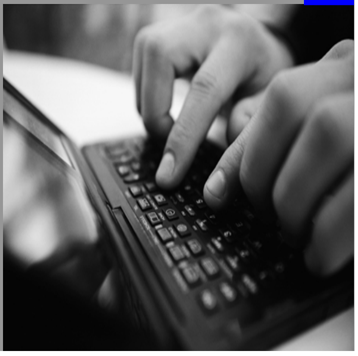
How-to Guide
SAP NetWeaver 7.0 (2004s)



How To... Run Planning Functions on Changed Records in SAP NetWeaver BW Integrated Planning

Version 2.00 – January 2009

Applicable Releases:
SAP NetWeaver 7.0 (2004s)
SAP NetWeaver BI
BI Integrated Planning

1 Scenario

In planning project we often deal with a great number of plan data for the entire project. The single planners usually are also responsible for a great amount of data but often only change a small subset of this data. Which data will be changed by the user cannot be foreseen. On the other hand the number of touched records is a crucial parameter that drives the performance of planning functions. Thus the number of those records should be restricted as much as possible. But how can this be done without knowing what the user will change?

This how to paper will show how to model a system where the records changed by the user will be marked by a flag. The flag can then be used to build up a minimal selection for the planning functions

2 Introduction

As stated above the aim of this paper is to build up a scenario where a user can change one or several records and then can execute a planning function that only works on the changed records. Especially in the case that a user changes a small set of data form a large set of available data this should lead to a great improvement in performance.

We want to give an outline of the solution first. In order to track which records have been changed we introduce a new characteristic in the InfoCube that we call ‘Z_DELTA’. We create a characteristic relationship that does a derivation and fills ‘Z_DELTA’ with some value, e.g. ‘X’. When we change a record in an aggregation level that does not contain the dummy characteristic each delta record carries an ‘X’ in the dummy characteristic and is thus marked as changed. Assume we change a key figure from 10 to 12. We then will have 2 data records, one with ‘Z_DELTA = #’ and the value 10 and a delta record with ‘Z_DELTA = X’ and the value 2.

Now we want to run our planning function. Depending on the nature of the planning function we either run it on the pure delta value 2 or on the “entire” value which is 12. In the first case we can run the planning function on an aggregation level containing the dummy characteristic. We simple use the value ‘Z_DELTA = X’ in the filter and the planning function only uses the changed values. If we want to run the planning function on the entire value (which is mostly the case) we have to run it on an aggregation level that does not contain the dummy characteristic and thus cannot simple select the data records buy setting a selection for ‘Z_DELTA’. In this paper we will show how we can build up a selection that contains all changed records: we can either use a new mapping in the Bex Web Application Designer or use an ABAP Exit.

As our last action we have to clear the delta indicator from the data records. This will be done by a repost function defined on an aggregation level containing ‘Z_DELTA’. Again we will need the same techniques as mentioned above to restrict the filter for this planning function.

You will have to keep in mind that the scenario produces zero records – the repost function deletes the key figure values in the delta records marked with the delta flag and thus writes them as zero records to the delta buffer. After a save these zero records are stored on the data base. In order to keep the size of the data base as small as possible you should compress the InfoCube regularly using zero compression.

Let us show an example of the records. Let us assume we start with a very simple data model just using one characteristic (Product) and the key figures Price, Quantity and Revenue. As a second characteristic we add our delta indicator (Z_DELTA).

We start with changing the existing data on an aggregation level that does not contain the delta indicator (AGL1).

Product	Price	Quantity	Revenue
Prod001	100	1	100
Prod002	200	2	400
Prod003	300	3	900

We now change two records:

Product	Price	Quantity	Revenue
Prod001	150	1	100
Prod002	250	2	400
Prod003	300	3	900

The system will create two data records and the characteristic relationship will fill the delta indicator. On the data base level (or an aggregation level containing all characteristics, AGL2) we have to following records:

Product	Z_DELTA	Price	Quantity	Revenue
Prod001		100	1	100
Prod001	X	50	0	0
Prod002		200	2	400
Prod002	X	50	0	0
Prod003		300	3	900

If we now run a query on AGL2 restricting Z_DELTA to ‘X’ we get records for Prod001 and Prod002. Thus we have the information which products have been changed. We now decide whether we want to run our planning function on the delta values (50 for both products) or on the entire values (150 and 250): If we want to work on the deltas only, we choose AGL2 for our planning function. If we need the entire values we have to use AGL1. In our case we want to recalculate the revenue. Obviously, if we use only the delta records then the Quantity is 0 in both cases and the planning function would not calculate the revenue correctly.

After we have run our revenue calculation we have to delete the delta indicator from the data records. This is done by a repost function. After the repost we have the following data records:

Product	Z_DELTA	Price	Quantity	Revenue
Prod001		150	1	150
Prod001	X	0	0	0
Prod002		250	2	500
Prod002	X	0	0	0
Prod003		300	3	900

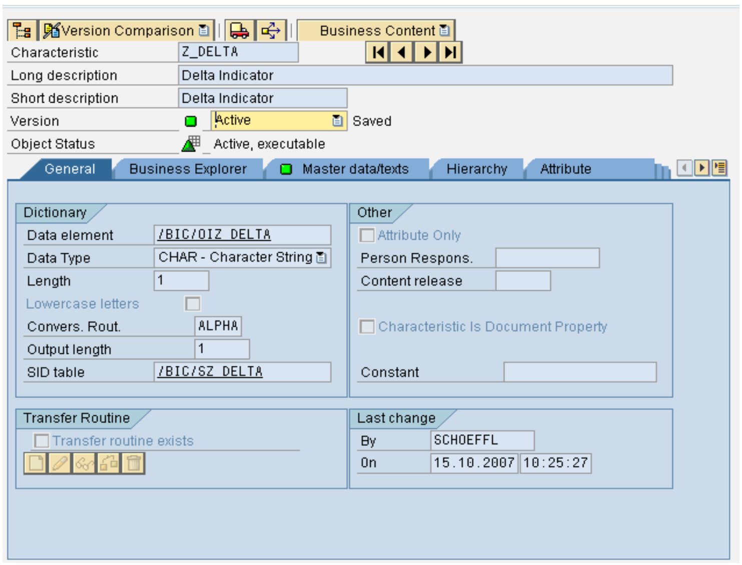
As we can see the model produces a zero record for every changed product. This has to be kept in mind (see above).

3 The Step By Step Solution

We will build up an example with a revenue calculation for a couple of products. The data records contain the key figures quantity, price, and revenue. If the price or the sold quantity for a product is change then the revenue should be calculated only for those products where we did the change.

3.1 Build up the InfoCube, Aggregation Levels, and Planning Functions

1. Create the delta indicator: Use transaction RSA1 and create a new characteristic used for the delta flag. We use the technical name ‘Z_DELTA’. The characteristic should be a character string of length 1. Also create a master data record with value ‘X’.



Characteristic	Z_DELTA
Long description	Delta Indicator
Short description	Delta Indicator
Version	Active Saved
Object Status	Active, executable

Dictionary

Data element	/BIC/OIZ_DELTA
Data Type	CHAR - Character String
Length	1
Lowercase letters	☐
Convers. Rout.	ALPHA
Output length	1
SID table	/BIC/SZ_DELTA

Other

Attribute Only	☐
Person Respons.	
Content release	
Characteristic Is Document Property	☐
Constant	

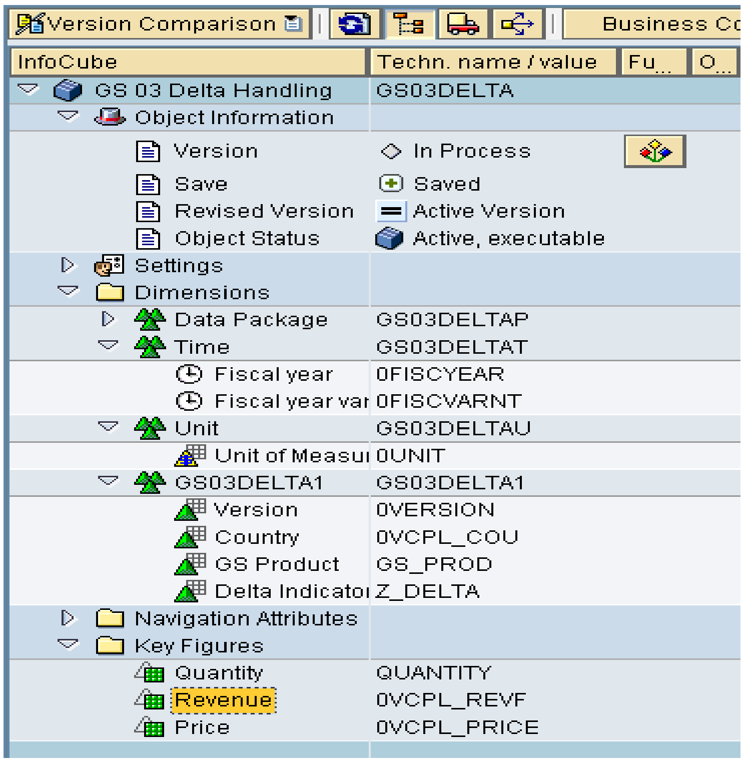
Transfer Routine

☐ Transfer routine exists

Last change

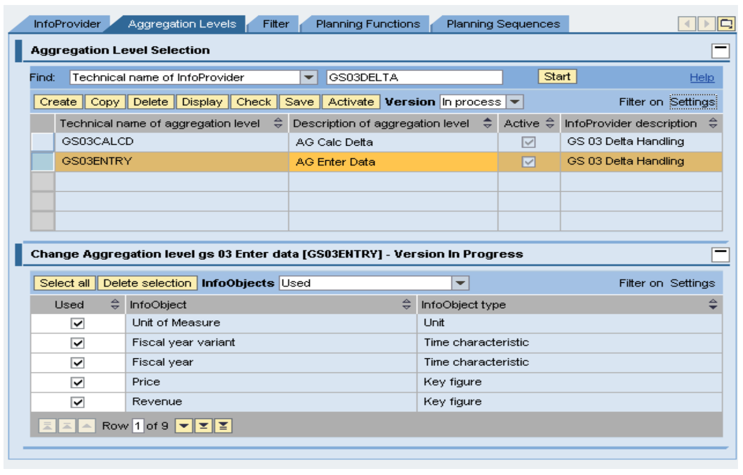
By	SCHOEFFL
On	15.10.2007 10:25:27

2. Create the InfoCube: Create a realtime InfoCube that uses the changed flag ‘Delta Indicator’. In our example we also use the characteristics Product, Version, Country, Unit, Fiscal Year, Fiscal Year Variant, and the key figures Quantity, Price (fixed currency Euro), and Revenue. Save and activate the InfoCube. You could also use one of you existing InfoCubes in this step.



InfoCube	Techn. name / value	Fu...	O...
GS 03 Delta Handling	GS03DELTA		
Object Information			
Version	In Process		
Save	Saved		
Revised Version	Active Version		
Object Status	Active, executable		
Settings			
Dimensions			
Data Package	GS03DELTAP		
Time	GS03DELTAT		
Fiscal year	0FISCYEAR		
Fiscal year var	0FISCVARNT		
Unit	GS03DELTAU		
Unit of Measui	0UNIT		
GS03DELTA1	GS03DELTA1		
Version	0VERSION		
Country	0VCPL_COU		
GS Product	GS_PROD		
Delta Indicatoı	Z_DELTA		
Navigation Attributes			
Key Figures			
Quantity	QUANTITY		
Revenue	0VCPL_REVF		
Price	0VCPL_PRICE		

3. Create the aggregation levels: Enter the planning modeler and create two aggregation levels. The first aggregation level should contain all characteristics and key figures necessary for the planning model (AG Enter Data), the second one should contain also the delta indicator (AG Calc Delta). Save and activate the aggregation levels



Aggregation Level Selection

Find: Technical name of InfoProvider GS03DELTA Start Help

Create Copy Delete Display Check Save Activate **Version** In process Filter on Settings

	Technical name of aggregation level	Description of aggregation level	Active	InfoProvider description
	GS03CALCD	AG Calc Delta	✓	GS 03 Delta Handling
	GS03ENTRY	AG Enter Data	✓	GS 03 Delta Handling

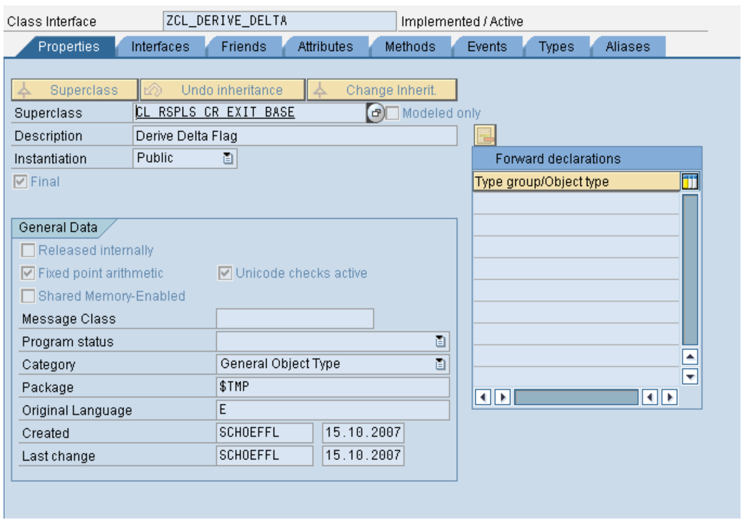
Change Aggregation level gs 03 Enter data [GS03ENTRY] - Version In Progress

Select all Delete selection **InfoObjects** Used Filter on Settings

Used	InfoObject	InfoObject type
✓	Unit of Measure	Unit
✓	Fiscal year variant	Time characteristic
✓	Fiscal year	Time characteristic
✓	Price	Key figure
✓	Revenue	Key figure

Row 1 of 9

4. Create a class for a characteristic relationship: Open transaction se80 and create a new class (ZCL_DERIVE_DELTA). Use the class CL_RSPLS_CR_EXIT_BASE as a super class. Press save. When you change to the tab 'Methods' all methods of the super class are visible then. When using the above class as super class our class will inherit a working implementation (that does nothing…) for all necessary methods. Thus we only have to change the derive method and do not have to care about the other methods.



Class Interface ZCL_DERIVE_DELTA Implemented / Active

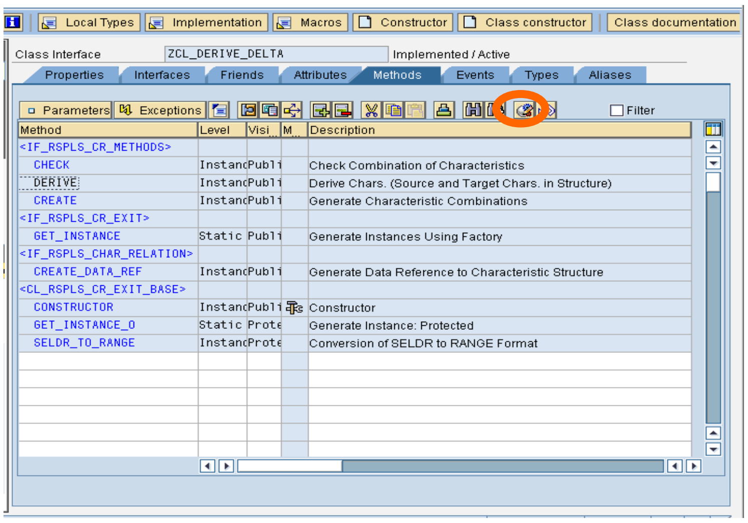
Superclass	CL_RSPLS_CR_EXIT_BASE ☐ Modeled only
Description	Derive Delta Flag
Instantiation	Public
✓ Final	

General Data

☐ Released internally
✓ Fixed point arithmetic ✓ Unicode checks active
☐ Shared Memory-Enabled

Message Class		
Program status		
Category	General Object Type	
Package	$TMP	
Original Language	E	
Created	SCHOEFFL	15.10.2007
Last change	SCHOEFFL	15.10.2007

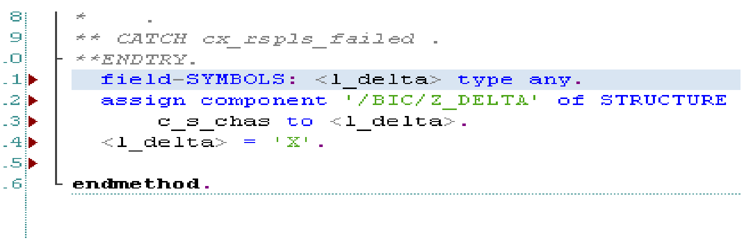
5. Redefine the derive method: mark the derive method and press the button circled in the picture.



Class Interface ZCL_DERIVE_DELTA Implemented / Active

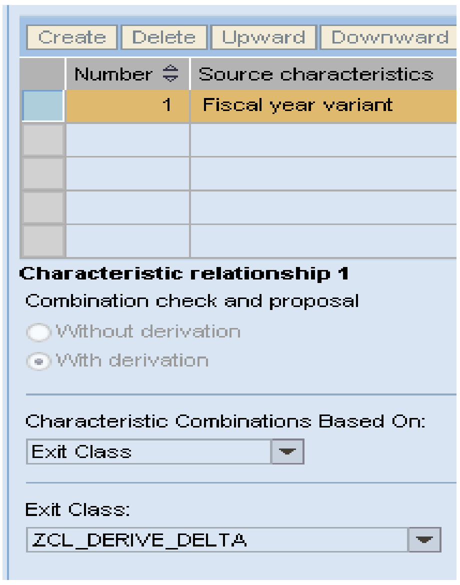
Method	Level	Visi...	M...	Description
<IF_RSPLS_CR_METHODS>				
CHECK	Instanc	Publi		Check Combination of Characteristics
DERIVE	Instanc	Publi		Derive Chars. (Source and Target Chars. in Structure)
CREATE	Instanc	Publi		Generate Characteristic Combinations
<IF_RSPLS_CR_EXIT>				
GET_INSTANCE	Static	Publi		Generate Instances Using Factory
<IF_RSPLS_CHAR_RELATION>				
CREATE_DATA_REF	Instanc	Publi		Generate Data Reference to Characteristic Structure
<CL_RSPLS_CR_EXIT_BASE>				
CONSTRUCTOR	Instanc	Publi		Constructor
GET_INSTANCE_O	Static	Prote		Generate Instance: Protected
SELDR_TO_RANGE	Instanc	Prote		Conversion of SELDR to RANGE Format

6. Finish the implementation: In the derive method enter the coding shown in the picture. If you have chosen another name for your changed flag you will have to adopt the coding accordingly. Save and activate you class. See also the appendix for the coding.



```
*       .
** CATCH cx_rspls_failed .
**ENDTRY.
  field-SYMBOLS: <l_delta> type any.
  assign component '/BIC/Z_DELTA' of STRUCTURE
       c_s_chas to <l_delta>.
  <l_delta> = 'X'.

endmethod.
```

7. Create a characteristic relationship: in the planning modeler go to the InfoProvider tab. Create a new characteristic relationship. Choose the radio button ‘with derivation’ and as a base for the characteristic relationship choose ‘Exit class’. Enter the name of our class (ZCL_DERIVE_DELTA). As the derivation should run any time we change a record we choose as the source characteristic a characteristic that is contained in every aggregation level. In our case we choose fiscal year variant. As target characteristic we choose our delta indicator



Number	Source characteristics
1	Fiscal year variant

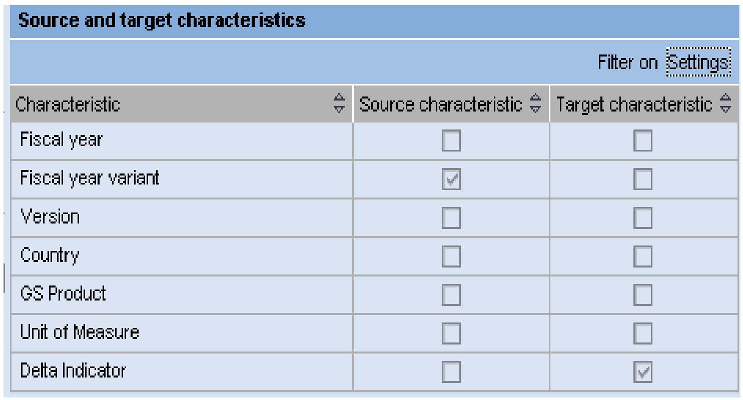
Characteristic relationship 1

Combination check and proposal

Without derivation [✓ With derivation]

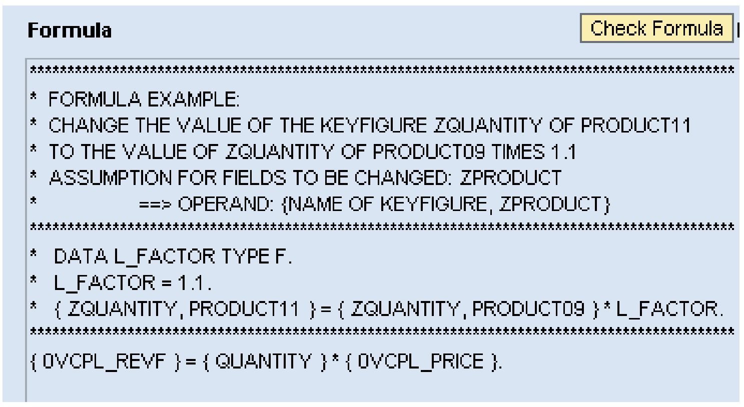
Characteristic Combinations Based On: Exit Class

Exit Class: ZCL_DERIVE_DELTA


Source and target characteristics

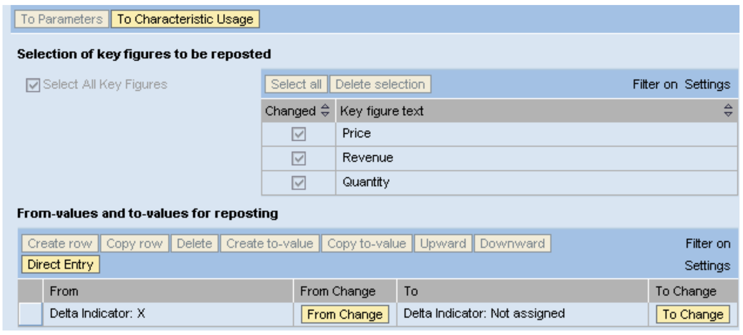
Characteristic	Source characteristic	Target characteristic
Fiscal year	☐	☐
Fiscal year variant	✓	☐
Version	☐	☐
Country	☐	☐
GS Product	☐	☐
Unit of Measure	☐	☐
Delta Indicator	☐	✓

8. Create a planning function to calculate the revenue: go to the tab ‘Aggregation Level’ and select the aggregation level that does not contain the delta indicator (AG Enter Data). This is the aggregation level used for entering data and calculating the revenue. Create a formula that multiplies quantity with price. [Note: the revenue cannot be calculated on the delta records only. If you only change the quantity in an existing record for a given product then the delta record in the aggregation level ‘AG Calc Delta’ will contain a ‘0’ as price. But for calculating the revenue you need the proper price.]


Formula

```
**************************************************************************************
*  FORMULA EXAMPLE:
*  CHANGE THE VALUE OF THE KEYFIGURE ZQUANTITY OF PRODUCT11
*  TO THE VALUE OF ZQUANTITY OF PRODUCT09 TIMES 1.1
*  ASSUMPTION FOR FIELDS TO BE CHANGED: ZPRODUCT
*              ==> OPERAND: {NAME OF KEYFIGURE, ZPRODUCT}
**************************************************************************************
*   DATA L_FACTOR TYPE F.
*   L_FACTOR = 1.1 .
*   { ZQUANTITY, PRODUCT11 } = { ZQUANTITY, PRODUCT09 } * L_FACTOR.
**************************************************************************************
{ 0VCPL_REVF } = { QUANTITY } * { 0VCPL_PRICE }.
```

9. Create a repost function: change to the aggregation level containing the delta flag and create a repost function. The function should change the characteristic ‘Z_DELTA’ from ‘X’ to ‘#’.



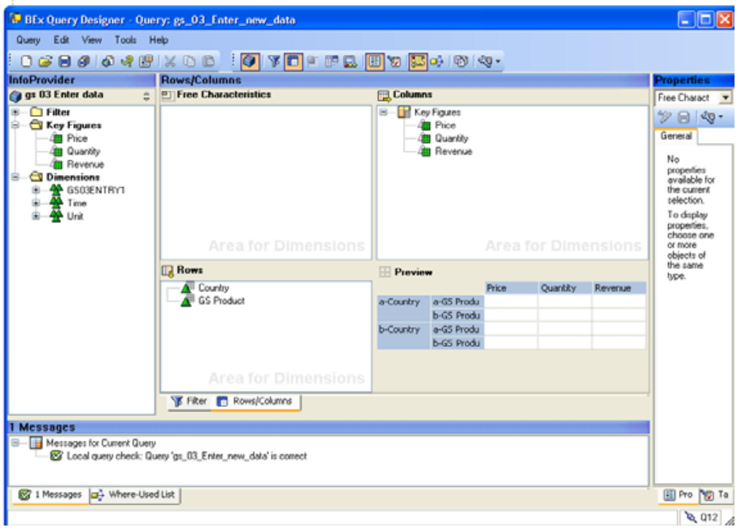
Selection of key figures to be reposted

Changed	Key figure text
✓	Price
✓	Revenue
✓	Quantity

From-values and to-values for reposting

From	From Change	To	To Change
Delta Indicator: X	From Change	Delta Indicator: Not assigned	To Change

10. Build a query to enter data: Open the Bex Query Designer and create an input enabled query on the aggregation level ‘AG Enter Data’. Use the three key figures in the columns and the product in the rows. Make sure the key figures are set to planning enabled and set some selections for the characteristics. Save the query using an appropriate name (e.g. ‘Enter Data’).

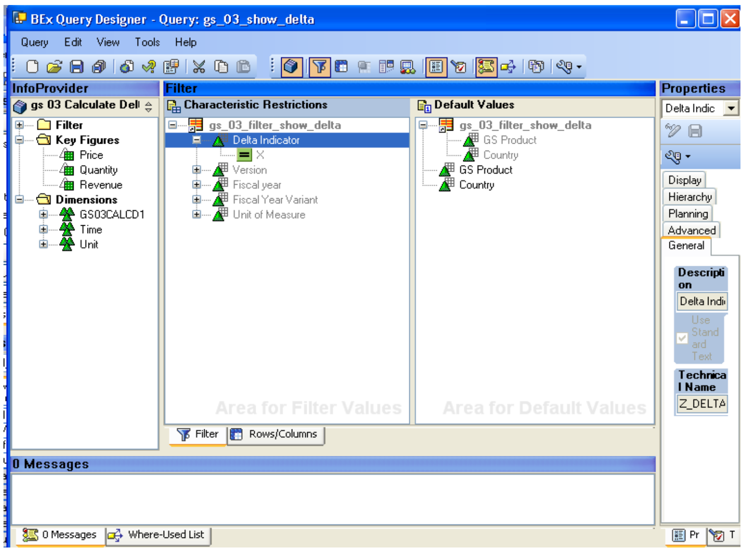


No properties available for the current selection.
To display properties, choose one or more objects of the same type.

		Price	Quantity	Revenue
a-Country	a-GS Produ			
	b-GS Produ			
b-Country	a-GS Produ			
	b-GS Produ			

1 Messages
Messages for Current Query
Local query check: Query 'gs_03_Enter_new_data' is correct

11. Build a query to display the delta records: create a second query, now use aggregation level ‘AG Calc Delta’. This query does not have to be a plan query. Use the same filter selections as in the above query. In addition use the delta indicator in the filter and set the filter value to ‘X’. Use the same characteristics in the rows as in the first query. Make sure that you switch on the zero elimination for rows and columns in your query. We will use this query to show all the records that have been changed in the current planning step. Nevertheless there might be already some zero records from previous executions in the plan buffer or in the data base which we do not want to show up in the query result. Call the query ‘Show Delta’.



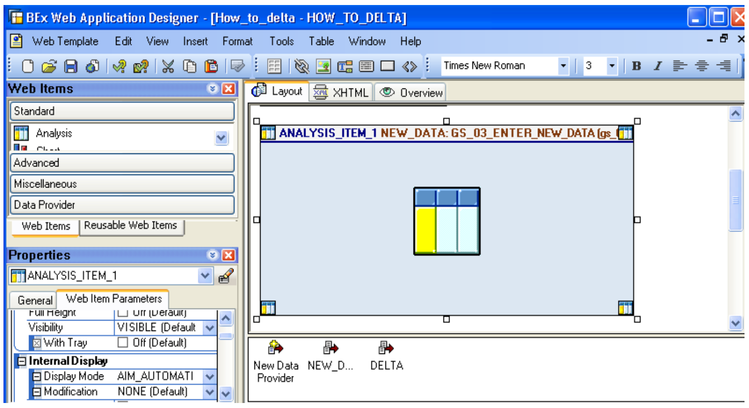
3.2 Using Delta Handling in a Web Application

We now have created the necessary objects to build up an application. We will start with a Web Application.

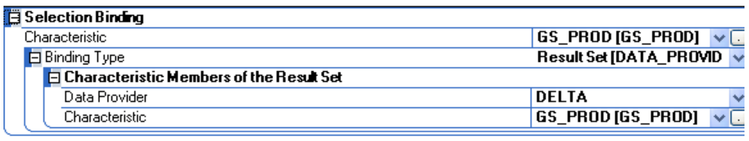
We want to build up an application where the user can change the price or the sold quantities of a product and the can calculate the revenue. The revenue should only be calculated for those products where either the price or the quantity has changed. We use the query ‘Enter Data’ for changing our plan data. Once the changed data is written into the plan buffer the derivation fills the delta indicator in each changed record. Thus if we have a look at query ‘Show Delta’ we will see all those delta records. The revenue calculation now has to run for all products which are contained in the result of our query ‘Show Delta’. Also the repost function has to run for exactly the products that are show in the result of this query.

With SAP NetWeaver 7.0 SPS 14 there is a new mapping type in the Web Application designer that allows use to use the result of a query and thus to use all products shown in ‘Show Delta’ as a filter of our planning functions.

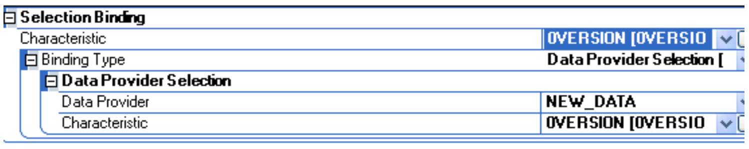
1. Create the Data Providers: Go to the Bex Web Application Designer and create a new web application. In the web application create a data provider based on the query ‘Enter Data’ (NEW_DATA) and an analysis item showing the data. Create a second data provider based on the query ‘Show Delta’ (DELTA).



2. Create buttons for the planning functions: Create a button group and within that create a button for the revenue calculation planning function. Choose ‘Execute Planning Function’ as command. Enter the name of the planning function (from the modeler). Now for each characteristic used in your aggregation level you should specify a selection by binding (you might leave out characteristics that have the same value in each record on the data base such as the fiscal year variant).



Selection Binding	
Characteristic	GS_PROD [GS_PROD]
Binding Type	Result Set [DATA_PROVID
Characteristic Members of the Result Set	
Data Provider	DELTA
Characteristic	GS_PROD [GS_PROD]



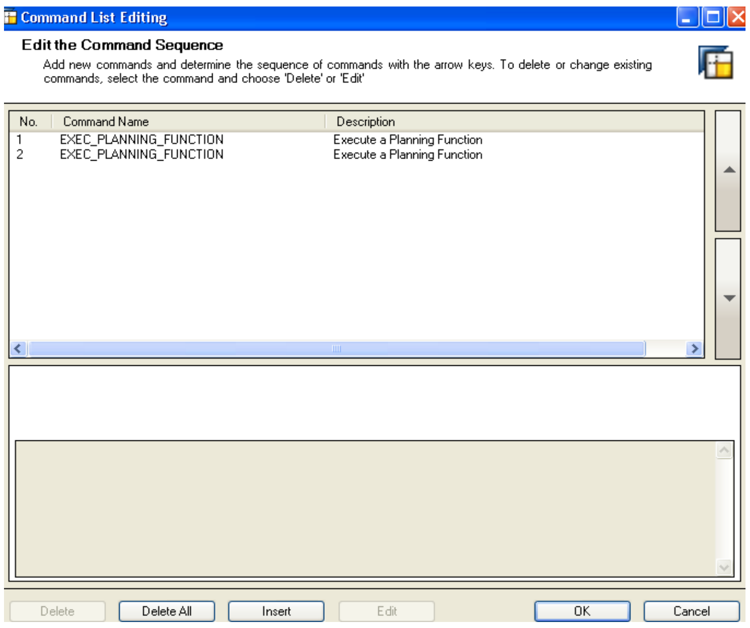
Selection Binding	
Characteristic	0VERSION [0VERSIO
Binding Type	Data Provider Selection [
Data Provider Selection	
Data Provider	NEW_DATA
Characteristic	0VERSION [0VERSIO

3. For the characteristic product you should use the binding type ‘Result

Set' and get the products from the Data Provider 'DELTA'.

4. For all other characteristics you can use the selection from the query 'Enter Data' and use the binding type 'Data Provider Selection' from Data Provider 'NEW_DATA'.

5. Create the repost function: on the same button create a second command (command sequence) that does the repost. Again use the command 'Execute Planning Function', enter the name of the repost function, and use the same mappings as with the revenue calculation. Save your Web Application.



Edit the Command Sequence

Add new commands and determine the sequence of commands with the arrow keys. To delete or change existing commands, select the command and choose 'Delete' or 'Edit'

No.	Command Name	Description
1	EXEC_PLANNING_FUNCTION	Execute a Planning Function
2	EXEC_PLANNING_FUNCTION	Execute a Planning Function

You can execute you planning application now. Change the quantity or the price for one product and run the planning function. From the messages sent from the planning function you will see that the planning function only runs on one product. Surely the model also works if you change more than one record (see note 1 below).

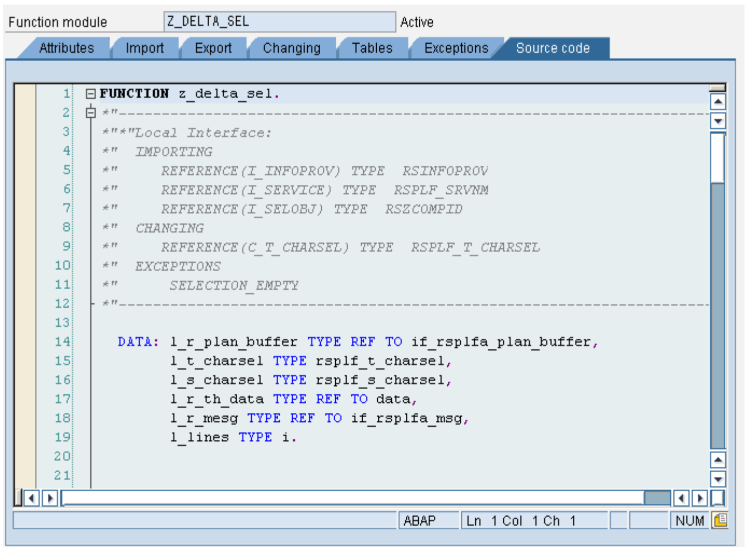
3.3 Using Delta Handling with a Front End - Independent Solution

The solution above depends on the special mapping only available in the Bex Web Application Designer. We will now show a solution that uses an ABAP Exit and is thus independent of the front end. The solution can be used in web applications, in Bex Analyzer Workbooks, and in planning sequences (online or in process chains).

We will build up the same application in the Bex Analyzer now. We will reuse the queries we have created above.

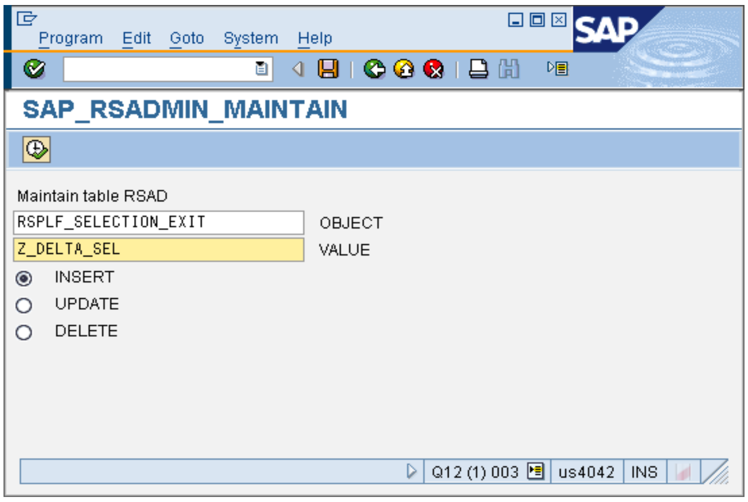
Before starting the solution have a look at note 1101726 and implement the ABAP coding.

1. Create the function module: go to the transaction se80, create a function group (or use an existing one) and in there create a function module (Z_DELTA_SEL). Have a look at the appendix for an example of the function module. You can also refer to note 1101726 for the interface and a sample implementation. The function module is used to change selection upon execution: before the planning function is executed, the function module is called and can change the selection that is used for executing the planning function.



```
FUNCTION z_delta_sel.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     REFERENCE(I_INFOPROV) TYPE  RSINFOPROV
*"     REFERENCE(I_SERVICE) TYPE  RSPLF_SRVNM
*"     REFERENCE(I_SELOBJ) TYPE  RSZCOMPID
*"  CHANGING
*"     REFERENCE(C_T_CHARSEL) TYPE  RSPLF_T_CHARSEL
*"  EXCEPTIONS
*"      SELECTION_EMPTY
*"----------------------------------------------------------------------

  DATA: l_r_plan_buffer TYPE REF TO if_rsplfa_plan_buffer,
        l_t_charsel TYPE rsplf_t_charsel,
        l_s_charsel TYPE rsplf_s_charsel,
        l_r_th_data TYPE REF TO data,
        l_r_mesg TYPE REF TO if_rsplfa_msg,
        l_lines TYPE i.
```

2. Register the function module: Run the report SAP_RSADMIN_MAINTAIN as described in the note. Please note that this function module now will be called for all selections.



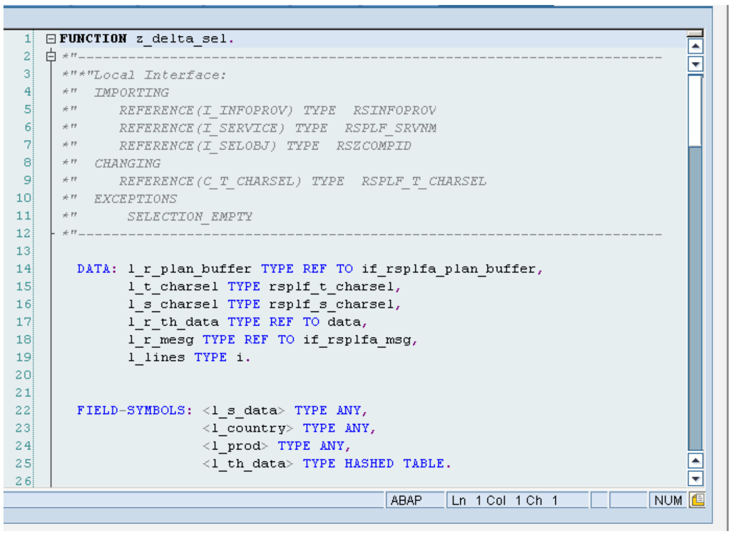
SAP_RSADMIN_MAINTAIN

Maintain table RSAD

RSPLF_SELECTION_EXIT	OBJECT
Z_DELTA_SEL	VALUE

[✓ INSERT] UPDATE DELETE

3. Fill the function module with coding: the exit is using the same trick as we have used in our solution in the WAD. The function module reads the changed records from the aggregation level 'AG Show Delta' and determines which products (and countries) have been changed. These products (and countries) are filled into the selection. Use the sample coding below and adapt the names of characteristics or aggregation levels if necessary.

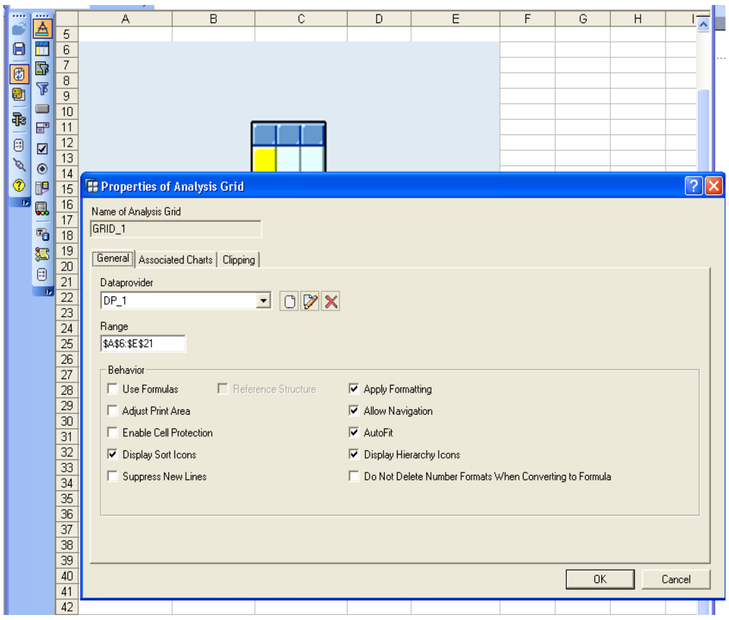


```
FUNCTION z_delta_sel.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     REFERENCE(I_INFOPROV) TYPE  RSINFOPROV
*"     REFERENCE(I_SERVICE) TYPE  RSPLF_SRVNM
*"     REFERENCE(I_SELOBJ) TYPE  RSZCOMPID
*"  CHANGING
*"     REFERENCE(C_T_CHARSEL) TYPE  RSPLF_T_CHARSEL
*"  EXCEPTIONS
*"      SELECTION_EMPTY
*"----------------------------------------------------------------------

  DATA: l_r_plan_buffer TYPE REF TO if_rsplfa_plan_buffer,
        l_t_charsel TYPE rsplf_t_charsel,
        l_s_charsel TYPE rsplf_s_charsel,
        l_r_th_data TYPE REF TO data,
        l_r_mesg TYPE REF TO if_rsplfa_msg,
        l_lines TYPE i.


  FIELD-SYMBOLS: <l_s_data> TYPE ANY,
                 <l_country> TYPE ANY,
                 <l_prod> TYPE ANY,
                 <l_th_data> TYPE HASHED TABLE.
```

4. At the beginning of the function module there is a check for the name of the current filter and/or the planning function. Use this check in order to make sure that the function module (and thus the delta mechanism) only runs when necessary. Note that the filter name is only filled if you execute the planning function via a planning sequence. In order to make your example runs correctly adapt the names of the planning functions accordingly.

```
  DATA: l_r_plan_buffer TYPE REF TO if_rsplfa_plan_buffer,
        l_t_charsel TYPE rsplf_t_charsel,
        l_s_charsel TYPE rsplf_s_charsel,
        l_r_th_data TYPE REF TO data,
        l_r_mesg TYPE REF TO if_rsplfa_msg,
        l_lines TYPE i.


  FIELD-SYMBOLS: <l_s_data> TYPE ANY,
                 <l_country> TYPE ANY,
                 <l_prod> TYPE ANY,
                 <l_th_data> TYPE HASHED TABLE.

* only proceed this exit with the right filter and function

*check i_selobj = 'GS_03_FILTER_SHOW_DELTA'.
  CHECK i_service = 'REPOST' OR
        i_service = 'CALCREVENUE'.


* Only delta records should be selected. Only country
* or product are relevant, the other selections are
* constant.
```

5. Create a BEx Analyzer workbook for testing: create a new workbook with a DataProvider DP_1 containing the query ‘Enter Data’. Also create an Analysis Item showing the data.


Properties of Analysis Grid

Name of Analysis Grid: GRID_1

General | Associated Charts | Clipping

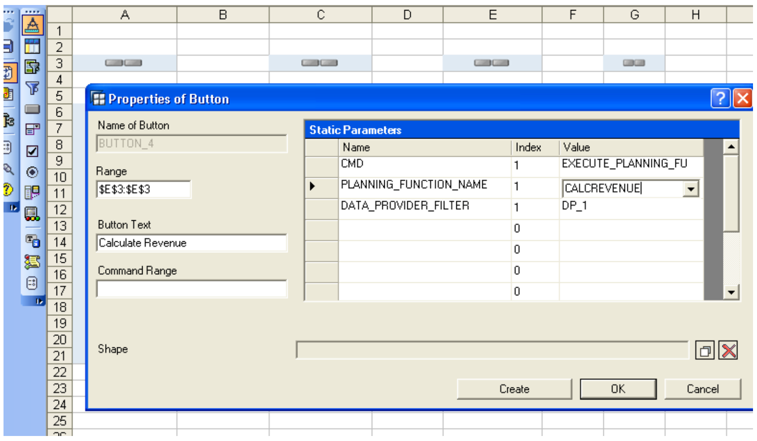
Dataprovider: DP_1

Range: $A$6:$E$21

Behavior
- ☐ Use Formulas
- ☐ Reference Structure
- ☑ Apply Formatting
- ☐ Adjust Print Area
- ☑ Allow Navigation
- ☐ Enable Cell Protection
- ☑ AutoFit
- ☑ Display Sort Icons
- ☑ Display Hierarchy Icons
- ☐ Suppress New Lines
- ☐ Do Not Delete Number Formats When Converting to Formula

OK | Cancel

6. Include the planning functions: create a button that executes a planning function. Choose the name of the revenue calculation and use DP_1 as a DataProvider. Create a second button for the repost function. Use the same DataProvider.


Properties of Button

Name of Button: BUTTON_4

Range: $E$3:$E$3

Button Text: Calculate Revenue

Command Range:

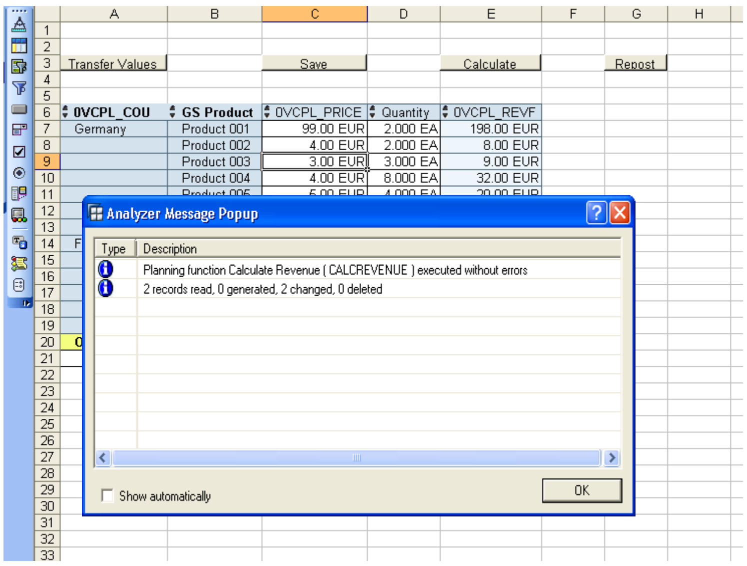
Static Parameters

Name	Index	Value
CMD	1	EXECUTE_PLANNING_FU
PLANNING_FUNCTION_NAME	1	CALCREVENUE
DATA_PROVIDER_FILTER	1	DP_1
	0	
	0	
	0	
	0	

Shape

Create | OK | Cancel

7. Test the planning application: start the workbook and test it. Again you will see from the messages that the planning function only calculates the revenues for the products that have been touched.



0VCPL_COU	GS Product	0VCPL_PRICE	Quantity	0VCPL_REVF
Germany	Product 001	99.00 EUR	2.000 EA	198.00 EUR
	Product 002	4.00 EUR	2.000 EA	8.00 EUR
	Product 003	3.00 EUR	3.000 EA	9.00 EUR
	Product 004	4.00 EUR	8.000 EA	32.00 EUR

Analyzer Message Popup

Type	Description
ℹ	Planning function Calculate Revenue (CALCREVENUE) executed without errors
ℹ	2 records read, 0 generated, 2 changed, 0 deleted

Note 1: The model also works if you change several records in one step. Nevertheless it might be the case that the planning function runs on a larger set of records than just the changed ones. Let us assume you have more characteristics in the aggregation level and the following records in your query:

Product	Country	Price
Prod001	**DE**	**100**
Prod001	FR	120
Prod002	DE	110
Prod002	**FR**	**130**
Prod002	USA	140

Now assume that we change the two records printed in bold face. The selection for the characteristic product is ‘Prod001 and Prod002’, the selected countries are ‘DE’ and ‘FR’. Thus the planning function will recalculate the revenue for the first four records. This is a little disadvantage due to the way data is selected in SAP BI but still we can use our model to restrict the planning function to the smallest BI selection containing the changed data.

Note 2: In our model we execute two planning functions (a revenue calculation and a repost) instead of just a revenue calculation on a larger set of data. As there is always some overhead when executing planning function the above model may lead to somewhat longer runtimes provided that either very few records are contained in the selection of the query ‘Enter Data’ or nearly all available data has been changed. In all other cases (which are quite normal in planning environments) our model will provide a better runtime.

Note 3: As stated above the application produces zero records that are stored in the data base. Thus if we do not switch on the zero elimination in our query ‘Show Delta’ the revenue calculation and the repost would run on all records within the selection, that have either been changed in the current session or in a session before.

4 Appendix

In this section we include the coding that we have used above.

Derive Method in class CZL_DERIVE_DELTA:

```
method IF_RSPLS_CR_METHODS~DERIVE.
**TRY.
*CALL METHOD SUPER->IF_RSPLS_CR_METHODS~DERIVE
**  IMPORTING
**    e_t_mesg =
*   CHANGING
*     C_S_CHAS =
*     .
** CATCH cx_rspls_failed .
**ENDTRY.
  field-SYMBOLS: <l_delta> type any.
  assign component '/BIC/Z_DELTA' of STRUCTURE
      c_s_chas to <l_delta>.
  <l_delta> = 'X'.

endmethod.
```

Function Module for changing the selection:
In order to use this function module you will have to adapt the names of the filter, the aggregation level, and the characteristics to your scenario.

```
FUNCTION z_delta_sel.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     REFERENCE(I_INFOPROV) TYPE  RSINFOPROV
*"     REFERENCE(I_SERVICE) TYPE  RSPLF_SRVNM
*"     REFERENCE(I_SELOBJ) TYPE  RSZCOMPID
*"  CHANGING
*"     REFERENCE(C_T_CHARSEL) TYPE  RSPLF_T_CHARSEL
*"  EXCEPTIONS
*"      SELECTION_EMPTY
*"----------------------------------------------------------------------

  DATA: l_r_plan_buffer TYPE REF TO if_rsplfa_plan_buffer,
        l_t_charsel TYPE rsplf_t_charsel,
        l_s_charsel TYPE rsplf_s_charsel,
        l_r_th_data TYPE REF TO data,
        l_r_mesg TYPE REF TO if_rsplfa_msg,
        l_lines TYPE i.


  FIELD-SYMBOLS: <l_s_data> TYPE ANY,
                 <l_country> TYPE ANY,
                 <l_prod> TYPE ANY,
                 <l_th_data> TYPE HASHED TABLE.
```

```
* only proceed this exit with the right filter and function

*check i_selobj = 'CHANGE_SEL'.
  CHECK i_service = 'REPOST' OR
        i_service = 'CALCREVENUE'.

* Only delta records should be selected. Only country
* or product are relevant, the other selections are
* constant.

  CALL METHOD cl_rsplfa_plan_buffer=>if_rsplfa_plan_buffer~get_instance
    EXPORTING
      i_aggregation_level = 'AG_CALC_DELTA'
    RECEIVING
      r_plan_buffer       = l_r_plan_buffer
    EXCEPTIONS
      error               = 1
      OTHERS              = 2.
  IF sy-subrc <> 0.
    EXIT.
  ENDIF.

*we only want to know the delta records for country and product
  l_t_charsel = c_t_charsel.

  DELETE l_t_charsel WHERE iobjnm = '0COUNTRY'.
  DELETE l_t_charsel WHERE iobjnm = '0PRODUCT'.

  l_s_charsel-iobjnm = 'Z_DELTA'.
  l_s_charsel-sign = 'I'.
  l_s_charsel-opt = 'EQ'.
  l_s_charsel-low = 'X'.
  CLEAR l_s_charsel-high.

  INSERT l_s_charsel INTO TABLE l_t_charsel.

* get the changed records
  CALL METHOD l_r_plan_buffer->get_data
    EXPORTING
      i_t_charsel            = l_t_charsel
      i_include_zero_records = rs_c_false
      i_r_msg                = l_r_mesg
    IMPORTING
      e_r_th_data            = l_r_th_data
    EXCEPTIONS
      error                  = 1
      OTHERS                 = 2.
  IF sy-subrc <> 0.
    EXIT.
  ENDIF.

  ASSIGN l_r_th_data->* TO <l_th_data>.

  DESCRIBE TABLE <l_th_data> LINES l_lines.

  IF l_lines IS INITIAL.
* no deltas!
    MESSAGE i001(rsplf)
      WITH 'No changed data yet' RAISING SELECTION_EMPTY.
  ENDIF.
  LOOP AT <l_th_data> ASSIGNING <l_s_data>.
    ASSIGN COMPONENT 'COUNTRY' OF STRUCTURE <l_s_data> TO <l_country>.
      IF sy-subrc <> 0.
        MESSAGE i001(rsplf)
          WITH 'Error in Function Module' RAISING SELECTION_EMPTY.
      ENDIF.
    ASSIGN COMPONENT 'PRODUCT' OF STRUCTURE <l_s_data> TO <l_prod>.
      IF sy-subrc <> 0.
        MESSAGE i001(rsplf)
          WITH 'Error in Function Module' RAISING SELECTION_EMPTY.
      ENDIF.

   READ TABLE l_t_charsel WITH KEY iobjnm = '0COUNTRY' low = <l_country>
     TRANSPORTING NO FIELDS.

    IF sy-subrc <> 0.
      l_s_charsel-iobjnm = '0COUNTRY'.
      l_s_charsel-sign = 'I'.
      l_s_charsel-opt = 'EQ'.
      l_s_charsel-low = <l_country>.
      CLEAR l_s_charsel-high.

      INSERT l_s_charsel INTO TABLE l_t_charsel.
    ENDIF.

    READ TABLE l_t_charsel WITH KEY iobjnm = '0PRODUCT' low = <l_prod>
      TRANSPORTING NO FIELDS.

    IF sy-subrc <> 0.
      l_s_charsel-iobjnm = '0PRODUCT'.
      l_s_charsel-sign = 'I'.
      l_s_charsel-opt = 'EQ'.
      l_s_charsel-low = <l_prod>.
      CLEAR l_s_charsel-high.

      INSERT l_s_charsel INTO TABLE l_t_charsel.
    ENDIF.

  ENDLOOP.

  DELETE l_t_charsel WHERE iobjnm = 'Z_DELTA'.


  c_t_charsel = l_t_charsel.



ENDFUNCTION.
```