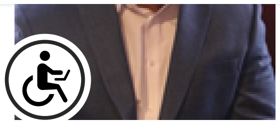
DON'T TRY CODING ABAP CORE DATA SERVICES WITHOUT READING THIS FIRST

ABAP for HANA | SAP HANA

March 18, 2017



Jon's focus has been in the areas of ABAP OOP and ABAP Web Dynpro. He can be reached at jon.andre@itpfed.com

As we described in the prior blog The ABAP Developer Road Map to SAP HANA, with the advent of SAP HANA, there has been a paradigm shift in the way business applications are developed. The rule-of-thumb is simple: Do as much as you can in the database to get the ***best performance***. This was coined as *"Code Pushdown"* by SAP. Well, this is also true for the underlying data models of the business applications.

Data modeling in ABAP typically involves organizing your data in database tables/views and often providing some additional high-level services for the applications using the appropriate ABAP frameworks. It is logical to conclude, from the Paradigm-shift of *Code* Pushdown, that to enable real-time businesses in HANA, we need some of these services ideally also brought closer to the database as well.

For SAP this presented several challenges. High-quality data models should provide a single definition and format for the data. They should be clear and unambiguous, reusable and flexible, even extensible. So how can you capture the semantics of the data model in the database so that the model can be easily reused by different consumers, e.g. by OData clients and by OLAP tools? How can you extend the meta-model to service your applications? What is the solution…?

INTRODUCTION TO CORED DATA SERVICES (CDS)

concerned with two specific ones…The lesser used option is HANA CDS, the database language that can used to create tables, views, and structures on the HANA database itself. Views created in HANA can consumed from the Netweaver AS using Native SQL. The second and most important variant of CDS that should concern ABAPers is the ABAP CDS. While significant differences have evolved between the two variants — for example, SAP HANA-based CDS obviously operates on SAP HANA, while ABAP-based CDS operates on most major database platforms as well as SAP HANA, and each has a different type of repository for development objects — both variants pursue the same goal: ***to represent central data definitions as a common basis for application development of all kinds.***

Let's look at each variant:

HANA CDS: the database language that can be used to create tables, views, and structures on the HANA database itself. Views created in HANA can be consumed from the Netweaver AS using Native SQL.

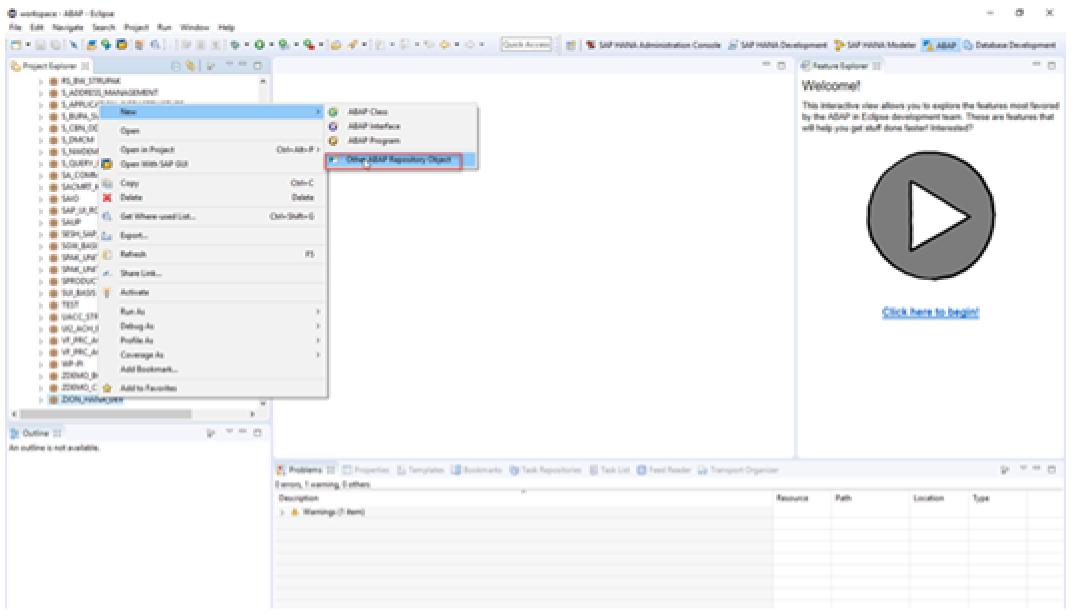
ABAP CDS: made available with SAP Netweaver 7.40 SP5, is a valuable tool to have when programming for HANA. However, ABAP CDS can be used even if the underlying database is not a HANA database, as it is an open DDL that is supported by many traditional databases as well. ABAP CDS is usually the best choice when designing and creating database views that will need to access the HANA database, and this will be the prime focus of this blog.

ABAP CDS uses an SQL-like syntax, enhanced with some useful additional features. Like any typical ABAP object, ABAP CDS files are also transportable between Netweaver AS systems, which is an advantage ABAP CDS has over its HANA CDS counterpart. Once transported, an ABAP CDS View will create and deploy the corresponding database view on the target database automatically (requiring no additional steps for the developer or transport manager).

t, we will go through a brief step-by-step guide on how to create a CDS view, as well as features of DDL source file.

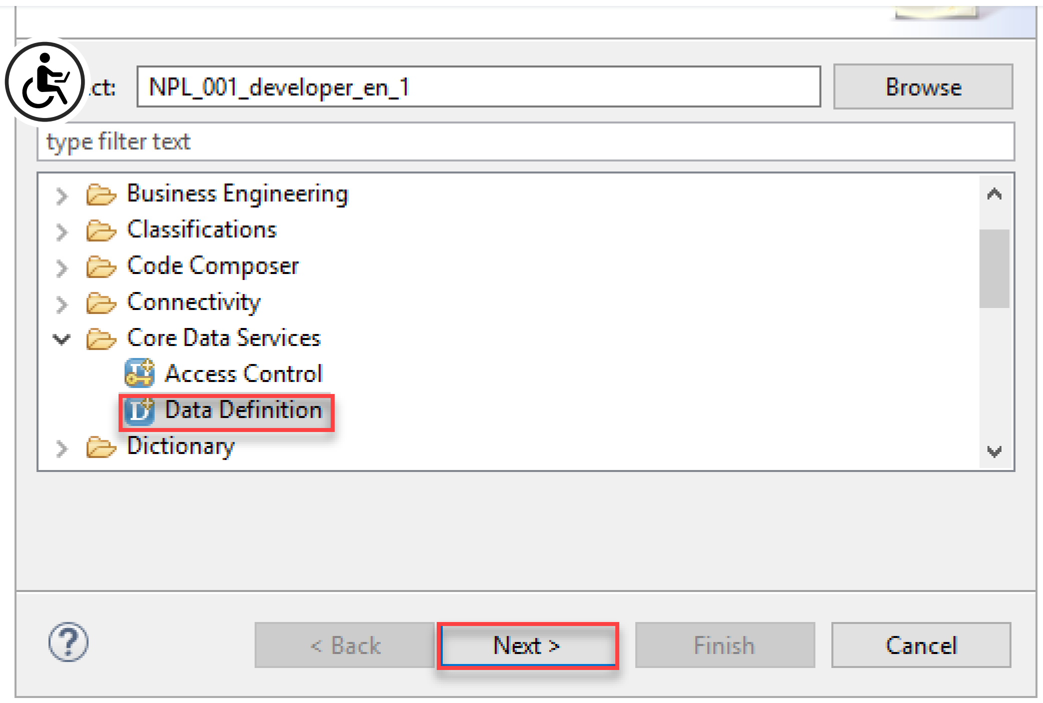
CREATING YOUR FIRST CDS VIEW

Prerequisites: You should have downloaded and installed the ABAP Development Tools extension for Eclipse already, as well as being within the ABAP perspective with desired package selected.

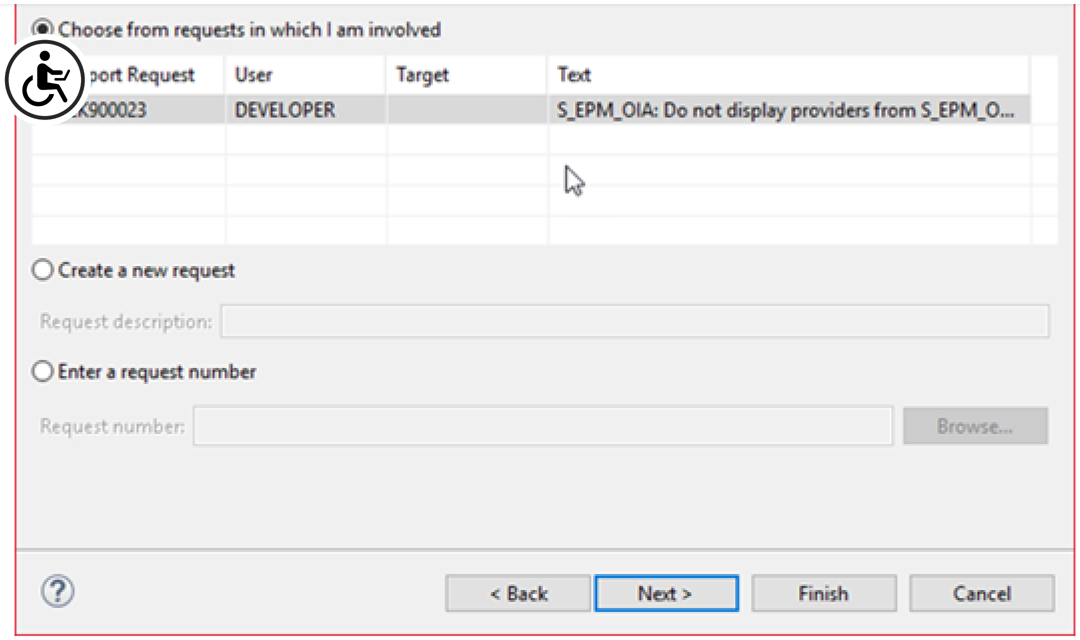


Step:1 Right-click on package the CDS View will be placed in

Step 2: Select New-> Other ABAP Repository Object



Step 3: Search for Core Data Services Folder and select Data Definition (depending on ABAP development tools version, this may say DDL Source)



[● Choose from requests in which I am involved]

port Request	User	Target	Text
K900023	DEVELOPER		S_EPM_OIA: Do not display providers from S_EPM_O...

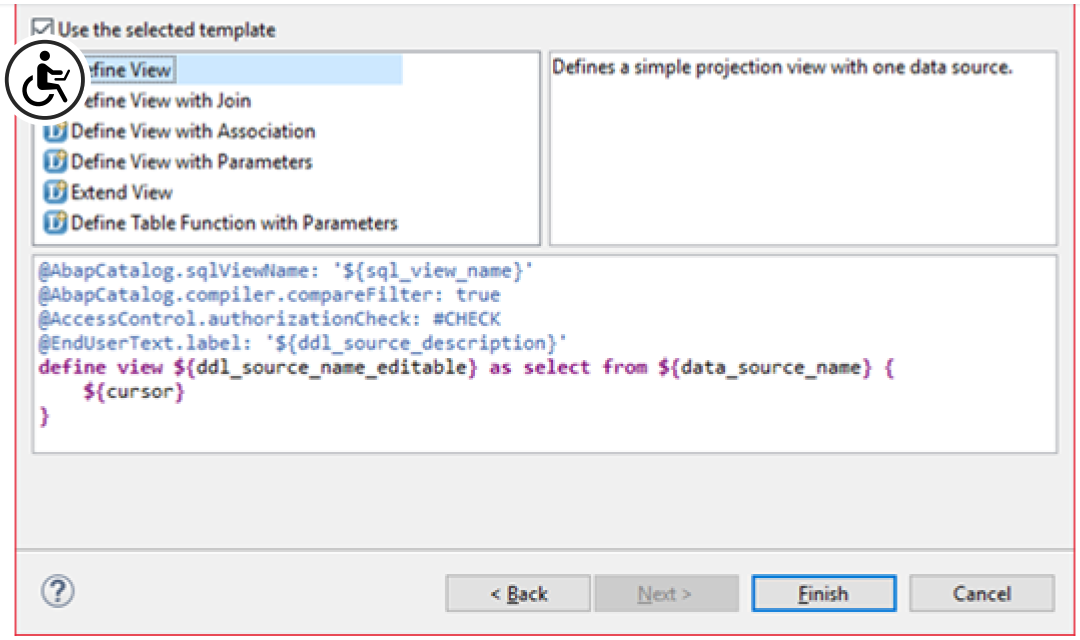
○ Create a new request

Request description:

○ Enter a request number

Request number:

Step 4: Select the transport to which you want the DDL source file attached. Once this has been done you can either select "Finish", which will automatically select the basic CDS view or you can select "Next", which will bring you to a list of different DDL source file templates to choose from. We will choose NEXT, so we can expore the differnet Templates.


Use the selected template
efine View
efine View with Join
Define View with Association
Define View with Parameters
Extend View
Define Table Function with Parameters
< Back
Next >
Finish
Cancel

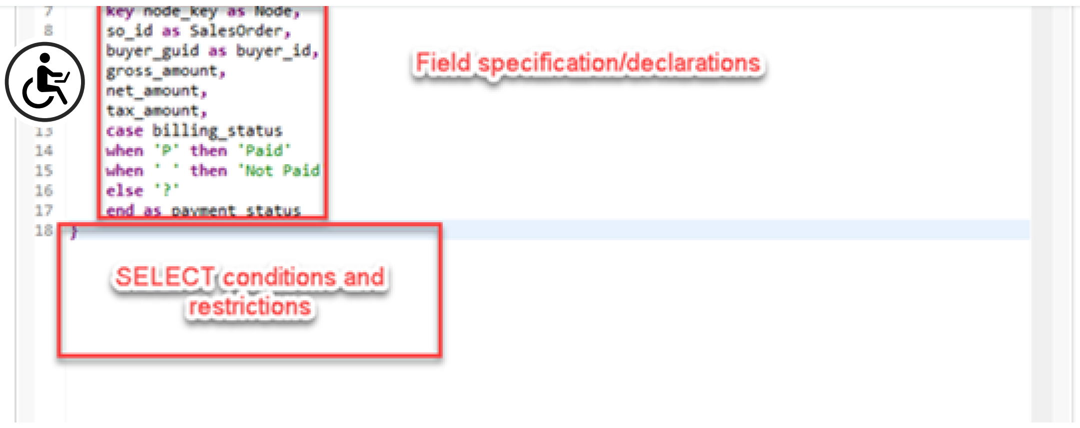
Defines a simple projection view with one data source.

```
@AbapCatalog.sqlViewName: '${sql_view_name}'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: '${ddl_source_description}'
define view ${ddl_source_name_editable} as select from ${data_source_name} {
    ${cursor}
}
```

Step 5: Choose the template you would like to begin with. For the first example, we will be starting with the basic view, which creates a CDS Entity beginning with the "Define View" keywords. Table functions are an option as well, but that will be covered in the future AMDP blog.

FIRST ABAP CDS EXAMPLE: VIEW ON SNWD_SO TABLE

This first CDS View example will be a simple SELECT of some fields from the SNWD_SO table, in addition to a generated field that will illustrate the CASE capabilities of CDS. This view will also illustrate the basic structure used for DDL source files. Take a look at the screen-shot below. We will go into each of the **red boxed** subsections in detail.



```
key node_key as Node,
so_id as SalesOrder,
buyer_guid as buyer_id,
gross_amount,
net_amount,
tax_amount,
case billing_status
when 'P' then 'Paid'
when ' ' then 'Not Paid
else '?'
end as payment_status
}
```

Field specification/declarations
SELECT conditions and restrictions

OPENING ANNOTATIONS – SUBSECTION

CDS Annotations contain metadata about the DDL source file in which they are contained. When Eclipse generates the CDS View template to work with, it also creates a skeleton for the initial annotations you see in this example. There are more Annotations available to CDS beyond the ones listed here, and fully covering all the Annotations would require an entire blog itself. For this introductory blog on CDS, I will just point out the two most important ones listed here: **@AbapCatalog.sqlViewName** and **@AbapControl.authorizationCheck**.

@AbapCatalog.sqlViewName is the only mandatory annotation for a non-extending CDS View source file (extending view will be explained in the next section). It precedes the SQL View name that will be attached to the current CDS View (also known as a CDS Entity). In the above example, the SQL View Name is 'zjontestview1'. While this SQL View name should NOT be used in your ABAP programs, it can be used in SE11 to display details about the current view (similar to how one would view a classical view).

@AbapControl.authorizationCheck specifies whether an authorization check should be performed for the current CDS view. Security and authorization is beyond the scope of this blog, but this annotation section will usually come after the opening Annotations but before the first curly bracket. Within this section, the developer specifies:

(1) The type of view (define or extend view)

(2) The CDS view name (in this case ZJON_CDS_VIEW_EXAMPLE),

(3) The source table or view (SNWD_SO)

(4) Any parameters (more on this later)

(5) Any joins/associations (more on this later)

FIELD DECLARATIONS AND SPECIFICATIONS – SUBSECTION

After the first curly bracket comes the desired fields from the table, as well as any fields that are to be computed. **This is the section that a developer will primarily utilize to take advantage of code-pushdown to allow the database to perform calculations.** Our example illustrates the ease of using aliases for fields, as well as the New ABAP CASE Construct. The CASE Construct allows a particular value to be returned based on the value of a table field. In our example, we are converting the billing status indicator into its real-world English meaning. If the billing status is 'P' for the current record, our calculated field payment_status will have the value 'Paid'. If the field is blank, the field will be 'Unpaid'. Finally, if the field is some unexpected value, the payment_status field will be a question mark.

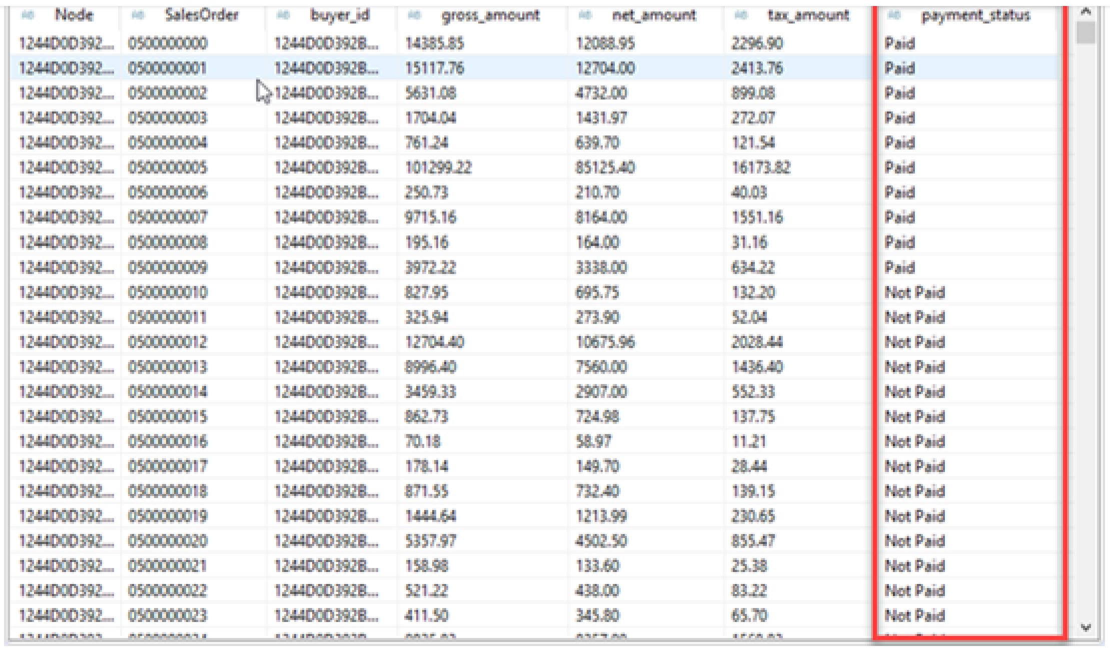
SELECT CONDITIONS AND RESTRICTIONS – SUBSECTION

CHECKING YOUR WORK

Ok now that we have created a basic CDS view for the table snwd_so table, let's check our work by a SQL preview with Eclipse.

CDS Views can be previewed right within the Eclipse editor. To do this, right-click on the created view, select "Open With" and then select "Data Preview" from the submenu. (Note that in other versions of the ABAP Development Tools, the "Data Preview" option may appear immediately in the menu when you right click.)

Below is the data preview for the example CDS View above, ZJON_CDS_VIEW_EXAMPLE:



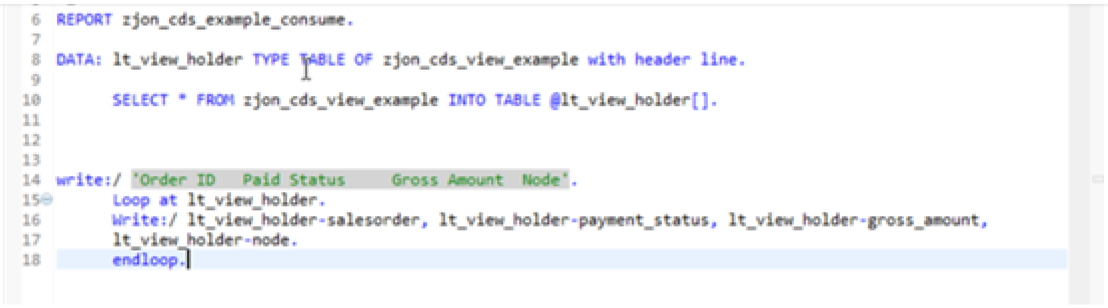
Node	SalesOrder	buyer_id	gross_amount	net_amount	tax_amount	payment_status
1244D0D392...	0500000000	1244D0D392B...	14385.85	12088.95	2296.90	Paid
1244D0D392...	0500000001	1244D0D392B...	15117.76	12704.00	2413.76	Paid
1244D0D392...	0500000002	1244D0D392B...	5631.08	4732.00	899.08	Paid
1244D0D392...	0500000003	1244D0D392B...	1704.04	1431.97	272.07	Paid
1244D0D392...	0500000004	1244D0D392B...	761.24	639.70	121.54	Paid
1244D0D392...	0500000005	1244D0D392B...	101299.22	85125.40	16173.82	Paid
1244D0D392...	0500000006	1244D0D392B...	250.73	210.70	40.03	Paid
1244D0D392...	0500000007	1244D0D392B...	9715.16	8164.00	1551.16	Paid
1244D0D392...	0500000008	1244D0D392B...	195.16	164.00	31.16	Paid
1244D0D392...	0500000009	1244D0D392B...	3972.22	3338.00	634.22	Paid
1244D0D392...	0500000010	1244D0D392B...	827.95	695.75	132.20	Not Paid
1244D0D392...	0500000011	1244D0D392B...	325.94	273.90	52.04	Not Paid
1244D0D392...	0500000012	1244D0D392B...	12704.40	10675.96	2028.44	Not Paid
1244D0D392...	0500000013	1244D0D392B...	8996.40	7560.00	1436.40	Not Paid
1244D0D392...	0500000014	1244D0D392B...	3459.33	2907.00	552.33	Not Paid
1244D0D392...	0500000015	1244D0D392B...	862.73	724.98	137.75	Not Paid
1244D0D392...	0500000016	1244D0D392B...	70.18	58.97	11.21	Not Paid
1244D0D392...	0500000017	1244D0D392B...	178.14	149.70	28.44	Not Paid
1244D0D392...	0500000018	1244D0D392B...	871.55	732.40	139.15	Not Paid
1244D0D392...	0500000019	1244D0D392B...	1444.64	1213.99	230.65	Not Paid
1244D0D392...	0500000020	1244D0D392B...	5357.97	4502.50	855.47	Not Paid
1244D0D392...	0500000021	1244D0D392B...	158.98	133.60	25.38	Not Paid
1244D0D392...	0500000022	1244D0D392B...	521.22	438.00	83.22	Not Paid
1244D0D392...	0500000023	1244D0D392B...	411.50	345.80	65.70	Not Paid

Notice that, along with the data we have selected from the snwd_so table, we also have the calculated field payment_status that was created using the CASE function.

USING CDS VIEWS IN ABAP PROGRAMS – WITHOUT PARAMETERS

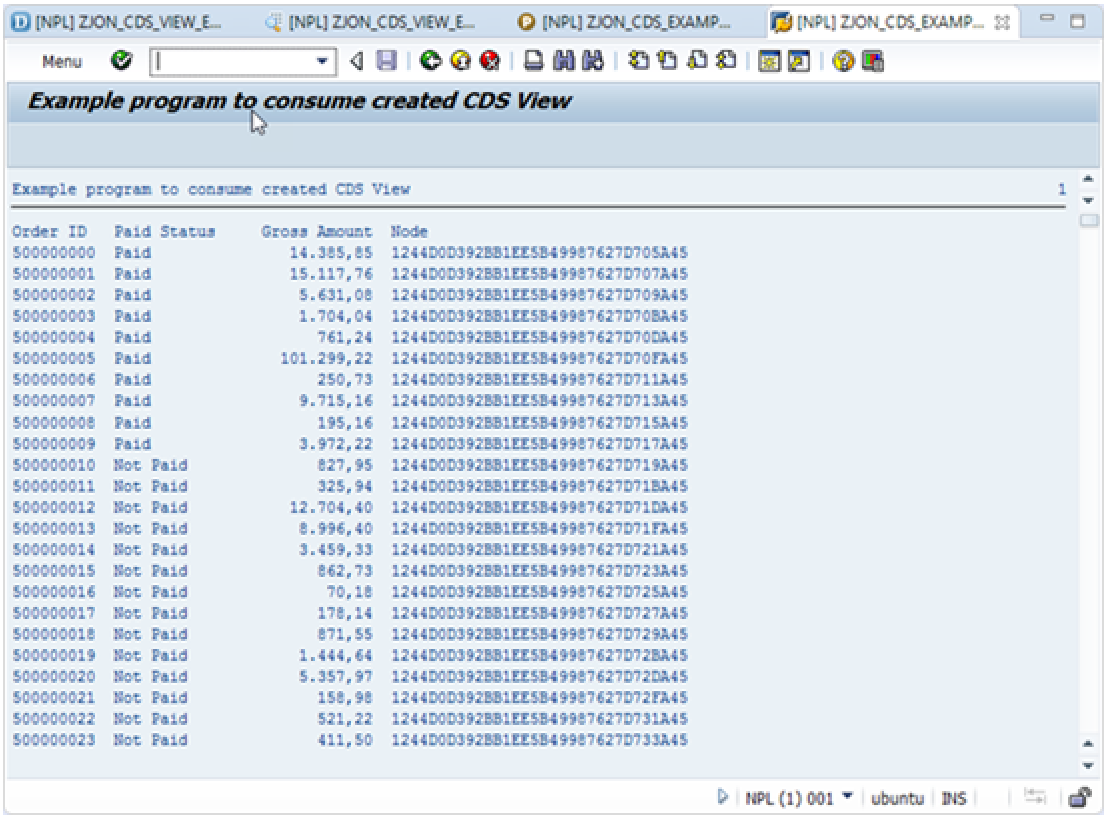
Next, we will look at an example of how to use the example CDS view within an ABAP Program. This simply requires using an Open SQL statement that selects from our CDS Entity (ZJON_CDS_VIEW_EXAMPLE). As an added benefit of ABAP CDS, a generated CDS Entity name can also be used in DATA declarations to create structures of a compatible TYPE.

(ZJON_CDS_EXAMPLE_CONSUME – Program)



```
REPORT zjon_cds_example_consume.

DATA: lt_view_holder TYPE TABLE OF zjon_cds_view_example with header line.

      SELECT * FROM zjon_cds_view_example INTO TABLE @lt_view_holder[].



write:/ 'Order ID   Paid Status     Gross Amount  Node'.
      Loop at lt_view_holder.
      Write:/ lt_view_holder-salesorder, lt_view_holder-payment_status, lt_view_holder-gross_amount,
      lt_view_holder-node.
      endloop.
```

The output of this program:



Example program to consume created CDS View

```
Example program to consume created CDS View                                        1

Order ID   Paid Status     Gross Amount  Node
500000000  Paid               14.385,85  1244D0D392BB1EE5B49987627D705A45
500000001  Paid               15.117,76  1244D0D392BB1EE5B49987627D707A45
500000002  Paid                5.631,08  1244D0D392BB1EE5B49987627D709A45
500000003  Paid                1.704,04  1244D0D392BB1EE5B49987627D70BA45
500000004  Paid                  761,24  1244D0D392BB1EE5B49987627D70DA45
500000005  Paid              101.299,22  1244D0D392BB1EE5B49987627D70FA45
500000006  Paid                  250,73  1244D0D392BB1EE5B49987627D711A45
500000007  Paid                9.715,16  1244D0D392BB1EE5B49987627D713A45
500000008  Paid                  195,16  1244D0D392BB1EE5B49987627D715A45
500000009  Paid                3.972,22  1244D0D392BB1EE5B49987627D717A45
500000010  Not Paid              827,95  1244D0D392BB1EE5B49987627D719A45
500000011  Not Paid              325,94  1244D0D392BB1EE5B49987627D71BA45
500000012  Not Paid           12.704,40  1244D0D392BB1EE5B49987627D71DA45
500000013  Not Paid            8.996,40  1244D0D392BB1EE5B49987627D71FA45
500000014  Not Paid            3.459,33  1244D0D392BB1EE5B49987627D721A45
500000015  Not Paid              862,73  1244D0D392BB1EE5B49987627D723A45
500000016  Not Paid               70,18  1244D0D392BB1EE5B49987627D725A45
500000017  Not Paid              178,14  1244D0D392BB1EE5B49987627D727A45
500000018  Not Paid              871,55  1244D0D392BB1EE5B49987627D729A45
500000019  Not Paid            1.444,64  1244D0D392BB1EE5B49987627D72BA45
500000020  Not Paid            5.357,97  1244D0D392BB1EE5B49987627D72DA45
500000021  Not Paid              158,98  1244D0D392BB1EE5B49987627D72FA45
500000022  Not Paid              521,22  1244D0D392BB1EE5B49987627D731A45
500000023  Not Paid              411,50  1244D0D392BB1EE5B49987627D733A45
```

EXTENDING CDS VIEWS

Another nice feature of CDS views is the ability to enhance them. This feature is beneficial when you want to augment an existing view, such as those that come with the standard SAP solution. We often need to modify SAP objects with custom Z fields, and view extension present a simple and transportable way to do this. To extend a view, follow the same as above when creating a basic view. However, when you arrive at the “New Data Definition” screen, select “Extend View” instead of “Define View”.

This template will allow you to select an existing CDS View to extend, and specify the new name for the view being created:

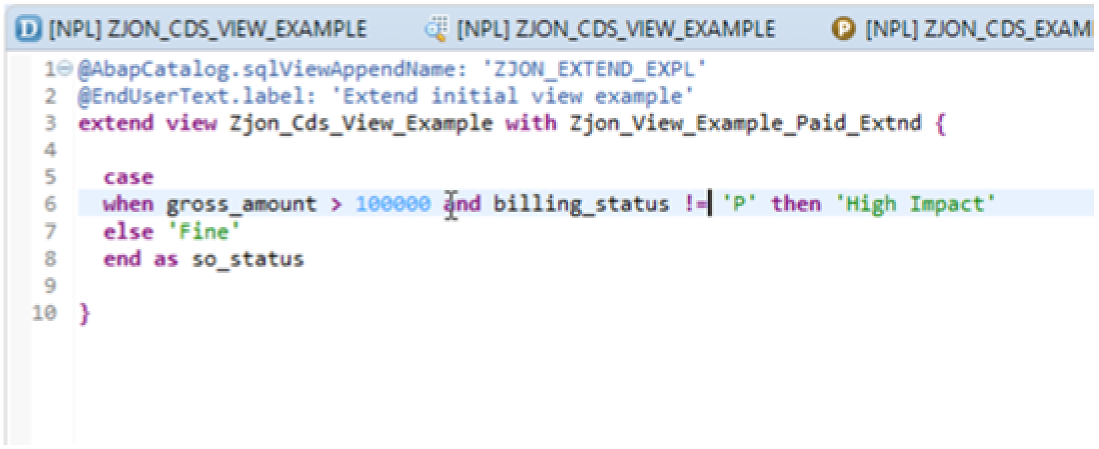
```
@AbapCatalog.sqlViewAppendName: 'sql_view_append_name'
@EndUserText.label: 'EXTENDO'
extend view view_name with Zextedn_Exmpl {
    data_source_name.element_name
}
```

In addition, view extension must include the following Annotation before an extend view definition:

@AbapCatalog.sqlViewAppendName: ‘${sql_view_append_name}’

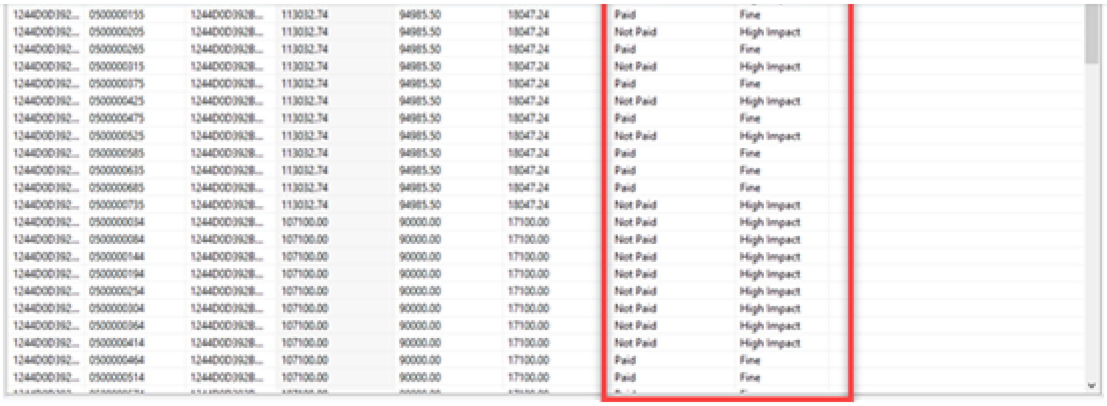
For this next example, we will extend the ZJON_CDS_VIEW_EXAMPLE with an additional field. To mimic a real-world example, let’s pretend we are modifying this view in order to achieve a particular business requirement.

Orders in this table are in US dollar amounts, to avoid needing to do currency conversions. To achieve this, the following view extension was created:



```
@AbapCatalog.sqlViewAppendName: 'ZJON_EXTEND_EXPL'
@EndUserText.label: 'Extend initial view example'
extend view Zjon_Cds_View_Example with Zjon_View_Example_Paid_Extnd {

  case
  when gross_amount > 100000 and billing_status != 'P' then 'High Impact'
  else 'Fine'
  end as so_status

}
```

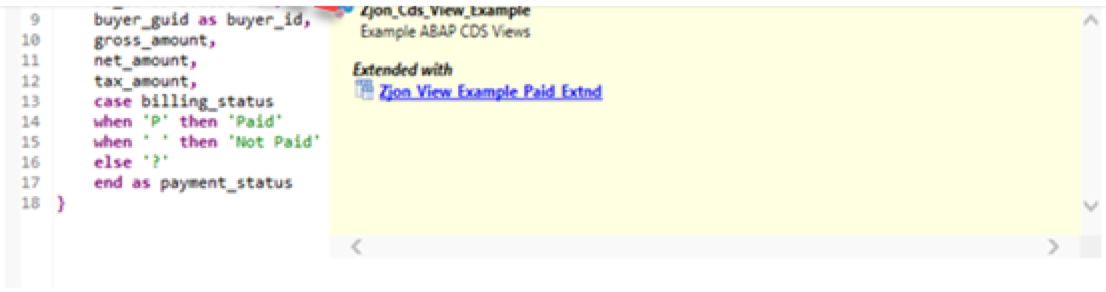
This view extension will append our new field to the view it is extending (ZJON_CDS_VIEW_EXAMPLE) and append the new field SO_STATUS. The new field SO_STATUS is calculated based on two qualifying criteria: when the gross_amount for the sales order is greater than $100000 and the billing status is not 'P'. This extended view displays exactly like the original view example, with the addition of the required "High Impact" indicator. Below is the output of this view, executed by running the "Data Preview" in Eclipse.



1244D0D392...	0500000155	1244D0D392B...	113032.74	94985.50	18047.24	Paid	Fine
1244D0D392...	0500000205	1244D0D392B...	113032.74	94985.50	18047.24	Not Paid	High Impact
1244D0D392...	0500000265	1244D0D392B...	113032.74	94985.50	18047.24	Paid	Fine
1244D0D392...	0500000315	1244D0D392B...	113032.74	94985.50	18047.24	Not Paid	High Impact
1244D0D392...	0500000375	1244D0D392B...	113032.74	94985.50	18047.24	Paid	Fine
1244D0D392...	0500000425	1244D0D392B...	113032.74	94985.50	18047.24	Not Paid	High Impact
1244D0D392...	0500000475	1244D0D392B...	113032.74	94985.50	18047.24	Paid	Fine
1244D0D392...	0500000525	1244D0D392B...	113032.74	94985.50	18047.24	Not Paid	High Impact
1244D0D392...	0500000585	1244D0D392B...	113032.74	94985.50	18047.24	Paid	Fine
1244D0D392...	0500000635	1244D0D392B...	113032.74	94985.50	18047.24	Paid	Fine
1244D0D392...	0500000685	1244D0D392B...	113032.74	94985.50	18047.24	Paid	Fine
1244D0D392...	0500000735	1244D0D392B...	113032.74	94985.50	18047.24	Not Paid	High Impact
1244D0D392...	0500000034	1244D0D392B...	107100.00	90000.00	17100.00	Not Paid	High Impact
1244D0D392...	0500000084	1244D0D392B...	107100.00	90000.00	17100.00	Not Paid	High Impact
1244D0D392...	0500000144	1244D0D392B...	107100.00	90000.00	17100.00	Not Paid	High Impact
1244D0D392...	0500000194	1244D0D392B...	107100.00	90000.00	17100.00	Not Paid	High Impact
1244D0D392...	0500000254	1244D0D392B...	107100.00	90000.00	17100.00	Not Paid	High Impact
1244D0D392...	0500000304	1244D0D392B...	107100.00	90000.00	17100.00	Not Paid	High Impact
1244D0D392...	0500000364	1244D0D392B...	107100.00	90000.00	17100.00	Not Paid	High Impact
1244D0D392...	0500000414	1244D0D392B...	107100.00	90000.00	17100.00	Not Paid	High Impact
1244D0D392...	0500000464	1244D0D392B...	107100.00	90000.00	17100.00	Paid	Fine
1244D0D392...	0500000514	1244D0D392B...	107100.00	90000.00	17100.00	Paid	Fine

NOTE THAT THIS DOES NOT CREATE A NEW VIEW, BUT MODIFIES THE ORIGINAL VIEW ZJON_CDS_VIEW EXAMPLE ITSELF! THIS IS INDICATED BY THE FAMILIAR ENHANCEMENT SYMBOL THAT WE NOW SEE IN THE ZJON_CDS_VIEW_EXAMPLE SOURCE FILE.

Once enhanced, our extending field will be available during all SELECTS against the ZJON_CDS_VIEW_EXMAPLE table.


```
    buyer_guid as buyer_id,
    gross_amount,
    net_amount,
    tax_amount,
    case billing_status
    when 'P' then 'Paid'
    when ' ' then 'Not Paid'
    else '?'
    end as payment_status
}
```

Zjon_Cds_View_Example
Example ABAP CDS Views

Extended with
Zjon View Example Paid Extnd

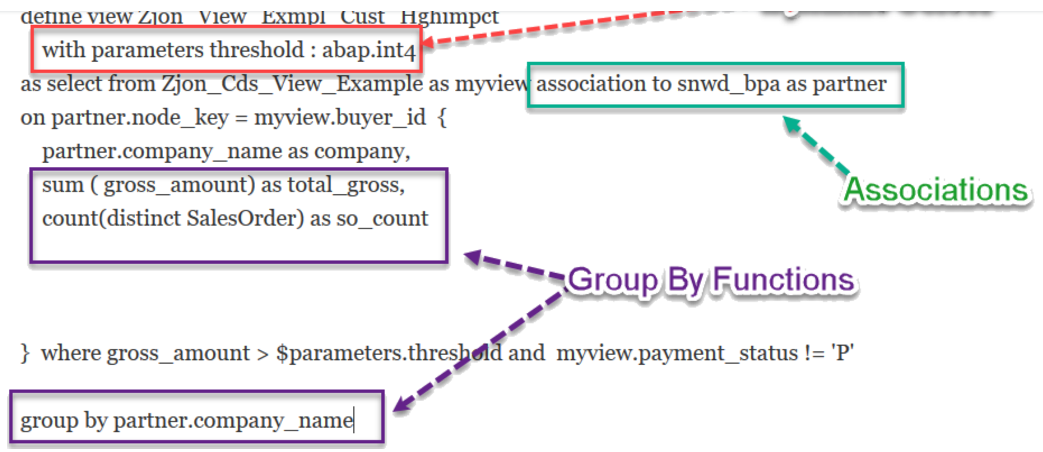
AGGREGATE FUNCTIONS, ASSOCIATIONS, AND PARAMETERS

As a last ABAP CDS example, let's explore a slightly more advanced scenario using a few aggregate functions, an association, and a parameter. We will explore each one of these features one at a time to illustrate the capabilities they provide.

To build on our real-world example, let's now imagine a scenario where we are trying to nail down not only the high impact sales orders, but also the companies responsible for them. Our end goal is to identify the companies with "High Impact" unpaid sales orders, the total count of these sales orders for each company, and the total gross amount of all "High Impact" sales orders for each company.

In addition, we will allow the definition of "High Impact" to be variable. While a sales order matching the criteria will remain an unpaid one, the threshold value that classifies it as "High Impact" won't be hard coded at $100,000. The threshold value will instead be left up to the user to determine at runtime.

The view used to accomplish this will again utilize our initial example ZJON_CDS_VIEW_EXAMPLE. It performs a SELECT from this view while utilizing an "Association" to map in the corresponding company from the snwp_bpa table with the matching buyer ID field.



```
define view Zjon_View_Exmpl_Cust_Hghimpct
  with parameters threshold : abap.int4
as select from Zjon_Cds_View_Example as myview association to snwd_bpa as partner
on partner.node_key = myview.buyer_id {
  partner.company_name as company,
  sum ( gross_amount) as total_gross,
  count(distinct SalesOrder) as so_count


} where gross_amount > $parameters.threshold and  myview.payment_status != 'P'

group by partner.company_name
```

USING ABAP CDS ASSOCIATIONS

Associations are essentially reusable JOINS that relate two CDS Entities (tables and views) to each other. CDS Associations have the added benefit of being able to specify cardinality ([1..1], [0..1], [*], etc.). Many of the features of ASSOCIATIONS are available as JOINS, however, ASSOCIATIONS are the preferred Best Practice and more elegant option when merging two CDS entities.

USING CDS AGGREGATE FUNCTIONS

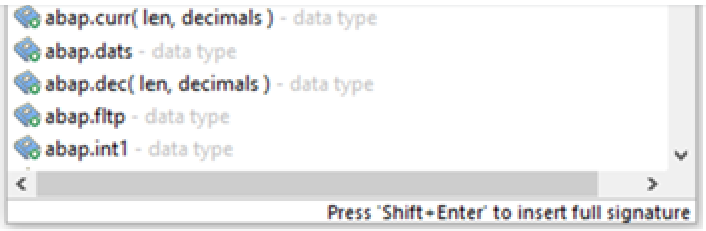
The aggregate functions available in CDS Views are the same ones that are available in the new Open SQL. Although they are readily available during a regular Open SQL Select (NO Core Data Services), it is still an extremely useful tool to have in CDS Views. Using the CDS approach, developers can make certain summary data is uniform across projects and systems without having to maintain entirely separate summary level tables.

FUNCTIONS IN THE FIELD DECLARATION SECTION, YOU ARE REQUIRED TO INCLUDE THE GROUP BY ADDITION IN THE SELECTION RESTRICTION AND CONDITION SECTION. REMOVING THE GROUP BY CLAUSE WHILE HAVING EITHER OF THESE FUNCTIONS REMAINING WILL RAISE A SYNTAX ERROR.

USING PARAMETERS IN CDS VIEWS

CDS Parameters is another very useful feature that allows the parameterization of CDS Views. In our specification example, we wanted to restrict the sales orders that were defined as “High Impact” to those sales orders that have a gross value of our choosing. By using a parameter THRESHOLD in our DDL source file, we are able to pass in this parameter at runtime and allow the results to be customized to that specific value.

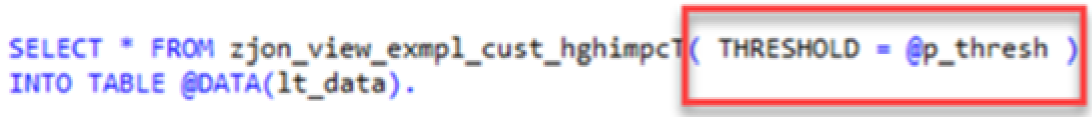
NOTE THAT THERE ARE NO OPTIONAL CDS PARAMETERS! IF A PARAMETER IS SPECIFIED WITHIN THE DDL SOURCE FILE, IT MUST BE PROVIDED OR A SYNTAX ERROR WILL OCCUR. ALSO, A DATA TYPE MUST BE SPECIFIED FOR THE PARAMETER IN USE. ABAP TYPES CAN BE USED FOR THIS PURPOSE, BUT TAKE NOTE OF THE SYNTAX <ABAP.DTYPE> WHEN USING THESE TYPES. YOU CAN ALSO USE DDIC DATA ELEMENTS SUCH AS SNWD_SO_ID, SNWD_CITY… ETC. OUR EXAMPLE USED THE ABAP.INIT4 DATA TYPE.


abap.curr(len, decimals) - data type
abap.dats - data type
abap.dec(len, decimals) - data type
abap.fltp - data type
abap.int1 - data type
Press 'Shift+Enter' to insert full signature

Now, we will bring this all together with a sample ABAP program that utilizes our new CDS View. This program will also illustrate how to call a CDS View with the parameter we have specified. take a look at the code below...

```
*&---------------------------------------------------------------------*
*& Report zjon_cds_example_para_assoc
*&---------------------------------------------------------------------*
*&
*&---------------------------------------------------------------------*
REPORT zjon_cds_example_para_assoc.

parameters p_thresh type int4.

SELECT * FROM zjon_view_exmpl_cust_hghimpcT( THRESHOLD = @p_thresh )
INTO TABLE @DATA(lt_data).

DATA: ls_data like line of lt_data.

Loop at lt_data into ls_data.
    write:/ ls_data-company, ls_data-so_count, ls_data-total_gross.
endloop.
```

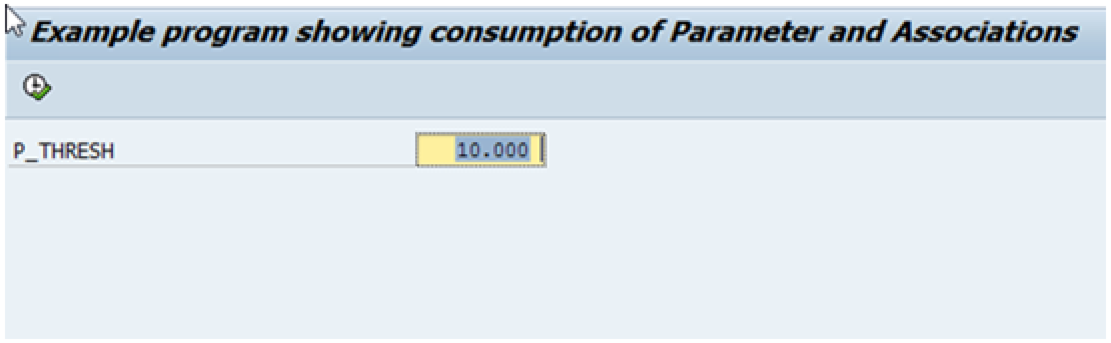
variable preceded by the '@' Escape symbol. This is similar to executing a method in ABAP OO with importing parameters.



```
SELECT * FROM zjon_view_exmpl_cust_hghimpcT( THRESHOLD = @p_thresh )
INTO TABLE @DATA(lt_data).
```

CHECKING YOUR WORK

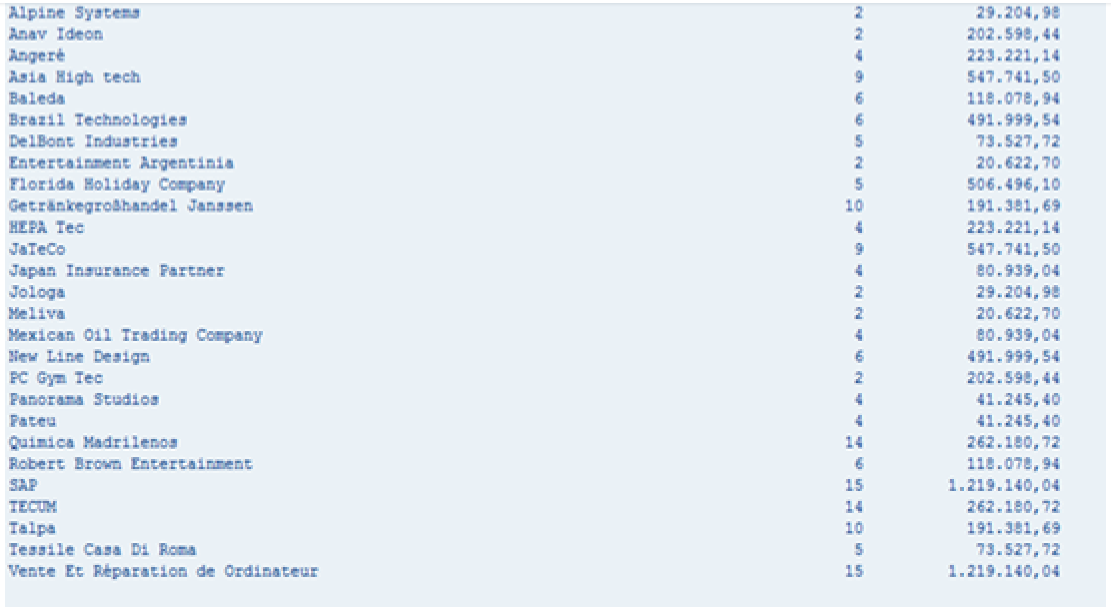
OK, let's run the program. We get the expected SELECTION screen. We will $10,000.00 as our threshold.



Example program showing consumption of Parameter and Associations

P_THRESH 10.000

Executing this report produces the summarized list of companies with unpaid sales orders above $10,000.



```
Alpine Systems                                   2        29.204,98
Anav Ideon                                       2       202.598,44
Angeré                                           4       223.221,14
Asia High tech                                   9       547.741,50
Baleda                                           6       118.078,94
Brazil Technologies                              6       491.999,54
DelBont Industries                               5        73.527,72
Entertainment Argentinia                         2        20.622,70
Florida Holiday Company                          5       506.496,10
Getränkegroßhandel Janssen                      10       191.381,69
HEPA Tec                                         4       223.221,14
JaTeCo                                           9       547.741,50
Japan Insurance Partner                          4        80.939,04
Jologa                                           2        29.204,98
Meliva                                           2        20.622,70
Mexican Oil Trading Company                      4        80.939,04
New Line Design                                  6       491.999,54
PC Gym Tec                                       2       202.598,44
Panorama Studios                                 4        41.245,40
Pateu                                            4        41.245,40
Quimica Madrilenos                              14       262.180,72
Robert Brown Entertainment                       6       118.078,94
SAP                                             15     1.219.140,04
TECUM                                           14       262.180,72
Talpa                                           10       191.381,69
Tessile Casa Di Roma                             5        73.527,72
Vente Et Réparation de Ordinateur               15     1.219.140,04
```

USING NATIVE HANA CDS

Generally speaking, ABAP developers should make very little use of HANA CDS when compared to ABAP CDS or Open SQL. ABAP CDS provides many of the features available in HANA CDS in a much simpler fashion, with the added benefit of being transportable. There are very few scenarios where HANA CDS would be a better option. Two scenarios where HANA CDS may have to be necessary used are:

You are using HANA as a sidecar/ secondary database.

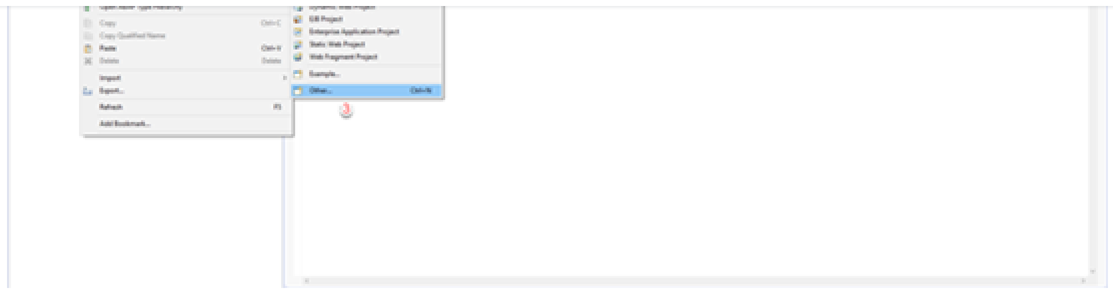
If you are using HANA as a secondary database, then ABAP CDS will simply not work. That is because ABAP CDS works with the data dictionary and assumes that all views are coming from the primary DB

AS ABAP on HANA becomes the standard, this scenario will become the more likely reason someone may opt to use HANA CDS vs ABAP CDS. While ABAP CDS and HANA CDS are being closely developed, there may be some branching in their functionalities. This is because ABAP CDS' chief purpose is to serve as an Open Source DDL for all databases, while HANA CDS is designed solely to model data on the HANA side.

Fortunately, if a developer does eventually need to venture into the HANA realm to utilize some feature not available on ABAP CDS, the syntax and structure of HANA CDS views are very similar to the ones in ABAP CDS.

CREATING AND CONSUMING HANA CDS VIEWS

Prerequisites: You have a SAP HANA System up and running. You are connected to the HANA system through Eclipse (with SAP Development Perspective). You have permissions to create views and run SQL on the SYSTEM schema. In addition, your SAP Netweaver AS system is connected to this HANA DB (as either a primary DB or a secondary DB).



Step 1: Open the SAP HANA Development Perspective

Step 2: Navigate to the Project Explorer tab (should be on the top left of the screen, unless you have rearranged your layout)

Step 3: Navigate to the SAP HANA Folder and hit the drop down. Select XS Project and hit next

Step 4: Enter the desired project name and hit next

Step 5: Select the desired workspace that utilizes your SAP HANA system. If a workspace doesn't exist that meets this criterion, hit "Add Workspace" and then select your HANA system on the next screen

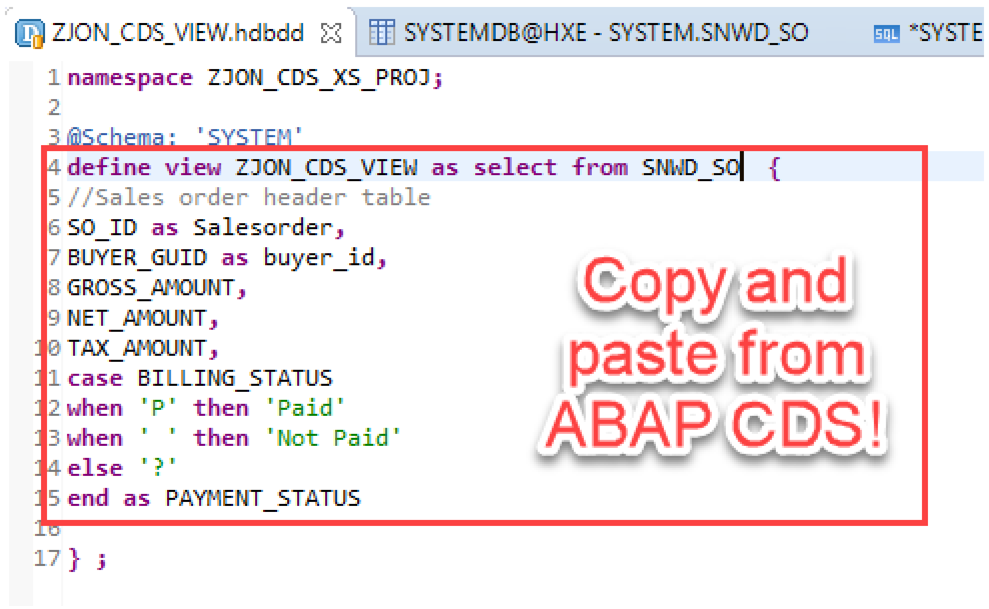
Step 6: For the purposes of this example, you can deselect both Access Objects on the following screen and hit Finish

Step 7: You should see your new project appear in the Project Explorer tab. Right-click this tab and select New->Other again.

Step 8: Navigate to SAP HANA->Database Development->DDL Source File

Step 9: Name your file, select your package, and hit Finish

For our example, we will recreate our original ZJON_CDS_VIEW_EXAMPLE CDS View, but this time directly on the HANA DB. As you will see below, the CDS syntax is so similar that we can literally copy and paste our original ABAP CDS source into the HANA CDS source after the annotations, add a semicolon, and we are finished.



```
namespace ZJON_CDS_XS_PROJ;

@Schema: 'SYSTEM'
define view ZJON_CDS_VIEW as select from SNWD_SO  {
//Sales order header table
SO_ID as Salesorder,
BUYER_GUID as buyer_id,
GROSS_AMOUNT,
NET_AMOUNT,
TAX_AMOUNT,
case BILLING_STATUS
when 'P' then 'Paid'
when ' ' then 'Not Paid'
else '?'
end as PAYMENT_STATUS

} ;
```

There are instances that would require many more changes to transfer an ABAP CDS to HANA CDS, however, it should be reassuring to developers that knowing ABAP CDS will make learning HANA CDS much easier.

The difficulty in using HANA CDS does not lie in the HANA CDS view creation itself, but within the ABAP Program source that needs to use some form of Native SQL to access it. As you will see below, Native

Some things to take note of:

– If you have a proper sidecar installation or proper data replication, you can just use the data dictionary type for that table. If not, you should create generic data types to hold the imported data (as I did with the “results’ type in the example)

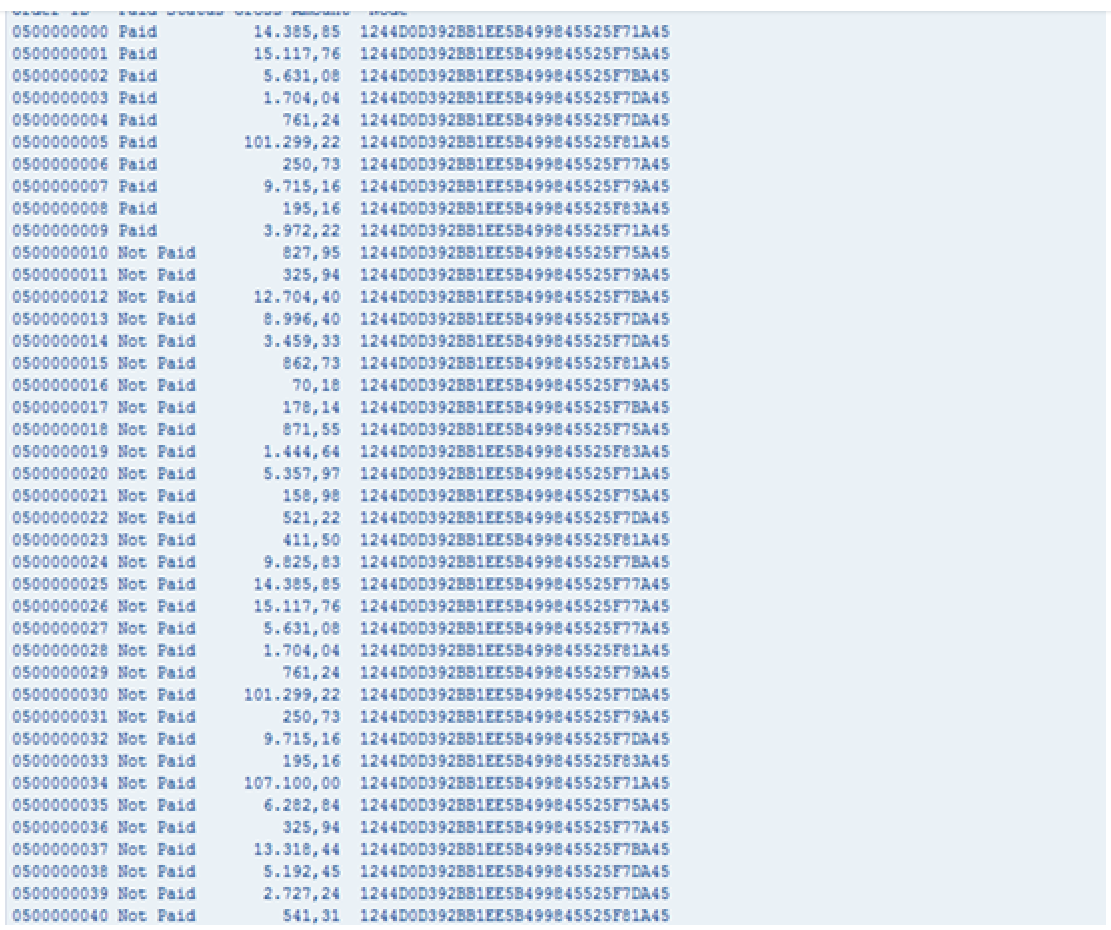
– The view location will be “<SCHEMA>”.”<PROJECT_FOLDER>::<VIEW_NAME>”

– This was a very simple SELECT scenario (made simpler by the fact we’re using CDS Views). Even a slightly more complicated SELECT would make this Native SQL SELECT much worse!

```
*&---------------------------------------------------------------------*
*& Report zjon_native_sql_exec
*&---------------------------------------------------------------------*
*&
*&---------------------------------------------------------------------*
REPORT zjon_native_sql_exec.

TYPES: BEGIN of results,
     salesorder(10) TYPE c,
     buyer_id(32) TYPE c,
     payment_status(9) type c,
     gross_amount TYPE dec10_2,
     net_amount TYPE dec10_2,
     tax_amount TYPE dec10_2,
     END of results.
```

```
        lv_sql      TYPE string,
        lr_data     TYPE REF TO data,
        lt_results TYPE TABLE OF results,
        ls_results TYPE results,
        lx_sql_exc TYPE REF TO cx_sql_exception.

  TRY.

      CONCATENATE 'SELECT "Salesorder", "buyer_id","PAYMENT_STATUS", "GROSS_AMOUNT", "NET_AMOUN
      'FROM "SYSTEM"."ZJON_CDS_XS_PROJ::ZJON_CDS"' INTO lv_sql.


      CREATE OBJECT lo_sql_stmt EXPORTING con_ref = cl_sql_connection=>get_connection('AB1').

      lo_result = lo_sql_stmt->execute_query( lv_sql ).

  *     Step 4: Read the result into the internal table lt_partner
      GET REFERENCE OF lt_results INTO lr_data.
      lo_result->set_param_table( lr_data ).
      lo_result->next_package( ).

  *     Step 5: close resources, i.e. the SQL statement and connection
      lo_result->close( ).

    CATCH cx_sql_exception INTO lx_sql_exc.
  write:/ 'Order ID   Paid Status Gross Amount  Node'.
      Loop at lt_results INTO ls_results.
      Write:/ ls_results-salesorder, ls_results-payment_status, ls_results-gross_amount,
      ls_results-buyer_id.
      endloop.
```

And the output (which is very similar to our first examples output):



```
0500000000 Paid          14.385,85  1244D0D392BB1EE5B499845525F71A45
0500000001 Paid          15.117,76  1244D0D392BB1EE5B499845525F75A45
0500000002 Paid           5.631,08  1244D0D392BB1EE5B499845525F7BA45
0500000003 Paid           1.704,04  1244D0D392BB1EE5B499845525F7DA45
0500000004 Paid             761,24  1244D0D392BB1EE5B499845525F7DA45
0500000005 Paid         101.299,22  1244D0D392BB1EE5B499845525F81A45
0500000006 Paid             250,73  1244D0D392BB1EE5B499845525F77A45
0500000007 Paid           9.715,16  1244D0D392BB1EE5B499845525F79A45
0500000008 Paid             195,16  1244D0D392BB1EE5B499845525F83A45
0500000009 Paid           3.972,22  1244D0D392BB1EE5B499845525F71A45
0500000010 Not Paid         827,95  1244D0D392BB1EE5B499845525F75A45
0500000011 Not Paid         325,94  1244D0D392BB1EE5B499845525F79A45
0500000012 Not Paid      12.704,40  1244D0D392BB1EE5B499845525F7BA45
0500000013 Not Paid       8.996,40  1244D0D392BB1EE5B499845525F7DA45
0500000014 Not Paid       3.459,33  1244D0D392BB1EE5B499845525F7DA45
0500000015 Not Paid         862,73  1244D0D392BB1EE5B499845525F81A45
0500000016 Not Paid          70,18  1244D0D392BB1EE5B499845525F79A45
0500000017 Not Paid         178,14  1244D0D392BB1EE5B499845525F7BA45
0500000018 Not Paid         871,55  1244D0D392BB1EE5B499845525F75A45
0500000019 Not Paid       1.444,64  1244D0D392BB1EE5B499845525F83A45
0500000020 Not Paid       5.357,97  1244D0D392BB1EE5B499845525F71A45
0500000021 Not Paid         158,98  1244D0D392BB1EE5B499845525F75A45
0500000022 Not Paid         521,22  1244D0D392BB1EE5B499845525F7DA45
0500000023 Not Paid         411,50  1244D0D392BB1EE5B499845525F81A45
0500000024 Not Paid       9.825,83  1244D0D392BB1EE5B499845525F7BA45
0500000025 Not Paid      14.385,85  1244D0D392BB1EE5B499845525F77A45
0500000026 Not Paid      15.117,76  1244D0D392BB1EE5B499845525F77A45
0500000027 Not Paid       5.631,08  1244D0D392BB1EE5B499845525F77A45
0500000028 Not Paid       1.704,04  1244D0D392BB1EE5B499845525F81A45
0500000029 Not Paid         761,24  1244D0D392BB1EE5B499845525F79A45
0500000030 Not Paid     101.299,22  1244D0D392BB1EE5B499845525F7DA45
0500000031 Not Paid         250,73  1244D0D392BB1EE5B499845525F79A45
0500000032 Not Paid       9.715,16  1244D0D392BB1EE5B499845525F7DA45
0500000033 Not Paid         195,16  1244D0D392BB1EE5B499845525F83A45
0500000034 Not Paid     107.100,00  1244D0D392BB1EE5B499845525F71A45
0500000035 Not Paid       6.282,84  1244D0D392BB1EE5B499845525F75A45
0500000036 Not Paid         325,94  1244D0D392BB1EE5B499845525F77A45
0500000037 Not Paid      13.318,44  1244D0D392BB1EE5B499845525F7BA45
0500000038 Not Paid       5.192,45  1244D0D392BB1EE5B499845525F7DA45
0500000039 Not Paid       2.727,24  1244D0D392BB1EE5B499845525F7DA45
0500000040 Not Paid         541,31  1244D0D392BB1EE5B499845525F81A45
```

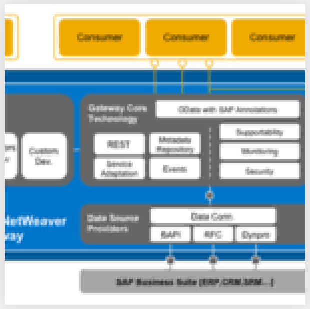
As you can see, HANA CDS requires much more effort to implement, then ABAP CDS. For this reason, HANA CDS should be avoided unless there is an absolute need to use it.

SUMMARY

developers are familiar with in OPEN SQL. The dual benefit of simpler and faster code makes the discipline well worth the effort.

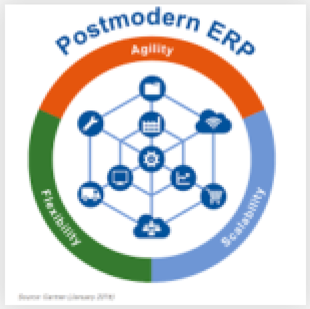
IF YOU ENJOYED THIS BLOG, DON'T TRY CODING ABAP CORE DATA SERVICES WITHOUT READING THIS FIRST, PLEASE FILL OUT THE FORM BELOW TO SIGN UP FOR OUR NEWSLETTER. WE DELIVER SAP TECHNICAL TIPS & TRICKS, SAP NEWS, AND THE CURRENT MONTH'S BLOG RIGHT TO YOUR INBOX!

RELATED POSTS



Understanding SAP NetWeaver Gateway

Understanding S/4HANA Cloud



Composable ERP: What It Is and Why It's Important – Part 1

IT PARTNERS, INC
EXPERIENCE MATTERS

5 Comments IT Partners, Inc Disqus' Privacy Policy 1 Login

Recommend Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

appsian • 8 months ago

Really enjoyed your article as its highly informative

• Reply • Share ›

Marc-Antoine • 3 years ago

Hello,

ITPARTNERS, INC
EXPERIENCE MATTERS

OUR PROFILE

Since 1993, IT Partners has been providing reliable, cost-effective solutions to meet our customer's goals and objectives in the Commercial and Federal ERP Marketplace.

Toll Free: (877) 288-6044
Local:(571) 485-2451
FAX: (571) 526-5593
Email: info@itpfed.com

OUR CERTIFICATIONS

Certified WBENC Women's Business Enterprise

CERTIFIED ISO 9001:2015 COMPANY

NEWS

IT Partners Announces Achievement
October 8, 2020

IT Partners is Officially Certified as a W
June 15, 2020

NEW

Sign

First Nam

Last Nam

Email Add

SIGN

OUR MEMBERSHIPS

IT PARTNERS, INC
EXPERIENCE MATTERS