Microsoft Dynamics AX AOT Table Relation Properties Guideline

This document explains the implications of the properties on AOT table relations and provides guidelines on how to fill them in when making changes to the AX application.

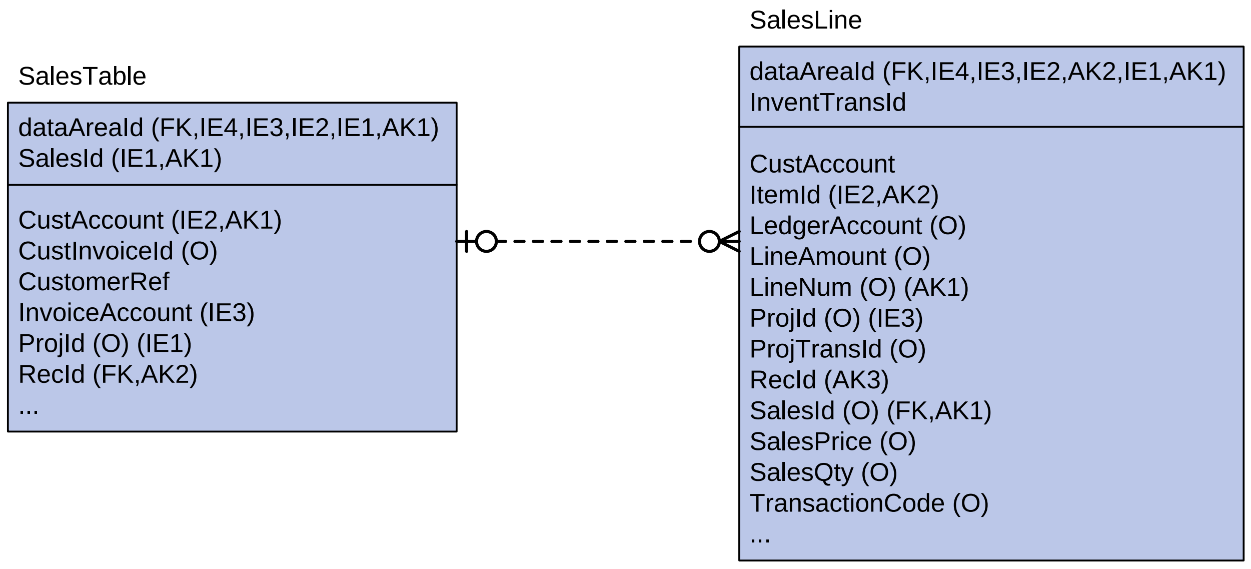
Note

This document uses the *SalesTable* relation from *SalesLine* to *SalesTable* as an example.

When working in AOT on relationships between two tables that have '*SaveDataPerCompany'* property set to true, *DataAreaId* field automatically becomes a part of any relationship but is not mentioned explicitly in this document.

Mapping physical data model to AOT Table Relations

The following physical data model shows the relationship between *SalesTable* and *SalesLine*. The model conveys some important information about this relationship which needs to be captured when modeled in AOT as table relation.

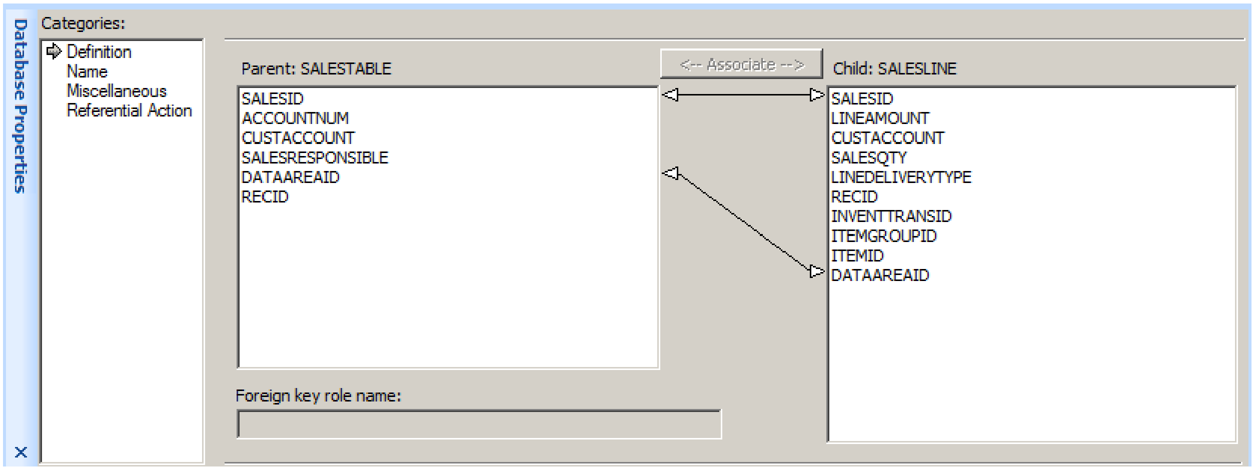


Recommended location for a Relation between two tables

AX Table Relation is specified on the Table that Holds the Foreign Key (FK). This is the norm for any relational database implementation.

The table that holds the FK is referred to as child table and the one that holds the primary key or referenced key is referred to as parent table.

In this case, the relationship between the two tables is manifested through the *SalesId* FK field on the *SalesLine* table pointing to the *SalesId* PK field on the *SalesTable* table.



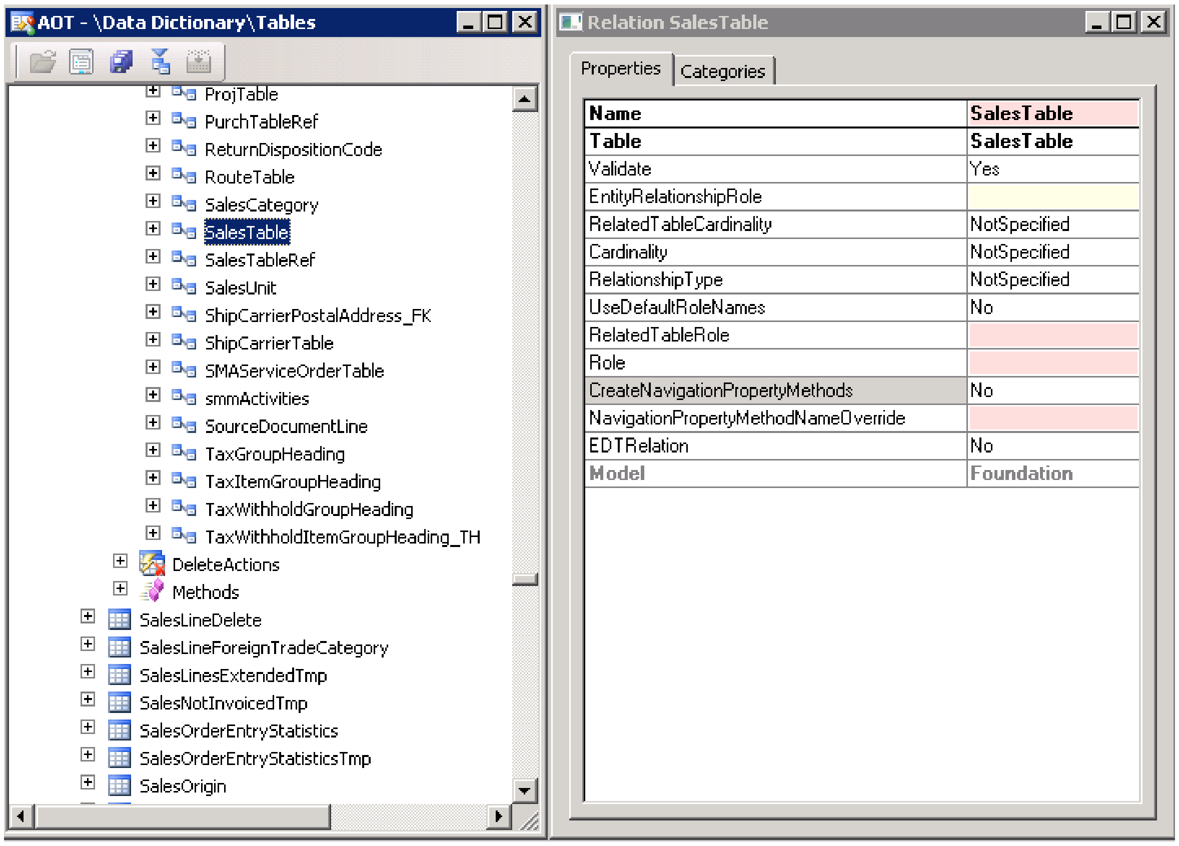
Parent: SALESTABLE

SALESID
ACCOUNTNUM
CUSTACCOUNT
SALESRESPONSIBLE
DATAAREAID
RECID

Child: SALESLINE

SALESID
LINEAMOUNT
CUSTACCOUNT
SALESQTY
LINEDELIVERYTYPE
RECID
INVENTTRANSID
ITEMGROUPID
ITEMID
DATAAREAID

The following shows the corresponding AOT table relation and available properties. Most of them are not populated yet in this case.


AOT - \Data Dictionary\Tables

- ProjTable
- PurchTableRef
- ReturnDispositionCode
- RouteTable
- SalesCategory
- SalesTable
- SalesTableRef
- SalesUnit
- ShipCarrierPostalAddress_FK
- ShipCarrierTable
- SMAServiceOrderTable
- smmActivities
- SourceDocumentLine
- TaxGroupHeading
- TaxItemGroupHeading
- TaxWithholdGroupHeading
- TaxWithholdItemGroupHeading_TH
- DeleteActions
- Methods
- SalesLineDelete
- SalesLineForeignTradeCategory
- SalesLinesExtendedTmp
- SalesNotInvoicedTmp
- SalesOrderEntryStatistics
- SalesOrderEntryStatisticsTmp
- SalesOrigin

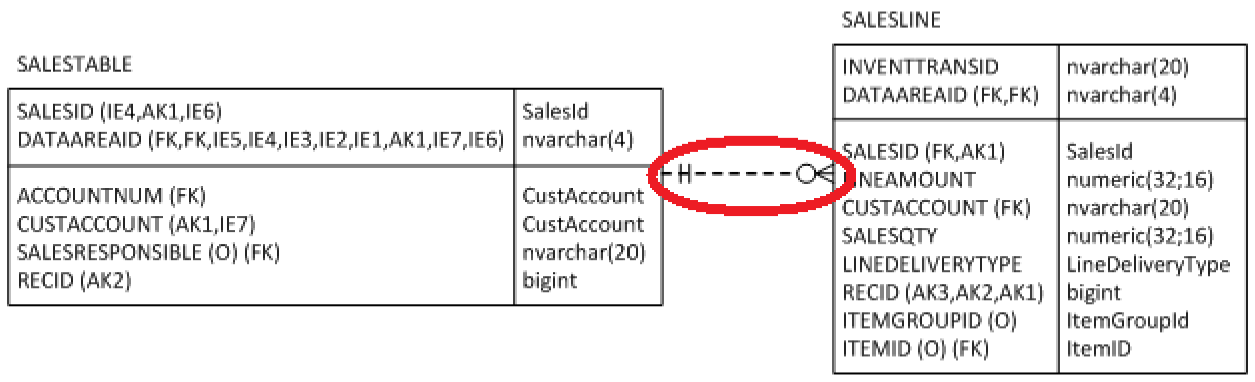
Relation SalesTable

Properties | Categories

Name	**SalesTable**
Table	**SalesTable**
Validate	Yes
EntityRelationshipRole	
RelatedTableCardinality	NotSpecified
Cardinality	NotSpecified
RelationshipType	NotSpecified
UseDefaultRoleNames	No
RelatedTableRole	
Role	
CreateNavigationPropertyMethods	No
NavigationPropertyMethodNameOverride	
EDTRelation	No
Model	**Foundation**

Cardinality and RelatedTableCardinality

These two properties capture the cardinality relationship between the two tables, as represented by the two sides of the ‘crow’s feet’ in the physical model.



SALESTABLE

SALESID (IE4,AK1,IE6)	SalesId
DATAAREAID (FK,FK,IE5,IE4,IE3,IE2,IE1,AK1,IE7,IE6)	nvarchar(4)
ACCOUNTNUM (FK)	CustAccount
CUSTACCOUNT (AK1,IE7)	CustAccount
SALESRESPONSIBLE (O) (FK)	nvarchar(20)
RECID (AK2)	bigint

SALESLINE

INVENTTRANSID	nvarchar(20)
DATAAREAID (FK,FK)	nvarchar(4)
SALESID (FK,AK1)	SalesId
LINEAMOUNT	numeric(32;16)
CUSTACCOUNT (FK)	nvarchar(20)
SALESQTY	numeric(32;16)
LINEDELIVERYTYPE	LineDeliveryType
RECID (AK3,AK2,AK1)	bigint
ITEMGROUPID (O)	ItemGroupId
ITEMID (O) (FK)	ItemID

Specifically, the *Cardinality* property corresponds to the notation on the child table (*SalesLine* in this case). So it should be *ZeroMore* (0...*). *RelatedCardinality* property corresponds to the notation on the parent table (*SalesTable*). So it should be *ExactlyOne* (1…1).

Semantically, *Cardinality* specifies how many instances of *SalesLine* row can be related to a single instance of *SalesTable* row. *ZeroMore* means that for every sale order, there can be zero, or more sales lines related to it. If the business requirement dictates that you need to have at least one sales line to be able to create a sales order, the *Cardinality* would be *OneMore* (1…*).

RelatedCardinality specifies how many instances of *SalesTable* row can be related to a single instance of *SalesLine* row. *ExactlyOne* means that every sales line should belong to one and only one sales order.

Runtime Implication

Currently, there is no runtime behavior for these two properties. In the future, validation rules can potentially leverage these properties. These rules can be enforced at both create and delete time.

RelationshipType

Unfortunately the physical model does not capture the *RelationshipType* property. So business requirements should dictate the value in this property. We have two options for this property in our example: *Association* and *Composition*. The deciding factor between the two is the life time relationship between the two. If one controls the life cycle of the other, it is a composition relationship.

In this case, the life time of a sales order determines the life time of the sale line so the relationship type should be set as *Composition*.

Runtime Implication

Currently, there is no runtime behavior associated with this property. However, cascade delete behavior should be in line with this property.

Role and RelatedTableRole

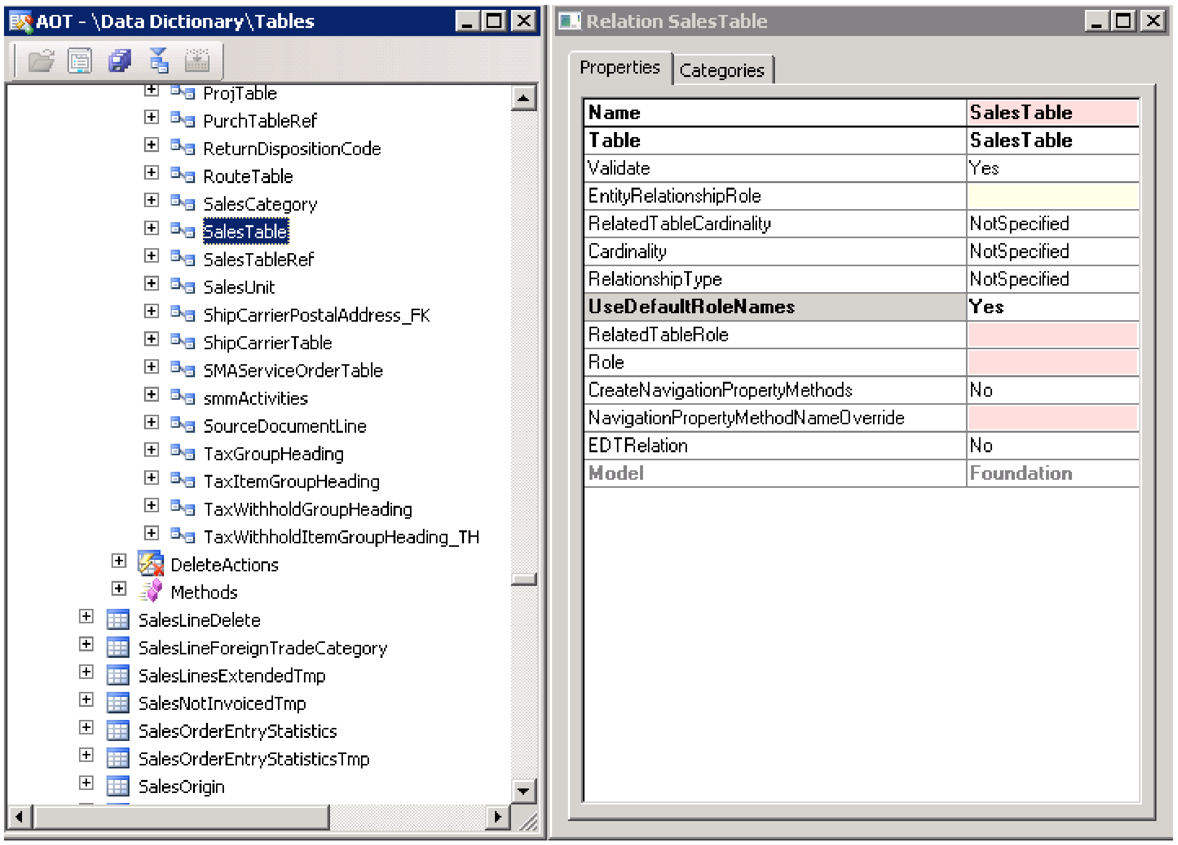
The *Role* and the *RelatedTableRole* properties specify the functions/roles played by the two tables in the relationship. The physical data model does not capture this information. They should be specified according to business domain requirement.

In this specific relationship, a row in *SalesTable* plays the role of a sales order and a row in *SalesLine* plays the role of a sales line (sales table has zero or many sales lines). So '*SalesOrder*' and '*SalesLine*' are used as *RelatedTableRole* and *Role*.

The following defaulting mechanism can be used for role names without having to fill out the two properties (by leaving them blank)

(1) <related table name> is used as the *RelatedTableRole*
(2) For Role
    a. <related table name>_<table name> is used if the table's *EntityRelationshipType* is *Relationship*.
    b. <table name> is used if the table's *EntityRelationshipType* is *Entity*

By default, the default role name mechanism is not turned on. This is to prevent role name collisions in the standard application. You will have to set '*DefaultRoleName*' to '*Yes*' if you want to use the default role names.


AOT - \Data Dictionary\Tables

- ProjTable
- PurchTableRef
- ReturnDispositionCode
- RouteTable
- SalesCategory
- SalesTable
- SalesTableRef
- SalesUnit
- ShipCarrierPostalAddress_FK
- ShipCarrierTable
- SMAServiceOrderTable
- smmActivities
- SourceDocumentLine
- TaxGroupHeading
- TaxItemGroupHeading
- TaxWithholdGroupHeading
- TaxWithholdItemGroupHeading_TH
- DeleteActions
- Methods
- SalesLineDelete
- SalesLineForeignTradeCategory
- SalesLinesExtendedTmp
- SalesNotInvoicedTmp
- SalesOrderEntryStatistics
- SalesOrderEntryStatisticsTmp
- SalesOrigin

Relation SalesTable

Name	**SalesTable**
Table	**SalesTable**
Validate	Yes
EntityRelationshipRole	
RelatedTableCardinality	NotSpecified
Cardinality	NotSpecified
RelationshipType	NotSpecified
UseDefaultRoleNames	**Yes**
RelatedTableRole	
Role	
CreateNavigationPropertyMethods	No
NavigationPropertyMethodNameOverride	
EDTRelation	No
Model	Foundation

Note that when *UseDefaultRoleNames* property is set to *Yes*, the defaulted role names are not shown in the *RelatedTableRole* and *Role* property. *RelatedTableRole* and *Role* property can be used to override the default role names.

Metadata APIs, like *DictRelation* methods, will automatically pick up the defaulted relation name. AOT tree node API, however, will return the actual value of the property.

Relation names and *RelatedTableRole* names should be unique within a table or the entire table hierarchy if the table participates in SCsc. The uniqueness is enforced by the kernel at compilation time.

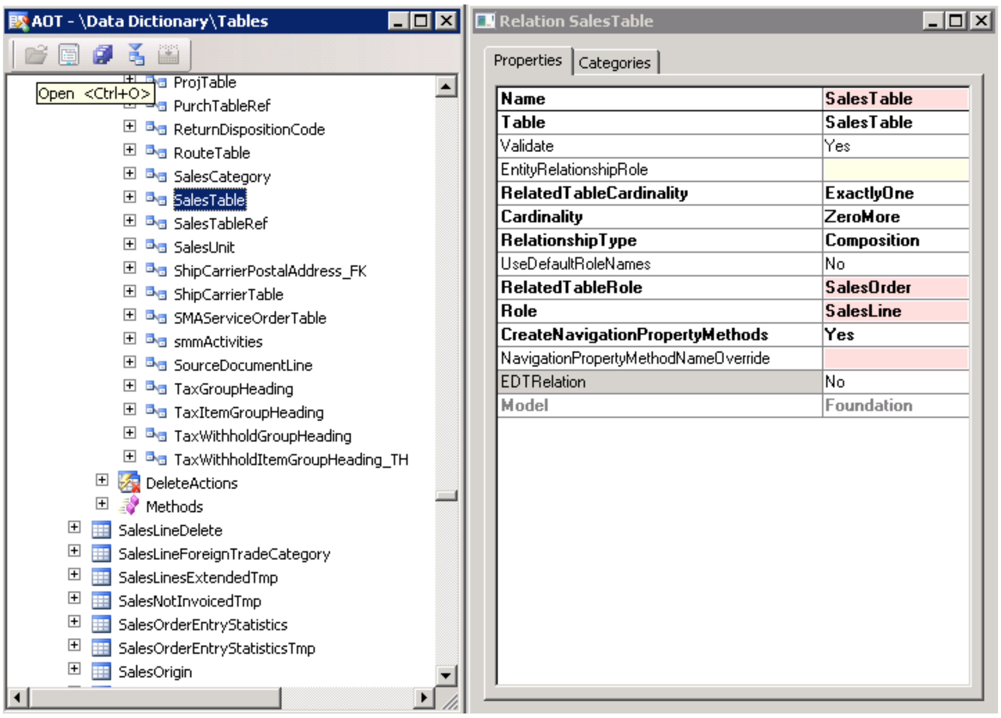
Runtime Implication

The *Role* and *RelatedTableRole* properties are used in two cases in AX 2012. They are described below.

RelatedTableRole as Navigation Property Method

Navigation property is a property method that allows you to link table buffers through a FK relationship. This is used in, for example, the Unit of Work framework, where table buffers are first linked by using navigation property methods and then handed over to the Unit of Work class for persisting changes to the database later.

Navigation property methods are generated by the kernel for a specific FK relationship. To trigger the generation, set the *CreateNavigationPropertyMethods* on an FK relation on a table.


AOT - \Data Dictionary\Tables

Open <Ctrl+O>

- ProjTable
- PurchTableRef
- ReturnDispositionCode
- RouteTable
- SalesCategory
- SalesTable
- SalesTableRef
- SalesUnit
- ShipCarrierPostalAddress_FK
- ShipCarrierTable
- SMAServiceOrderTable
- smmActivities
- SourceDocumentLine
- TaxGroupHeading
- TaxItemGroupHeading
- TaxWithholdGroupHeading
- TaxWithholdItemGroupHeading_TH
- DeleteActions
- Methods
- SalesLineDelete
- SalesLineForeignTradeCategory
- SalesLinesExtendedTmp
- SalesNotInvoicedTmp
- SalesOrderEntryStatistics
- SalesOrderEntryStatisticsTmp
- SalesOrigin

Relation SalesTable

Properties | Categories

Name	**SalesTable**
Table	**SalesTable**
Validate	Yes
EntityRelationshipRole	
RelatedTableCardinality	**ExactlyOne**
Cardinality	**ZeroMore**
RelationshipType	**Composition**
UseDefaultRoleNames	No
RelatedTableRole	**SalesOrder**
Role	**SalesLine**
CreateNavigationPropertyMethods	**Yes**
NavigationPropertyMethodNameOverride	
EDTRelation	No
Model	Foundation

So where does the kernel get the name for the navigation property? There are two ways.

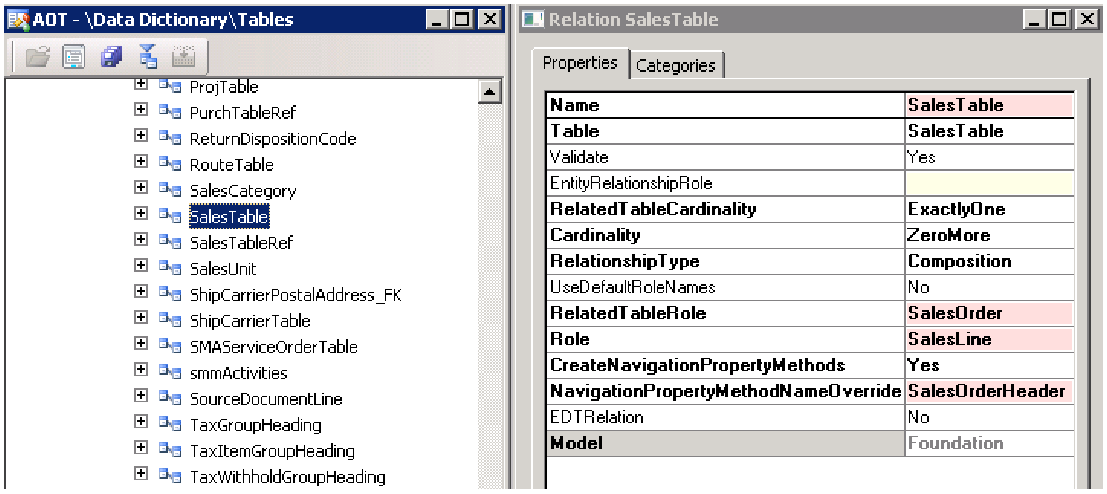
For a foreign key relation (the one that ensures the referenced fields form a unique key), if *RelatedTableRole* is available, either through defaulting or through overriding, it is used as navigation property method name.

In the *SalesLine* example, if the table relation was modeled as a FK relation in AOT, the kernel would automatically generate a navigation method on the *SalesLine* table with the following signature:

public final SalesTable SalesOrder([SalesTable relatedTable])

Because any navigation property method is actually a method, what if the name conflicts with an existing method on the table?

In these cases, developers can use the *NavigationPropertyMethodNameOverride* property to specify a name different than the *RelatedTableRole*. Another case where this property can be used is when the length of the *RelatedTableName* (which can be as long as 128 char) exceeds that allowed for table methods (40 char).


AOT - \Data Dictionary\Tables

- ProjTable
- PurchTableRef
- ReturnDispositionCode
- RouteTable
- SalesCategory
- SalesTable
- SalesTableRef
- SalesUnit
- ShipCarrierPostalAddress_FK
- ShipCarrierTable
- SMAServiceOrderTable
- smmActivities
- SourceDocumentLine
- TaxGroupHeading
- TaxItemGroupHeading
- TaxWithholdGroupHeading

Relation SalesTable

Properties | Categories

Name	**SalesTable**
Table	**SalesTable**
Validate	Yes
EntityRelationshipRole	
RelatedTableCardinality	**ExactlyOne**
Cardinality	**ZeroMore**
RelationshipType	**Composition**
UseDefaultRoleNames	No
RelatedTableRole	**SalesOrder**
Role	**SalesLine**
CreateNavigationPropertyMethods	**Yes**
NavigationPropertyMethodNameOverride	**SalesOrderHeader**
EDTRelation	No
Model	Foundation

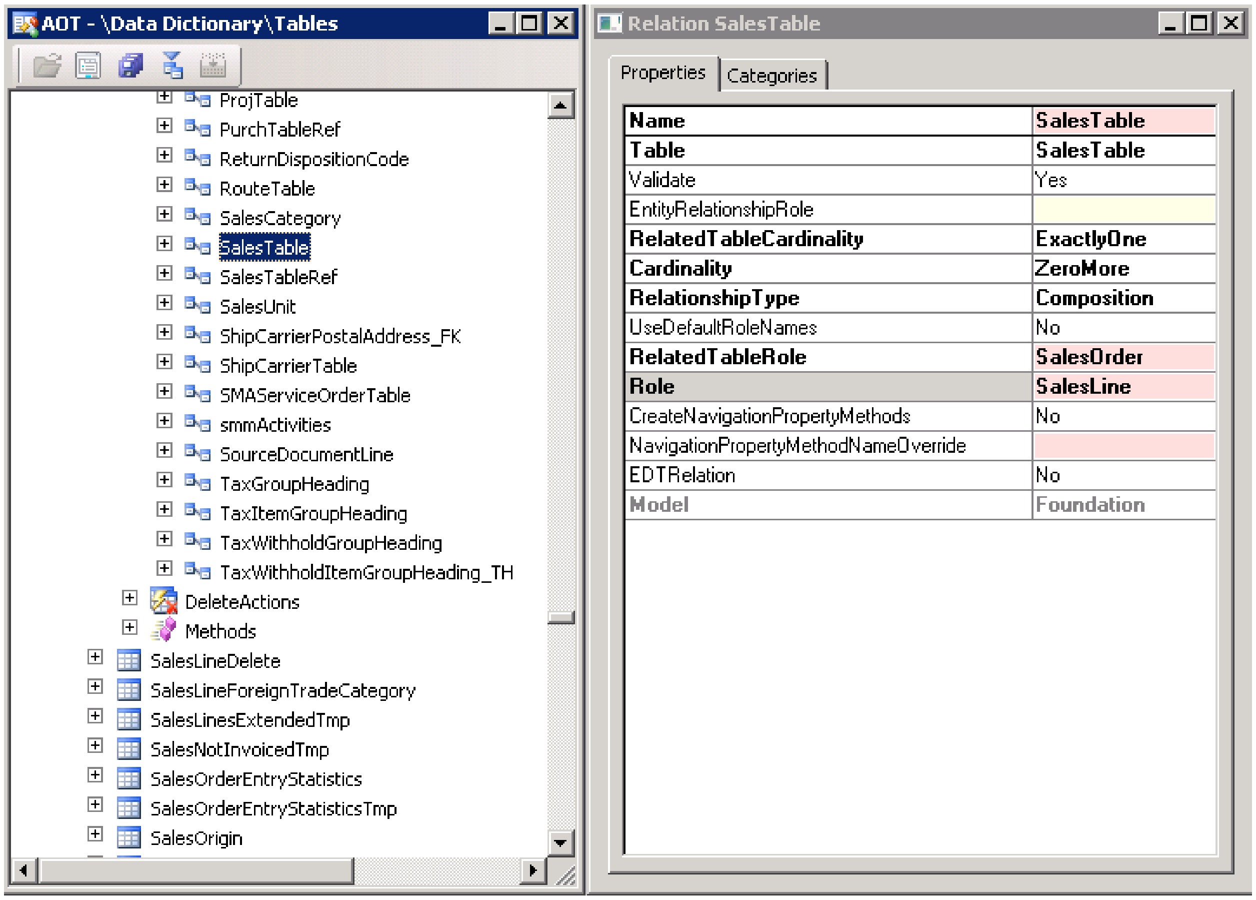
Again, compiler checks for name conflicts with other methods specified on the same table or on the table hierarchy if this table is part of SCsc.

Role and RelatedTableRole as Join Relation in Query

In both AOT query node and query API, you can choose to specify a join relation between two query data sources by pointing to a directional relation through the *RelatedTableRole* property on the query data source. This eliminates the need for re-creating field joins which is redundant and error prone.

Final result

The following shows the table relation properties populated with the recommended values.



Name	**SalesTable**
Table	**SalesTable**
Validate	Yes
EntityRelationshipRole	
RelatedTableCardinality	**ExactlyOne**
Cardinality	**ZeroMore**
RelationshipType	**Composition**
UseDefaultRoleNames	No
RelatedTableRole	**SalesOrder**
Role	**SalesLine**
CreateNavigationPropertyMethods	No
NavigationPropertyMethodNameOverride	
EDTRelation	No
Model	**Foundation**