Semantic Compression in SAP NetWeaver Mobile 7.1

Applies to:

SAP Net Weaver Mobile 7.1 and above (Data Orchestration Engine). For more information visit Mobile Home Page

Summary

In case of push scenario, data is pushed from (usually an SAP) backend whenever there is a relevant transaction carried out and there can be many such transactions carried out each day. Every push results in a data load cycle in the Data orchestration engine (DOE). Semantic compression is used to reduce the periodicity of these data load cycles.

Author: Pramod M

Company: SAP Labs India Pvt. Ltd.

Created on: 26 October 2010

Author Bio

Pramod M Has been a developer in the SAP Net Weaver Mobile (DOE) team from the past 22 months.

Table of Contents

Overview

Whenever transactions are carried out in a transaction system which acts as a back-end to the DOE, there are ways to push the state change of the instance (create/update/delete) onto the DOE. One of them is pushing the entire instance (Instance Push), the other only the keys and the relevant action (Key Push). This is in contrast to the pull where the DOE pulls all the instances and compares them with the existing data in the CDS. Only the delta is persisted in DOE and the rest ignored.

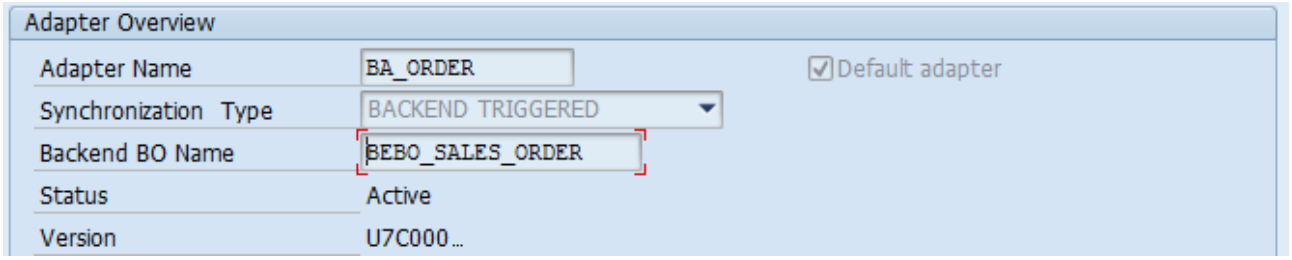
However, during push, every transaction results in backend pushing the data to the DOE, effectively causing the DOE to undergo a data load cycle. This can lead to performance hindrances as there can be innumerable number of transactions in the back-end and each of them triggering a data load cycle will increase the processing on the DOE side. More so in case of Key Push because the back-end only gives the keys that are to be processed and the DOE has to again make an RFC call to the back-end to get the complete instance. To overcome this hurdle and to improve the performance, instead of triggering data load cycle for every key pushed from the back-end the keys are grouped and a periodically run job triggers performs the load with the grouped keys. This is called Semantic Compression. This is nothing but bulk instance processing.

Semantic Compression

Suitable Model in the DOE

There has to minor changes made in the back-end adapter definition for it to be used in a push scenario. The change that is mandatory for both Key and instance push is to make the adapter 'Back-end triggered'. Also to make sure that the data is being pushed to the right Data Object, the backend business object name has to be provided. It is using the back-end business object (BEBO) name the back-end will push the data into the DOE.

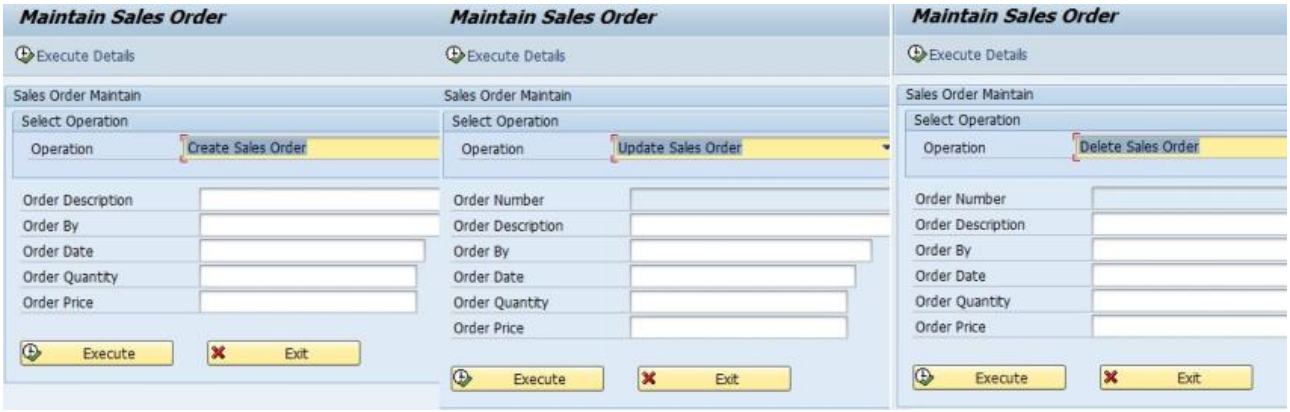
NOTE: It may so happen that there might be 2 different data objects in the DOE whose source of data may be same in the backend (i.e. same backed business object). In this case when the backend pushes the data using Key Push, it does so for both these data objects in undefined order.



Backend Application

For key push to work, backend application has to be modified to push the keys and the corresponding operations on them to the DOE. As an example we have developed a simple backend application which maintains sales orders as well as pushes them (keys) as and when some operations are performed on them. For this purpose an RFC Function Module (SMMW_BE_CALL_DELTABO) has been exposed by DOE to the backend. This can be used to push keys from the backend to DOE, which in turn calls the Backend and picks up the entire instance.

The sample back-end application looks like below.



Code sample for when a sales order is created in the backend. The same can be applied for modification and deletion as well.

```
case w_operation.
  when 'CREATE'.
    data : ls_order_temp type zpm_sorder_hdr-sales_order.
    select max( sales_order ) from zpm_sorder_hdr into ls_order_temp.
    ls_sorder-sales_order = ls_order_temp + 1.
    insert into zpm_sorder_hdr values ls_sorder.
    if sy-subrc = 0.
      " Preparing for key push
```

```
        ls_r3keys-r3key = ls_sorder-sales_order.
        ls_r3keys-msgfn = '009'. " This value is for new entry.
                                 " 003 is for delete and 004 for modify.
        append ls_r3keys to lt_r3keys.
        condense ls_sorder-sales_order no-gaps.
        concatenate 'Order' ls_sorder-sales_order
              'successfully created' into lv_message separated by space.
        message lv_message type 'I'

      call function 'SMMW_BE_CALL_DELTABO'
        destination 'NONE'
        exporting
          bebo_name = 'BEBO_SALES_ORDER'
  " This is the name of the Backend Business Object.
        tables
          r3keys    = lt_r3keys.  " This table contains the keys to be pushed and the
                                     " operations to be performed on them
      endif.
```

Note that the destination NONE here has to be replaced with a suitable RFC connection connecting your backend to the DOE.
* Also kindly note that the Initial load has to be performed before semantic compression for push is implemented.

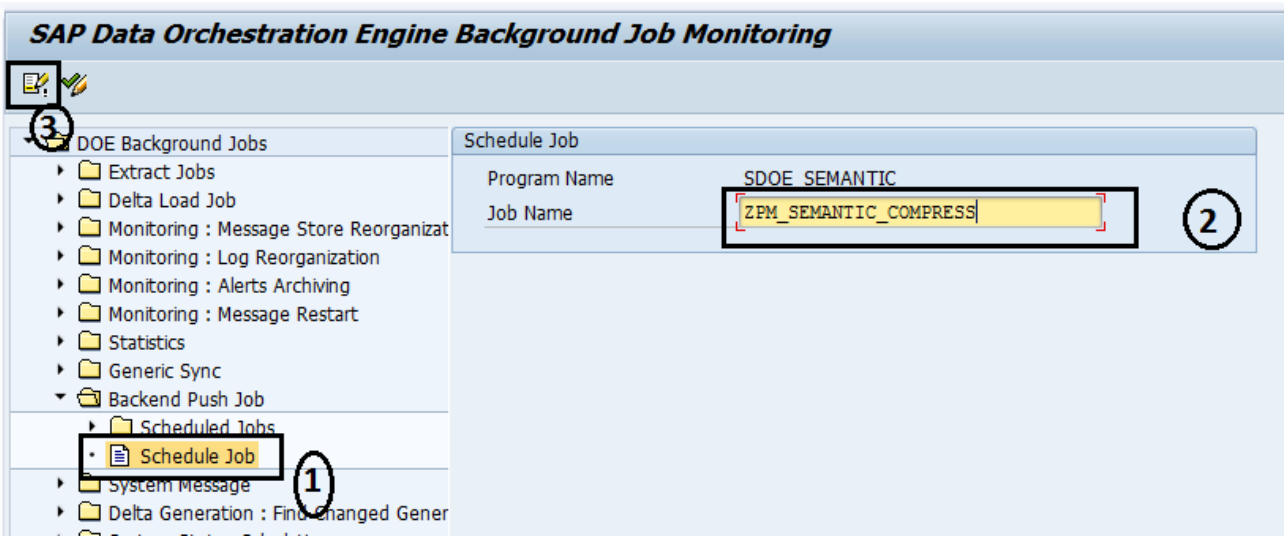
What the piece of code above does is, it saves the data entered by the user on the screen onto a database table in the backend. On successful order creation this order is pushed to the DOE using the given RFC function module along with the keys and the operation to be performed on them.

Periodic job processing the pushed keys.

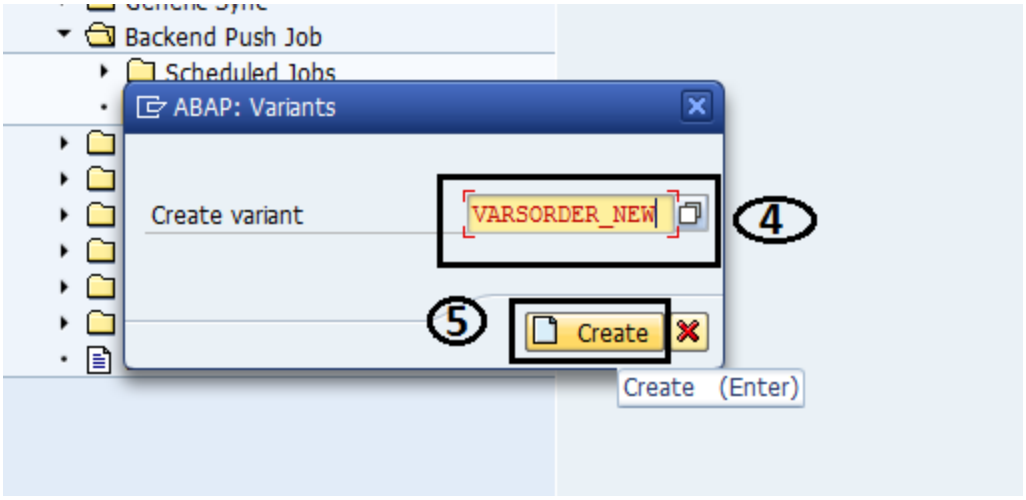
By this time the initial load to DOE would have been performed and the back-end would have started pushing keys to the DOE and the DOE would have started processing them instantly.

But semantic compression will not have been achieved yet. For this to happen a background job has to be scheduled which runs periodically(on your choice) using the report SDOE_SEMANTIC and a variant with your BEBO as the input.

Use Transaction SDOE_BG_JOB_MONITOR to schedule a periodic job. This is a central transaction containing a list of DOE related background jobs that can be scheduled and monitored.

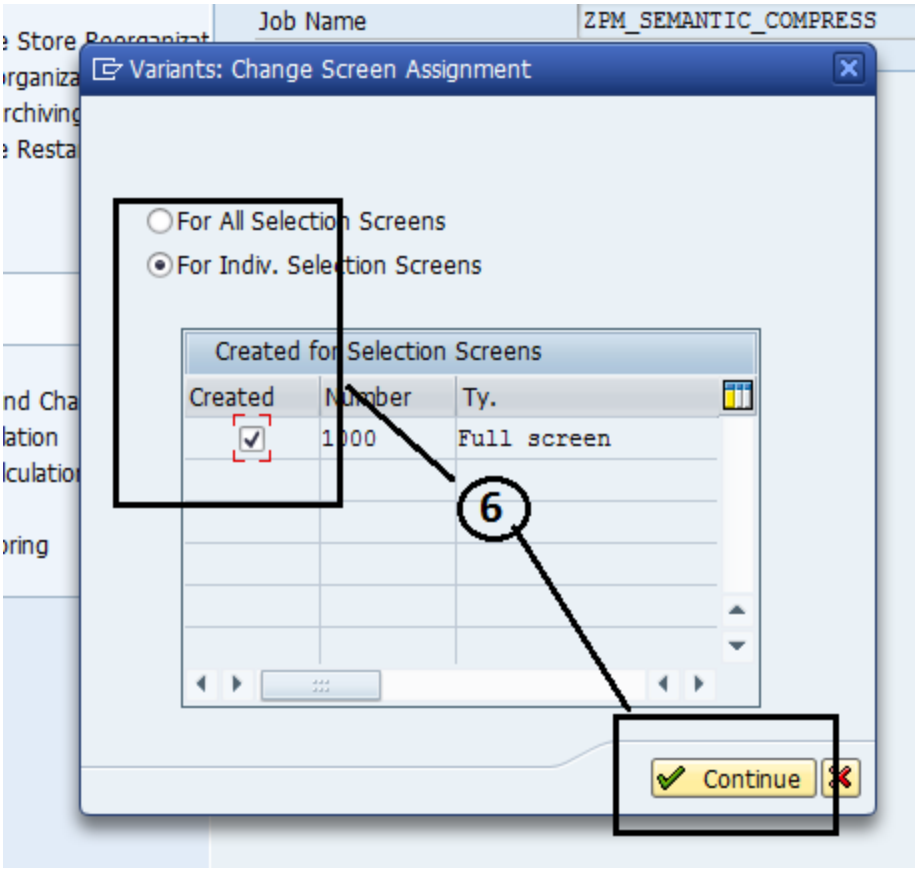


1. Click on ‘Schedule Job’ under the ‘Backend Push Job’ in BG job monitor.
2. Give a name for the job.
3. Click on ‘Create Variant’ to create a variant for the job.

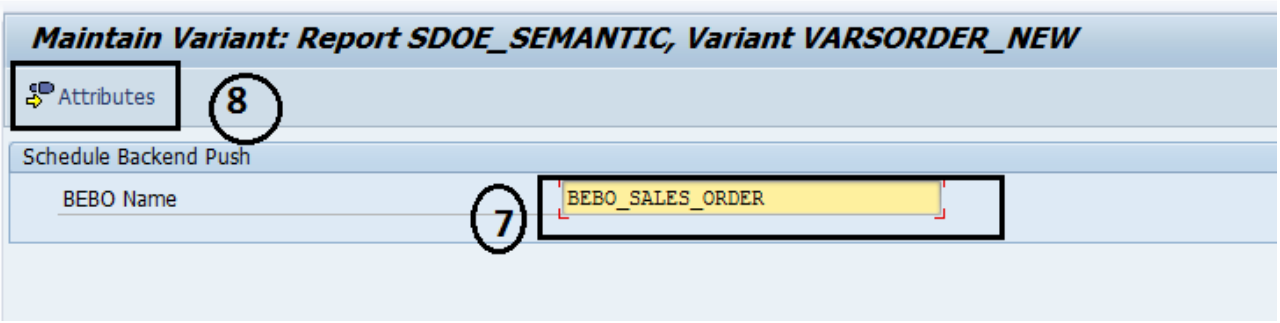


Give a name for the variant.

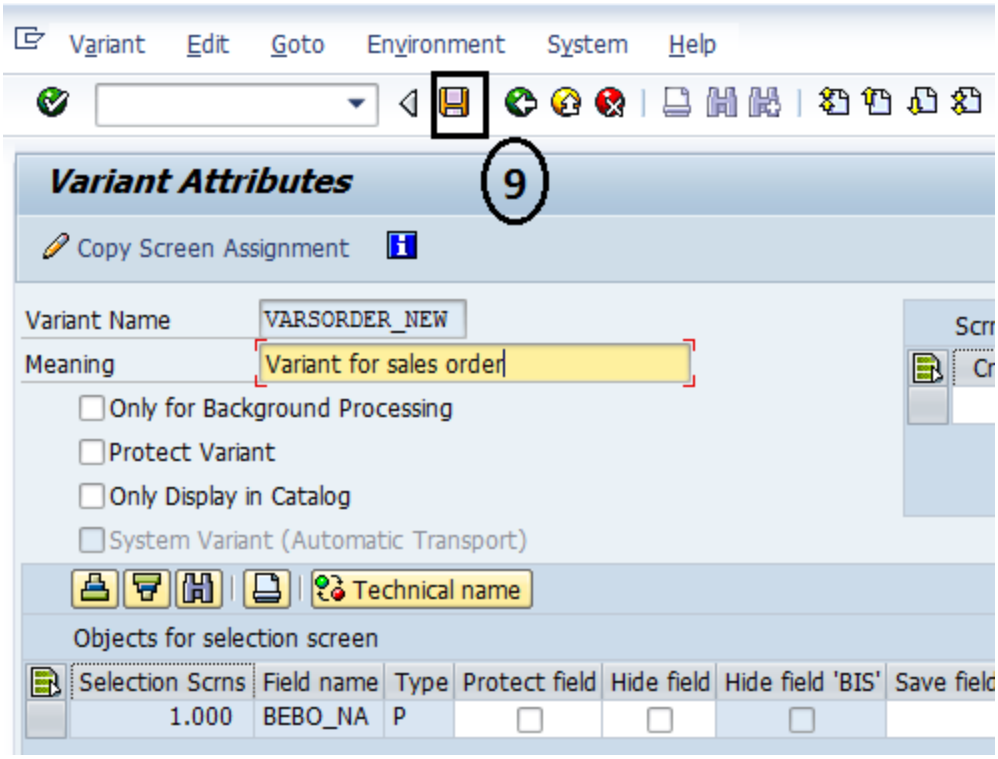
1. Click on ‘Create’.



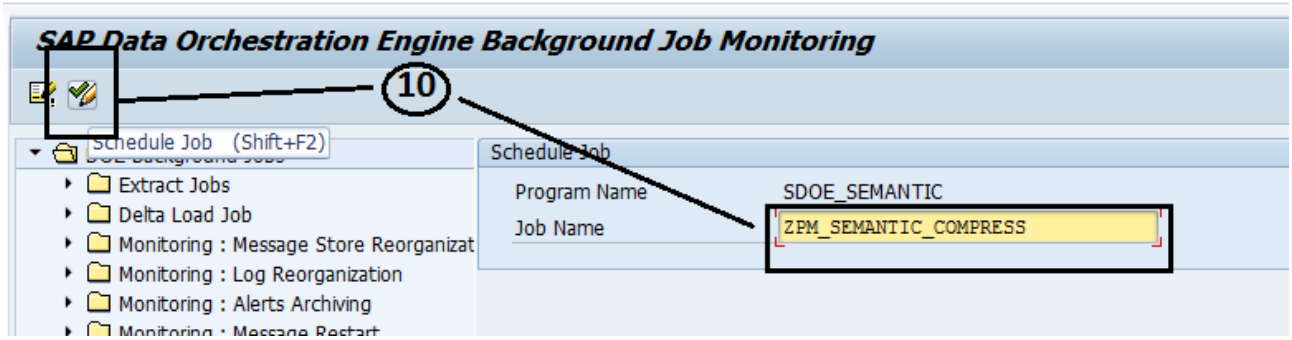
1. This screen may or may not pop up(depends on whether you have scheduled a job before in the same session). If it pops up please select choose as shown and continue. It takes you to the next screen which is nothing but the selection screen of the report SDOE_SEMANTIC.



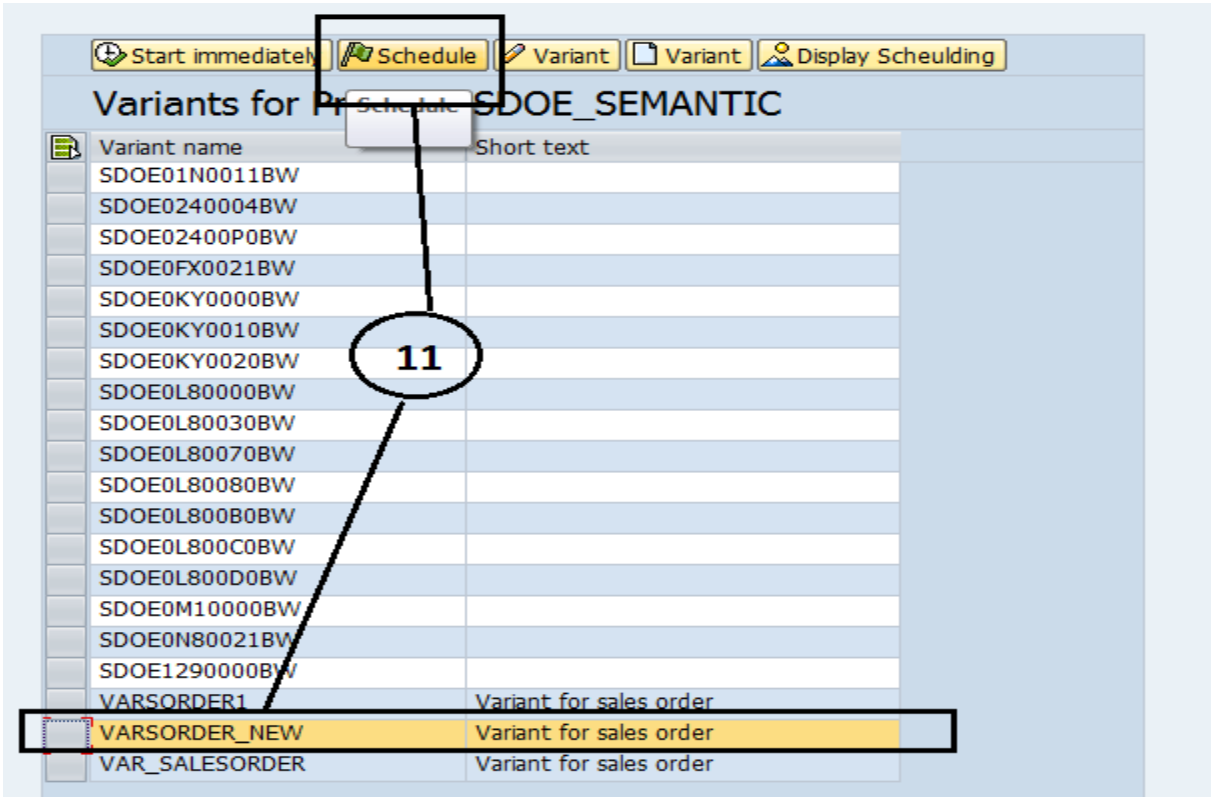
2. In this screen give the name of the Back-end business object that you are referring to. (Which you have entered in the back-end adapter screen)
3. Click on ‘attributes’ to set the attributes of the variant.



4. Set the attributes (nothing to set explicitly here) and click on ‘Save’ and go back to the BG job monitor screen

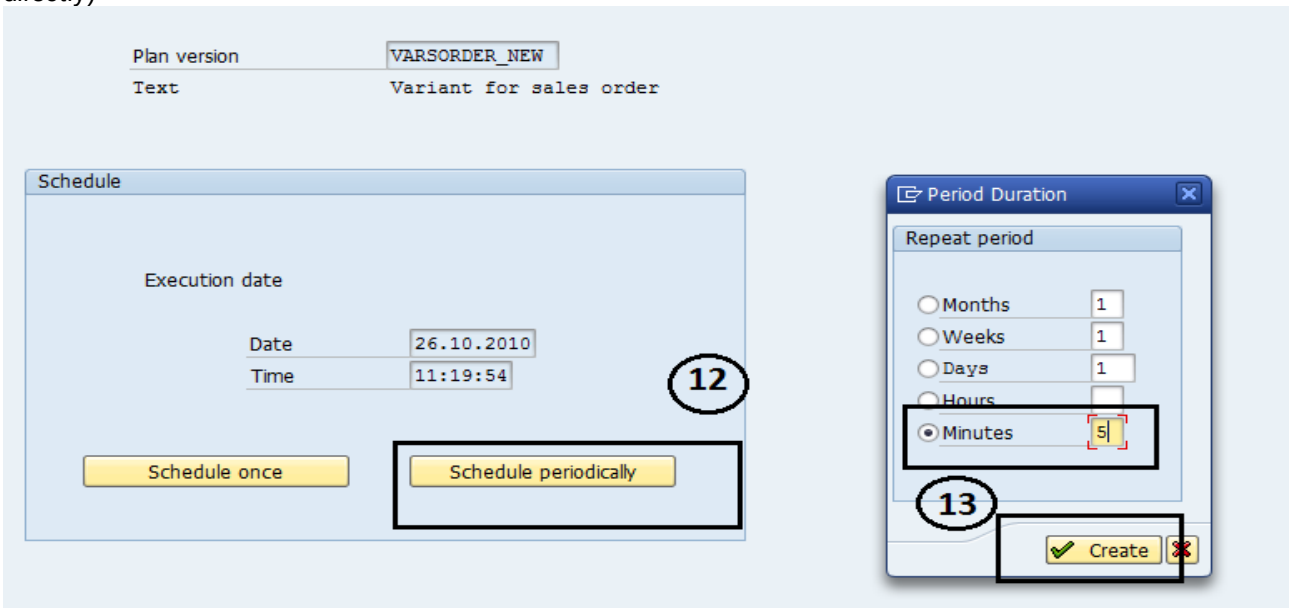


5. Enter the job name and click on ‘schedule job’.

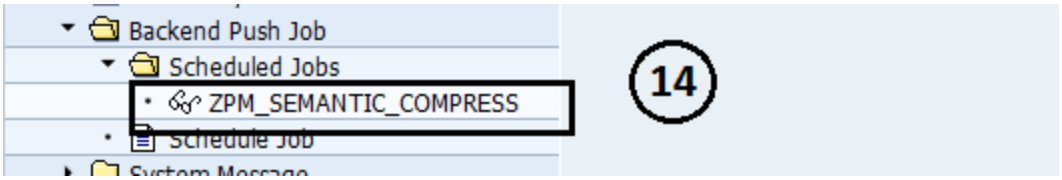


6. Select the variant name and click on schedule to run it periodically. It will open a new screen.

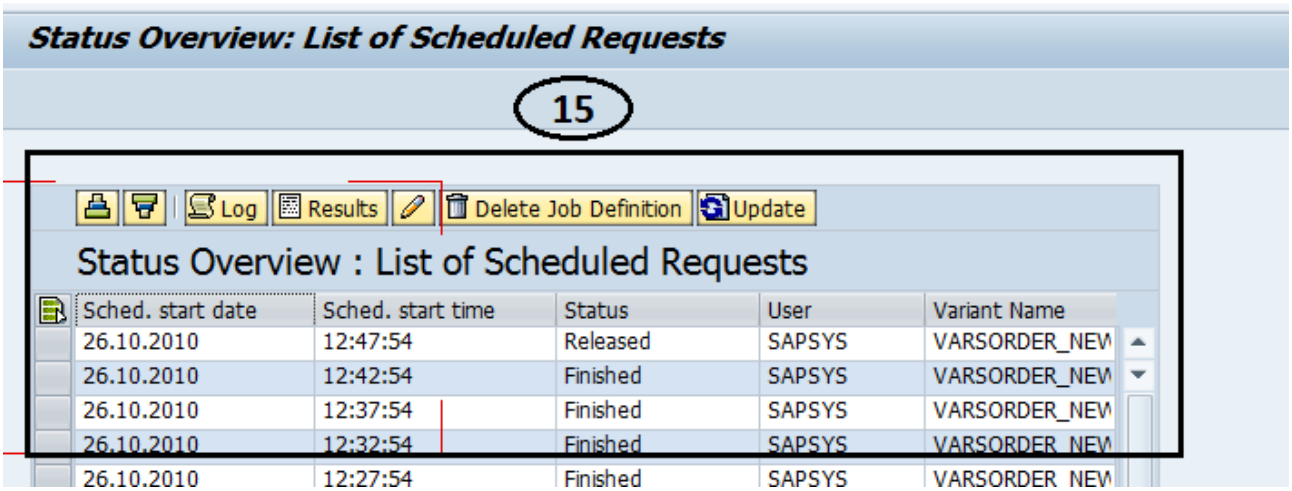
(Alternatively 1 - you can run it once by clicking start immediately 2 - also create a variant here directly)



7. In the next screen, click on ‘Schedule Periodically’.
8. Select the periodicity and click on ‘Create’. It will give a message stating successful job scheduling. On receipt of this message, go back to the BG Job monitor screen.



9. Under the 'Scheduled Jobs' folder of 'Backend Push Job' we can now see a job scheduled. Double Click on that.



10. This screen is used to for monitoring the job status.

Some scenarios during semantic compression in push

Between the 2 consecutive executions of the semantic compression job, there can be numerous changes to the instances in the back-end. Hundreds of data instances might be created and another hundred might be updated and deleted.

Amongst these transactions there might be cases where same instance might be operated on numerous times. Although the instance might be processed twice in the back-end, semantic compression ensures that it is processed only once on the DOE. Below given are all the possible combinations of change the instance might undergo in between 2 executions of the semantic compression job and the end result.

Instances ↓

T1	T2	T3	Final operation
Create	Update		Create
Create	Update	Delete	No operation
Update	Delete		Delete
Delete	Create (with same key)		Create (with same synckey)
Delete	Create (with same key)	Update	Create (with same synckey)

* DOE = Data Orchestration Engine

* BEBO = BackEnd Business Object

Related Content

Date Orchestration Engine - FAQs

For more information visit Mobile Home Page

Copyright