Contents

DevForce supports querying for entities using **stored procedure queries**. The need arises most frequently when we require the entities resulting from an extraordinarily complex query involving large volumes of intermediate data that are not themselves required on the client.

One might imagine a multi-step query that touched several tables, performed multi-way joins, ordered and aggregated the intermediate results, and compared values with many thousands of records, all so as to return a handful of qualifying results. All of the other data were needed only to satisfy the query; the user won't see any of them and there is no point to transmitting them to the client.

This is a clear case for a stored procedure because we can and should maximize performance by performing all operations as close to the data source as possible.

Chances are that the entities returned by the stored procedure are entities we already know. That procedure could be just an especially resource-consuming query for Order entities that we retrieve and save in the usual way under normal circumstances.

The*StoredProcQuery* is perfect for this situation. We define such a query, identify *Order* as the query return type, and turn it loose on the database. We accept the sproc-selected *Order* objects and work with them in our typical merry way.

Note that a stored procedure query, by its nature, must be executed by the database: we can't run it against the entity cache. So we may not invoke it while the application is running offline.

Stored procedure queries

Suppose your data source includes a stored procedure named *SalesByYear*. It is defined as follows: (This example uses SQL Server TSQL, but any stored procedure than is supported via the Entity Framework will also be supported by DevForce).

TSQL

```
ALTER procedure "SalesbyYear"
    @Beginning_Date DateTime, @Ending_Date DateTime
    AS
    SELECT OrderSummary.ShippedDate, OrderSummary.id, "Order Subtotals".Subtotal, DATENAME(yy,ShippedDate) AS Year
    FROM OrderSummary INNER JOIN "Order Subtotals"
    ON OrderSummary.Id = "Order Subtotals".OrderSummaryId
    WHERE OrderSummary.ShippedDate Between @Beginning_Date And @Ending_Date
```

Along with tables and views, stored procedures can be added to the model using the EDM Designer. Adding the stored procedure above results in the following Function element in the schema (SSDL) section of the Entity Model file:

XML

```
<Function Name="SalesbyYear" Schema="dbo" Aggregate="false"
  BuiltIn="false" NiladicFunction="false" IsComposable="false"
     ParameterTypeSemantics="AllowImplicitConversion">
  <Parameter Name="Beginning_Date" Type="datetime" Mode="In" />
  <Parameter Name="Ending_Date" Type="datetime" Mode="In" />
</Function>
```

To make this conveniently available for calling directly off of our entitymanager (as you would equally have to do to make it available on the ADO.NET *ObjectContext*), you must also add a *FunctionImport* element to the conceptual model, using the EDM Designer (see http://msdn.microsoft.com/en-us/library/bb896231.aspx for more information on the mechanics of adding stored procedures to the Entity Model). The resulting *FunctionImport* would be defined as follows within the CSDL portion of the model:

XML

```
<FunctionImport Name="GetSalesByYear" EntitySet=
  "SalesByYearResults" ReturnType=
     "Collection(IdeaBladeTest1Model.EF.SalesbyYear)">
  <Parameter Name="Beginning_Date" Type="DateTime" Mode="In" />
  <Parameter Name="Ending_Date" Type="DateTime" Mode="In" />
</FunctionImport>
```

DevForce will generate query and execute methods into your EntityManager for every function import in the conceptual model. In a Silverlight application, you'd call the query method to construct a query and then execute that query asynchronously; the execute method cannot be used because it will execute the query synchronously.

```
public IEnumerable<IdeaBladeTest1Model.SalesbyYear> GetSalesByYear(
  Nullable<DateTime> Beginning_Date, Nullable<DateTime> Ending_Date) {}
public StoredProcQuery GetSalesByYearQuery(
  Nullable<DateTime> Beginning_Date, Nullable<DateTime> Ending_Date) {}
```

```
Public Function GetSalesByYear(ByVal Beginning_Date? As Date, _
 ByVal Ending_Date? As Date) As  _
   IEnumerable(Of IdeaBladeTest1Model.SalesbyYear)
End Function
Public Function GetSalesByYearQuery(ByVal Beginning_Date? As Date, _
 ByVal Ending_Date? As Date) As StoredProcQuery
End Function
```

Having done all of that in your Entity Model, you can now use the resultant methods as shown below:

```
var _em1 = new IdeaBladeTest1Entities();
DateTime dt1 = DateTime.Parse("1/1/1990");
DateTime dt2 = DateTime.Parse("1/1/2000");
var results = _em1.GetSalesByYear(dt1, dt2);
// Or asynchronously
IEnumerable results;
var q = _em1.GetSalesByYearQuery(dt1, dt2);
var op = _em1.ExecuteQueryAsync(q);
op.Completed += (o, e) => {
  results = e.Results;
};
```

```
Dim _em1 = New IdeaBladeTest1Entities()
Dim dt1 As Date = Date.Parse("1/1/1990")
Dim dt2 As Date = Date.Parse("1/1/2000")
Dim results = _em1.GetSalesByYear(dt1, dt2)
' Or asynchronously
Dim results As IEnumerable
Dim q = _em1.GetSalesByYearQuery(dt1, dt2)
Dim op = _em1.ExecuteQueryAsync(q)
AddHandler op.Completed, Sub(o, e) results = e.Results
```

Stored procedure entity navigation

Dot Navigation is a bit tricky for entities that are defined only by a stored procedure. Navigating to these entities via navigation properties is not supported, since EF itself does not support mapping functions for an entity to a query function (as it does for insert, update and delete functions). You can of course define custom properties within your entities to perform this navigation via a stored procedure query. Navigating from these stored-procedure backed entities to table-backed entities is not a problem.