Bottom Line

Your code has retrieved data (entities) via entity-framework with lazy-loading enabled and after the DbContext has been disposed, your code is referencing properties (related/relationship/navigation entities) that was not explicitly requested.

More Specifically

The `InvalidOperationException` with this message always means the same thing: you are requesting data (entities) from entity-framework after the DbContext has been disposed.

A simple case:

(these classes will be used for all examples in this answer, and assume all navigation properties have been configured correctly and

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

have associated tables in the database)

```
public class Person
{
  public int Id { get; set; }
  public string name { get; set; }
  public int? PetId { get; set; }
  public Pet Pet { get; set; }
}

public class Pet
{
  public string name { get; set; }
}

using (var db = new dbContext())
{
  var person = db.Persons.FirstOrDefaultAsync(p => p.id == 1);
}

Console.WriteLine(person.Pet.Name);
```

The last line will throw the `InvalidOperationException` because the dbContext has not disabled lazy-loading and the code is accessing the Pet navigation property after the Context has been disposed by the using statement.

Debugging

How do you find the source of this exception? Apart from looking at the exception itself, which will be thrown exactly at the location where it occurs, the general rules of debugging in Visual Studio apply: place strategic breakpoints and inspect your variables, either by hovering the mouse over their names, opening a (Quick)Watch window or using the various debugging panels like Locals and Autos.

If you want to find out where the reference is or isn't set, right-click its name and select "Find All References". You can then place a breakpoint at every location that requests data, and run your program with the debugger attached. Every time the debugger breaks on such a breakpoint, you need to determine whether your

navigation property should have been populated or if the data requested is necessary.

Ways to Avoid

Disable Lazy-Loading

```
public class MyDbContext : DbContext
{
  public MyDbContext()
  {
    this.Configuration.LazyLoadingEnabled = false;
  }
}
```

Pros: Instead of throwing the InvalidOperationException the property will be null. Accessing properties of null or attempting to change the properties of this property will throw a NullReferenceException.

How to explicitly request the object when needed:

```
using (var db = new dbContext())
{
  var person = db.Persons
    .Include(p => p.Pet)
    .FirstOrDefaultAsync(p => p.id == 1);
}
Console.WriteLine(person.Pet.Name);  // No Exception Thrown
```

In the previous example, Entity Framework will materialize the Pet in addition to the Person. This can be advantageous because it's a single call the the database. (However, there can also be huge performance problems depending on the number of returned results and the number of navigation properties requested, in this instance, there would be no performance penalty because both instances are only a single record and a single join).

or

```
using (var db = new dbContext())
{
  var person = db.Persons.FirstOrDefaultAsync(p => p.id == 1);

  var pet = db.Pets.FirstOrDefaultAsync(p => p.id == person.PetId);
}
Console.WriteLine(person.Pet.Name);  // No Exception Thrown
```

In the previous example, Entity Framework will materialize the Pet independently of the Person by making an additional call to the database. By default, Entity Framework tracks objects it has retrieved from the database and if it finds navigation properties that match it will *auto-magically* populate these entities. In this instance because the `PetId` on the `Person` object matches the `Pet.Id`, Entity Framework will assign the `Person.Pet` to the `Pet` value retrieved, before the value is assigned to the pet variable.

I always recommend this approach as it forces programmers to understand when and how code is request data via Entity Framework. When code throws a null reference exception on a property of an entity, you can almost always be sure you have not explicitly requested that data.

edited Mar 9 '18 at 20:23

answered Oct 25 '17 at 17:01

Erik Philips

41.2k 6 92 126

5

It's a very late answer but I resolved the issue turning off the lazy loading:

```
db.Configuration.LazyLoadingEnabled = false;
```

answered May 23 '18 at 17:33

Ricardo Pontual

3,203 3 19 34

To me, StackOverflow works wonders with one liners. And this did it for me, kudos to you! – Harold_Finch Aug 14 '18 at 7:05

Downside is you have to use .Include and things like that to load navigation properties. – boylec1986 Nov 29 '18 at 16:18

1

In my case, I was passsing all models 'Users' to column and it wasn't mapped correctly, so I just passed 'Users.Name' and it fixed it.

```
var data = db.ApplicationTranceLogs
             .Include(q=>q.Users)
             .Include(q => q.LookupItems)
             .Select(q => new { Id = q.Id, FormatDate = q.Date.ToStri
***Users = q.Users,*** ProcessType = q.ProcessType, CoreProcessId = q
= q.Data })
             .ToList();

var data = db.ApplicationTranceLogs
             .Include(q=>q.Users).Include(q => q.LookupItems)
             .Select(q => new { Id = q.Id, FormatDate = q.Date.ToStri
***Users = q.Users.Name***, ProcessType = q.ProcessType, CoreProcessI
Data = q.Data })
             .ToList();
```

edited Jul 4 '17 at 13:21

Karl Gjertsen

3,120 6 27 53

answered May 17 '17 at 14:44

Michael Mora Montero

21 4

1

If you're using ASP.NET Core and wonder why you get this message in one of your async controller methods, make sure you return a `Task` rather than `void` - ASP.NET Core disposes injected contexts.

(I'm posting this answer as this question is high in the search results to that exception message and it's a subtle issue - maybe it's useful to people who Google for it.)

answered Nov 8 '18 at 11:10

John

3,168 2 30 65

1

Most of the other answers point to eager loading, but I found another solution.

In my case I had an EF object `InventoryItem` with a collection of `InvActivity` child objects.

```
class InventoryItem {
...
   // EF code first declaration of a cross table relationship
   public virtual List<InvActivity> ItemsActivity { get; set; }

   public GetLatestActivity()
   {
       return ItemActivity?.OrderByDescending(x => x.DateEntered).Sir
   }
...
}
```

And since I was pulling from the child object collection instead of a context query (with `IQueryable`), the `Include()` function was not available to implement eager loading. So instead my solution was to create a context from where I utilized `GetLatestActivity()` and `attach()` the returned object:

```
using (DBContext ctx = new DBContext())
{
    var latestAct = _item.GetLatestActivity();

    // attach the Entity object back to a usable database context
    ctx.InventoryActivity.Attach(latestAct);

    // your code that would make use of the latestAct's lazy loading
    // ie   latestAct.lazyLoadedChild.name = "foo";
}
```

Thus you aren't stuck with eager loading.

answered Jul 13 '17 at 17:04

Zorgarath

337 6 14

This is basically eager loading, you've loaded the object via a context. There is only two options; eager loading and lazy loading. – Erik Philips Apr 13 '18 at 18:50

@ErikPhilips right, it's lazy loading with a new data context – Zorgarath Apr 16 '18 at 5:08