Contents

This sample shows how to define and execute a stored procedure.

- **Platform:** Silverlight
- **Language:** C#, VB
- **Download:** Stored procedure queries (Silverlight)

Problem

You'd like to use a *query* stored procedure, i.e., one returning data. Once defined, how do you call it?

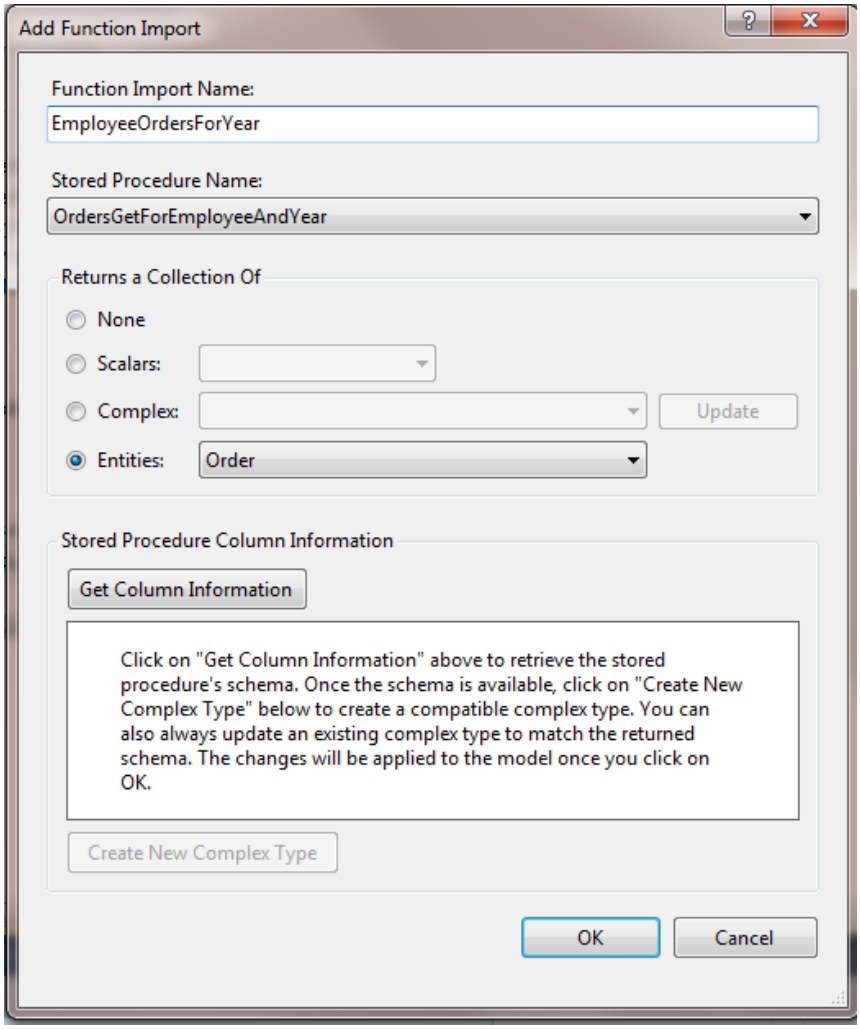
Solution

This sample shows how to define a stored procedure in the Entity Data Model and then how to execute the StoredProcQuery in a Silverlight application.

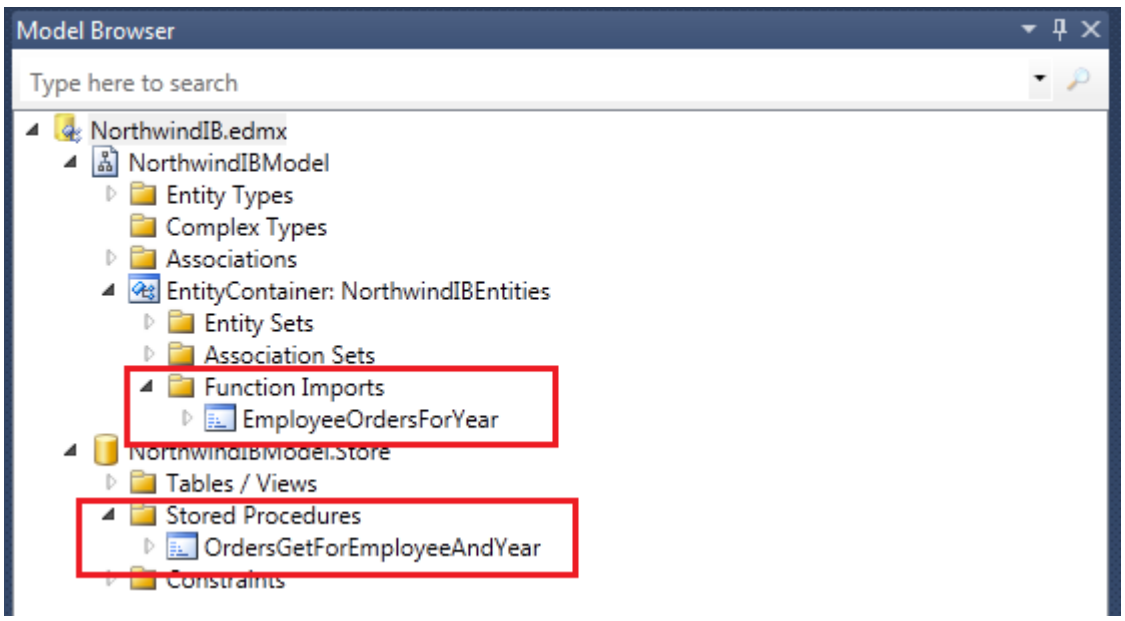
Defining the procedure in the EDM

The NorthwindIB database contains a stored procedure called “OrdersGetForEmployeeAndYear”, which takes two arguments, an employee ID and the year, and returns all columns from the Order table meeting the criteria. We'll use this stored procedure in the sample.

When creating the NorthwindIBModel.edmx model with the wizard we included this stored procedure along with several tables. This added the stored procedure to the "storage definition" of the model. We must also add a “Function Import” to represent the procedure in the conceptual model.



Here's what the resulting model looks like in the Model Browser:



Note that we changed the name of the function import to "EmployeeOrdersForYear".

In the auto-generated code, DevForce generates two methods for each stored procedure – one method to create and execute the query, and another method to create and return the query:

```
IEnumerable<Order> EmployeeOrdersForYear(Nullable<int> EmployeeID, Nullable<int> Year);
StoredProcQuery EmployeeOrdersForYearQuery(Nullable<int> EmployeeID, Nullable<int> Year);
```

```
IEnumerable(Of Order) EmployeeOrdersForYear(Integer? EmployeeID, Integer? Year)
StoredProcQuery EmployeeOrdersForYearQuery(Integer? EmployeeID, Integer? Year)
```

Using the sample

In Silverlight we can't use the "EmployeeOrdersForYear" generated method, because it will synchronously execute the *StoredProcQuery* and return the results, and synchronous queries to the server are not supported in Silverlight. So instead we'll use the second generated method, "EmployeeOrdersForQuery", which will build a *StoredProcQuery* for us with all query parameters set. We don't have to use this helper method; we could build up the *StoredProcQuery* in code, but the helper makes things easier.

```
StoredProcQuery query = _mgr.EmployeeOrdersForYearQuery(EmployeeID: 1, Year: 1996);
query.ExecuteAsync(GotOrders);
```

```
Dim query As StoredProcQuery = _mgr.EmployeeOrdersForYearQuery(EmployeeID:= 1, Year:= 1996)
query.ExecuteAsync(GotOrders)
```

When the query completes, we also show how to use asynchronous navigation to retrieve scalar and list navigation properties for related data. Asynchronous navigation of related entities is another feature which can be used in Silverlight applications, since by default DevForce uses lazy loading of related data. In non-Silverlight applications this lazy loading is performed synchronously, but since queries will be sent to the server to retrieve the related data, we must use asynchronous queries in Silverlight, what we call "asynchronous navigation". Asynchronous navigation is not specific to use with stored procedure queries, but since these queries cannot have 'Include" methods to also bring in related data, asynchronous navigation becomes especially useful.

For scalar properties, you can listen on the *PendingEntityResolved* event to be notified when the entity is loaded into cache:

```
anOrder.OrderDetails.PendingEntityResolved += (o, e) => {
```

```
AddHandler details.PendingEntityResolved, Sub(o, e)
```

You can also test whether an entity is "pending", i.e., an asynchronous query has been issued for it but the entity is not yet in cache –

```
if (anOrder.SalesRep.EntityAspect.IsPendingEntity) {}
```

```
If anOrder.SalesRep.EntityAspect.IsPendingEntity Then
End If
```

For list properties, you'll listen on the *PendingEntityListResolved* event:

```
anOrder.OrderDetails.PendingEntityListResolved += (o, e) => {
```

```
Dim details As RelatedEntityList(Of OrderDetail) = anOrder.OrderDetails
AddHandler details.PendingEntityListResolved, Sub(o, e)
```

And can test if the list is pending -

```
if (anOrder.SalesRep.EntityAspect.IsPendingEntityList) {}
```

```
If anOrder.SalesRep.EntityAspect.IsPendingEntityList Then
End If
```

Note that you should remove event handlers when they're no longer needed to avoid memory leaks.