Adding items to a collection using Entity Framework

Asked 7 years, 2 months ago Active 3 years, 5 months ago Viewed 21k times

13

6

I'm trying to follow the DDD Repository pattern with Entity Framework 4. But I'm having problems saving changes to collection properties of my aggregate roots. Consider my classes below. Item is my aggregate root which contains a collection of SubItem entities.

```
public class Item
{
    public int ItemId { get; set; }
    public string Name { get; set; }
    public ICollection<SubItem> SubItems { get; private set; }

    public Item()
    {
        this.SubItems = new HashSet<SubItem>();
    }
}

public class SubItem
{
    public int ItemId { get; set; }
    public int SubItemId { get; set; }
    public string Name { get; set; }
}
```

Next I defined a repository interface for my aggregate root class

```
public interface IItemRespository
{
    Item Get(int id);
    void Add(Item i);
    void Save(Item i);
}
```

Now here is my DbContext class that sets up the EF mapping

```
public class ItemContext : System.Data.Entity.DbContext
{
    protected override void OnModelCreating(System.Data.Entity.DbModelBuilder
modelBuilder)
    {
        modelBuilder.Entity<Item>().HasKey(i => i.ItemId);
        modelBuilder.Entity<Item>().Property(i => i.Name);

        modelBuilder.Entity<Item>().HasMany(i => i.SubItems)
            .WithRequired()
            .HasForeignKey(si => si.ItemId);

        modelBuilder.Entity<SubItem>().HasKey(i => i.SubItemId);
        modelBuilder.Entity<SubItem>().Property(i => i.Name);
    }
}
```

Finally here's my implementation of IRepository using the DBContext

```
public class Repository : IItemRespository
{
    public void Save(Item i)
    {
        using (var context = new ItemContext())
        {
            context.Set<Item>().Attach(i);
            context.SaveChanges();
        }
    }

    public Item Get(int id)
    {
        using (var context = new ItemContext())
        {
            var result = (from x in context.Set<Item>() where x.ItemId == id select
x).FirstOrDefault();
            return result;
        }
    }

    public void Add(Item i)
    {
        using (var context = new ItemContext())
```

```
        }
    }
}
```

The following code creates a new Item, adds it to the repository, adds some new SubItems, and then saves the changes.

```
IItemRespository repo = new Repository();

//Create a new Item
Item parent = new Item() { Name = "Parent" };
repo.Add(parent);

//A long period of time may pass .. . .

//later add sub items
parent.SubItems.Add(new SubItem() { Name = "Child 1" });
parent.SubItems.Add(new SubItem() { Name = "Child 2" });
parent.SubItems.Add(new SubItem() { Name = "Child 3" });

//save the added sub items
repo.Save(parent);
```

I get the following exception when the Save() method tries to attach the item to the context.

> A referential integrity constraint violation occurred: The property values that define the referential constraints are not consistent between principal and dependent objects in the relationship.

I realized I'm creating a new context for each method in the repository. This is intentional. A long period of time may pass between when an Item is added and then later edited, and I don't want to keep the context or database connection open for the entire time.

Now if I attach the newItem to the second context before adding the sub items as in the code below it works.

```
//Create a new item
Item newItem = new Item() { Name = "Parent" };

using (ItemContext context1 = new ItemContext())
{
    //Create a new aggrgate
    context1.Set<Item>().Add(newItem);
    context1.SaveChanges();
```

```
using (ItemContext context2 = new ItemContext())
{
    context2.Set<Item>().Attach(newItem);

    newItem.Name = "Edited Name";
    newItem.SubItems.Add(new SubItem() { Name = "Child 1" });
    newItem.SubItems.Add(new SubItem() { Name = "Child 2" });
    newItem.SubItems.Add(new SubItem() { Name = "Child 3" });

    context2.SaveChanges();
}
```

However, if I want to be true to the repository pattern the code that edits Item shouldn't know anything about how the repository works or the ItemContext class. It should simply be able to make changes to an aggregate root entity, and then save those changes through a repository Save() method.

So how do I modify my Save() method so that changes to Item.SubItems are saved correctly?

c# .net entity-framework-4 domain-driven-design ddd-repositories

asked Nov 8 '12 at 4:50

Eric Anastas
19.3k 31 120 215

2 Answers

13

You'll need to help EF by setting some properties to get this to work.

When you create a new subitem, you'll need to set the FK yourself:

```
parent.SubItems.Add(new SubItem() { Name = "Child 1", ItemId = parent.ItemId});
parent.SubItems.Add(new SubItem() { Name = "Child 2", ItemId = parent.ItemId });
parent.SubItems.Add(new SubItem() { Name = "Child 3", ItemId = parent.ItemId });
```

And then in vour save function. add or attach the items to vour context:

```
public void Save(Item i)
{
    using (var context = new ItemContext())
    {
        foreach (var subitem in i.SubItems)
        {
            if (subitem.SubItemId == 0)
                context.Set<SubItem>().Add(subitem);
            else
                context.Set<SubItem>().Attach(subitem);
        }
        context.Set<Item>().Attach(i);
        context.SaveChanges();
    }
}
```

The reason is, because your entity isn't attached to a context when you're doing the attach, EF doesn't actually know where the entities came from - it thought that the FK not being set (probably 0) was a valid state - which is where your error was coming from. The reason you need to attach the child objects first, is so that you can actually add rather than attach. Again, since your context wasn't alive when the subitem was attached, EF isn't sure where the entity came from, and assumes the 0 PK is correct, creating your error.

answered Nov 8 '12 at 16:27

Mark Oreta

9,846 1 28 35

-1

Model

```
public virtual ICollection<CostCenter> CostCenters { get; set; }

[NotMapped]
public int CostCenterId { get; set; }//Just to get item in view
```

Create

```
public ActionResult Create()
{
    ViewBag.CostCenterId = new SelectList(db.CostCenters, "Id", "Name");
```

Create Post

```
public ActionResult Create(Campaign campaign)
{
    ...
    campaign.CostCenters.Add(db.CostCenters.FirstOrDefault(f => f.Id ==
campaign.CostCenterId);
    return RedirectToAction("Index");
}
```

View

```
<%: Html.LabelFor(model => model.CostCenterId ) %>
<div class="input-control">
    <%: Html.DropDownList("CostCenterId ", String.Empty) %>
    <%: Html.ValidationMessageFor(model => model.CostCenterId ) %>
</div>
```

answered Aug 3 '16 at 13:59

Gustavo

17 2

2 This doesn't explain a darn thing – lc. Aug 25 '17 at 5:52