Contents

Relationships between your database tables are translated into navigation properties in your entities. We'll look at **relationships and navigation properties** in more depth here.

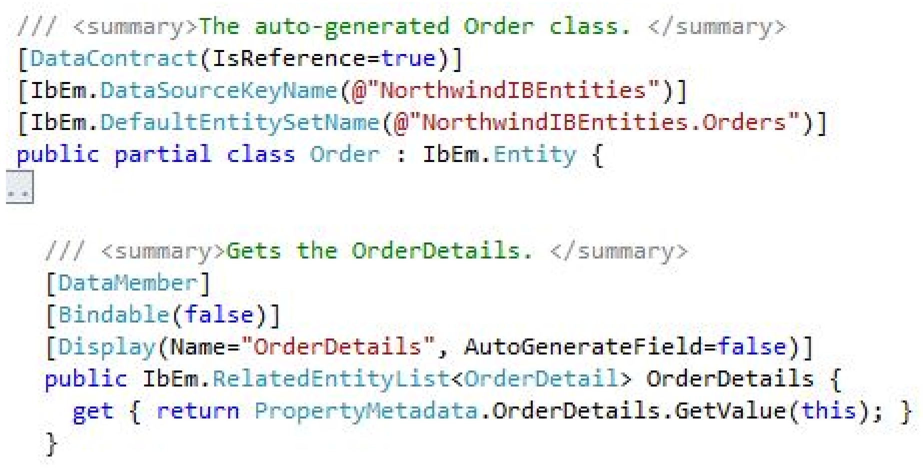
See the next section for how to retrieve related entities using navigation properties.

One-to-many

The most common type of relationship is a one-to-many (or 1:M), sometimes termed **parent->child** or **principal->dependent** relationships. Here, the child entity contains both a foreign key, sometimes nullable, and a scalar navigation property to its parent, while the parent entity contains a collection of child entities.

For example, an *Order* contains a collection of *OrderDetails* - Order.OrderDetails - while each *OrderDetail* contains a reference to its parent *Order* - OrderDetail.Order .

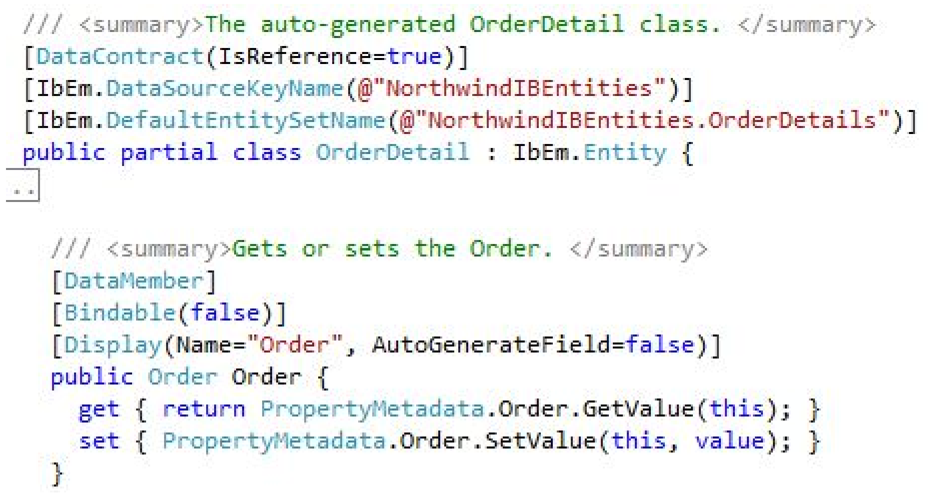
In generated code, you'll see the parent with its child collection, a *RelatedEntityList<T>*:



```
/// <summary>The auto-generated Order class. </summary>
[DataContract(IsReference=true)]
[IbEm.DataSourceKeyName(@"NorthwindIBEntities")]
[IbEm.DefaultEntitySetName(@"NorthwindIBEntities.Orders")]
public partial class Order : IbEm.Entity {
..

  /// <summary>Gets the OrderDetails. </summary>
  [DataMember]
  [Bindable(false)]
  [Display(Name="OrderDetails", AutoGenerateField=false)]
  public IbEm.RelatedEntityList<OrderDetail> OrderDetails {
    get { return PropertyMetadata.OrderDetails.GetValue(this); }
  }
```

While the child will contain a reference to its parent:



```
/// <summary>The auto-generated OrderDetail class. </summary>
[DataContract(IsReference=true)]
[IbEm.DataSourceKeyName(@"NorthwindIBEntities")]
[IbEm.DefaultEntitySetName(@"NorthwindIBEntities.OrderDetails")]
public partial class OrderDetail : IbEm.Entity {
..

  /// <summary>Gets or sets the Order. </summary>
  [DataMember]
  [Bindable(false)]
  [Display(Name="Order", AutoGenerateField=false)]
  public Order Order {
    get { return PropertyMetadata.Order.GetValue(this); }
    set { PropertyMetadata.Order.SetValue(this, value); }
  }
```

One-to-one

A one-to-one relationship might be a shared primary key association, in which the primary key acts as a foreign key, or a one-to-one foreign key association, where a unique constraint is used with the foreign key. The relationship can be optional.

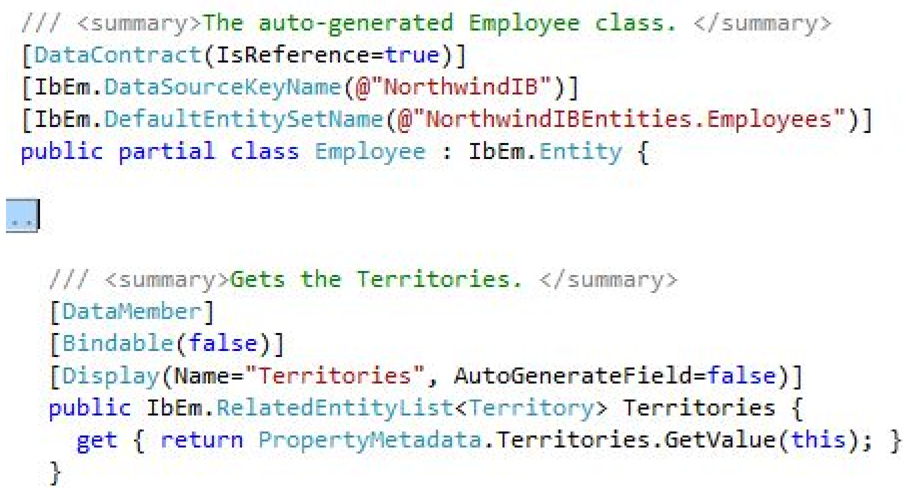
You'll also use a one-to-one relationship with table splitting - when you map multiple entities to a single database table. Table splitting allows you to lazily load less frequently used properties or those with large payloads, such as an image.

Each entity will contain a scalar navigation property to the related end.

Many-to-many

A many-to-many relationship involves defining a third table (called a junction or join table), whose primary key is composed of the foreign keys from both related tables. This join table is abstracted out of the object model, so that both ends of the relationship contain a collection of related entities.

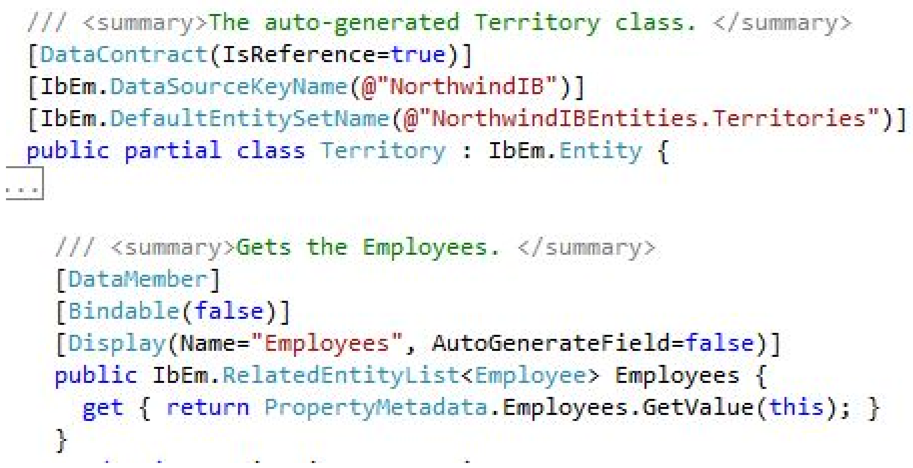
In the *NorthwindIB* database, we see *Employee* and *Territory* joined by the unfortunately named *EmployeeTerritoryNoPayload* table. The generated entities each contain a collection of related entities:



```
/// <summary>The auto-generated Employee class. </summary>
[DataContract(IsReference=true)]
[IbEm.DataSourceKeyName(@"NorthwindIB")]
[IbEm.DefaultEntitySetName(@"NorthwindIBEntities.Employees")]
public partial class Employee : IbEm.Entity {

...

  /// <summary>Gets the Territories. </summary>
  [DataMember]
  [Bindable(false)]
  [Display(Name="Territories", AutoGenerateField=false)]
  public IbEm.RelatedEntityList<Territory> Territories {
    get { return PropertyMetadata.Territories.GetValue(this); }
  }
```



```
/// <summary>The auto-generated Territory class. </summary>
[DataContract(IsReference=true)]
[IbEm.DataSourceKeyName(@"NorthwindIB")]
[IbEm.DefaultEntitySetName(@"NorthwindIBEntities.Territories")]
public partial class Territory : IbEm.Entity {
...

  /// <summary>Gets the Employees. </summary>
  [DataMember]
  [Bindable(false)]
  [Display(Name="Employees", AutoGenerateField=false)]
  public IbEm.RelatedEntityList<Employee> Employees {
    get { return PropertyMetadata.Employees.GetValue(this); }
  }
```

Working with many-to-many relationships differs from other relationship types since there is no foreign key defined within the model. You will always add or remove entities from the *RelatedEntityList* to add and remove entries from the join table in the database. Due to the lack of a defined foreign key in the object model, certain actions are not available or work differently than in other relationship types. For example, cache-only navigation on a many-to-many navigation property is not supported.