Contents

- Problem
- Solution
  - Entity Framework support for cascading deletes
    - Case 1. Cascading deletes set in database
    - Case 2. Cascading Deletes not set in database
  - Performing a cascading delete using a custom method

When an entity is deleted we often want to delete it's dependent entities too. The samples here show a few different approaches to cascading a delete to dependent entities.

- **Platform:** Console
- **Language:** C#, VB
- **Download:** Performing a Cascading Delete Using a Custom Method, Cascading deletes (Console)

Problem

You generally don't want order detail information without a parent *Order*, and sometimes you'll want neither if the parent *Customer* is deleted. Generally, when you have a dependent entity graph you'll want to set a rule for how to handle deletion of the parent entity.

Solution

The best approach to handling cascaded deletes is to ensure that both your database and your *Entity Model* agree on the delete rule. This ensures unpleasant problems. Nevertheless, there are several approaches to handling the deletion of dependent entities, and we'll walk through them here.

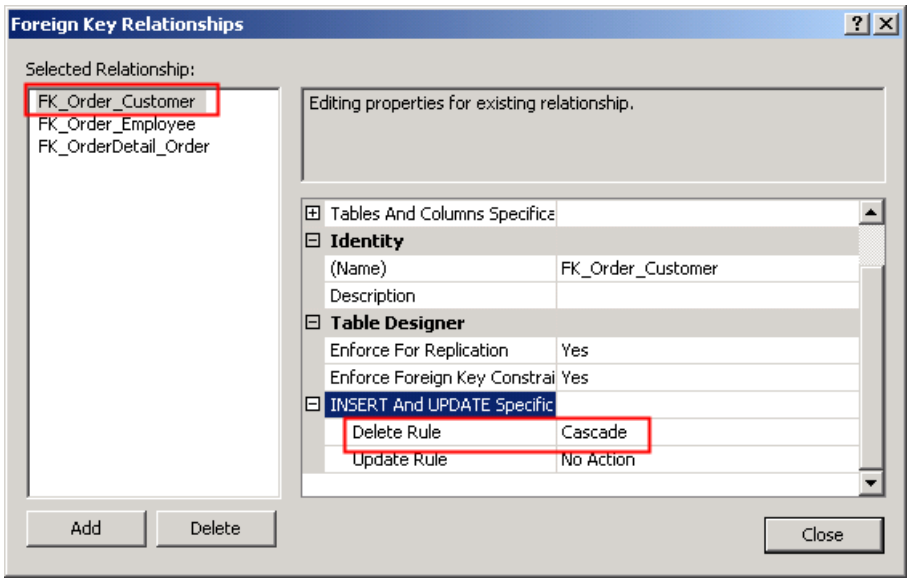
Entity Framework support for cascading deletes

When relationships in the database behind an Entity Data Model (EDM) have their Delete Rule set to “Cascade”, the EDM designer discovers this fact and propagates the instruction through to the model it generates.

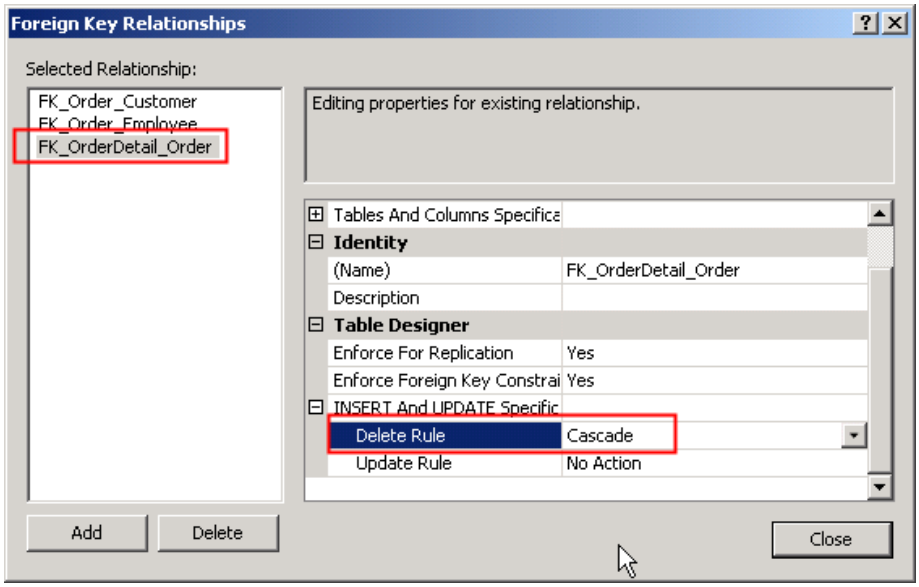
Case 1. Cascading deletes set in database

Suppose, for example, that we have the following settings on the Order table in NorthwindIB (we don't in fact have this rule in place in the database, so you will need to use your imagination).

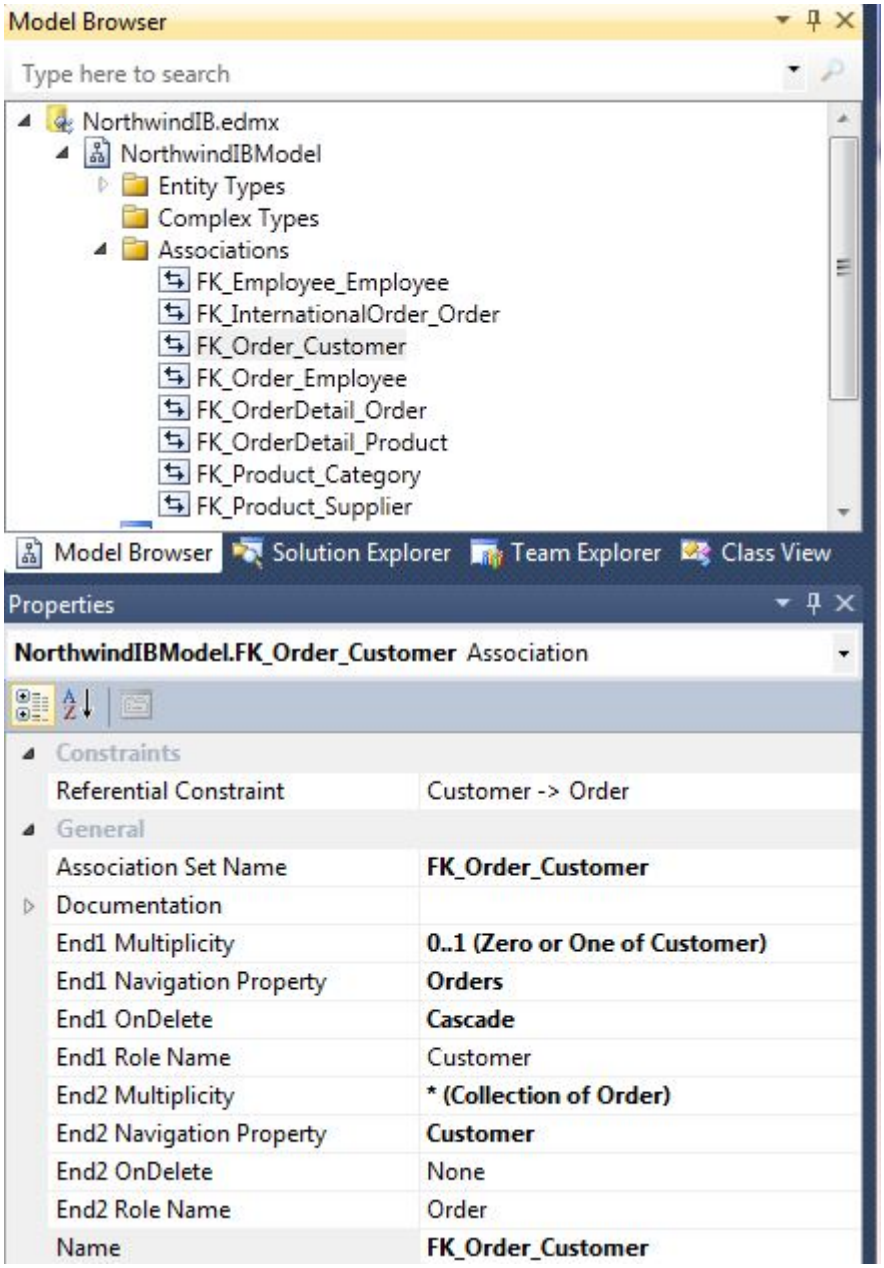
Cascading deletes are turned on between Customer and Order …



... and between Order and OrderDetail ...



We'll see these rules carried forward in the associations in the EDM Designer. Here's Customer-Order:



With cascade rules in place - in both the database and the EDM - you can delete an entity in cache and 1) all dependent entities in cache with the *cascade delete* rule will be marked for deletion, and 2) when the *SaveChanges* is called these entities, and any other dependent entities not in the entity cache at the time of the save, will all be removed.

Case 2. Cascading Deletes not set in database

But suppose we cannot, or do not wish to, turn cascading deletes on in the database. Can we still implement cascading deletes in the Entity Data Model?

Yes, we can. We need only include the *<OnDelete Action="Cascade" />* element in the Association as defined in the conceptual model.

You should exercise caution with this approach, since it's very easy to not have all dependent entities loaded into cache, and to therefore receive a database error when the *SaveChanges* is called.

Performing a cascading delete using a custom method

You can create a custom method to delete a parent entity and its graph too. Such a method would walk the hierarchy descending from the targeted parent, deleting the most remote descendents first and working back to the parent itself.

For example, *aCustomer.DeleteObjectGraph()* would find the Orders connected to the target Customer, then the *OrderDetail* records connected to those Orders; it would then delete the *OrderDetails*, the Orders, and finally the Customer; and in that sequence. Here's a sample of such a method:

```
public void DeleteObjectGraph() {
 foreach (var order in this.Orders.ToList()) {
   foreach (var dtl in order.OrderDetails.ToList()) {
     dtl.EntityAspect.Delete();
    }
   order.EntityAspect.Delete();
  }
 this.EntityAspect.Delete();
}
```

```
Public Sub DeleteObjectGraph()
 For Each order In Me.Orders.ToList()
   For Each dtl In order.OrderDetails.ToList()
     dtl.EntityAspect.Delete()
   Next dtl
    order.EntityAspect.Delete()
 Next order
 Me.EntityAspect.Delete()
End Sub
```