# [Содержание](#Содержание)

# [SAP HANA Studio](#SAP_HANA_Studio)

[DDL и DCL](#DDL_и_DCL)

# [Таблица TADIR](#Таблица_TADIR)

[CDS Entity](#CDS_Entity)

[CDS DB View](#CDS_DB_View)

[The syntax rules for defining CDS objects](#The_syntax_кгдуы_for_defining_CDS_object)

[Сlient column](#Сlient_column)

[Ключи CDS Entity и CDS DB View](#Ключи_CDS_Entity_и_CDS_DB_View)

[Аннотации](#АННОТАЦИИ)

[*Классы для обработки аннотаций*](#КЛАССЫ_ДЛЯ_ОБРАБОТКИ_АННОТАЦИЙ)

[Ассоциации](#Ассоциации)

### [CDS View с параметрами](#CDS_View_с_параметрами)

[‘where like’ in CDS view](#where_like_in_CDS_view)

[ABAP CDS - Table Functions](#ABAP_CDS_Table_Functions)

[Вопросы](#Вопросы)

# [Ресурсы](#Ресурсы)

[Table function](#Table_function_ресурсы)

[CDS View](#CDS_View)

[*Текстовый и графический редакторы CDS*](#редакторы_CDS)

# [ABAP CDS](#ABAP_CDS)

## [Classification of CDS views](#Classification_of_CDS_views)

# [Общая инф.](#Общая_инф)

# [Buffering](#Buffering)

# [Create a Basic CDS view](#Create_a_Basic_CDS_view)

# [CDS View with Expressions and Session Variables](#CDS_View_with_Expressions)

### [Code Pushdown](#Code_Pushdown)

# [Consume CDS View in an ABAP Program](#Consume_CDS_View_in_an_ABAP_Program)

# [CDS View Extension](#CDS_View_Extension) - *extend view … with …*

# [Parameters in a CDS view](#Parameters_in_a_CDS_view)

# [How to search for SAP standard CDS Views](#How_to_search_for_SAP_standard_CDS_Views)

# [Joins and Associations](#Joins_and_Associations)

[OData service creating](#How_to_create_an_OData_service)

# [VDM Annotation](#VDM_Annotation)

# [End to End Data Modeling and Reporting with CDS views](#End_to_End_Data_Modeling_and_Reporting)

# [CDS based data extraction](#CDS_based_data_extraction)

# [Delta Handling](#Delta_Handling)

### [Detecting deletion records using the generic delta](#Delta_on_Generic_Date)

### [Filters](#Filters)

# [CDS views - Ресурсы](#CDS_views_Ресурсы)

# [*CDS View accessing external system remotely*](#CDS_V_accessing_ext_system_remotly)

# [*How to analyze query performance for ABAP CDS Views*](#Analyze_query_performance)

# [ALV](#ALV)

# [CDS ALV report with selection criteria](#CDS_ALV_with_selection_criteria)

# Currency and Unit conversion in ABAP CDS views

<https://blogs.sap.com/2019/07/11/currency-and-unit-conversion-in-abap-cds-views/>

# Formatting amounts with Currency symbol – An ABAP Class solution

<https://blogs.sap.com/2021/05/24/formatting-amounts-with-currency-symbol-an-abap-class-solution/>

# AMDP class and methods and how to use in an extractor

<https://blogs.sap.com/2018/11/27/amdp-class-and-methods-and-how-to-use-in-an-extractor/>

# CDS based data extraction – Part I Overview

<https://blogs.sap.com/2019/12/13/cds-based-data-extraction-part-i-overview/>

# CDS based data extraction – Part II Delta Handling

<https://blogs.sap.com/2019/12/16/cds-based-data-extraction-part-ii-delta-handling/>

# CDS based data extraction – Part III Miscellaneous

<https://blogs.sap.com/2019/12/20/cds-based-data-extraction-part-iii-miscellaneous/>

# SAP HANA Studio

# [Содержание](#Содержание)

# Workspace is nothing but a folder where all your offline work and configurations get saved. Enter a folder path and press Ok.

The different type of major tasks are performed in different areas*/****perspectives****/*

* *SAP HANA Administration Console*- is used by administrators for administration and monitoring of the HANA Database.
* *SAP HANA Development* - is used for HANA model developments and other HANA related developments.

To do SAP HANA Development, you have two options, either use the ***SAP HANA Development*** Perspective or the ***Modeler*** Perspective. I recommend using the *SAP HANA Development* perspective as your default perspective and for all your HANA modeling needs as it allows you to create many more HANA related content like XS files, HDB tables etc.

# System pane

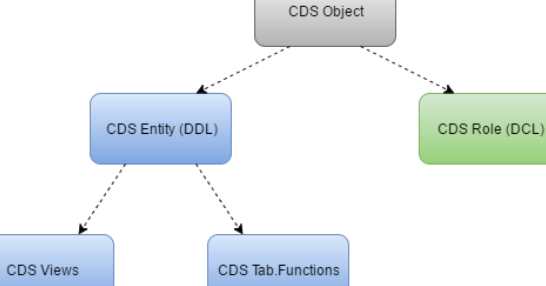
* ***Catalog***- this is where all the source metadata /*Tables, views* etc/ is grouped under.
* ***Content*** - this is where all your HANA development takes place. The HANA models that you create go under here.
* ***Provisioning*** - this is mostly used for *Smart data access*. All the source systems connected via *SDA* will have their tables displayed here - you can choose which one you want and build a virtual table for it in the *Catalog* section.
* ***Security*** - this is mostly for security consultants to maintain users and roles according to your role in the project – developers, administrators, testers and so on.

# ABAP Core Data Services - CDS

<https://abap-blog.ru/slovar/abap-cds-core-data-services/>

ABAP CDS поддерживается AS ABAP.

***ABAP CDS***



**DDL и DCL**

**DDL** -*CDS Views* | *CDS View Enhancements* | *CDS Table Functions.*

**DCL** - Data control language - *CDS Role*.

### **DDL**

# [Содержание](#Содержание)

**CDS Views - Keywords and Additions**

|  |  |
| --- | --- |
| **Language Element** | **Meaning** |
| DEFINE VIEW ... AS | Defines a CDS view |
| WITH PARAMETERS | Defines input parameters of a CDS view |
| SELECT [DISTINCT] ... FROM | SELECT statement of a CDS view |
| INNER JOIN ... ON | Inner join in a SELECT statement |
| LEFT|RIGHT OUTER JOIN ... ON | Outer join in a SELECT statement |
| ASSOCIATION ... TO ... AS ... ON | Defines an association for a path expression in a SELECT statement |
| 1: | Attribute of a path expression |
| INNER ... WHERE | Attribute of a path expression |
| LEFT OUTER ... WHERE | Attribute of a path expression |
| KEY ... AS | Element of a SELECT list |
| $EXTENSION.\* | Elements of an enhancement in a SELECT list |
| WHERE ... | WHERE clause of a SELECT statement |
| GROUP BY ... | GROUP-BY clause of a SELECT statement |
| HAVING ... | HAVING clause of a SELECT statement |
| UNION ALL ... | UNION clause of a SELECT statement |

**CDS Views - Expressions**

|  |  |
| --- | --- |
| **Operators** | **Meaning** |
| MAX, MIN, AVG, SUM, COUNT | Aggregate expression in a SELECT statement |
| +, -, \*, / | Arithmetic expression in a SELECT statement |
| CASE, WHEN, THEN, ELSE, END | Case distinction in a SELECT statement |
| CAST | Cast expression in a SELECT statement |
| =, <>, <, >, <=, >=, BETWEEN, LIKE, IS NULL, NOT, AND, OR | Relational expression in a SELECT statement |

**CDS Views - Predefined Functions**

|  |  |
| --- | --- |
| **Function** | **Meaning** |
| ABS, CEIL, DIV, DIVISION, FLOOR, MOD, ROUND | Numeric function in a SELECT statement |
| CEIL(*<number>*)- siːl перекрывать | Returns *the first integer that is greater* than *or equal* to the specified value. |
| FLOOR(*<number>*) | Returns *the largest integer that is not greater* than the specified numeric argument. |
| CONCAT, CONCAT\_WITH\_SPACE, *INSTR*, LEFT, LENGTH, LPAD, LTRIM, REPLACE,RIGHT, RPAD, RTRIM, SUBSTRING | String function in a SELECT statement |
| BINTOHEX, HEXTOBIN | Byte string function in a SELECT statement |
| COALESCE(*<expression\_list>*) – kəuə'les срастаться - сливаться | Returns *the first non-NULL expression from a list*. *At least two expressions must be contained in* <expression\_list> and all expressions must be comparable. The result is NULL if all the expressions are NULL. |
| UNIT\_CONVERSION, CURRENCY\_CONVERSION, DECIMAL\_SHIFT |  |
|  | Conversion functions in a SELECT statement |
| DATS\_IS\_VALID, DATS\_DAYS\_BETWEEN, DATS\_ADD\_DAYS, DATS\_ADD\_MONTHS | Date function in a SELECT statement |
| TIMS\_IS\_VALID | Time function in a SELECT statement |
| TSTMP\_IS\_VALID, TSTMP\_CURRENT\_UTCTIMESTAMP, TSTMP\_SECONDS\_BETWEEN, TSTMP\_ADD\_SECONDS | Time stamp function in a SELECT statement |

**CDS View Enhancements - Keywords and Additions**

|  |  |
| --- | --- |
| **Language Element** | **Meaning** |
| EXTEND VIEW ... WITH | Enhancement of a CDS view with an CDS view enhancement |

**CDS Table Functions - Keywords and Additions**

|  |  |
| --- | --- |
| **Language Element** | **Meaning** |
| DEFINE TABLE FUNCTION ... RETURNS ... IMPLEMENTED BY METHOD ... | Defines a CDS table function |

### **DCL**

**CDS Roles - Keywords and Additions**

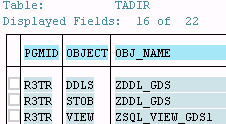
|  |  |
| --- | --- |
| **Language Element** | **Meaning** |
| DEFINE ROLE ... GRANT SELECT ON | Defines a CDS role |
| WHERE, AND, OR | WHERE clause in the definition of a CDS role |
| ... = ASPECT pfcg\_auth ... | Condition for mapping roles |
| COALESCE | Coalesce function in a SELECT statement |
| ... =|<>|<|>|<=|>=|LIKE|IS NULL ... | Literal condition |

**Таблица TADIR**

# [Содержание](#Содержание)

Ключи по которым можно найти сущности в таблице TADIR

* *DDL* - R3TR *DDLS* <DDL SourceName>.
* *CDS Entity* - R3TR *STOB* <EntityName>, где STOB – Structured object
* *DCL* *или CDS Role* - R3TR *DCLS* <DCL SourceName>.
* ***CDS Database View*** - R3TR *VIEW*.



**CDS Entity**

# [Содержание](#Содержание)

CDS entity*- CDS view | CDS table function*

* is a structured ABAP object *managed by ABAP Dictionary;*
* *is not transported;*
* *is created* *after* the transport of a piece of CDS source code when this *code is activated in the target system*;
* It’srecommended that *CDS source code of a data definition for a CDS entity need to have the same name as the entity*.

**Использование CDS entity**

***The ABAP runtime environment*** don’t access the internal metadata of CDS entities ⇒

CDS entities

* *cannot be used as a data type* for definitions of classic dictionary objects in ABAP
* *type of* a substructure
* *include* structure
* *row type of* a table type;
* *не доступны из Native SQL*; OpenSQL поддерживает только операции чтения.

Использование CDS entity

1. In ***ABAP CDS*** - as a *data source in other CDS entities*.
2. In ***ABAP*** - *as a data type* - that is being specified after the *TYPE* addition.

The components of the structured type are specified using the structure component selector *cds\_entity-comp*

DATA: xx *type zcds\_gds-/bic/bsprcac0*,

t1 *type* table of *zcds\_gds*.

1. In ***Open SQL*** can be used as a *data source*

select \* from *zddl\_gds*.

A piece of CDS source code *already used* for a CDS entity or CDS database view cannot be used *for a different* CDS entity or CDS database view.

CDS views are defined as platform-independent views. They can, however, contain components that are not currently supported by all database systems ⇒ the class **CL\_ABAP\_DBFEATURES** can be used *to determine whether the feature is supported in the current system*.

**Активация CDS data definition**

When the CDS data definition of a CDS view is activated - two ABAP Dictionary objects are created

* + *CDS entity*.
  + *CDS database view*.

CDS data definition *CDS entity*

*activating* *CDS database view*

**Просмотр CDS entity | CDS DB Virew**

ABAP Dictionary tool – ***se11***

* doesn’t recognize the CDS entity;
* can display CDS database view.

Program ***DEMO\_SHOW\_DDL\_SOURCE*** *displays CDS source code in SAP GUI*.

**The syntax rules for defining CDS objects**

# [Содержание](#Содержание)

***Keywords*** must be

* *all uppercase*
* *all lowercase*
* *in lowercase with an uppercase initial letter* - Select

***Names***

* are *not case-sensitive*;
* can have a *maximum of 30 characters*;
* can contain underscores - \_ | *slashes* – /.
* must *start with a letter* | *slash character | underscore*.

*Rem*

Underscores are recommended as the naming convention for *associations*.

* The separator for names with multiple parts – paths | columns with a prefixed data source | annotations is a period - .

***Literals***

* ***Numeric***literals must always be specified *in full* and a decimal point (.) used as a decimal separator if necessary - 0.5 | ~~.5~~.
* ***Character literals*** are enclosed in single quotation marks (').

**Комментарии**

Синтаксис DDL и DCL позволяет использовать *комментарии* *//* | *--* для однострочных и */\* … \*/* – для многострочных.

**Separators**

Statements can be closed using a *semicolon* (;). This is optional.

**Protected words**

The reserved names that cannot be used are specified in the database table ***TRESE***.

**Сlient column**

# [Содержание](#Содержание)

The CDS annotation ***@ClientDependent*** can be used to switch client dependency for a CDS view in ABAP CDS. The default setting is on - @ClientDependent: *true*.

* If a client-specific CDS entity is used as a data source in another CDS view - this view is also client-specific.
* The CDS DB view of a client-specific CDS view always has a client column - *unlike the CDS entity*.
* CDS views for application data should usually be client-specific.

***The name of the client column*** of a CDS DB view with activated *client* handling is defined in accordance with the following hierarchy

1. If the client column is specified in the SELECT list - its name or its alternative name is used.
2. If the client column *is not specified* in the SELECT list - MANDT is used.

*RTTI*is used to read the components of the associated data types

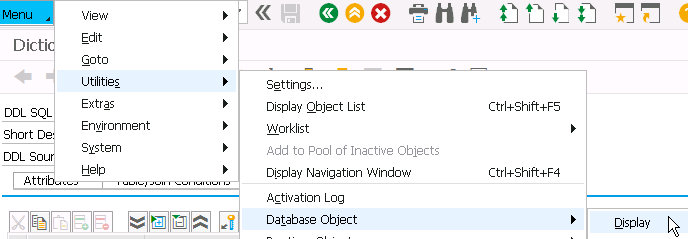
* *The structure of the client-specific CDS entity* without client column in the SELECT list - doesn’t have a client column.
* *The DB View of the client-specific CDS view* without client column in the SELECT list - has a client column.

**CDS DB View**

# [Содержание](#Содержание)

***Native SQL*** can only be used *to access the CDS DB view*.

*Просмотр DB View в SE11* - не учшая идея т.к. в нём вы не увидите специфичных для CDS вещей - UNION ALL | LEFT OUTER JOIN и пр. Для того чтобы их увидеть, необходимо перейти по следующему меню





…

**Ключи CDS Entity и CDS DB View**

# [Содержание](#Содержание)

CDS View содержат два типа определения ключей

1. Для ***CDS Entity*** ключевое слово [*key*](http://help.sap.com/abapdocu_750/en/abencds_f1_select_list_entry.htm#!ABAP_ADDITION_2@2@) может задать ***семантический ключ*** который используется при *неявных проверках полномочий*, определенных в DCL. SAP рекомендует всегда указывать семантический ключ для CDS View т.к. его наличие необходимо для ABAP фреймворков использующих CDS - например – IDA ALV.

*key* gds./bic/bcom0000

1. Ключ для ***CDS DB View*** задаётся автоматически и явл. ***техническим*** - актуален *для буферизации*.

Для обработки ***NULL значений*** можно в DDL использовать условие **IS [NOT] NULL** и ф-цию **COALESCE /**kəuə'les сливать(ся)/. Т.к. в ABAP нет типа, обозначающего NUL, - это значение будет автоматически преобразовано к *INITIAL* значению типа столбца.

*COALESCE*(<expression\_list>) - *returns the first non-NULL expression from a list*. At least *two* expressions must be contained in <expression\_list> and all expressions must be comparable. The result is NULL if all the expressions are NULL.

Начиная с ABAP 7.5 OpenSQL позволяет использовать *ассоциации* и выбирать из них данные

select cityfrom, countryfr,

*\geoinfo-latitude* as latitude

from Zddl\_assoc\_Sample.

**Аннотации**

# [Содержание](#Содержание)

**Syntax**

... @annotation[.annotation1[.annotation2]...][:*value* | *subannos* | *array*]  ...

... @*<*annotation[.annotation1[.annotation2]...] [:*value* | *subannos* | *array*]  .. - is used to introduce *annotations* in *comma-separated lists* thatare specified after a *list* element.

Возможные значения аннотаций

* *value*  - literals | true or false | enumeration values #value
* *subannos* – сокр.от *subannotations* - in curly brackets - {}
* *array* - in square brackets- []

The *annotation* individual *names* can contain *letters* | *digits* | *underscores* and *must start with a letter*.

Подробная инф. о *существующих аннотациях* есть в [документации](http://help.sap.com/abapdocu_740/en/abencds_annotations.htm) для [ABAP 7.5](http://help.sap.com/abapdocu_750/en/abencds_annotations_sap.htm).

В ABAP 7.5 с помощью аннотаций можно [опубликовать](http://help.sap.com/saphelp_nw75/helpdata/en/8d/ec506e13f949c2a82e8c775a94b73b/content.htm) *OData сервис* на базе *CDS View*. При этом используя описательные аннотации можно [задавать](http://scn.sap.com/docs/DOC-69643) свойства UI элементов в UI5 приложениях - *Fiori*. [Блог](http://scn.sap.com/community/abap/blog/2016/03/10/my-cds-view-self-study-tutorial--part-1-how-to-test-odata-service-generated-by-cds-view) в котором рассматриваются эти и другие возможности аннотаций.

*CDS View*

*annotations**OData сервис*

Типы аннотаций

* *SAP annotation* - *ABAP annotations* | *Component annotations*.
* *CDS view annotation*

**Классы для обработки аннотаций**

# [Содержание](#Содержание)

For *CDS entities* annotations

* *CL\_DD\_DDL\_ANNOTATION\_SERVICE* - for evaluations of the CDS entities annotations.
* The method *GET\_ANNOTATIONS* of the class *CL\_DD\_DDL\_ANALYZE* also returns the CDS entities annotations.

For *ABAP* annotations

* The value specified in the source code should consist of text in the original language of the CDS source code and is translated into the required languages. The methods of the class *CL\_DD\_DDL\_ANNOTATION\_SERVICE* read these texts in accordance with an input parameter for the language. There are also special methods *GET\_LABEL****\_***... that are designed for these texts only.

Пример – *Различные виды аннотаций*

@EndUserText.*label*: 'Demo View with Annotations'

@AbapCatalog.*sqlViewName*: 'DEMO\_VIEW\_ANNOT'

@AccessControl.*authorizationCheck*: #NOT\_REQUIRED

@ClientDependent: false

@v\_annot0

@v\_annot1:'abc'

@v\_annot2:123

*// v\_annot3 - is an array consisting of two elements which are identified in the evaluation using the indexes*

*// $1$ and $2$.*

@v\_annot3:[ 'abc', 123 ] *-- Array*

@v\_annot4:{ annot0, annot1:'abc', annot2:123 } *-- Subannos*

@v\_annot5.annot0

@v\_annot5.annot1:'abc'

@v\_annot5.annot2:123

*// v\_annot6 - is an array, in which the third component /a nested square bracket/ is ignored. The other*

*// elements are identified for evaluation using the indexes $1$ | $2$ | $4$.*

@v\_annot6:[ 'abc', 123, [ 'abc', 123 ], { annot0, annot1:'abc', annot2:123 } ]

@v\_annot7:{ annot0, annot1:[ 'abc', 123 ], annot2:{ annot0, annot1:'abc', annot2:123 } }

define view **demo\_cds\_view\_annotations**

  with parameters  @p\_annot1:*'abc'*

                    @p\_annot2:*123*

                    @EndUserText.label:*'Input Parameter'*

                   param : SYST\_UNAME

                   @<Environment.systemField:#USER

  as select from demo\_expressions

    { @f\_annot0

*key* id as key\_field

      @<f\_annot1:'abc'

      @<f\_annot2:123,

      @EndUserText:{ label:'Some field', quickInfo:'Some info' }

      @f\_annot1:[ 'abc', 123 ]

      num1 as some\_field

      @<f\_annot2:{ annot0, annot1:'abc', annot2:123 }

    }

Пример – Использование аннотаций для вывода только тех записей из CDS View где не все поля заполнены

CDS View

CDS View *sqlViewName*

*zcds\_semantic\_annotation ZCDS\_SEMANNO*

@AbapCatalog.*sqlViewName*: 'ZCDS\_SEMANNO'

@AccessControl.*authorizationCheck*: #NOT\_REQUIRED

define view **zcds\_semantic\_annotation** as

select from zcds\_gds as gds {

gds./bic/bcom0000,

@Semantics.*GoodsName* gds.txtlg,

@Semantics.*AccountPrice* gds./bic/bsprcac0,

@Semantics.*SupplierPrice* gds./bic/bsprcpvd,

@Semantics.*Margin* gds./bic/bsexc000,

@Semantics.*Discount* gds./bic/bsexc000 as bsexc000

}

where gds./bic/bemit000 = '0530'

ABAP Report

REPORT **zdemo\_cds\_semantics\_annotation**.

CLASS **demo** DEFINITION.

PUBLIC SECTION.

CLASS-METHODS ***main***.

ENDCLASS.

CLASS **demo** IMPLEMENTATION.

METHOD ***mai***n.

DATA: tbl\_gds TYPE STANDARD TABLE OF zcds\_semantic\_annotation WITH EMPTY KEY.

*\* The program uses cl\_dd\_ddl\_analyze=>get\_annotations to read the element annotations of a CDS entity*

*\* and gets the names of the elements defined as the required components of an Goods in the internal table*

*\* gds\_components.*

cl\_dd\_ddl\_analyze=>get\_annotations(

EXPORTING

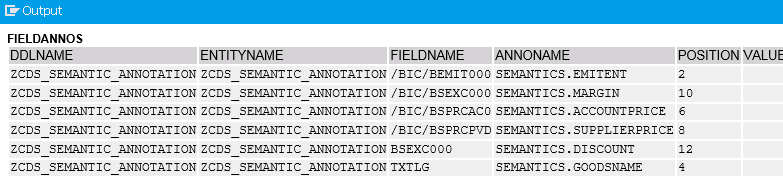
ddlnames = VALUE #( ( 'ZCDS\_SEMANTIC\_ANNOTATION' ) )

leaves\_only = abap\_true

IMPORTING

fieldannos = DATA(fieldannos) ).

cl\_demo\_output=>display( fieldannos ).



TYPES field\_anno LIKE LINE OF fieldannos.

DATA *gds\_annos* TYPE STANDARD TABLE OF *field\_anno-annoname* WITH EMPTY KEY.

gds\_annos = VALUE #(

( 'SEMANTICS.GOODSNAME' )

( 'SEMANTICS.ACCOUNTPRICE' )

( 'SEMANTICS.SUPPLIERPRICE' )

( 'SEMANTICS.MARGIN' )

( 'SEMANTICS.DISCOUNT' ) ).

DATA *gds\_components* TYPE STANDARD TABLE OF *field\_anno-fieldname* WITH EMPTY KEY.

gds\_components = VALUE #(

FOR gds\_anno IN gds\_annos

( VALUE #( fieldannos[ annoname = gds\_anno ]-fieldname DEFAULT '---' ) ) ).

SELECT \* FROM zcds\_semantic\_annotation INTO @DATA(gds).

LOOP AT gds\_components INTO DATA(component).

*" A dynamic ASSIGN statement is used to check whether all required elements have a non-initial value*

*" for each read row - the output of rows that have at least one blank field.*

ASSIGN COMPONENT component OF STRUCTURE gds TO FIELD-SYMBOL(<value>).

IF sy-subrc <> 0 OR <value> IS INITIAL.

tbl\_gds = VALUE #( BASE tbl\_gds ( gds ) ).

EXIT.

ENDIF.

ENDLOOP.

ENDSELECT.

cl\_demo\_output=>display( tbl\_gds ).

ENDMETHOD.

ENDCLASS.

START-OF-SELECTION.

demo=>main( ).

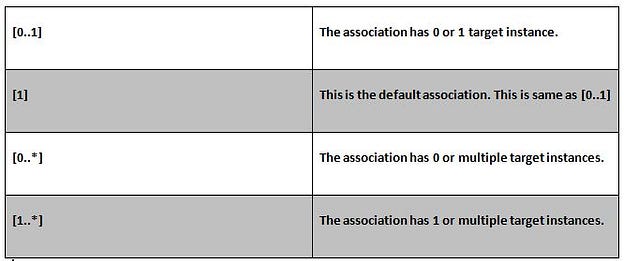
**Ассоциации**

# [Содержание](#Содержание)

CDS view with association came into the picture since there was a drawback with joins. For example - if you have created a CDS view with joins on 3 different tables, then this join condition gets triggered whenever the CDS view gets called. Here arises the problem - even if the end-user wants to see the data only for 2 tables, he has to by default see the output for the entire 3 tables. *By using joins, the performance gets delayed* since it fetches all the data from 3 tables. And so, SAP came with an enhancement of joins called ***association*** /***joins on demand*/**.

The syntax of Association is***association* [min….max].**

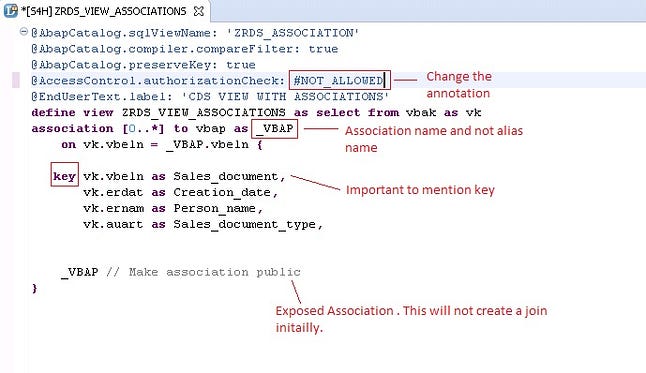
**Where *min* and *max* are the instances of the target association.**



There are two types of associations

* *exposed*;
* *ad-hoc*.

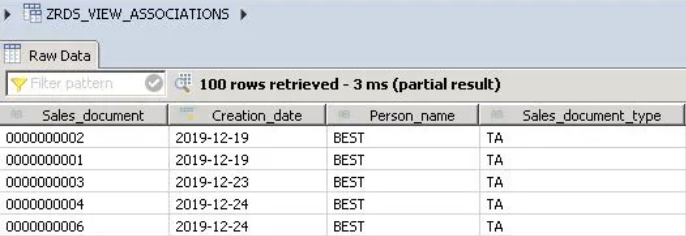
*EXPOSED* ASSOCIATION



NOTE

If you get a warning of *no access control* for an entity, change *@AccessControl.authorizationCheck: #NOT\_ALLOWED*.

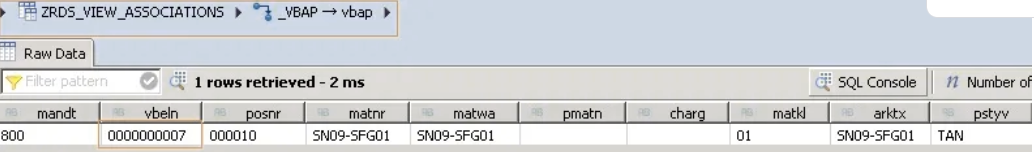
Activate /Ctrl + F3/ and run the code - you will get the following execution



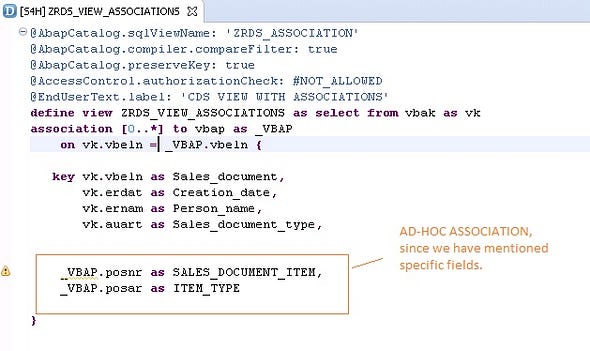
You will notice that only the fields which we have mentioned for table *VBAK* will be displayed. There is no join performed /i.e no fields of table *VBAP*/. To see further information for *vbeln* 000000007 from table *VBAP*, right-click and choose *Follow Association* - another pop-up window will open. Click the Association type



Check the output - you will be able the see the data of *vbeln* 00000007 from the table *VBAP*. This is called as *JOIN ON-DEMAND*.

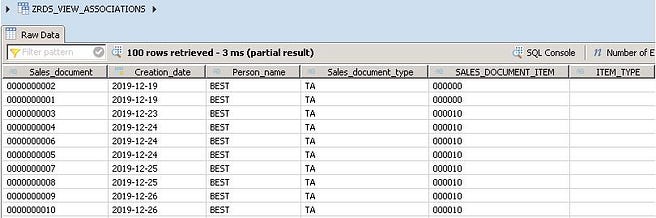


AD-HOC ASSOCIATION



You can see the SQL statement. Right-click on the above code snippet -> *Show SQL* create a statement.

Now to check the output, save, activate, and click on the run icon.



Here you will be able to see all the fields of the select query including table VBAP. This is called as AD-HOC Association.

<https://medium.com/@richa.solanki18/introduction-to-sap-hana-cds-c7c3b27e634f>

**See**

# ABAP CDS (Core Data Services)

<https://abap-blog.ru/osnovy-abap/slovar/abap-cds-core-data-services/>

# Introduction To SAP HANA CDS

<https://medium.com/@richa.solanki18/introduction-to-sap-hana-cds-c7c3b27e634f>

define view **zcds\_gds** as select from /bic/abdgds0002 as gds

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

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

…

{

key gds./bic/bcom0000,

…

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

…

}

**Кардинальность ассоциаций**

Кардинальность не обязательно к указанию - по умолчанию будет выставлена [0..1].

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

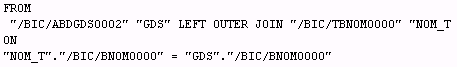
on nom\_t./bic/bnom0000 = gds./bic/bnom0000



{

left outer join /bic/tbnom0000 as nom\_t

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



Ключевое слово ***$projection*** может быть использовано *для доступа к полю в ассоциации* для которого задан alias

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

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

Используем ассоциацию в другом CDS View

define view **zcds\_use\_assoc** as select from zcds\_gds

{

*gds\_t*.txtlg as Gds\_name,

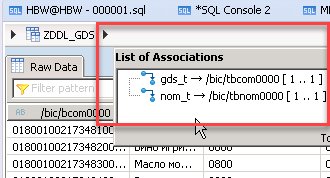
*nom\_t*.txtlg as Nomenclature\_name

}

*Тип JOIN’a может быть переопределен* когда мы получаем доступ к ассоциации через *path* expression

nom\_t*[inner]*.txtlg as Nomenclature\_name

Инструмент **Data Preview** позволяет ***проходить по ассоциациям*** -



### **CDS View с параметрами**

# [Содержание](#Содержание)

При указании типа можно ссылаться *на* 1) *элементы данных* или 2) *DDIC типы*, указав префикс*abap* - например *abap*.int4

define view **zcds\_gds**

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

as … { …}

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

***The possible options for dtype and their meanings -***

|  |  |
| --- | --- |
| ***dtype*** | ***Predefined Data Type in ABAP Dictionary*** |
| abap.char( len ) | CHAR with length len |
| abap.clnt[(3)] | CLNT |
| abap.cuky(5) | CUKY with length 5 |
| abap.curr(len,dec) | CURR with length len and with dec decimal places |
| abap.dats[(8)] | DATS |
| abap.dec(len,dec) | DEC with length len and with dec decimal places |
| abap.fltp[(16,16)] | FLTP |
| abap.int1[(3)] | INT1 |
| abap.int2[(5)] | INT2 |
| abap.int4[(10)] | INT4 |
| abap.int8[(19)] | INT8 |
| abap.lang[(1)] | LANG |
| abap.numc( len ) | NUMC with length len |
| abap.quan(len,dec) | QUAN with length len and with dec decimal places |
| abap.raw(len) | RAW with length len |
| abap.sstring(len) | SSTRING with length len |
| abap.tims[(6)] | TIMS |
| abap.unit(3) | UNIT with length 3 |

**Note** - Currently no *structured* or *tabular parameters* are supported - only elementary data types.

Вызов из SQL Console

SELECT TOP 1000 \* FROM "SAPABAP1"."ZDDL\_GDS"(p\_emit = 020)

*или*

SELECT TOP 1000 \* FROM "SAPABAP1"."ZDDL\_GDS"(p\_emit = 020)

INTO TABLE @DATA(lt\_data).

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

*$session.user* as User\_.

Параметры могут быть использованы как элементы в перечне полей для выбора

*:p\_emit* as Emitent

When using a *CDS entity with input parameters* as

* + *data source* or in a *path expression of a different CDS view*

or

* + *data source of a SELECT statement from Open SQL in ABAP*

– each input parameter must be assigned a suitable actual parameter whose value is then used in the operand positions in which the parameter is specified. The ***actual parameters*** are assigned using a parenthesized comma-separated list

... cds\_entity( pname1 : act1, pname2 : act2, ...) ...,

…as

select from zcds\_gds(*p\_emit : '0020'*) as gds

{ … }

where gds./bic/bemit000 = *:p\_emit*

*или*

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

as select from zcds\_gds(*p\_emit* : *$parameters.p\_emit*) as gds

{ … }

where gds./bic/bemit000 = *:p\_emit*

**In Open SQL** an equals sign is used instead of a colon -

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

The annotation *@Environment.systemField* can be used to make Open SQL *pass ABAP system fields to an input parameter implicitly*.

Note

*@Environment.systemField* *is ignored in parameter passing to the CDS entity in other CDS entities* - explicit actual parameters must be specified here.

The value *#CLIENT* of *@Environment.systemField* is particularly significant for client-specific CDS table functions.

### **IDA ALV - *List Viewer with Integrated Data Access***

**IDA ALV** – это расширение библиотеки Simple ALV - SALV которое *позволяет обрабатывать большие массивы данных не передавая сразу все данные на AS*. Вам не требуется заполнять внутреннюю таблицу для передачи её в salv класс - достаточно указать data source.

# ‘where like’ in CDS View

# [Содержание](#Содержание)

Коротко

where kunnr like ($parameters.p\_payer || '%')

where unit like (**:**p || '%');

I have the following CDS

@AbapCatalog.sqlViewName: 'ZAMPAYERINFO'

@AbapCatalog.compiler.compareFilter: true

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: 'Read payer information'

define view **zam\_payer\_info**

with parameters p\_payer: abap.char(10)

as select distinct from knvp

join kna1 on knvp.kunnr = kna1.kunnr

{

*key* knvp.kunnr as Payer,

kna1.name1 as Name

} where knvp.kunnr like $parameters.p\_payer

that the compiler complains

*Comparison value of LIKE condition must be a character-type literalZ AM\_PAYER\_INFO (Data Definition)*

As you can see on the CDS code, I am using character-type.

I understand, what LIKE statement do, for example

where knvp.kunnr like 'a%'

Search all kunnr that starts with a. But my problem is, how to combine it with input parameter p\_payer, at the end it should be LIKE *%p\_payer*, where p\_payer will be replace through the value that I passed.

− *You can combine the input parameter and the '%'* wildcard as follows in SQLScript for HANA databases

declare p varchar(10);

P = 'D';

select \* from "A00077387"."ORGANIZATION" where unit like (**:**p || '%');

You see, you can use "||" to concatenate '%' to your parameter

You will need to modify your WHERE clause as follows

knvp.kunnr like ($parameters.p\_payer || '%')

− You may *push back the like statement the ABAP*.

@AbapCatalog.sqlViewName: 'ZAMPAYERINFO'

@AbapCatalog.compiler.compareFilter: true

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: 'Read payer information'

define view zam\_payer\_info

with parameters

p\_payer : abap.char(10) // you can remove this also.

as select distinct from knvp

join kna1 on knvp.kunnr = kna1.kunnr

{

key knvp.kunnr as Payer,

kna1.name1 as Name

}

*//where*

*//knvp.kunnr like 'a%' remove the kunnr where statement here.*

When you selecting from this cds view

data lv\_parameter type kunnr.

lv\_parameter = 'A%'.

select \* from ZAMPAYERINFO into table @data(lt\_payer\_info)

where name like lv\_parameter.

**ABAP CDS - Table Functions**

# [Содержание](#Содержание)

**Client dependency**

The CDS annotation **@ClientDependent** can be used to switch client dependency.

It’s advisable to use an input parameter of the dictionary type **CLNT**, which *needs to be annotated with the annotation* ***@Environment.systemField*** *and the predefined value* ***#CLIENT***. In this case, SELECT passes the correct client ID implicitly.

… @Environment.systemField: #CLIENT implicitly passing the value of ***sy-mandt*** to that parameter.

@ClientDependent: true.

…

with parameters @Environment.systemField: #CLIENT

clnt:abap.clnt, ...

The element list of a client-specific CDS must have an explicit client field with the predefined dictionary type CLNT as its first element

… { CLNT, …

When a client-specific CDS table function is accessed using SELECT without the addition **CLIENT SPECIFIED**, only

1. those rows are selected implicitly from the results set of the function that contain the ID of the current client;

*or*

1. the client specified in the addition **USING CLIENT** in the client field.

It is not advisable to access the current client in the implementation of a CDS table function instead of using an input parameter and the ABAP-specific session variable CLIENT. In this case, the addition USING CLIENT of the Open SQL statement SELECT is ignored.

*If the addition* ***CLIENT SPECIFIED*** *is specified, the column is added to the results set* and is filled with the associated client ID for each row. Before this column can be used in the SELECT statement, a name must be assigned to it after the addition CLIENT SPECIFIED.

When a cross-client CDS table function is accessed using SELECT, an element of the type CLNT doesn’t have a special meaning and is handled like any other element.

**Создание CDS Table function**

# 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/>

# Create Dates Table CDS Table Function using SAP HANA AMDP Class

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

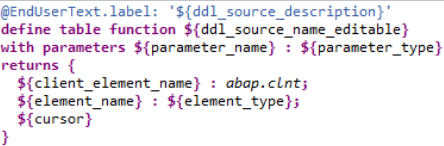
With ABAP 7.50 ABAP CDS also supports CDS table functions as CDS entities.

With ABAP 7.50 AMDP supports also database functions with a **tabular** return value.

CDS table functions cannot be called as functional methods in ABAP- while AMDP-procedures can be called as ABAP methods.

***In order to create a CDS table function, you have two things to do***

1. define it in a CDS DDL source code;



1. implement it in an AMDP method with a  return value.

Пример

@ClientDependent: true  
define table function **DEMO\_CDS\_GET\_SCARR\_SPFLI\_INPCL**  
  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   
    CL\_DEMO\_AMDP\_FUNCTIONS\_INPCL=>GET\_SCARR\_SPFLI\_FOR\_CDS;

After activating the CDS table function you can go on implement the functional AMDP method in an AMDP class, that is a class with the marker interface **IF\_AMDP\_MARKER\_HDB**. ***An AMDP method for a CDS table function*** must be a *static functional method of a static AMDP class* –

CLASS-METHODS **get\_scarr\_spfli\_for\_cds**

              FOR TABLE FUNCTION demo\_cds\_get\_scarr\_spfli\_inpcl.

***The parameter interface*** is implicitly derived from the table function’s definition –

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.

Обращение в Open к SQL CDS table function as data source –

SELECT \* FROM demo\_cds\_get\_scarr\_spfli\_inpcl( carrid = @carrid )  
         INTO TABLE @DATA(result)  
        ##db\_feature\_mode[amdp\_table\_function].

# AMDP class and methods and how to use in an extractor

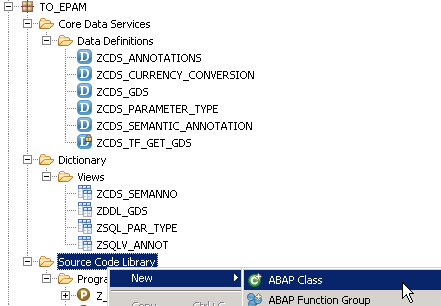
<https://blogs.sap.com/2018/11/27/amdp-class-and-methods-and-how-to-use-in-an-extractor/>

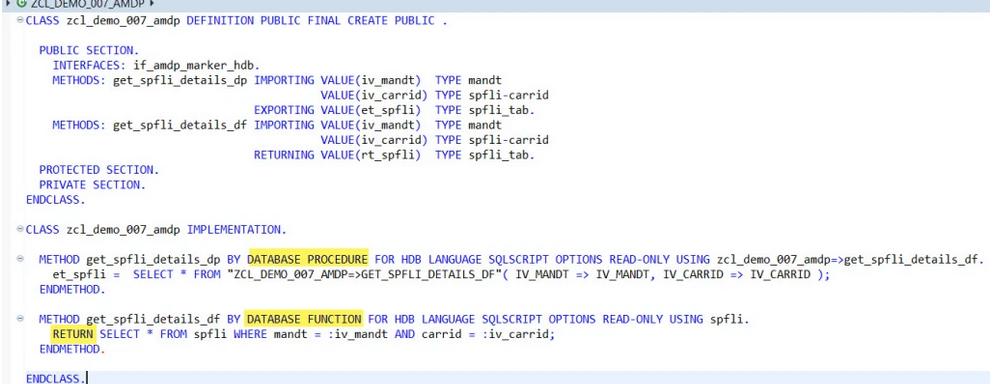
**Создание AMDP класса**

An AMDP class is a global class in the class library that contains one or more of the following tag interfaces –

IF\_AMDP\_MARKER\_HDB - for the SAP HANA database.

The names of the interfaces all start with IF\_AMDP\_MARKER and a suffix indicates the database system for which the ABAP Managed Database Procedures can be implemented in AMDP methods of the class.





class Zимя\_класса definition

public

final

create public .

public section.

INTERFACES: if\_amdp\_marker\_hdb.

methods: имя\_метода importing value(iv\_mandt) type mandt

value(iv\_carrid) type spfli-carrid

exporting value(et\_spfli) type spfli\_tab

protected section.

private section.

ENDCLASS.

CLASS Zимя\_класса IMPLEMENTATION.

method имя\_метода BY DATABASE PROCEDURE FOR HDB

LANGUAGE SQLSCRIPT

OPTIONS READ-ONLY

USING ZCL\_DEMO\_007\_AMDP=>GET\_SPLFLI\_DETAILS\_DF.

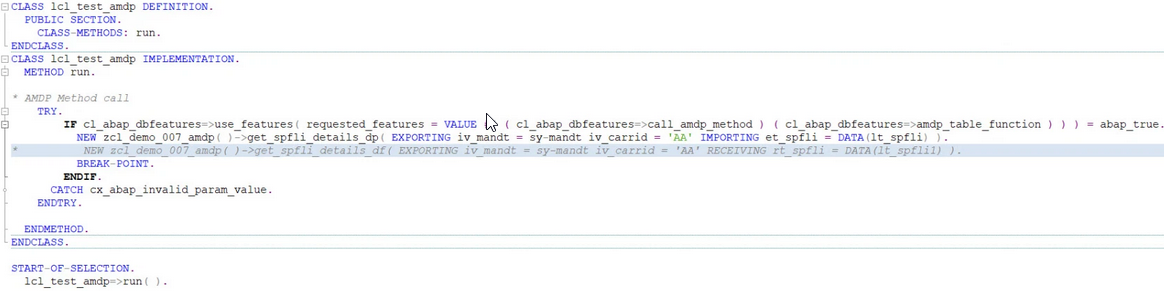
et\_spfli = select \* from “ZCL\_DEMO\_007\_AMDP=>GET\_SPLFLI\_DETAILS\_DF”(iv\_mand => iv\_mand,

iv\_carrid => iv\_carrid);

endmethod.

ENDCLASS.

**Использование CDS Table function в программе**



class ZCL\_USE\_TF\_GET\_GDS definition

public

final

create public .

public section.

class-methods: run.

protected section.

private section.

ENDCLASS.

CLASS ZCL\_USE\_TF\_GET\_GDS IMPLEMENTATION.

method run.

try

if cl\_abap\_dbfeatures=>use\_features( requested\_features = VALUE # ( ( cl\_abap\_dbfeatures=>call\_amdp\_method ) ( cl\_abap\_dbfeatures=>amdp\_table\_function ) ) ) = abap\_true.

New zcl\_amdp\_functions\_inpcl ( ) ->

endif.

endtry.

endmethod.

ENDCLASS.

--select sum("/BIC/BSSLS000"), "/BIC/BEMIT000", inc."/BIC/BSLST000", t\_tp.txtlg

**select** **sum**("/BIC/BSREC000"), "/BIC/BEMIT000", inc."/BIC/BSLST000", t\_tp.txtlg

--from "/BIC/ABDREFGDS2" as inc

--from "/BIC/ABDREC0002" as inc

**from** "/BIC/ABDREC0012" **as** inc

--from "/BIC/ABDINGDS02" as inc

--from "/BIC/ABDIN00002" as inc

**inner** **join** "/BIC/TBSLST000" **as** t\_tp

**on** t\_tp."/BIC/BSLST000" = inc."/BIC/BSLST000"

**where** **year**(date0)=2020 **and** **month**(date0)=07 **and** "/BIC/BEMIT000"=020

**group** **by** "/BIC/BEMIT000", inc."/BIC/BSLST000", t\_tp.txtlg

**order** **by** inc."/BIC/BSLST000"

**Вопросы**

# [Содержание](#Содержание)

1. Обращение к CDS из SQL Console.
2. Вывод клиента в ABAP программе обращающейся к client-dependant CDS в котором клиент не указан в select?
3. ~~+ Обращение к CDS из ABAP Open SQL?~~

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

*или*

SELECT \* FROM zcds\_gds( p\_emit = ‘0020’ ) INTO TABLE @DATA(result1) UP TO 1000 ROWS.

1. Как из CDS обратиться к системным полям – например - SY-DATUM?
2. Association
3. Обращение к CDS DB View из Composite Provider
4. ~~+ Как из одного CDS обратиться к другому CDS с параметрами?~~

as select from zcds\_gds(p\_emit:'0020') as gds

1. IDA ALV

# Ресурсы

# [Содержание](#Содержание)

# BW Query on CDS View, OData from BW and value of BW Query in S/4HANA

<https://blogs.sap.com/2018/08/08/bw-query-on-cds-view-odata-from-bw-and-value-of-bw-query-in-s4hana/>

**CDS View**

# [Содержание](#Содержание)

# *from* ABAP CDS (Core Data Services)

<https://abap-blog.ru/slovar/abap-cds-core-data-services/>

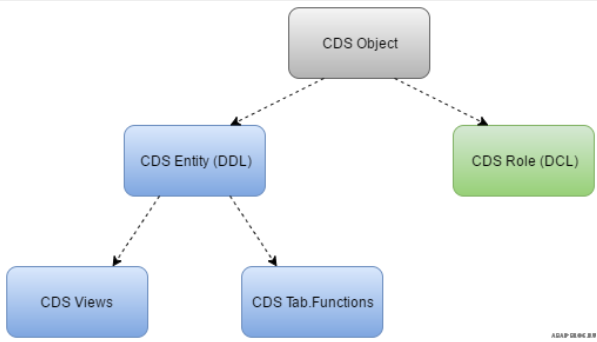
# *CDS Entity ABAP CDS*

# *HANA CDS*

Кроме **ABAP CDS** есть отдельная реализация для HANA – **HANA CDS**. Horst Keller в своём [блоге](http://scn.sap.com/community/abap/blog/2015/07/20/cds--one-model-two-flavors)  подробно рассматривает разницу в обоих реализациях.

ABAP CDS поддерживается сервером приложений ABAP вне зависимости от того какая СУБД будет им использована - однако некоторые специфические вещи могут и не поддерживаться

* в ABAP 7.4. нет поддержки параметров у CDS для всех СУБД [появилась в ABAP 7.5];
* скорость работы  CDS на HANA и Oracle может быть [различной](http://scn.sap.com/community/abap/blog/2016/02/08/a-kill-to-an-abap-cds-view).



**CDS Entity** – *CDS объект управляемый ABAP словарём*.

Св-ва **CDS Entity**

* Объект *не переносится транспортом* - генерируется в системе во время активации исходного кода DDL с которым он связан.
* Хранится в таблице *TADIR* с ключом R3TR *STOB* <EntityName> - где STOB – Structured object.
* В ABAP 7.4 к CDS Entity относится лишь *CDS View*; в 7.5 - добавили [*CDS Table function*](http://scn.sap.com/community/abap/blog/2016/02/24/how-to-use-amdp-function-implementation-for-a-cds-table-function-as-a-data-source-in-cds-views).
* На CDS View можно ссылаться *при объявлении ABAP переменных*, но нельзя использовать при *описании словарных объектов* [*type of* | *include* | *row type of*].

**CDS Database View** – объект словаря; как и CDS View - генерируется во время активации DDL объекта.

**Св-ва CDS DB View**

* Объект в *TADIR* можно найти по ключу R3TR *VIEW*.
* По сути - это классический read-only database view из словаря.
* Просмотреть данный view можно через SE11; отредактировать - только в ADT.
* В ABAP можно считать данные напрямую из CDS DB View - однако данный способ считается устаревшим в ABAP 7.5.
* В отличие от CDS View - CDS DB View можно использовать *при описании объектов словаря*.

**Текстовый и графический редакторы CDS**

# [Содержание](#Содержание)

# Начиная с ABAP 7.50 SP01, кроме текстового редактора - доступен [графический редактор CDS](http://scn.sap.com/community/abap/eclipse/blog/2015/10/28/representing-core-data-services-concepts-in-graphical-form).

# Просмотреть данные можно и *из SQL консоли ADT*

# 

# ABAP CDS

# [Содержание](#Содержание)

# The CDS entity

# 

# can be

# used as a *data source* in other CDS entities;

# specified after the *type* addition in ABAP [*the components of the structured type* are specified using the structure componentselector - *cds\_entity-comp*];

# can’t be

# accessed i*n* *Native SQL*;

# used as a *data type* in ABAP Dictionary [*type of* | *include* | *row type of*].

The possible CDS entities and the associated DDL statements

* CDS views
* CDS table functions
* DDL statements

# Имена

* The CDS *source code* doesn’t need to have the ***same name*** as the CDS *entity* [but this is recommended].

# In activations, the *names* of CDS *source code*, the CDS *entity*, and the CDS *database view* construct an *indivisible defining unit* ⇒ none of these names can be modified afterwards.

# When the CDS data definition of a CDS view is activated - two ABAP Dictionary objects area created

* The CDS ***entity*** - its name is defined after *DEFINE VIEW*.
* A CDS ***database view*** [read-only classic database view] - its name is defined in the annotation *@AbapCatalog.sqlViewName.*

CDS database view

# can be

* used in ABAP Dictionary *like* any classic ***structure*** [*type of* | *include* | *row type of*].

# should not be

* used in ***ABAP***.

# can’t be

* used in reading ***Open SQL*** statements [is forbidden from Release 7.50].

## 

### **ABAP CDS vs. HANA CDS**

# [Содержание](#Содержание)

***You will not be able to copy DDL sources*** from HANA to ABAP and vice versa *without modifications*.

While HANA CDS has to function on SAP HANA only, ABAP CDS is open ⇒ there are some restrictions for ABAP CDS that do not exist for HANA CDS [in the same way as Open SQL is more restricted than Native SQL].

A good example are ***built-in functions***. A built-in function like [CURRENCY\_CONVERSION](http://help.sap.com/abapdocu_740/en/index.htm?file=abencds_f1_conversion_functions.htm) offered in ABAP CDS must be available on any database platform and  - very important - must behave in the same way on all platforms. Same for expressions like arithmetic expressions, aggregates or the CAST expression. Before releasing such a functionality in ABAP CDS, all platforms have to participate. And thats why ABAP CDS cannot offer all [SQL Functions](http://help.sap.com/saphelp_hanaplatform/helpdata/en/20/a61f29751910149f99f0300dd95cd9/content.htm?frameset=/en/2e/1ef8b4f4554739959886e55d4c127b/frameset.htm&current_toc=/en/2e/1ef8b4f4554739959886e55d4c127b/plain.htm&node_id=18&show_children=false) of SAP HANA yet.

On the other hand,it is open! And there are also ABAP specialties like client handling or table buffering that are not available in HANA CDS, but supported in ABAP CDS.

## **Classification of CDS views**

# [Содержание](#Содержание)

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.

@ObjectModel.dataCategory

* ***#TEXT*** - for text views;
* ***#HIERARCHY*** - for hierarchy views.

# Общая инф.

# [Содержание](#Содержание)

# CDS views *with input parameters* support only full extraction.

# Data Source on CDS view -

# CDS view entities *don’t need* to carry the @*AbapCatalog.sqlViewName* anymore ⇒ the name to search for will be the ***CDS view name*** itself [for classic**CDS views** - it will be the ***@AbapCatalog.sqlViewName***].

# *Buffering*

# [Содержание](#Содержание)

# Annotations

# *@AbapCatalog.buffering.status* - to define whether and how SAP buffering is allowed for a CDS view in ABAP CDS.

# *@AbapCatalog.buffering.type* - define the buffering type.

# *@AbapCatalog.buffering.numberOfKeyFields* - if generic buffering is used, define the number of key fields involved.

# Notes

# CDS view can be buffered only if it does not contain any views [database views or CDS views] as data sources.

# In SAP buffering of CDS views, only the key fields of the CDS database view are relevant and not the key elements of the CDS entity defined using KEY.

# Example *// Enables full buffering for a CDS view.*

# @AbapCatalog.sqlViewName: '...'

# @AbapCatalog.Buffering.status: *#ACTIVE*

# @AbapCatalog.Buffering.type: *#FULL*

# define view ...

# as select from ...

# { ... }

# 

# Create a Basic CDS view

# <https://blogs.sap.com/2019/10/22/part2.-create-a-basic-cds-view/>

# [Содержание](#Содержание)

# There has to be at least one ABAP project in the workspace. Open the *ABAP Project Creation Wizard* in order to create a new ABAP project.

1. ***Create an ABAP project*** within Eclipse by logging in to S4 HANA system.

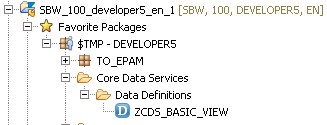
*Window\Perspective\Open Perspective\Other…\ABAP*.

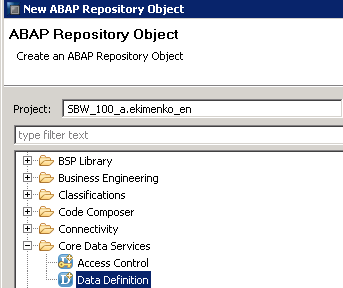
Then

*File\New\ABAP project*

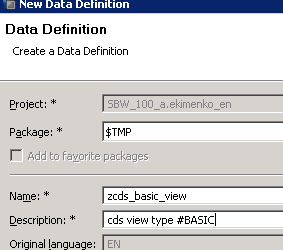


1. Right click on your username under Local Objects and choose *New\Other ABAP Repository Objects\Core Data Services\Data Definition*.





1. Создать ***zcds\_basic\_view*** /Description *CDS view type #BASIC*/



Аннотации

***@AbapCatalog.sqlViewName***: *'sql\_view\_name'* - *DDIC SQL view*.

The *CDS view* name and the *SQL view* name can NOT be same ⇒ choose a logical name, probably something similar to the CDS view name to keep it consistent.

***@AbapCatalog.compiler.compareFilter***: *true* - this first compare the filter conditions and if they match then only the data is fetched.

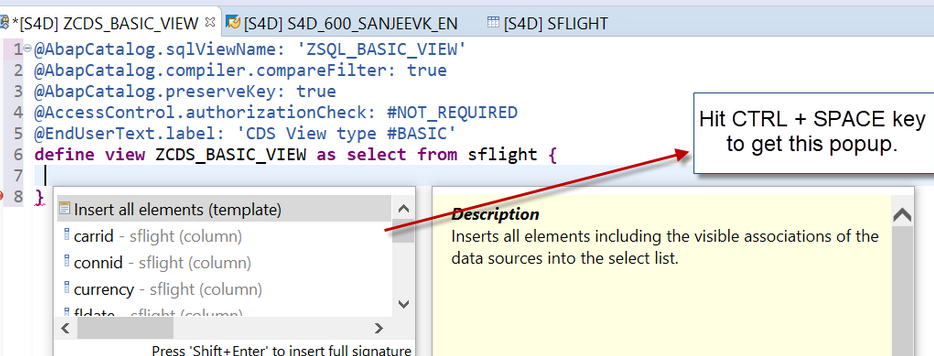
* If the CDS view has join conditions, they are executed only after the filter conditions are matched.
* In DDIC views data if first fetched and then filtered.

***@AbapCatalog.preserveKey*: *true*** - any DB table can have multiple keys defined and you might not want those keys to be the key fields of your view ⇒

* *true* - the only fields you define as Key Fields within your CDS view by adding word *key* in front of those fields will be the Key fields for the CDS view and the DDIC SQL view which gets generated;
* *false* - the DB table key fields will be defined as Key fields for the views as well, regardless of you adding word ‘Key’ in front of fewer fields.

**@*AccessControl.authorizationCheck***: *#NOT\_REQUIRED*-when CDS view is ready to be rolled out to a bigger audience within or outside the organization, it’s very important to restrict the data based on the authority.

***@EndUserText.label***: ‘CDS View type #BASIC’ -override the DB table business label and provides what you specify with this annotation.



# The field *Client* or *MANDT* disabled due to the reason that this CDS view is created in the ABAP system and the client is handled by the system itself.

@AbapCatalog.sqlViewName: '***zsql\_basic\_view***'

@AbapCatalog.compiler.compareFilter: true

@AbapCatalog.preserveKey: *true*

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: 'CDS view type #BASIC'

define view ***zcds\_basic\_view*** as select from *sflight* {

*key* carrid,

seatsmax\_b,

seatsocc\_b,

seatsmax\_f,

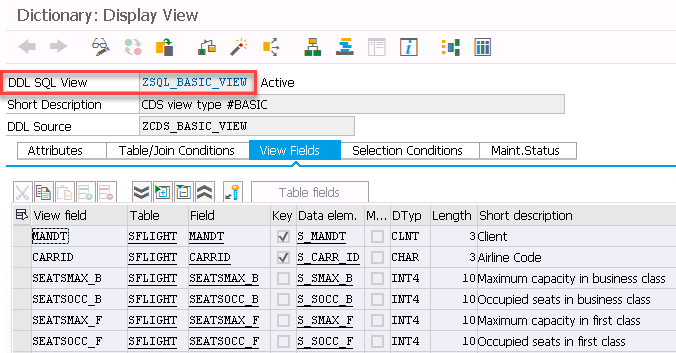
seatsocc\_f

# }

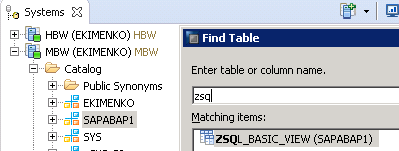
1. Now ***activate the CDS view***.

Two more objects get created when the CDS view is activated - a *DDIC SQL view* and a *HANA view*.

4.1 Goto Tcode ***SE11*** and check the *DDIC SQL view*.



4.2 Check the *HANA view* in Eclipse or HANA Studio platform.



SELECT TOP 1000 \* FROM "SAPABAP1"."*ZSQL\_BASIC\_VIEW*"

# CDS View with Expressions and Session Variables

# <https://blogs.sap.com/2019/10/23/part3.-cds-view-with-expressions-and-session-variables/>

# [Содержание](#Содержание)

# Let’s create a simple CDS view with an Expression to calculate the *available seats* in business class from table *SFLIGHT*

@AbapCatalog.sqlViewName: '***zsql\_view\_exprsn***' *// no more than 16 characters*

@AbapCatalog.compiler.compareFilter: true

@AbapCatalog.preserveKey: true

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: 'CDS view with Expression'

define view ***zcds\_view\_with\_expression*** as select from *sflight* {

*key* carrid,

seatsmax\_b,

seatsocc\_b,

*(seatsmax\_b - seatsocc\_b)* as Seats\_Avlbl

# }

# SELECT TOP 1000 \* FROM "SAPABAP1"."*ZSQL\_VIEW\_EXPRSN*"

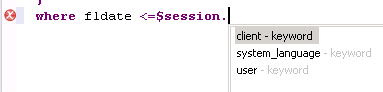
***WHERE* clause and SESSION variables**

Similar to ABAP session variables - *sy-datum* /current Date/, *sy-uzeit* /current Time/ etc., CDS views editor also has session variables which can provide these values.

where fldate <=$session.*system\_date*

*Rem*

*system\_date – нет*; есть только



### **Code Pushdown**

<https://abap-blog.ru/osnovy-abap/slovar/abap-cds-core-data-services/>

# [Содержание](#Содержание)

***Code Pushdown*** подразумевает собой перенос логики расчёта на уровень СУБД.

Сам перенос расчёта подразумевает широкое использование выражений SQL в CDS View, которые имеют больше возможностей, чем в OpenSQL [в планах развития OpenSQL поддержка всех функций CDS].

Стандартный пример CDS View с SQL функциями

@AbapCatalog.sqlViewName: 'DEMO\_CDS\_SQLFUNC'

define view **demo\_cds\_sql\_functions**

as

select from *demo\_expressions*

{ abs(num1) as abs\_num1,

  ceil(fltp1) as ceil\_fltp1,

  floor(dec1) as floor\_dec1,

  div(num1, num2) as div\_num1\_num2,

  mod(num1, num2) as mod\_num1\_num2,

  division(dec2, dec3, 3) as dvision\_dec2\_dec3,

  round(dec3, 2) as round\_dec3,

  concat(char1, char2) as concat\_char1\_char2,

  lpad(char1, 10, 'x') as lpad\_char1,

  replace(char2, 'GHI', 'XXX' ) as replace\_char2,

  substring(char2, 2, 3) as substring\_char2

}

Описание функций можно найти в официальной [документации](http://help.sap.com/abapdocu_740/en/abencds_f1_builtin_functions.htm) или по F1.

Пример использования функции [конвертации валюты](http://help.sap.com/abapdocu_740/en/abencds_f1_conversion_functions.htm#!ABAP_VARIANT_1@1@)

@AbapCatalog.sqlViewName: 'DEMO\_CDS\_CRRCONV'

 define view **demo\_cds\_currency\_conversion**

with parameters *to\_currency*:abap.cuky(5),

*exc\_date*:abap.dats

as

select from *demo\_prices*

{ id,

  currency\_conversion( amount => amount,

                           source\_currency => currency,

                           round => 'X',

                           target\_currency => :to\_currency,

                           exchange\_rate\_date => :exc\_date,

                           error\_handling => 'SET\_TO\_NULL' ) as amount,

  :to\_currency as currency

}

Кроме непосредственно функций можно так же использовать литералы, арифметические выражения и условный оператор CASE.

Пример с CASE

@AbapCatalog.sqlViewName: 'DEMO\_CDS\_SCASE'

define view **demo\_cds\_searched\_case**

as

select from *spfli*

{ key carrid,

  key connid,

  distance,

  distid,

  case when distance >= 2000 then 'long-haul flight'

         when distance >= 1000 and distance <  2000 then 'medium-haul flight'

        when distance <  1000 then 'short-haul flight' else 'error'

end as flight\_type

}

where distid = 'MI'

В ***ABAP 7.5*** набор возможностей CDS [еще шире](http://help.sap.com/abapdocu_750/en/abencds_f1_builtin_functions.htm).

# *Читаем Calculation View из ABAP через CDS*

<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/>

# *HANA CDS Views vs Calculation Views vs Table Functions*

# <https://stackoverflow.com/questions/59682110/hana-cds-views-vs-calculation-views-vs-table-functions>

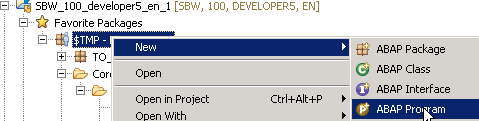
The main ***difference between a CDS view and a CDS table function*** is that a *CDS view* represents a *virtual table* that can be queried directly, while a *CDS table function* represents a *database function* that returns a result set. CDS views are suitable for complex queries and reporting scenarios, whereas CDS table functions are used when you need to execute a reusable database function and retrieve tabular data as a result.

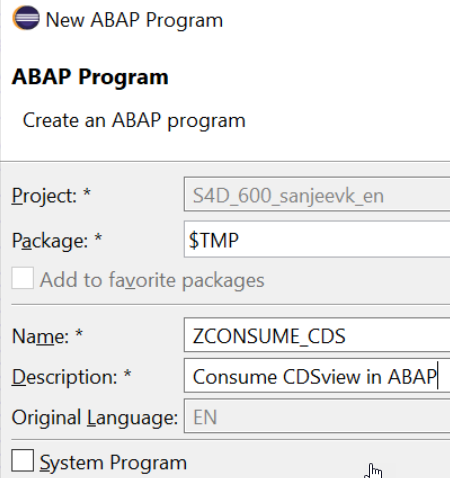
# Consume CDS View in an ABAP Program

<https://blogs.sap.com/2019/10/24/part4.-consume-cds-view-in-an-abap-program/>

# [Содержание](#Содержание)

Within ABAP perspective in Eclise or HANA Studio, right click on your user under Lcal Objects within an ABAP project to create an ABAP program.



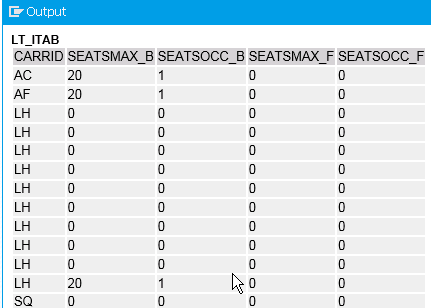


REPORT ***zconsume\_cds***.

select \* from *ZCDS\_BASIC\_VIEW* into table @data(*lt\_itab*).

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

Hit ***F8*** or *Execute* the program to check the output.



# CDS View Extension

# <https://blogs.sap.com/2019/10/25/part5.-cds-view-extension/>

# [Содержание](#Содержание)

Instead of creating new CDS view from scratch, we can use the concept of CDS View extension and re-use the previous view we created and extend it with 3 new fields.

* 1. ***Create a CDS view with Template ‘Extend View’***

# …

# Let’s extend the view we created before - *ZCDS\_BASIC\_VIEW* - add the fields you would like to extend.

@AbapCatalog.sqlViewAppendName: '***ZSQL\_EXTEND\_VIEW***'

@EndUserText.label: 'CDS View extension concept'

*extend view* *ZCDS\_BASIC\_VIEW* with ***zcds\_extend\_view*** {

seatsocc,

fldate,

paymentsum

# }

# Hit *F8* and check the output.  You can see that the 3 new fields are available in this view together with all other fields of the view we extended.

# C:\Users\ABD50~1.EKI\AppData\Local\Temp\SNAGHTML557359d.PNG

# Rem

# Once you save and activate the above code snippet, navigate to your old CDS view /взят из другого примера/. You will be able the see the *spiral symbol* indicating that the extension is successful. Now hit the run icon of the old CDS view

# 

# Further, we can also consume this CDS view in our ABAP report. Navigate to New-> Other ABAP Repository Object->ABAP Program. Or you can also go to SAP LOGON->SE38

# 

# Parameters in a CDS view

# <https://blogs.sap.com/2019/10/28/part6.-parameters-in-a-cds-view/>

# [Содержание](#Содержание)

# …

# To the *with parameters* clause - add the parameter name before colon and the data type after colon.  For *Data Type* you can either add the data element directly or add ***primitive data type*** like *abap*.<data ype>(<len>).

@AbapCatalog.sqlViewName: '***zsql\_view\_param***'

@AbapCatalog.compiler.compareFilter: true

@AbapCatalog.preserveKey: true

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: 'CDS View with Parameters'

define view ***zcds\_view\_param***

with parameters *in\_curr* : *abap.char(3)*

as select from *sflight* {

*key* carrid,

price,

currency,

planetype,

seatsmax,

seatsocc\_b

}

# where currency = *:in\_curr*

# Active the CDS view now and hit *F8* to execute. Checkout the user input pop up to enter the currency code.

# 

# 

**Consume CDS view with parameters in ABAP program**

Create a new ABAP program ZCONSUME\_CDS\_PARAM

REPORT ***zconsume\_cds\_param***.

data: *lv\_curr* type *S\_CURRCODE*.

cl\_demo\_input=>new( )->add\_field( changing field = *lv\_curr* )->request( ).

select \* from *ZCDS\_VIEW\_PARAM*( in\_curr = *@lv\_curr* ) into table @data(lt\_tab).

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

# 

# 

# ****How to search for SAP standard CDS Views****

# <https://blogs.sap.com/2019/10/29/part7.-how-to-search-for-sap-standard-cds-views/>

# [Содержание](#Содержание)

# Joins and Associations

# <https://blogs.sap.com/2019/10/30/part8.-cds-views-joins-and-associations/>

# [Содержание](#Содержание)

# **Inner join**

# 

@AbapCatalog.sqlViewName: ***'zsql\_view\_join***'

@AbapCatalog.compiler.compareFilter: true

@AbapCatalog.preserveKey: *true*

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: 'CDS View with Join'

define view ***zcds\_view\_join*** as select from *sflight* as sf

*inner join* *spfli* as sp on sp.carrid = sf.carrid {

sf.carrid,

sf.connid,

sf.fldate,

sf.price,

sf.seatsocc\_b, sf.seatsmax\_f, sf.seatsocc\_f,

sp.airpfrom,

sp.airpto

# }

# Associations

# CDS view are not directly accessed by the business users - rather they will be consumed by *ABAP program* | *Fiori apps* | *BI front end tools*.

# If you have created a CDS view with JOINS on 5 different tables then this JOIN conditions will be executed every time this CDS view is triggered [even though the business user is looking at only fields from 2 tables].  To overcome this problem, SAP did an enhancement to this SQL way of getting data and beautifully developed the concept of *Associations* - data will be fetched only when user want to see it.

Associations are kind of Joins but these are *JOINS ON-DEMAND* i.e. they will only be triggered when user would access the required data which needs the Association of tables.

For example, your CDS view has 4 Associations configured and user is fetching data for only 2 tables, the Association on other 2 tables will not be triggered.

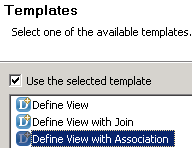
Associations are defined with ***Cardinality*** - association[*<cardinality>*]

There are 4 types of Cardinality

* 0..1
* 0..n or 0..\*
* 1..0
* 1..n or 1..\*

*Rem*

If you are confused on what kind of association to configure in your CDS view then you can apply a rule of thumb - always use *association[1]*-this will always trigger an *OUTER join* and will work in all the cases.



@AbapCatalog.sqlViewName: '***zsql\_view\_asstn***'

@AbapCatalog.compiler.compareFilter: true

@AbapCatalog.preserveKey: *true*

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: 'CDS View with Association concept'

define view ***zcds\_view\_association*** as select from *sflight* as sf

*association [1]* to *spfli* as \_flights on sf.carrid = \_flights.carrid { *// OUTER join*

key sf.carrid,

sf.connid,

sf.fldate,

sf.price,

sf.seatsocc\_b, sf.seatsmax\_f, sf.seatsocc\_f,

*//\_flights.airpfrom, \_flights.airpto,*

\_flights *// Make association public*

}

# Check the SQL CREATE statement

# 

# Note that a LEFT OUTER *MANY TO ONE* JOIN is created /*association [1]* -> *\*..1*/.  The join created because we selected individual fields instead of exposing the whole Association.

CREATE VIEW "ZSQL\_VIEW\_ASSTN" AS SELECT

"SF"."MANDT" AS "MANDT",

"SF"."CARRID",

"SF"."CONNID",

"SF"."FLDATE",

"SF"."PRICE",

"SF"."SEATSOCC\_B", "SF"."SEATSMAX\_F", "SF"."SEATSOCC\_F"

*//,* *"=A0"."AIRPFROM", "=A0"."AIRPTO"*

FROM "SFLIGHT" "SF"

*//LEFT OUTER MANY TO ONE JOIN "SPFLI" "=A0"*

*//ON ( "SF"."MANDT" = "=A0"."MANDT" AND "SF"."CARRID" = "=A0"."CARRID" )*

# Execute and note that no data from table SPFLI is displayed as no join is yet performed.

# Now if you want to see further Airport information for CARRID ‘AA’ from table SPFLI - right click on any row with CARRID ‘AA’ and choose *Follow Association*

# 

# Another pop-up window will open - click the Association type .

# 

**OData service creating**

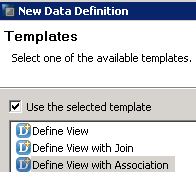
<https://blogs.sap.com/2019/10/31/part9.-cds-view-odata-service-publishing-to-the-outer-world/>

# [Содержание](#Содержание)

To publish a CDS view with OData Service we have to use annotation ***@OData.publish: true*** which will create an OData Service and we must register it within SAP system via GUI interface.  Once registered, CDS view is ready to be consumed in any application within or outside SAP environment.

***Step 1*** – Создание CDS view

We will use the concept of *Association* again.



@AbapCatalog.sqlViewName: '***zsql\_odata\_view***'

@AbapCatalog.compiler.compareFilter: true

@AbapCatalog.preserveKey: *true*

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: 'CDS view with OData Service->>FIORI app'

define view ***zcds\_odata\_view*** as select from *sflight* as sf

association [1] to *spfli* as \_flights on sf.carrid = \_flights.carrid

{

*key* sf.carrid,

sf.connid,

sf.fldate,

sf.price,

sf.seatsocc\_b, sf.seatsmax\_f, sf.seatsocc\_f,

\_flights *// Make association public*

# }

# Add the ***OData* annotation** and activate again

# @*OData.publish*: true

# Hover over the mouse pointer on the yellow icon showed up in front of *@OData* annotation. It will show the *service name* which gets created and indicates that service is not yet active. Copy the service name to register and activate. This is a mandatory one-time activity we need to perform so that our CDS view can be published with OData.

# *Rem*

# The *service name* generated by the system is almost like your CDS view name with ****\_CDS****

# 

***Step 2*** – Регистрация сервиса

Login to SAP GUI interface to register the service. You can hot CTRL+6 to login within ADT or login separately via SAP Logon pad.

Goto tcode **/n/IWFND/MAINT\_SERVICE** and click *Add Service* button.

# Choose *LOCAL* as System Alias and pass the service name - *ZCDS\_ODATA\_VIEW\_CDS* we copied from CDS view to *External Service Name*.

# Hit Enter.

# 

# Select the Service and click *Add Selected Services*.

# 

# Now pass the *‘package’* to transport OR select *Local Object* if no transport is required.

# 

# Hit *Continue*.

# To check if Service is registered, Go back screen and filter it

# 

# Select the Service and click on *Call Browser*

# 

A browser window will open and you have to enter your SAP credentials - username/psw.  You can see the service code.  Note the CDS View/Entity name and the Association Table name *SPFLI* also appears.

<http://bl-sap-sbw.it.beloil.by:8013/sap/opu/odata/sap/ZCDS_ODATA_VIEW_CDS/?$format=xml>

<app:service *xmlns:app*="http://www.w3.org/2007/app"

*xmlns:atom*="http://www.w3.org/2005/Atom"

*xmlns:m*="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"

*xmlns:sap*="http://www.sap.com/Protocols/SAPData"

xml:lang="en"

*xml:base*="http://BL-SAP-SBW.IT.BELOIL.BY:8013/sap/opu/odata/sap/*ZCDS\_ODATA\_VIEW\_CDS*/">

<app:workspace>

<atom:title type="text">Data</atom:title>

<app:collection sap:creatable="false" sap:updatable="false" sap:deletable="false" sap:content-version="1"

href="*zcds\_odata\_view*">

<atom:title type="text">zcds\_odata\_view</atom:title>

<sap:member-title>zcds\_odata\_viewType</sap:member-title>

</app:collection>

<app:collection sap:creatable="false" sap:updatable="false" sap:deletable="false" sap:content-version="1"

href="*SPFLI*">

<atom:title type="text">SPFLI</atom:title>

<sap:member-title>SPFLIType</sap:member-title>

</app:collection>

</app:workspace>

<atom:link rel="*self*"

href="http://BL-SAP-SBW.IT.BELOIL.BY:8013/sap/opu/odata/sap/ZCDS\_ODATA\_VIEW\_CDS/"/>

<atom:link rel="*latest-version*"

href="http://BL-SAP-SBW.IT.BELOIL.BY:8013/sap/opu/odata/sap/ZCDS\_ODATA\_VIEW\_CDS/"/>

</app:service>

To check if OData service returns the data, copy the CDS view name and add it to the link in the browser –

From

[http://bl-sap-sbw.it.beloil.by:8013/sap/opu/odata/sap/ZCDS\_ODATA\_VIEW\_CDS/*?$format=xml*](http://bl-sap-sbw.it.beloil.by:8013/sap/opu/odata/sap/ZCDS_ODATA_VIEW_CDS/?$format=xml)

To

[http://bl-sap-sbw.it.beloil.by:8013/sap/opu/odata/sap/ZCDS\_ODATA\_VIEW\_CDS/*zcds\_odata\_view*](http://bl-sap-sbw.it.beloil.by:8013/sap/opu/odata/sap/ZCDS_ODATA_VIEW_CDS/zcds_odata_view%20).

Note that the data displayed is only from table *sflight* - no join is yet performed and no data from table *spfli* is displayed yet.

<entry>

<id>[http://BL-SAP-SBW.IT.BELOIL.BY:8013/sap/opu/odata/sap/ZCDS\_ODATA\_VIEW\_CDS/ zcds\_odata\_view('AF')](http://BL-SAP-SBW.IT.BELOIL.BY:8013/sap/opu/odata/sap/ZCDS_ODATA_VIEW_CDS/%20zcds_odata_view('AF'))</id>

<title type="text">zcds\_odata\_view('*AF*')</title>

<updated>2023-06-21T11:36:27Z</updated>

<category term="ZCDS\_ODATA\_VIEW\_CDS.zcds\_odata\_viewType"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>

<link href=*"zcds\_odata\_view('AF')"* rel="self" title="zcds\_odata\_viewType"/>

<link href*="zcds\_odata\_view('AF')/to\_flights"*

rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/to\_flights"

type="application/atom+xml;type=entry" title="to\_flights"/>

<content type="application/xml">

<m:properties xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"

xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">

<d:carrid>AF</d:carrid>

<d:connid>0820</d:connid>

<d:fldate>2002-12-23T00:00:00</d:fldate>

<d:price>2222.000</d:price>

<d:seatsocc\_b>1</d:seatsocc\_b>

<d:seatsmax\_f>0</d:seatsmax\_f>

<d:seatsocc\_f>0</d:seatsocc\_f>

</m:properties>

</content>

</entry>

Check the highlighted link - <link href*="zcds\_odata\_view('AF')/to\_flights"* … - this is the link to the table *spfli* to display data *On-Demand* as we have applied the concept of *associations* within the CDS view.  Copy and paste this link in the browser window by replacing it with the CDS view name as below

From

[http://bl-sap-sbw.it.beloil.by:8013/sap/opu/odata/sap/ZCDS\_ODATA\_VIEW\_CDS/*zcds\_odata\_view*](http://bl-sap-sbw.it.beloil.by:8013/sap/opu/odata/sap/ZCDS_ODATA_VIEW_CDS/zcds_odata_view%20).

To

<http://bl-sap-sbw.it.beloil.by:8013/sap/opu/odata/sap/ZCDS_ODATA_VIEW_CDS/>*zcds\_odata\_view('AF')/to\_flights*.

Check that the data from *spfli* is now displayed.

# VDM Annotation

# <https://blogs.sap.com/2019/11/04/part10.-cds-views-vdm-annotation/>

# [Содержание](#Содержание)

Annotations types are used while configuring CDS views

* ***@OData*, *@UI*** - used for FIORI applications.
* ***@Analytics*** - used to consume CDS view into Analytical tools like Bex, Lumira, Webi, Analysis for Office.
* *dataCategory*: #DIMENSION | #FACT | #CUBE
* *dataExtraction.enabled*: true | false
* *dataExtraction.delta.byElement.name*: elementName as String
* *dataExtraction.delta.byElement.maxDelayInSeconds*: the default is 1800 s [between taking the time stamp and the successful database commit].
* *dataExtraction.delta.byElement.detectDeletedRecords*: - if a key combination does not occur in the view anymore, this will automatically generate a delete image in the extracted data.
* *dataExtraction.delta.byElement.ignoreDeletionAfterDays*: - The extraction will ignore deleted records if they are older than the specified number of days. The main purpose is archiving.
* *hidden*: true - the view cannot be consumed by analytic clients. The default is true, if this annotation is used.
* *planning.enabled*: An input-enabled query provides writeback capabilities and can be used in planning scenarios.
* *query*: true - The query view will be exposed to the analytic manager. The default is true, if this annotation is used.
* *writeBack.className*: name of an ABAP class, which implements the interface IF\_RODPS\_ODP\_WRITEBACK. Views with Analytics.dataCategory: #AGGREGATIONLEVEL on top of this type of view can be used for planning scenarios. The ABAP class is used for saving the data, authorization checks and enqueuing.
* ***@ObjectModel*** - used for Transactional applications using *BOPF*.
* ***@Semantics*** - used for S4HANA Embedded Analytics.

You can find detailed information on all type of Annotations here

<https://help.sap.com/viewer/cc0c305d2fab47bd808adcad3ca7ee9d/7.5.9/en-US/630ce9b386b84e80bfade96779fbaeec.html>

***@VDM***

To standardize the CDS view technical development, SAP came up with the Best Practices with @VDM annotation. This annotation has absolutely NO impact on the behavior or output of the CDS view rather it just show below features

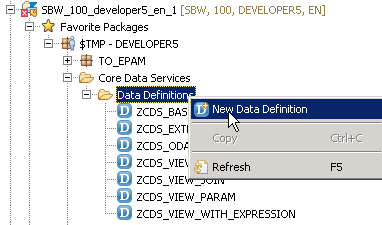
* What *kind of data* the CDS view expose.
* What is the *category* of the CDS view.

@VDM annotation is further broken down into the type of views.  There are 4 type of CDS views can be developed

1. ***BASIC***  - are developed to expose the Master Data; they are also called *INTERFACE views* and for naming standards SAP use ‘I’ in between of the view name - *\*\_I\_\**.
2. ***COMPOSITE***-are configures as an Association of Master data sets or Master data and Transactional data. For naming standards, SAP use ‘CO’ in between of the view name - ***\*\_CO\_\****.
3. ***CONSUMPTION*** - are created as the last layer of CDS analytical Model.  These are the final views ready for consumption by the UI tools.  For naming standards, SAP start the view name with ‘C’ - ***C\_\****.
4. ***PRIVATE*** - are private to SAP and must not be Used or Created by anyone other than SAP.  SAP can change their configuration without any notice to the customers.  For naming standards, SAP start the view name with ‘P’ - ***P\_\****.

Let’s create a VDM based CDS view to show case how this looks

Step 1 - Create a new view with ASSOCIATION template.



Step 2 - Copy the code from our previous Association view [to save time].

@AbapCatalog.sqlViewName: '***zsql\_view\_vdm***'

@AbapCatalog.compiler.compareFilter: true

@AbapCatalog.preserveKey: *true*

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: *'CDS View with VDM annotations'*

define view ***zcds\_view\_vdm*** as select from *sflight* as sf

*association [1]* to *spfli* as \_flights on sf.carrid = \_flights.carrid { *// OUTER join*

key sf.carrid,

sf.connid,

sf.fldate,

sf.price,

sf.seatsocc\_b, sf.seatsmax\_f, sf.seatsocc\_f,

*//\_flights.airpfrom, \_flights.airpto,*

\_flights *// Make association public*

}

# Step 3 - Add the VDM and Analytics annotations.

*@VDM*: { viewType: #BASIC}

*@Analytics*: {dataExtraction: {enabled: true}, dataCategory: #DIMENSION} - this annotation

* defines if this CDS view is consumable by the Analytical applications [we can create a custom datasource out of a CDS view by using this annotation];
* tells that we can extract data out of ECC system to BW for example and the Data Category ‘DIMENSION’ defines that it is a Dimension data which could be a Master data set and later this can be combined with the Fact Table or CDS view type #CUBE.

REPORT ***zconsume\_cds***.

select \* from *ZCDS\_VIEW\_VDM* into table @data(lt\_itab).

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

# *F8* и получаем результат

# 

# End to End Data Modeling and Reporting with CDS views

# <https://blogs.sap.com/2019/11/05/part11.end-to-end-data-modeling-and-reporting-with-cds-views/>

# [Содержание](#Содержание)

To create any analytics data model we need DIMENSION Data which are the Master Data sets and FACT data which is a Transaction Data Set.

Dimension Tables

1. SCUSTOM - Customer information.
2. SCARR - Airline information
3. SPFLI - Connection information.

Fact Table

1. SBOOK - Flight Bookings information

***Rem***

For Time Dimension we can use SAP standard Basic Interface view ***I\_CalendarDate*** and for Country and Country text we can use ***I\_Country***.

**Step 1** - Create the first view for Airline information - ***ZCDS\_VIEW\_AIRLN***

@AbapCatalog.sqlViewName: '***ZSQL\_VIEW\_AIRLN***'

@AbapCatalog.compiler.compareFilter: true

@AbapCatalog.preserveKey: true

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: *'Airline Information'*

@VDM: {viewType: *#BASIC*}

@Analytics: {*dataExtraction*: {enabled: *true*}, dataCategory: *#DIMENSION* }

define view ***ZCDS\_VIEW\_AIRLN*** as select from *scarr* {

*key* carrid as Airline,

carrname as AirlineNam e,

*@Semantics.currencyCode*: true

currcode as currency

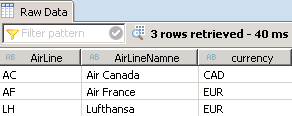
}

где

*@VDM:{viewType: #BASIC}* - this is a Basic Interface view which delivers Master Data [Airline Information in this scenario].

*@Analytics: dataCategory: #DIMENSION* – shows this is Dimension data.

*@Semantics.currencyCode: true*–field *currcode* is treated as Currency field.



**Step 2** - Create second view for Connection information - ***ZCDS\_VIEW\_CONN***

@AbapCatalog.sqlViewName: '***ZSQL\_VIEW\_CONN***'

@AbapCatalog.compiler.compareFilter: true

@AbapCatalog.preserveKey: true

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: *'Connection inf.'*

@VDM: {viewType: *#BASIC*}

@Analytics: {dataExtraction: {enabled: true}, dataCategory: #DIMENSION }

define view ***ZCDS\_VIEW\_CONN*** as select from *spfli*

*association* [1] to *ZCDS\_VIEW\_AIRLN* as \_Airline

on *$projection*.Airline = \_Airline.Airline {

*key* carrid as Airline,

*key* connid as FlConnectNumber,

cityfrom as Source,

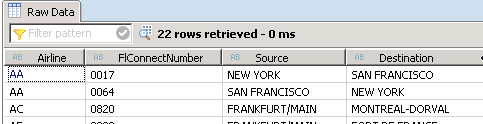
cityto as Destination,

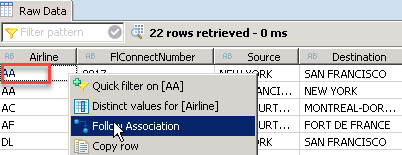
\_Airline // Make association public

}

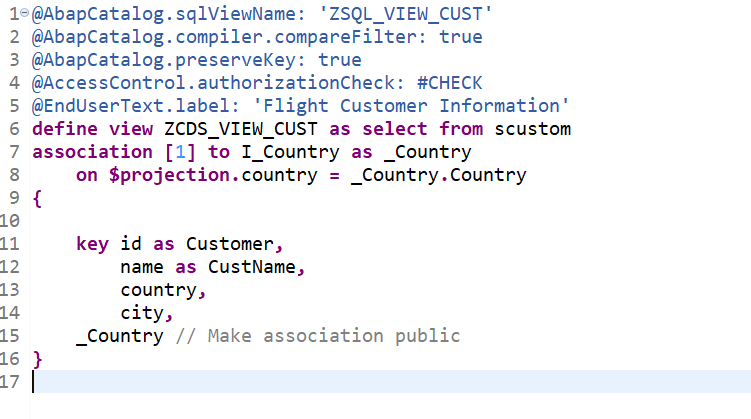
где

***$Projection*** - instead of *source table name or alias*, we can use *$Projection* as well which will take care of Association.  But with $Project, you can *only use the fields which have listed in the CDS view*.



**Step 3** - Create a view for Customer information - ***ZCDS\_VIEW\_CUST***



SAP standard Basic Interface view **I\_Country**

Output: check the Association jump to Country and Country Text via standard view **I\_Country.**

**Step 4** - Create a COMPOSITE view ***ZCDS\_VIEW\_FLBOOK*** to combine all the Dimension views created above with Transaction or Fact data

@AbapCatalog.sqlViewName: '***ZSQL\_VIEW\_FLBOOK***'

@AbapCatalog.compiler.compareFilter: true

@AbapCatalog.preserveKey: true

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: 'Flight Booking Inf.'

@VDM : {viewType: #COMPOSITE}

@Analytics: {dataExtraction: {enabled: true}, dataCategory: #CUBE }

define view ***ZCDS\_VIEW\_FLBOOK*** as select from *sbook*

association [1] to *ZCDS\_VIEW\_AIRLN* as \_Airline on $projection.Airline = \_Airline.Airline

association [1] to *ZCDS\_VIEW\_CONN* as \_Connection on $projection.Airline = \_Connection.Airline and

$projection.FlightConnection = \_Connection.FlConnectNumber

association [1] to *ZCDS\_VIEW\_CUST* as \_Customer on $projection.Customer = \_Customer.Customer {

*-- Dimensions*

*key* carrid as Airline,

*key* connid as FlightConnection,

*key* fldate as FlightDate,

*key* bookid as BookNumber,

*key* customid as Customer,

*key* agencynum as TravelAgency,

\_Customer.country as CustomerCountry,

\_Customer.city as CustomerCity,

*-- Measures*

@EndUserText.label: 'Booking Price'

@DefaultAggregation: #SUM

@Semantics.amount.currencyCode: 'Currency'

forcuram as BookingPrice,

@EndUserText.label: 'Currency'

@Semantics.currencyCode: true

forcurkey as Currency,

@EndUserText.label: 'Luggage Weight'

@DefaultAggregation: #SUM

@Semantics.quantity.unitOfMeasure: 'WeightUOM'

luggweight as WeightOfLuggage,

@EndUserText.label: 'Weight Unit'

@Semantics.unitOfMeasure: true

wunit as WeightUOM,

*-- Associations*

\_Airline,

\_Connection,

\_Customer

}

где

@VDM : {viewType: *#COMPOSITE*} - this is a COMPOSITE view which is *an association of Master data and Transaction data*.

@Analytics: dataCategory: *#CUBE* - define CDS view as a data Cube.  All other Dimension table are associated with this cube to built the final Reports.

* No Dimension view can consume it.  Only views which can consume a Cube view should be of type cube themselves.

**Step 5** - Create the final view ***ZCDS\_VIEW\_FREPORT*** which will be ready for the consumption by UI tools and finally by business users in the from of Reports/Dashboards.



где

@VDM: {viewType: *#CONSUMPTION*} – it’s the final view ready to consumed; no joins or associations are performed here.

@Analytics.*query*: true - set this view to be consumption ready by different UI tools - Bex, Webi, Lumira, SAC – SAP Analytics Cloud etc.

There are 2 tools in SAP S4HANA where this view can be consumed

* ***RSRTS\_ODP\_DIS***
* ***RSRT***

**RSRTS\_ODP\_DIS**

This tcode *display the structure of the Cube*.

Последовательность

1. Run T-code ***RSRTS\_ODP\_DIS***.
2. Choose ODP Context as *ABAP Core Data Services.*
3. For ODP Name, enter the DDIC SQL View name followed by ‘*2C*’ – *2CZSQL\_VIEW\_FLBOOK*.
4. Execute.

Click on button *Standard Query* - now you can actually place data elements into rows and columns like a report.

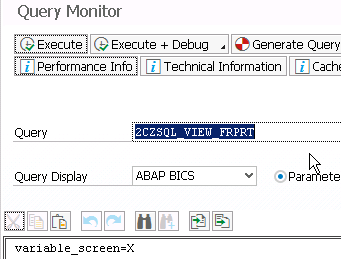
Since it is kind of a virtual cube now, you can also pull it into ***Bex Query Designer*** to create a Bex Query

1. Open Bex Query Designer.
2. Hit *Create* and search for the view *2CZSQL\_VIEW\_FLBOOK* and *Open* - now within Bex designer, you can create a regular report with all the advanced Bex features.

**RSRT**

To enable the business user to access a CDS view as type of an analytical report, we need to create the *final view* which is ready for user consumption. This view needs to be created of type @VDM: {viewType: *#CONSUMPTION*} and we need to add another annotation which will make this view as a query - @Analytics.*query*: true.

Now run **T-code RSRT** and enter the name of the view followed by ***2C*** – *2CZSQL\_VIEW\_FRPRT* and hit *Execute*



# How to Create a Basic Interface CDS View with ABAP

<https://blog.sap-press.com/how-to-create-a-basic-interface-core-data-services-view-with-abap>

# Delta Handling

# <https://blogs.sap.com/2019/12/16/cds-based-data-extraction-part-ii-delta-handling/>

# [Содержание](#Содержание)

You have actually two options for delta handling

1. ***Generic Timestamp / Date based Delta*** - this requires date/time inf. being available in the relevant application tables that is updated based on changes of the application data.
2. ***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***

You can use the following field types as delta criterion

* ***UTC timestamp*** [preferred way]
* ***date field*** [ABAP type DATS]

delta.byElement

If no appropriate UTC time stamp is available in the application tables/CDS view, a relevant *time/date field* *date field* can be used as well.

@Analytics:{

dataCategory: #FACT,

dataExtraction: { enabled: true *delta.byElement*: { name: 'LastChangeDateTime' } }

}

# Using this field [*delta.byElement*], the ODP framework determines up to which record a data consumer has extracted records already. On a subsequent extraction request, only records with a *higher* time stamp/date value are extracted.

# Safety interval

A ***safety interval*** can accommodate technical delays like waiting for a database commit on the application system side. You can *specify the safety interval* using this annotation @Analytics.dataExtraction.*delta.byElement*.***maxDelayInSeconds***.

If you do not add this annotation, a default delay of *1800 sec*, i.e. half an hour, is applied.

@Analytics:{

dataCategory: #FACT,

dataExtraction: {

enabled: true, delta.*byElement* : { *name*: 'LastChangeDateTime', *maxDelayInSeconds* : 1800 }

}

}

A record with a time stamp falling in this time safety interval will be selected twice from the CDS view.

### Detecting deletion records using the generic delta

# [Содержание](#Содержание)

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.

Rem

This concept is only feasible for low level volumes of data [~< 1.000.000 data records] and ***should not be used* for high volume data applications *[из-за необходимости хранить отдельную таблицу всех ключей]* ⇒** 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. This means you have a trailing limit and *only the set of extracted records falling into this time frame* are compared against the currently available records in the CDS view.

@Analytics:{

dataCategory: *#DIMENSION*,

dataExtraction: {

enabled: true,

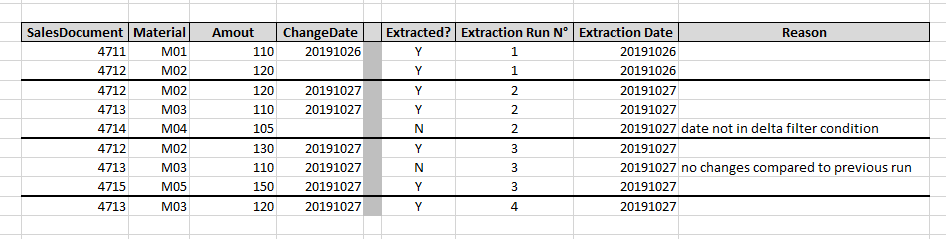
delta.b*yElement* : { *name*: 'LastChangeDateTime', *detectDeletions* : true, *ignoreDeletionAfterDays* : '365' }

}

}

### Miscellaneous facts for Generic Delta

Please note the extraction example in the following figure, uses a *field ChangeDate*, as the delta identifier.



* Data records *with an empty field using as the delta identifier* /ChangeDate в таблице выше/ 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. /Record 4714/.
* ***Timestamp based*** delta extraction *only**supports one time field* [time stamp or date] ⇒ you will need to make sure, that the change date field *is equally filled at creation time* and not only after a first change to the record.
* **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.

The second delta method goes by the name of *Change Data Capture* - ***CDC***. From a technology point of view this delta method makes use of real-time database triggers on table level based on SLT technology.

INSERT, UPDATE and DELETE operations can be recorded by the framework.

To function properly, the ***key fields*** of all underlying tables need to be exposed as elements of the CDS view and be mapped accordingly. In case of

* an ***INSERT*** | ***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.

The main task for exposing a CDS view with CDC delta method is to provide the *mapping inf.* between the *fields of a CDS view* and the *key fields of the underlying tables*. Given one record changes in possibly only one of the underlying tables, the framework needs to determine which record/s are affected by this change in all other underlying tables and need to provide as consistent set of delta records to the ODQ.

### ***Projections CDS view on a table***

In this case the frame work can derive the relation between the *fields of the CDS view* and *key fields of the underlying table* itself. Whenever a record is inserted | updated | 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*** annotation

@Analytics:{

dataCategory: #DIMENSION

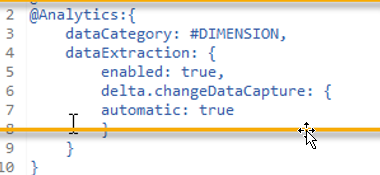
dataExtraction: {

enabled: true, delta.*changeDataCapture*: { automatic : true }

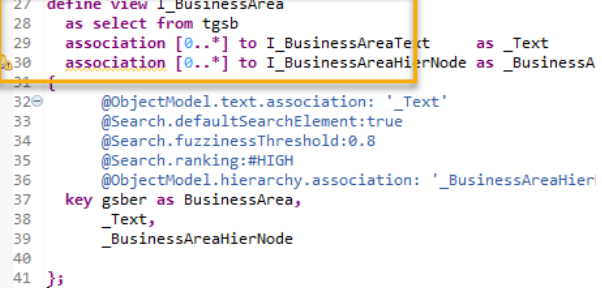
}

}

Пример



…



### ***Joins***

In joins the developer must *explicitly provide the mapping* to the frame work.

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 annotations relevant for this mapping task are subsumed /включена/ 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* -** to identify the *name of the underlying table(s) to be logged*.
* @Analytics.dataExtraction.delta.changeDataCapture.mapping.***role* -** is used for identifying *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*, [identified by role *#MAIN*] - enumerates *all exposed CDS view element names* corresponding to the *key fields* of the main table
* an *outer table* [, identified by role *#LEFT\_OUTER\_TO\_ONE\_JOIN*] - enumerates *all exposed CDS view element names* corresponding to *the foreign key fields* used in the on-conditions of the joins.
* @Analytics.dataExtraction.delta.changeDataCapture.mapping.**tableElement -** enumerates the respective *key fields* of the underlying tables to be logged.

Rem

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.

For technical reasons, *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**

# [Содержание](#Содержание)

Annotation @Analytics.dataExtraction.*delta.changeDataCapture.mapping*.**filter** can be used to define filter values on the table to be logged.

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' } ], ...

}, ...

]

}

}

}

*где*

…delta.changeDataCapture.mapping.filter.***tableElement* -** the table field to be restricted.

…delta.changeDataCapture.mapping.filter.**operator - r**estriction operator.

…delta.changeDataCapture.mapping.filter.**value -** the actual filter value.

…delta.changeDataCapture.mapping.filter.***highValue* -** the upper value in case of a range in conjunction with .operator #BETWEEN or #NOT\_BETWEEN

Пример

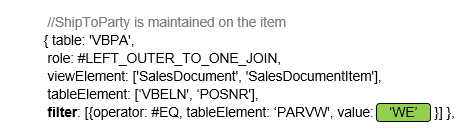
To complicate things the *ShipToParty* can either be maintained in the *header document* or individually directly *on a sales order item level* in the application.

Regarding the extraction, a *ShipToParty* maintained on the item *overrules* the one in the header document, evaluated by the COALESCE statement in the CDS view; similarly for the PayerParty and BillToParty.

cast(COALESCE(ShiptoPartyItem.Customer, ShiptoPartyHeader.Customer) as *kunwe*) as *ShipToParty*, cast(COALESCE(PayerPartyItem.Customer, PayerPartyHeader.Customer) as *kunrg*) as *PayerParty*, cast(COALESCE(BillToPartyItem.Customer, BillToPartyHeader.Customer) as *kunre*) as *BillToParty*,

If the *ShipToParty* is maintained on item level, the on-condition of the join between *Sales Document [Item Data] -* VBAP and *SalesDocument [Partner] -* VBPA consists of

* ? Sales Document Number (VBELN)
* ? Sales Document Item (POSNR)
* ? Constant value ‘WE’ for Partner Function (PARVW)



*//ShipToParty is maintained on the header*

{table: 'VBPA',

role: #LEFT\_OUTER\_TO\_ONE\_JOIN,

viewElement: ['SalesDocument'],

tableElement: ['VBELN'],

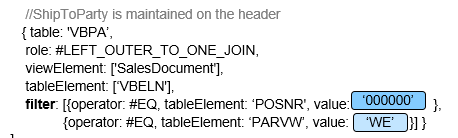
*filter*:[{operator: #EQ, tableElement: ' POSNR', valued '000000' },

{operator: #EQ, tableElement: ' PARVW', value: 'WE' }]}

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.

**Update (July 2021)**

You should be using view C\_SALESDOCUMENTITEMDEX\_1 instead of C\_SALESDOCUMENTITEMDEX.



# CDS based data extraction

# [Содержание](#Содержание)

# CDS based data extraction – Part I Overview

<https://blogs.sap.com/2019/12/13/cds-based-data-extraction-part-i-overview/>

# CDS based data extraction – Part II Delta Handling

<https://blogs.sap.com/2019/12/16/cds-based-data-extraction-part-ii-delta-handling/>

# CDS based data extraction – Part III Miscellaneous

<https://blogs.sap.com/2019/12/20/cds-based-data-extraction-part-iii-miscellaneous/>

[**Part# 7. How to search for SAP standard CDS Views**](https://blogs.sap.com/2019/10/29/part7.-how-to-search-for-sap-standard-cds-views/) : Find ways to search for SAP delivered CDS content.

[**Part# 11. End to End Data Modeling and Reporting with CDS views**](https://blogs.sap.com/2019/11/05/part11.end-to-end-data-modeling-and-reporting-with-cds-views/): Hands on with a complete business scenario for a simple analytical requirement.

# CDS views - Ресурсы

# [Содержание](#Содержание)

# ABAP Core Data Services – Part 1(ABAP CDS Entities)

# ABAP CDS views with Authorization based on Access Control

<https://blogs.sap.com/2017/02/27/abap-cds-views-with-authorization-based-on-access-control/>

# Как надо и как не надо использовать ABAP CDS View

<https://www.kaznacheev.me/article/kak-nado-i-kak-ne-nado-ispolzovat-abap-cds-view/>

# Currency and Unit conversion in ABAP CDS views

<https://blogs.sap.com/2019/07/11/currency-and-unit-conversion-in-abap-cds-views/>

### **ABAP CDS - Data Definitions**

<http://sapabapcentral.blogspot.com/p/abap-cds-data-definitions.html>

# CDS – One Concept, Two Flavors

<https://blogs.sap.com/2015/07/20/cds-one-model-two-flavors/>

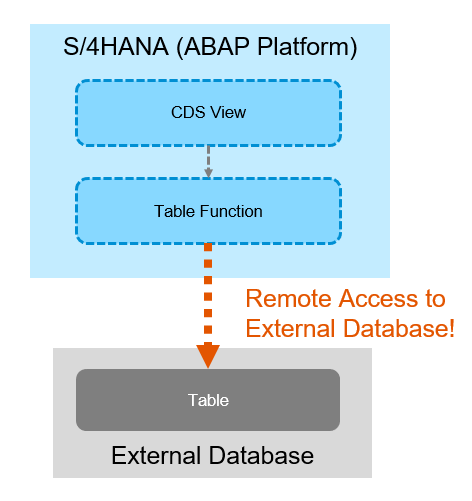
# CDS View accessing external system remotely in S/4HANA

<https://blogs.sap.com/2021/10/23/cds-view-accessing-external-system-remotely-in-s-4hana/>

# [Содержание](#Содержание)

Scenario /sɪˈnɑːrɪəʊ/

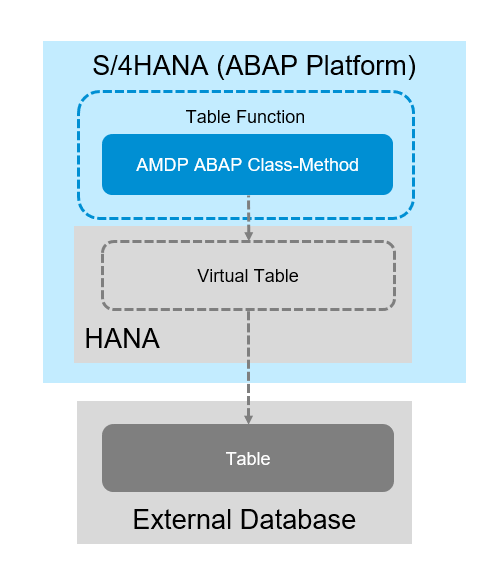
1. Create cross system analytical app in S/4HANA in which data in S/4HANA and external system are integrated.
2. Enjoy prediction for data in external system leveraging S/4HANA embedded *Machine Learning* capability.
3. Extract data from external system and store in S/4HANA without other tools, e.g. data Data transformation is possible with SELECT statement in S/4HANA.



**The main point is the program to create *Virtual Table* which access the data in external system remotely [Available in SAP HANA *Runtime Edition for Apps and SAP BW* - HANA *REAB*].**

## **Technology Overview**

[HANA *Virtual Table*](https://www.youtube.com/watch?v=BomjFbJ25vo) should be created for remote access in HANA Database layer [[***SDA*** - Smart Data Access](https://help.sap.com/viewer/6b94445c94ae495c83a19646e7c3fd56/2.0.02/en-US/a07c7ff25997460bbcb73099fb59007d.html)] and the *table* in HANA Database layer is accessible with [Table Function](https://help.sap.com/viewer/f2e545608079437ab165c105649b89db/7.5.8/en-US/e5529f75afbc43e7803b30346a56f963.html#:~: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.) in which AMDP ABAP Class-Method is used as the source and HANA Native SQL statement can be used in the AMDP.



**How can the Virtual Table be created?**

Thanks to the program ***ZICA\_CREATE\_VIRTUAL\_TABLE*** in [the blog “Using Remote Data Source in ICMR”](https://blogs.sap.com/2021/04/07/using-remote-data-source-in-icmr/), Virtual Table can be created in ABAP Layer [so, it is possible with HANA REAB edition].

**Which external sources are available?**

The sources available for HANA SDA

* HANA
* HANA Cloud
* ASE
* IQ
* Oracle
* MSSQL
* Google Big Query, etc.

Rem

All SDA supported remote sources require installation and configuration of ODBC driver.

## **Remote CDS View and scenarios**

This blog shows the objects created for the scenario

1. [***Remote CDS View***](https://blogs.sap.com/2021/10/23/cds-view-accessing-external-system-remotely-in-s-4hana/#1. Remote CDS View)

Table BSEG and T001 in the external HANA system is accessed remotely in S/4HANA system via CDS View/Table Function.

1. [**Analytical app**](https://blogs.sap.com/2021/10/23/cds-view-accessing-external-system-remotely-in-s-4hana/#2. Analytical app)

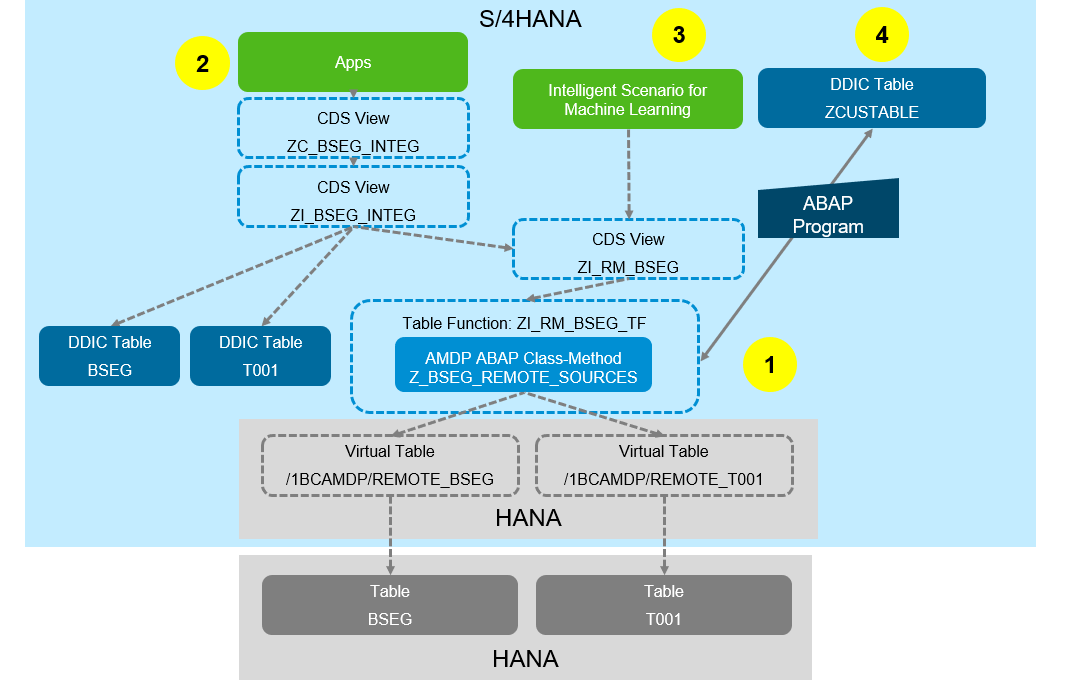
Web Dynpro Grid app [Multidimensional Analytics] is created in which data in S/4HANA and external HANA system are merged. The Interface View [CUBE View] is created in which data in BSEG/T001 in S/4HANA and the created Table function are combined [UNION]. Consumption View [Analytic Query] is created on top. It can be consumed with Web Dynpro Grid App.

1. [**Embedded Machine Learning**](https://blogs.sap.com/2021/10/23/cds-view-accessing-external-system-remotely-in-s-4hana/#3. Embedded Machine Learning)

Consume the external data for S/4HANA embedded Machine Learning. Defining and training the Machine Learning model of S/4HANA using this remote CDS View with *Intelligent Scenario Lifecycle Management* - ***ISLM***.

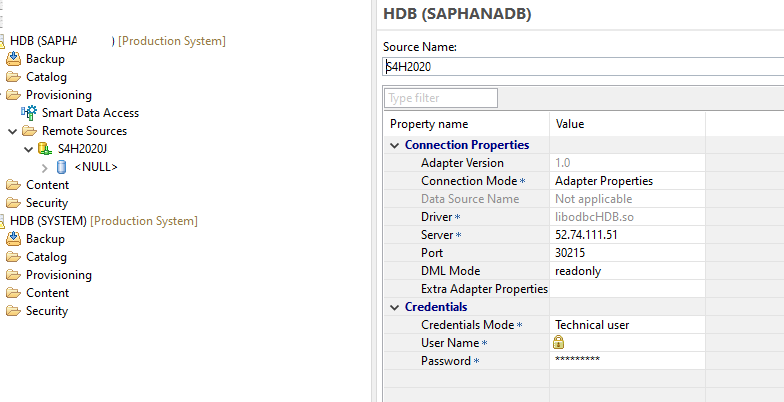
1. [**Data extraction from external system**](https://blogs.sap.com/2021/10/23/cds-view-accessing-external-system-remotely-in-s-4hana/#4. Data extraction from external system)

Extract external data from this remote CDS View in S/4HANA and store the data in the custom table in ABAP Program.



**Remote CDS View**

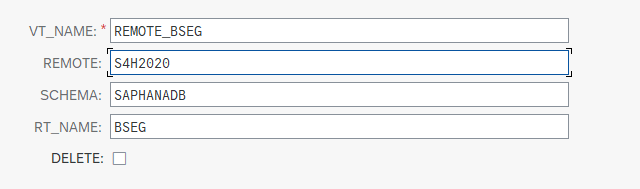
Remote source for the external HANA Database is created



Create Virtual table with the program [ZICA\_CREATE\_VIRTUAL\_TABLE](https://blogs.sap.com/2021/04/07/using-remote-data-source-in-icmr/).

When running the program, you are expected to enter

* *VT\_NAME* - Virtual View name
* *REMOTE* - Name of the remote source
* *SCHEME* - Scheme name in which the virtual table is generated [normally SAP<SID>]
* *RT\_NAME* - Source table name.



By executing the program with the input values, the Virtual table is generated [View name - /1BCAMDP/<VT\_NAME].

*bseg* [remote]

*t001* [remote]AMDP class *Z\_BSEG\_REMOTE\_SOURCES*

*call\_01()* tf *zi\_rm\_bseg\_tf*

[Remote] CDS view *zi\_rm\_bseg*

*bseg*

*t001* CDS view *zi\_bseg\_integ*

Table Function and AMDP to read from the Virtual Table

@ClientHandling.type: #CLIENT\_DEPENDENT

define table function ***ZI\_RM\_BSEG\_TF***

returns {

*key* RCLNT : mandt; *key* bukrs : bukrs; *key* belnr : belnr\_d; *key* GJAHR : gjahr; *key* BUZEI : buzei;

SAKNR : saknr; AUGDT: augdt; DMBTR : dmbtr; WAERS : waers; KOKRS : kokrs; KOSTL : kostl;

}

implemented by method *z\_bseg\_remote\_sources=>CALL\_01*;

AMDP ABAP Class-Method

class **z\_bseg\_remote\_sources** *definition*

public

final

create public .

public section.

interfaces if\_amdp\_marker\_hdb .

class-methods ***call\_01*** for table function *ZI\_RM\_BSEG\_TF*.

protected section.

private section.

endclass.

class **z\_bseg\_remote\_sources** *implementation*.

method ***call\_01***

by database function for hdb language sqlscript

options read-only.

return

*-- Native SQL*

select

SESSION\_CONTEXT('CDS\_CLIENT') as RCLNT,

bseg.bukrs, belnr, GJAHR, BUZEI, SAKNR, AUGDT, DMBTR, WAERS, KOKRS, KOSTL

from *"/1BCAMDP/REMOTE\_BSEG"* as bseg *-- virtual table*

inner join *"/1BCAMDP/REMOTE\_T001"* as t001 -- *or use "ZHANGVIN"."REMOTE\_T001"*

on bseg.mandt = t001.mandt and bseg.bukrs = t001.bukrs

where bseg.mandt = '100';

endmethod.

endclass.

Remote CDS View in which the Table Function is used as the source

@AbapCatalog.sqlViewName: *'ZIRM\_BSEG'*

@AbapCatalog.compiler.compareFilter: true

@AbapCatalog.preserveKey: true

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: 'ZI\_RM\_BSEG'

define view ***ZI\_RM\_BSEG*** as

select from *ZI\_RM\_BSEG\_TF* {

*key* bukrs, key belnr, *key* GJAHR, *key* BUZEI,

SAKNR, AUGDT,

@DefaultAggregation: *#SUM*

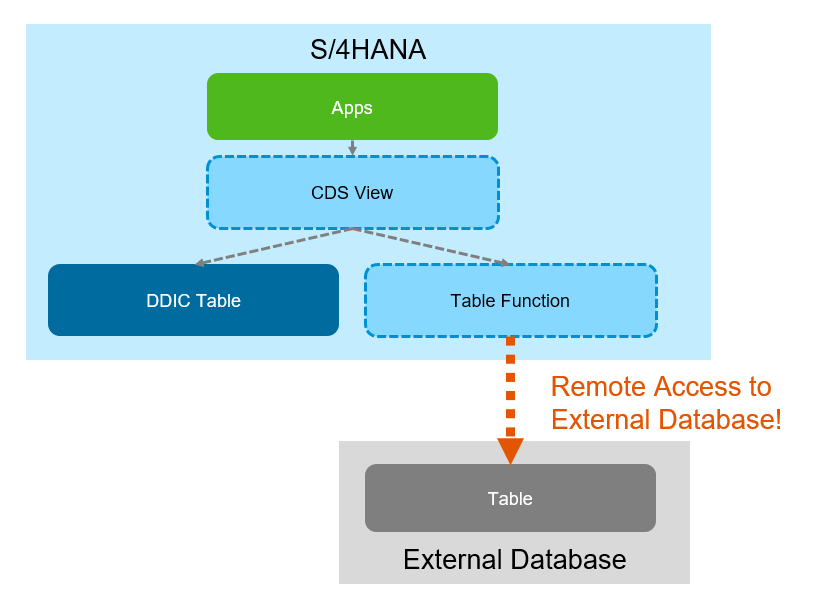
@Semantics.amount.currencyCode: *'WAERS'*

DMBTR, WAERS, KOKRS, KOSTL

}

**Analytical app**

Analytical app in which internal and external data are merged can be created with *Remote CDS View*.



CDS View for Interface View for integration [CUBE View]

In this view, data of *BSEG* in S/4HANA *itself* and data of *BSEG* in *external* HANA Database are merged [UNION]. New field *SOURCE* is added in which the value is set to *“ORIGINAL”* for the data from S/4HANA and *“REMOTE1”* for the data from external HANA Database.

@AbapCatalog.sqlViewName: *'ZIBSEG\_INTEG'*

@AbapCatalog.compiler.compareFilter: true

@AbapCatalog.preserveKey: true

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: 'ZI\_BSEG\_INTEG'

@*Analytics.dataCategory*: #CUBE

define view ***ZI\_BSEG\_INTEG*** as

select from *bseg*

inner join *t001*

on bseg.mandt = t001.mandt and bseg.bukrs = t001.bukrs {

*key* bseg.bukrs, *key* belnr, *key* gjahr, *key* buzei,

saknr, augdt,

@DefaultAggregation: *#SUM*

@Semantics.amount.currencyCode: *'WAERS'*

dmbtr, waers, kokrs, kostl,

@Environment.sql.passValue: true

cast(*'ORIGINAL'* as abap.char( 10 )) as Source

}

union all

select from *ZI\_RM\_BSEG* {

*key* bukrs, *key* belnr, *key* GJAHR, *key* BUZEI,

SAKNR, AUGDT,

@DefaultAggregation: *#SUM*

@Semantics.amount.currencyCode: *'WAERS'*

DMBTR, WAERS, KOKRS, KOSTL,

@Environment.sql.passValue: true

cast(*'REMOTE1'* as abap.char( 10 )) as Source

}

Consumption View [Analytic Query]

@AbapCatalog.sqlViewName: *'ZCBSEG\_INTEG'*

@AbapCatalog.compiler.compareFilter: true

@AbapCatalog.preserveKey: true

@AccessControl.authorizationCheck: #CHECK

@EndUserText.label: 'ZC\_BSEG\_INTEG'

@*Analytics.query*: true

define view ***ZC\_BSEG\_INTEG*** as select from *ZI\_BSEG\_INTEG* {

@AnalyticsDetails.query.axis: *#ROWS*

bukrs, belnr, gjahr, buzei,

@AnalyticsDetails.query.axis: *#ROWS*

saknr, augdt, dmbtr,

@AnalyticsDetails.query.axis: *#COLUMNS*

waers, kokrs, kostl,

@AnalyticsDetails.query.axis: *#COLUMNS*

Source

}

The example of URL of *Web Dynpro Grid* to run the Analytic Query is below.

https://vhcals4hcs.dummy.nodomain:44300/sap/bc/webdynpro/sap/fpm\_bics\_ovp?bsa\_query=*2CZCBSEG\_INTEG*&sap-client=100&sap-language=EN#

Rem

This URL would not work in your site, but is only the example. *2CZCBSEG\_INTEG* is the *analytic query name* [“**2C**<SQL View name>”].

**UNION view by avoiding the access to remote CDS View thanks for *UNION Node [view] pruning*** - it happens when setting the annotation [*@Environment.sql.passValue*: true](https://blogs.sap.com/2018/11/28/safeguard-performance-of-abap-cds-views-part-2-hana-sql-optimizer-and-plan-cache/) for constant value in each view.

[UNION Node Pruning](https://help.sap.com/viewer/9de0171a6027400bb3b9bee385222eff/2.0.04/en-US/92f5022be9234ed19d91d16c70c3b4b5.html) /ˈpruːnɪŋ обрезка/:

In the case the CDS View *ZI\_BSEG\_INTEG*, two select statements are combined with UNION

* the first - accesses BSEG and T001
* the second  - *ZI\_RM\_BSEG* [remote CDS View].

In case the SELECT statement to ZI\_BSEG\_INTEG  includes where SOURCE = *‘ORIGINAL'* - the access happens only to BSEG and T001 internally thanks to the *SQL optimization*.

SELECT "BUKRS", "SOURCE", "WAERS", "SAKNR",

COUNT(\*) , SUM( "DMBTR" )

FROM *"****ZIBSEG\_INTEG****"* "A1"

WHERE "MANDT" = ‘100’ AND "SOURCE" = *'ORIGINAL'*

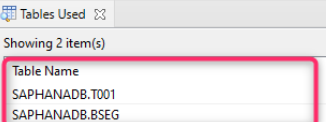
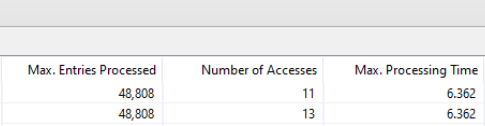
GROUP BY "BUKRS" , "SOURCE" , "WAERS" , "SAKNR"

ORDER BY "A1" . "BUKRS" , "A1" . "SOURCE" , "A1" . "WAERS" , "A1" . "SAKNR"

WITH HINT(RESULT\_LAG ('hana\_long'))

WITH RANGE\_RESTRICTION('CURRENT')

The [Plan Visualization](https://blogs.sap.com/2020/01/30/how-to-analyze-query-performance-for-abap-cds-views/) for the SELECT statement shows the acces happens only to BSEG and T001 as below

***Static Result Cache* is available also for the CDS View using the Table Function accessing external system remotely as the source.** Although update of the static cache is not in real-time normally, it would help improving the performance to access external system.

Firstly, the configuration of HANA should be changed as below /T-cd ***DBACOCKPIT*** or *Administration* in HANA Studio/.

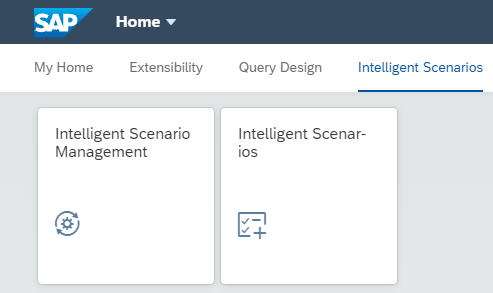
indexserver.ini -> [result\_cache] -> enabled = yes [default: no];

Then add the cache to the SQL View of the CDS View.

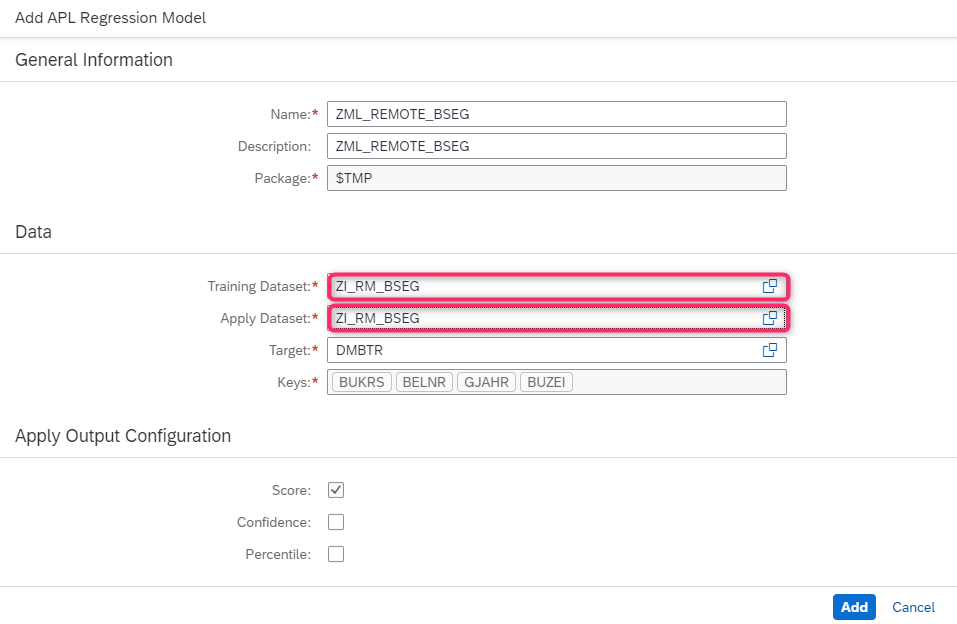
See more in [SAP Note 2336344 – FAQ: SAP HANA Static Result Cache](https://launchpad.support.sap.com/#/notes/2336344).

**Embedded Machine Learning**

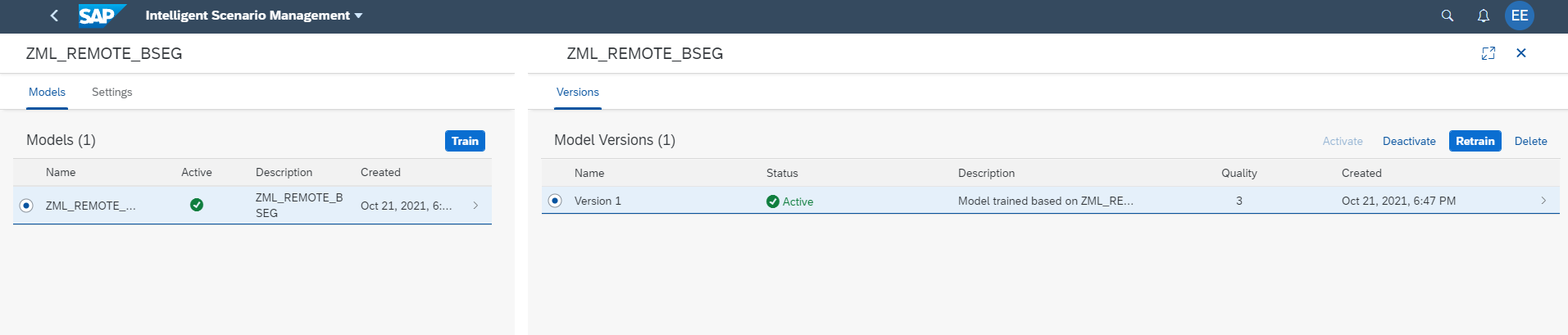
CDS View *ZI\_RM\_BSEG* is used as direct consumption of Table Function is not possible in ISLM. See more in [this blog](https://blogs.sap.com/2021/10/16/custom-embedded-machine-learning-app-in-s-4hana/) about ISLM.Create ***Intelligent Scenario*** /*Machine Learning Model*/ with Intelligent Scenario.



The created CDS View can be used as the *Training Dataset* and *Apply Data set*. [For Apply Dataset /data for which the value/attribute is predicted/, it would be appropriate to create another custom CDS View].



With ***Intelligent Scenario Management***, the model can be trained



***ABAP Apply View*** is generated

@EndUserText.label: ‘ISLM AutoGenerated View ZML\_REMOTE\_BSEG\_CDS01 ’ @AbapCatalog.sqlViewName: ’Z\_9C6344F4F8D0CF’ *--ISLM Random generated SQL name* @AbapCatalog.dataMaintenance: #DISPLAY\_ONLY

@ClientHandling.type: #CLIENT\_DEPENDENT

@ClientHandling.algorithm: #SESSION\_VARIABLE

@VDM.private: false

@VDM.viewType: *#COMPOSITE*

define view ***ZML\_REMOTE\_BSEG\_CDS01***

as

select from *ZML\_REMOTE\_BSEG\_TF01* ( p\_clnt: $session.client )

as a

association [1..1] to *ZI\_RM\_BSEG* as \_CKE\_toBase

on a.bukrs = \_CKE\_toBase.bukrs and a.belnr = \_CKE\_toBase.belnr and

a.GJAHR = \_CKE\_toBase.GJAHR and a.BUZEI = \_CKE\_toBase.BUZEI

{*key* a.bukrs as bukrs, *key* a.belnr as belnr, *key* a.GJAHR as GJAHR, *key* a.BUZEI as BUZEI,

a.DMBTR as DMBTR, a.Rr\_DMBTR as Rr\_DMBTR,

*/\* Associations \*/*

@ObjectModel.text.element: null

@Consumption.hidden: true

\_CKE\_toBase as \_CKE\_toBase

}

**Data extraction from external system**

Data in external database can be extracted in S/4HANA via Virtual Table through 1) *table function* with ABAP SQL or 2) directly with *Native SQL*.

**Through Table function**

Data transformation is possible with Native SQL in AMDP and ABAP SQL and ABAP syntax in ABAP Program

REPORT ***ZPRG100***.

insert *ZBSEGTEST*

from (

select BUKRS, BELNR, GJAHR, BUZEI, SAKNR, AUGDT, DMBTR, ' EUR' as WAERS, KOKRS, KOSTL

from *ZI\_RM\_BSEG\_TF*

).

**Directly from Virtual table in ABAP Program with Native SQL**

Data transformation is possible with Native SQL in ABAP Program.

REPORT ***ZPRG100***.

exec sql.

insert into *ZBSEGTEST* (

RCLNT, BURRS, BELNR, GJAHR, BUZEI, SAKNR, AUGDT, DMBTR, WAERS, KOKRS, KOSTL

)

select SESSION\_CONTEXT('CDS\_CLIENT') as RCLNT, bseg.bukrs, belnr, GJAHR, BUZEI, SAKNR,

AUGDT, DMBTR, WAERS, KOKRS, KOSTL

from *"/1BCAMDP/REMOTE\_BSEG"* as bseg

inner join *"/1BCAMDP/REMOTE\_T001"* as t001 *-- or use "ZHANGVIN"."REMOTE\_T001*

on bseg.mandt = t001.mandt and bseg.bukrs = t001.bukrs

where bseg.mandt = ’100'

endexec.

## **Other ways to access external system [but too complex!]**

When creating Open ODS View in embedded BW functionality in S/4HANA, Virtual Table is created internally.

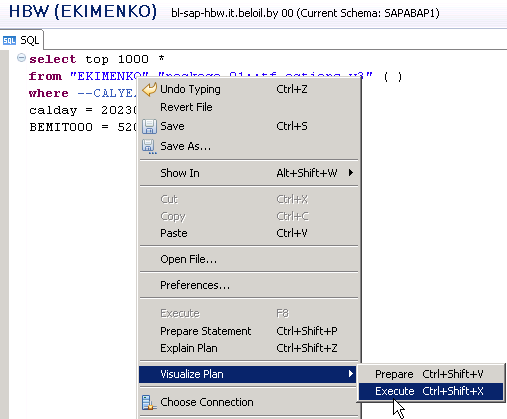
Virtual Table can be consumed in ABAP Program via [*ABAP External View*](https://blogs.sap.com/2014/01/08/consuming-hana-views-procedures-external-views-in-abap-740-syntax-part-2/), in which HANA Calculation View can be the source, and in which Virtual Table can be the source of HANA Calculation View [Virtual Table can’t be the source of ABAP External Table directly]. But now ABAP External View is not recommended by the development team. See the comment in [the blog.](https://blogs.sap.com/2018/05/24/abap-dictionary-external-views-for-overcoming-abap-cds-views-limitiations/) By adding the hint “no\_calc\_view\_unfolding” = “1” in the HANA Calculation View, the performance would be optimized.

For Virtual Table internally generated by BW Open ODS View, firstly Composite Provider has to be created using the Open ODS View as the source and generate [*HANA External View*](https://blogs.sap.com/2020/12/01/external-hana-view-configuration-in-bw-on-hana/) from the Composite Provider to generate HANA Calculation View. It is not supported to use the generated Virtual Table directly. In addition, renaming of the field name to set to start with alphabet, not number (4~) to use it as the source of ABAP External View [Too complex!].

# How to analyze query performance for ABAP CDS Views

<https://blogs.sap.com/2020/01/30/how-to-analyze-query-performance-for-abap-cds-views/>

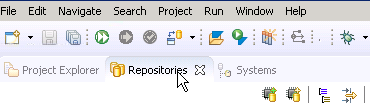
# [Содержание](#Содержание)

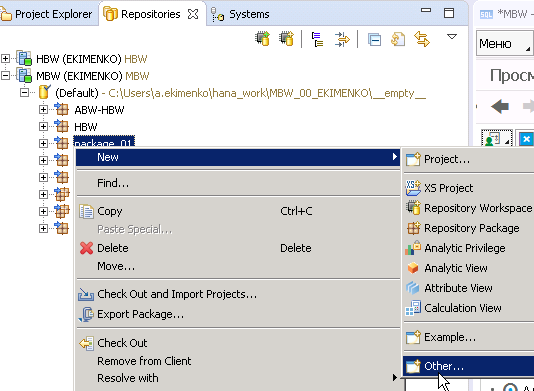


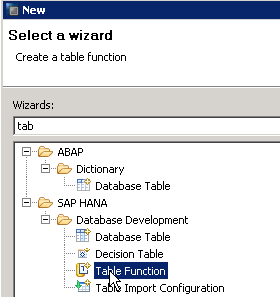
**Table function**

# [Содержание](#Содержание)

A table function is a development artifact and must be created in the repositories tab.

Switch to the ***Repositories*** tab - .





The extension .hdbtablefunction gets auto-appended once you click anywhere outside this text box.

**There are 2 security modes**

1. *INVOKER* - the function executes with the privileges of the invoker of the procedure.
2. *DEFINER* - the function executes with the privileges of the definer of the procedure.

It’s always best to leave it as INVOKER so that whenever this function is called, it will run based on the invoking user’s privileges.

# Data types

# <https://www.tutorialspoint.com/sap_hana/sap_hana_data_types.htm>

# *SAP HANA supports 7 categories of SQL data types*

# 

## **Date Time**

* **DATE** − /Default format is YYYY-MM-DD.
* **TIME** − /Default format is HH: MI: SS.
* **SECOND DATE** − /Default format is YYYY-MM-DD HH:MM:SS.
* **TIMESTAMP** − /Default format is YYYY-MM-DD HH:MM:SS:FFn, where FFn represents fraction of second.

## **Numeric**

* **TinyINT** − 8 bit – 0-255
* **SMALLINT** − 16 bit signed / -32,768.. 32,767
* **Integer** − 32 bit signed / -2,147,483,648.. 2,147,483,648
* **BIGINT** − 64 bit signed / -9,223,372,036,854,775,808.. 9,223,372,036,854,775,808
* **SMALL** − -1038 +1.. 1038 -1
* **REAL** − -3.40E38..3.40E38
* **DOUBLE** − -1.7976931348623157E308.. 1.7976931348623157E308

## **Character**

* **Varchar** − maximum of 8000 characters.
* **Nvarchar** − maximum of 4000 characters
* **ALPHANUM** – 1-127.
* **SHORTTEXT** − stores variable length character string which supports text search features and string search features.

**Способы извлечения данных**

* *Непосредственно в самой TF*

return select … from ;

* *Через AMDP класс*,

Например - читающий данные из CDS view.

returns

{

*--client : abap.clnt;*

number : *abap*.*int8*;

date : *abap*.*dats*;

}

implemented by method zcl\_datestable=>GetNDatesStartingFrom;

## **How to query Table Functions?**

SELECT \*

FROM “TEST\_SCHEMA”.”TMP::TF\_NUMBER\_OF\_WORKDAYS” (‘2003-01-01’, ‘2003-02-01’)

SELECT \* FROM "PUBLIC"."package\_01::tf\_clock\_number"

# Возможности применения

# SAP HANA Table function is wrapped in CV.

# SAP HANA Table Functions

# <http://teachmehana.com/sap-hana-table-functions-1/>

# SAP HANA Table function – Calculation View Wrap

# <http://teachmehana.com/sap-hana-table-function-calculation-view-2/>

# Table Functions in SAP HANA – step by step guide

# <https://blogs.sap.com/2018/11/05/table-functions-in-sap-hana-step-by-step-guide/>

# Create Dates Table CDS Table Function using SAP HANA AMDP Class

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

# 

# 

**To** [**create a dates table on SAP HANA database**](http://www.kodyaz.com/sap-abap/create-dates-table-using-sqlscript-on-sap-hana-database.aspx)

1. it is easy and performs best to use **SERIES\_GENERATE\_DATE** function.
2. But *if you have problems or limitations to consume native HANA database development objects* - an alternative way is to create CDS table functions.

Пример 1

*Получить список дат - начиная с определенной даты и заданным кол-вом дат*

**CDS table function** 

@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;

Rem

ABAP programmer can save and activate the CDS table function though the class and the class method have not been created yet.

**AMDP класс предоставляющий данные для CDS table function** - 

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

*“ SQLScript Data Series function SERIES\_GENERATE\_DATE is used to populate and return the desired*

*“ dates table for the ABAP programmer.*

FROM SERIES\_GENERATE\_DATE( 'INTERVAL 1 DAY',

:starting\_date,

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

ENDMETHOD.

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.

**ABAP программа для тестирования** - 

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]

*“ lt\_data - таблица*

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

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

ENDLOOP.

Rem

1. В AMDP значение параметра - **:**starting\_date; в ABAP - **@**lv\_startdate .

2. ABAP developers will see a warning - 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.

Пример 2 – получить список дат в заданном диапазоне

**CDS table function** 

@ClientDependent: false

@EndUserText.label: 'Get Dates Between'

define table function ZGetDatesBetween

with parameters

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

*end\_date* : *abap*.*dats*

returns {

*-- client : abap.clnt;*

number : *abap*.*int8*;

date : *abap*.*dats*;

}

implemented by method zcl\_datestable=>GetDatesBetween;

**ABAP программа для тестирования** - 

REPORT **zgetdatesbetween\_test**.

DATA lv\_startdate TYPE dats.

DATA lv\_enddate TYPE dats.

DATA lv\_i TYPE i VALUE 10.

lv\_startdate = sy-datum.

lv\_enddate = lv\_startdate + 4.

*\* list N date values starting from D*

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

INTO TABLE @DATA(lt\_data)

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

*\* list dates between D1 and D2*

SELECT \* FROM ZGetDatesBetween( starting\_date = @lv\_startdate, end\_date = @lv\_enddate )

INTO TABLE @DATA(lt\_data2)

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

LOOP AT lt\_data REFERENCE INTO DATA(lr\_data).

AT FIRST.

WRITE :/ |List of { lv\_i } Dates Starting From { lv\_startdate }|.

ENDAT.

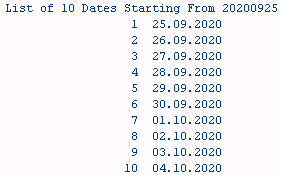
WRITE :/ lr\_data->number, lr\_data->date.

AT LAST.

WRITE :/.

ENDAT.

ENDLOOP.



LOOP AT lt\_data2 REFERENCE INTO DATA(lr\_data2).

AT FIRST.

WRITE :/ |Dates Between { lv\_startdate } and { lv\_enddate }|.

ENDAT.

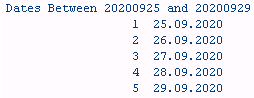
WRITE :/ lr\_data2->number, lr\_data2->date.

AT LAST.

WRITE :/ .

ENDAT.

ENDLOOP.



# ALV

# CDS ALV report in seconds with 7.5 SP 03

# <https://blogs.sap.com/2016/10/11/cds-alv-report-seconds-7.5-sp-03/>

# Sshm70720

# Qwert123456

# Ыырь70720

# CDS ALV report with selection criteria

# <https://blogs.sap.com/2016/11/04/cds-alv-report-with-selection-criteria/>

# [Содержание](#Содержание)

# 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/>

# Passing multi-value input parameter from Calculation View to Table Function in SAP HANA – step by step guide

# ABAP on SAP HANA. Part VII. SQL Script and SAP HANA Stored Procedure

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

**Diagnosis**

A reference field and a reference table were defined for the field of a base table of the view. The reference table is the base table of the view, but the reference field was not included in the view. Therefore it is not possible to replace the reference field with the corresponding view field.

**Procedure**

Check whether the specifications for the reference field and reference table in the base table make sense. If so, include the reference field as a view field.

The reference table and reference field of the base table are transferred without changes to the view field without changing the base field or view.

# [CDS Association with *where* condition containing a SELECT](https://stackoverflow.com/questions/67271967/cds-association-with-where-condition-containing-a-select)

− I have two views - *ZC\_PurRequisitionFs* and *ZMM\_ONAYT005*.

First view extended from C\_PurRequisitionFs.

Second view gets data from Z\* table.

How can I get only first view data that does not exist on second view ?

@AbapCatalog.sqlViewAppendName: 'ZCPURREQUISFS'

@EndUserText.label: 'Sat belgeleri'

*extend* view *C\_PurRequisitionFs* with ***ZC\_PurRequisitionFs*** {

\*

} where ZC\_PurRequisitionFs.object\_id not in( SELECT \* FROM ZMM\_ONAYT005 ).

− While CDS views do not support subqueries, they do support JOINs.

Usually you would use a JOIN to get only those entries which exist in both tables. But when you want all entries from table A which *don't* exist in table B, you can do a left outer join and then add a where-condition for only entries where the right table is null.

define view **Z\_TEST** as select

from *table\_a*

left outer join *table\_b* on table\_a.object\_id = table\_b.object\_id

{

... fields....

}

where *table\_b.object\_id* is null;

# [Calculated date in WHERE condition of CDS view](https://stackoverflow.com/questions/60991357/calculated-date-in-where-condition-of-cds-view)

− I'm trying to get a list of valid system status for the notification object, in order to not check all the notifications in the table, I want to execute the selection by checking only the last 2 years of data.

Maybe there is a better solution to my problem, but I'm still curious about this technical limitation. To my knowledge, the system status in SAP are kind of hardcoded and can't be determined per object via any table (SAP could add new system status any moment).

I tried to create the below CDS view, but the function *dats\_add\_months* can't be used in the *whe*re condition, is there a solution to that? Notice that 7.50 doesn't have session parameter for system date so I use an environment variable

define view **ZNOTIF\_SYS\_STATUS**

with parameters *sydat* : abap.dats *@<Environment.systemField: #SYSTEM\_DATE*

as

select distinct from *qmel* as *notif*

inner join *jest* as *notif\_status*

on notif\_status.objnr = notif.objnr and notif\_status.inact = ''

inner join *tj02t* as *sys\_status*

on sys\_status.istat = notif\_status.stat and sys\_status.spras = *$session.system\_language*

{

*key* sys\_status.txt04 as statusID,

sys\_status.txt30 as description

}

where notif.erdat > *dats\_add\_months*($parameters.sydat, -12, '') *//THIS CAN'T BE USED!!*

− Putting built-in functions in RHS */the right hand side of an equation/* position of WHERE is supported only [since 7.51](https://help.sap.com/doc/abapdocu_751_index_htm/7.51/en-US/abennews-751-abap_cds.htm#!ABAP_MODIFICATION_2@2@) and you have 7.50 as you said. That is why it works for Haojie and not for you.

What can be done here? Possible option is CDS table function which consumes AMDP-class. Consider this sample

**Table function**

@EndUserText.label: 'table\_func months'

define table function **ZTF\_MONTHS**

with parameters

@Environment.systemField : #SYSTEM\_DATE

*p\_datum* : syst\_datum

returns {

mandt : abap.clnt;

num : qmnum;

type : qmart;

}

*implemented by method* zcl\_cds\_qmel=>get\_last\_two\_years;

**AMDP**

CLASS **zcl\_cds\_qmel** DEFINITION

PUBLIC

FINAL

CREATE PUBLIC.

PUBLIC SECTION.

INTERFACES if\_amdp\_marker\_hdb.

TYPES: tt\_statuses TYPE STANDARD TABLE OF qmel.

CLASS-METHODS ***get\_last\_two\_years*** FOR TABLE FUNCTION *ztf\_months*.

PROTECTED SECTION.

PRIVATE SECTION.

ENDCLASS.

CLASS **zcl\_cds\_qmel** IMPLEMENTATION.

METHOD ***get\_last\_two\_years*** BY DATABASE FUNCTION

FOR HDB

LANGUAGE SQLSCRIPT

OPTIONS READ-ONLY.

two\_years := *add\_months* (CURRENT\_DATE, -12)

return

SELECT mandt, qmnum AS num, qmart AS type

FROM *qmel* WHERE erdat > two\_years;

ENDMETHOD.

ENDCLASS.

It is very simplified compared to your original task but gives you the idea how to do this.