Applies to:
SAP BusinessObjects XI 4.0 Support Package 4, the information design tool. For more information, visit the Business Objects homepage.

Summary
This paper is discussing the different applications of the new function @Execute that was introduced in the SAP BusinessObjects BI Suite 4.0 Support Package 4. We will see through concrete examples how the stepwise approach that the @Execute function permits allows us to address questions that cannot be answered using a single SQL query or to accelerate a query by breaking it down into two easier queries. We will start with a use case on a Microsoft SQL Server database to describe how the @Execute function works. Then we will show different scenarios against SAP HANA, Teradata and Microsoft SQL Server again. You can find the resources that were used for this article here.

Author: Marc Daniau
Company: SAP
Created on: 17 December 2012

Author Bio
Marc Daniau joined Business Objects in 1992 as project consultant in France. He moved to the product group in San Jose in 1998 to work on EPM products. He moved back to Paris in 2003 to work within the semantic layer team.
Table of Contents

The @Execute function explained ........................................................................................................ 3
Working with an SAP HANA analytical view ...................................................................................... 5
  Time filter ............................................................................................................................................ 5
  User-based filter ................................................................................................................................. 10
  Percent of total .................................................................................................................................. 11
Trend Alert with Teradata .................................................................................................................... 13
Team Members ...................................................................................................................................... 15
Related Content ..................................................................................................................................... 17
Copyright ............................................................................................................................................ 18
The @Execute function explained

In order to explain how the @Execute function works we chose a typical BI use case that is identifying and understanding the products that perform best. Rather than trying to answer the business question with one single SQL query we break down the question and proceed in two steps.

In the first step we run a query that lists the primary keys of the best performing products. That preliminary query is formulated through a SQL list of values in the data foundation or in the business layer. It encapsulates the business rule that defines what an over-performing product is and therefore the query will be more or less complex depending on that rule. A simple way of evaluating products performance is to extract the top ten products based upon the total amount sold. Another way is to find the products which total sales exceed the sales average of their category. We will implement the latter rule in our coming exercise.

In the second step we run a subsequent query that returns the product related attributes and/or facts filtered with the product keys obtained from the first step. That main query is expressed through the query panel in a usual fashion.

Let us put our example into practice using a Microsoft SQL server database that stores sales orders. Here is what the data foundation looks like in the information design tool.

Within that data foundation we define a SQL list of values in order to obtain the products which total ordered amount is two times above the average of their respective categories.
Then in the business layer we add a filter that invokes the list of values through the @Execute function.

![Filter: Products 2 times above their category average](image)

When used in the query panel, the filter will be automatically and under the cover transformed by the Business Objects query engine into the following SQL predicate:

```
PRODUCT.PRODUCT_ID in (35, 33, 2, 5, 20)
```

Here is an example of query using the product filter:

<table>
<thead>
<tr>
<th>Category Name</th>
<th>Line Name</th>
<th>Product Id</th>
<th>Product Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tennis</td>
<td>Sports</td>
<td>2</td>
<td>Slamit Tennis Racket</td>
</tr>
<tr>
<td>Golf</td>
<td>Sports</td>
<td>5</td>
<td>Berta Golf Clubs</td>
</tr>
<tr>
<td>Home Theater</td>
<td>Electronics</td>
<td>20</td>
<td>Speaker System</td>
</tr>
<tr>
<td>Appliances</td>
<td>Electronics</td>
<td>33</td>
<td>Dishwasher</td>
</tr>
<tr>
<td>Bikes</td>
<td>Sports</td>
<td>35</td>
<td>Descent Competition Bicycle</td>
</tr>
</tbody>
</table>

We found five products that over performs relative to their category average.

One might think that an alternative solution is to define a derived table “Products 2 times above average”. Such derived table will not work knowing that Microsoft SQL Server like other databases does not support a sub-query that contains a WITH clause.

The backup file SPL_Warehouse.bak of the database we used for this example can be downloaded from: [http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/7029cd48-01a3-2e10-b18e-b770c9f37a57?QuickLink=index&51887500359053](http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/7029cd48-01a3-2e10-b18e-b770c9f37a57?QuickLink=index&51887500359053)
Working with an SAP HANA analytical view

Throughout this chapter we will use data from SAP Profitability Analysis (CO-PA) a sub module of the controlling (CO) module. Using the SAP HANA Studio, we modeled the analytical view CEA1 on top of the CE1IDEA table.

Time filter

Among the most common requests related to time is the ability to get the current period based upon the system date, so that one can automatically fetch the latest data. Let us see how we can build a current period filter on a CO-PA analytical view.

In addition to the CEA1 analytical view, we store the business periods with their date intervals in a SAP HANA column table.

<table>
<thead>
<tr>
<th>Name</th>
<th>SQL Data Type</th>
<th>Dim</th>
<th>Column Store Data Type</th>
<th>Key</th>
<th>Not Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERIOD</td>
<td>NVARCHAR</td>
<td>7</td>
<td>STRING</td>
<td>X(1)</td>
<td>X</td>
</tr>
<tr>
<td>YEAR</td>
<td>NVARCHAR</td>
<td>4</td>
<td>STRING</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PER_OF_YEAR</td>
<td>NVARCHAR</td>
<td>3</td>
<td>STRING</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>BEGIN_DATE</td>
<td>DATE</td>
<td></td>
<td>DAYDATE</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>END_DATE</td>
<td>DATE</td>
<td></td>
<td>DAYDATE</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Using the information design tool, we include the SAP HANA analytical view in a data foundation.

In the business layer, we create the following filter that for now does not use the @Execute function. The filter consists of a SQL predicate involving a non-correlated sub-query.

```sql
WHERE @Select(CO-PA Actuals:\Period) = ( SELECT PERIOD FROM MDA.PER WHERE BEGIN_DATE <= NOW() AND END_DATE >= NOW() )
```
We define a query to get the net revenue by material and sales organization for the current period.

<table>
<thead>
<tr>
<th>Material Id</th>
<th>Material Description</th>
<th>Sales Organization</th>
<th>Net Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We copy the SQL script generated behind the scene by the information design tool, and go back to SAP HANA Studio in order to obtain the explain plan of the above query.

The query plan does not look optimal. The question is: can we make the query simpler? Now comes the time to implement the two-step approach. For that we define the first-step query in a SQL list of values in the data foundation. The goal here is to retrieve the current period.

The SQL List of Values: CUR_PERIOD

We modify the filter “Current Period” so that it references the list of values we have just defined.

Such filter if you run the Validate command will return an error because the information design tool parser assumes multiple values. However the filter will work when used in a query. In order to get rid of the validation error, one can substitute the equal operator with the IN operator.
After copying the SQL scripts of the list of values (1st step) and the main query (2nd step) and after analyzing them in SAP HANA Studio we obtain the following plans.

Explain plan for the 1st-step query

<table>
<thead>
<tr>
<th>OPERATOR_NAME</th>
<th>OPERATORDETAILS</th>
<th>EXECUTIONENGINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN SEARCH</td>
<td>PER_PERIOD</td>
<td>COLUMN</td>
</tr>
<tr>
<td>COLUMN TABLE</td>
<td>FILTER_CONDITION: TO_TIMESTAMP(PERBEGIN_DATE) &lt;= CURRENT_TIMESTAMP...</td>
<td>COLUMN</td>
</tr>
</tbody>
</table>

Explain plan for the 2nd-step query

<table>
<thead>
<tr>
<th>OPERATOR_NAME</th>
<th>OPERATORDETAILS</th>
<th>EXECUTIONENGINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN SEARCH</td>
<td>TABLE_1.MATNR, TABLE_1.MATNR.DESCRIPTION, TABLE_1.VKORG, SUM(TABLE_1.NETRE...</td>
<td>COLUMN</td>
</tr>
<tr>
<td>Aggregation</td>
<td>TABLE_1.MATNR, TABLE_1.MATNR.DESCRIPTION, TABLE_1.VKORG, AGGREG...</td>
<td>COLUMN</td>
</tr>
<tr>
<td>COLUMN VIEW</td>
<td>FILTER_CONDITION: TABLE_1.PERIOD = '2012011'</td>
<td>COLUMN</td>
</tr>
</tbody>
</table>

The implementation of the @Execute function in this example results in simpler queries thus improving the query performance.

Beyond the filter “Current Period”, the business user needs a “Last n Periods” filter in order to track the latest trend. That can be achieved with another filter. Let us see how.

We first enter in the data foundation a list of time window sizes that the business user can choose from when prompted for the number of periods required in his report.

![Static list of values: TIME_WINDOW_SIZE](image)

We then define, still in the data foundation, the prompt requesting a number of periods.
We set the default value to 12.
We define a SQL list of values that retrieves the last \( n \) periods, \( n \) being the number of periods supplied by the answer to the prompt that we created previously.

Finally, we add in the business layer the following filter.

That filter can be used in the query panel the same way we used the “Current Period” filter in our first example.
There are many kinds of time windows needed for BI purposes. We just scratched the surface with the above examples. One can go further by defining time windows in a table as we did in SAP HANA Studio with the PER_VIEW column table presented next.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Schema</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PER_VIEW</td>
<td>MDA</td>
<td>Column Store</td>
</tr>
</tbody>
</table>

The begin and end periods defining the period interval are stored together in the column RANGE as if it was an array of two periods in order to make only one call to the table. Let us define a SQL list of values to get the types of time views available.

```
SELECT distinct VIEW FROM MDA.PER_VIEW ORDER BY 1
```

We then define a prompt allowing the end-user to select the time view needed. We decided to choose Year-To-Date as the default time view.
Using another SQL list of values, we write the select statement that returns the range of periods corresponding to the current period and for a given time view. In the illustration below we answered Year-To-Date to the prompt from the Preview command.

Finally we invoke the SQL list of values TIME_VIEW through a filter using the @Execute function.

When invoking the same @Execute command twice as shown in the above expression, the underlying query is run only once and the replacement in the filter expression is done twice by the Business Objects engine.

User-based filter

Another requirement is to restrict the data based upon the logged-on user. The need may be to simply help the user focus on the data corresponding to his scope of responsibility or interest. Or for security reason we need to limit the user to a given subset of the data. We will develop next how that can be achieved using the @Execute function.

Our SAP HANA database includes a column table that lists for each BI user the sales organizations he has access to.
We define a SQL list of values that restricts the USER_SORG table using the system variable BOUSER. The output of that query is the list of ids for the organizations that the current user is allowed to see.

We can now reference this restricted list from a filter using the @Execute function.

If needed, we can enforce the user-based filter by making it mandatory. If we do, any query based on that universe will be restricted with respect to the sales organizations that were granted to the BI user.

Percent of total
The use cases we have seen so far involve the new @Execute function for filtering purposes. However filtering is not the only application of the @Execute function as we are going to see next.

The business user requests a report that evaluates how much his own product portfolio contributes to the revenue of the company. We can answer that question by computing first the overall revenue using a list of values like this.
We then invoke that list of values in a measure.

![Measure: Net Grand Total](image)

We explicitly specified the analytical view in the Extra Tables box. When validating the measure alone SAP HANA will return an error, but when used in a query along with a measure that uses an aggregation function, the measure will work as expected as shown below.

![Data Preview](image)

From WebI, we apply a local formula to get the ratio of net revenue over the grand total.

![Local formula](image)
Trend Alert with Teradata

The trend alert scenario we will cover in this chapter is about finding time series patterns like cost increasing n times in a row or sales decreasing n times in a row. In order to perform such analysis we will leverage the Teradata SQL extension: RESET WHEN.

Our Teradata schema includes shop sales measures by article and by week. Assuming we have a relational connection set up against the Teradata server, we build a data foundation as the one shown below.

Then, still within the data foundation, we define a SQL list of values to get the ids of the articles that have a shrinking margin over weeks nine times in a row.

Finally in the business layer, we define a filter that references the list of values we have just created.
Here is the query showing the articles which margin has decreased 9 weeks in a row.

<table>
<thead>
<tr>
<th>Article Id</th>
<th>Article Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>158152</td>
<td>Blazer</td>
</tr>
<tr>
<td>159705</td>
<td>Jet Pearl Choker</td>
</tr>
<tr>
<td>167042</td>
<td>Long-Sleeved Stitch Shirt</td>
</tr>
<tr>
<td>168230</td>
<td>Lycra Trousers</td>
</tr>
<tr>
<td>160556</td>
<td>Spotty Leggings</td>
</tr>
<tr>
<td>171910</td>
<td>Violon T-Shirt</td>
</tr>
</tbody>
</table>

If we check the trend data by computing the number of decreases in a row for the article #160556, we observe that it indeed went down 9 times in a row.

<table>
<thead>
<tr>
<th>WEEK_ID</th>
<th>MARGIN</th>
<th>DECREASES_IN_A_ROW</th>
</tr>
</thead>
<tbody>
<tr>
<td>181</td>
<td>681.00</td>
<td>0</td>
</tr>
<tr>
<td>182</td>
<td>1153.10</td>
<td>0</td>
</tr>
<tr>
<td>183</td>
<td>1079.70</td>
<td>1</td>
</tr>
<tr>
<td>184</td>
<td>722.10</td>
<td>2</td>
</tr>
<tr>
<td>185</td>
<td>521.00</td>
<td>3</td>
</tr>
<tr>
<td>186</td>
<td>390.20</td>
<td>4</td>
</tr>
<tr>
<td>187</td>
<td>307.40</td>
<td>5</td>
</tr>
<tr>
<td>188</td>
<td>209.70</td>
<td>6</td>
</tr>
<tr>
<td>189</td>
<td>203.80</td>
<td>7</td>
</tr>
<tr>
<td>190</td>
<td>104.00</td>
<td>8</td>
</tr>
<tr>
<td>191</td>
<td>72.80</td>
<td>9</td>
</tr>
</tbody>
</table>
Team Members

Another application of the `@Execute` function is to extract a subset from a parent-child hierarchy like the team members of a given employee for instance. To illustrate that use case, we chose the Microsoft demo database ‘Adventure Works’ which includes among many other tables the DimEmployee table.

First of all we insert in the data foundation the FactResellerSales table with some of its related tables including the DimEmployee table.

We then define a SQL list of values that retrieves the list of supervisors.

![SQL List of Values: SUPERVISORS](image)

We create a prompt relying on that list of values.

![Parameter: Supervisor](image)

We define a second SQL list of values in order to find the employees that belong to the team of a given employee chosen from the supervisor list.
In the example above, we found the employees from Amy Alberts' team. Instead of prompting the supervisor to be chosen we could use the system variable BOUSER as we did in our SAP HANA example thus allowing us to provide a filter 'My Team' based upon the logged on user.

Finally we build a filter using the @Execute function.

When used in a query, the filter will ask the end-user to choose a supervisor as illustrated below.
Related Content

Create a Relational Connection to SAP HANA

Create a data foundation based on an SAP HANA view information design tool - eLearning

For more information, visit the Business Objects homepage.
Copyright

© Copyright 2012 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Excel, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, System i, System i5, System p, System p5, System x, System z, System z10, System z9, z10, z9, iSeries, pSeries, xSeries, eServer, z/VM, z/OS, i5/OS, S/390, OS/390, OS/400, AS/400, S/390 Parallel Enterprise Server, PowerVM, Power Architecture, POWER6+, POWER6, POWER5+, POWER5, POWER, OpenPower, PowerPC, BatchPipes, BladeCenter, System Storage, GPFS, HACMP, RETAIN, DB2 Connect, RACF, Redbooks, OS/2, Parallel Sysplex, MVS/ESA, AIX, Intelligent Miner, WebSphere, Netfinity, Tivoli and Informix are trademarks or registered trademarks of IBM Corporation.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP Business ByDesign, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects S.A. in the United States and in other countries. Business Objects is an SAP company.

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.