

# SSAS Cube Exploration: Digging Through the Details with Drillthrough

When exploring the data stored in a SQL Server Analysis Services (SSAS) cube, we sometimes need to know the details of the transactions that generated the information that we're visualizing. We need to go into the multidimensional model and execute a *drillthrough operation* to see the transactional details.

What's a drillthrough operation and how does it differ from other drill operations? This question can best be answered by looking at the four types of drill operations:

- Drill up. Drilling up entails going up a level in the hierarchy to explore data.
- Drill down. Drilling down entails going down a level in the hierarchy to explore data.
- Drill across. Drilling across is about exploring another table of facts related to the present table through a common dimension.
- Drillthrough. Drillthrough is about obtaining, from a cell in a cube, the details behind the information or measure that we're visualizing.

But how do we execute a drillthrough operation to get the details? If we detach ourselves from the multidimensional model, we know that we can obtain information (which is typically aggregated) from our cube and this information originates from a series of fact tables or individual transactions in the relational database. So, we might think that if we want to see the transactions that give a certain measure, we would look for them in the relational data warehouse. However, we can find these details inside the SSAS model itself, and we can extract these details regardless of the relational engine.

Nevertheless, most cubes have more than physical measures (e.g., a sum of sales) in them. We often find calculated measures. For example, a cube might include a measure that shows the amount of profit for a product, and this profit has been calculated by subtracting the product's total cost from the product's total sales (profit=sale-cost). Although it's a very simple calculation, obtaining the details for this measure isn't going to be easy.

We're also going to find other obstacles when it comes to analyzing the details. For example, if we execute a drillthrough operation on a subtotal obtained by filtering several elements from one or more dimensions, the results won't be what we expected.

Let's look at the options we have to extract the maximum level of detail from our information. Those options include:

- Using the DRILLTHROUGH statement in a Multidimensional Expression (MDX) query
- Using the drillthrough action in Business Intelligence Development Studio (BIDS)
- Transforming a calculated measure into a physical measure
- Using a personalized action
- Transforming a subtotal into a true cell

## Using the DRILLTHROUGH Statement in an MDX Query

The easiest way to get the details is executing the DRILLTHROUGH statement in an MDX query. In this operation, we need to specify the cell that we're visualizing and the attributes that we want to obtain. We can limit the number of results returned. For example, to limit the number of returned rows to 100, we would specify:

```
Maxrows 100
```

To identify the cell, we would use the code:

```
Select  
(  
  [Measures].[Sales Amount],  
  [Product].[Product].[Product Name].&[1],  
  [Date].[Calendar YQMD].[Date].&[20070101]  
) on 0  
From [Operation]
```

To specify the attributes and measures to be returned, we would use the code:

```
RETURN  
[Sales].[Sales Unit Cost],  
[Sales].[Sales Unit Price],  
[Sales].[Sales Quantity],  
[Sales].[Sales Amount],  
[$Product].[Product Name],  
[$Product].[Product Subcategory Name],  
[$Product].[Product Category Name],  
[$Date].[Date Description]
```

Listing 1 shows the complete query. Figure 1 shows sample results, which have been formatted for easy viewing.

**Listing 1: DRILLTHROUGH statement in an MDX query**

```

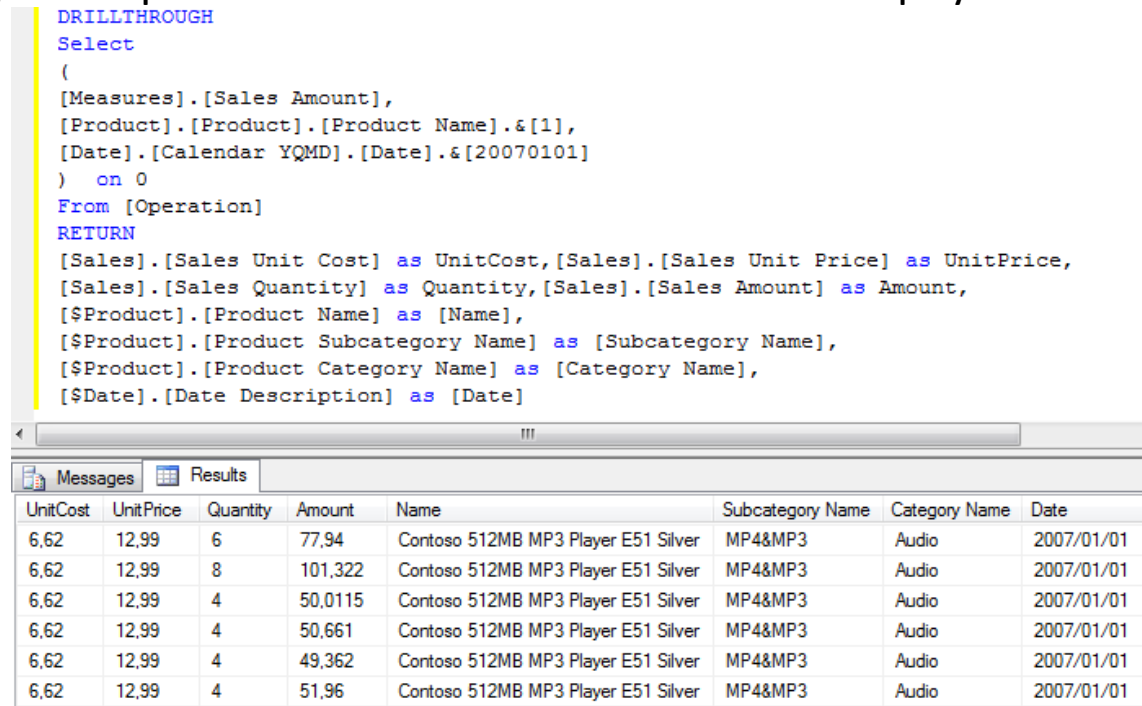
DRILLTHROUGH

maxrows 100

Select
(
[Measures].[Sales Amount],
[Product].[Product].[Product Name].&[1],
[Date].[Calendar YQMD].[Date].&[20070101]
) on 0
From [Operation]

RETURN
[Sales].[Sales Unit Cost],[Sales].[Sales Unit Price],
[Sales].[Sales Quantity],[Sales].[Sales Amount],
[$Product].[Product Name],[Product].[Product Subcategory Name],
[$Product].[Product Category Name],
[$Date].[Date Description]
    
```

**Figure 1: Sample results from a DRILLTHROUGH statement in an MDX query**



```

DRILLTHROUGH
Select
(
[Measures].[Sales Amount],
[Product].[Product].[Product Name].&[1],
[Date].[Calendar YQMD].[Date].&[20070101]
) on 0
From [Operation]
RETURN
[Sales].[Sales Unit Cost] as UnitCost,[Sales].[Sales Unit Price] as UnitPrice,
[Sales].[Sales Quantity] as Quantity,[Sales].[Sales Amount] as Amount,
[$Product].[Product Name] as [Name],
[$Product].[Product Subcategory Name] as [Subcategory Name],
[$Product].[Product Category Name] as [Category Name],
[$Date].[Date Description] as [Date]
    
```

UnitCost	UnitPrice	Quantity	Amount	Name	Subcategory Name	Category Name	Date
6.62	12.99	6	77.94	Contoso 512MB MP3 Player E51 Silver	MP4&MP3	Audio	2007/01/01
6.62	12.99	8	101.322	Contoso 512MB MP3 Player E51 Silver	MP4&MP3	Audio	2007/01/01
6.62	12.99	4	50.0115	Contoso 512MB MP3 Player E51 Silver	MP4&MP3	Audio	2007/01/01
6.62	12.99	4	50.661	Contoso 512MB MP3 Player E51 Silver	MP4&MP3	Audio	2007/01/01
6.62	12.99	4	49.362	Contoso 512MB MP3 Player E51 Silver	MP4&MP3	Audio	2007/01/01
6.62	12.99	4	51.96	Contoso 512MB MP3 Player E51 Silver	MP4&MP3	Audio	2007/01/01

Although this approach is easy to implement, we can't analyze the details of calculated measures because these measures aren't stored in the data structure. Instead, they're calculated during the query runtime.

## Using the Drillthrough Action in BIDS

If we edit our cube from BIDS, we can insert a series of actions in our cube. These actions will allow us to quickly and simply establish access to an analysis that obtains details.

To begin, we need to open the cube in BIDS and go to the Actions tab in the editor. We then need to click the New Drillthrough Action button shown in Figure 2. A form like the one in Figure 3 will appear in the display pane.

**Figure 2: Creating a new drillthrough action**



**Figure 3: The form for the new drillthrough action**

Name:

Action Target  
 Condition (Optional)  
 Drillthrough Columns  
 Additional Properties

In the form, we must specify a name for the drillthrough action (Sales Details in our example) and which group of measures will be affected. We can choose all (All) or one in particular. In this example, the group of sales measures (Sales) is selected, as Figure 4 shows.

**Figure 4: Selecting the group of measures for the drillthrough action**

Name:

Action Target

Measure group members:

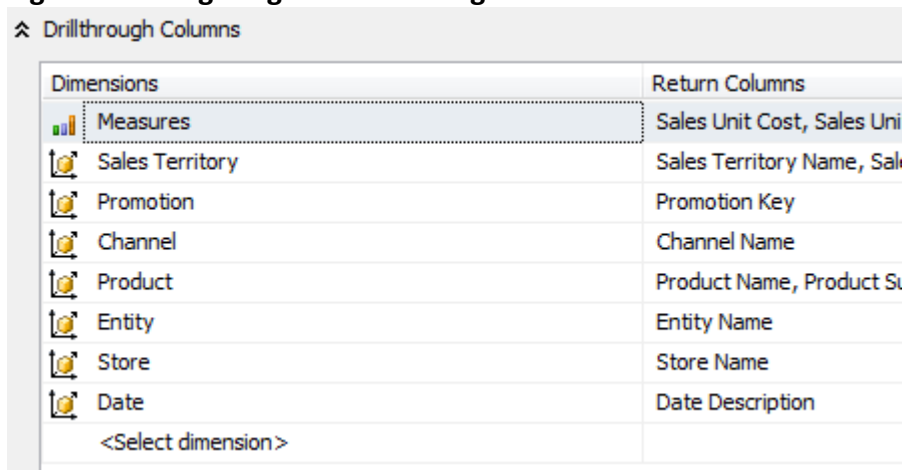
The condition, which is optional as seen in Figure 5, will be an MDX expression of Boolean type. If its value is true, the action will be carried out. If its value is false, the action won't be carried out.

**Figure 5: Configuring the Condition (Optional) section for the drillthrough action**



In the Drillthrough Columns section (see Figure 6), we need to configure the columns in the drillthrough analysis, which measures we want to visualize, and which attributes of the dimensions we will show together. We must take into account that only those attributes that have been marked as Hierarchizable can be selected for their inclusion in the action.

**Figure 6: Configuring the Drillthrough Columns section for the drillthrough action**



Dimensions	Return Columns
Measures	Sales Unit Cost, Sales Uni
Sales Territory	Sales Territory Name, Sal
Promotion	Promotion Key
Channel	Channel Name
Product	Product Name, Product St
Entity	Entity Name
Store	Store Name
Date	Date Description
<Select dimension>	

Finally, we need to configure the Additional Properties section, which Figure 7 shows. Here is a rundown of the properties in this section:

- **Default.** Using the Default drop-down list, we indicate whether additional properties are used (True) or not used (False). The additional properties are disabled by default.
- **Maximum rows.** In the *Maximum rows* property, we specify the maximum number of rows to recover.
- **Invocation.** The Invocation property is used to indicate how to execute the action. By default, it's Interactive (by user request) although it can be configured as Batch (with a batch command) or On Open (when opening the cube).
- **Application.** The Application property can be used to describe the application of the action.
- **Description.** The Description property can be used to provide a description of the action.
- **Caption.** The Caption property can be configured to display the title of the action. It can be an MDX if the *Caption is MDX* property is configured properly.

- Caption is MDX. If the caption is an MDX, this property must be set to True. In this case, the content of the Caption field will be evaluated. If the caption isn't an MDX, this field must be set to False.

**Figure 7: Configuring the Additional Properties section for the drillthrough action**

After the drillthrough action is configured, we can access it from a client application, such as a dynamic table in Microsoft Excel. In this example, if we need to know the underlying details for the computer sales in 2007, we can double-click the cell. Alternatively, we can right-click the cell, choose Additional Actions, and select the action that we created, as shown in Figure 8.

**Figure 8: Using the drillthrough action to see the details for the computer sales in 2007**

Row Labels	Year 2007	Year 2008	Year 2009	Grand Total
Audio	€29.734.671,91			€151.614.364,31
TV and Video	€426.671.354,26			€1.360.121.114,76
Computers	€1.146.469.996,57	€990.173.504,69	€1.072.783.640,15	€3.209.427.141,42
Cameras and camcorders	€1.102.693.917,48			€2.562.023.774,06
Cell phones	€363.847.591,08			€892.233.264,30
Music, Movies and Audio Books	€74.975.760,83			€165.804.705,98
Games and Toys	€42.429.666,08			€149.696.456,86
Home Appliances	€1.375.117.996,81			€3.922.736.787,19
<b>Grand Total</b>	<b>€4.561.940.955,02</b>		<b>€3.119,18</b>	<b>€12.413.657.608,89</b>

This will generate a new sheet in the Excel workbook. As seen in Figure 9, the workbook will contain the details that we configured in the action and the columns that we selected for the drillthrough analysis.



**Figure 11: Trying to execute the drillthrough analysis on a calculated measure**

A	B	C	D	E
Sales Profit	Column Labels			
Row Labels	Year 2007	Year 2008	Year 2009	Grand Total
Audio	€17.336.920,45			€86.738.826,00
TV and Video	€226.644.269,68			€742.436.049,33
Computers	€651.122.146,36	€564.492.139,96	€609.545.621,63	€1.825.159.907,96
Cameras and camcorders	€666.989.743,58		740.954,52	€1.536.955.209,75
Cell phones	€203.458.975,53		33.064,69	€496.656.067,13
Music, Movies and Audio Books	€45.990.449,19		95.202,27	€100.870.824,27
Games and Toys	€23.746.689,51		52.945,91	€80.459.517,70
Home Appliances	€759.744.791,89		761.991,10	€2.179.484.604,96
<b>Grand Total</b>	<b>€2.595.033.986,21</b>		<b>142.325,83</b>	<b>€7.048.761.007,11</b>

## Transforming a Calculated Measure into a Physical Measure

When we want to drillthrough the details for a calculated measure, we can transform the calculated measure into a physical measure in the cube. We have several ways of doing this: in the tables in the relational database, in a view in the relational engine, and in the Data Source View (DSV) associated with the cube. In this case, we'll see how to transform it using a view in the relational engine. That way, we'll be able to access the data by a means other than SSAS. We'll have the same information, with the same business rules applied, in a source that we can consume directly.

For our example, let's suppose that, for business reasons, we're interested in monitoring the measures for a specific product. Listing 2 shows the code that creates a calculated measure for this product.

**Listing 2: Code that creates a calculated measure for a specific product**

```
CREATE MEMBER CURRENTCUBE.[Measures].[Sales Amount Product number1]
AS ([Product].[Product].[Product Name].&[1],
[Measures].[Sales Amount]),
NON_EMPTY_BEHAVIOR = {[Sales Amount] }, VISIBLE = 1 ;
```

To transform this calculated measure into a physical measure, we have several options. The following options are the easiest to implement:

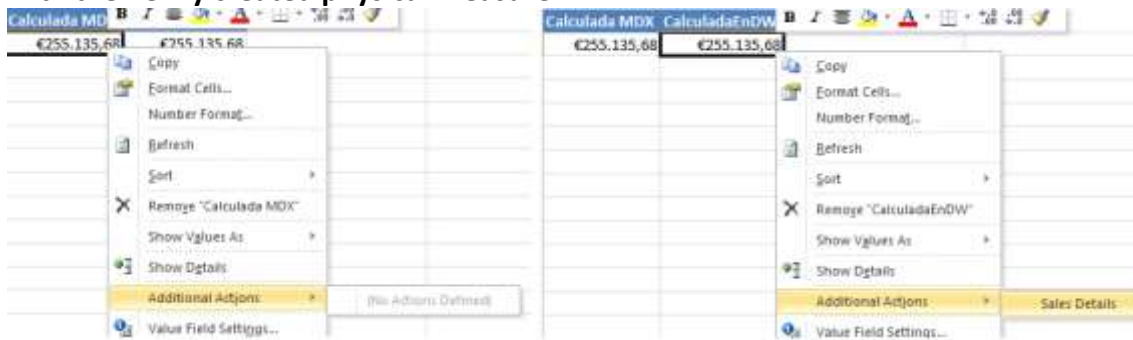
- Option 1. In the facts table, add a new column of type bit that fills during the extraction, transformation, and loading (ETL) process. In this column, the bit of 1 is assigned to the product that interests us and the bit of 0 is assigned to all the other products. This way, we can build a calculated column by multiplying the sales by this bit to obtain the sales for the desired product in a simple and fast way when reading the facts table.
- Option 2. Include all the logic for the calculated measure in the view. The disadvantage of this method is that all the logic must be followed during the data reading. However, we become independent of the ETL process, so if a business rule changes, we can apply the change to the view.

In our example we're going to use option 1 to represent the calculated measure in Listing 2 as a physical measure in a view in the relational engine. After creating the column of type bit (which is named ProductKey) in the facts table, we need to run the following code to add the calculated column (which is named SalesAmountProduct1:

```
case when (ProductKey=1)
then SalesAmount else 0 end
as SalesAmountProduct1
```

Subsequently, after the column is added to the DSV, a new measure will be generated in the measures group when we deploy and process the cube. At this point, we will have a physical measure to which we can apply the drillthrough action. In Figure 12, we see the calculated measure on the left and the physical measure we incorporated into the view on the right. Notice that only the physical measure has the drillthrough action.

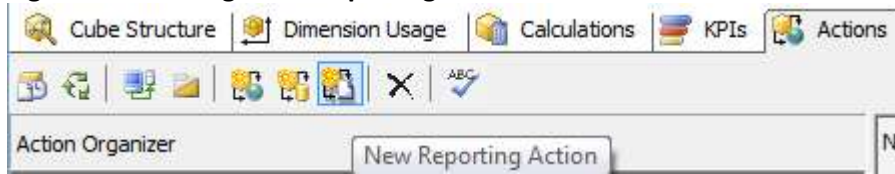
**Figure 12: Comparing the drillthrough actions available for the original calculated measure with the newly created physical measure**



## Using a Personalized Action

As we've already seen, we can add drillthrough actions to a multidimensional database, but there are other actions that we can use, such as the reporting action, as Figure 13 shows.

**Figure 13: Creating a new reporting action**



If we generate this kind of action, we again get a form to fill out, as shown in Figure 14. First, we specify the name of our reporting action, which is Sales Report.

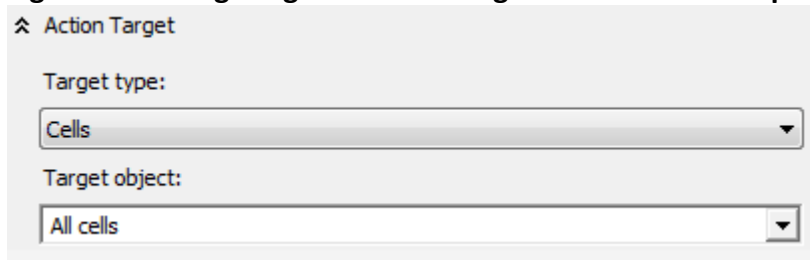
**Figure 14: The form for the new reporting action**

Name:

- ✖ Action Target
- ✖ Condition (Optional)
- ✖ Report Server
- ✖ Parameters (Optional)
- ✖ Additional Properties

In the Action Target section shown in Figure 15, we need to specify which cube elements to use for this action. First, we specify the type of target (in this case, the cells). Other options include dimensions, their attributes, hierarchies, or the cube itself. Once the type is defined, we specify the target object (in this case, all the cells).

**Figure 15: Configuring the Action Target section for the reporting action**



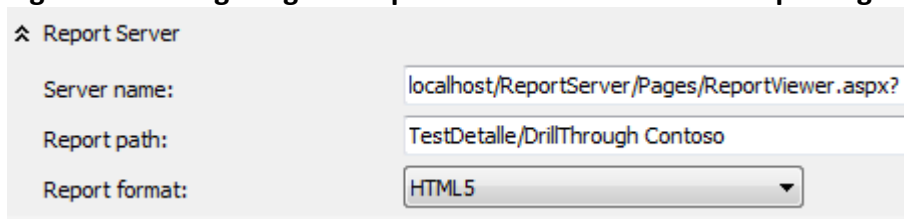
If we want to specify some condition (e.g., analyzing a specific year), we can use MDX syntax to specify it in the Condition (Optional) section seen in Figure 16.

**Figure 16: Configuring the Condition (Optional) section for the reporting action**



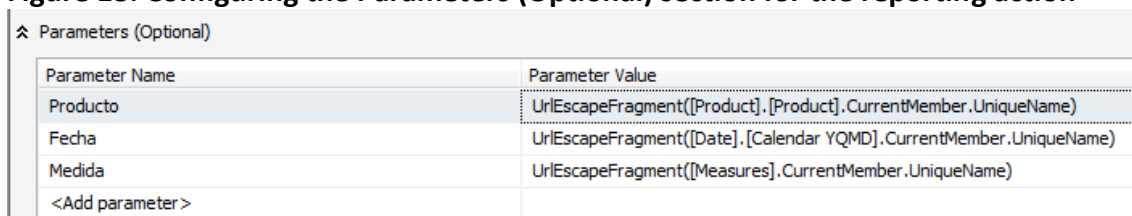
It's important to configure the Report Server section shown in Figure 17. In our case, the report server is on the local machine, so we include *localhost* in the name of the server, as seen in Figure 17. In the report path, we indicate where the report is located on the server (i.e., folder and report name).

**Figure 17: Configuring the Report Server section for the reporting action**



In the Parameters (Optional) section shown in Figure 18, we need to configure the source information—that is, in which measure we'll execute the action (in this case, Medida) and which members we're analyzing (in this case, Producto and Fecha). Later, we'll discuss the purpose of these values in more detail.

**Figure 18: Configuring the Parameters (Optional) section for the reporting action**



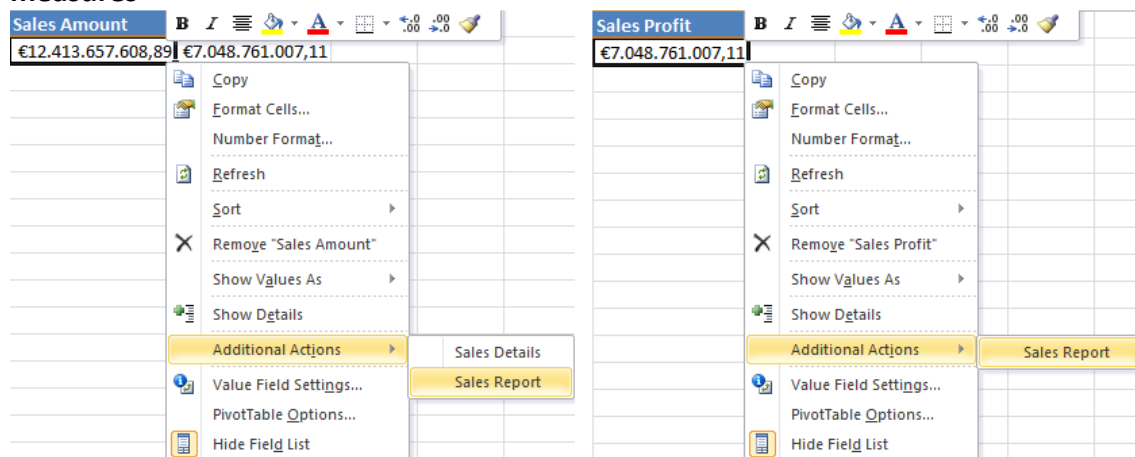
Parameter Name	Parameter Value
Producto	UrlEscapeFragment([Product].[Product].CurrentMember.UniqueName)
Fecha	UrlEscapeFragment([Date].[Calendar YQMD].CurrentMember.UniqueName)
Medida	UrlEscapeFragment([Measures].CurrentMember.UniqueName)
<Add parameter >	

We can configure other properties in the Additional Properties section shown in Figure 19. The properties in this section have already been explained in the “Using the Drillthrough Action in BIDS” section in this article.

**Figure 19: Configuring the Additional Properties section for the reporting action**

As seen in Figure 20, after the reporting action is configured, it appears as available for the physical measures (left) as well as for the calculated measures (right). However, we need to complete an additional step—create the report—before we can use it.

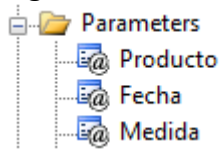
**Figure 20: Comparing the reporting actions available for the calculated and physical measures**



For this example, we’ll create the report in BIDS as a SQL Server Reporting Services (SSRS) project. We’ll configure a data source in our cube and create a new report named “Drillthrough Contoso.rdl.”

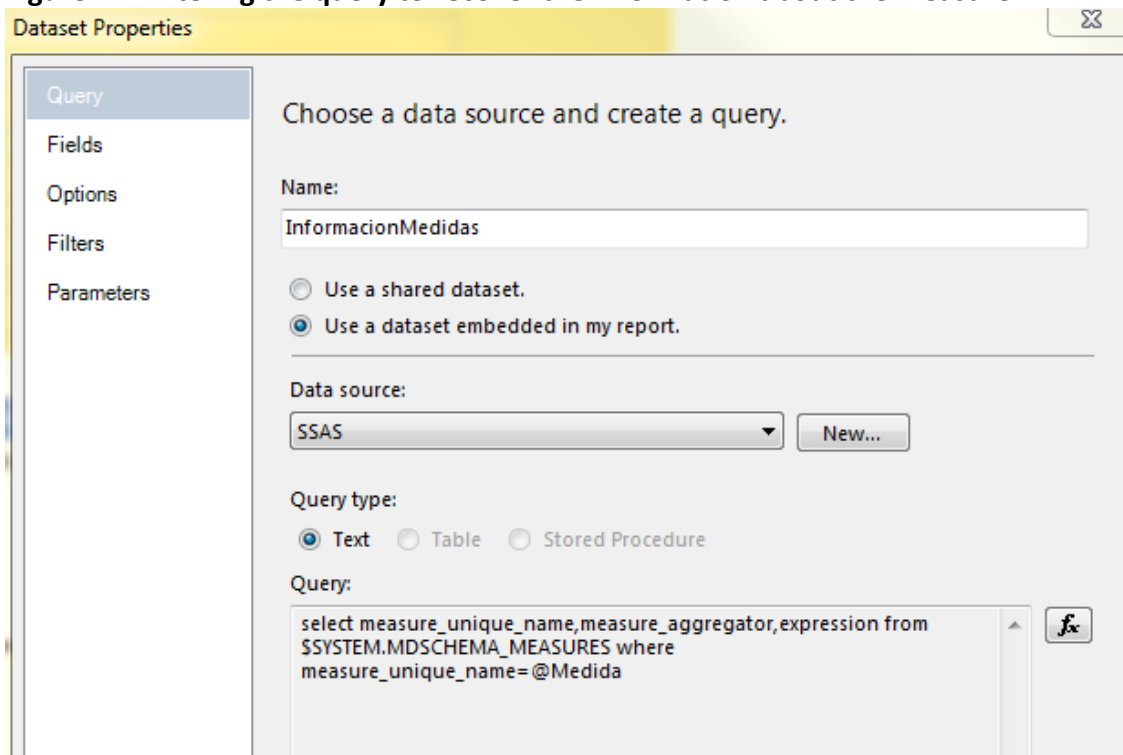
First, we must recollect the data that was transferred from the reporting action to the cube. In other words, we need to recollect the name of measure (Medida), the member of the Product hierarchy (Producto), and the member of the Calendar YQMD hierarchy (Fecha). We’ll begin by creating the three parameters shown in Figure 21 and establishing default values for each one of them.

**Figure 21: Creating the three parameters for recollecting the data**



Next, we'll build the dataset needed to recover the information about the measure from which the reporting action was executed. The name of that measure will be passed to a query as a parameter. This query will be run against an SSAS dynamic management view (DMV) named \$SYSTEM.MDSHEMA\_MEASURES to recover the metadata related to that measure from our cube. That query gets entered into Dataset Properties page, as Figure 22 shows.

**Figure 22: Entering the query to recover the information about the measure**



Let's take a closer look at the base query (i.e., the query without the filter):

```
select measure_unique_name,
measure_aggregator,
expression
from $SYSTEM.MDSHEMA_MEASURES
```

As you can see, the query gets information from three parameters: `measure_unique_name` (which contains the measure's unique name), `measure_aggregator` (which specifies the aggregated function used), and `expression` (which specifies the expression used). If we

execute this query in SQL Server Management Studio (SSMS), we obtain results like those shown in Figure 23.

**Figure 23: Sample results from running the base query**

```

select measure_unique_name,measure_aggregator,expression
from $SYSTEM.MDSHEMA_MEASURES
    
```

measure_unique_name	measure_aggregator	expression
[Measures].[Month Average Rate]	10	
[Measures].[End Of Day Rate]	14	
[Measures].[Sales Discount Amount Growth (Fiscal Year)]	127	case when [Measures].[Period on Period Growth Sale...
[Measures].[Sales Gross Margin]	127	[Measures].[Sales Amount]-[Measures].[Sales Total C...
[Measures].[Sales Gross Margin Growth (Fiscal Year)]	127	case when [Measures].[Period on Period Growth Sale...
[Measures].[Sales Amount Growth (Fiscal Year)]	127	case when [Measures].[Period on Period Growth Sale...
[Measures].[Sales Total Cost Growth (Fiscal Year)]	127	case when [Measures].[Period on Period Growth Sale...
[Measures].[Sales Gross Margin Growth (Calendar Year)]	127	case when [Measures].[Period on Period Growth Sal...
[Measures].[Sales Amount Growth (Calendar Year)]	127	case when [Measures].[Period on Period Growth Sale...
[Measures].[Sales Return Amount Growth (Calendar Year)]	127	case when [Measures].[Period on Period Growth Sale...
[Measures].[Sales Return Amount Growth (Fiscal Year)]	127	case when [Measures].[Period on Period Growth Sale...
[Measures].[Sales Total Cost Growth (Calendar Year)]	127	case when [Measures].[Period on Period Growth Sale...
[Measures].[Sales Discount Amount Growth (Calendar Year)]	127	case when [Measures].[Period on Period Growth Sale...
[Measures].[IT Machine Down Time]	1	
[Measures].[IT Machine Down Number Of Times]	2	

In the results, there are different values in the “measure\_aggregator” field, such as 1 (Sum), 2 (Count), 10 (AverageOfChildren), and 14 (LastNonEmpty). The value of 127 represents a calculated measure. Note that the measures associated with 127 have a value in the “expression” field, which indicates how they’re calculated. From these values we’ll generate a fourth field that lists the associated measures used in the calculation. As Figure 24 shows, the new field will be named listaMedidas. Its Field Source expression is:

```

=Code.CalculaListaMetricas
(Fields!expression.Value,
Fields!measure_aggregator.Value,
Fields!measure_unique_name.Value)
    
```

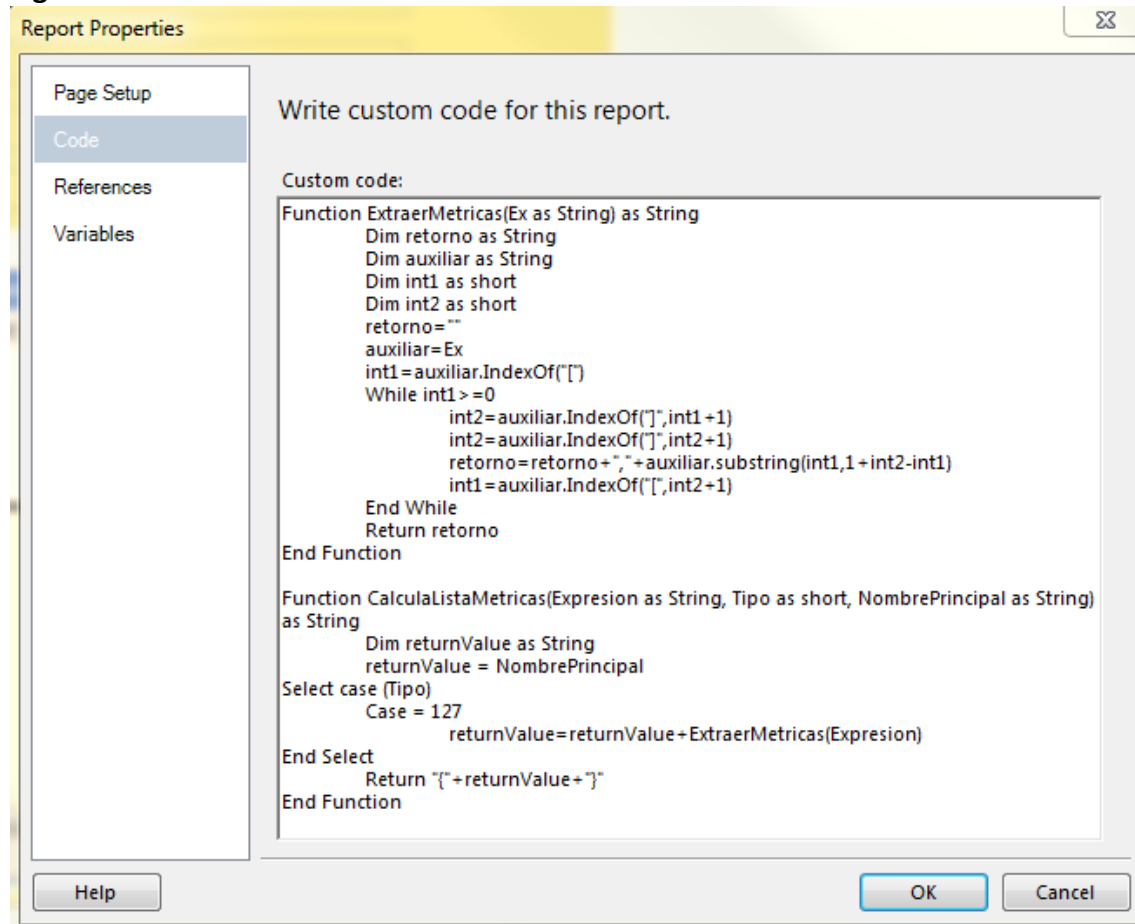
**Figure 24: Adding a fourth field**

Field Name	Field Source
measure_unique_name	measure_unique_name
measure_aggregator	measure_aggregator
expression	expression
listaMedidas	«Expr»

The list of associated measures is created by a function that’s defined in the Code tab of the Report Properties page, as Figure 25 shows. This function processes the value returned in the “expression” field, extracting each pair of bracketed words (e.g., [Measures].[Sales

Amount]). These pairs, together with the original measure, will be returned, separated by commas.

**Figure 25: The function that creates the list of associated measures**



Now we need a second dataset that recovers the source data. In this case, we will show all the measures related to sales. Therefore, the query looks like the one in Listing 3. This query receives the date of operation (Fecha) and the product (Producto) as parameters. The data analysis must match up with the analysis we're carrying out when invoking the reporting action. In other words, it needs the same member of the Product hierarchy and the same member of the Calendar YQMD hierarchy. If we don't do it this way, we could receive unexpected results.

### Listing 3: The query for the second dataset

```

select non empty {
[Measures].[Sales Unit Cost],
[Measures].[Sales Unit Price],
[Measures].[Sales Quantity],
[Measures].[Sales Return Quantity],
[Measures].[Sales Return Amount],
[Measures].[Sales Discount Quantity],
[Measures].[Sales Discount Amount],
[Measures].[Sales Amount],
[Measures].[Sales Total Cost]} on columns,
non empty {[Product].[Product].[Product Name].allmembers*[Date].[Calendar YQMD].[Date].allmembers} on
rows
from (
select {strtomember(@Fecha)} on 0 from (
select {strtomember(@Producto)} on 0 from [Operation])
    
```

To clearly see how the query is behaving, we need a series of expressions in the report that represent the metadata with which we are working. For example, Figure 26 shows the report details from executing the query for a physical measure. As you can see in the first three lines, the selected product is member 1 of the Products dimension, the report was run on January 12, 2009, and the report was launched from the Sales Amount measure. The next two lines show that there was only one associated measure (Sales Amount) and no expressions involved.

**Figure 26: Report details from running the query for a physical measure**

#### Report Details:

Selected Product: [Product].[Product].[Product Name].&[1]

Selected Date: [Date].[Calendar YQMD].[Date].&[20090112]

Source Measure: [Measures].[Sales Amount]

Associated measures: {[Measures].[Sales Amount]}

No expression associated

If we execute the query for the Sales Profit calculated measure, we'll see the report details shown in Figure 27. In this case, the first three lines are similar, but there are more associated measures in the fourth line (Sales Amount, Sales Profit, and Sales Total Cost). The last line shows the calculation used to create the calculated measure. In this case, we can see it was a simple subtraction.

**Figure 27: Report details from running the query for a calculated measure**

**Report Details:**

Selected Product: [Product].[Product].[Product Name].&[1]

Selected Date: [Date].[Calendar YQMD].[Date].&[20090112]

Source Measure: [Measures].[Sales Profit]

Associated measures: {[Measures].[Sales Profit],[Measures].[Sales Amount],[Measures].[Sales Total Cost]}

Calculation: [Measures].[Sales Amount] - [Measures].[Sales Total Cost]

Now we have to show the information that is returned. For that, we'll create a table in the report that has one column for each field returned from our second dataset. To make the information more legible, we'll use blue as the background color for the dimension-related column headings and yellow as the background color for the measure-related column headings. Moreover, for easy monitoring, we'll highlight the associated measures that were used by adding yellow shading to those cells. Another possibility would be to show only the associated measures that were used.

For example, suppose you run the report for the Sales Profit calculated measure for a certain product on a certain day and you receive the table shown in Figure 28. (Because of the row's length, it has been wrapped in this figure.) From the yellow shading in the 17598 and 8092.6 cells, you know that Sales Profit has been calculated using the Sales Amount and Sales Total Cost associated measures.

**Figure 28: Sample table in a report**

Product Category Name	Product SubCategory Name	Product Name	Date	Sales Unit Cost	Sales U		
Computers	Laptops	Fabrikam Laptop17W M7080 Black	2007-04-01	404.63			
Unit Price	Sales Quantity	Sales Return Quantity	Sales Return Amount	Sales Discount Quantity	Sales Discount Amount	Sales Amount	Sales Total Cost
879.9	20	0	0	0	0	17598	8092.6

We would like to thank our SolidQ colleague Idefonso Mas for his invaluable collaboration in the elaboration of the personalized action approach.

## Transforming a Subtotal into a True Cell

So far we have seen how to get the detail of a cell from a cube by defining the cell as a coordinate of the cube, where each member of each dimension affecting a measure intersect. Therefore, the MDX query associated with that intersection returns a value only. However, there are situations in which MDX queries can return a value, but that value isn't a true cell. Instead, it's a subtotal obtained by filtering several elements from one or more dimensions. In these cases, we're not seeing the stored value associated with the "All" member of the dimension but rather a value generated by the aggregation of the selected elements. Thus, we need a special way to access the details of a subtotal.

Suppose that we run the query in Listing 4 to see the internet sales that have been made in France, Germany, the United Kingdom, and the United States, together with the general total. As the results in Figure 29 show, the “All customers” member returns the value of all the sales and not the total of only the countries selected. This occurs because “All customers” is another member—typically, the member that’s used by default when we don’t select another group of members of a dimension.

#### Listing 4: Query to see the internet sales

```
SELECT [Measures].[Internet Sales Amount] on columns,
{[Customer].[Customer Geography].[Country].&[France],
[Customer].[Customer Geography].[Country].&[Germany],
[Customer].[Customer Geography].[Country].&[United Kingdom],
[Customer].[Customer Geography].[Country].&[United States],
[Customer].[Customer Geography].[All Customers]} ON rows
FROM [Adventure Works];
```

**Figure 29: Results from the query to see the internet sales**

	Internet Sales Amount
France	2,644,017.71 \$
Germany	2,894,312.34 \$
United Kingdom	3,391,712.21 \$
United States	9,389,789.51 \$
All customers	29,358,677.22 \$

To obtain the sales total for France, Germany, the United Kingdom, and the United States, we have to modify the query to “filter” the cube by those members. For this, we have two options: using the WHERE clause or attaching a subject that returns a subcube with the information from those countries. The code in Listing 5 demonstrates both approaches. In both cases, the query returns a unique value, which Figure 30 shows.

**Listing 5: Code that demonstrates the two ways to filter the cube**

```

SELECT [Measures].[Internet Sales Amount] on columns
FROM (
SELECT {[Customer].[Customer Geography].[Country].&[France],
[Customer].[Customer Geography].[Country].&[Germany],
[Customer].[Customer Geography].[Country].&[United Kingdom],
[Customer].[Customer Geography].[Country].&[United States]} ON COLUMNS
FROM [Adventure Works]);

SELECT [Measures].[Internet Sales Amount] on columns
FROM [Adventure Works]
WHERE {[Customer].[Customer Geography].[Country].&[France],
[Customer].[Customer Geography].[Country].&[Germany],
[Customer].[Customer Geography].[Country].&[United Kingdom],
[Customer].[Customer Geography].[Country].&[United States]};
    
```

**Figure 30: Results from the code that filters the cube**

Internet Sales Amount
18,319,831.77 \$

At this point, we might think that we can just add the DRILLTHROUGH statement and we'll have our details. However, when we do this, we receive an unexpected result in both approaches. When we use the DRILLTHROUGH statement with the WHERE clause code, we get an error indicating that it isn't able to find the coordinate that has that value. Figure 31 shows the error message.

**Figure 31: Error message indicating the DRILLTHROUGH statement failed**

Drillthrough failed because the coordinate identified by the SELECT clause is out of range.

When we use the DRILLTHROUGH statement with the subcube code, data is returned but that data isn't what we expected. For example, take a look at the results from using the DRILLTHROUGH statement with a subcube in Figure 32. Notice how it returns data from Australia, which isn't one of our selected countries. The problem basically is that the subcube generated in the FROM clause stays inside the query field only, so the DRILLTHROUGH statement returns the details that the "All Customers" member generates for the whole cube.

**Figure 32: Sample results from using the DRILLTHROUGH statement with a subcube**

[Date]	[\$Sales Territory]	[\$Sales Territory]	[\$Delivery Date]	[\$Sales Amount]
15	Southwest	United States	July 27, 2005	N
15	Southwest	United States	July 27, 2005	N
15	Northwest	United States	July 27, 2005	N
15	Northwest	United States	July 27, 2005	N
15	Southwest	United States	July 28, 2005	N
15	Southwest	United States	July 28, 2005	N
15	Germany	Germany	July 28, 2005	N
15	Germany	Germany	July 28, 2005	N
15	Australia	Australia	July 26, 2005	N
15	Australia	Australia	July 26, 2005	N

This situation is the same for Excel when we try to access to the details of a total or subtotal that has been created through a multiselection. In this situation, Excel shows an error message and doesn't give us access to the details. Thus, we have to compile it the information manually.

How can we solve this problem? In the same way we had to transform a calculated measure into physical measure in order to get its details, we have to transform the subtotal in a true cell. To carry out that transformation, we can use code like that in Listing 6. In this code, we create a subcube delimited by the selected countries. Next, we apply the DRILLTHROUGH statement to this subcube. Because the "All customers" member will only be associated with the countries that we used to create the subcube, the code will return the results that we expected. Finally, we drop the subcube once we obtain the details. It's important to drop the subcube so that we again have access to all the information in the original cube.

### Listing 6: Code that transforms the subtotal into a true cell

```

CREATE SUBCUBE [Adventure Works] AS
SELECT {[Customer].[Customer Geography].[Country].&[France],
[Customer].[Customer Geography].[Country].&[Germany],
[Customer].[Customer Geography].[Country].&[United Kingdom],
[Customer].[Customer Geography].[Country].&[United States]} ON COLUMNS
FROM [Adventure Works];

// Obtain the "All customers" member, which will contain the filtered total.
SELECT [Measures].[Internet Sales Amount] on columns,
[Customer].[Customer Geography].[All Customers] on rows
FROM [Adventure Works];

// Execute the DRILLTHROUGH statement on "All customers" to obtain the correct data.
DRILLTHROUGH
SELECT [Measures].[Internet Sales Amount] on columns,
[Customer].[Customer Geography].[All Customers] on rows
FROM [Adventure Works];

// Do not forget to drop the subcube.
drop subcube [Adventure Works];
    
```

Note that this problem can occur in dynamic Excel tables when we try to access the details of a subtotal or total that has been obtained by filtering several elements. In this situation, Excel shows an error message and doesn't give us access to the details. There are two ways to solve this problem in Excel. We can compile the details manually, or we can create an add-in that carries out the following steps:

1. Collect the filters applied to the dynamic table.
2. Generate the code that will create the subcube, run the drillthrough operation, and delete the subcube.
3. Use the dynamic table's connection to execute the code.
4. Collect the rows returned by the drillthrough operation and display them in a new worksheet.

## Conclusion

Although aggregated information is important, users often need to access the details behind that information in order to correctly understand the information that they're seeing. In this article, we saw the types of problems that users can encounter when they try to access the details. We also saw how providing access to those details isn't always easy. When the default actions and the functionalities don't provide the drillthrough capabilities that users need, we must know how to overcome the limitations in the best possible way.

## About the Authors



**Javier Torrenteras** ([blog](#) | [twitter](#)) is a SolidQ mentor focused on business intelligence (BI) with the Microsoft Data Platform. He has more than 14 years of IT experience working on a large variety of projects, including client/server and web development, management, and BI. He has extensive experience with SQL Server Reporting Services and SQL Server Analysis Services, including using them to design and build several datamarts. Javier is an MCP on Designing Business Intelligence Infrastructure Using SQL Server 2008. He is currently the BI director of the Spanish subsidiary of SolidQ.



**Carlos Martinez** works at SolidQ in the business intelligence (BI) area as a data platform engineer. He has a master's degree in Computing Science. He has worked on many projects, including an embedded systems project in collaboration with University of Alicante and a Microsoft Dynamics AX project, getting certified in one aspect of this technology. Since then he has continued working in the BI area, helping several telecommunications companies. He has collaborated with Microsoft Iberica Spain, conducting a webcast about server mode enhancements in Report Builder 3.0 as part of a SQL Server 2008 R2 launch event.



Your data is trying to tell you something — are you listening?

# Business Intelligence

[www.solidq.com](http://www.solidq.com)