Mapping composite foreign key to composite primary key where the foreign key is also a primary key

Asked 2 years, 7 months ago Active 9 months ago Viewed 5k times

3

2

I want to make VM_hostname,datetime and name properties as a composite Key for **Disk class** . At the same time VM_hostname and datetime of **Disk class** should refer to VM_hostname and datetime of **VirtualMachine** class (ie Foreign keys) .

I did this but it gives me this exception : The ForeignKeyAttribute on property 'datetime' on type 'WebJob1.Historical.Disk' is not valid. The navigation property 'Datetime' was not found on the dependent type 'WebJob1.Historical.Disk'. The Name value should be a valid navigation property name

Anyone have a clue ? Also, please note that im using Data Annotation.

```
public class VirtualMachine
{

    [Key]
    [Column(Order = 0)]
    public string VM_Hostname { get; set; }
    [Key]
    [Column(Order = 1)]
    public DateTime Datetime;
    public virtual List<Disk> disks { get; set; }
}

 public class Disk
{
    [Key,ForeignKey("VirtualMachine"),Column(Order = 0)]
    public string VM_hostname { get; set; }
    [Key,ForeignKey("Datetime"), Column(Order = 1)]
    public DateTime datetime { get; set; }
    [Key, Column(Order = 2)]
    public string name { get; set; }

    public virtual VirtualMachine VirtualMachine{ get; set; }


}
```

entity-framework entity-framework-6 data-annotations

asked Mar 15 '17 at 19:48
NeuralLynx
50 1 10

@GertArnold It was not the same question mister ! I tried the solution you pointed to and have been trying ever since until now. So im asking a simpfied version of the quesion. – NeuralLynx Mar 15 '17 at 19:55

@GertArnold That solution does not have a scenario where the primary key and foreign key are the same. Atleast read the question before lazily marking it as a dupicate – NeuralLynx Mar 15 '17 at 19:57

Data annotations just make your life harder (with fluent config it would have been a matter of a single line of code that you never can make mistake). Anyway, remove the current `ForeignKey` attributes and apply single `ForeignKey("VM_hostname,datetime")` on `VirtualMachine` navigation property. And make sure the property names match exactly (including the casing). – Ivan Stoev Mar 15 '17 at 20:20

1 Answer

5

The main difference between your question and the one I suggested as duplicate is that your `ForeignKey` attributes don't refer -

- from a primitive property to a navigation property
- from a navigation property to a primitive property

In your case, the reference is from a primitive property to another primitive property, in another type. Also, little detail, `VirtualMachine.Datetime` should be a property, not a member. But I have to admit that the "duplicate" didn't cover your case.

So let's try to make this into a comprehensive answer how to handle this situation in Entity Framework 6. I'll use an abstracted model to explain the various options:

```
public class Parent
{
    public int Id1 { get; set; } // Key
    public int Id2 { get; set; } // Key
    public string Name { get; set; }
    public virtual List<Child> Children { get; set; }
}
```

```
public class Child
{
    public int Id1 { get; set; } // Key
    public int Id2 { get; set; } // Key
    public int Id3 { get; set; } // Key
    public string Name { get; set; }
    public virtual Parent Parent { get; set; }
}
```

There are three options to setup the mappings.

Option 1

Data annotations, `ForeignKey` attribute:

```
public class Parent
{
    [Key]
    [Column(Order = 1)]
    public int Id1 { get; set; }
    [Key]
    [Column(Order = 2)]
    public int Id2 { get; set; }

    public string Name { get; set; }

    public virtual List<Child> Children { get; set; }
}

public class Child
{
    [Key]
    [Column(Order = 0)]
    public int Id1 { get; set; }
    [Key]
    [Column(Order = 1)]
    public int Id2 { get; set; }
    [Key]
    [Column(Order = 2)]
    public int Id3 { get; set; }

    public string Name { get; set; }

    [ForeignKey("Id1,Id2")]
    public virtual Parent Parent { get; set; }
}
```

As you see, here the `ForeignKey` attribute refers from a navigation property to primitive properties. Also, the absolute numbers in the column order don't matter, only their sequence.

Option 2

Data annotations, `InverseProperty` attribute:

```
public class Parent
{
    [Key]
    [Column(Order = 1)]
    public int Id1 { get; set; }
    [Key]
    [Column(Order = 2)]
    public int Id2 { get; set; }

    public string Name { get; set; }

    public virtual List<Child> Children { get; set; }
}

public class Child
{
    [Key]
    [Column(Order = 0)]
    [InverseProperty("Children")]
    public int Id1 { get; set; }
    [Key]
    [Column(Order = 1)]
    [InverseProperty("Children")]
    public int Id2 { get; set; }
    [Key]
    [Column(Order = 2)]
    public int Id3 { get; set; }

    public string Name { get; set; }

    public virtual Parent Parent { get; set; }
}
```

`InverseProperty` points from one or more properties in a type at one end of a relationship to a navigation property in the type on the other end of the relationship. Another way to achieve the same mapping is to apply `[InverseProperty("Parent")]` on both key properties of `Parent`.

Option 3

Fluent mapping:

```
modelBuilder.Entity<Parent>().HasKey(p => new { p.Id1, p.Id2 });
modelBuilder.Entity<Child>().HasKey(p => new { p.Id1, p.Id2, p.Id3 });
modelBuilder.Entity<Parent>()
    .HasMany(p => p.Children)
    .WithRequired(c => c.Parent)
    .HasForeignKey(c => new { c.Id1, c.Id2 });
```

As said in the comments, fluent mapping is less error-prone than data annotations. Data annotations offer too many options to configure mappings and it's not always easy to see which parts are connected. That's why fluent mapping is my favorite.

Entity Framework Core

In EF-core (current version 2.2.1) composite primary keys can't be modeled by data annotations. It throws a run-time exception:

> Entity type 'Parent' has composite primary key defined with data annotations. To set composite primary key, use fluent API.

So for EF-core only option 3 is feasible. The mapping is almost identical:

```
modelBuilder.Entity<Parent>().HasKey(p => new { p.Id1, p.Id2 });
modelBuilder.Entity<Child>().HasKey(p => new { p.Id1, p.Id2, p.Id3 });
modelBuilder.Entity<Parent>()
    .HasMany(p => p.Children)
    .WithOne(c => c.Parent) // Different here
    .HasForeignKey(c => new { c.Id1, c.Id2 });
```

edited Jan 19 at 20:07

answered Mar 15 '17 at 21:32
Gert Arnold
83.6k 17 147 225

Thanks, I used the 2nd method and it works perfectly. Also, I could use this too right ? [Key,ForeignKey("Parent"),Column(Order = 1) to define the relationship – NeuralLynx Mar 16 '17 at 7:01

I tried (i.e. these ForeignKey annotations on `Child`'s first two key properties), but it fails with "Multiplicity is not valid in Role 'Child_Parent_Source' ...". It looks like the ForeignKey attribute can't be split like the InverseProperty attribute. – Gert Arnold Mar 16 '17 at 8:15