**ABAP for Hana**

[Содержание](#Содержание)

[Передача параметров по ссылке/значению](#Передача_параметров_по_ссылке_значению)

[Массив](#Массив)

[NO-ZERO | SKIP | ULINE commands](#NO_ZERO_SKIP_ULINE)

## [Messages](#Messages)

# [Data Types](#DATA_TYPES)

[Variables](#VARIABLES)

[System Variables](#System_Variables)

# [Operators](#Operators)

# [IS INITIAL](#IS_INITIAL)

## [Bitwise Operators](#Bitwise_Operators)

## [Character String Operators](#Character_String_Operators)

[DATA - Inline Declaration](#DATA_Inline_Declaration)

[REDUCE](#REDUCE)

[VALUE Operator](#VALUE_Operator)

# [Loop Control](#Loop_Control) DO | WHILE

# [Decisions](#Decisions) IF | CASE -> COND

# [COND instead of IF - ELSE](#COND_instead_of_IF_ELSE)

[SWITCH instead of CASE](#Using_SWITCH_statement)

# [Strings](#Strings)

[New String Features](#New_String_Features)

# [Date & Time](#Date_Time)

# [ABAP Selection Screen](#ABAP_Selection_Screen)

[Database Access](#Database_Access)

# AMDP

[AMDP – основы](#AMDP_основы)

# [*AMDP Функции*](#AMDP_Функции)

# [AMDP with SELECT OPTIONS](#AMDP_with_SELECT_OPTIONS)

# [Accessing tables from different schema through AMDP](#Accessing_tables_from_different_schema)

# [Create an AMDP and Analyze Its Performance](#Create_an_AMDP)

# Examples

# [Чтение данных из CV в ABAP](#Чтение_данных_из_CV_в_ABAP)

[*Create the filter using Expression*](#Create_the_filter_using_Expression)

# [*ABAP AMDP Procedure Output Parameter Type must be Structured*](#AMDP_Procedure_Output_Parameter_Type)

# [Create Dates Table CDS Table Function using SAP HANA AMDP Class](#Create_Dates_Table)

## [AMDP Method Call from Program and Calling an AMDP method from another AMDP method](#AMDP_Method_Call)

# [For All Entries Alternatives](#For_All_Entries_Alternatives)

[Using Internal table as Data Source From Release 7.52](#Using_Internal_table_as_Data_Source)

[Pushing Internal table to DB by using AMDP](#Pushing_Internal_table_to_DB_by_AMDP)

# [Internal Tables](#Internal_Tables)

# [Classes](#Classes)

# [Object Events](#OBJECT_EVENTS)

# [Report Programming](#REPORT_PROGRAMMING)

# [CL\_DEMO\_OUTPUT](#CL_DEMO_OUTPUT)

# [TRY and RETRY, CATCH and RESUME](#TRY_CATCH)

[Events](#Events)

# [Dialog Programming](#Dialog_Programming)

# [Параметры в ABAP программе](#Параметры_в_ABAP_программе)

# [Вызов программы из другой с передачей параметров](#Вызов_программы_с_передачей_параметров)

[INTEGRATING AN AMDP INTO YOUR ABAP CODE](#INTEGRATING_AN_AMDP_INTO_ABAP)

[CDS View](#CDS_View)

# [Dynamic Programming in ABAP](#Dynamic_Programming_in_ABAP)

# [ABAP on SAP HANA. Part XI. Are Native SQL and Open SQL Competitors?](#Native_SQL_and_Open_SQL)

# [ODATA](#ODATA)

# [BAdI](#BAdI)

# [BAdI Implementation in ABAP on HANA for BW Extractors](#BAdI_for_BW_Extractor)

# [Using ABAP with Process Chain Decision Processes](#ABAP_with_Process_Chain_Decision_Process)

# [CDS based data extraction](#CDS_based_data_extraction)

# [Overview](#CDS_bde_Overview)

# [Delta Handling](#CDS_bde_Delta_Handling)

# CDS based data extraction

# ABAP on SAP HANA. Part X. AMDP with SELECT OPTIONS

<https://sapyard.com/abap-on-sap-hana-part-x-amdp-with-select-options/>

**Концепция блокировок в SAP**

<https://abap-blog.ru/database-work/koncepciya-blokirovok-v-sap/>

# *from*

# ABAP on SAP HANA. Part I. First Program in ABAP HANA

<https://sapyard.com/abap-on-sap-hana-part-i/>

+

# ABAP on SAP HANA. Part II. ADT Eclipse and HANA Studio

<https://sapyard.com/abap-on-sap-hana-part-ii/>

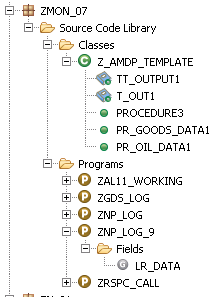
*\*& Report ZNP\_LOG\_9*

REPORT **ZNP\_LOG\_9**.

DATA(lr\_data) = new Z\_AMDP\_TEMPLATE( ).

CALL METHOD lr\_data->PROCEDURE3.





**Передача параметров по ссылке/значению**

[Содержание](#Содержание)

*The variable, passed by reference* can’t be changed inside the method - we only can change the value of variable which is passed by value.

Пример

REPORT **zsr\_test** NO STANDARD PAGE HEADING.

CLASS **cl\_pass\_para** *DEFINITION*.

PUBLIC SECTION.

METHODS ***m\_pass\_para*** IMPORTING *v\_ref* TYPE i *“ passed by reference*

*value*(v\_val) TYPE i. *” passed by value*

ENDCLASS.

CLASS ***cl\_pass\_para*** *IMPLEMENTATION*.

METHOD ***m\_pass\_para***.

WRITE: / 'Parameter passed by reference', *v\_ref*,

/ 'Parameter passed by value', *v\_val*.

v\_val = 10.

WRITE: / 'Parameter passed by reference can''t be changed', v\_ref,  
 / 'Parameter passed by value can be changed', v\_val.

ENDMETHOD.

ENDCLASS.

START-OF-SELECTION.

DATA *obj* TYPE REF TO *cl\_pass\_para*.

CREATE OBJECT obj.

CALL METHOD obj->*m\_pass\_para* *EXPORTING v\_ref* = 5 *v\_val* = 7.

**Массив**

[Содержание](#Содержание)

SELECT *mandt vbeln posnr* FROM vbap INTO TABLE *itab* WHERE erdat in s\_erdat.

IF *itab[]* IS NOT INITIAL.  
   select *mandt vbeln posnr* from vbap into table *itab* for ALL ENTRIES IN *itab*

   where vbeln = itab-vbeln and posnr = itab-posnr.

ENDIF.

**NANA Studio – ABAP Development Tool - ADT**

**ADT - *ABAP Development Tool*.**

ADT does not come by default - it has to be installed as a plugin on Eclipse / in Studio -> *Help -> Add New Softwar*e/.

It is easy to create a testing Environment with a hosted [Citrix XenDesktop](https://www.clouddesktoponline.com/citrix-xen-desktop/) from CloudDesktopOnline.com. You can also go for Integrated SharePoint with SAP HANA with the help of SharePoint experts from [Apps4Rent](http://www.apps4rent.com/).

Отличия от se80

Some advanced features like creating 1) external views for exposing HANA view to ABAP DDIC\* & 2) database proxy procedures\* *-* are available only when using ADT.

Can we directly write and execute ABAP program in HANA studio?

− *No* it has to be connected to an ABAP system first - *ABAP project*helps to connect the Eclipse base IDE to ABAP backend system.

**How do we view ABAP programs in HANA Studio?**

− Expand the System Library and go to your custom package and program



**Can we edit the same program in GUI and in HANA Studio simultaneously?**

**− *No*** - both ADT and SE80 have same source code repository and locking mechanism *⇒* we cannot interfere when other is editing it at the same time.

Do not forget - the *Z\** or *Y\** naming convention *holds good* /хорошо действует/ even while creating custom objects from ADT.

**Полезные сочетания клавиш**

* ***Ctrl + 1*** - open the *Quick Fix menu*.
* *Ctrl + space* – *автозавершение кода | список полей* – если курсор находится в выбранных полях данного data source.

Type initial letters of the syntax you want to use and then use ***Ctrl + Space*** and ***Shift + Enter*** to insert the full signature - e.g. for function module / method selected.

* *F2*- *список полей* – если курсор находится на данном data source.

**Pretty Printer**

Go to ***Windows -> Preference -> ABAP Development -> Source Code Editor -> Formatter* *to set up the formatting*** needs.

**Revision History**

We can compare changes from one transport of source code to another in ADT - right click on the source code area of the program and choose ***Compare with -> Revision History***.

# SAP ABAP Tutorial

<https://www.tutorialspoint.com/sap_abap/index.htm>

**ABAP 4** - *Advanced Business Application Programming* - a 4GL - 4th generation language.

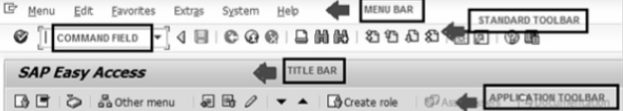
Unlike C++ and Java *ABAP programs* are not stored in separate external files. Inside the database ABAP code exists in two forms – 1) *source code*; 2) *generated code* which is a binary representation - it’s somewhat comparable with Java byte code.

A key component of the ABAP run-time system is *the database interface* that turns database independent statements - Open SQL into the statements understood by the underlying database - Native SQL.

***AS*** is an *application server* with its own – 1) database; 2) run-time environment; 3) development tools - *ABAP Editor –* ***se38***. The AS offers a development platform that is independent of hardware | operating system | database.

The***customer namespace*** includes all objects with the prefix *Y* or *Z*.

The ***names of ABAP objects*** are *not case sensitive*.



***Title Bar*** − displays *the name of the application/business process* you are currently in.

***Application Toolbar*** − *application specific menu options*.

The ***Report***will always be the *first line* of any executable program.

*The ABAP editor* converts *all text to uppercase* except text *strings* which are surrounded by single quotation marks.

## Comments

* *Full line comments* - *\** in the first position of the line.
* *Partial line comments* - *"* after a statement.

**NO-ZERO | SKIP | ULINE commands**

[Содержание](#Содержание)

***NO-ZERO*** command *suppresses all leading zeros of a number field* containing blanks.

DATA:

W\_NUR(10) TYPE N.

MOVE 50 TO W\_NUR.

WRITE W\_NUR *NO-ZERO*. *" 50*

*Rem*

Without NO-ZERO command the output would be 0000000050.

The ***SKIP*** command helps in *inserting blank lines on the page*. We may use the SKIP command to insert multiple blank lines

*SKIP* TO LINEline\_number.

The ***ULINE*** command automatically *inserts a horizontal line* across /ə'krɔs через – от края до края/ the output

WRITE 'This is Underlined'.

*ULINE*.



## **Messages**

[Содержание](#Содержание)

The ***messages*** are numbered *from 000 to 999*. Associated with each *number* is a message *text* up to a maximum of 80 characters - when message number is called the corresponding text is displayed.

|  |  |  |
| --- | --- | --- |
| **Message** | **Type** | **Consequences** |
| **E** | Error | The message appears and the application halts at its current point. If the program is running *in background mode* - the job is canceled and the message is recorded in the job log. |
| **W\*** | Warning | The message appears and the user must press *Enter* for the application to continue. |
| **I\*** | Information | A pop-up window opens with the message text and the user must press *Enter* to continue. |
| **A** | Abend | Сокр. от *ABnormal END*. This message class *cancels the transaction* that the user is currently using. |
| **S** | Success | This provides an *informational message* at the bottom of the screen. |
| **X** | Abort | This message *aborts the program* and generates an ABAP short dump. |

Примечание

\* - *in background mode* - the message is recorded in the job log.

### Example

When we create a message for message the id *AB* - the MESSAGE command MESSAGE *E011* - gives the following output –

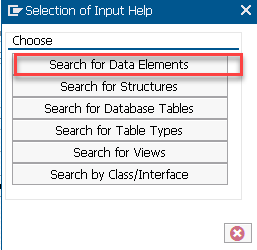
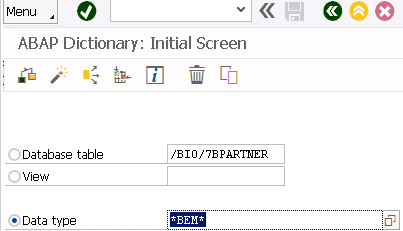
*EAB011* This report does not support sub-number summarization.

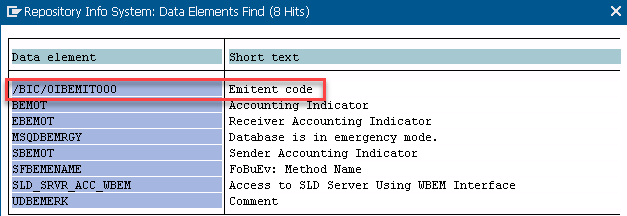
# Data Types

[Содержание](#Содержание)

Поиск типов данных SAP BW в ABAP Dictionary

Тр. ***se11***





## **Elementary Data Types**

|  |  |  |
| --- | --- | --- |
| **Type** | **Typical Length** | **Typical Range** |
| X - byte field | 1 byte | Any byte values – 00..FF |
| C | 1 character | 1..65535 |
| N - numeric text filed | 1 character | 1..65535 |
| D - character-like date | 8 characters | 8 characters |
| T - character-like time | 6 characters | 6 characters |
| I | 4 bytes | -2147483648..2147483647 |
| F | 8 bytes | 2.2250738585072014E-308..1.7976931348623157E+308 positive or negative |
| P - packed number | 8 bytes | [-10^(2len -1) +1] to [+10^(2len -1) 1] - where len = fixed length |
| STRING | Variable | Any alphanumeric characters |
| XSTRING - byte string | Variable | Any byte values – 00..FF |

*Чем P отличается от F?*

Type ***STRING*** - should be used where fixed length is not important.

DATA:

text\_line TYPE *C* LENGTH 40 value 'A Chapter on Data Types',

text\_string TYPE *STRING* value 'A Program in ABAP'.

d\_date TYPE *D* value SY-DATUM.

## **Complex and Reference Types**

In ABAP *arrays* are called internal tables –

1. *Row* of an internal table can be of *elementary* | *complex* | *reference type*.
2. A *key* contains the fields of *elementary* types.
3. *Access method* describes how ABAP programs access individual table entries.

*The ABAP OOP run-time type services* - **RTTS** enables declaration of data items at run-time.

**Ommiting Data Type Declarations */from abap 7.4/***

The ABAP Compiler knows what data type it should be - instead of declaring it yourself

*data*(lv\_string) = 'ABAP Gurus SAP ABAP Tutorial'.

**Avoiding Type Mismatch Dump when Calling Function Module**

Earlier *the dump error occured* when we declare *different type* of *variable* than parameter of function module.

Now *parameters* at the instanthavetheir *values* filled by the method.

CALL FUNCTION 'ZMY\_FM'

EXPORTING ID = LV\_MYID

IMPORTING NAMEZ = *data*( LV\_NAMEZ ).

**Variables**

[Содержание](#Содержание)

The basic form of a variable declaration is –

*DATA* <f> *TYPE* <type> *VALUE* <val>.

Other possible values for *<val>* can be a *literal* | *constant* | an *explicit clause* /such as *Is INITIAL/*.

The ***name of the variable*** can be up to 30 characters long.

***Hyphens* /**'haɪf(ə)n дефис/ *are reserved* to represent the components of structures ⇒ you are supposed to avoid hyphens in variable names

DATA d1(2) TYPE C.

DATA d2 *LIKE* d1.

DATA minimum\_value TYPE I VALUE 10.

## ***Static Variables***

* With *CLASS-DATA* statement you can declare variables within the classes.
* The *PARAMETERS*statement can be used to declare the *elementary data objects* that *are linked to input fields on a selection screen*.
* You can also declare the *internal tables* that *are linked to input fields on a selection screen* by using *SELECT-OPTIONS* statement.

### Example

REPORT ZTest123\_01.

*PARAMETERS*:

NAME(10) TYPE C,

CLASS TYPE I,

SCORE TYPE P DECIMALS 2,

CONNECT TYPE MARA-MATNR.

## ***Reference Variables***

The ***REF TO*** addition is used to create a *bound reference type* and as a start value and only *IS INITIAL* can be specified after the VALUE addition.

DATA <ref> TYPE *REF TO* <type> VALUE *IS INITIAL*.

### Example

CLASS **с1** DEFINITION.

PUBLIC SECTION.

DATA bi TYPE i VALUE 1.

ENDCLASS.

DATA:

o\_ref TYPE *REF TO* c1 ,

d\_ref1 LIKE *REF TO* o\_ref, *“ d\_ref1 - ссылка на ссылку o\_ref*

d\_ref2 TYPE *REF TO* I.

CREATE OBJECT o\_ref.

*“ o\_ref* → *c1*

*GET REFERENCE* OF o\_ref INTO d\_ref1.

*“ d\_ref1 → o\_ref* → *c1*

CREATE DATA d\_ref2.

*“ d\_ref2 → integer*

*" Both data reference variables – d\_ref1 | d\_ref2 are fully typed and can be dereferenced using the*

*" dereferencing operator →\***at operand positions.*

*d\_ref2→\** = *d\_ref1→\*→[\*]*bi.

**System Variables**

[Содержание](#Содержание)

* You can find the complete *list of system variables* in the **SYST** table.
* *Individual fields of the* ***SYST*** structure can be accessed by using either **SYST-** or **SY-**

### Example

REPORT Z\_Test123\_01.

WRITE:

/ SY-ABCDE, *" ABCDEFGHIJKLMNOPQRSTUVWXYZ*

/ SY-DATUM, *" 24.08.2020*

/ SY-DBSYS, *" HDB*

/ SY-HOST, *" bl-sql-sbw*

/ SY-LANGU, *" EN*

/ SY-MANDT, *" 100*

/ SY-OPSYS, *" Linux*

/ SY-SAPRL, *" 750*

/ SY-SYSID, *" SBW*

/ SY-TCODE, *" SADT\_START\_WB\_UI*

/ SY-UNAME, *" DEVELOPER5*

/ SY-UZEIT*. " 14:25:48*

**NEW keyword for Creating Objects**

*Before ABAP 7.4*

DATA : lo\_myclass TYPE REF TO ZCL\_MYCLASS.

*CREATE OBJECT* lo\_myclass EXPORTING myname = 'ABAPGurus'.

*Now*

…

lo\_myclass = *new* zcl\_Myclass( myname = 'ABAPGurus' ).

# Constants & Literals

# *Trailing blanks in*

# the *text field* literals - are *ignored*;

# the *string* literals - are taken into account.

*CONSTANTS* <f> TYPE <type> VALUE <val>.

*Rem*

We should use the *VALUE* clause in the CONSTANTS statement.

***We have 3 types of constants***

1. *elementary*;
2. *complex*

BEGIN OF *EMPLOYEE*,

Name(25) TYPE C VALUE 'Management Team',

Organization(40) TYPE C VALUE 'Tutorials Point Ltd',

Place(10) TYPE C VALUE 'India',

END OF EMPLOYEE.

1. *reference* constants –

CONSTANTS null\_pointer TYPE *REF TO* object VALUE *IS INITIAL*.

# Operators

[Содержание](#Содержание)

**IS INITIAL** - the condition becomes *true* if the contents of the variable haven’t changed *and* it has been automatically assigned its initial value.

DATA: A TYPE I.

IF A *IS INITIAL*.

WRITE: / 'A is assigned'.

ENDIF.

## **Bitwise Operators**

[Содержание](#Содержание)

|  |
| --- |
| **BIT-NOT** - *flips* /flɪp транспонировать/ all the bits in a hexadecimal number to the opposite value. |
| **BIT-AND** - compares each field bit by bit using the boolean AND operator. |
| **BIT-XOR** - compares each field bit by bit using the boolean XOR - exclusive OR operator. |
| **BIT-OR** - compares each field bit by bit using the boolean OR operator. |

## **Character String Operators**

[Содержание](#Содержание)

**&&** - concatenation operator - str = str && … .

**|**…**|** - character string templates - str = |{ str }…|.

### **Bad News**

Both assignments have a **string expression** as an **RHS** *- right hand side* - a temporary/intermediate result is created that must be copied to the **LHS** - *left hand side*.

new\_str = str\_1 *&&* str\_2

1. temp\_obj = str\_1 *&&* str\_2;
2. new\_str = temp\_obj

Since the *length* of the *intermediate result* increases with each loop pass and there is a copy operation for each loop pass ⇒ *the runtime dependency* from the number of loop passes is ***quadratic***.

### **Good News**

As long as the target string *str* occurs at the RHS only *once* and as the *leftmost* operator and there are 1) *no formatting options* for str; 2) *no other expressions or function calls* involved in the RHS - ***no intermediate result*** *is created*but the characters are **concatenated directly to the target string** str.

The same is true for the CONCATENATE statement - with other words ***no problems*** in writing something like this

DATA(*html*) = `<html><body><table border=1>`.

DO … TIMES.

*html* = |{ *html* }<tr><td>{ sy-index }</td></tr>|. *“html occurs at the RHS only once*

ENDDO.

*html* = *html* && `</table></body></html>`.

There is in fact a problem writing something looking as *harmless* like this

DATA(*html*) = `<html><body><table border=1>`.

DO … TIMES.

*\* Concatenating the ipow expression to str breaks the optimization.*

*\* ipow( base = arg exp = n ) - raises the argument arg to the exponent n.*

*html* = |{ *html* }<tr><td>{ CONV *string*( ipow( base = sy-index exp = 2 ) ) }</td></tr>|.

ENDDO.

html = html && `</table></body></html>`.

You can easily circumvent /ˌsɜːkəm'vent обойти/ it with *helper variables*

DATA(*html*) = `<html><body><table border=1>`.

DATA *square* type string.

DO ... TIMES.

*\* By assigning the ipow expression to helper variable square and concatenating that to str  the optimization*

*\* takes place again.*

*square* = ipow( base = sy–index exp = 2 ).

*html* = |{ *html* }<tr><td>{ *square* }</td></tr>|.

ENDDO.

html = html && `</table></body></html>`.

For loops realized by control statements is also true for *FOR loops* in expressions

DATA(result) =

  REDUCE string( INIT s = “ FOR i = 1 UNTIL i > …

                  NEXT s = s && CONV string( i ) ).

vs.

DATA(result) =

  REDUCE string( INIT s = “ FOR i = 1 UNTIL i > …

*“ Only by using the helper variable num the optimization is enabled.*

                 LET *num* = CONV string( i ) IN

NEXT s = s && *num* ).

|  |
| --- |
| **CO** *- Contains Only -*checks whether A is solely composed of the characters in B. |
| **CN** *- Not Contains ONLY -*checks whether A contains characters that are not in B. |
| **CA** *- Contains ANY -*checks whether A contains at least one character of B. |
| **NA** *- NOT Contains Any* |
| **CS** *- Contains a String -*checks whether A contains the character string B. |
| **NS** *- NOT Contains a String* |
| **CP** *- Contains a Pattern -*checks whether A contains the pattern in B. |
| **NP** *- NOT Contains a Pattern* |

DATA:

P(10) TYPE C VALUE 'APPLE',

Q(10) TYPE C VALUE 'CHAIR'.

IF P *CA* Q. *“Contains ANY*

WRITE: / 'P contains at least one character of Q'.

ENDIF.

Работа со строками

<http://razumsap.blogspot.com/2014/10/blog-post.html>

**DATA - Inline Declaration**

[Содержание](#Содержание)

**DATA(var)**

* The declared variable is visible *statically* in the program and is valid in the current [context](javascript:call_link('abencontext_2_glosry.htm')).
* The declaration is made when the program *is compiled*.
* The *date type* of the variable is determined by the [operand type](javascript:call_link('abenoperand_type_glosry.htm')).

***Notes***

* [Only use inline declarations *locally*](javascript:call_link('abendeclaration_inline_guidl.htm')).
* An inline declaration cannot occur *on the right-hand side* of an assignment or *within* an expression.

Example

TYPES *t\_itab* TYPE TABLE OF i WITH NON-UNIQUE KEY table\_line.

*DATA*(itab) = VALUE *t\_itab*( ( 1 ) ( 2 ) ( 3 ) ).

LOOP AT itab INTO DATA(wa).

  ...

ENDLOOP.

**REDUCE - Reduction Operator**

[Содержание](#Содержание)

Reduce operator creates a result of specified data type after going through iterations.

TYPES : BEGIN OF *lty\_alphabet*,

letter TYPE c,

END OF lty\_alphabet,

*ltty\_alphabet* TYPE STANDARD TABLE OF lty\_alphabet WITH EMPTY KEY.

DATA(*lt\_alpha*) = VALUE ltty\_alphabet( ( letter = ’A’ ) ( letter = ’B‘ ) ( letter = ’C‘ ) ( letter = ’D' ) ).

DATA(*lv\_letters*) = REDUCE string( INIT lv\_y = ’ ’

FOR *wa\_alpha* IN lt\_alpha

NEXT lv\_y = |{ *lv\_y* }| & |{ *wa\_alpha-letter* }| ).

*Результат*

*lv\_letters = ‘ABCD’*

TYPES :

BEGIN OF *lty\_value*,

field1 TYPE i,

field2 TYPE i,

END OF lty\_value,

*ltty\_value* TYPE STANDARD TABLE OF lty\_value WITH EMPTY KEY,

*ltty\_field2* TYPE STANDARD TABLE OF i WITH EMPTY KEY.

DATA(*lt\_values*) = VALUE ltty\_value( ( field1 = 1 field2 = 2 ) ( field1 = 3 field2 = 4 ) ( field1 = 5 field2 = 6 )

( fieldl = 5 field2 = 7 ) ).

DATA(*lv\_count*) = REDUCE i( INIT lv\_x = 0 FOR wa\_value IN lt\_values WHERE ( field1 = 5 )

NEXT lv\_x = lv\_x + 1 ).

In this example, we will see, how to use reduce operator, instead of loop inside loop for doing the sum of a field.

REPORT ZKT\_REDUCE\_OPERATOR.

data : *gv\_kunnr* type bsid-kunnr.

SELECTION-SCREEN begin of BLOCK b1 WITH FRAME TITLE text-001.

 select-OPTIONS : *s\_kunnr* for gv\_kunnr OBLIGATORY.

SELECTION-SCREEN end of BLOCK b1.

START-OF-SELECTION.

     perform get\_data.

end-of-SELECTION.

form ***get\_data*** .

    select kunnr, name1, cast( 0 as DEC ) as Amount from kna1 into TABLE @data(*gt\_kna1*)

    where kunnr in *@s\_kunnr*.

   if *gt\_kna1*[] is NOT INITIAL.

   select \* from bsid into TABLE @data(*gt\_bsid*) FOR ALL ENTRIES IN *@gt\_kna1*

   WHERE kunnr = @gt\_kna1-kunnr and BLART = 'BR'.

loop at *gt\_kna1* ASSIGNING FIELD-SYMBOL(<fs1>).

     <fs1>-amount = *REDUCE* i( INIT i type dmbtr for wa\_bsid in gt\_bsid WHERE ( kunnr = <fs1>-kunnr )  
                                  NEXT i = i + wa\_bsid-dmbtr ).

   endloop.  
    endif.  
       cl\_demo\_output=>display( gt\_kna1 ).  
 endform.

**VALUE Operator**

[Содержание](#Содержание)

**VALUE type( )** - creates a result of a data type specified using *type*.

**VALUE #( )** - if no parameters are specified in the parentheses the return value is set to its type-specific *initial value*. This is possible for any non-generic data types dtype.

Example

Creates a suitable initial structure for a *non-optional* input parameter of a method.

CLASS **c1** *DEFINITION*.

  PUBLIC SECTION.

    TYPES: BEGIN OF *t\_struct*,

             col1 TYPE i,

             col2 TYPE i,

           END OF t\_struct.

*CLASS-METHODS* ***m1*** IMPORTING *p* TYPE t\_struct.

ENDCLASS.

CLASS **c1** *IMPLEMENTATION*.

  METHOD ***m1***.

    ...

  ENDMETHOD.

ENDCLASS.

START-OF-SELECTION.

  c1*=>*m1( *VALUE #( )* ).

## [**ABAP 740 – VALUE Operator to create ITAB entries**](http://zevolving.com/2014/09/abap-740-value-operator-create-itab-entries/)

This ***VALUE*** operator works similarly to the [***NEW*** Operator to create the ITAB entries](http://zevolving.com/2014/09/abap-740-new-operator-create-itab-entries/).

Rule of thumb to use VALUE would be same as NEW for ITAB

1. Start a new row with **(**,
2. Specify the name of the component and relevant value,
3. When done with that particular row finish it with**)**

data(*itab*)= *VALUE* t\_itab( (100 )( )( 300 ) ).

DATA *itab\_2* TYPE t\_itab.

itab\_2 = *VALUE #* (100)( )(300 ) ).

data(itab\_sorted) = *VALUE* t\_itab\_sorted ( (100)( )(300 ) ) .

data(itab\_sorted\_c) = *VALUE* tt\_sorted( ( num = 100 ) ( ) ( num = 300 )).

### Example 1 – Standard table with Component as TABLE\_LINE

Since I declared the variable using the TYPE for the itab, I can use the # to let system determine the type.

TYPES t\_itab TYPE STANDARD TABLE OF i

WITH DEFAULT KEY .

\* classical

DATA itab\_o TYPE t\_itab.

APPEND: 100 TO itab\_o,

0 TO itab\_o,

300 TO itab\_o.

\* using VALUE - Variation 1

DATA(itab) = VALUE t\_itab( ( 100 ) ( ) ( 300 ) ).

\* using VALUE - Variation 2

DATA itab\_2 TYPE t\_itab.

itab\_2 = VALUE #( ( 100 ) ( ) ( 300 ) ).

### Example 2 – Sorted table with component as TABLE\_LINE

Sorted table with TABLE\_LINE as component

TYPES t\_itab\_sorted TYPE SORTED TABLE OF i

WITH UNIQUE KEY table\_line.

\* classical

DATA itab\_s\_o TYPE t\_itab\_sorted.

READ TABLE itab\_s\_o TRANSPORTING NO FIELDS WITH KEY table\_line = 100.

IF sy-subrc NE 0.

INSERT 100 INTO itab\_s\_o INDEX sy-tabix.

ENDIF.

READ TABLE itab\_s\_o TRANSPORTING NO FIELDS WITH KEY table\_line = 0.

IF sy-subrc NE 0.

INSERT 0 INTO itab\_s\_o INDEX sy-tabix.

ENDIF.

READ TABLE itab\_s\_o TRANSPORTING NO FIELDS WITH KEY table\_line = 300.

IF sy-subrc NE 0.

INSERT 300 INTO itab\_s\_o INDEX sy-tabix.

ENDIF.

\* using VALUE

DATA(itab\_sorted) = VALUE t\_itab\_sorted( ( 100 ) ( ) ( 300 ) ).

### Example 3 – Sorted table with a specific component

Sorted table with Specific component

TYPES:

BEGIN OF ty\_sorted,

num TYPE i,

END OF ty\_sorted,

tt\_sorted TYPE SORTED TABLE OF ty\_sorted

WITH UNIQUE KEY num.

\* Using VALUE

DATA(itab\_sorted\_c) = VALUE tt\_sorted( ( num = 100 )

( )

( num = 300 )

).

### Example 4 – Table with more components

Standard table with more components

TYPES:

BEGIN OF ty\_data,

kunnr TYPE kunnr,

name1 TYPE name1,

ort01 TYPE ort01,

land1 TYPE land1,

END OF ty\_data.

TYPES: tt\_data TYPE STANDARD TABLE OF ty\_data

WITH DEFAULT KEY.

\* classical

DATA: itab\_multi\_c TYPE tt\_data.

FIELD-SYMBOLS: <fs> LIKE LINE OF itab\_multi\_c.

APPEND INITIAL LINE TO itab\_multi\_c ASSIGNING <fs>.

<fs>-kunnr = '123'.

<fs>-name1 = 'ABCD'.

<fs>-ort01 = 'LV'.

<fs>-land1 = 'NV'.

APPEND INITIAL LINE TO itab\_multi\_c ASSIGNING <fs>.

<fs>-kunnr = '456'.

<fs>-name1 = 'XYZ'.

<fs>-ort01 = 'LA'.

<fs>-land1 = 'CA'.

\* Using VALUE

DATA(itab\_multi\_comp) =

VALUE tt\_data( ( kunnr = '123' name1 = 'ABCD' ort01 = 'LV' land1 = 'NV' )

( kunnr = '456' name1 = 'XYZ' ort01 = 'LA' land1 = 'CA' )

### Example 5 – Table with Deep structure

Standard table with deep structure

TYPES:

BEGIN OF ty\_alv\_data,

kunnr TYPE kunnr,

name1 TYPE name1,

ort01 TYPE ort01,

land1 TYPE land1,

t\_color TYPE lvc\_t\_scol,

END OF ty\_alv\_data.

TYPES: tt\_alv\_data TYPE STANDARD TABLE OF ty\_alv\_data

WITH DEFAULT KEY.

\* classical

DATA: itab\_alv\_c TYPE tt\_alv\_data.

FIELD-SYMBOLS: <fs\_a> LIKE LINE OF itab\_alv\_c,

<fs\_col> LIKE LINE OF <fs\_a>-t\_color.

APPEND INITIAL LINE TO itab\_alv\_c ASSIGNING <fs\_a>.

<fs\_a>-kunnr = '123'.

<fs\_a>-name1 = 'ABCD'.

<fs\_a>-ort01 = 'LV'.

<fs\_a>-land1 = 'NV'.

APPEND INITIAL LINE TO <fs\_a>-t\_color ASSIGNING <fs\_col>.

<fs\_col>-fname = 'KUNNR'.

<fs\_col>-color-col = col\_negative.

<fs\_col>-color-int = 0.

<fs\_col>-color-inv = 0.

APPEND INITIAL LINE TO <fs\_a>-t\_color ASSIGNING <fs\_col>.

<fs\_col>-fname = 'ORT01'.

<fs\_col>-color-col = col\_total.

<fs\_col>-color-int = 1.

<fs\_col>-color-inv = 1.

APPEND INITIAL LINE TO itab\_alv\_c ASSIGNING <fs\_a>.

<fs\_a>-kunnr = '456'.

<fs\_a>-name1 = 'XYZ'.

<fs\_a>-ort01 = 'LA'.

<fs\_a>-land1 = 'CA'.

\* Using VALUE

DATA(itab\_alv) =

VALUE tt\_alv\_data(

"First Row

( kunnr = '123' name1 = 'ABCD'

ort01 = 'LV' land1 = 'NV'

" color table

t\_color = VALUE #(

" Color table - First Row

( fname = 'KUNNR'

color-col = col\_negative

color-int = 0

color-inv = 0

)

" Color Table - 2nd Row

( fname = 'ORT01'

color-col = col\_total

color-int = 1

color-inv = 1

)

)

)

"Second row

( kunnr = '456' name1 = 'XYZ'

ort01 = 'LA' land1 = 'CA'

)

).

# Loop Control

[Содержание](#Содержание)

|  |
| --- |
| **WHILE** <logical expression>  …  ENDWHILE. |
|  |
| **DO** [n TIMES].  …  ENDDO. |

## **Loop Control Statements**

|  |
| --- |
| [**CONTINUE**](https://www.tutorialspoint.com/sap_abap/sap_abap_continue_statement.htm) - causes the loop to *skip* the remainder of its body and *starts the next loop pass*. |
| [**CHECK**](https://www.tutorialspoint.com/sap_abap/sap_abap_check_statement.htm) - if the condition is *false* then the remaining statements after the CHECK are just ignored and the system *starts the next loop pass*. |
| [**EXIT**](https://www.tutorialspoint.com/sap_abap/sap_abap_exit_statement.htm) - *terminates the loop entirely* and transfers execution to the statement immediately following the loop. |

## Example

Report YH\_SEP\_15.

DO 5 TIMES.

*CHECK* SY-INDEX BETWEEN 3 AND 4.

Write / SY-INDEX.

ENDDO.

# Decisions IF | CASE -> COND

[Содержание](#Содержание)

**IF - ELSE**

|  |  |
| --- | --- |
| *if*<condition\_1>.  …  endif. | *if*<condition\_1>.  <statement block 1>.  *else*.  <statement block 2>.  endif. |

**COND instead of IF - ELSE /from abap 7.4/**

[Содержание](#Содержание)

DATA : lv\_indicator TYPE i.

DATA : lv\_desc TYPE char30.

START-OF-SELECTION.

*lv\_desc* = ***cond*** char30(

when *lv\_indicator = 1* then 'January'

when *lv\_indicator = 2* then 'February'

else 'Nothing').

WRITE lv\_desc.

**CASE**

*case* <field>.

when *<abc>*. <statement block>.

when *<def>*. …

when OTHERS. …

endcase.

*Rem*

* *No logical expressions* can be used for the ***<field>*** field.
* The field strings used in the CASE statement *are treated as type C variables*.

**SWITCH instead of CASE /from abap 7.4/**

[Содержание](#Содержание)

Using ***case*** you need to keep mentioning *what variable you’re filling* in every branch in CASE statement

***case*** *lv\_indicator*.

when 1. *lv\_day* = 'January'.

when 2. *lv\_day* = 'February'.

...

endcase.

With ***switch*** statement you don’t need mention variable in every branch

lv\_indicator = 1.

data(*lv\_day*) = ***switch*** char10(*lv\_indicator*)

when 1 then 'January'.

when 2 then 'February'.

write lv\_day.

# Strings

[Содержание](#Содержание)

We use data type **C** variables for holding alphanumeric characters with a minimum of 1 character and a maximum of 65,535 characters. By default these *are aligned to the left*.

Data my\_char*(5)* VALUE *'Hello'*.

The ***strlen***() function returns the number of characters contained in the string

DATA:

title\_1(10) VALUE 'Tutorials',

length\_1 TYPE I.

length\_1 = *strlen*( title\_1 ).

***Statements for strings manipulation***

|  |
| --- |
| **CONCATENATE**  *CONCATENATE* title\_1 title\_2 *INTO* dest2 [*SEPARATED BY* sep].  *Rem*  sep – SPACES | ' / ' |
| **CONDENSE** - *removes blank spaces between the fields* leaving only 1 character’s space.  *CONDENSE* spaced\_title [*NO-GAPS*].  *Rem*  *NO-GAPS* - *removes all spaces* |
|  |
| **REPLACE** - used to make replacements in characters. |
| **SEARCH** - to run searches in character strings. |
| **SHIFT** - used to *move the contents of a string left or right*. |
| **SPLIT** - used to *split the contents of a field into two or more fields*. |

**New String Features**

[Содержание](#Содержание)

1. ***pipes*** and ***curly bracket*** instead of *concatenate*.

*Before ABAP 7.4*

*CONCATENATE* lv\_temp lv\_status *INTO* lv\_result *SEPARATED BY* ' / '.

*Now*

lv\_string = *|* My First String *{* lv\_number *}* / *{* lv\_status *} |.*

1. ***alpha*** with OUT or IN instead of ***conversion\_exit\_ alpha\_input*** and ***conversion\_exit\_alpha\_output***

DATA : lv\_kunnr TYPE kunnr VALUE '110000010'.

START-OF-SELECTION.

DATA(lv\_real) = |{ lv\_kunnr *ALPHA = IN* } |.

WRITE lv\_real.

**Abap pipes**

ABAP [***string template***](http://help.sap.com/abapdocu_731/en/abenstring_templates.htm) is defined with two pipe sign symbols *|* - the text in between the two pipe sign is considered as the character string. The string template can only be used with the data objects of a type String. String templates can have [*expressions*](http://help.sap.com/abapdocu_731/en/abenstring_templates_expressions.htm)| [*characters strings*](http://help.sap.com/abapdocu_731/en/abenstring_templates_literals.htm) and [*control characters*](http://help.sap.com/abapdocu_731/en/abenstring_templates_separators.htm) within it - this would greatly reduce the need of the helper variables similar as [method chaining](http://zevolving.com/2013/02/abap-method-chaining-new-feature-in-abap-7-0-ehp2/).

*To distinguish an expression from regular text*, you would need to wrap the expression in the curly brackets { .. } - *the expression within the brackets would be evaluated first* and result would be than converted to character string.

lv\_string && *|Value|* & *|{sy-index \* 10}|* && ‘,’.

## **Subfields**

Subfields are used to save time on creating unnecessary variables in memory.

DATA: int\_phone\_num(17) TYPE c,

country\_code(3) TYPE c,

phone\_num(14) TYPE c.

int\_phone\_num = '+44-(0)207-123456'.

country\_code = int\_phone\_num(3). *' +44*

phone\_num = int\_phone\_num+4(13). *' (0)207-123456*

*country\_code + 1*(2) = '01'. *' +01*

# Date & Time

[Содержание](#Содержание)

**D -** YYYYMMDD.

**T –** HHMMSS.

**TIMESTAMP** - YYYYMMDDhhmmss.

**TIMESTAMPL** - YYYYMMDDhhmmss,mmmuuun; где mmmuuun - the fractions of a second.

## ***Current Date and Time*** –

## ***SY-DATUM*** – TYPE D.

## ***SY-UZEIT*** – TYPE T.

The system field **SY-ZONLO** is used to display *the local time zone* configured in the user’s preferences.

TIME ZONE *SY-ZONLO*.

# ABAP Selection Screen

# <https://www.saptutorial.org/abap-selection-screen-part-5/>

[Содержание](#Содержание)

| **Keyword** | **Description** |
| --- | --- |
| SELECTION-SCREEN | Defines a screen |
| PARAMETER | Gets input for a single variable from the user |
| SELECT-OPTIONS | Gets a range of data from the user |
| BLOCK | Used to organize the different input items on the screen |

You can gather data from the user using the **PARAMETER** and **SELECT-OPTIONS** keywords but *those keywords must be contained* inside a SCREEN or a within a BLOCK inside a screen.

### **SELECTION-SCREEN**

Screens are given a unique number; every ABAP program contains a standard screen 1000 when created.

If you want to create additional screens to use in addition to the default screen 1000 - you can do so with the **BEGIN OF SCREEN** keyword. You can then call your screen using the **CALL SELECTION-SCREEN** keyword - because the *default screen* is always 1000 - *your new screen* will not be called unless you specifically call it.

SELECTION -SCREEN *BEGIN OF SCREEN* 2000.

PARAMETER p\_input TYPE string .

SELECTION-SCREEN *END OF SCREEN 2000.*

*CALL SELECTION-SCREEN* 2000.

**BLOCK**

The ***BLOCK*** is used to organize the input elements within the selection.

***A block can***

* have any alphanumeric name;
* be reused across multiple screens in an ABAP program.

| **BLOCK Option** | **Description** |
| --- | --- |
| WITH FRAME | Adds a frame around the input options inside of the block |
| WITH FRAME TITLE ( title ) | Adds a title to the top of the block |
| NO INTERVALS | Removes the second input box from SELECT -OPTIONS within the block to make the entire block have a smaller width. |

SELECTION-SCREEN *BEGIN OF BLOCK* blk1 *WITH FRAME TITLE* TEXT-f01.

PARAMETERS: p\_file LIKE rlgrap-filename *OBLIGATORY*.

PARAMETERS : cb\_test *AS CHECKBOX  DEFAULT* 'X'.

SELECTION-SCREEN END OF BLOCK blk1.

The **title** can either be a *free name* with a maximum of *eight* characters or the name of a *text symbol* from the program in the form *text-idf* where *idf* is the three-character ID of the text symbol.

INITIALIZATION. title = *'Selection'*.

***To create text symbol*** for title of block and Selection Text for description of parameter in SE80 you can open menu Goto -> Text Elements -> Text Symbols or the other way creating text symbol you can *double click on ABAP code on after FRAME TITLE TEXT-F01*.

### **PARAMETER**

| **Parameter Option** | **Description** |
| --- | --- |
| *OBLIGATORY* | User cannot execute the program until a value is entered. |
| *DEFAULT* | Set a default value for the parameter. |
| *AS CHECKBOX* | Parameter is set as a checkbox that will return ABAP \_TRUE or ABAP FALSE when selected or not selected. |
| *RADIOBUTTON GROUP* group | Works like a checkbox but will only allow one parameter to be selected within a defined group. |
| *AS LISTBOX VISIBLE LENGTH* n | Will create a dropdown selection for all possible optiom for the parameter data type. The dropdown list width is defined by n. |

SELECTION-SCREEN BEGIN OF BLOCK blk2 WITH FRAME TITLE TEXT-f02.

*PARAMETERS*:

rb\_sto *RADIOBUTTON GROUP* b1 DEFAULT 'X',

            rb\_srv RADIOBUTTON GROUP b1,

            rb\_chg RADIOBUTTON GROUP b1,

            rb\_stk RADIOBUTTON GROUP b1,

            rb\_pro RADIOBUTTON GROUP *b1*.

SELECTION-SCREEN END OF BLOCK blk2.

### **SELECT-OPTIONS**

The SELECT-OPTIONS keyword allows for the user *to select from a range of options*. Unlike PARAMETER SELECT-OPTIONS *requires a variable based on a data types database table* in order to define what options are available to be selected.

*SELECT-OPTIONS*: s\_banfn *FOR* eban-banfn .

**Database Access */****from ABAP 7.4****/***

[Содержание](#Содержание)

**CASE statement in SQL Queries**

START-OF-SELECTION.

SELECT *CASE* WHEN auart = 'Z1IN' THEN 'Z1IN'

WHEN auart = 'Z2KP' THEN 'Z2KP'

*ELSE* 'OTHERS'

END as group, vbeln

FROM vbak INTO TABLE *@data*(li\_vbeln).

*Rem*

**@** - *you are not talking about field in the database*.

@DATA(li\_vbeln) - SAP System will create internal table *li\_vbeln* directly - so you don’t need declare before.

**INNER JOIN Improvement**

Befor 7.4 you must to list field one by one from the main table - this improvement *you can list all of the fields*

SELECT *a~\**, b~posnr, b~matnr FROM vbak AS a

INNER JOIN vbap as b ON a~vbeln = b~vbeln

WHERE a~auart = 'Z1IN'

INTO TABLE @data(li\_vbeln).

**AMDP – основы**

[Содержание](#Содержание)

# ABAP Managed Database Procedures (AMDP)

<https://abap-blog.ru/database-work/abap-managed-database-procedures-amdp/>

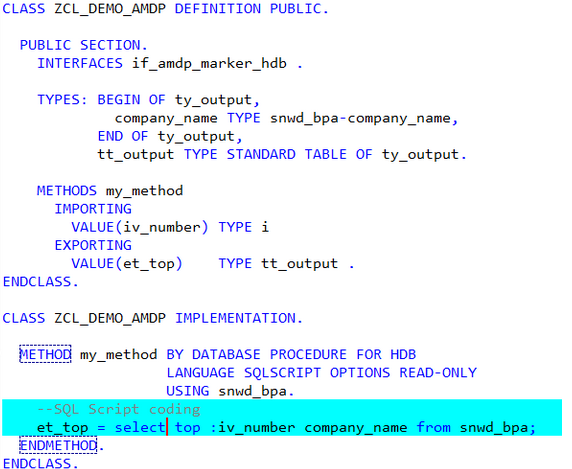
Технически AMDP это обёртка над [хранимыми процедурами](https://ru.wikipedia.org/wiki/%D0%A5%D1%80%D0%B0%D0%BD%D0%B8%D0%BC%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%B0) SAP HANA, которая заворачивает Native SQL код в AMDP методы, которые можно использовать в abap как обычные методы обычных классов.

Во время первого доступа к такому методу, на сервере БД будет создана хранимая процедура, которая создаётся в основной схеме для сервера приложений с именем SAPSid, где Sid – идентификатор системы. Доступ из хранимой процедуры можно получить к объектам БД текущей схемы (SAPSid) и объектам других схем, явно указав их имена.

***Интерфейс IF\_AMDP\_MARKER\_HDB***

Данный интерфейс является меткой для системы, что в методах этого класса может быть встроен вызов AMDP. Кроме AMDP методов такой класс может содержать и обычные методы.

В ABAP 7.4. можно реализовать только метод AMDP процедуры, в 7.5 добавилась так же возможность создания метода AMDP функции. AMDP функции можно использовать для получения данных в других AMDP процедурах и функциях, в ABAP CDS и даже в OpenSQL. Для функций параметр RETURNING является обязательным. В статье реализация функций не рассматривается, подробнее о функциях смотрите в [документации](http://help.sap.com/abapdocu_750/en/abenamdp_function_methods.htm#@@ITOC@@ABENAMDP_FUNCTION_METHODS_2).



# AMDP Функции

<https://abap-blog.ru/osnovy-abap/amdp-funkcii/>

[Содержание](#Содержание)

AMDP функция реализуется как AMDP метод, но с указанием дополнения *BY DATABASE FUNCTION*

METHOD ***meth*** *BY DATABASE FUNCTION*

            FOR db

            LANGUAGE db\_lang

            [OPTIONS db\_options]

            [USING   db\_entities]

            [USING   SCHEMA schema1 OBJECTS db\_entities]

            [USING   SCHEMA schema2 OBJECTS db\_entities]

            ... .

  ...

ENDMETHOD.

Существует следующие виды AMDP функций

* **AMDP *табличная функция*** - функция БД, которая в качестве результата возвращает табличный результат. В SQLScript может быть использована как источник данных вместо таблиц или ракурсов.
  + AMDP табличная функция *для AMDP методов*.
  + AMDP табличная функция *для табличной функции ABAP CDS*.
* **AMDP *скалярная функция*** - функция БД, которая в качестве результата возвращает результат [элементарного типа](https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-US/abenelementary_data_type_glosry.htm).

*Rem*

Несмотря на возможность вызова AMDP функций из объектов HANA Native разработки, таких как хранимые процедуры или calculation view, SAP [не рекомендует](https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-US/abendatabase_access_recomm.htm) так делать.

### ***Табличная функция для AMDP методов***

Особенности

* Функция должна иметь *RETURNING* параметр табличного типа, все компоненты структуры которого должны иметь [элементарный тип](https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-US/abenelementary_data_type_glosry.htm).
* Не допускается использование *~~CHANGING~~* или *~~EXPORTING~~* параметров.
* Не допускается объявление исключений в определении через дополнение *~~RAISING~~*.
* Обязательно должно быть указание *READ-ONLY*.
* Нельзя вызвать непосредственно из ABAP кода.

CLASS **zcl\_amdp\_function** *DEFINITION*

  …

  PUBLIC SECTION.

    INTERFACES: if\_amdp\_marker\_hdb.

    CLASS-METHODS:

***get\_flight\_with\_max\_sum***

IMPORTING VALUE(*iv\_carrid*)  TYPE s\_carr\_id

                                      VALUE(*iv\_connid*)  TYPE s\_conn\_id

                RETURNING VALUE(*rt\_flights*) TYPE ty\_flights.

  …

ENDCLASS.

CLASS **zcl\_amdp\_function** *IMPLEMENTATION*.

  METHOD ***get\_flight\_with\_max\_sum*** *BY DATABASE FUNCTION*

                                 FOR HDB LANGUAGE SQLSCRIPT

                                 OPTIONS READ-ONLY

                                 USING sflight

                                 zcl\_amdp\_scalar\_function=>get\_max\_payment\_sum.

      RETURN SELECT \* FROM sflight

WHERE mandt = session\_context('CLIENT') AND paymentsum =

"ZCL\_AMDP\_SCALAR\_FUNCTION=>GET\_MAX\_PAYMENT\_SUM" (

iv\_mandt => SESSION\_CONTEXT('CLIENT'),

                                       iv\_carrid => :iv\_carrid,

                                                                     iv\_connid => :iv\_connid );

  ENDMETHOD.

ENDCLASS.

При вызове из ABAP на этапе компиляции не будет предупреждения, но в момент запуска возникнет дамп - ***CALL\_METHOD\_AMDP\_FUNC\_ILLEGAL***

PARAMETERS: p\_carrid TYPE s\_carr\_id,

             p\_connid TYPE s\_conn\_id.

START-OF-SELECTION.

    DATA(lt\_flights) = zcl\_amdp\_function=>get\_flight\_with\_max\_sum( iv\_carrid = p\_carrid

                                                                        iv\_connid = p\_connid ).

### ***Табличная функция для ABAP CDS Table Functions***

В ABAP CDS можно объявить табличную функцию, основное предназначение которой - предоставить набор данных, сформированный внутри AMDP функции. Сделан такой функционал прежде всего для возможностей использования HANA Native функционала, который пока по той или иной причине не доступен в ABAP CDS.

Особенности

* Ф-ция может быть объявлена только как статический метод с публичной областью видимости. Не допускается объявление через интерфейсы.
* В определении метода должно использоваться ключевое дополнение ***FOR TABLE FUNCTION*** имя\_ABAP\_CDS\_функции [⇒ одна и та же ф-ция может быть использована только для одной ABAP CDS Table function].
* Параметры AMDP табличной ф-ции не объявляются в описании метода, а должны быть описаны как параметры ABAP CDS. Параметры всегда имеют элементарный тип, не допускается опциональных параметров.
* Возвращаемый табличный тип определяется относительно структуры ABAP CDS, для зависимых от манданта CDS в структуре компонентов будет так же поле мандант. Тип таблицы - стандартная, ключ по умолчанию с именем ***result*** включает в себя все поля таблицы.
* Не могут быть напрямую вызваны из ABAP.
* Сначала создаётcя ABAP CDS с описанием вх. параметров и структуры, а уже затем создаётся AMDP ф-ция.

Созданный таким образом ABAP CDS можно использовать в других CDS или непосредственно для вызова из ABAP кода.

ABAP CDSTable function

@ClientHandling.type: #CLIENT\_DEPENDENT

define *table function* ***ZDEMO\_CDS\_GET\_SCARR\_SPFLI***

*with parameters*

*@Environment.systemField*: #CLIENT

*clnt*   :abap.clnt,

*carrid* :s\_carr\_id

returns {

  client   :s\_mandt;

  carrname :s\_carrname;

  connid   :s\_conn\_id;

  cityfrom :s\_from\_cit;

  cityto   :s\_to\_city;

}

*implemented by method* zcl\_amdp\_cds\_function=>get\_scarr\_spfli\_for\_cds;

AMDP ф-ция

CLASS **zcl\_amdp\_cds\_function** *DEFINITION*

  …

  PUBLIC SECTION.

    INTERFACES: if\_amdp\_marker\_hdb.

    CLASS-METHODS ***get\_scarr\_spfli\_for\_cds*** *FOR TABLE FUNCTION* zdemo\_cds\_get\_scarr\_spfli.

…

ENDCLASS.

CLASS **zcl\_amdp\_cds\_function** *IMPLEMENTATION*.

  METHOD ***get\_scarr\_spfli\_for\_cds***

         BY DATABASE FUNCTION FOR HDB

         LANGUAGE SQLSCRIPT

         OPTIONS READ-ONLY

         USING scarr spfli.

    RETURN SELECT sc.mandt as client, sc.carrname, sp.connid, sp.cityfrom, sp.cityto

                   from *scarr* as sc

                    inner join *spfli* as sp on sc.mandt = sp.mandt and sc.carrid = sp.carrid

                    where sp.mandt = :clnt AND sp.carrid = :carrid

                    ORDER BY sc.mandt, sc.carrname, sp.connid;

  endmethod.

ENDCLASS.

Вызов из ABAP кода

PARAMETERS: *p\_carrid* TYPE s\_carr\_id,

*p\_connid* TYPE s\_conn\_id.

START-OF-SELECTION.

   SELECT \* FROM zdemo\_cds\_get\_scarr\_spfli( carrid = @p\_carrid )

    WHERE connid = @p\_connid INTO TABLE @DATA(*lt\_flights*).

# AMDP with SELECT OPTIONS

<https://sapyard.com/abap-on-sap-hana-part-x-amdp-with-select-options/>

<https://setevalapinsap.com/2021/11/abap-managed-database-procedures-with-select-options-sap#google_vignette>

[Содержание](#Содержание)

We cannot directly pass *SELECT options* *as is it* /как есть/ to AMDP methods ⇒ we need to select the data from the database and then apply the filter using the function ***APPLY\_FILTER***

*ex\_it\_tcode\_role* = *APPLY\_FILTER*( *:ex\_it\_tcode\_role*, :ip\_filters );

EX\_IT\_TCODE\_ROLE would have all the data and ***APPLY\_FILTER*** would keep the subset using IP\_FILTERS value /**IP\_FILTERS** has to be passed as STRING/.

# Accessing tables from different schema through AMDP

<https://blogs.sap.com/2015/02/18/accessing-tables-from-different-schema-through-amdp/>

[Содержание](#Содержание)

I got a business requirement on the data has to be taken from  different schema’s table which is **SLT** replicated.

We can point schema tables like *“RD2.VBRP”* or *RD2.VBRP*.

If you want to access current schema tables then you can add tables in *USING* clause.

INTERFACES : IF\_AMDP\_MARKER\_HDB.

CLASS-METHODS :

***GET\_DATA***

IMPORTING VALUE(ip\_where) TYPE STRING

EXPORTING VALUE(ex\_final) TYPE GT\_FINAL,

***F4\_SPART*** EXPORTING VALUE(ex\_tspatT) TYPE GT\_TSPAT,

METHOD ***GET\_DATA*** BY DATABASE PROCEDURE FOR HDB LANGUAGE SQLSCRIPT

OPTIONS READ-ONLY.

it\_final = SELECT … FROM *”RD2"."VBRP"*

INNER JOIN "RD2"."VBRK" ON …

WHERE VBRK.FKSTO NOT IN ('N',’E’,’U’)

GROUP BY VBRP.ARKTX, …;

it\_query = select \* from *:it\_final* as A;

ex\_final = *APPLY\_FILTER*(:it\_query, :ip\_where);

ENDMETHOD.

METHOD ***F4\_SPART*** BY DATABASE PROCEDURE FOR HDB LANGUAGE SQLSCRIPT

OPTIONS READ-ONLY.

ex\_tspat = SELECT SPART, VTEXT FROM *RD2.TSPAT*

WHERE SPRAS = ’E‘

ORDER BY SPART;

*Program*

AT SELECTION-SCREEN ON VALUE-REQUEST FOR S\_SPART-LOW. *”F4 help*

call method ZCL\_DISPATCH=>*F4\_SPART* importing ex\_tspat = GT\_SPART.

START-OF-SELECTION.

TRY.

call method ZCL\_DISPATCH=>*GET\_DATA*

exporting ip\_where = GW\_WHERE *"where condition in string*

importing ex\_final = DATA(GT\_FINAL).

CATCH CX\_AMDP\_ERROR INTO DATA(AMDP\_ERROR).

ENDTRY.

IF GT\_FINAL IS NOT INITIAL.

*"Create field catalog and display ALV*

CALL SCREEN 9000.

ENDIF.

# Create an ABAP Managed Database Procedure /AMDP/ and Analyze Its Performance

<https://developers.sap.com/tutorials/abap-environment-amdp-profiling.html>

[Содержание](#Содержание)

*Step 1* ***Create an ABAP class***

ZCL\_AMDP\_DEMO\_XXX

*Step 2* ***Add two interfaces***

Add two interfaces by adding this code to the public section. Ignore the warning for now

INTERFACES: if\_amdp\_marker\_hdb,

if\_oo\_adt\_classrun.

* ***if\_amdp\_marker\_hdb*** - defines the class as an *AMDP* class, allowing you to implement AMDP methods - that is, ABAP methods that call a *SAP HANA database procedure* from within a global ABAP class.
* ***if\_oo\_adt\_classrun*** - allows you to output the results to the ABAP Console.

*Step 3* ***Create structures and table types***

Add these structures and types to the public section, just after the interface definitions

TYPES:

BEGIN OF ty\_result\_line,

airline TYPE /dmo/carrier\_name,

flight\_connection TYPE /dmo/connection\_id,

old\_price TYPE /dmo/flight\_price,

old\_currency TYPE /dmo/currency\_code,

new\_price TYPE /dmo/flight\_price,

new\_currency TYPE /dmo/currency\_code,

END OF ty\_result\_line,

BEGIN OF ty\_flights\_line,

airline TYPE /dmo/carrier\_name,

flight\_connection TYPE /dmo/connection\_id,

price TYPE /dmo/flight\_price,

currency TYPE /dmo/currency\_code,

END OF ty\_flights\_line,

ty\_result\_table TYPE STANDARD TABLE OF ty\_result\_line WITH EMPTY KEY,

ty\_flights\_table TYPE STANDARD TABLE OF ty\_flights\_line WITH EMPTY KEY,

ty\_flights TYPE STANDARD TABLE OF /dmo/flight.

*Step 4* ***Add method definitions***

METHODS:

get\_flights EXPORTING VALUE(result) TYPE ty\_result\_table

RAISING cx\_amdp\_execution\_error,

convert\_currency IMPORTING VALUE(flights) TYPE ty\_flights\_table

EXPORTING VALUE(result) TYPE ty\_result\_table

RAISING cx\_amdp\_execution\_error.

*Step 5* ***Implement get\_flights***

You must specify all ABAP tables, views, and AMDP procedures in the **USING** clause.For more details on these clauses, see [ABAP Keyword Documentation: Method - By Database Procedure, Function](https://help.sap.com/doc/abapdocu_753_index_htm/7.53/en-US/index.htm?file=abapmethod_by_db_proc.htm).

METHOD ***GET\_FLIGHTS*** by database procedure

for hdb language sqlscript options read-only

using

/dmo/flight

/dmo/carrier

ZCL\_AMDP\_DEMO\_XXX=>*convert\_currency*.

flights = select distinct c.name as airline,

f.connection\_id as flight\_connection,

f.price as price, f.currency\_code as currency

from "/DMO/FLIGHT" as f

inner join "/DMO/CARRIER" as c on f.carrier\_id = c.carrier\_id;

call "ZCL\_AMDP\_DEMO\_XXX=>*CONVERT\_CURRENCY*"( :flights, result );

ENDMETHOD.

*Step 6* ***Implement the method convert\_currency***

METHOD ***convert\_currency*** BY DATABASE PROCEDURE

FOR HDB LANGUAGE SQLSCRIPT OPTIONS READ-ONLY.

declare today date;

declare new\_currency nvarchar(3);

select current\_date into today from dummy;

new\_currency := 'EUR';

result = select distinct airline, flight\_connection, price as old\_price, currency as old\_currency,

*convert\_currency*(

"AMOUNT" => price,

"SOURCE\_UNIT" => currency,

"TARGET\_UNIT" => :new\_currency,

"REFERENCE\_DATE" => :today,

"CLIENT" => '100',

"ERROR\_HANDLING" => 'set to null',

"SCHEMA" => current\_schema

) as new\_price,

*:new\_currency* as new\_currency

from :flights;

ENDMETHOD.

*Step 7* ***Implement the method main of the interface if\_oo\_adt\_classrun***

Implement the main method of the interface if\_oo\_adt\_classrun. This will allow you to output your results to the ABAP Console.

METHOD *if\_oo\_adt\_classrun~****main***.

TRY.

me->get\_flights(IMPORTING result = DATA(lt\_result) ).

CATCH cx\_amdp\_execution\_error INTO DATA(lx\_amdp).

out->write( lx\_amdp->get\_longtext( ) ).

ENDTRY.

*“ Output the result to the console*

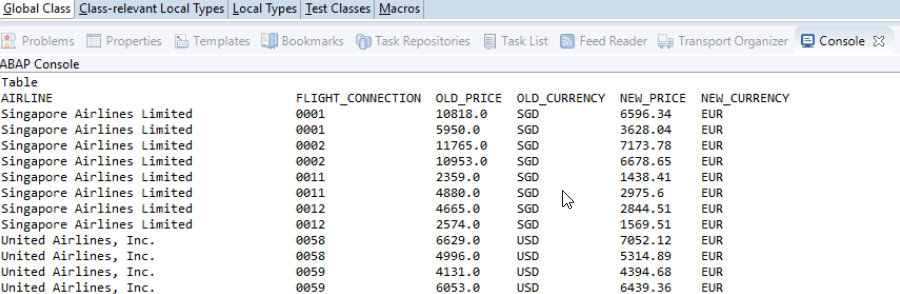
out->write( lt\_result ).

ENDMETHOD.

*Step 8* ***Format, save, activate, and test your code***

1. Format - **Shift+F1**, save - **Ctrl+S**, and activate - **Ctrl+F3** your code.
2. Test your class by running it the ABAP Console - **F9**.

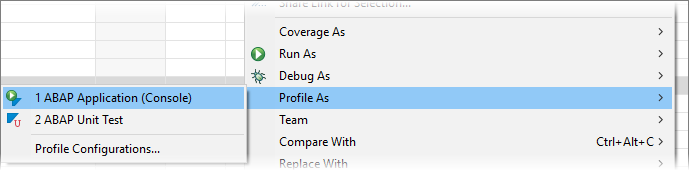
Your output should look like this



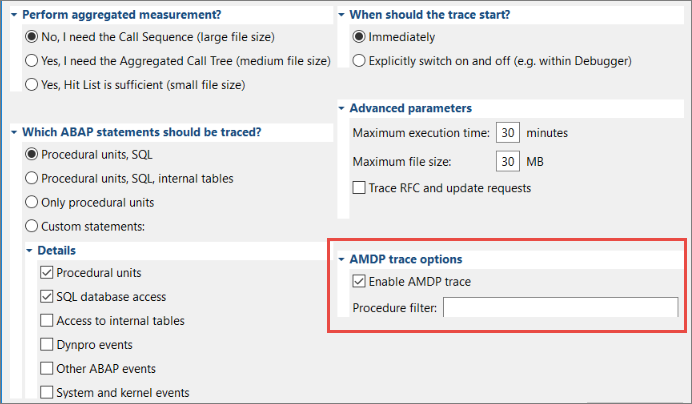
*Step 9* ***Set the Run configuration to ABAP Console***

You will run your class again in the ABAP Console. First, however, you will specify that the **AMDP trace** is to be enabled this time.

1. From the context menu of the editor, choose ***Profile As > ABAP Application (Console)***



1. In the dialog that appears, choose ***AMDP trace options > Enable AMDP trace***. Leave the other settings as they are, then choose ***Finish***

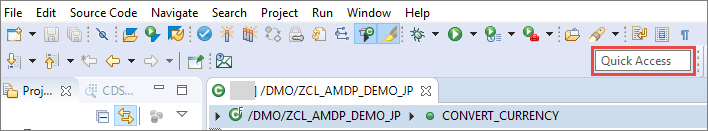


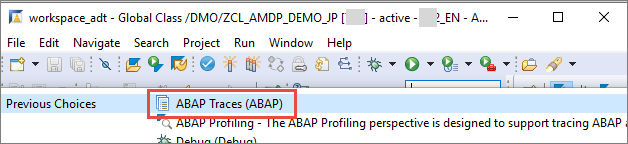
The class will run again, outputting the same data to the Console and also creating an ABAP Trace.

*Step 10* ***Open the AMDP tab***

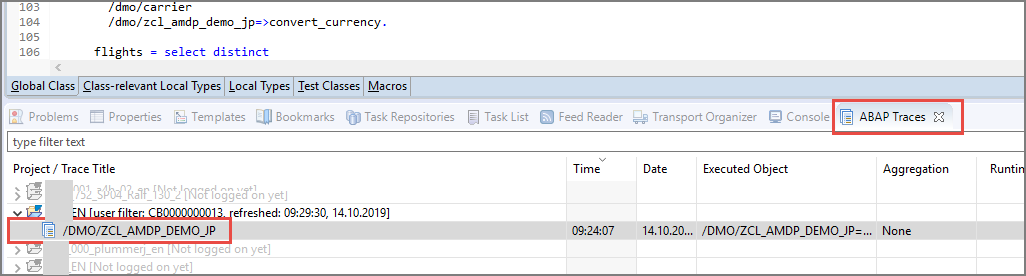
Now, you will examine the ABAP Trace you have created.

1. Open the **ABAP Traces** view, by choosing it from **Quick Access**

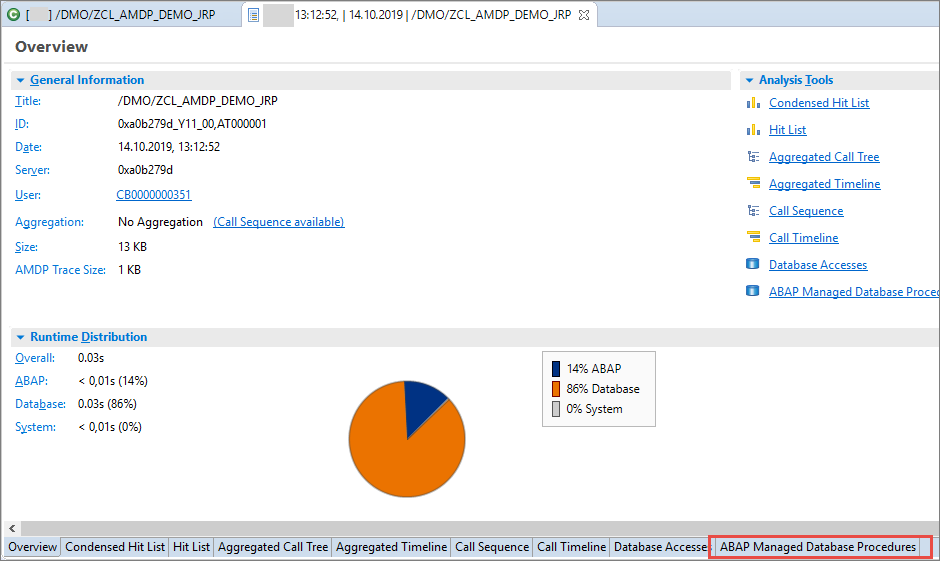




1. The view opens below the Class Editor, beside the Console. If necessary, refresh the view /**F5**/. Double-click on your class



1. The **Trace Results** view appears. The **ABAP Managed Database Procedures** tab appears to the right



# ****Чтение данных из C****V в ABAP

<https://helpbw.wordpress.com/2022/01/21/%D1%87%D0%B8%D1%82%D0%B0%D0%BC-calculation-view-%D0%B8%D0%B7-abap-%D1%87%D0%B5%D1%80%D0%B5%D0%B7-cds/>

**Через CDS Table function**

[Содержание](#Содержание)

# Предпосылки

# Условие *where* в ABAP при чтении данных из CV через *ракурс* /синоним/ может не срабатывать.

# Вариант чтения через *CDS Table function* позволяет

# использовать *Input* параметры;

# опускать *фильтры* в самый низ CV [на уровень Projection];

# использовать *where* условие корректно.

Пример будет на служебной таблице *RSAFORMMAP*, которая содержит поля

* FORMEX - его и будем фильтровать;
* OBJVERS;
* FORMIN;
* DELFLAG.

Создаем CV

"\_SYS\_BIC"."package\_01/***CV\_RSA\_FOR\_MMAP***"

# 

# Создадим input parameter /*pəˈræmɪtə*/ *****ip\_tst***** в CV и укажем его в фильтре.

# 

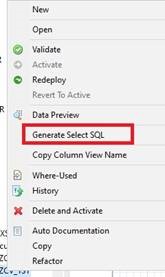
# На примере ниже он введён таким образом, что бы в случае фильтрации через  \* показывались все значения, в противном случае – только введённые. В самом input параметре при его создании указываем значение по умолчанию *Constant* и значение *****\******.

# Filter Expression

# *case*('$$ip\_formex$$'!= '\*')\*(*in*("FORMEX",'$$ip\_formex$$'))

# +

# *case*('$$ip\_formex$$'='\*')\**in*('\*','$$ip\_formex$$')

На созданном CV щелкаем пр. кнопкой мыши и выбираем пункт меню, чтобы сгенерировать SQL запрос .

SELECT "FORMEX", "OBJVERS", "FORMIN", "DELFLAG", sum("counter") AS "counter"

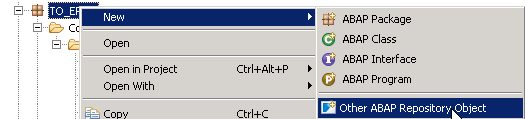
FROM "\_SYS\_BIC"."package\_01/CV\_RSA\_FOR\_MMAP"

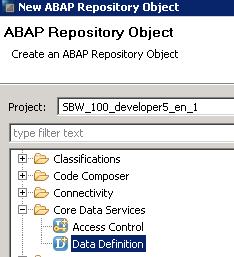
('PLACEHOLDER' = ('$$*ip\_formex*$$','<Enter Value>'))

GROUP BY "FORMEX", "OBJVERS", "FORMIN", "DELFLAG"

# Создаем CDS Table Function

Создаём CDS типа Table Function. Для этого в ABAP перспективе нажимаем на пакете или на всей системе пр. кнопкой мыши и выбираем ***New/Other ABAP Repository Object***.

 .

Далее - ***Core Data Service\Data Definition***  .

Выбираем указанный тип CDS - ***Define Table Function with Parameters***

@ClientDependent: false

@EndUserText.label: 'ztf\_rsa\_for\_mmap'

define table function ***ztf\_rsa\_for\_mmap***

with parameters *ip\_formex* : *abap*.*char*(25)

returns {

FORMEX : *abap*.*char*(25);

OBJVERS : *abap*.*char*(1);

FORMIN : *abap*.*char*(25);

DELFLAG : *abap*.*char*(1);

}

*implemented by* method *ZCL\_RSA=>ZM\_MMAP*;

@ClientDependent: *false* - без автообработки клиента, чтобы не использовать системный параметр CLNT дополнительно, который будет обязателен, если включить эту аннотацию.

Создаем AMDP класс с методом чтением данных из CV

class **ZCL\_RSA** *definition*

public

final

create public .

public section.

interfaces *if\_amdp\_marker\_hdb*.

class-methods:

***ZM\_MMAP*** for table function *ztf\_rsa\_for\_mmap*.

protected section.

private section.

ENDCLASS.

CLASS **ZCL\_RSA** *IMPLEMENTATION*.

method ***ZM\_MMAP***

*by database function for hdb*

*language sqlscript*

options *READ-ONLY*.

return

SELECT "FORMEX", "OBJVERS", "FORMIN", "DELFLAG"

FROM "\_SYS\_BIC"."package\_01/*CV\_RSA\_FOR\_MMAP*"

where FORMEX = *:ip\_formex*

GROUP BY "FORMEX", "OBJVERS", "FORMIN", "DELFLAG";

endmethod.

ENDCLASS.

Обратите внимание - параметр *ip\_formex* в AMDP классе нигде не определяется. Метод о параметре знает из нашей CDS Table function.

Создаем AMDP класс - потребителя CV

Вариант 1 - Данные из CV будем читать через CDS Table Function

class **ZCL\_RSA\_CONS** *definition*

public

final

create public .

public section.

class-methods ***get\_mmap***.

protected section.

private section.

ENDCLASS.

CLASS **ZCL\_RSA\_CONS** *IMPLEMENTATION*.

method ***get\_mmap***.

data: *lt\_data* type standard table of *rsaformmap*.

*##db\_feature\_mode[AMDP\_table\_function].*

select \*

from *ztf\_rsa\_for\_mmap*( ip\_formex = '00O2TN6ZGQI1S5MK1S6WABI8F' )

into corresponding fields of table *@lt\_data*.

cl\_demo\_output=>display( lt\_data ).

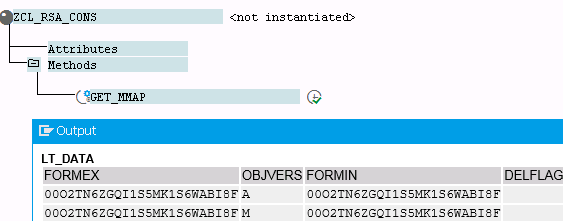
endmethod.

ENDCLASS.

Rem

Обратите внимание на прагму *##db\_feature\_mode[AMDP\_table\_function].* - она необходима для предотвращения ошибок синтаксиса при работе с CDS типа Table Function в ABAP.

Запускаем наш метод и видим результат



Вариант 2 - Данные из CV будем читать напрямуя [минуя CDS Table Function]

В созданном выше AMDP классе *ZCL\_RSA*создаемметод чтения CV.

class **ZCL\_RSA** *definition*

…

public section.

…

types: tt\_rsa\_mmap type standard table of rsaformmap.

class-methods: …, ***get\_direct*** EXPORTING VALUE(out\_tab) TYPE tt\_rsa\_mmap.

…

endclass.

class **ZCL\_RSA** *implementation*.

…

method ***get\_direct***

by database procedure for hdb

language sqlscript

options READ-ONLY.

out\_tab = select "FORMEX", "OBJVERS", "FORMIN", "DELFLAG"

from "\_SYS\_BIC"."package\_01/CV\_RSA\_FOR\_MMAP"

('PLACEHOLDER' = ('$$ip\_formex$$', '00O2TN6ZGQI1S5MK1S6WABI8F'));

endmethod.

endclass.

Сам AMDP метод вызываем из ABAP, как вызов обычного статичного метода класса.

method ***get\_direct***.

data: lt\_data type standard table of rsaformmap.

ZCL\_RSA=>*get\_direct*( importing out\_tab = lt\_data ).

cl\_demo\_output=>display( lt\_data ).

endmethod.

Rem

Чтение CV из AMDP с передачей input параметра в виде *переменной* и в виде *конкретного значения* может отличаться по синтаксису.

* *“ Передача ЗНАЧЕНИЯ*

**…from** "\_SYS\_BIC"."package\_01/*CV\_RSA\_FOR\_MMAP*"

('PLACEHOLDER' = ('$$ip\_formex$$', '36'))

* *“ Передача ПЕРЕМЕННОЙ*

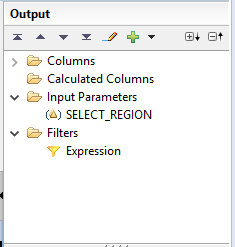
**…from** "\_SYS\_BIC"."package\_01/*CV\_RSA\_FOR\_MMAP*"

('PLACEHOLDER' = ('$$ip\_formex$$', :ip\_var))

**Create the filter using Expression**

<https://kabilsapworld.blogspot.com/2017/07/fliter-data-based-on-input-parameter.html>

[Содержание](#Содержание)

[](https://2.bp.blogspot.com/-KtHFgHp2VQA/WXrsPOxz3MI/AAAAAAAABb4/SAii1okix98Y2Qzg8W-S0h7cYWZBQ5fbgCLcBGAs/s1600/exp-output_panel.PNG)

Now enter the following expression in Expression Editor

*in*("LAND1",'$$SELECT\_REGION$$')

or

*match* ("LAND1",'***\****$$SELECT\_REGION$$*\**')

Rem

* ***in*** clause - to allow multiple values.
* ***match*** - to allow *"All"* value if *User Passes*

# [Input Parameter filtering with empty value](https://stackoverflow.com/questions/59050568/input-parameter-filtering-with-empty-value)

− I want to display all the data in the CV when the input parameter is left empty. However, when I pass a value in the input parameter, it does not work

***if***('$$CTUSER$$' = '',*match*("USER",'\*'),*in*("USER", '$$CTUSER$$'))

When I remove the single quotes around $$CTUSER$$ - the passed value is filtered correctly, but when the parameter is left blank, it does not work. What am I doing wrong?

− The ***single-quotes*** around the input parameter names need to stay in order to make them work correctly.

In order to make the filter condition a bit more *resilient* /rɪˈzɪlɪənt устойчивый, стойкий/, I changed it to this /note that I filter on ProductID, but you get the idea/

***if***( '$$ProductID$$'= '' or *isNULL*('$$ProductID$$')

, 1=1

, *in*("PRODUCTID", '$$ProductID$$')

)

Instead of the *match*() condition, I used the constant *1=1* expression to make it clearer, that this should always resolve to TRUE [and to reduce computation effort].

I also included a check for ***NULL*** of the parameter, as empty-string and NULL are two different "empty" values.

Note

* That it is possible to include ***line-breaks*** in the expression editor, which I highly recommend in order to keep the expressions readable.
* The ***IP\_*** notation for input parameters or values is not mandatory. Since there is no way to confuse those with column names, I do not see any value in using such prefixes..

# ABAP AMDP Procedure Output Parameter Type must be Structured

<https://www.kodyaz.com/sap-abap/amdp-procedure-method-output-parameter-type-must-be-structured.aspx>

[Содержание](#Содержание)

If method contains a *database procedure*

* row type of output parameters must be structured;
* all components of the row type must be elementary.

ABAP programmers developing AMDP procedures must convert output table parameters' *row types* to *structured row types*.

Пример

class-methods: ***get\_plant\_by\_salesorg***

AMDP OPTIONS READ-ONLY

CDS SESSION CLIENT CURRENT

importing

value(p\_mandt) type sy-mandt

value(p\_vkorg) type vkorg

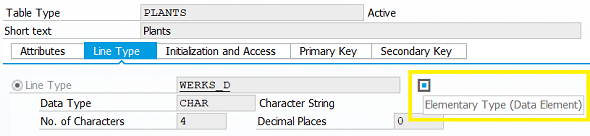
exporting

value(out\_tab) type *plants*

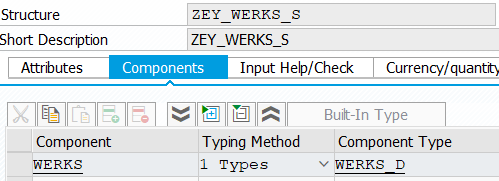
raising

  cx\_amdp\_error.

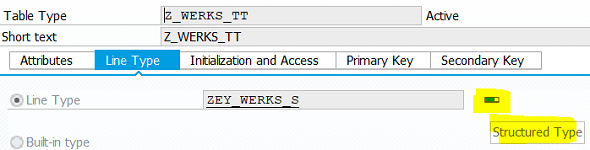
The AMDP activation error is because the *PLANTS* table type is based on an elementary type *WERKS\_D* data element.



But for AMDP output parameters, the row type of the table must be structured not elementary /*структурой, а не отдельными элементами*/. To resolve this issue, I created a structure which includes only the WERKS field of WERKS\_D data element as seen in below.



Additionall I created a new ABAP Data Dictionary object as Table Type using above structure as its row type.



When I modify AMDP class code for the procedure method definition by replacing the *PLANTS* table type with new table type, the AMDP activation is succeeded immediately.

class-methods: ***get\_plant\_by\_salesorg***

AMDP OPTIONS READ-ONLY

CDS SESSION CLIENT CURRENT

importing

value(p\_mandt) type sy-mandt

value(p\_vkorg) type vkorg

exporting

value(out\_tab) type *Z\_WERKS\_TT*

raising

  cx\_amdp\_error.

# Handling of SELECT-OPTIONS parameters within AMDP

# <https://blogs.sap.com/2015/03/30/handling-of-select-options-parameters-within-amdp/>

**Полезные ресурсы по чтению CV из ABAP и прочих мест:**

[ABAP News for Release 7.50 – CDS Table Functions Implemented by AMDP](https://blogs.sap.com/2015/10/21/abap-news-for-release-750-cds-table-functions-implemented-by-amdp/) — как вставить CDS в AMDP метод.

[ABAP CDS Table Functions](https://help.sap.com/doc/saphelp_nw75/7.5.5/en-US/e5/529f75afbc43e7803b30346a56f963/frameset.htm#:~:text=The%20table%20function%20is%20associated,to%20implement%20the%20table%20function.&text=The%20public%20ABAP%20class%20(AMDP,of%20the%20table%20function%20tab_function_example%20.) — что такое CDS TF.

[How to consume HANA Calculation views in S/4HANA CDS views](https://blogs.sap.com/2020/08/26/how-to-consume-hana-calculation-views-in-s-4hana-cds-views/) — как потреблять данные из CV в обычной CDS, почти такой же вариант как и потребление через ABAP, только последний шаг другой.

# Create Dates Table /таблицы дат/ CDS Table Function using SAP HANA AMDP Class

<https://www.kodyaz.com/sap-abap/create-cds-table-function-for-dates-table-with-amdp-class.aspx>

[Содержание](#Содержание)

*Table Function* [*ZGetNDatesStartingFrom*] *ABAP class* [ZCL\_DatesTable]

AMDP function [*GetNDatesStartingFrom*]

*ABAP program* [*ZGetNDatesStartingFrom*]

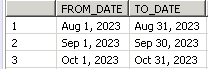
To create a *dates table* /таблицы дат/ on SAP HANA database it’s easy and performs best to use series data function ***SERIES\_GENERATE\_DATE***.

The example below illustrates how to generate a *date series table* with closed-closed semantics where the *period\_end* belongs to the interval

SELECT GENERATED\_PERIOD\_START AS *from\_date*,

*ADD\_DAYS*(GENERATED\_PERIOD\_END, 1) AS *to\_date*

FROM *SERIES\_GENERATE\_DATE*('INTERVAL 1 MONTH', '2023-08-01', '2023-11-07')



Rem – для чего [в коде выше] нужен ***ADD\_DAYS***(GENERATED\_PERIOD\_END, *-1*)

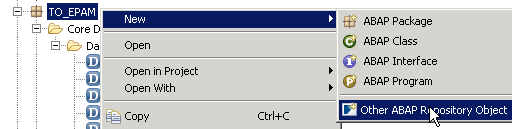
Просто GENERATED\_PERIOD\_END AS to\_date - дает



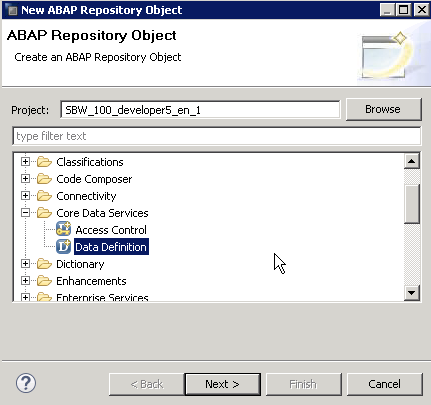
But if you have problems or limitations to consume native HANA database development objects - an alternative way is to create CDS table functions, for example.

1. *GetNDatesStartingFrom*;
2. *GetDatesBetween*.

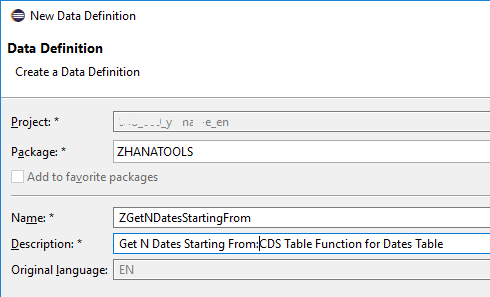
On the ABAP development package in which you will create the CDS table functions, right click on the package and choose New in context menu.



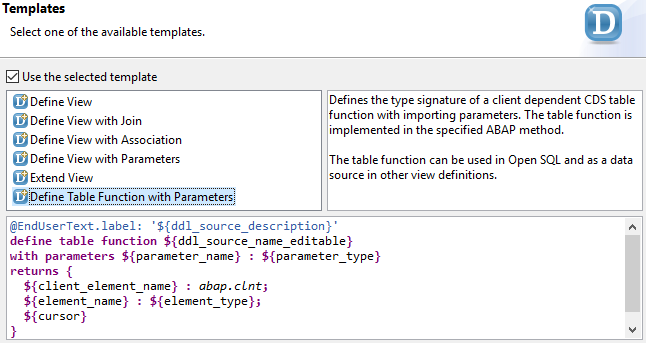
We will create a new *Core Data Services\Data Definition* object



Click Next and on the following screen define a name and description for your first CDS table function



SAP HANA Studio will provide predefined templates. Among the existing templates, choose *Define Table Function with Parameters*



Click Finish button to start coding for our first CDS table function.

I modified the initially displayed template as follows

@ClientDependent: false

@EndUserText.label: *'Get N Dates Starting From'*

define *table function* ***ZGetNDatesStartingFrom***

with parameters

starting\_date : *abap*.*dats*,

number\_of\_dates : *abap*.*int2*

returns {

--client : abap.clnt;

number : *abap*.*int8*;

date : *abap*.*dats*;

}

*implemented by* method zcl\_DatesTable*=>GetNDatesStartingFrom*;

Let's now continue with creating our AMDP class code and method to implement CDS table function code.

On package name, right click and choose *New -> ABAP Class*

Type a meaningful brief *class name* and a *descriptive text* for the AMDP class created for methods which will be consumed by CDS table functions.

class **ZCL\_DatesTable** *definition*

public

final

create public .

public section.

INTERFACES if\_amdp\_marker\_hdb.

*class-methods* *GetNDatesStartingFrom* for table function *ZGetNDatesStartingFrom*.

*class-methods* *GetDatesBetween* for table function *ZGetDatesBetween*.

protected section.

private section.

ENDCLASS.

CLASS **ZCL\_DatesTable** *implementation*.

METHOD ***GetNDatesStartingFrom***

BY DATABASE FUNCTION FOR HDB LANGUAGE SQLSCRIPT.

return

SELECT *element\_number* as number,

*dats\_from\_date*(*generated\_period\_start*) as date

FROM *SERIES\_GENERATE\_DATE*('INTERVAL 1 DAY', :starting\_date,

*add\_days*(:starting\_date, :number\_of\_dates));

ENDMETHOD.

Rem

select \* from *SERIES\_GENERATE\_DATE*('INTERVAL 1 DAY', '2023-12-01', add\_days('2023-12-01', 3))



METHOD ***GetDatesBetween***

BY DATABASE FUNCTION FOR HDB LANGUAGE SQLSCRIPT.

return

SELECT -- '060' as client,

*element\_number* as number, *dats\_from\_date*( *generated\_period\_start* ) as date

FROM *SERIES\_GENERATE\_DATE*('INTERVAL 1 DAY', :starting\_date,

*add\_days*(:end\_date, 1));

ENDMETHOD.

ENDCLASS.

After you save and activate this AMDP Table Function, ABAP programmers are ready to call this AMDP class method or CDS Table Function within an ABAP program.

REPORT ***ZGetNDatesStartingFrom\_test***.

data *lv\_startdate* TYPE dats.

data *lv\_i* TYPE i VALUE 10.

lv\_startdate = sy-datum.

select \*

from *ZGetNDatesStartingFrom*( starting\_date = @lv\_startdate, number\_of\_dates = @lv\_i )

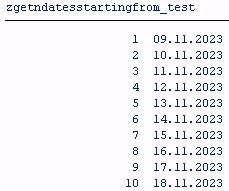
INTO TABLE *@data*(lt\_data).

##db\_feature\_mode[amdp\_table\_function]

LOOP AT lt\_data *REFERENCE* INTO *data*(lr\_data).

WRITE :/ lr\_data->number, lr\_data->date.

ENDLOOP.



Rem

ABAP developers will see a warning indicating that

*The database feature "AMDP\_TABLE\_FUNCTION" is used here (read the long text) on SAP HANA Studio editor.*

To resolve this warning, place *##db\_feature\_mode[amdp\_table\_function]* pragma at the end of the SELECT statement where AMDP Table Function is referenced in FROM clause.

## **AMDP Method Call from Program and Calling an AMDP method from another AMDP method**

# <https://sapcodes.com/2017/12/17/amdp-method-call/>

[Содержание](#Содержание)

# *ABAP class* [ZCL\_FlIGHT\_CALC] *ABAP program* [ZFlight\_Inf]

# *AMDP method* [get\_flight\_detail]

# Here we have a single AMDP method.

class *ZCL\_FLIGHT\_CALC* *definition*

public

final

create public .

public section.

interfaces: *if\_amdp\_marker\_hdb*.

methods: ***get\_flight\_detail*** importing value(*iv\_mandt*) type sy-mandt value(*iv\_carrid*) type sflight-carrid

exporting value(*et\_flight*) type flighttab.

protected section.

private section.

methods: ***get\_flight\_detail\_db*** importing value(*iv\_mandt*) type sy-mandt

value(*iv\_carrid*) type sflight-carrid

exporting value(*et\_flight*) type flighttab.

endclass.

class ZCL\_FLIGHT\_CALC *IMPLEMENTATION*.

method ***get\_flight\_detail*** by database procedure for hdb language sqlscript

using *ZCL\_FLIGHT\_CALC=>GET\_FLIGHT\_DETAIL\_DB*.

call *"ZCL\_FLIGHT\_CALC=>GET\_FLIGHT\_DETAIL\_DB"*( iv\_mandt=>iv\_mandt, iv\_carrid=>iv\_carrid,

et\_flight=>et\_flight );

endmethod.

method ***get\_flight\_detail\_db*** by database procedure for hdb language sqlscript

options read-only

using *sflight*.

et\_flight = select \* from sflight where mandt = iv\_mandt and carrid = iv\_carrid;

endmethod.

# endclass.

# Above we have two AMDP methods with the *public* AMDP method *get\_flight\_detail v3* calling the *private* AMDP method *get\_flight\_detail\_db*.

# The call features of the private AMDP method

# The public AMDP method must include the private AMDP method in the *USING* option .

# It must not a call method but CALL and AMPD method must be called inside *double quotes* and *capital letters*.

# For All Entries Alternatives

<https://blogs.sap.com/2018/12/26/for-all-entries-alternatives/>

See also <https://www.meltingpointathens.com/what-can-i-use-instead-of-all-entries-in-sap-abap/>

[Содержание](#Содержание)

In ***ABAP*** *FOR ALL ENTRIES* statement is used to fetch corresponding data for all the records in the source or result package. The same functionality can be achieved using *LEFT OUTER JOIN* in the ***AMDP*** script.

FOR ALL ENTRIES alternatives

1. [Pushing Internal table to DB by using AMDP](#Pushing_Internal_table_to_DB_by_AMDP) /after release 7.40/
2. [Usage GTT /Global Temporary table/](#Using_GTT) –  /before 7.52/
3. Direct select from internal table – /from 7.52/

**Using Internal table as Data Source From Release 7.52**

[Содержание](#Содержание)

The SELECT statement handles the *internal table* of the application server like a *database table* on the database and *internal table* must be prefixed with@symbolas a data source of a query.

TYPES : BEGIN OF lty\_flt,

carrid TYPE s\_carr\_id,

connid TYPE s\_conn\_id,

END OF lty\_flt.

DATA : *lt\_flt* TYPE STANDARD TABLE OF lty\_flt.

*" Insert record into local internal table*

lt\_flt = *value #*( ( carrid = 'LH' ) ( connid = 2402 ) ) .

SELECT a~carrid, a~connid, SUM( b~price ) AS price, b~currency

FROM *@lt\_flt* AS a

LEFT OUTER JOIN sflight AS b ON a~carrid = b~carrid AND a~connid = b~connid

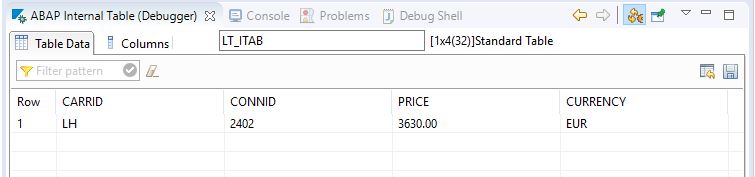
GROUP BY a~carrid, a~connid, b~currency

ORDER BY a~carrid

INTO TABLE *@data*(lt\_itab)

##db\_feature\_mode[itabs\_in\_from\_clause].

*Output*



For more about Internal Table as Data Source

<https://help.sap.com/doc/abapdocu_752_index_htm/7.52/en-US/abapselect_itab.htm>

**Pushing Internal table to DB by using AMDP**

[Содержание](#Содержание)

While developing AMDP class there must be a situation like how to handle *internal table* inside AMDP.

***Scenario***

Suppose you have Airline code /*carrid*/ and connection number /*connid*/ in an internal table based on that you need to fetch data from inside AMDP.

INTERFACES : IF\_AMDP\_MARKER\_HDB.

TYPES :

BEGIN OF *lty\_flight*, *" local type flight*

*carrid* TYPE S\_CARR\_ID,

*connid* TYPE S\_CONN\_ID,

END OF lty\_flight,

BEGIN OF *lty\_flight\_dtl*,

*carrid* TYPE S\_CARR\_ID,

*connid* TYPE S\_CONN\_ID,

*fldate* TYPE S\_DATE,

*price* TYPE S\_PRICE,

*currency* TYPE S\_CURRCODE,

*planetype* TYPE S\_PLANETYE,

END OF lty\_flight\_dtl,

*lt\_sflight* TYPE STANDARD TABLE OF *lty\_flight* WITH EMPTY KEY,

*lt\_sflight\_dtl* TYPE STANDARD TABLE OF *lty\_flight\_dtl* WITH EMPTY KEY.

CLASS-METHOOS : ***get\_flight\_details***

IMPORTING VALUE(*it\_sflight*) TYPE *lt\_sflight*

EXPORTING VALUE(*et\_sflight\_dtl*) TYPE *lt\_sflight\_dtl*.

METHOD ***get\_flight\_details***

BY DATABASE PROCEDURE FOR HDB

LANGUAGE SQLSCRIPT

OPTIONS READ-ONLY USING *sflight*.

*-- По carrid и connid подтягиваем остальные поля - fldate | price | currency | planetype*

*et\_sflight\_dtl* = SELECT A.CARRID, A.CONNID, B.FLDATE, B.PRICE, B.CURRENCY, B.PLANETYPE

FROM *:it\_sflight* AS A *'' :it\_sflight – вх. параметр*

LEFT OUTER JOIN *sflight* AS В

ON A.CARRID = B.CARRID AND A.CONNID = B.CONNID

ORDER BY CARRID;

ENDMETHOD.

Calling AMDP in abap program

*'' Заполняем internal table*

*it\_sflight* = value #( ( CARRID = 'LH' CONNID = '2402' ) ( CARRID = 'SQ' CONNID = '26' ) ).

TRY.

CALL METHOD ZCL\_AMDP\_TEST =>*get\_flight\_details*

EXPORTING *it\_sflight* = *lt\_sflight*

IMPORTING *et\_sflight\_dtl* = *data*(g*t\_sflight\_dtl*).

CATCH CX\_AMDP\_ERROR INTO AMDP\_ERROR.

ENDTRY.

REPORT **Z\_Get\_Flight\_Inf**.

TYPES :

BEGIN OF lty\_flight, " local type flight

carrid TYPE S\_CARR\_ID,

connid TYPE S\_CONN\_ID,

END OF lty\_flight,

BEGIN OF lty\_flight\_details, " local type flight\_details

carrid TYPE S\_CARR\_ID,

connid TYPE S\_CONN\_ID,

fldate TYPE S\_DATE,

price TYPE S\_PRICE,

currency TYPE S\_CURRCODE,

planetype TYPE S\_PLANETYE,

END OF lty\_flight\_details,

lt\_sflight TYPE STANDARD TABLE OF lty\_flight WITH EMPTY KEY, " local table sflight

lt\_sflight\_details TYPE STANDARD TABLE OF lty\_flight\_details WITH EMPTY KEY. " local table flight\_details

data: obj type ref to ZCL\_FLIGHT\_INF,

it\_sflight type lt\_sflight.

*" Заполняем internal table*

it\_sflight = *value #*( ( CARRID = 'LH' CONNID = '2402' ) ( CARRID = 'LH' CONNID = '0455' ) ( CARRID = 'SQ' CONNID = '0026' ) ).

obj = new ZCL\_FLIGHT\_INF( ).

TRY

CALL METHOD obj->*get\_flight\_detail\_after\_7\_40*

EXPORTING it\_sflight = it\_sflight

IMPORTING et\_sflight\_details = data(gt\_sflight\_dtl).

cl\_demo\_output=>*display*( *gt\_sflight\_dtl* ).

CATCH CX\_AMDP\_ERROR INTO *data*(AMDP\_ERROR).

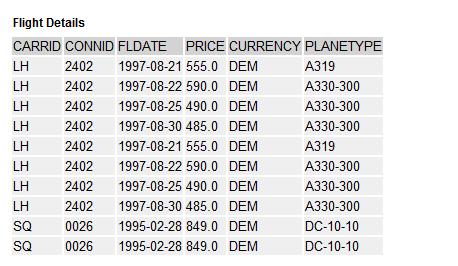
ENDTRY.

Rem

VALUE *type*( ) - creates a result of a data type specified using *type*.

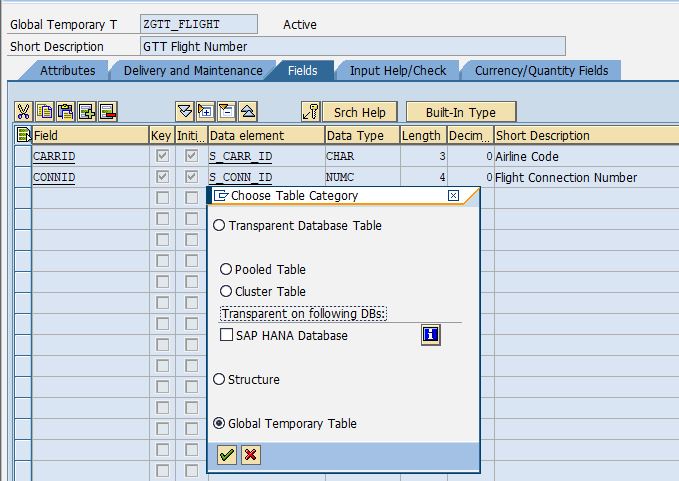
VALUE *#*( ) - if no parameters are specified in the parentheses the return value is set to its type-specific *initial value*. This is possible for any non-generic data types *dtype*.

Output



**Using GTT**

Create a GTT to hold Airline code and Flight Connection Number



After Creation of GTT table and insertion of records into GTT we can write direct query.

*"Insert record into Global Temporary Table*

INSERT *zgtt\_flight* FROM *@*(*value #*( carrid = 'LH' connid = 2402 )).

SELECT a~carrid, a~connid, SUM( b~price ) AS price, b~currency

FROM *zgtt\_flight* AS a

LEFT OUTER JOIN sflight AS b

ON a~carrid = b~carrid AND a~connid = b~connid

GROUP BY a~carrid, a~connid, b~currency

ORDER BY a~carrid

INTO TABLE @data*(lt\_itab)*.

IF sy-subrc = 0.

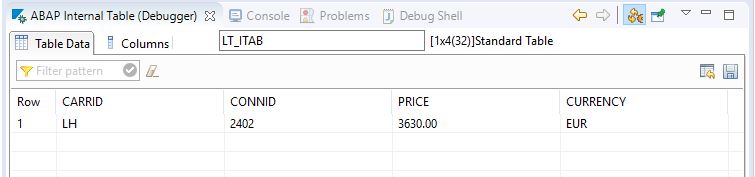
DELETE FROM *zgtt\_flight*.

ENDIF.

Rem

The GTT concept specifies that a *GTT is* always *empty at the start of a database LUW*  ⇒ always *has to be cleared at the end of each database* LUW otherwise will give dump Runtime Errors *COMMIT\_GTT\_ERROR*.

Output



For more about GTT

<https://help.sap.com/doc/abapdocu_751_index_htm/7.51/en-US/abenddic_database_tables_gtt.htm>

# Internal Tables

[Содержание](#Содержание)

# <https://itpfed.com/new-features-in-abap-7-4-internal-tables/>

# Добавление записей в *таблицу БД*

INSERT <dbtab> FROM <wa>.

INSERT <dbtab> FROM TABLE <itab> [ACCEPTING DUPLICATE KEYS].

# Добавление записей в *internal table*

INSERT <line> | LINES OF <jtab> [FROM <n1>] [TO <n 2>]

INTO TABLE <itab>

[ASSIGNING <FS> | REFERENCE INTO <dref>].

*Rem*

1. FROM and TO addition will specify the lines to add.
2. ASSIGNING and REFERENCE INTO addition is used for assigning the inserted record into a field symbol.

# data:

# *itab* type *hashed table* of *spfli* with unique key carrid connid,

# *wa* like line of itab.

# wa-carrid = ‘UA’. wa-connid = ‘0011’. wa-cityfrom = …

# insert *wa* into table *itab*.

# insert *spfli* from table *itab* accepting duplicate keys.

# if sy-subrc = 0.

# …

# elseif sy-subrc = 4.

# …

# endif.

# *Secondary keys*

# From ABAP 7.2 we can declare *secondary keys* for internal tables - secondary key could increases *read access performance*. But, on the other hand, secondary keys also incur additional administration costs due to memory consumption and run-time.

Lets create a secondary index into the internal table *IT\_MARA* for the column *BISMT*

DATA: IT\_MARA TYPE *HASHED* TABLE OF mara

WITH UNIQUE KEY matnr

WITH *NON-UNIQUE SORTED KEY* sort\_key *COMPONENTS* bismt.

READ TABLE it\_mara INTO wa\_mara WITH KEY *sort\_key* COMPONENTS *bismt* = lv\_bismt.

Even though IT\_MARA is a HASHED table it’s also a SORTED table with the key BISMT ⇒ when we go looking for the record using BISMT a *BINARY SEARCH* is automatically performed.

## [**ABAP 7.4**](http://twitter.com/share?url=https%3A%2F%2Fitpfed.com%2Fnew-features-in-abap-7-4-internal-tables%2F&text=Declaring+Table+Work+Areas+in+ABAP+7.4)

You no longer need to do a *DATA* declaration *for elementary data types* – it’s exactly the same for the work areas

READ TABLE lt\_mara WITH KEY matnr = lv\_matnr INTO *DATA(ls\_mara)*.

LOOP AT lt\_mara INTO *DATA(ls\_mara)*.

You no longer need *FIELD-SYMBOL* declarations for the situations in which you want *to change the data in the work area* while looping through an internal table. *If you want to use field symbols for the work area* then the syntax is shown below

READ TABLE lt\_mara WITH KEY matnr = lv\_matnr ASSIGNING *FIELD-SYMBOL()*.

LOOP AT lt\_mara ASSIGNING *FIELD-SYMBOL()*.

***The syntax for using a table expression*** consists of an internal table followed by a row specified in square brackets [ ]

Prior to ABAP 7.4

READ TABLE flight\_schedules INTO *DATA*(flight\_schedule)

WITH KEY carrid = 'AA' connid = '0017'.

lo\_mara = zcl\_mara\_factory( ls\_mara-matnr ).

ABAP 7.4

*\* The result of a table expression is a single table line*

DATA(flight\_schedule) = flight\_schedules*[ carrid = 'AA' connid = '0017' ]*.

lo\_mara = zcl\_mara\_factory( lt\_mara*[ matnr = lv\_matnr ]*-matnr ).

If a table line is not found - the exception CX\_SY\_ITAB\_LINE\_NOT\_FOUND is raised - one way around this is to use the built-in table functions ***line\_exists***()

IF *line\_exists*( vendors*[ id = '00AED' ]* ).

vendors*[ id = '00AED' ]*.

ENDIF.

From 7.40 we are able to define default values for avoiding the exception CX\_SY\_ITAB\_LINE\_NOT\_FOUND

DATA(default\_customer) = VALUE customer( id = '00000' name = 'not found' ... ).

DATA(lv\_customer) = *VALUE #*( customers*[ id = '00024' ] DEFAULT* default\_customer ).

# Classes

[Содержание](#Содержание)

CLASS xx *Definition.* … ENDCLASS.

CLASS xx *Implementation*. … ENDCLASS.

A ***static attribute*** is declared by using the *CLASS-DATA* statement and are accessed directly with the help of class name like class\_name=>name\_1 = 'Some Text'.

*CLASS-DATA*: cv\_name1 Type char45, cv\_data1 Type I.

METHOD <m\_name>.

…

ENDMETHOD.

You can ***call a method*** by using the *CALL METHOD* statement.

***Class methods*** are called using **=>** ; ***instance methods*** are called using **->**.

REPORT z\_invoice\_items\_euro.

class **lcl\_main** definition *create private*. *“ create private – локальный класс*

public section.

*class-methods* ***create\_1*** returning value(r\_result) type ref to lcl\_main.

methods ***run***.

protected section.

private section.

endclass.

class **lcl\_main** implementation.

method ***create\_1***.

create object r\_result.

endmethod.

method ***run***.

write / 'Hello World'.

endmethod.

endclass.

start-of-selection.

lcl\_main=>create\_1( )->run( ).

***Scope of the variables***

**PUBLIC** Section.

…

**PROTECTED** Section.

…

**PRIVATE** Section.

By default all the members of a class would be *private*.

## ***ME Operator in Methods***

A variable can be declared with an initial value in public section. We may declare the variable again inside a method with a different value. When we write the variable inside the method the system will print the changed value. To reflect the *previous value of the variable* we have to use ***ME*** operator.

ME – аналог *this*.

# *Inheritance*

CLASS <subclass> DEFINITION *INHERITING FROM* <superclass>.

Within the redefined method you can *access components of the direct super class* using the **super** reference.

# *Polymorphism*

ABAP **polymorphism** means that *a call to a method will cause a different method* to be executed depending on the type of object that invokes the method.

CLASS **xx** Definition *Abstract*.

PUBLIC Section.

Methods: ***m1*** *Abstract*.

ENDCLASS.

CLASS **xx\_1** Definition *Inheriting From* xx.

PUBLIC Section.

Methods: ***m1*** *Redefinition*.

# ENDCLASS.

# *Encapsulation*

*Interface* **intf\_1**.

Data text1 Type char35.

Methods ***m1***.

EndInterface.

Class **xx\_1** Definition.

PUBLIC Section.

*Interfaces* intf\_1.

EndClass.

Class **xx\_1** Implementation.

Method ***intf\_1~m1***.

*intf\_1~text1* = 'Class 1 Interface method'.

Write / intf\_1~text1.

EndMethod.

EndClass.

Start-Of-Selection.

Data obj Type Ref To xx\_1.

Create Object obj.

CALL Method *obj->intf\_1~method1*.

# *Interfaces*

The components defined in the declaration of an interface are always integrated in the public visibility section of the classes ⇒ *the declaration of an interface doesn’t include the visibility sections*.

*Iinerface* <intf\_name>.

Data …

Class-Data …

Methods …

Class-Methods …

EndInterface.

The following syntax is used to implement the methods of an interface inside the implementation of a class

Method *<intf\_name~method\_m*>.

…

EndMethod.

*Пример*

Interface **zz**.

Methods ***msg***.

EndInterface.

Class **xx** Definition.

PUBLIC Section.

*Interfaces* zz.

Methods: ***m1***, ***m2***.

…

EndClass.

Class **xx** Implementation.

Method ***zz~msg***. … EndMethod.

…

EndClass.

# Object Events

[Содержание](#Содержание)

For an event *the event handler*method can be defined in the same or different class by using the ***FOR EVENT*** clause

*FOR EVENT* <event\_name> OF <class\_name>.

*An event* can have parameter interface but it has only *output parameters*. The *output parameters* are passed to the event handler method by the ***RAISE EVENT*** statement that receives them as input parameters. An *event* is linked to its handler method dynamically in a program by using the ***SET HANDLER*** statement.

When an event is triggered - appropriate event handler methods are supposed to be executed in all the handling classes.

CLASS **cl\_main** DEFINITION.

PUBLIC SECTION.

DATA: num1 TYPE I.

METHODS: ***pro*** importing num2 TYPE i.

*EVENTS*: CUTOFF.

CLASS.

CLASS **cl\_eventhandler** DEFINITION.

PUBLIC SECTION.

METHODS: ***handling\_CUTOFF*** *for event* CUTOFF OF cl\_main.

CLASS.

START-OF-SELECTION.

DATA: main1 TYPE REF TO cl\_main.

DATA: eventhandler1 TYPE REF TO cl\_eventhandler.

CREATE OBJECT main1.

CREATE OBJECT eventhandler1.

*set handler* eventhandler1->handling\_CUTOFF *for* main1.

main1->pro( 4 *).*

CLASS **cl\_main** IMPLEMENTATION.

METHOD ***pro***.

num1 = num2.

IF num2 >= 2. *raise event CUTOFF*. ENDIF.

ENDMETHOD.

CLASS.

CLASS **cl\_eventhandler** IMPLEMENTATION.

METHOD ***handling\_CUTOFF***.

WRITE: 'Handling the CutOff'.

WRITE: / 'Event has been processed'.

ENDMETHOD.

ENDCLASS.

# Report Programming

[Содержание](#Содержание)

A classical report is created by using the output data in the WRITE statement inside a loop. They don’t contain any sub-reports. SAP also provides some standard reports –

* RSCLTCOP – to copy tables across clients;
* RSPARAM - to display instance parameters;
* ..

# CL\_DEMO\_OUTPUT

[Содержание](#Содержание)

# CL\_DEMO\_OUTPUT, Part 1 of 2 – Usage

# <https://blogs.sap.com/2016/05/10/cldemooutput-part-1-of-2-usage/>

# Introducing CL\_DEMO\_OUTPUT

# <https://medium.com/@abapcoder/introducing-cl-demo-output-34943ddda21a>

# EXEC SQL

# <http://www.sapnet.ru/abap_docu/ABAPEXEC.htm>

Using the **WRITE** statements to review code progression is not good practice to begin with. In a dialog session you may never see the output from WRITE statements unless you use the LEAVE LIST-PROCESSING statement.

***The CL\_DEMO\_OUTPUT class***

1. Cannot parse a data object and turn it into XML - that is the domain of the **CALL TRANSFORMATION** statement to generate XML and JSON output.

Rem

You could manually set up the XML output by using the various WRITE methods defined in the CL\_DEMO\_OUTPUT\_STREAM class.

1. Is **not** intended for use in production programs - it replace standard WRITE statements with a more user friendly framework.
2. Cannot parse complex data objects - such as class instances for output - you must convert the data to XML or JSON first.

**Depricated версия**

***Использование неявного курсора - EXEC SQL PERFORMING***

REPORT **znative\_sql\_perform**.

TYPES: BEGIN OF ty\_gds,

emit TYPE /bic/abdgds0002-/bic/bemit000,

gds\_name TYPE /bic/tbcom0000-txtlg,

gds\_price TYPE /bic/abdgds0002-/bic/bsprcac0,

price\_curr type /bic/abdgds0002-currency,

END OF ty\_gds.

DATA: wa\_gds TYPE ty\_gds-gds\_name.

PARAMETERS p\_emit TYPE ty\_gds-emit.

EXEC SQL PERFORMING evaluate.

SELECT top 1000 t\_gds.txtlg INTO :wa\_gds

FROM "/BIC/ABDGDS0002" as gds left outer join "/BIC/TBCOM0000" as t\_gds

on t\_gds."/BIC/BCOM0000" = gds."/BIC/BCOM0000"

WHERE gds."/BIC/BEMIT000" = :p\_emit

*\*Rem*

*\* After the ENDEXEC statement* ***sy-dbcnt*** *contains the total number of rows are readed.*

ENDEXEC.

cl\_demo\_output=>begin\_section( 'Native SQL Example' ).

cl\_demo\_output=>display( ).

FORM evaluate.

cl\_demo\_output=>write( | { wa\_gds } | ).

ENDFORM.

***Использование явного курсора - EXEC SQL OPEN CUR FOR***

REPORT **znative\_sql\_fetch**.

…

EXEC SQL.

OPEN CUR FOR

SELECT top 1000 t\_gds.txtlg

FROM "/BIC/ABDGDS0002" as gds

left outer join "/BIC/TBCOM0000" as t\_gds on t\_gds."/BIC/BCOM0000" = gds."/BIC/BCOM0000"

WHERE gds."/BIC/BEMIT000" = :p\_emit

ENDEXEC.

DO.

EXEC SQL.

FETCH NEXT CUR INTO :wa\_gds

ENDEXEC.

IF sy-subrc <> 0.

EXIT.

ENDIF.

cl\_demo\_output=>write( |{ wa\_gds } | ).

ENDDO.

cl\_demo\_output=>display( ).

EXEC SQL xx cannot be a table or a reference and cannot contain either of these objects or strings.

# TRY and RETRY, CATCH and RESUME

# <https://blogs.sap.com/2016/07/07/try-and-retry-catch-and-resume/>

[Содержание](#Содержание)

|  |
| --- |
| **Event & Description**  [Содержание](#Содержание) |
| **INITIALIZATON** - triggered before displaying the selection screen. |
| **AT SELECTION-SCREEN** - triggered after processing of the user input on the selection screen. This event verifies the user input prior to the execution of a program. After processing the user input, the selection screen remains in the active mode. |
| **START-OF-SELECTION** - triggered only after the processing of the selection screen is over; that is, when the user clicks the ***Execute icon*** on the selection screen. |
| **END-OF-SELECTION** - triggered after the last statement in the START-OF-SELECTON event is executed. |
| **TOP-OF-PAGE** - triggered by the first WRITE statement to display the data on a new page. |
| **END-OF-PAGE** - triggered to display the text at the end of a page in a report. Note, that this event is the last event while creating a report, and should be combined with the ***LINE-COUNT*** clause of the REPORT statement. |

REPORT **ZREPORT2** LINE-SIZE 75 LINE-COUNT 30(3) NO STANDARD PAGE HEADING.

Tables: MARA.

TYPES: Begin of itab,

MATNR TYPE MARA-MATNR,

MBRSH TYPE MARA-MBRSH,

MEINS TYPE MARA-MEINS,

MTART TYPE MARA-MTART,

End of itab.

DATA: wa\_ma TYPE itab,

it\_ma TYPE STANDARD TABLE OF itab.

SELECT-OPTIONS: MATS FOR MARA-MATNR *OBLIGATORY*.

INITIALIZATION.

MATS-LOW = '1'.

MATS-HIGH = '500'.

*APPEND MATS.*

AT SELECTION-SCREEN. *“after processing of the user input on the selection screen*

IF MATS-LOW = ' '. MESSAGE I000(ZKMESSAGE).

ELSEIF MATS-HIGH = ' '. MESSAGE I001(ZKMESSAGE).

ENDIF.

TOP-OF-PAGE. *“triggered by the first WRITE statement*

WRITE:/ 'CLASSICAL REPORT CONTAINING GENERAL MATERIAL DATA

FROM THE TABLE MARA' COLOR 7.

ULINE.

WRITE:/ 'MATERIAL' COLOR 1,

*24* 'INDUSTRY' COLOR 2,

*38* 'UNITS' COLOR 3,

*53* 'MATERIAL TYPE' COLOR 4.

ULINE.

END-OF-PAGE.

START-OF-SELECTION. *“after the processing of the selection screen is over*

SELECT MATNR MBRSH MEINS MTART

FROM MARA

INTO TABLE it\_ma WHERE MATNR IN MATS.

LOOP AT it\_ma into wa\_ma.

WRITE:/ wa\_ma-MATNR,

25 wa\_ma-MBRSH,

40 wa\_ma-MEINS,

55 wa\_ma-MTART.

ENDLOOP.

END-OF-SELECTION. *“after the last statement in the START-OF-SELECTON event is executed*

ULINE.

WRITE:/ 'CLASSICAL REPORT HAS BEEN CREATED' COLOR 7.

ULINE.

SKIP.

# Dialog Programming

[Содержание](#Содержание)

## Dialog programs should be developed by the object browser - SE80

## **Screens** are made up of screen attributes & screen layout & fields and flow logic.

## The **module pool** consists of modularized syntax that is placed inside include programs of the dialog program. These modules can be invoked by the flow logic, which is processed by the dialog processor.

## The Toolset

* **se80** – Object Browser
* **se51** – Screen Painter
* **se38** - ABAP/4 Editor
* **se11** - ABAP/4 Dictionary
* **se41** – Menu Painter
* **se93** – Maintain Transaction

**To add a screen to the program**, /being in se80/ right-click on the program name and select the options ***Create → Screen***.

**To create a transaction code** for your program, simply right click on the program name and choose the option ***Create → Transaction*** and enter a transaction code.

# Параметры в ABAP программе

[Содержание](#Содержание)

# *Два способа*

# 1.

# data p\_emit type zcds\_gds-/bic/bemit000.

# …

# cl\_demo\_input=>new( )->add\_field( CHANGING field = p\_emit )->request( ).

# Вызов программы из другой с передачей параметров

[Содержание](#Содержание)

Когда нужно сделат предварительную выборку данных перед запуском саповских программ и не хочется ломать стандарт есть выход. Отрисовываем аналогичный селект-скрин → делаем с данными все что нужно → вызываем стандартную программу с передачей данных в нее.

SUBMIT zfu\_opm\_mass\_rel

        with bearbver ...

        with pa\_lfrty ...

        and return.

# Добавка в конце AND RETURN позволит вернуться на программу с которой происходил вызов. Если же оператор не указан - мы останемся в стандартной.

# SAP scripts

The SAPscript tool of the SAP system can be used to build and manage business forms such as invoices and purchase orders.

The **Form Painter tool – se71** provides the graphical layout of a SAPscript form and various functionalities to manipulate the form.

# Customer Exits

# Web Dynpro

Web Dynpro - WD for ABAP can be used in the development of web-based applications in the SAP ABAP environment.

# ABAP on SAP HANA. Part I. First Program in ABAP HANA

# <https://sapyard.com/abap-on-sap-hana-part-i/>

**ADT - ABAP Development Tool**.

It is easy to create a testing Environment with a hosted [*Citrix XenDesktop*](https://www.clouddesktoponline.com/citrix-xen-desktop/) from CloudDesktopOnline.com. You can also go for Integrated SharePoint with SAP HANA with the help of SharePoint experts from [Apps4Rent](http://www.apps4rent.com/).

ADT is more powerful than SE80 - some advanced features like

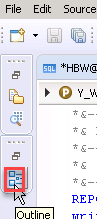
* creating external views for exposing HANA view to ABAP DDIC\*;
* creating Database proxy procedures\* -

are available only when using ADT.

**ABAP project** helps to connect the Eclipse base IDE to ABAP backend system. The project provides eclipse based frameworks for creating | processing and testing development objects –



***Pattern Icon -*** Type initial letters of the syntax you want to use and then use ***Ctrl + Space*** and ***Shift + Enter*** to insert the full signature (e.g. for function module / method selected).

The Outline view ***-***  ***displays the internal structure of a program or class*** that is currently open in the ABAP source code editor.

***Keyword Completion/Suggestion - Ctrl +Space***

***Rename elements/texts.***

Although we have Find and Replace (or Ctrl + H) option in SAP GUI, but Eclipsed based ADT has better renaming experience. Just right click on the source editor and select ***Rename*** or hit ***Alt+Shift+R***, to open the replace wizard.

**INTEGRATING AN AMDP INTO YOUR ABAP CODE**

<https://itpfed.com/want-to-avoid-hana-amdp-mistakes-read-this-2/>

[Содержание](#Содержание)

**CDS View**

[Содержание](#Содержание)

**There are two components of CDS Views in HANA**

1. **DDL SQL View** - read-only classical database view which is visible in ABAP Dictionary SE11.
2. **CDS View Entity** - the DDL Source File and the actual CDS View. It makes attributes possible - such as authorization checks.

# DDL SQL is automatically deleted when the CDS View is deleted.

# It’s a good idea to *separate SQL View Name and actual CDS View Name*. For consistency we name

# SQL View Name with DDLS

# *and*

# **CDS View with **CDSV****.

# *Аннотации*

# @AccessControl.authorizationCheck #NOT\_REQUIRED | #CHECK

# @ClientDependent: true | false.

# @BufferStatus:on | off.

# @BufferType:single | generic | fully.

# Annotation @AbapCatalog.sqlViewName is mandatory for the definition of a CDS view.

# Only the CDS View entity is saved in the transport.

# ****Join in CDS View****

## ***Association with cardinality - [min..max]***

***The rules for min and max values***

* *max* cannot be 0;
* an asterisk \* for *max* means any number of rows - association [1..\*] to … ;
* *min* can be omitted - set to 0 if omitted;
* *min* cannot be \*;
* when an association is used in a WHERE condition - 1 must be specified for *max*.

The association defined in an ABAP CDS view will be converted to join type at run-time. By default the join type is **LEFT OUTER JOIN**.

To achieve **INNER JOIN** you need to define the attribute in the path expression

@AbapCatalog.sqlViewName: 'ZV\_ORD\_HDR'

@AbapCatalog.compiler.compareFilter: true

@AccessControl.authorizationCheck: #NOT\_REQUIRED

@EndUserText.label: 'Order Header Information'

define view ZCDS\_SALESORDER\_HDR

as select from vbak

association [1..\*] to ZCDS\_SALESORDER\_ITM as \_OrderItems

on $projection.vbeln = \_OrderItems.vbeln

{

key vbeln,

erdat,

vbtyp,

auart,

netwr,

waerk,

\_OrderItems[inner].matnr

}

<https://www.saplearners.com/create-associations-abap-cds-views/>

## **Usage of CDS view with association in ABAP using OpenSQL**

We can access the CDS view which has associations defined in ABAP program using Open SQL statement using the path expressions like below

DATA(is\_supported) = cl\_abap\_dbfeatures=>use\_features(

requested\_features = VALUE #( ( cl\_abap\_dbfeatures=>views\_with\_parameters ) ) ).

IF is\_supported IS NOT INITIAL.

SELECT \* FROM zcds\_salesorder\_hdr INTO TABLE @DATA(lt\_data).

ENDIF.

Want to learn about new ABAP syntax - [click here](http://www.saplearners.com/inline-declarations-abap-7-4/)

DATA(is\_supported) = cl\_abap\_dbfeatures=>use\_features(

requested\_features = VALUE #( ( cl\_abap\_dbfeatures=>views\_with\_parameters ) ) ).

IF is\_supported IS NOT INITIAL.

SELECT vbeln, erdat,

\\_orderitems-matnr AS material

FROM zcds\_salesorder\_hdr INTO TABLE @DATA(lt\_data).

ENDIF.

If we consume **DDL SQL View** in ABAP SELECT statement - we would not see the performance improvement. Theoretically when the DDL SQL View is used - a database connection from ABAP Layer to Database Layer is established and this process would consume some resources for database connection.

Consumption of **CDS View** by DDL Source name invokes Database Object which is residing at Database Layer ⇒ we can execute an SQL without creating a database connection between ABAP Layer and Database. Only results will be transferred back to ABAP layer.

# Huawei Y6s\_3Gb\_ 6,09”\_средняя\_ камера\_надежный

# Huawei P20 Lite\_4Gb\_5,84”

# Xiaomi Redmi 6 3Gb\_3Gb\_5,45”\_отличная камера\_нет быстрой\_зарядки

# Huawei Y6s или Xiaomi Redmi 6

**Обращение из ABAP к CDS view**

DATA: i\_wo\_gds TYPE STANDARD TABLE OF zcds\_gds.

…

PARAMETERS: p\_emit TYPE I.

…

SELECT \* FROM zcds\_gds( p\_emit = @p\_emit ) INTO TABLE @i\_wo\_gds.

Пример

@AbapCatalog.sqlViewName: 'ZDDL\_GDS'

@AbapCatalog.compiler.compareFilter: true

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: 'zcds\_gds'

@ClientDependent: false

define view **zcds\_gds**

with parameters p\_emit: *abap*.*numc*( 4 )

as

select from /bic/abdgds0002 as gds

*-- По умолчанию кардинальность - [0..1]*

*--association [1..1] to /bic/tbcom0000 as gds\_t*

left outer join /bic/tbcom0000 as gds\_t

*--on $projection./bic/bcom0000 = gds./bic/bcom0000*

on gds\_t./bic/bcom0000 = gds./bic/bcom0000

*--association [1..1] to /bic/tbnom0000 as nom\_t*

left outer join /bic/tbnom0000 as nom\_t

on nom\_t./bic/bnom0000 = gds./bic/bnom0000

{

key gds./bic/bcom0000,

gds\_t.txtlg,

*-- gds./bic/bemit000,*

gds./bic/bnom0000,

*--gds\_t, // Ассоциация gds\_t будет доступна извне*

*--nom\_t, // Ассоциация nom\_t будет доступна извне*

nom\_t.txtlg as nom\_txtlg,

gds./bic/bsprcpvd,

gds./bic/bsvat000,

gds./bic/bsexc000,

gds./bic/bsprcac0,

gds.currency,

-- currency\_conversion(

-- amount=> gds./bic/bsprcac0,

-- source\_currency => gds.currency,

-- target\_currency => cast('USD' as abap.cuky), --тип CUKY используется для source\_currency | target\_currency

-- exchange\_rate\_date => cast('20200730' as abap.dats),

-- client => cast('100' as abap.CLNT)

-- )

-- as AccountPriceInUSD,

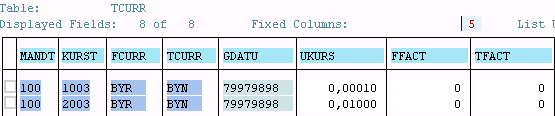
:p\_emit as Emitent,

$session.user as User\_

}

where gds./bic/bemit000 = $parameters.p\_emit

Таблица **TCURR** - курсы валют –



В ABAP 7.5 добавилась **возможность использования переменных сессии** - например ***$session.user*** будет содержать внутри себя sy-uname.

SQL запрос к CDS View с параметром –

SELECT \* FROM "SAPABAP1".ZDDL\_GDS( p\_emit = '020' ) INTO TABLE @DATA(lt\_data)

При выборке данных с передачей параметров в ABAP 7.4. будет показано предупреждение о том что нужно **проверить поддерживает ли СУБД передачу параметров в CDS View**.

Можно спрятать предупреждение и выполнить проверку

IF cl\_abap\_dbfeatures=>use\_features( VALUE #( ( cl\_abap\_dbfeatures=>views\_with\_parameters ) ) ).

  SELECT \*

FROM zddl\_assoc\_cds( p\_carrid = 'UA' ) ##DB\_FEATURE\_MODE[VIEWS\_WITH\_PARAMETERS]

    INTO TABLE @DATA(lt\_data).

ENDIF.

Если СУБД не поддерживает передачу параметров - при выполнении SQL запроса будет вызвано исключение CX\_SY\_SQL\_UNSUPPORTED\_FEATURE. Напомню что в ABAP 7.5 параметры стали доступны для всех СУБД.

Пример – *вывод структуры и содержимого CDS*

REPORT **zcds\_client\_handling**.

CLASS **demo** DEFINITION.

PUBLIC SECTION.

CLASS-METHODS ***main***.

ENDCLASS.

CLASS **demo** IMPLEMENTATION.

METHOD ***main***.

data p\_emit type zcds\_gds-/bic/bemit000.

DATA(out) = cl\_demo\_output=>new( ).

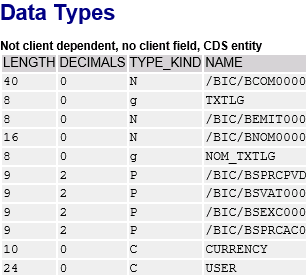
out->next\_section( 'Data Types' ).

out->write(

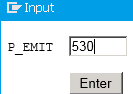
name = 'Not client dependent, no client field, CDS entity'

data = CAST cl\_abap\_structdescr( cl\_abap\_typedescr=>describe\_by\_name( 'ZCDS\_GDS' ) )

->components ).



cl\_demo\_input=>new( )->add\_field( CHANGING field = p\_emit )->request( ).



out->next\_section( 'SELECT Statements' ).

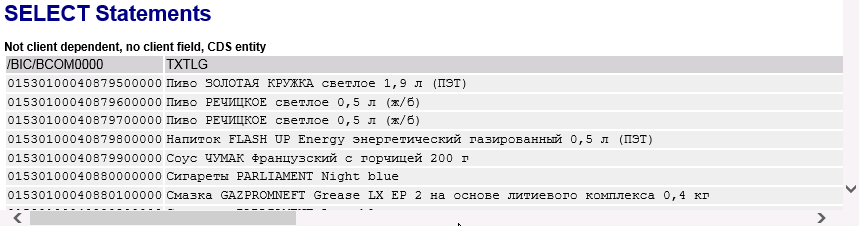
SELECT \* FROM zcds\_gds( p\_emit = @p\_emit ) INTO TABLE @DATA(result1) UP TO 1000 ROWS.

out->write(

name = 'Not client dependent, no client field, CDS entity'

data = result1 ).

out->display( ).



ENDMETHOD.

ENDCLASS.

START-OF-SELECTION.

demo=>main( ).

**IDA ALV**

# SAP List Viewer with Integrated Data Access (ALV with IDA)

<https://blogs.sap.com/2018/07/16/sap-list-viewer-with-integrated-data-access-alv-with-ida/>

# Dynamic Programming in ABAP

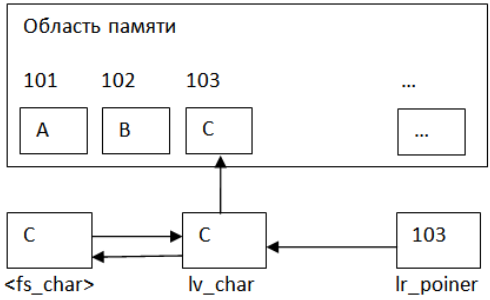
[Содержание](#Содержание)

# Dynamic Programming in ABAP – Part 1 – Introduction to Field Symbols

# <https://blogs.sap.com/2017/09/05/dynamic-programming-in-abap-part-1-introduction-to-field-symbols/>

# <https://abap-blog.ru/osnovy-abap/dinamicheskoe-programmirovanie-v-abap/>

**Field symbol** is a placeholder for data object which points to the value present at the memory address of a data object ⇒ не является указателем на область памяти – это алиас/синоним переменной.



***Field symbols are of two types***

1. **typed** - FIELD-SYMBOLS: <fs\_num> *TYPE i*.
2. **generic** – *any* | *any table***.**

Символы **<>** являются частью синтаксиса и обязательны при объявлении

FIELD-SYMBOLS <fs\_xx> TYPE CSEQUENCE.

FIELD-SYMBOLS: <fs\_num> TYPE i.

По умолчанию для field-symbols нет присвоения к какой-либо переменной ⇒ если мы обратимся к не присвоенному field-symbols - программа упадет в дамп c динамической не обрабатываемой ошибкой GETWA\_NOT\_ASSIGNED. Но - если объявить field-symbol глобально без указания типа - тип будет определен как символьный - а полю будет присвоена константа **space**

REPORT  ZTEST.

FIELD-SYMBOLS: <gv\_trick>.

Проверить присвоение можно с помощью операторов **IS ASSIGNED** | **IS NOT ASSIGNED**

FIELD-SYMBOLS <gfs\_any\_string> TYPE CSEQUENCE.

DATA: gv\_string TYPE string VALUE 'TEST'.

*\** ***Cтатическое присвоение*** *- т.е. мы знаем на какую переменную мы хотим получить ссылку.*

ASSIGN gv\_string TO <gfs\_any\_string>.

IF <gfs\_any\_string> IS ASSIGNED.

  …

ELSE. … ENDIF.

Оператор ASSIGN позволяет использовать ***динамическое присвоение***

PARAMETERS: p\_vari TYPE string.

START-OF-SELECTION.

  FIELD-SYMBOLS <gfs\_any\_string> TYPE CSEQUENCE.

  ASSIGN (p\_vari) TO <gfs\_any\_string>.

Допускается присвоение field-symbols к field-symbols

ASSIGN <gfs\_any\_string\_1> TO <gv\_any\_string\_2>.

Также присвоение может быть выполнено в операторах обработки внутренних таблиц

READ TABLE gt\_tab INDEX 1 ASSIGNING <gfs\_any\_struct>.

*или*

LOOP AT gt\_tab ASSIGNING <gfs\_any\_struct>.

*\* поле field-symbol будет ссылаться на текущую запись*

ENDLOOP.

Применяя field-symbols при обработке больших таблиц можно получить существенную оптимизацию в производительности, т.к. если бы мы использовали для получения данных из таблицы переменную структурного типа - системе пришлось бы каждый раз копировать данные из таблицы в переменную.

Если поле field-symbol присваивается переменной с определенной структурой и явно указывается тип field-symbol - получение доступа к её компонентам аналогично работе с обычной структурой

DATA: ls\_flight TYPE sflight.

FIELD-SYMBOLS: <lfs\_flight> LIKE ls\_flight.

ls\_flight-connid = '182S'.

ls\_flight-seatsocc = 60.

ASSIGN ls\_flight TO <lfs\_flight>.

WRITE: 'На рейс', <lfs\_flight>-connid, 'забронировано мест:', <lfs\_flight>-seatsocc.

Когда структура для field-symbol имеет обобщенный тип - значение компонента можно получить используя оператор **ASSIGN COMPONENT**

DATA: ls\_flight TYPE sflight.

FIELD-SYMBOLS:

<lfs\_flight> TYPE ANY,

               <connid> TYPE sflight-connid,

               <seatsocc> TYPE sflight-seatsocc.

ls\_flight-connid = '182S'.

ls\_flight-seatsocc = 60.

ASSIGN ls\_flight TO <lfs\_flight>.

ASSIGN COMPONENT 'CONNID' OF STRUCTURE ls\_flight TO <connid>.

ASSIGN COMPONENT 'SEATSOCC' OF STRUCTURE ls\_flight TO <seatsocc>.

WRITE: 'На рейс',<connid>, 'забронировано мест:', <seatsocc>.

В момент присвоения, с помощью дополнения **CASTING** возможна адаптация значений если типы данных не совпадают но при этом они совместимы

DATA: BEGIN OF gs\_test,

        char1 VALUE '1',

        char2 VALUE '2',

      END OF gs\_test.

TYPES: ty\_num\_2 TYPE n LENGTH 2.

FIELD-SYMBOLS: <fs\_int> TYPE ty\_num\_2.

ASSIGN gs\_test TO <fs\_int> CASTING.

Начиная с версии ABAP 7.40 field-symbols на переменные можно объявлять непосредственно в месте где они будут использованы

DATA: lt\_tab TYPE STANDARD TABLE OF i WITH DEFAULT KEY.

lt\_tab = VALUE #( ( 1 ) ( 2 ) ( 3 ) ).

READ TABLE lt\_tab INDEX 1 ASSIGNING FIELD-SYMBOL(<ls\_one>).

WRITE <ls\_one>.

LOOP AT lt\_tab FROM 2 ASSIGNING FIELD-SYMBOL(<ls\_two>).

  WRITE <ls\_two>.

ENDLOOP.

В качестве примера динамического программирования с помощью field-symbols рассмотрим программу способную вывести в отчёте любую простую таблицу, состоящую из элементарных типов

DATA: lt\_spfli TYPE STANDARD TABLE OF spfli.

START-OF-SELECTION.

  SELECT \* FROM spfli UP TO 100 ROWS

    INTO CORRESPONDING FIELDS OF TABLE lt\_spfli.

  PERFORM write\_table USING lt\_spfli.

FORM write\_table USING it\_table TYPE ANY TABLE.

  FIELD-SYMBOLS:

    <lt\_table> TYPE ANY TABLE,

    <ls\_line>  TYPE any,

    <lv\_val>   TYPE any.

  ASSIGN it\_table TO <lt\_table>.

  LOOP AT <lt\_table> ASSIGNING <ls\_line>.

    WRITE: /.

    DO.

      ASSIGN COMPONENT sy-index OF STRUCTURE <ls\_line> TO <lv\_val>.

      IF sy-subrc NE 0.

        EXIT.

      ENDIF.

      IF sy-index EQ 1.

        WRITE: <lv\_val>.

      ELSE.

        WRITE: `,`, <lv\_val>.

      ENDIF.

    ENDDO.

  ENDLOOP.

ENDFORM.

**Обобщенные типы**

|  |  |
| --- | --- |
| **any** | любой тип данных; |
| **any table** | любая внутренняя таблица; |
| **clike** | обобщенный символьный тип – **c** | **d** | **n** | **t** | **string** - а так же плоские структуры, состоящие из элементов символьных типов; |
| **csequence** | текстовая последовательность – **c** | **string**; |
| **data** | любой тип данных; данный тип может быть использован в ссылочных переменных; |
| **decfloat** | числовой тип с плавающей запятой - **decfloat16** | **decfloat34**; |
| **hashed table** | любая хеш таблица; |
| **index table** | любая стандартная или сортированная внутренняя таблица; |
| **numeric** | числовой тип - **i** (**b**, **s**) | **p** | **decfloat16** | **decfloat34** | **f**. |
| **object** | любой объектный тип; |
| **simple** | любой элементарный тип данных включая плоские структуры состоящие из символьных элементов; |
| **sorted table** | любая сортированная таблица; |
| **standard table** | любая стандартная таблица; |
| **table** | ≡ **standard table**; |
| **xsequence** | байтовая последовательность - **x** | **xstring**. |

# ****Field symbol as a replacement of Work area****

DATA: lt\_mara TYPE STANDARD TABLE OF mara.

FIELD-SYMBOLS: <fs\_mara> TYPE mara.

SELECT \* FROM mara INTO TABLE lt\_mara UP TO 10 ROWS.

LOOP AT lt\_mara ASSIGNING <fs\_mara>.

<fs\_mara>-matkl = 'DEMO'.

ENDLOOP.

# Rem

# We don’t need to write the ****MODIFY**** statement which we would have written if we had used work area. This is because work area stores a copy of the internal table row whereas field symbol directly references the internal table row ⇒ *processing of internal table with field symbol is faster than the processing of internal table with work area*.

# ****Appending to internal table****

DATA: lt\_mara TYPE STANDARD TABLE OF mara.

FIELD-SYMBOLS: <fs\_mara> TYPE mara.

APPEND INITIAL LINE TO lt\_mara ASSIGNING <fs\_mara>.

IF <fs\_mara> IS ASSIGNED.

<fs\_mara>-matnr = 'MAT1'.

<fs\_mara>-matkl = '001'.

UNASSIGN <fs\_mara>.

ENDIF.

APPEND INITIAL LINE TO lt\_mara ASSIGNING <fs\_mara>.

IF <fs\_mara> IS ASSIGNED.

<fs\_mara>-matnr = 'MAT2'.

<fs\_mara>-matkl = '001'.

UNASSIGN <fs\_mara>.

ENDIF.

# Rem

# Always perform a check on field symbol that if it is assigned before doing any operation to avoid short dump.

# After doing the operation - unassign the field symbol.

# ****Reading internal table****

READ TABLE lt\_mara ASSIGNING <fs\_mara> WITH KEY matnr = 'MAT1'.

# ****TYPE ANY****

FIELD-SYMBOLS: <fs\_str> TYPE ANY.

FIELD-SYMBOLS: <fs\_data> TYPE ANY.

DATA: lw\_mara TYPE mara.

ASSIGN lw\_mara TO <fs\_str>.

IF <fs\_str> IS ASSIGNED.

ASSIGN COMPONENT 'MATNR' OF STRUCTURE <fs\_str> TO <fs\_data>.

IF <fs\_data> IS ASSIGNED.

<fs\_data> = 'MAT001'.

UNASSIGN <fs\_data>.

ENDIF.

UNASSIGN <fs\_str>.

ENDIF.

# Rem

# After assigning ****lw\_mara**** to ****<fs\_str>**** we cannot directly use the ‘-‘ operator on field symbol to access the fields of MARA structure i.e. ****<fs\_str>-matnr**** would produce syntax error. *This is because the field symbol type is declared only at runtime not at compile time* ⇒ to access the ****matnr**** field with field symbol first we need to assign that field component to a different field symbol and then use the new field symbol to update the ****matnr**** field

# ****TYPE ANY TABLE****

FIELD-SYMBOLS: <fs\_tab> TYPE ANY TABLE.

FIELD-SYMBOLS: <fs\_str> TYPE any.

FIELD-SYMBOLS: <fs\_data> TYPE any.

DATA: lt\_mara TYPE STANDARD TABLE OF mara.

DATA: lw\_mara TYPE mara.

ASSIGN lt\_mara TO <fs\_tab>.

SELECT \* FROM mara INTO TABLE lt\_mara UP TO 10 ROWS.

LOOP AT <fs\_tab> ASSIGNING <fs\_str>.

IF <fs\_str> IS ASSIGNED.

ASSIGN COMPONENT 'MATKL' OF STRUCTURE <fs\_str> TO <fs\_data>.

IF <fs\_data> IS ASSIGNED.

IF <fs\_data> EQ '01'.

*\* Do some processing*

ENDIF.

UNASSIGN <fs\_data>.

ENDIF.

ENDIF.

ENDLOOP.

# ****Reading internal table using generic field symbol****

FIELD-SYMBOLS: <fs\_tab> TYPE ANY TABLE.

FIELD-SYMBOLS: <fs\_str> TYPE any.

DATA: lt\_mara TYPE STANDARD TABLE OF mara.

ASSIGN lt\_mara TO <fs\_tab>.

SELECT \* FROM mara INTO TABLE lt\_mara UP TO 10 ROWS.

READ TABLE <fs\_tab> ASSIGNING <fs\_str> WITH KEY ('MATNR') = 'MAT001'.

# **Rem**

# Since **<fs\_tab>** is a generic field symbol its type will be known only at runtime ⇒ we cannot directly write the fields of MARA structure after WITH KEY - instead we have to write the field name within parenthesis as shown above. In ABAP this parenthesis indicates the compiler that the value of the operand will be decided at runtime ⇒ we don’t get any compilation error.

# Dynamic Programming in ABAP – Part 2 – Introduction to Data Reference

# <https://blogs.sap.com/2017/09/11/dynamic-programming-in-abap-part-2-introduction-to-data-reference/>

# Dynamic Programming in ABAP – Part 3 – An Example – ABAP RTTS

# <https://blogs.sap.com/2017/09/29/dynamic-programming-in-abap-part-3-an-example-abap-rtts/>

ABAP **Runtime Type Services - RTTS** consists of two components

1. **Runtime Type Identification - RTTI** – provides the methods to get the type definition of data objects at runtime.
2. **Runtime Type Creation - RTTC** – provides the methods to create the data objects at runtime with any type definition.

Basically ABAP RTTS provides a set of classes whose methods can be used for runtime type identification and runtime type creation. To know more about ABAP RTTS you can follow below link -

<https://wiki.scn.sap.com/wiki/pages/viewpage.action?pageId=42965>

**An example of dynamic programming**

**Requirement**

We need to write data from an internal table to a file on application server.

**Solution**

We will build one class having a method which will take any internal table as input and write its content in a file on application server.

CLASS **cl\_appserver\_writer** DEFINITION.

PUBLIC SECTION.

CLASS-METHODS: ***write*** IMPORTING

iv\_filename TYPE string

it\_data TYPE ANY TABLE *“can receive any internal table*

write\_header TYPE abap\_bool DEFAULT space

EXPORTING

ev\_message TYPE string.

ENDCLASS.

CLASS **cl\_appserver\_writer** IMPLEMENTATION.

METHOD ***write***.

TYPES: BEGIN OF ty\_comp\_detail,

name TYPE abap\_compname,

descr TYPE scrtext\_m,

END OF ty\_comp\_detail.

DATA: lo\_type\_def TYPE REF TO cl\_abap\_typedescr.

DATA: lo\_struct\_def TYPE REF TO cl\_abap\_structdescr.

DATA: lo\_table\_def TYPE REF TO cl\_abap\_tabledescr.

DATA: lo\_data\_def TYPE REF TO cl\_abap\_datadescr.

DATA: lo\_element\_def TYPE REF TO cl\_abap\_elemdescr.

DATA: lt\_components TYPE abap\_compdescr\_tab.

DATA: wa\_components LIKE LINE OF lt\_components.

DATA: lv\_str TYPE string.

DATA: lv\_filerow TYPE string.

DATA: lv\_counter TYPE i VALUE 0.

DATA: lw\_field\_info TYPE dfies.

DATA: ls\_comp\_detail TYPE ty\_comp\_detail.

DATA: lt\_comp\_detail TYPE TABLE OF ty\_comp\_detail.

FIELD-SYMBOLS: <row> TYPE any.

FIELD-SYMBOLS: <field\_value> TYPE any.

\* Using RTTS to get the runtime type information of the internal table

lo\_type\_def = cl\_abap\_tabledescr=>describe\_by\_data( it\_data ).

lo\_table\_def ?= lo\_type\_def.

lo\_data\_def = lo\_table\_def->get\_table\_line\_type( ).

lo\_struct\_def ?= lo\_data\_def.

\* Get the components of the structure

lt\_components = lo\_struct\_def->components.

CLEAR: lo\_data\_def.

\* If the WRITE\_HEADER is ABAP\_TRUE then fetch the label

\* of data element associated to each component of the

\* line type structure of internal table, if no data element

\* is associated then use component name as the header text

IF write\_header EQ abap\_true.

LOOP AT lt\_components INTO wa\_components.

lo\_data\_def = lo\_struct\_def->get\_component\_type( wa\_components-name ).

lo\_element\_def ?= lo\_data\_def.

lw\_field\_info = lo\_element\_def->get\_ddic\_field( ).

ls\_comp\_detail-name = lw\_field\_info-rollname. "Get the data element name

\* Calling FM to get data element text

CALL FUNCTION 'WCGW\_DATA\_ELEMENT\_TEXT\_GET'

EXPORTING

i\_data\_element = lw\_field\_info-rollname

i\_language = sy-langu

IMPORTING

e\_scrtext\_m = ls\_comp\_detail-descr

EXCEPTIONS

error = 1.

IF ls\_comp\_detail-descr IS INITIAL.

ls\_comp\_detail-descr = wa\_components-name.

ENDIF.

APPEND ls\_comp\_detail TO lt\_comp\_detail.

CLEAR: ls\_comp\_detail.

ENDLOOP.

ENDIF.

OPEN DATASET iv\_filename FOR OUTPUT IN TEXT MODE ENCODING DEFAULT.

IF sy-subrc EQ 0.

\* Writing header text for each column separated by comma

IF write\_header EQ abap\_true.

LOOP AT lt\_comp\_detail INTO ls\_comp\_detail.

lv\_counter = lv\_counter + 1.

IF lv\_counter EQ 1.

lv\_filerow = ls\_comp\_detail-descr.

ELSE.

CONCATENATE lv\_filerow ',' ls\_comp\_detail-descr INTO lv\_filerow.

ENDIF.

ENDLOOP.

TRANSFER lv\_filerow TO iv\_filename.

CLEAR: lv\_filerow, lv\_counter.

ENDIF.

\* Writing internal table content separated by comma

LOOP AT it\_data ASSIGNING <row>.

LOOP AT lt\_components INTO wa\_components.

lv\_counter = lv\_counter + 1.

ASSIGN COMPONENT wa\_components-name OF STRUCTURE <row> TO <field\_value>.

IF <field\_value> IS ASSIGNED.

lv\_str = <field\_value>.

IF lv\_counter EQ 1.

lv\_filerow = lv\_str.

ELSE.

CONCATENATE lv\_filerow ',' lv\_str INTO lv\_filerow.

ENDIF.

UNASSIGN <field\_value>.

ENDIF.

ENDLOOP.

TRANSFER lv\_filerow TO iv\_filename.

CLEAR: lv\_filerow, lv\_counter.

ENDLOOP.

CLOSE DATASET iv\_filename.

ev\_message = 'Success'.

ELSE.

ev\_message = 'Failure'.

ENDIF.

ENDMETHOD.

ENDCLASS.

**Typed Data Reference**

DATA lr\_num TYPE REF TO i.

CREATE DATA lr\_num.

**Dereferencing operator** **->\***.

lr\_num->\* = 2.

With ABAP 7.40 instead of **CREATE DATA** - the **NEW** operator can be used to create an anonymous data object –

lr\_num = NEW #( ).

**Assigning data object to data reference**

DATA: lv\_num TYPE i VALUE 2.

DATA: lr\_num TYPE REF TO i.

GET REFERENCE OF lv\_num INTO lr\_num.

lr\_num->\* = 4.

WRITE: / lv\_num.

With ABAP 7.40 instead of **GET REFERENCE** - the **REF** operator also can be used to assign the reference of an existing data object to a data reference

**REFERENCE INTO**

While processing internal table row we can use **REFERENCE INTO** statement to set references to table rows

DATA: lr\_mara TYPE REF TO mara.

DATA: lt\_mara TYPE TABLE OF mara.

SELECT \* FROM mara INTO TABLE lt\_mara UP TO 10 ROWS.

LOOP AT lt\_mara *REFERENCE INTO* lr\_mara.

WRITE: / lr\_mara->matnr.

ENDLOOP.

**Generic Data Reference**

DATA: lr\_num TYPE REF TO data.

Since **lr\_num** is generic - **lr\_num->\*** cannot be directly used at operand position

lr\_num->\* = 2. *“ так нельзя*

*In case of generic data reference - it can be dereferenced using a field symbol* -

FIELD-SYMBOLS: <num> TYPE any.

CREATE DATA lr\_num TYPE i.

ASSIGN lr\_num->\* TO <num>.

<num> = 3.

***Working with structures***

DATA: lr\_str TYPE REF TO data.

FIELD-SYMBOLS: <str> TYPE any.

FIELD-SYMBOLS: <data> TYPE any.

CREATE DATA lr\_str TYPE mara.

ASSIGN lr\_str->\* TO <str>.

ASSIGN COMPONENT 'MATNR' OF STRUCTURE <str> TO <data>.

<data> = '112'.

***Dynamically create data objects***

PARAMETERS: p\_tname TYPE tabname.

DATA: lr\_tab TYPE REF TO data.

FIELD-SYMBOLS: <tab> TYPE ANY TABLE.

CREATE DATA lr\_tab TYPE TABLE OF (p\_tname).

ASSIGN lr\_tab->\* TO <tab>.

IF sy-subrc EQ 0.

SELECT \* FROM (p\_tname) INTO TABLE <tab> UP TO 10 ROWS.

cl\_demo\_output=>display( <tab> ).

ENDIF.

**Difference between data reference and object reference:**

There are two types of reference variable

1. **Data reference** - can store the reference to any data object – variable | structures | internal tables etc. Either *the generic data type* or a *completely specified data type* can be specified.
2. **Object reference** - can store the reference to any class object. Either a *class* or an *interface* can be specified.

# ABAP on SAP HANA. Part XI. Are Native SQL and Open SQL Competitors?

<https://sapyard.com/abap-on-sap-hana-part-xi-are-native-sql-and-open-sql-competitors/>

[Содержание](#Содержание)

**Native SQL syntax**.

EXEC SQL.

…

ENDEXEC.

***Reasons for adopting Native SQL Approach***

1. Access tables that are not available on DDIC layer.
2. To use some of the special features supported by DB-Specific SQL - like passing hints to Oracle Optimizer / for the index which boosts performance/ etc.

***Pitfalls of Native SQL***

1. No syntax check at compile time for Native SQL - statements are directly sent to the database system ⇒ handle exception CX\_SQL\_EXCEPTION.
2. No automatic client handling | no table buffering.
3. All tables in all schemas can be accessed.

**Developers are responsible** for

* client handling;
* accessing correct schema;
* releasing DB resources;
* proper locking and handle the COMMIT.

**Open SQL - platform independent - it** is the only DB abstraction layer.

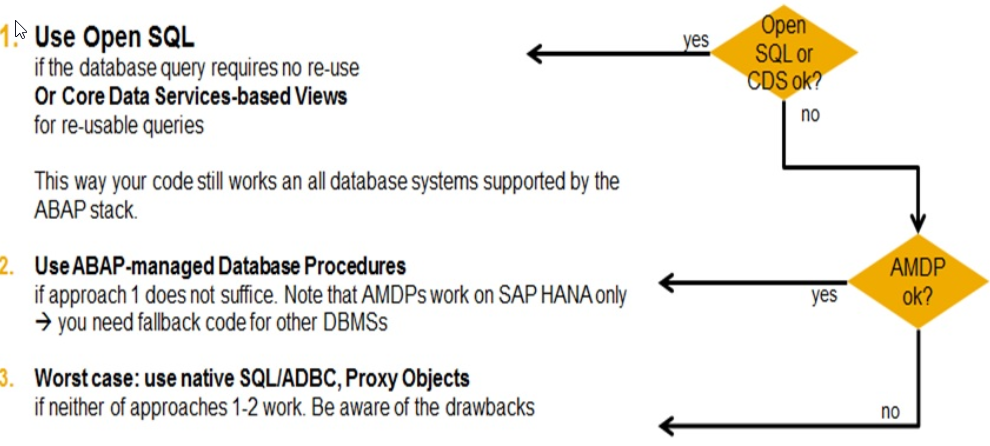
Open SQL supports **Code Push down** - data intense computations and calculations to the HANA DB layer.

# ABAP on SAP HANA. Part XII. Open SQL, CDS or AMDP, which Code to Data Technique to use?

<https://sapyard.com/abap-on-sap-hana-part-xii-open-sql-cds-or-amdp-which-code-to-data-technique-to-use/>

Interdependent SQL queries which will be used across applications (programs/object) - are the best example for choosing **CDS Views**.

**AMDP - p**owerful features of native SQL such as currency conversion and CE functions can be leveraged.



# ODATA

[Содержание](#Содержание)

<https://sapyard.com/tutorials-on-odata-sap-netweaver-gateway/>

**Odata**

**SAP NetWeaver Gateway**

**SAPUI5**

**SAP Fiori**

**OData** - **Open Data Protocol** interface is an open standard that can be consumed by any application | program | software | device of the Non-SAP World that can connect with SAP using the ***HTTP(s) protocol*** and can manipulate an XML document*.*

The OData protocol supports serialization in multiple popular formats including JSON | Atom/XML. With OData developers are able to build cross-platform Web and mobile Applications.

**SAP NetWeaver Gateway** sits in SAP Application Layer - it is the Window for outside world to peep (pi:p заглядывать) into SAP and transfer data to/from SAP. **Outer world** can send **HTTP(s)** message and **SAP** would provide them with **OData**.

Prior to OData external non-SAP developers have connected to SAP using RFC/BAPI or web services. In those cases the non-SAP developers /Web developers/ needed to know before hand the structures of the data passed from SAP.

OData offers database-like access to server-side resources ⇒ OData is also described as ODBC for the Web.

**ODBC** is a standard API to access the DBMS, independent of the database management systems or operating systems. ODBC achieves this by adding drivers between the Application layer and the DBMS to translate the queries requested by the applications into instructions which DBMS can understand. Similarly OData acts like middleware between producers and consumers to communicate data.

**GData** from Google **- the nearest competitor of OData**.

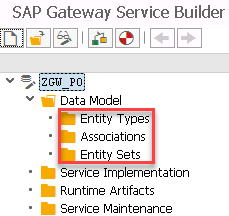
HTTP is **stateless** *-* every single HTTP request that is received by the Web Server is forgotten after a response has been sent across.

**REST** - **RE**presenational **S**tate **T**ransfer – it’s an alternative to the RPC and Web Services. REST is ***resource-based*** unlike RPC or SOAP which are ***action-based*** - REST services are really working with *r****esources*** instead of ***operations***.

In RESTful service,once you identified the resource - you will be working with a uniform interface because it uses HTTP methods GET | PUT | POST | DELETE.

**POST** is used to add new resource into the system.

**PUT** is to modify the existing resource.



**Entity Type** - is *our very own* **(наша собственная) structure** / work area - holds just one row.

**Entity Set** - is an **internal table.**

# BAdI

[Содержание](#Содержание)

# BAdI Implementation in ABAP on HANA for BW Extractors

# <https://blogs.sap.com/2021/04/01/badi-implementation-in-abap-on-hana-for-bw-extractors/>

## **Conversion routine Alpha**

<https://runyoncanyon-losangeles.com/blog/what-is-conversion-routine-alpha/>

# How To Build Dynamic Selections And Filters In SAP BW BEx Queries

<https://www.saptraininghq.com/how-to-building-dynamic-selections-and-filters-in-sap-bw-bex-queries/>

# Value help of Bex Variables used in SAP ANALYTICS CLOUD via SAP BW Live Connection in with BADI „Refining Input Help in Variable Screen“

<https://www.zpartner.eu/using-f4-help-badi-with-sap-analytics-cloud-and-sap-bw-live-connection/>

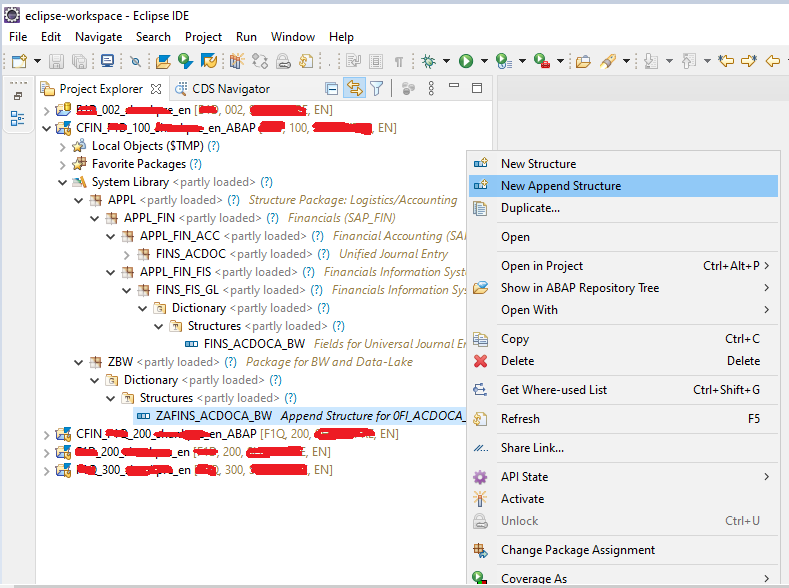
**Requirement**

In our example we have taken *NETDT* /Net Due Date/ field from ACDOCA table itself - this field is not present in extractor itself and hence we are doing *append in structure*.

**Step1** - ***Append the Structure of Datasource*** 0FI\_ACDOCA\_10

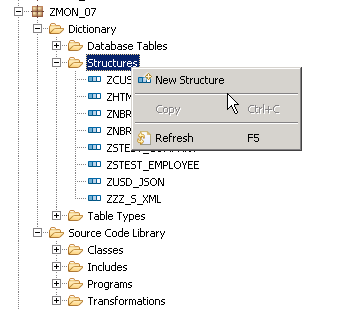
Go to the Eclipse ABAP Perspective and find *extract structure* for the datasource 0FI\_ACDOCA\_10.

Then create an *Append structure* and provide the field details

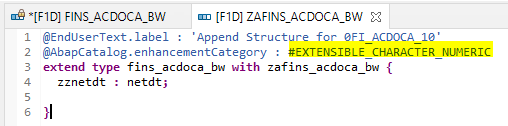


*Rem*

В текущей реализации пункта *New Append Structure* нет

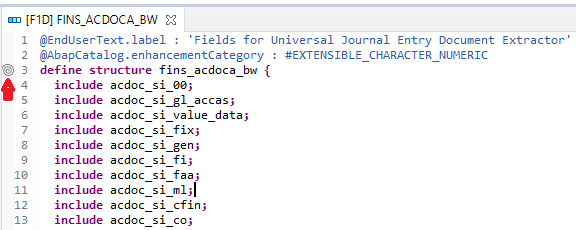


The Structure *ZAFINS\_ACDOCA\_BW* is created for single field ZZNETDT

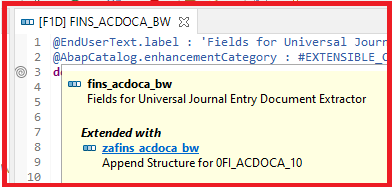


When the structure is created first, the Line number 2 will say ***#NOT\_EXTENSIBLE*** https://blogs.sap.com/wp-content/uploads/2021/04/Line-2-Non-Extendable.png - change this to ***#EXTENSIBLE\_CHARACTER\_NUMERIC*** - then only it will allow to extend the structure.

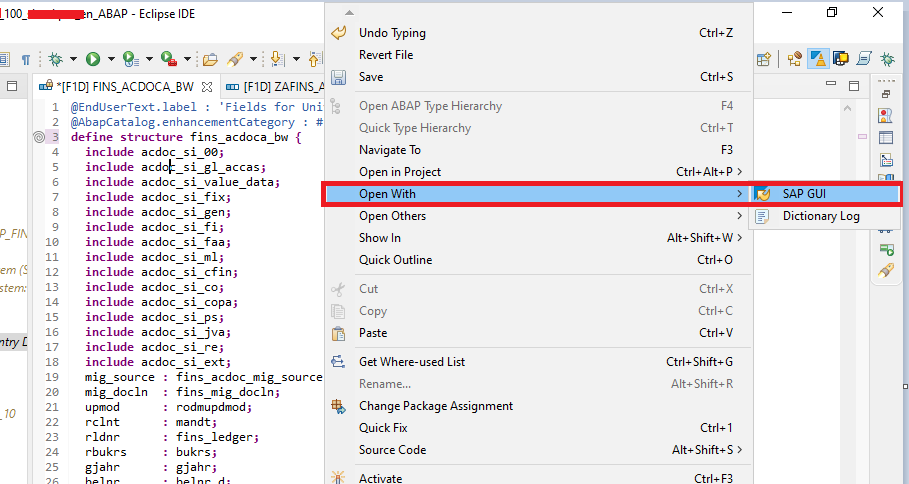
Once the Structure is enhanced, a *small icon* will appear in the left of define structure - that Icon is enhancement indicator/append indicator to see if we have any extension to this structure



The Structure can be seen by scrolling on the icon.

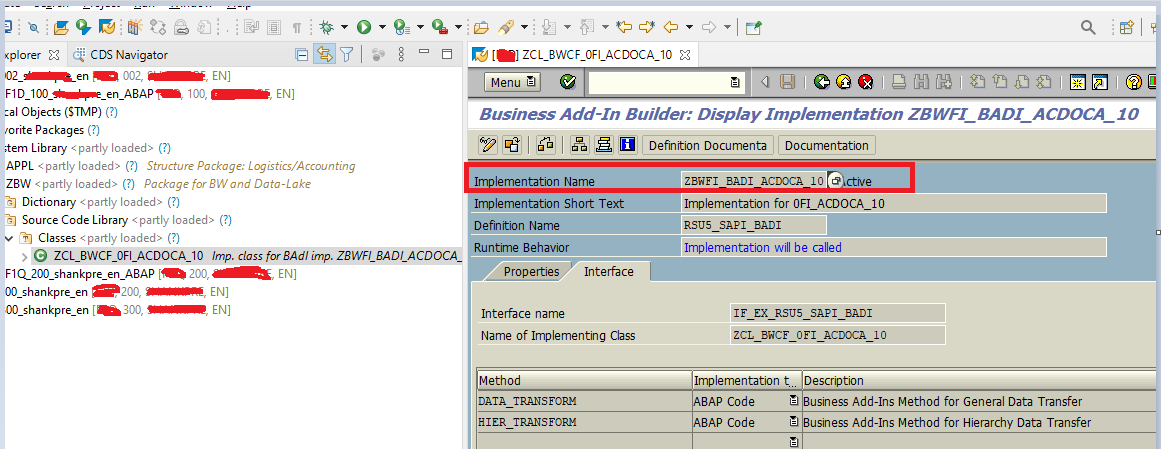


If required, we can also switch back to SAP GUI familiar screen by right clicking in the blank space and navigate to SAP GUI

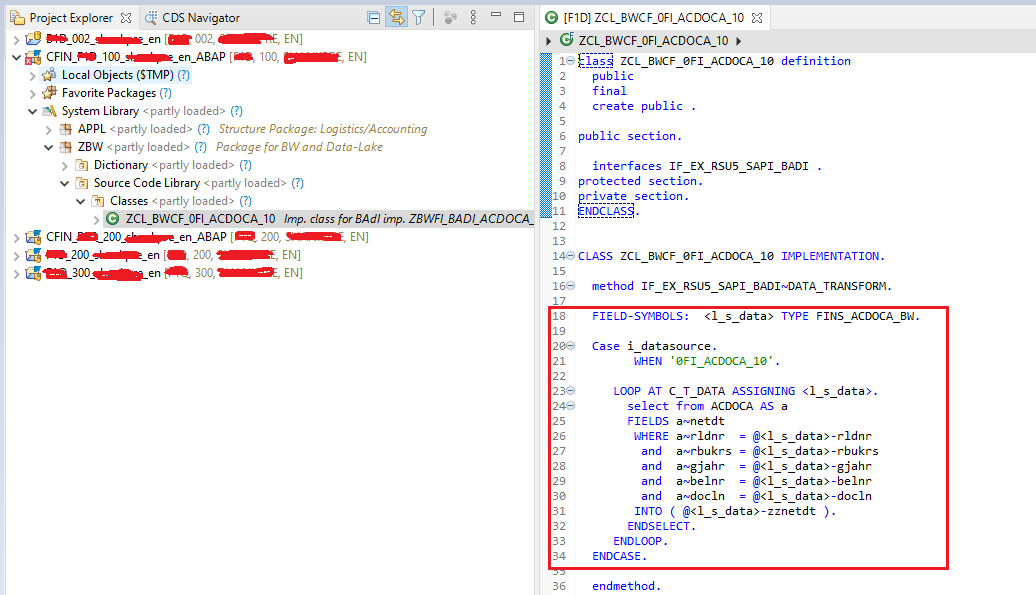


Now the Extract structure of the datasource is enhanced, next Step is create a Class Implementation.

**Step 2 - *Create a Class and Method*** in SE18/19 – GUI



**Step 3 -** Write the logic to populate the data in the Appended field in ABAP Perspective



The logic is written in new HANA Optimized ABAP code - this code will do the calculation in the database layer result in performance improvement.

**BAdI Implementation Code**

Below is the Code for BAdI written in Eclipse

FIELD-SYMBOLS: <l\_s\_data> TYPE FINS\_ACDOCA\_BW.

case i\_datasource.

WHEN '0FI\_ACDOCA\_10'.

LOOP AT *C\_T\_DATA* ASSIGNING <l\_s\_data>.

select from ACDOCA AS a fields a~*netdt*

where a~rldnr = @<l\_s\_data>-rldnr and

a~rbukrs = @<l\_s\_data>-rbukrs and

a~gjahr = @<l\_s\_data>-gjahr and

a~belnr = @<l\_s\_data>-belnr and

a~docln = @<l\_s\_data>-docln

into ( @<l\_s\_data>-*zznetdt* ).

endselect.

ENDLOOP.

endcase.

**Explanation of Code**

In this new approach of coding, the variables have a prefix **@** - this a new annotation based coding which can also be seen in ABAP CDS view.

This is a simple level of code where select statement in written under loop. In normal ABAP it will be an expensive statement, but with ABAP on HANA this will work pretty fast.

But if you are dealing with a more than 100 millions or billions of records, then this logic can also *start giving performance impact*.

**How can we optimize this**

In normal ABAP we use to select the relevant records from required table with respect of all the entries present in our C\_T\_DATA, and then we loop the C\_T\_DATA and read the internal table to populate the newly enhance field.

For All Entries statement is nothing but an inner-join between the two tables.

Hence, please find the *optimized code* below

types: begin of *ls\_acdoca*, *“ local structure*

rldnr type fins\_ledger,

rbukrs type bukrs,

gjahr type gjahr,

belnr type belnr\_d,

docln type docln6,

*netdt* type netdt,

end of ls\_acdoca.

DATA: *it\_acdoca* type standard table of *ls\_acdoca*,

SOURCE\_DATA type standard table of *FINS\_ACDOCA\_BW*,

*wa\_data* type *FINS\_ACDOCA\_BW*.

FIELD-SYMBOLS: <fs\_acdoca> type ls\_acdoca,

<l\_s\_data> type *FINS\_ACDOCA\_BW*.

sort SOURCE\_DATA by rldnr rbukrs gjahr belnr docln.

case i\_datasource.

WHEN '0FI\_ACDOCA\_10'.

IF C\_T\_DATA is not initial.

SOURCE\_DATA[] = C\_T\_DATA[].

select a~rldnr, a~rbukrs, a~gjahr, a~belnr, a~docln, a~netdt

from acdoca as a

*inner join* @SOURCE\_DATA as b

on a~rldnr = b~rldnr and a~rbukrs = b~rbukrs and a~gjahr = b~gjahr and a~belnr = b~belnr and

a~docln = b~docln

into table @it\_acdoca.

ENDIF.

sort *it\_acdoca* by rldnr rbukrs gjahr belnr docln.

LOOP AT C\_T\_DATA ASSIGNING <l\_s\_data>.

read table *it\_acdoca* assigning <fs\_acdoca>

with key rldnr = <l\_s\_data>-rldnr rbukrs = <l\_s\_data>-rbukrs gjahr = <l\_s\_data>-gjahr

belnr = <l\_s\_data>-belnr docln = <l\_s\_data>-docln.

if sy-subrc = 0.

<l\_s\_data>-*zznetdt* = <fs\_acdoca>-*netdt*.

endif.

ENDLOOP.

clear SOURCE\_DATA.

endcase.

**Explanation of Code**

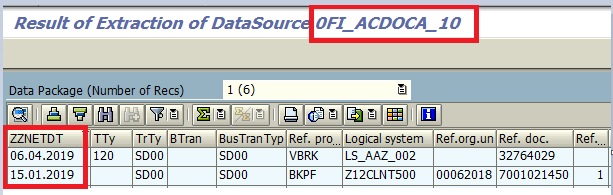
Required fields are selected from the table ACDOCA with all entries in the C\_T\_DATA on Key Fields of the table.

As discussed above For All Entries statement is nothing but an inner-join between the two tables.

Hence you can see inner join between table ACDOCA and SOURCE\_DATA /Copied from C\_T\_DATA/ in the code. It will select only required records from ACDOCA table and stores in internal table. The internal table will read when looping the C\_T\_DATA to populate the field ZZNETDT.

**Result of the Extractor**

The field ZZNETDT is populated as shown in the screenshot



# Introduction – ABAP Programming on HANA

<https://blogs.sap.com/2018/06/24/introduction-abap-programming-on-hana/>

**Code PushDown is carried out with** /is possible by/

1. OpenSQLprogramming /though limited at this point/
2. CDS Views
3. ABAP Managed Database Procedures

**OpenSQL Programming**

While writing code in OpenSQL the fields in select statement are comma separated, all *the host  variables are escaped using* ***@*** *sign*, the concatenation can be done in a single statement using **||** and so on.

REPORT **zact\_vendor\_osql**.

SELECT a~lifnr, b~bukrs, c~ekorg, d~name1, d~city1, d~region, d~country, d~post\_code1

INTO TABLE *@*data(*gt\_vendor*)

FROM lfa1 AS a

INNER JOIN lfb1 AS b ON b~lifnr = a~lifnr

INNER JOIN lfm1 AS c ON c~lifnr = a~lifnr

LEFT OUTER JOIN adrc AS d ON d~addrnumber = a~adrnr

WHERE a~loevm EQ *@*abap\_false AND a~sperr EQ @abap\_false AND

a~sperm EQ @abap\_false AND a~node1 EQ @abap\_false AND

b~sperr EQ @abap\_false AND c~loevm EQ @abap\_false AND c~sperm EQ @abap\_false.

GET RUN TIME FIELD data(lv\_end\_time).

DATA(lv\_time) = lv\_end\_time - lv\_start\_time.

cl\_demo\_output=>display\_data(EXPORTING value = *gt\_vendor*

name = *|*Duration *{ lv\_time }* ms|.



Here you can see that report ran for 1621 ms and returned us the desired results.

Though the code push code opportunity is limited at the moment when using OpenSQL, the syntax for this has been enhanced since 7.4. release. SAP has many syntaxes that can be utilized in code to improve its performance. To start with you can find very descriptive examples and code snippets in the ABAP glossary itself.

**Core Data Services /CDS/ Views**

SAP introduced a new data modeling infrastructure known as core data services or CDS. With CDS, data models are defined and consumed on database server rather than on application server. As a result, the table result view is created at the database level. CDS are completely compatible with openSQL and can be written using ABAP development tools like Eclipse Oxygen. These *can be consumed by reports and AMDPs* as well.

The above code will be created as a data definition in Eclipse and defined as follows

@AbapCatalog.sqlViewName: *'ZCDS\_ACT\_VEN'* *//SE11 SQL view name*

@AbapCatalog.Complier.compareFilter: true

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: 'CDS View data definition'

define view **ZCDS\_ACT\_VENDOR** *//CDS view name*

as

select from lfa1 as a

inner join lfb1 as b on a.lifnr = b.lifnr and b.sperr = ''

inner join lfm1 as c on a.lifnr = c.lifnr and c.sperm = ''

left outer join adrc as d on a.adrnr = d.addrnumber {

*key* a.lifnr, *key* b.bukrs, *key* c.ekorg, d.name1, d.city1, d.region, d.country, d.post\_code1

}

where a.loevm = '' and a.sperr = '' and a.sperm = ''

The CDS view returned result in 39ms.

Now CDS views could also be created with *parameters* or with *associations*. You may choose to create a CDS with parameters if you have a fixed result set and some input parameters to pass.

You could also create a CDS with association for a similar scenario if you have many tables to address in the view and if you want to keep the result set flexible.

**ABAP Managed Database Procedures /AMDP/**

AMDPs as the name says are database procedures that run on database directly and are written directly in ABAP. AMDPs are written using AMDP classes. Below is an example using the above scenario of how to create an AMDP class. The interface *IF\_AMDP\_MARKER\_HDB* distinguishes an AMDP class from other classes.

*Class definition*

CLASS **zcl\_act\_vendor\_amdp** *DEFINITION* PUBLIC FINAL CREATE PUBLIC.

PUBLIC SECTION.

INTERFACES *if\_amdp\_marker\_*hdb.

TYPES: BEGIN OF ty\_vendor,

lifnr TYPE lifnr,

bukrs TYPE bukrs,

ekorg TYPE ekorg,

name1 TYPE name1,

city1 TYPE adrc-city1,

region TYPE adrc-region,

country TYPE adrc-country,

post\_code1 TYPE adrc-post\_code1,

END of ty\_vendor,

*tt\_vendor* type sorted table of *ty\_vendor* with non-unique key *lifnr bukrs ekorg*.

METHODS ***get\_vendors\_amdp*** IMPORTING VALUE(lv\_clnt) type mandt

EXPORTING VALUE(*lt\_vendor*) type tt\_vendor.

ENDCLASS.

Similarly, an AMDP class implementation will have methods define with a syntax …BY DATABASE PROCEDURE FOR <database> LANGUAGE <language>.

In our case database will be *HDB* /HANA DB/ and language will always be *SQLSCRIPT*.

CLASS **zcl\_act\_vendor\_amdp** *IMPLEMENTATION*.

METHOD ***get\_vendors\_amdp***

BY DATABASE PROCEDURE FOR *HDB* LANGUAGE *SQLSCRIPT*

OPTIONS READ-ONLY USING lfa1 lfb1 lfm1 adrc.

lt\_vendor = SELECT DISTINCT a.lifnr, b.bukrs, c.ekorg,

d.name1, d.city1, d.region, d.country, d.post\_cdoe1

FROM lfa1 AS a

INNER JOIN lfb1 AS b ON b.mandt = a.mandt AND b.lifnr = a.lifnr AND

b.loevm = '' AND b.sperr = ''

INNER JOIN lfm1 AS c ON c.mandt = a.mandt AND c.lifnr = a.lifnr AND

c.loevm = '' AND c.sperm = ''

LEFT OUTER JOIN adrc AS d ON d.client = a.mandt AND d.addrnumber = a.adrnr

WHERE a.mandt = lv\_clnt AND a.loevm = '' AND a.sperr = '' AND a.sperm = '';

ENDMETHOD.

ENDCLASS.

This AMDP class can then be consumed in an ABAP program to achieve the code push down functionality.

REPORT **z\_act\_vendor\_amdp**.

TYPES: BEGIN OF ty\_vendor,

lifnr TYPE lifnr,

bukrs TYPE bukrs,

ekorg TYPE ekorg,

name1 TYPE name1,

city1 TYPE adrc-city1,

region TYPE adrc-region,

country TYPE adrc-country,

post\_code1 TYPE adrc-post\_code1,

END OF ty\_vendor.

DATA: *gt\_vendors* type sorted table of *ty\_vendor* with non-unique key *lifnr bukrs ekorg*.

GET RUN TIME FIELD data(gv\_start).

data(go\_ref) = NEW *zcl\_act\_vendor\_amdp*( ).

go\_ref->*get\_vendors\_amdp*( EXPORTING lv\_clnt = sy-mandt

IMPORTING lt\_vendor = *gt\_vendors* ).

GET RUN TIME field data(gv\_end).

data(gv\_time) = gv\_end - gv\_start. *// 3573 ms*

cl\_demo\_output=>display\_data( value = gt\_vendors

name = |Duration { gv\_time }ms| ).

*Rem*

You can measure the relative runtime of program segments in units of microseconds

GET RUN TIME FIELD <f>.

The first execution of this statement sets the value of field <f>, which should be of type I, to zero. Each subsequent execution of the statement sets the value of field <f> to the runtime of the program since the first execution of the statement.

**What to choose OpenSQL or CDS or AMDP?**

In the above example you can see that the performance was CDS > OpenSQL > AMDP.

Does that mean for the above scenario the best choice is to create a CDS? Not exactly! - if I do not reuse the CDS view then openSQL could be an equally effective choice.

Also note that CDS views and AMDP can only be created using ABAP Development Tools like Eclipse Oxygen.

The points below can give an idea on how to proceed to making the most productive choice.

***Choose Open SQL when***

1. The table selection is program specific and will not be reused.
2. When you do not have an ABAP Development Tool to create CDS or AMDP. The two can be consumed in GUI but cannot be created in GUI.
3. When the data in question does not involve intensive calculations, and can be managed easily by OpenSQL.
4. When you have a tricky selection screen with lot of select options that will be passed as single values too.

**Choose CDS views when**

1. The view can be reused among other views or programs.
2. When a large volume of data is involved from various data sources.
3. When you have good knowledge on how to write annotations to enhance your CDS view.
4. Only *single result set* is required.

**Choose AMDPs When**

1. You are fluent with SQL scripting because your entire code will be written in SQL script and the compiler fails in determining the runtime SQL script errors like divide by zero.
2. When you *have to handle cross client data* because AMDP does not do client handling on its own.
3. When *multiple result sets* are required.

# Using ABAP with Process Chain Decision Processes

[Содержание](#Содержание)

<https://blogs.sap.com/2015/02/24/using-abap-with-process-chain-decision-processes/>

There is no way to pass values or results from the ABAP process to the next step in the Process Chain short of causing the program to error or using events.

*Of the functions that allow external inputs there are a couple that I’ve found handy*

1. ***SYST-DATUM*** – This is a system field to give the date - it can be used with any of the DATE functions or string manipulation function /e.g. LEFT/ to determine the day or month for your decision. Examples for this sort of functionality would be process chains that you only want to run at month end or year end.
2. ***PREDECESSOR\_PARAMETER*** /and ***PROCESS\_PARAMETER***/ – These will return *values from a previous process in the Process Chain* based on the previous process’s implementation of the interface IF\_RSPC\_GET\_INFO.

*On top of these you also have some other options to control the flow of a Process Chain using ABAP*

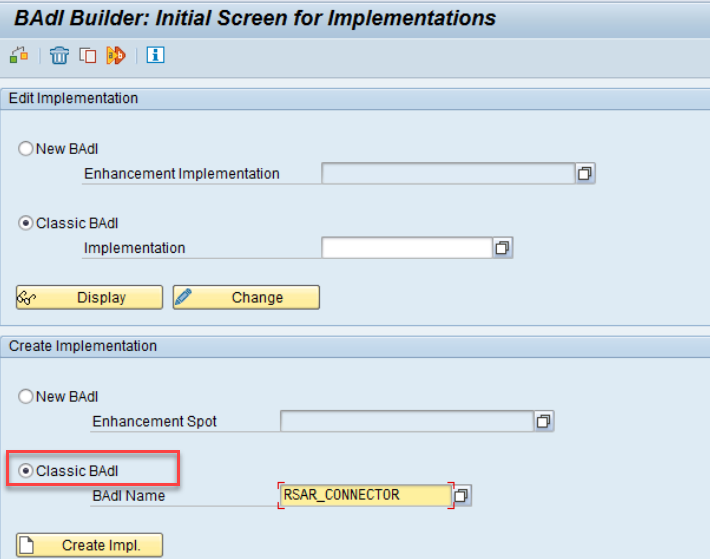
1. ***Raise an exception*** – In your ABAP code you set the program to raise an exception if a set of criteria occur which *would stop the the process chain completely* and no further processing is done.
2. ***Raise an event*** – Events /managed in **SM64**/ can be used *to start any Process Chain, Interrupt process or* even *background job* with the specific event raised as a start parameter.

But there is no easy way to determine what Process Chains will be called by which event at any given time.

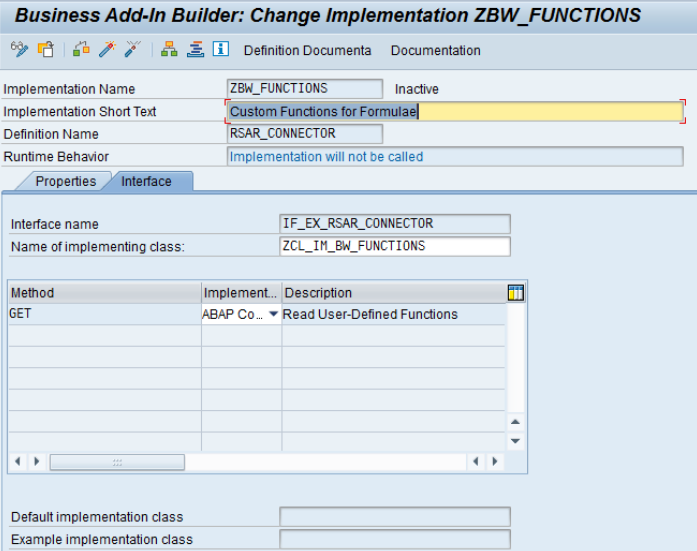
What if you’re decision isn’t based on any of these? In the following example the technical requirement is to *decide whether or not to delete the indexes on a cube before a DTP load* based on the number of records yet to be loaded. The reason we are using ABAP is because we cannot see how many records there are in the delta before the DTP starts to load and once the load is started we cannot delete the indexes. The threshold to delete the index is set at 1000 records.

**Steps to create ABAP decision**

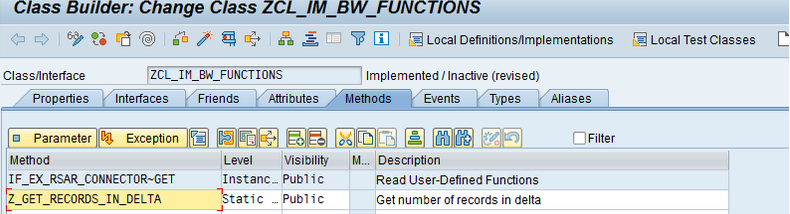
1) Using **SE19** create a new implemenation of BAdI RSAR\_CONNECTOR



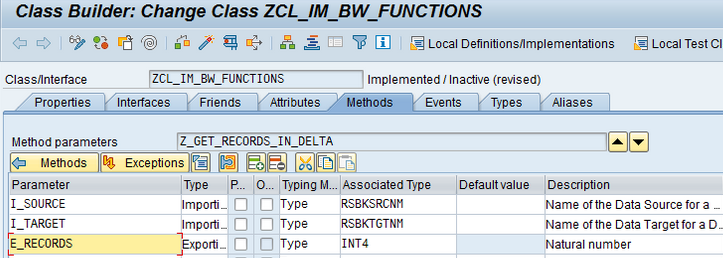
Give you implementation a name and enter the short text. In the Interface tab make a note of the implementing class name



Save your implementation then using **SE24** edit the implementing class. Enter your new method here that will be where you put your ABAP code. The method should be *static* and *public*.



1. Click the Parameter button to enter the parameters of the method. Note that *changing paramaters are not allowed* /only importing, exporting and returning are valid/ and only a single exporting or returning value is permitted.



1. Click on the Methods button then double click on the method name to enter the ABAP code

method ***Z\_GET\_RECORDS\_IN\_DELTA***.

DATA: l\_t\_req TYPE STANDARD TABLE OF rsbkrequest,

l\_s\_req LIKE LINE OF l\_t\_req,

l\_t\_req\_src TYPE STANDARD TABLE OF rsbkrequest,

l\_v\_count TYPE i.

FIELD-SYMBOLS: <fs\_req> TYPE rsbkrequest.

*"Get list of all requests loaded from source to target*

SELECT requid tstmp\_finish

APPENDING CORRESPONDING FIELDS OF TABLE l\_t\_req\_src

FROM rsbkrequest

WHERE tgt = i\_source AND tgt = i\_target

ORDER BY tstmp\_finish DESCENDING.

*"Get latest timestamp request*

READ TABLE l\_t\_req INTO l\_s\_req INDEX 1.

*"Get list of all requests loaded to source after timestamp of last load to target*

SELECT requid linesread

APPENDING CORRESPONDING FIELDS OF TABLE l\_t\_req\_src

FROM rsbkrequest

WHERE tgt = i\_source AND tstmp\_finish >= l\_s\_req-tstmp\_finish.

LOOP AT l\_t\_req\_src ASSIGNING <fs\_req>.

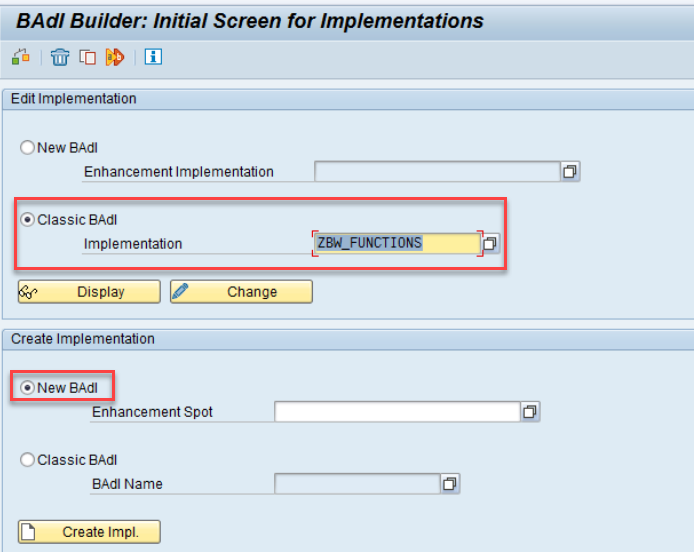
l\_v\_count = l\_v\_count + <fs\_rec>-linesread.

ENDLOOP.

e\_records = l\_v\_count.

5) Test, save and activate your method then come back to the main SE24 screen and activate the class.

6) Using **SE19** edit your BAdI implementation



7) On the Interface tab double click on the GET method to edit the code. The code here allows the method we’ve just created to be seen in the formula screen in the Decision process

method ***IF\_EX\_RSAR\_CONNECTOR~GET***.

data: l\_function type sfbeoprnd.

case i\_key.

when ' '.

l\_function-descriptn = 'BW Custom Functions'.

l\_function-tech\_name = 'C\_BW\_CUSTOM'.

append l\_function to c\_operands.

when 'C-BW-CUSTOM'.

l\_function-tech\_name = 'Z\_GET\_RECORDS\_IN\_DELTA' .

l\_function-descriptn = 'Get number of records in next del’.

l\_function-method = ‘Z\_GET\_RECORDS\_IN\_DELTA'.

l\_function-class = 'ZCL\_IM\_BW\_FUNCTIONS'.

append l\_function to c\_operands.

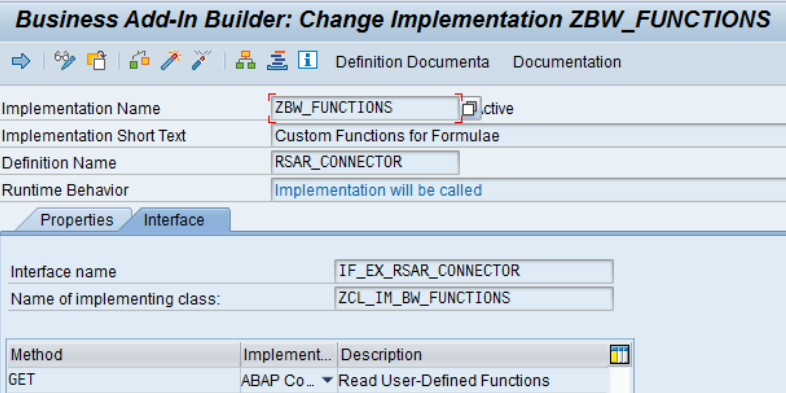
clear: l\_function.

endcase.

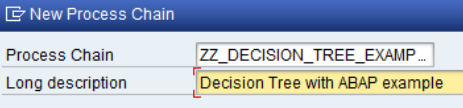
endmethod.

  The first part of the code - WHEN ‘ ‘ - sets up a new category in the Functions drop down menu, the second part of the code adds the new method created to that new category.

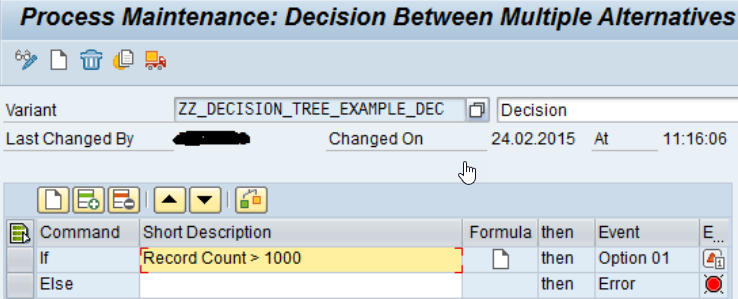
8) Activate your code then activate the BadI. Note that when it is activated the Runtime Behaviour field will be set to *Implementation will be called*



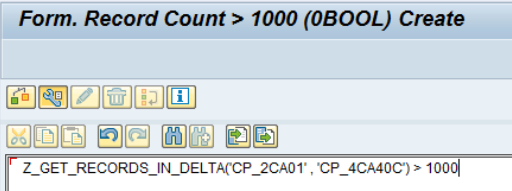
9) Create/Modify Process Chain using RSA1 – Here we are creating a new process chain



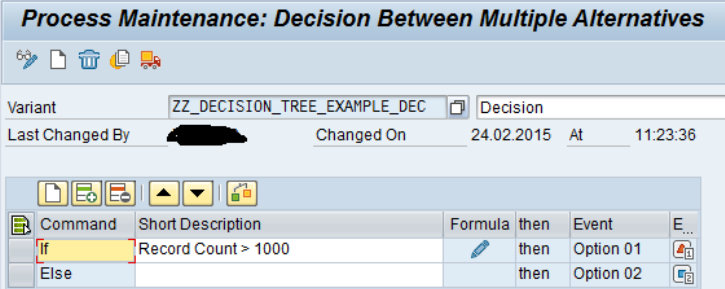
Create a new Decision Process and enter a name and description



Edit the formula, enter a name and click ok. In the drop down menu on the function side you can now see *BW Custom Functions* and in there is our new function



Exit and save your Decision process. Note that in this example the Else section event has been changed to Option 2 rather than the default Error



Now create the logic flow in the Process Chain

# SAP BW Process Chain - Passing parameters to ABAP Program

In the process chain, you add object abap with something like this

CALL FUNCTION ZXX\_XXX *"Your function module*

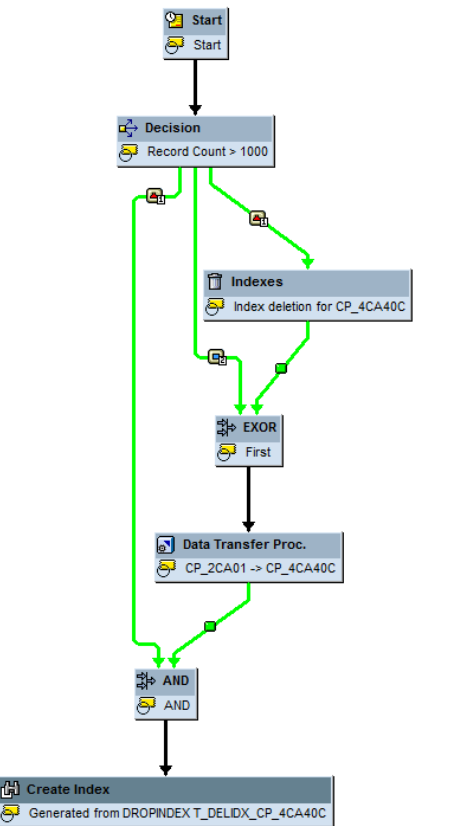
EXPORTING

val1 = your\_first\_parameter

var2 = your\_second\_parameter

IMPORTING

val3 = your\_result



The first step determines if the number of records is over 1000 in the next delta (Option 1). If so then it deletes the index and executes the DTP, if not (Option 2) then it just executes the DTP without deletion of indexes. The decision step is also used to determine if index creation is necessary and will do this once the DTP has completed it’s load (AND step at the end).

First run - Delta has 865000 records.

Second run - Delta has 0 records.

To make this a more global solution I’d probably include a value in a parameter table for the threshold value and possibly have a default value with the option to specify thresholds for those loads that do not fit.

In conclusion this is just a simple example of how you can include ABAP within a Process Chain Decision process. I hope it helps.

# CDS based data extraction

# Overview

<https://blogs.sap.com/2019/12/13/cds-based-data-extraction-part-i-overview/>

[Содержание](#Содержание)

***CDS based extraction*** is handled by the *Operational Data Provisioning* - ***ODP*** framework, the context is called ODP – ABAP CDS Views  /ODP\_CDS/. More info on ODP can be found in Rudolf Henneckes [blog](https://blogs.sap.com/2017/07/20/operational-data-provisioning-odp-faq/).

Starting from SAP S/4HANA 2020 and SAP S/4HANA Cloud Edition you can use the CDS view **I\_DataExtractionEnabledView** to *identify CDS views available for data extraction*.

A CDS view can be enabled for data extraction by just adding the following annotation

@Analytics.*dataExtraction.****enabled***

or

@Analytics.*dataExtraction.****enabled* : true**

## ***Classification of CDS views***

You can classify your CDS view *either* by specifying a value for *@Analytics*.*dataCategory*

* *#FACT* - for transaction data
* *#DIMENSION* - for master data attributes
* *#CUBE* - only to be used in exceptional cases

or specifiying a value for *@ObjectModel.dataCategory.*

* *#TEXT* - for text views
* *#HIERARCHY* for hierarchy views

***Extraction from CDS in full mode***

With these steps in place you have a first easy extraction enabled CDS view, in **full** mode - all available data is extracted in one data extraction run. For potentially smaller data volumes (~ < 106 data records) without any complex logic and load time considerations - a full extraction can cover the most straightforward cases.

Full extraction can mostly be applied for *master data* and *text* loads.

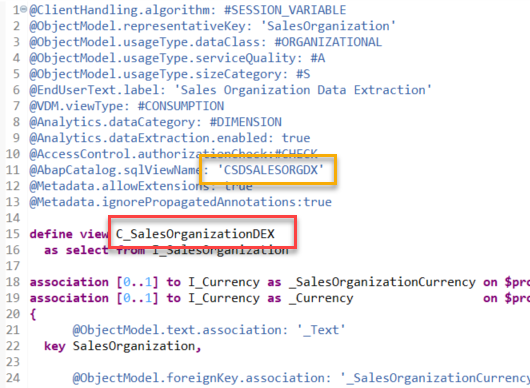
Please note that ***CDS views with input parameters*** only support ***full*** extraction.

After that you will be able to find this CDS view on SAP BW side, i.e. create a DataSource.

Rem

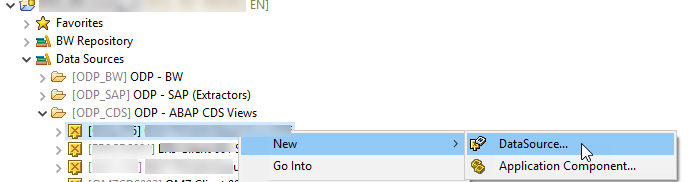
You will need to search for the *technical SQL view name* - *@AbapCatalog.sqlViewName* of the CDS view, not the *CDS view name*.

CDS View in source system

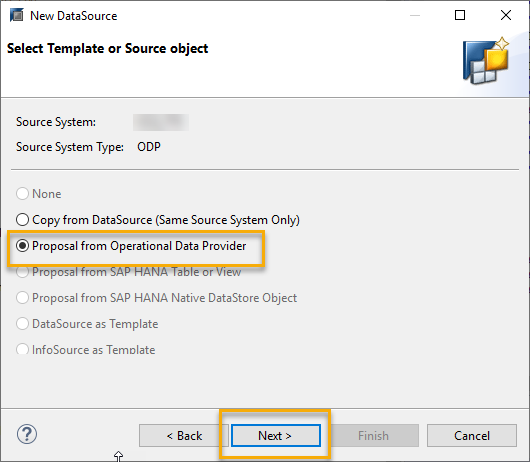


Data Source view on SAP BW/4HANA side

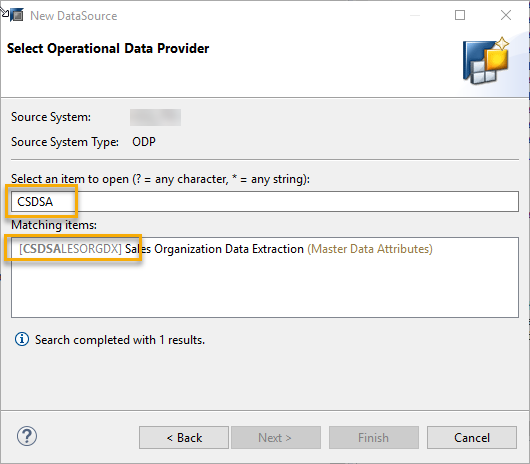
Right-click on your *ODP\_CDS* source system in the SAP BW modeling tools. In case you do not see the New -> DataSource option please make sure you have the latest version of the [BW Modeling tools](https://tools.hana.ondemand.com/#bw) installed.



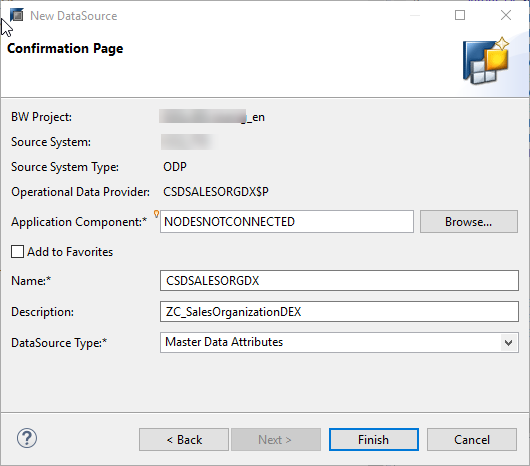
Choose *Next* -> Choose option Proposal from Operational Data Provider



Search for the *sqlViewName* of your CDS view



Provide an application component and a Description. The *DataSource Type* is derived from the *@Analytics.dataCategory | @ObjectModel.dataCategory* defined in the CDS view.



# CDS based data extraction – Delta Handling

<https://blogs.sap.com/2019/12/16/cds-based-data-extraction-part-ii-delta-handling/>

[Содержание](#Содержание)

You have two options for delta handling

* *Generic Timestamp / Date based* Delta - This requires date/time information being available in the relevant application tables that is updated based on changes of the application data.
* *Change Data Capture* - CDC Delta - This delta option captures changes to the application tables based on *data base triggers* and enables the ODP framework to just provision these newly created/changed/deleted records to the consumers.

## Generic Date/Timestamp Delta

The generic delta based on date / timestamp has been around since release SAP S/4HANA 1809 on-premise /локальный/.

You can use the following field types as delta criterion

* *UTC timestamp* [is the preferred way]
* date field /ABAP type *DATS*/

The following annotation identifies the relevant time/date field to be used by the ODP framework

*@Analytics*:{

dataCategory: #FACT,

dataExtraction: {

*enabled*: true

*delta.byElement*: {

name: 'LastChangeDateTime'

}

}

}

In case of a ***real-time delta*** subscription by a streaming process chain, the real-time daemon checks for new records in the CDS views every 15 seconds by default.

For ***non-real-time delta*** subscriptions new records according to the delta criterion are directly pulled from the CDS view during extraction.

As safeguarding measure, a ***safety interval*** can be specified - this can accommodate *technical delays* like waiting for a database commit on the application system side.

@*Analytics*:{

*dataCategory*: #FACT,

dataExtraction: {

enabled: true,

*delta.byElement* : {

name: 'LastChangeDateTime',

*maxDelayInSeconds* : 1800

}

}

}

Rem

If you do not add @Analytics.dataExtraction.delta.byElement.***maxDelayInSeconds*** annotation - a default delay of 1800 seconds, i.e. half an hour, is applied.

A record with a *time stamp* falling in this time safety interval will be selected *twice* from the CDS view - once with extraction run 1 and once with extraction run 2.

1. In the extraction ***run 1*** records belonging to the time interval *Start Timestamp of Extraction run 1 – maxDelayInseconds* will be stored for later comparison.
2. In subsequent extraction ***run 2*** - records belonging to this time interval will be selected again and compared against the formerly saved records/hashes of extraction run 1. Only records with changed hashes belonging to the safety interval will be extracted again to reflect the changes in the target system.

### Detecting deletion records using the generic delta

With the annotations so far, you will get newly created records as well as updates to existing records, but no deletions. The annotation @Analytics.dataExtraction.delta.byElement.*detectDeletions*enables the view to detect deleted records as part of the generic delta mechanism.

@*Analytics*:{

*dataCategory*: #DIMENSION,

dataExtraction: {

enabled: true,

*delta.byElement* : { name: 'LastChangeDateTime', *detectDeletions* : true }

}

}

Including this annotation will store all record key combinations being extracted in a *separate* data storage.

To identify deletions *all records*of this *data storage* are compared against *all records* still available in the *CDS view* during each extraction run. Records not available in the view anymore are sent to the consuming clients as deletions.

Needless to mention that this concept is only feasible for low level volumes of data /~< 10-6 data records/ ⇒ this mechanism is mainly applicable for small master data and text extractions.

The annotation @Analytics.dataExtraction.delta.byElement.*ignoreDeletionAfterDays* can be used to reduce the *time frame* of which records are considered for the deletion comparison.

@*Analytics*:{

*dataCategory*: #DIMENSION,

dataExtraction: {

enabled: true,

*delta.byElement* : {name: 'LastChangeDateTime', *detectDeletions* : true, *ignoreDeletionAfterDays* : '365'}

}

}

### Miscellaneous facts for Generic Delta

* Data records with an ***empty*** ChangeDate are only extracted during a *Delta Init with data*.

After the “Delta Init with data” has run, newly created records having an empty ChangeDate field will not be extracted anymore during further delta requests.

* *Timestamp* based delta extraction only supports one time field [time stamp or date]. This means only ***one field*** can be identified in the @Analytics.dataExtraction.delta.*byElement.name* annotation.
* ***Use only persisted time-stamp or date fields*** and refrain from using virtually derived or calculated fields in CDS views. This can lead to severe performance penalties.

## Change Data Capture Delta

Starting with SAP S/4HANA Cloud 1905 and SAP S/4HANA 1909 **FPS01** /on-premise/ an additional delta capability facilitating database triggers can be used.

For a CDS view using this delta method, changes in tables belonging to this view are recorded by the Change Data Capture mechanism. From a technology point of view this delta method makes use of real-time database triggers on table level based on SLT technology.

To function properly, the ***key fields*** of all underlying tables need to be exposed as elements of the CDS view and be mapped accordingly - the framework records the key field values of the changed table rows.

In case of

* an ***INSERT*** or ***UPDATE*** operation - a scheduled job is selecting the records based on these key combinations from the CDS view and pushes them as complete records into the *Operational Delta Queue* - ***ODQ***).
* a ***DELETE*** operation - the job generates an empty record with just the key field(s) filled, sets the deletion indicator and hands them over to ODQ.

### *Projections*

The easy case is a *projection* CDS view on a table. Whenever a record is inserted, updated or deleted in the underlying table - a record with the respective table key is stored in a generated *logging* table. Based on this info the scheduled job selects the data record from the CDS view and pushes it into the ODQ.

***To specify the CDC delta for simple projections***, you can use the  
@Analytics.dataExtraction.delta.*changeDataCapture*.***automatic***

@*Analytics*:{

*dataCategory*: #DIMENSION

dataExtraction: {

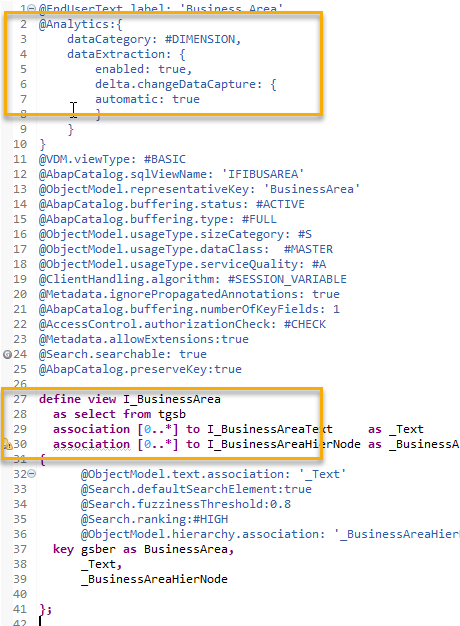
enabled: true,

*delta.changeDataCapture*: { *automatic* : true }

}

}

As an example for this, you can have a look at the CDS view for Business Area /I\_BUSINESSAREA/, which is a projection on table TGSB. In this case you only need the @ @Analytics.dataExtraction.changeDataCapture.automatic annotation.



### Joins

Currently only ***Left-outer-to-One* joins** are supported by the CDC framework.

Deletions of a record with regards to this CDS view only happen, if the record in the main table is deleted.

The developer needs to provide the mapping between the key fields of the underlying tables and their exposure as elements in the CDS view.

The annotations relevant for this mapping task are subsumed under  
@Analytics.dataExtraction.delta.changeDataCapture.*mapping*

@Analytics:{

dataCategory: #DIMENSION

dataExtraction: {

enabled: true,

delta.*changeDataCapture*: {

*mapping* : [ {

table : 'name of table',

role : #MAIN|#LEFT\_OUTER\_TO\_ONE,

viewElement : ['list of CDS view elements'],

tableElement: ['list of table fields']

}, ...

]

}

}

}

For each of the underlying tables for which a change should trigger a delta record, the following four mapping annotations must be maintained

* + @Analytics.dataExtraction.delta.changeDataCapture.mapping.***table*** - identify the name of the underlying table(s) to be logged.
  + @Analytics.dataExtraction.delta.changeDataCapture.mapping.***role*** - identify the role of a participating table.

The ***main*** table receives the value *#MAIN*. The key(s) of the CDS view correspond(s) exactly to the key(s) of the underlying main table to be logged. The ***outer*** table(s) receive(s) the value *#LEFT\_OUTER\_TO\_ONE\_JOIN*.

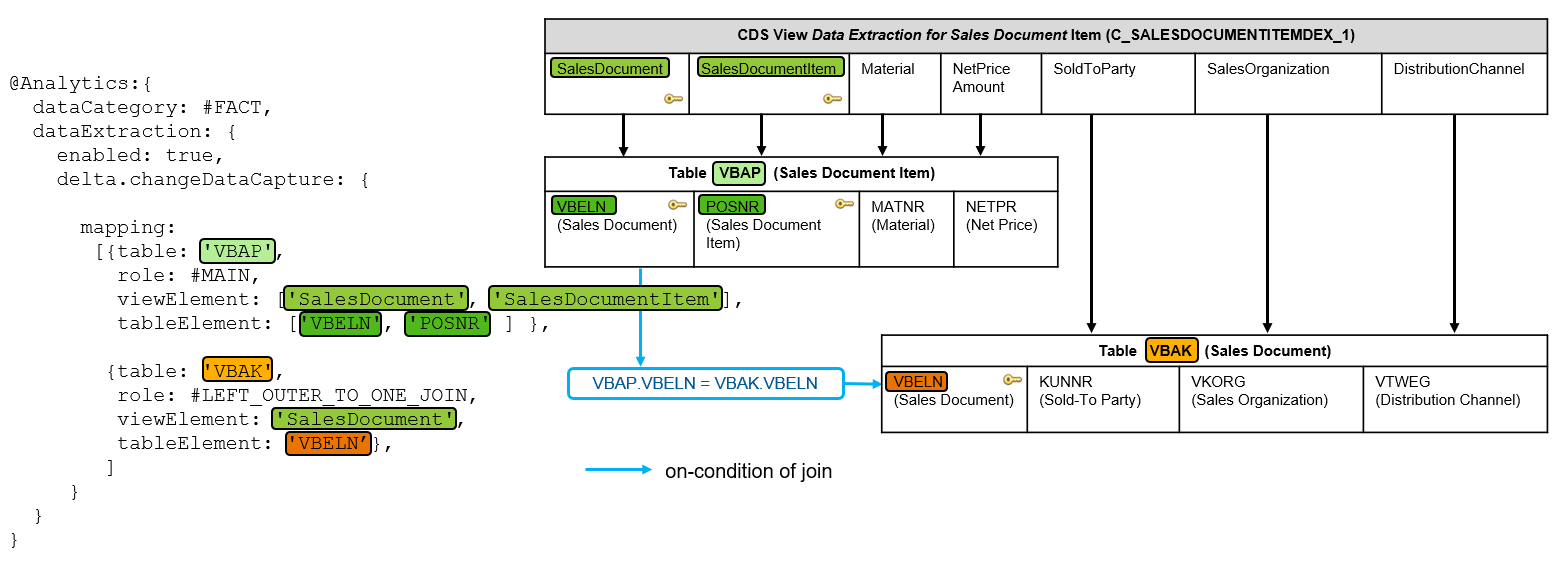
* + @Analytics.dataExtraction.delta.changeDataCapture.mapping.***viewElement***.

In case of

* + the ***main*** table - this list enumerates all exposed CDS view element names corresponding to the key fields of the main table
* an ***outer*** table - this list enumerates all exposed CDS view element names that correspond to key fields in this underlying outer table [*foreign key fields*].
  + @Analytics.dataExtraction.delta.changeDataCapture.mapping.***tableElement*** - enumerates the respective key fields of the underlying tables to be logged.

Both lists, *viewElement* and *tableElement*, must contain an equal number of elements and must list them in the ***same order*** so that corresponding fields match.

Important for the tables for which you want to have delta records recorded - ***all foreign key fields*** pointing to the respective outer tables which are used in joins need to be exposed as fields in the CDS view. If some of these fields are not meant to be consumed by the end user, they can be hidden by the annotation  
@Consumption.*hidden*



### Filters

You might only want to track certain values which you deem relevant for your extraction. For this purpose you can use the filter annotation  
@Analytics.dataExtraction.delta.changeDataCapture.mapping.***filter*.**

If you only have a single filter value for a field - It can also be used to replace a missing mapping between a viewElement and a tableElement [an on-condition for a join].

@Analytics:{

dataCategory: #DIMENSION

dataExtraction: {

enabled: true,

delta.*changeDataCapture*: {

*mapping* : [ {

table : 'name of table',

role : #MAIN|#LEFT\_OUTER\_TO\_ONE,

viewElement : ['list of CDS view elements'],

tableElement : ['list of table fields'],

*filter*: [{ tableElement : 'table element to be filtered'

operator : #EQ|#NOT\_EQ|#GT|#GE|#LT|#LE|#BETWEEN|#NOT\_BETWEEN;

value : 'filter value'

highValue : 'upper filter value in case of range' } ], ...

}, ...

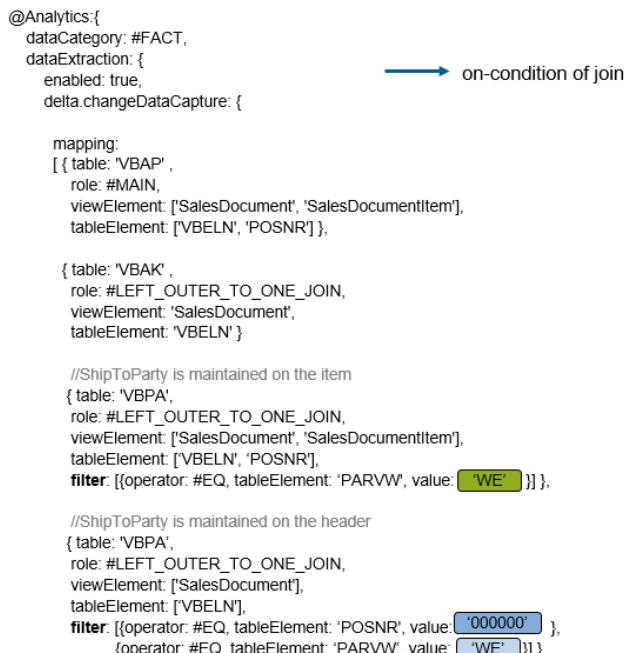
]

}

}

}

In some cases, a field for an explicit on-condition might be missing in one table, i.e. an additional constant value is used for determination of a unique record in the joined outer table.



methods LOAD\_TEXT\_ELEMENTS

final

returning

value(RT\_TEXT\_ELEMENTS) type TT\_TEXT\_ELEMENTS

raising

/IWBEP/CX\_MGW\_MED\_EXCEPTION .

methods DEFINE

redefinition .

methods GET\_LAST\_MODIFIED

redefinition .

protected section.

private section.

methods DEFINE\_ENTITYSPFLI

raising

/IWBEP/CX\_MGW\_MED\_EXCEPTION .

methods DEFINE\_ENTITYSFLIGHT

raising

/IWBEP/CX\_MGW\_MED\_EXCEPTION .

# CDS based data extraction – Part III Miscellaneous

<https://blogs.sap.com/2019/12/20/cds-based-data-extraction-part-iii-miscellaneous/>

### Entity Manipulation Language – EML | RAP Series

<https://sapabapcentral.blogspot.com/2020/12/entity-manipulation-language-eml-rap.html>

How do you define a ***range*** in SAP?

Defining ***Global Range***Type in Dictionary - SE11 -> enter the *name* of Range Table you would like to create in Data type input -> press *Create* -> On next popup select *Table* type -> Enter *short text* to describe you object and select menu option Edit-> Define as *range table* type.

How do you ***fill a range*** in SAP?

DATA lr\_vkorg TYPE *RANGE OF* vkorg.

TYPES: lr\_range\_t TYPE RANGE OF vkorg.

lr\_vkorg = VALUE *lr\_range\_t*( LET s = ‘I’ o = ‘EQ’ IN sign = s option = o ( low = ‘1100’ ) ( low = ‘1200’ ) ( low = ‘1300’ ) ( low = ‘1400’ ) ( low = ‘1500’ ) ).

How do you ***SELECT a range*** in SAP?

They are declared with syntax TYPE RANGE OF….About ranges and select-options.

|  |  |  |
| --- | --- | --- |
| OPTION | the selection operator | ‘EQ’ = Equal ‘BT’ = Between ‘GT’ = Greater than … |
| LOW | the lower limit of the interval | … |
| HIGH | the higher limit of the interval | … |