SAP NetWeaver
How-To Guide

How To...Use Data from another Aggregation Level in a PAK SQL-Script Procedure - Implemented as an AMDP Example

Applicable Releases:

SAP NetWeaver BW 7.40, SP6 and higher, Notes 1976514 and 1976522

SAP HANA 1.0

Topic Area:

Business Information Management

Version 1.4

June 2015

THE BEST-RUN BUSINESSES RUN SAP™

Document History

Document Version	Description
1.00	First official release of this guide
1.10	Remark added: currently the name of the method for the AMDP procedure has to be written in capital letters (section 3.3)
1.20	Remark added regarding how to read data from arbitrary InfoCubes
1.30	Remark added regarding read-only procedures (chapter 3.4)
1.40	Small correction in chapter 3.1: class needs to contain at least one interface

Typographic Conventions

Type Style	Description
Example Text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, graphic titles, and table titles
`Example text`	File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

Icons

Icon	Description
⚠	Caution
💡	Note or Important
❖	Example
⬆	Recommendation or Tip

Table of Contents

1. Scenario

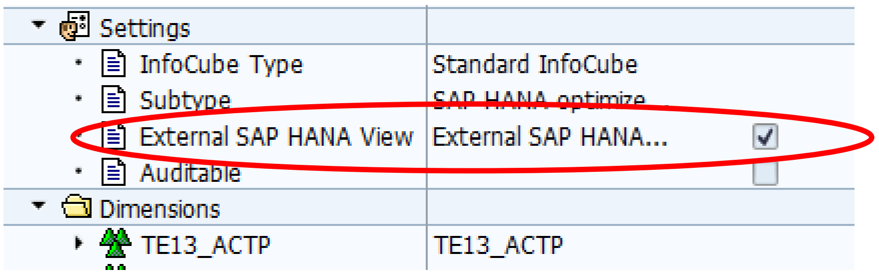
In PAK planning functions it is often necessary to use data that does not lie within the aggregation level. This data can be stored in the InfoCube as well or in a data base table. One example might be a table storing rates and a planning function that multiplies a key figure value with the corresponding rate.

In Fox formulas in releases before BW 7.40, SP8 it is not possible to use such 'external' data within a planning function. Only such kind of data can be used that can be selected on the given aggregation level (maybe by building a MultiProvider over different InfoProviders). Thus for the given use case the planning functions are often realized as custom defined planning functions ('ABAP Exit functions') or by using ABAP function module calls in a Fox formula. Unfortunately both types of planning functions cannot be executed in memory and the planning functions cannot profit from the performance boost that SAP HANA offers.

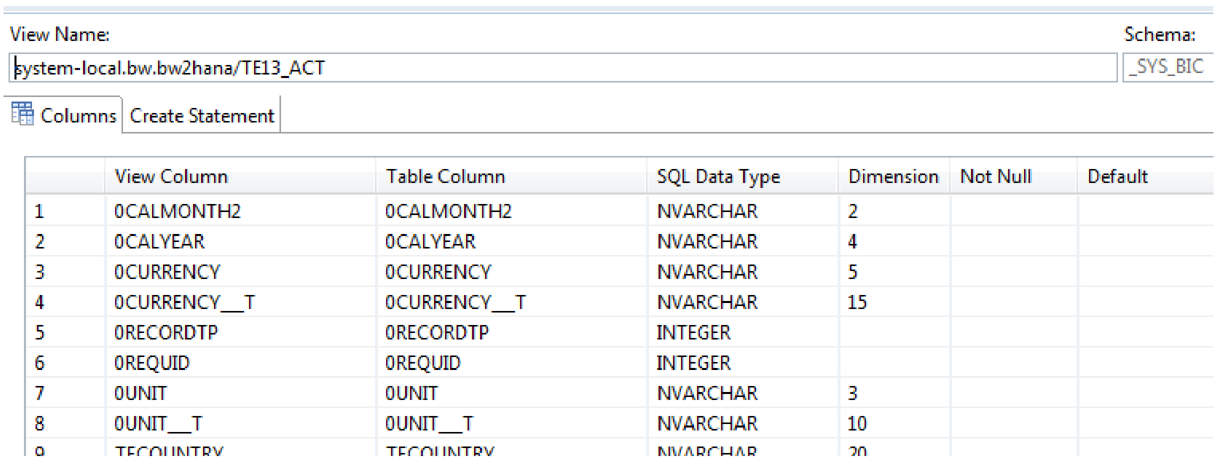
[From BW 7.40, SP8 on Fox does offer the option to read data from other aggregation levels but still it is not possible yet to read 'arbitrary data'.]

If the 'external' data is stored in a data base table it is very easy to access and use the data via a planning function realized via a SQL-Script procedure.

If the data is stored in a non- plannable InfoProvider it is possible to create a HANA view that enables read access to the InfoProvider data from SQL-Script. For an InfoCube you can create such a view simply by opening the InfoCube in the transaction RSA1 and setting the flag 'External SAP HANA View' in the settings. After regeneration of the InfoCube a corresponding view is available in HANA.



Settings	
InfoCube Type	Standard InfoCube
Subtype	SAP HANA optimize...
External SAP HANA View	External SAP HANA... ✓
Auditable	☐
Dimensions	
TE13_ACTP	TE13_ACTP



View Name: system-local.bw.bw2hana/TE13_ACT

Schema: _SYS_BIC

	View Column	Table Column	SQL Data Type	Dimension	Not Null	Default
1	0CALMONTH2	0CALMONTH2	NVARCHAR	2		
2	0CALYEAR	0CALYEAR	NVARCHAR	4		
3	0CURRENCY	0CURRENCY	NVARCHAR	5		
4	0CURRENCY__T	0CURRENCY__T	NVARCHAR	15		
5	0RECORDTP	0RECORDTP	INTEGER			
6	0REQUID	0REQUID	INTEGER			
7	0UNIT	0UNIT	NVARCHAR	3		
8	0UNIT__T	0UNIT__T	NVARCHAR	10		
9	TECOUNTRY	TECOUNTRY	NVARCHAR	20		

If the 'external' data is also manipulated in a planning session and thus the latest values are not yet stored in the InfoCube but only reside in the planning buffer it does not make sense to read the data via such a HANA view but it is necessary to access the data in the buffer. Up to now there was no possibility to access arbitrary data within an aggregation level from a SQL-Script procedure.

With BW 7.40 SP6 and notes 1976514 and 1976522 it is now possible to easily access such data in a SQL-Script procedure. In this paper we show you how this can be done.

2. General Description of the Solution

When implementing a SQL-Script planning function several objects need to be created: a table type corresponding to the aggregation level the planning function is defined on, the SQL-Script procedure itself, and the ABAP class that is used in the planning function type and that calls the procedure. There is a system report that helps in creating those objects: RSPLS_SQL_SCRIPT_TOOL. It can generate the table types and templates for the SQL-Script procedure as well as templates for the ABAP class.

This report has now been enhanced and it can also generate sample coding for SQL-Script procedures that can use buffer data from other aggregation levels. It also can create the template for the SQL-Script procedure as an AMDP (ABAP Managed Database Procedures), that is some SQL-Script coding that is created within an ABAP class in the ABAP environment. For those procedures it is not necessary to access SAP HANA via HANA studio and those procedures can be transported via the standard transport mechanisms. You can still create your procedures and types directly in SAP HANA but in this example we want to show how this can be done as an AMDP.

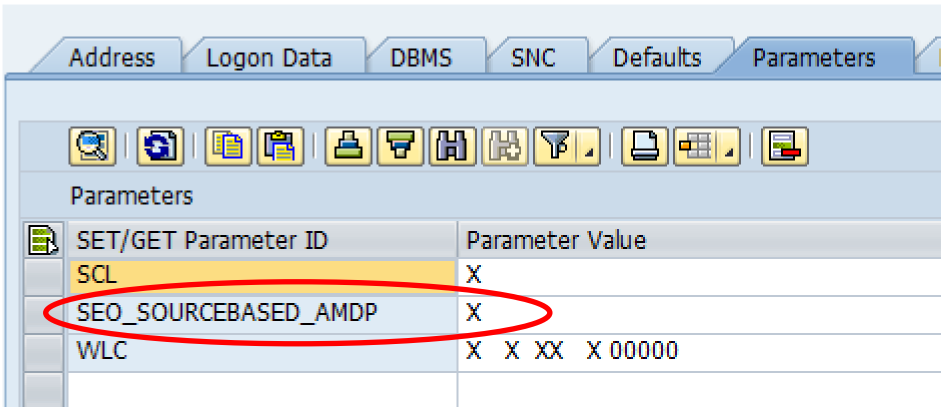
In order to have a minimum amount of work when creating such a planning function (including all the preliminary work) we would recommend the following steps.

3. Step by Step Solution

3.1 Create a Class Implementing the Planning Function

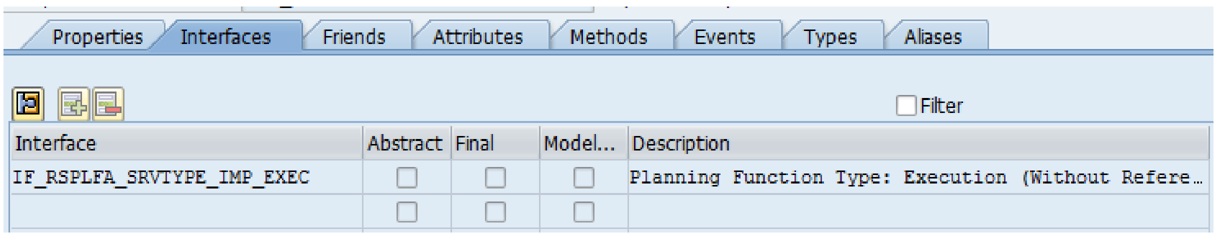
ABAP Managed Database Procedures can be created in the ABAP Eclipse environment. They can also be created in the standard transactions se80 or se24 but only when a special user parameter is set.

Open transaction su01, enter the name of your user, and press ‘change’. Go to the tab ‘Parameters’ and add the parameter SEO_SOURCEBASED_AMDP.



SET/GET Parameter ID	Parameter Value
SCL	X
SEO_SOURCEBASED_AMDP	X
WLC	X X XX X 00000

Now open se80 (or se24) and create a new class. The class needs to contain at least one of the relevant interfaces because otherwise the planning function type cannot be generated. If you want to use no reference data from the aggregation level the function should be based on the use the interface IF_RSPLFA_SRVTYPE_IMP_EXEC. If you need additional reference data from the aggregation level then please use IF_RSPLFA_SRVTYPE_IMP_EXEC_REF.



Interface	Abstract	Final	Model...	Description
IF_RSPLFA_SRVTYPE_IMP_EXEC	☐	☐	☐	Planning Function Type: Execution (Without Refere…
	☐	☐	☐	

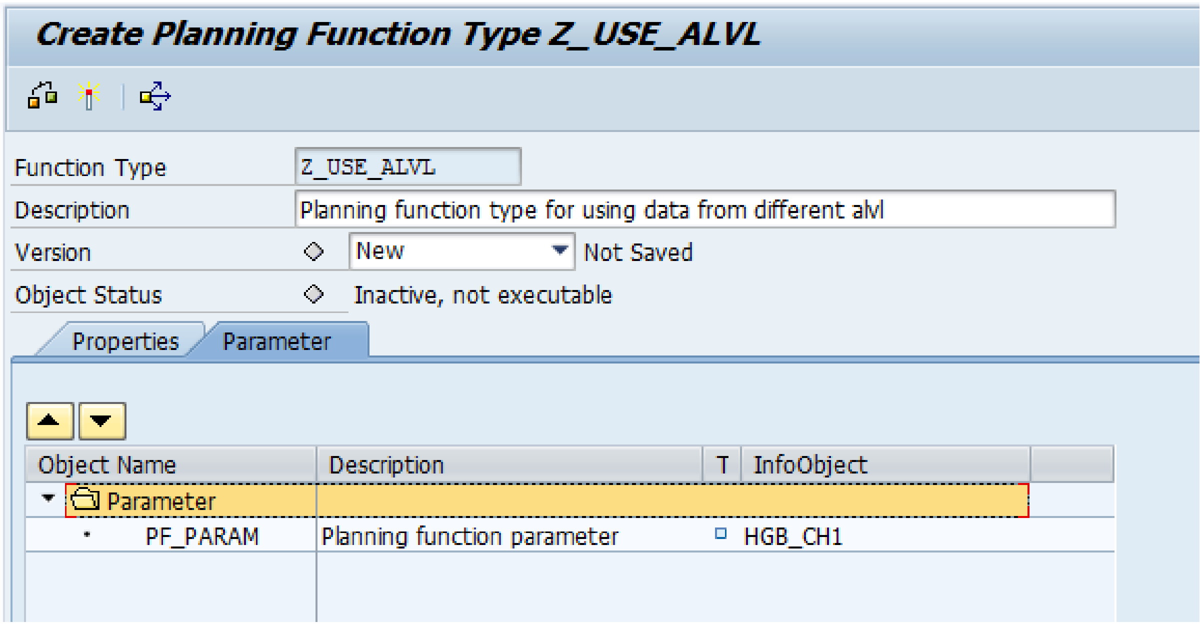
Otherwise leave the implementation of the class empty. Save and activate the class. We will fill in the coding for the necessary methods from the template later.

3.2 Create a New Planning Function Type

Now go to transaction ‘`rsplan`’, choose ‘`Goto`’, and ‘`Maintain Planning Function types`’. Create a new planning function type. Make sure you create all the parameters you need in your planning function. The name of the procedure could either be hard coded in the class later or can be transferred to the planning function via parameter. When you are using an AMDP then it is rather useful to hard-code the name in the corresponding class.

As implementing class for the planning function type enter the name of the class you have just created.

We add one parameter to our planning function type in order to demonstrate how they are used in the SQL-Script procedure.



Create Planning Function Type Z_USE_ALVL

Function Type	Z_USE_ALVL
Description	Planning function type for using data from different alvl
Version	New Not Saved
Object Status	Inactive, not executable

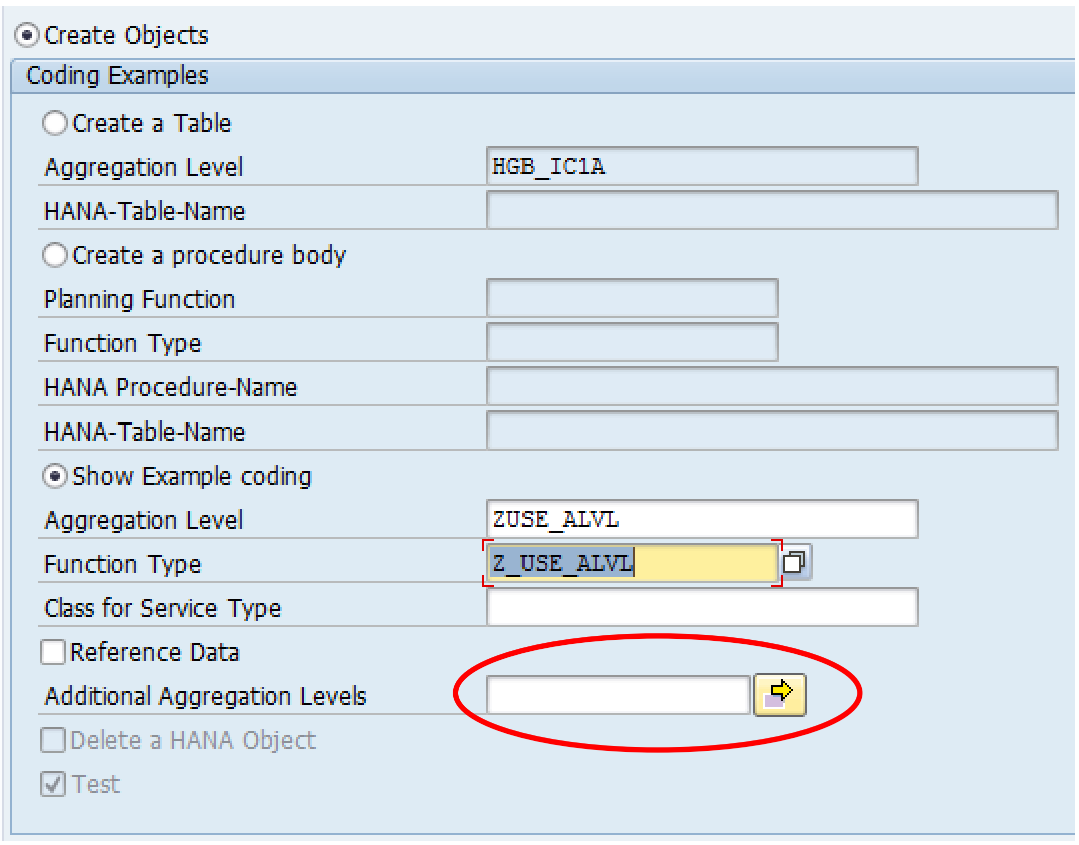
Object Name	Description	T	InfoObject
Parameter			
PF_PARAM	Planning function parameter		HGB_CH1

Save and activate the planning function type.

3.3 Implement the Class from the Template

Call the report RSPLS_SQL_SCRIPT_TOOL and choose the option ‘Show sample coding’. Enter the name of your planning function type and the name of the aggregation level the planning function should be defined on.

Now as we want to read data from additional aggregation levels either enter the name of the additonal aggregation level (if it is just one) or call the button ‘Multiple selection’ and enter as many aggregation levels as you need.



(● Create Objects)

Coding Examples

○ Create a Table	
Aggregation Level	HGB_IC1A
HANA-Table-Name	
○ Create a procedure body	
Planning Function	
Function Type	
HANA Procedure-Name	
HANA-Table-Name	
● Show Example coding	
Aggregation Level	ZUSE_ALVL
Function Type	Z_USE_ALVL
Class for Service Type	
☐ Reference Data	
Additional Aggregation Levels	
☐ Delete a HANA Object	
☑ Test	

In our example we use the two aggregation levels which are called ZREVALVL1 and ZREFALVL2 .

Now execute the report by pressing F8. The report returns a list of sample coding that you can use in your class.

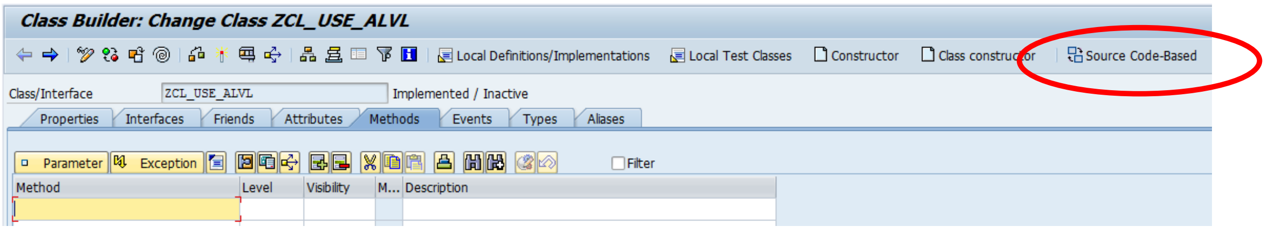
```
Report RSPLS_SQL_SCRIPT_TOOL

*  Definition of a table to be used as interface parameter for a ABAP Managed Databasse Procedure.
*  Copy this defintion into the public section of the definition of the class.
*
TYPES: BEGIN OF y_s_zuse_alvl,
        CALMONTH TYPE /BI0/OICALMONTH,
        HGB_CH1 TYPE /BIC/OIHGB_CH1,
        HGB_CH2 TYPE /BIC/OIHGB_CH2,
        CURRENCY TYPE /BI0/OICURRENCY,
        UNIT TYPE /BI0/OIUNIT,
        HGB_KY1 TYPE /BIC/OIHGB_KY1,
        HGB_KY2I TYPE /BIC/OIHGB_KY2I,
        HGB_KY3N TYPE /BIC/OIHGB_KY3N,
        HGB_KY4Q TYPE /BIC/OIHGB_KY4Q,
     END OF y_s_zuse_alvl.
TYPES: y_t_zuse_alvl TYPE STANDARD TABLE OF y_s_zuse_alvl.
*  Definition of a table to be used as interface parameter for a ABAP Managed Databasse Procedure.
*  Copy this defintion into the public section of the definition of the class.
*
TYPES: BEGIN OF y_s_zrefalvl1,
        CALMONTH TYPE /BI0/OICALMONTH,
        HGB_CH1 TYPE /BIC/OIHGB_CH1,
        HGB_CH2 TYPE /BIC/OIHGB_CH2,
        CURRENCY TYPE /BI0/OICURRENCY,
        UNIT TYPE /BI0/OIUNIT,
        HGB_KY2I TYPE /BIC/OIHGB_KY2I,
        HGB_KY4Q TYPE /BIC/OIHGB_KY4Q,
     END OF y_s_zrefalvl1.
TYPES: y_t_zrefalvl1 TYPE STANDARD TABLE OF y_s_zrefalvl1.
*  Definition of a table to be used as interface parameter for a ABAP Managed Databasse Procedure.
*  Copy this defintion into the public section of the definition of the class.
*
```

Let us go through this sample coding. First of all the report shows some type definitions. We need those types in order to define the structure of the interface of our SQL-Script procedure. [If you create the SQL-Script procedure directly in SAP HANA then you also have to create those types in SAP HANA. As we are using an AMDP we can define those types in our ABAP class].

We have one type for the aggregation level on which the planning function is defined and one type for each of the reference aggregation levels we want to read data from. The name of the type is derived from the names of the aggregation levels.

In a second mode open your class in change mode and go to the source code-based mode of the class.

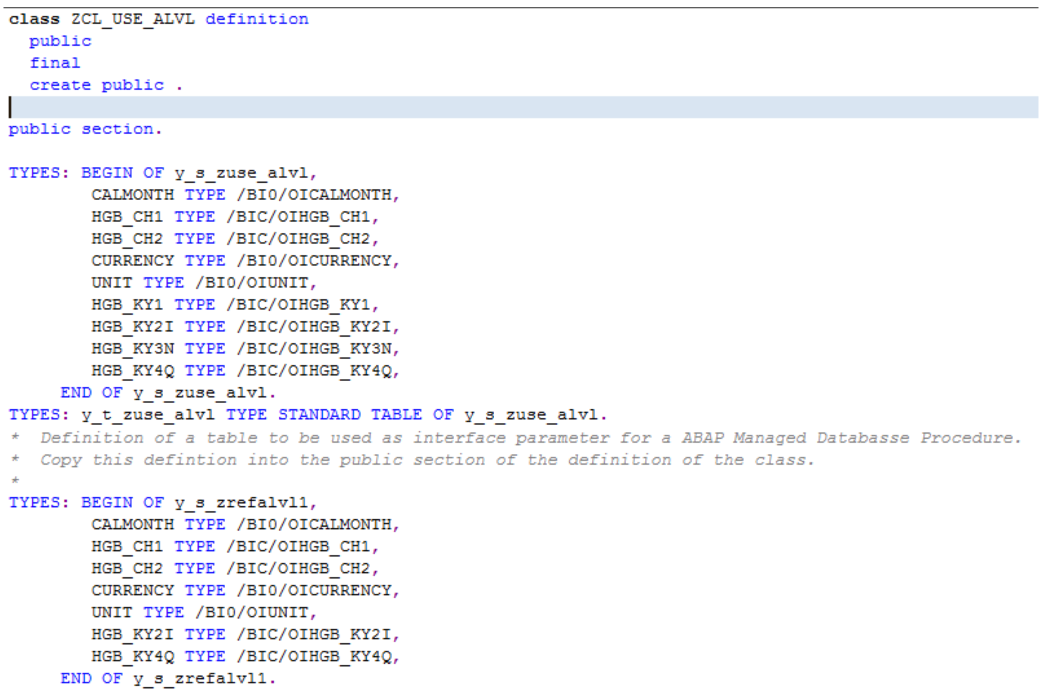


Class Builder: Change Class ZCL_USE_ALVL

Class/Interface ZCL_USE_ALVL Implemented / Inactive

[As we will implement an AMDP we can only work in the source code-based mode from now on.]

Now copy and paste the types from the template into the public section of the definition of the class.

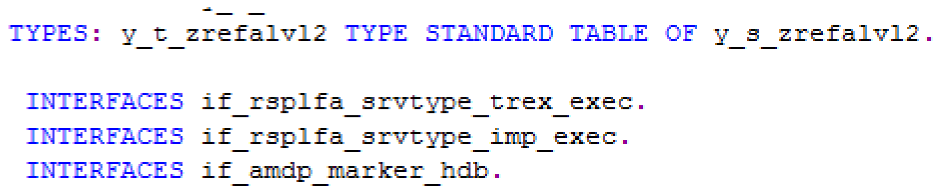


```
class ZCL_USE_ALVL definition
  public
  final
  create public .

public section.

TYPES: BEGIN OF y_s_zuse_alvl,
        CALMONTH TYPE /BI0/OICALMONTH,
        HGB_CH1 TYPE /BIC/OIHGB_CH1,
        HGB_CH2 TYPE /BIC/OIHGB_CH2,
        CURRENCY TYPE /BI0/OICURRENCY,
        UNIT TYPE /BI0/OIUNIT,
        HGB_KY1 TYPE /BIC/OIHGB_KY1,
        HGB_KY2I TYPE /BIC/OIHGB_KY2I,
        HGB_KY3N TYPE /BIC/OIHGB_KY3N,
        HGB_KY4Q TYPE /BIC/OIHGB_KY4Q,
     END OF y_s_zuse_alvl.
TYPES: y_t_zuse_alvl TYPE STANDARD TABLE OF y_s_zuse_alvl.
*  Definition of a table to be used as interface parameter for a ABAP Managed Databasse Procedure.
*  Copy this defintion into the public section of the definition of the class.
*
TYPES: BEGIN OF y_s_zrefalvl1,
        CALMONTH TYPE /BI0/OICALMONTH,
        HGB_CH1 TYPE /BIC/OIHGB_CH1,
        HGB_CH2 TYPE /BIC/OIHGB_CH2,
        CURRENCY TYPE /BI0/OICURRENCY,
        UNIT TYPE /BI0/OIUNIT,
        HGB_KY2I TYPE /BIC/OIHGB_KY2I,
        HGB_KY4Q TYPE /BIC/OIHGB_KY4Q,
     END OF y_s_zrefalvl1.
```

The next section in the template report is for the interfaces we need for our class. Just copy and paste the interface names behind the type in the public section of the class definition.

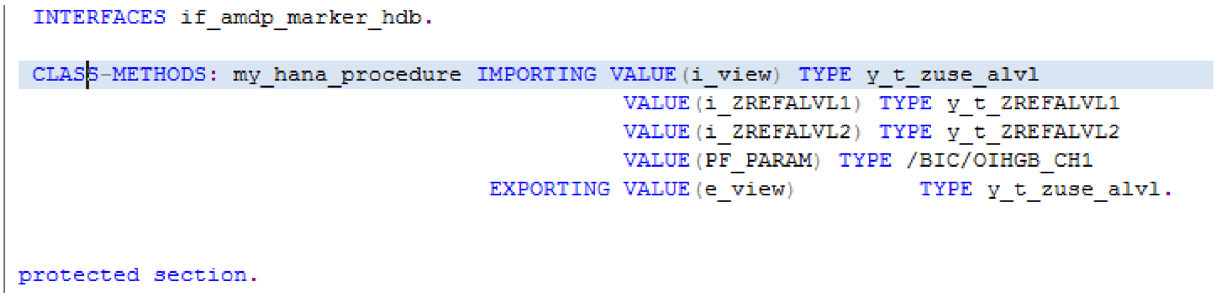


```
TYPES: y_t_zrefalvl2 TYPE STANDARD TABLE OF y_s_zrefalvl2.

  INTERFACES if_rsplfa_srvtype_trex_exec.
  INTERFACES if_rsplfa_srvtype_imp_exec.
  INTERFACES if_amdp_marker_hdb.
```

Note that the system checks whether the planning function type is with or without reference data. If reference data (from the underlying aggregation level) is used in the planning function then the system will generate the corresponding interfaces into the template (if_rsplfa_srvtype_trex_exec_r and if_rsplfa_srvtype_imp_exec_ref).

As a last part in the class definition we need the definition of our AMDP method. Copy the corresponding coding from the template and paste it under the interface definition.



```
  INTERFACES if_amdp_marker_hdb.

  CLASS-METHODS: my_hana_procedure IMPORTING VALUE(i_view) TYPE y_t_zuse_alvl
                                             VALUE(i_ZREFALVL1) TYPE y_t_ZREFALVL1
                                             VALUE(i_ZREFALVL2) TYPE y_t_ZREFALVL2
                                             VALUE(PF_PARAM) TYPE /BIC/OIHGB_CH1
                                   EXPORTING VALUE(e_view)         TYPE y_t_zuse_alvl.


 protected section.
```

Note that the procedure already has all necessary parameters in the signature: the table of the existing data in the underlying aggregation level, a table with the data from the first ‘external’ aggregation level, a table for the second aggregation level, a parameter for the parameter of the planning function type we have defined, and a returning table for the changed data (delta or after image).

The next part of the template contains the coding for the methods. Copy and paste this remaining content of the template into the IMPLEMENTATION (!) part of the class.

```
CLASS ZCL_USE_ALVL IMPLEMENTATION.

    METHOD if_rsplfa_srvtype_trex_exec~trex_execute.
    DATA: l_r_sql_script   TYPE REF TO if_rspls_sql_script,
          l_procedure_name TYPE string,
          l_t_iobj_param   TYPE if_rsr_pe_adapter=>tn_t_iobj_param.
    l_r_sql_script = cl_rspls_session_store_manager=>get_sql_script_instance( i_r_store = i_r_store ).
*   The method if_rspls_sql_script~get_parameter_values returns a table of parameters
*   with the values given in the function definition
    l_r_sql_script->get_parameter_values(
      EXPORTING
        i_r_param_set              = i_r_param_set
        i_para_name_for_procedure = 'HANA_PROCEDURE_NAME'
      IMPORTING
        e_procedure_name           = l_procedure_name
        e_t_iobj_param             = l_t_iobj_param ).
*   The function parameter given mehtod paramenter i_para_name_for_procedure
*   will not be returned in the table e_t_iobj_param but the value of this
```

(only start of the coding, for the full coding see the generated template)

Now save and activate your class. You might get an information message that the class cannot be changed in the GUI. That means you can only change the class in the source code-based mode.

Note: If you need to change the name of the procedure please make sure that in the implementation the name of the procedure is written in capital letters – just as in the generated template

```
    l_procedure_name = 'ZCL_USE_ALVL=>MY_HANA_PROCEDURE'.
```

Otherwise the system will not ba able to call the procedure.

When you are reading data from another aggregation level you naturally want to specify a selection for this data. Please have a look at the implementation of the method `if_rsplfa_srvtype_trex_execute~trex_execute`. You will find the commented coding below which can be used as a sample how a selection can be set. Just uncomment the coding and put the selection you want to use.

```
    DATA: l_t_aggr_view    TYPE if_rspls_sql_script=>t_aggr_view,
          l_s_aggr_view    TYPE if_rspls_sql_script=>s_aggr_view,
          l_s_charsel      TYPE rsplf_s_charsel.
    CLEAR l_s_aggr_view-t_charsel.
    l_s_aggr_view-aggregation_level = 'ZREFALVL1'.
 *   l_s_charsel-iobjnm = 'IOBJNM'.
 *   l_s_charsel-opt = 'EQ'.
 *   l_s_charsel-sign = 'I'.
 *   l_s_charsel-low = 'Value'.
 *   APPEND l_s_charsel TO l_s_aggr_view-t_charsel.
    APPEND l_s_aggr_view TO l_t_aggr_view.
```

You can set one of these selection tables for each of the aggregation levels you want to read data from.

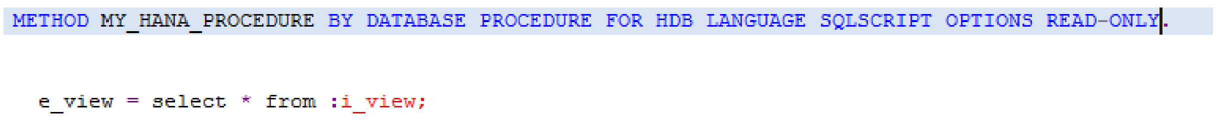
Let us finally have a look at the implementation of the planning logic itself. It can be found in the method `my_hana_procedure`. This method contains the SQL-Script coding. In the template you can find a very simple implementation in SQL-Script. This implementation just reads data from the aggregation level (table i_view) and inserts the data into the result table e_view. If you want to access the data from the addition aggregation levels in our example you just have to enter some SQL-Script coding that reads (and uses) data from the tables i_zrefalvl1 and i_zrefalvl2. The information for the parameter value can be found in the field PF_PARAM.

Remark: In this paper we have described how you can create the SQL-Script planning function using AMDP. You can also create a SQL-Script procedure reading additional aggregation levels directly in SAP HANA. You will find additional functionality in the report RSPLS_SQL_SCRIPT_TOOL. The interface for the SQL-Script procedure and the ABAP coding for calling the procedure are the same as the ones we have seen above. You just have to set the proper names of the procedure in the ABAP coding. As the procedure will need the table types of all the involved aggregation levels you will have to create the table types in SAP HANA directly (instead of defining them in our ABAP class).

For finalizing your model implement the SQL-Script logic, create a planning function using the type you have just created and test you planning function.

3.4 Side-Effect Free AMDP Procedures

As described in our introduction into using SQL-Script procedures for planning functions (How to... Use SQLScript for Planning Functions in PAK) by default the procedures used in planning functions have to be side-effect free. This has to be declared in the header of the procedure. When using an AMDP the statement is ‘OPTIONS READ-ONLY‘:



```
METHOD MY_HANA_PROCEDURE BY DATABASE PROCEDURE FOR HDB LANGUAGE SQLSCRIPT OPTIONS READ-ONLY.

  e_view = select * from :i_view;
```

As described in the paper mentioned above non side-effect free procedures can be allowed.