Chapter No. 6
"You Talking to Me? – Scripted Calculation Views"
In this package, you will find:
A Biography of the author of the book
A preview chapter from the book, Chapter NO.6 "You Talking to Me? – Scripted Calculation Views"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Mark Walker has been working with computers since a very early age, starting out with programming the Sinclair ZX Spectrum. After graduating from Cardiff University, he moved to France, starting a 17-year (so far) career in SAP development, first on the SAP ERP, then moving to SAP BW in 2004. He specializes in process optimization, especially in the SAP BPS and BI-IP modules. He’s been following SAP HANA since its release in 2011, and is a certified HANA Application Associate since early 2012.

Mark has been working for the last six years for Censio, a consultancy in Paris, France, which has expertise in SAP, Microstrategy, Business Objects, and Microsoft BI technologies. During that time he's worked on several end-to-end projects for some prestigious clients, and done some missions in SAP’s name.


I would like to thank all those people who have helped me throughout my career, and especially my wife, who puts up with the many hundreds of hours I spend in front of a keyboard, tinkering.

For More Information:
Welcome to the world of development with SAP HANA. More than just the latest technological buzzword, even more than the result of SAP's marketing effort, SAP HANA is a complete development system, including a database system, complete development environment, and application server.

While SAP HANA is often considered as an extension of SAP's BW data warehouse, and more recently of their ERP system, in this book we will see that it can be used for pure development purposes, and indeed, provides tools ideally suited to this use case.

From loading data to modeling it into reusable objects through authorizations, creating reporting applications, and developing a website with the system, the journey we will be taking together will allow us to explore the multifaceted system that is SAP HANA.

What This Book Covers

Chapter 1, So, What Is This SAP HANA Thing Anyways?, introduces SAP HANA and examines the characteristics that make it special.

Chapter 2, SAP HANA Studio – Installation and First Look, describes the SAP HANA Studio—the software you will use throughout the book, and every day you work with SAP HANA—to do database modeling and development work. You will also discover three different ways of getting access to a SAP HANA server.

Chapter 3, Your First SAP HANA Development – An Attribute View, starts your development journey with SAP HANA, building an attribute view, the first building block of database development with the system. Your attribute view will use data extracted from several tables, joined together to provide description for a customer in the system, in a coherent way.

Chapter 4, Painting with Numbers – An Analytic View, will create the next step of SAP HANA database development—an analytic view, showing how this object can use and present numeric information, combined with the description of this information provided by an attribute view.

For More Information:
Chapter 5, Let's Get Graphical – Graphical Calculation Views, continues our database development work, introducing the notion of graphical calculation view, which is an object providing aggregation and combination functionality. This object can use data from one or more analytic views to present it to your users for consumption.

Chapter 6, You Talking to Me? – Scripted Calculation Views, is where you will learn that in SAP HANA, for each function you can perform using the graphical designers provided by Studio, there is a way of doing the same job by writing code. In this chapter, you will create a calculation view using SQL Script, the SAP HANA SQL scripting language. This calculation view will fulfill the same function as the one you created in Chapter 5, Let's Get Graphical – Graphical Calculation Views.

Chapter 7, Hey! That's My Data! – Authorizations in SAP HANA, introduces the notion of authorizations in the SAP HANA system, allowing you, as a developer, to dictate who can access which object in the database, and even which values (that you have created in the different modeling views) can be seen by which of your users. You will learn that user groups can be created by developing authorization roles; you will create a database user to be able to test the authorizations you create.

Chapter 8, On Another Level – Hierarchies in SAP HANA, examines the notion of hierarchies as they apply to data visualization, showing the different hierarchy types you can create with SAP HANA. You will learn how a hierarchy can affect the data your users can see, providing automatic subtotals and aggregation.

Chapter 9, Deploying your Reporting Application to Reporting Software, will show how you can deploy the SAP HANA data models that you will create in three different tools: SAP's Lumira, Tableau, and Microsoft Excel so that your users can analyze their data. You will see the differences between the three tools, both in their approach, their capabilities, and their requirements.

Chapter 10, Data Provisioning Using Data Services, explains how you can use another of SAP's tools, Business Objects Data Services, to import data into your SAP HANA instance in an easy way.

Chapter 11, Application Development Using the XS Engine, is where you will learn how you can create web applications using SAP HANA's inbuilt application server, the XS Engine. From a simple "Hello, World" application to database access with your web-based application, to user interface development with SAP's SAPUI5 HTML-based framework, you will learn all the techniques needed to develop Internet or intranet applications on the SAP HANA system. You will also learn about accessing the data in the SAP HANA database from your XS Engine applications.

Appendix, So Long and Thanks – Where to go from here, gives an overview of all that you will have learned throughout the book, and gives some tips and pointers on your possible next steps in the SAP HANA development world.
Writing actual code is not something we've had to do yet with SAP HANA; we've been using the graphical modeling tools provided in the Studio. This is one of the strong points of the system. Writing SQL code in itself is relatively easy. It doesn't take much training to be able to write a simple `SELECT` statement. After a couple of tries, we can add a `JOIN` or two, `GROUP BY`, `ORDER BY`, and we think that we've mastered everything SQL has to offer.

The truth of the matter is, of course, very different. Writing complex SQL is DIFFICULT. Writing a large SQL instruction to do half of what the graphical calculation view we built in the previous chapter can do, without bringing the machine to its knees, is not something that many developers can do, especially those who are not specialized database developers.

Graphical developments and wizards are great things; however, there are some things you just cannot do with them. Complicated developments with specialized business rules sometimes cannot simply be modeled by joining blocks to each other. If this is the case in a project you're working on, you'll have to write the SQL instructions to do the work yourself.

As touched upon briefly in the previous chapter, SAP provides a mechanism that allows you to create calculation views using SQL statements and an internal development language, `SQLScript`. In this chapter, we'll be building a calculation view with the same characteristics as the one we made in the previous chapter, only this time we'll be using `SQLScript` to create it, not the graphical editor.
Creating a scripted calculation view

The creation of a scripted calculation view is done in exactly the same way as a graphical calculation view:

1. Open the Modeler perspective of the Studio (see the previous chapter for detailed instructions on how to do this).
2. Open the Quick Launch panel.
3. Select Calculation View | Create. You will be presented with the New Calculation View dialog we saw for the first time in the previous chapter.
4. Enter a Name and Description values (we'll use BOOK_CV_SC, for Book, Calculation View, Script for both Name and Description).
5. Select our book package.
6. Set the View Type option to SQL Script, as we can see in the following screenshot:

![New Calculation View dialog screenshot](image)

For More Information:
Setting the **View Type** option to **SQL Script** changes the options available at the bottom of the dialog. We now have the following options available to us:

- **Default Schema**: This option lets us define the database schema holding the tables that contain the data with which we want to work. In our case, this database schema is **BOOK**.

- **Run With**: Like for a graphical calculation view, this option defines whose authorization will be used to do the data selection in the database. A graphical calculation view can use only the authorization of the defining user, which, if you recall, is actually the system user `_SYS_REPO`. A scripted calculation view can use the authorization of the **invoker**, the user calling the view. This might be useful in some cases where data is user-sensitive. Authorizations in SAP HANA will be explained in full in *Chapter 7, Hey! That’s My Data! – Authorizations in SAP HANA*.

- **Parameter Case Sensitive**: When set to **true**, parameter (user input parameters) names are case sensitive and therefore should be written in capital letters. If not, their case does not matter.

7. Click on the **Finish** button to create the view, and see the scripted view creation screen, shown here:

![Screenshot of scripted view creation](image)

As we can see in the screenshot, a scripted calculation view starts life just like a graphical calculation view. We are given an output block (which in the scripted calculation view exists as a variable called `var_out`) and nothing else. All the work is left for us to do. Let's build our view in the same way we built our graphical calculation view in the previous chapter. The section headings toward the end of this chapter are exactly the same as those in the previous chapter, so that you can more easily compare the two ways of doing things.

---

For More Information:

Understanding CE functions

SQLScript provides a number of functions that enable optimized, parallel access to database objects. We can write native SQL to perform most tasks in the database, like this:

```sql
SELECT CUSTID, YEAR, SUM(AMOUNT)
from BOOK.REVENUE
where MANDT = 100
GROUP BY CUSTID, YEAR
```

It is, however, recommended to use the equivalent CE function to perform the task where possible. CE stands for Calculation Engine.

CE functions are preferred to standard SQL firstly because they are a lot easier to work with, secondly because their use can be parallelized by the SAP HANA server, and finally because they don’t require that the server use the SQL processor, which is time-saving. You can mix-and-match the use of CE functions and SQL inside your scripts; however, if there's a CE function that will do the job you need, it's a better idea to use it than to write the SQL yourself.

The CE function equivalent of the previous SQL statement would be something like the following screenshot. We’ll be taking a look at the individual functions used in a minute.

For More Information:

In this example, we use two CE functions, **CE_COLUMN_TABLE** and **CE_PROJECTION**. **CE_COLUMN_TABLE**, as its name suggests, is a function that selects data from a table and returns the requested columns in a scalar variable (here called \( \text{rev} \)).

Note the syntax of the functions:

- \( \text{output} = \text{function1}(\text{parameter1}, \text{parameter2}, \ldots) ; \)
- \( \text{output2} = \text{function2}(\text{output}, \text{parameter3}, \ldots) ; \)

Function1 returns its results, which we put in a variable called output. We can then use this variable in the call to function2, prefixing it with a colon.

The **CE_PROJECTION** function performs the same action as the projection block we saw in the previous chapter. It allows us to select the fields with which we want to work, and additionally, allows us to filter on them. The first **CE_PROJECTION** function filters the results of the **CE_COLUMN_TABLE** function, keeping only lines where the **MANDT** field has a value of 100. The second **CE_PROJECTION** function presents to the user only the **CUSTID**, **YEAR**, and **AMOUNT** fields.

For a full list of CE functions available in SAP HANA, and how you can use them in your developments, the SAP HANA SQLScript development guide is available for download from the sap.com website at http://help.sap.com/hana/hana_dev_sqlscript_en.pdf.

Now that we have had a first look at SQLScript and CE functions, let's create our scripted calculation view, and see in more detail the precise syntax of SQLScript.

**Adding analytic views**

Adding an analytic view to a scripted calculation view is, depending on your point of view, either easier or harder than adding one to a graphic calculation view. Rather than searching for the analytic view in the **Navigator** panel and dragging it to the view definition screen, in a scripted calculation view we just make a reference to the analytic view in our code.

The CE function allowing us to read the contents of an analytic view is **CE_OLAP_VIEW**. This function takes two parameters: the view name and the list of fields we wish to use. If we want to obtain the same results as in our graphical view from Chapter 5, *Let’s Get Graphical – Graphical Calculation Views*, we need to select all the fields, except **EUROPE_REVENUE** and **row_count**. As you will have noticed in the previous screenshot, this means that we have to enumerate all the fields we wish to use.

For More Information:
The `CE_OLAP_VIEW` function call therefore looks like this:

```plaintext
cust_rev = CE_OLAP_VIEW(
    "SYS_BIC"."book/CUST_REV",
    [
        "YEAR", "CUSTID", "CURRENCY", "SECTOR",
        "COUNTRY", "NAME", "SECTORNAME", "REGION",
        "REGTEXT", "CTRYNAME", SUM("AMOUNT") ,
        "VAT_RATE", SUM("VAT_AMOUNT")
    ]
);  
```

There are several things to note in this code snippet:

- The view name has been prefixed with a schema name. This is necessary since, as hinted-on earlier, SAP HANA transforms all your development objects into real SQL statements and views when you activate them. Your analytic view `CUST_REV` has been transformed into a real SQL view, in the `_SYS_BIC` database schema. All your development objects can be found in the `_SYS_BIC` schema. You can open it up in the Navigator panel to check for yourself if you like.

- The list of fields you select from the database is surrounded by square brackets. This is because SAP HANA expects one parameter, not a list of them. By surrounding your field list with square brackets, you turn the list of fields into a set.

- The order of fields you select from the view is unimportant. You can put them in any order you like (here, the fields have been listed in the same order they appear in the screenshots from the previous chapter).

- Any numeric fields (Measures) need to have an aggregation function associated with them. This can be seen in the case of the `AMOUNT` and `VAT_AMOUNT` fields, which have been summed. There are currently four aggregation functions available:
  - `count()` – returns the number of values
  - `sum()` – returns the sum of all values
  - `min()` – returns the smallest value in the list
  - `max()` – returns the largest value in the list

- The results of the `CE_OLAP_VIEW` function have been placed into a scalar variable, `cust_rev`. This variable can be used in further developments.
With the CUST_REV view being read by our calculation view, we can add the PROJ_REV view, in the same way, with the following code:

```sql
proj_rev = CE_OLAP_VIEW(
    "SYS_BIC"."book/PROJ_REV",
    ["PROJ_YEAR","CUSTID","CURRENCY","SECTOR",
    "COUNTRY","NAME","SECTORNAME","REGION",
    "REGTEXT","CTRYNAME",SUM("PROJECTED_AMOUNT"),
    "VAT_RATE",SUM("VAT_AMOUNT")
]
);
```

Your scripted calculation view should now look like the following screenshot. We cannot test it yet as no output has been defined, and in order to define output, more development work is required:

The projection block

Now that we have the contents of the analytic views in our calculation view, we need to add the DATA_TYPE field, as we did in the graphical view. To do this, we used a projection block, and we will be doing the same using SQLScript. The CE function we need to create a projection is called CE_PROJECTION, as we saw in an earlier screenshot.

For More Information:
The `CE_PROJECTION` function takes three parameters:

- The **source** of the data for the projection. This will usually be the result of another CE function, such as `CE_COLUMN_TABLE` or `CE_OLAP_VIEW`, for example. `CE_PROJECTION` cannot actually read data from the database as such, it only works with data which has already been read.
- The **data fields** to use in the projection. These fields must exist in the source, or be created dynamically in the projection. We'll be looking in a second at how to create fields dynamically.
- An optional **filter** to apply to the input fields.

In the previous section, we used the `CE_OLAP_VIEW` function to read the data from our `CUST_REV` and `PROJ_REV` views. The result of these data reads was put into two variables, `cust_rev` and `proj_rev`, respectively. We can now use these variables as input for the `CE_PROJECTION` function.

This is analogous to joining the input view block to a projection block in a graphical calculation view.

Let's use our `cust_rev` and `proj_rev` variables in our projections now. This will allow us to define the `DATA_TYPE` field, which should contain, as you recall, the word `REAL` for our `CUST_REV` view, and `PROJ` for `PROJ_REV`.

The SQLScript code we need to create the two projections is given as follows. This code is the equivalent of the definition of the two projection blocks in the graphical view we created in the last chapter.

```sql
projection_cust_rev = CE_PROJECTION(
    :cust_rev,
    [
        "YEAR","CUSTID","CURRENCY","SECTOR",
        "COUNTRY","NAME","SECTORNAME","REGION",
        "REGTEXT","CTRYNAME","AMOUNT",
        "VAT_RATE","VAT_AMOUNT",
        CE_CALC('''REAL''', VARCHAR(4)) AS "DATA_TYPE"
    ]
);
projection_proj_rev = CE_PROJECTION(
    :proj_rev,
    [
        "YEAR","CUSTID","CURRENCY","SECTOR",
        "COUNTRY","NAME","SECTORNAME","REGION",
        "CTRYNAME","AMOUNT",
        "VAT_RATE","VAT_AMOUNT",
        CE_CALC('''PROJ''', VARCHAR(4)) AS "DATA_TYPE"
    ]
);
```

For More Information:
As we can see in the code snippet, the CE_PROJECTION function lets us select fields from the underlying view (in our case we chose to work with all the fields we read in from the analytic view). It also allows us to create a calculated column, just like we did in the graphical view.

The creation of our calculated column DATA_TYPE is done using yet another CE function, this time CE_CALC.

The CE_CALC function takes two parameters:

- The calculation expression: This is exactly the same expression we enter in the calculated column dialog in the graphical calculation view creation wizard. In our example, we use the expressions 'REAL' and 'PROJ'. The expression itself needs to be enclosed by single quotes.

  ```sql
  CE_CALC('"REAL"', VARCHAR(4)) AS "REAL"
  ```

  Any single quote in the expression (such as the ones around the word REAL) need to be escaped. This means that we need to tell the SQLScript editor that it has not reached the end of the expression yet when it sees the single quote. To escape a single quote in the editor, we use another single quote. Hence the three single quotes in a row.

- The data type of the created column: Any valid SQL data type can be created. In our examples, the created columns are of type VARCHAR, and of length 4.

The final action we have performed in our projection is to rename two fields. In the PROJ_REV analytic view, the data we are interested in is in the PROJ_YEAR and PROJECTED_AMOUNT fields, and not the basic YEAR and AMOUNT fields. When we read the underlying view with the CE OLAP VIEW function, we decided not to read the YEAR and AMOUNT fields, since we do not need them in our development. In the CE_PROJECTION function we have just seen, we have renamed the PROJ_YEAR field to YEAR, using the AS keyword. Similarly, PROJECTED_AMOUNT has been renamed to AMOUNT.
At this point, we have created two more variables in our view, `projection_cust_rev` and `projection_proj_rev`. These two projections have allowed us to create our calculated `DATA_TYPE` column, and to rename some columns from the underlying view.

### The aggregation block

Once again, in this chapter we won't be using aggregation blocks in our view. However, if you need to create one in your developments, it is possible to do so in a scripted calculation view, just like it was using the graphical editor.

The CE function to create an aggregation block in a scripted view is called `CE_AGGREGATION`. This function takes the following three parameters:

- **The data source**: This needs to be the output of another CE function, such as `CE_OLAP_VIEW` or `CE_PROJECTION`. `CE_AGGREGATION` cannot read database data directly.
- **The aggregation columns**: This is a list, surrounded by square brackets, of the aggregations to perform. The four aggregation functions allowed are `COUNT()`, `SUM()`, `MIN()`, and `MAX()`. As you can guess, each of these functions takes a measure name as parameter.
- **The optional group by column(s)**: These define the grouping of the aggregation. For example, perform `count()` of different `YEAR` values; the input parameter will be parsed, and each value of the `YEAR` column will be counted. The output will contain one line per unique value in the `YEAR` column, along with a column containing the number of times each value was present in the input.

As you can imagine, and in the same way as we saw in the previous chapter, `CE_AGGREGATION` defines an implicit projection on the data; any column from the input that is not present in the list of aggregation columns nor in the group-by list, will not be present in the output of the function.

### The join block

In the previous chapter, we examined briefly how to use a join block in our graphical calculation view. We can create joins in a scripted calculation view also, and they can be of the following three types:

- **Inner Join**: The standard join type, where entries in the join field list from the left-hand table must exist in the right-hand table to be included in the output. This join type is created with the CE function `CE_JOIN`.

For More Information:

Chapter 6

- **Left Outer Join**: Entries in the left-hand table will be included in the output whether they exist in the right-hand table or not. The CE function to create this join type is `CE_LEFT_OUTER_JOIN`.

- **Right Outer Join**: Similar to the left outer join, but with the right-hand table taking precedence. Entries in the right-hand table will always be included in the output, whether they have correspondence in the left-hand table or not. To create a join of this type, use the `CE_RIGHT_OUTER_JOIN` function.

> For readers familiar with SQL, you might like to note that SQLScript does not currently support the notion of `FULL_OUTER_JOIN` in CE functions.

Each of these functions uses the same syntax, and they all take four parameters:

- **Left-hand table**: The variable containing the data from the table considered to be on the left. Like `CE_PROJECTION` and `CE_AGGREGATION`, the CE join functions cannot read data from the database directly, so data needs to be contained in variables already.

- **Right-hand table**: The variable containing the data from the table considered to be on the right.

- **Join attributes**: A list of one or more fields on which to perform the join between the tables. The field or fields need to exist in both tables. If you need to create a join using tables with field names that are different (maybe `CUSTID` and `CUSTOMER_ID`, which mean the same thing but the names are just different), then you will need to use a `CE_PROJECTION` beforehand in order to rename one of the fields.

- **Projection list**: The CE join functions allow you to provide a list of field names that should be in the result of the function. This, in effect, allows you to create a projection on the result of the join. This parameter is optional. If you do not provide it, only the fields on which you create the join (the join attributes) will be present in the result. If you wish to use other fields from the input tables, you need to specify their names here. If you provide this parameter, you must include at least all of the join attributes, or the activation of your view will fail.

### The union block

In our graphical calculation view, we used a union block to join our two projections, and we’ll be doing exactly the same thing in our scripted view. In order to create a union between two tables, SAP HANA provides yet another CE function, `CE_UNION_ALL`, which will allow us to create a union between our two projections.

For More Information:

The `CE_UNION_ALL` function takes only two parameters; the two variables containing the data on which to perform a union. These two tables must have the same field names in them, or the activation of the view will fail.

The result of the `CE_UNION_ALL` function is yet another variable, which we can use elsewhere in our view.

Here is the `CE_UNION_ALL` statement used in our scripted calculation view:

```javascript
union_result = CE_UNION_ALL(
  :projection_cust_rev,
  :projection_proj_rev
);
```

As you can see, the `CE_UNION_ALL` function is simple to use. This function is the direct equivalent of the union block we used in our graphical view earlier. At this stage, our view is almost finished. We just need to define the output of our view for it to be usable. It should look like the following screenshot at the moment:
Defining the view's output

In our graphical calculation view from the previous chapter, we were provided with an output block. We joined this output block to the output of the union block, and then added the fields from the union to the result field list of our view.

In a scripted calculation view, we do something similar. You will notice in the previous screenshot that we have an output block in the scripted calculation view, in the same way we had one in the graphical calculation view. This block works in exactly the same way; it is here that we define the output fields of the view.

The output block is linked to the script block in our view. What's more, the link between the two cannot be changed.

This means that the script block (which is the SQLScript code we've been writing) needs to give the data to the output block in some way.

It does this using the var_out variable you are given when you create the view. All we need to do is to assign a value to this variable, and then define the field names and types that we are assigning to the variable, and we will be able to pass data to our output block, and hence to our user.

Let's define our output variable first, then we can assign values to it. To define the output of the script, right-click on the right-hand pane of the display, the Output of <Script_View> panel, on the Output Parameter node, then select Create from the menu, to see the Define Output Parameter dialog, which is pictured as follows:

![Define Output Parameter dialog](image)

This dialog lets us specify the output fields of our script, their names, data type, and length. This will then let SAP HANA check that the values we assign in the script match the data we said we were going to assign. It makes syntax-checking for the SQLScript easier.

For More Information:
In the **Output Parameter** dialog, we need to list all the fields that are present in our `union_result` variable. This is somewhat fastidious as you need to list each field and its data type. To find the data type for the field you will need to go right back to the definition of the original tables that hold the base data.

To add a field to the output parameter, click on the green plus sign icon in the first column of the dialog to request a new line, then type the field name, its type and length. Here is the result of our **Output Parameter** dialog when completely filled-in for our view:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Len...</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR</td>
<td>VARCHAR</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>CUSTID</td>
<td>INTEGER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CURRENCY</td>
<td>VARCHAR</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>SECTOR</td>
<td>VARCHAR</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>COUNTRY</td>
<td>VARCHAR</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>VARCHAR</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>SECTORNAME</td>
<td>VARCHAR</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>REGION</td>
<td>VARCHAR</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>REGTEXT</td>
<td>VARCHAR</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>CTRYNAME</td>
<td>VARCHAR</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>AMOUNT</td>
<td>INTEGER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAT_RATE</td>
<td>DECIMAL</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>VAT_AMOUNT</td>
<td>DECIMAL</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>VARCHAR</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Once this has been done, click on **OK** to save the output parameter definition.

Now all that remains for us to do is to assign values to the `var_out` variable in order to be able to use them in our output block. This is done easily; we already have a variable containing the data we need – the `union_result` variable. It is enough that we rename the `union_result` variable to `var_out`, and we're done with the SQLScript.

For More Information:

The code for our `CE_UNION_ALL` function call therefore becomes:

```sql
var_out = CE_UNION_ALL(
    :projection_cust_rev,
    :projection_proj_rev
);
```

And our script is finished! We've read in the data from our analytic views, created projections on them, joined them together in a union, and indicated to SAP HANA what the output of the script will be in terms of field names, types, and lengths.

The complete code for the scripted calculation view is now as follows:

```sql
/******************** Begin Procedure Script ********************/
BEGIN
cust_rev = CE_OLAP_VIEW("_SYS_BIC"."book/CUST_REV",
    ["YEAR","CUSTID","CURRENCY","SECTOR","COUNTRY","NAME",
    "SECTORNAME","REGION","REGTEXT","CTRYNAME",SUM("AMOUNT"),
    "VAT_RATE",SUM("VAT_AMOUNT"))];
proj_rev = CE_OLAP_VIEW("_SYS_BIC"."book/PROJ_REV",
    ["PROJ_YEAR","CUSTID","CURRENCY","SECTOR","COUNTRY","NAME",
    "SECTORNAME","REGION","REGTEXT","CTRYNAME",
    SUM("PROJECTED_AMOUNT"),"VAT_RATE",SUM("VAT_AMOUNT"))]);
projection_cust_rev = CE_PROJECTION(:cust_rev,
    ["YEAR","CUSTID","CURRENCY","SECTOR","COUNTRY","NAME",
    "SECTORNAME","REGION","REGTEXT","CTRYNAME","AMOUNT",
    "VAT_RATE","VAT_AMOUNT",CE_CALC('''REAL''', VARCHAR(4)) AS "DATA_TYPE");
projection_proj_rev = CE_PROJECTION(:proj_rev,
    ["PROJ_YEAR" AS "YEAR","CUSTID","CURRENCY","SECTOR",
    "COUNTRY","NAME","SECTORNAME","REGION","REGTEXT",
    "CTRYNAME","PROJECTED_AMOUNT" AS "AMOUNT","VAT_RATE",
    "VAT_AMOUNT",CE_CALC('''PROJ''', VARCHAR(4)) AS "DATA_TYPE");
var_out = CE_UNION_ALL(:projection_cust_rev,
    :projection_proj_rev);
END /********** End Procedure Script ***********/
```

For More Information:
If you now click on the output block in the graphical depiction of the view on the left-hand side of the view definition screen, you will see that the list of fields we defined as output for the script is available, and we can add them as output for our view. Please add, by right-clicking on the field and then selecting the appropriate entry from the menu, AMOUNT and VAT_AMOUNT as measures and all other fields as attributes. This will then give the following output definition for the view:

There's just one more thing we need in order to finish our scripted calculation view; as always we need to activate the view. The view is then finished, and we can test it to make sure it performs as we expect.

Results

If we now open a data preview for our scripted view, we can see the contents of the view, and compare these results with those given by the graphical calculation view we created in the previous chapter.
In the following screenshot, we have a data preview of the graphical calculation view on the left, side-by-side with a data preview of our scripted calculation view on the right.

As you can see from the screenshot, the data present in both of our views is identical; we now possess the knowledge enabling us to create calculation views, either using the graphical designer provided in the Studio, or using SQLScript:

Summary
In this chapter, we have taken a look at the possibilities provided to us when we create scripted calculation views in SAP HANA. We have seen that scripted calculation views can achieve exactly the same results as graphical calculation views. Scripted calculation views are created by writing code, instead of by manipulating objects on the screen. We have been introduced to the CE functions provided by SAP HANA, and have used some of these, notably the CE_OLAP_VIEW, CE_PROJECTION, CE_CALC, and CE_UNION_ALL functions to read data from an analytic view, create a projection on the data read, calculate columns on the fly, and then join data from two views together. We have seen that the results of CE functions are assigned to variables, which we can then use in other functions, prefixing the variable name with a colon. Finally, we did a comparison of the output of our scripted view and its graphical counterpart, and saw that the view contents of both were identical.
In the next chapter, we’ll be looking at an important part of any data modeling or reporting project, the concept of **authorizations**. It is very rare that we wish for all users to have access to all the data in the system, and so we need to restrict who can see what. SAP HANA provides very fine-grained authorizations we can implement to achieve this, as we will see in the next chapter.

**Quick reference**

Here is a quick overview of the main topics covered in this chapter.

<table>
<thead>
<tr>
<th>Object</th>
<th>Usage</th>
<th>Creation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scripted calculation</td>
<td>Like a graphical calculation view, this allows development of complex</td>
<td>Quick Launch</td>
</tr>
<tr>
<td>view</td>
<td>data models, combining data from several sources, and presenting the data in one coherent place.</td>
<td></td>
</tr>
<tr>
<td>CE functions</td>
<td>These allow us to interact with the database, and the data we have read from it, without writing SQL.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CE_OLAP_VIEW: This reads an analytic view</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CE_COLUMN_TABLE: This reads a database table (column table only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CE_PROJECTION: This creates a projection on data read into a scalar variable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CE_CALC: This creates a calculated column in a projection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CE_JOIN, CE_LEFT_OUTER_JOIN, and CE_RIGHT_OUTER_JOIN: This joins two scalar variables together</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CE_UNION_ALL: This creates a union of two scalar variables (which have the same field list)</td>
</tr>
</tbody>
</table>

For More Information:

Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.