01

Introduction to ABAP Core Data Services

WHY SAP HANA DATABASE

- SAP HANA (High-performance ANalytic Appliance) is a multi-model database that stores data in its memory instead of keeping it on a disk
- This results in data processing that is magnitudes faster than that of disk-based data systems, allowing for advanced, real-time analytics
- Multiple operations can be parallelly processed in SAP HANA as opposed to only a single operation processing in the traditional database for executing one query. Thus, parallel processing increases the speed of operation manifold
- Aggregation tables are no more required

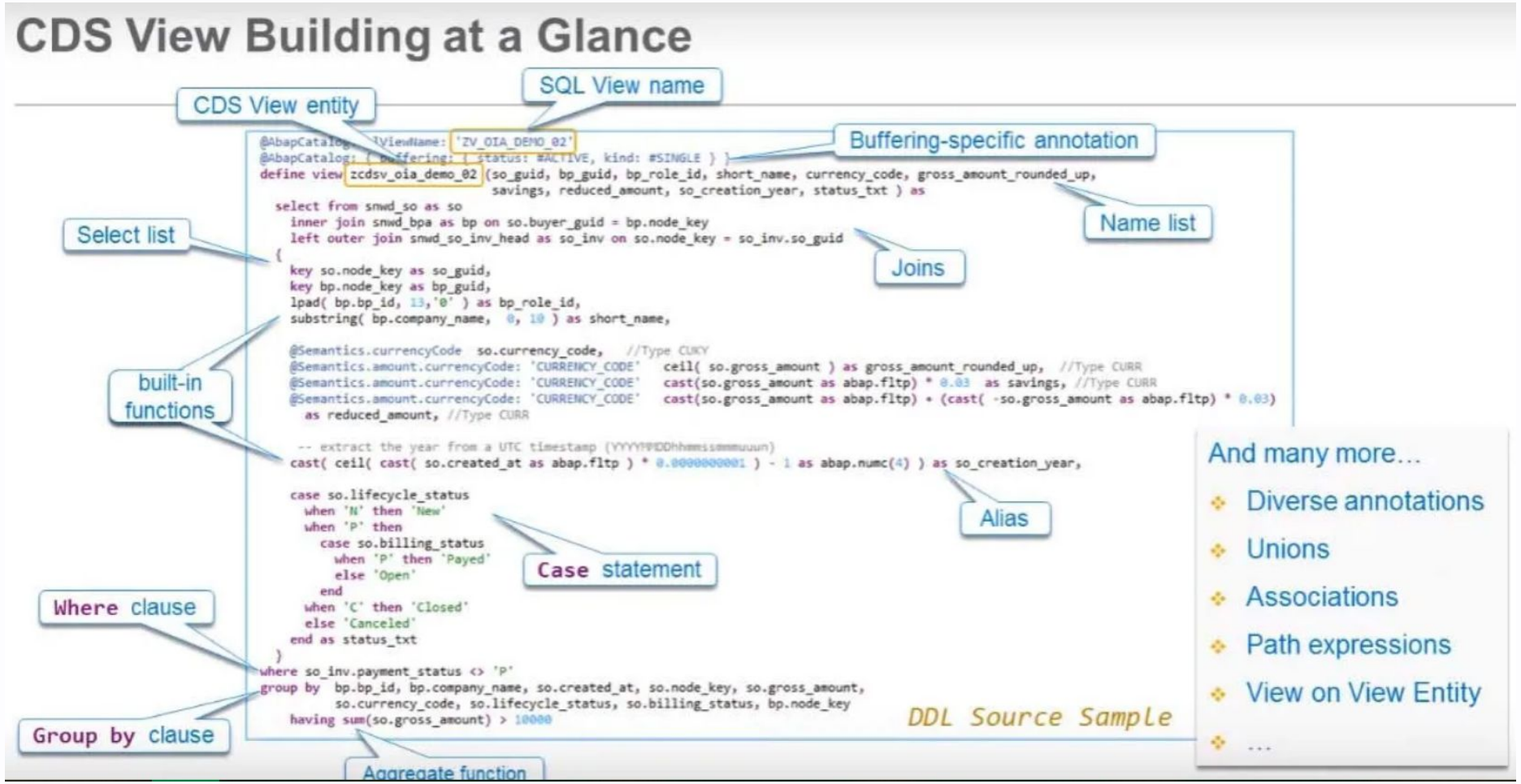
WHAT IS CDS VIEW

- Core Data Services (CDS) is a data modeling infrastructure
- They are the virtual data models of SAP HANA which allows direct access to underlying tables of the HANA database
- CDS as an infrastructure layer enables developers to define semantically rich data models
- Annotations are used to make the CDS data models more semantically rich
- When we create a CDS view, a SQL view also gets created in the database which we can see in the SE11 transaction in the SAP GUI

WHAT IS CDS VIEW

- Data can be fetched from the CDS view directly from report/global class using the Open SQL
- CDS view can be transported by the Standard ABAP Transport CTS (Change & transport System)

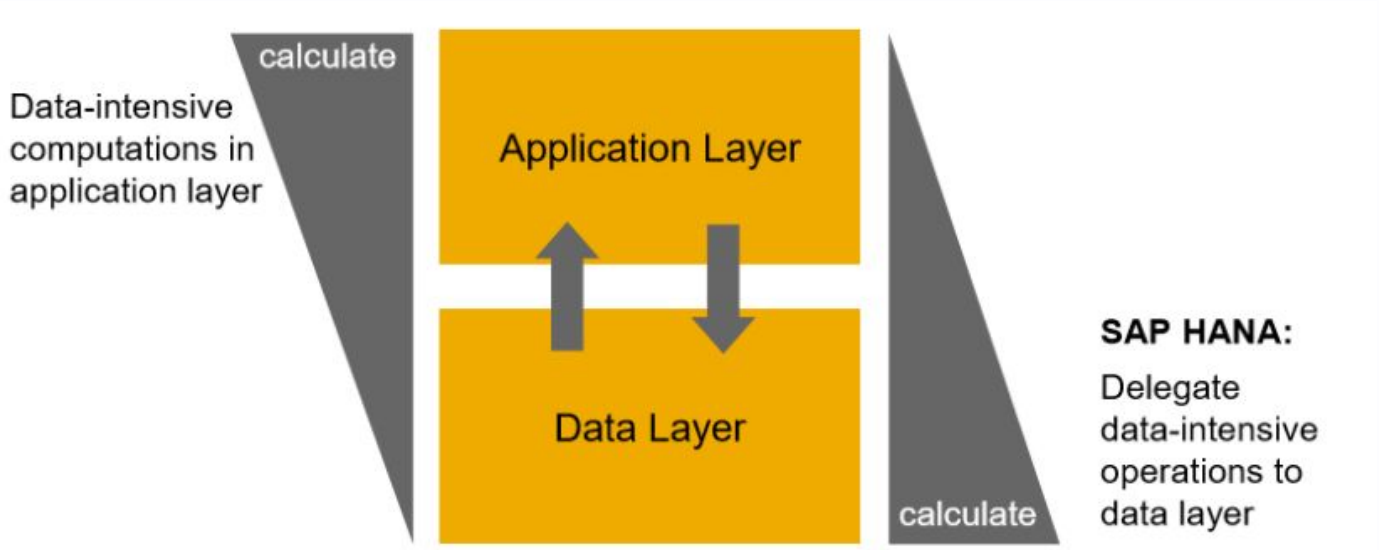
CDS View Building at a Glance




```
@AbapCatalog.sqlViewName: 'ZV_OIA_DEMO_02'
@AbapCatalog.buffering: { status: #ACTIVE, kind: #SINGLE } }
define view zcdsv_oia_demo_02 (so_guid, bp_guid, bp_role_id, short_name, currency_code, gross_amount_rounded_up,
                               savings, reduced_amount, so_creation_year, status_txt ) as

  select from snwd_so as so
    inner join snwd_bpa as bp on so.buyer_guid = bp.node_key
    left outer join snwd_so_inv_head as so_inv on so.node_key = so_inv.so_guid
  {
    key so.node_key as so_guid,
    key bp.node_key as bp_guid,
    lpad( bp.bp_id, 13,'0' ) as bp_role_id,
    substring( bp.company_name,  0, 10 ) as short_name,

    @Semantics.currencyCode  so.currency_code,   //Type CUKY
    @Semantics.amount.currencyCode: 'CURRENCY_CODE'   ceil( so.gross_amount ) as gross_amount_rounded_up,  //Type CURR
    @Semantics.amount.currencyCode: 'CURRENCY_CODE'   cast(so.gross_amount as abap.fltp) * 0.03  as savings, //Type CURR
    @Semantics.amount.currencyCode: 'CURRENCY_CODE'   cast(so.gross_amount as abap.fltp) + (cast( -so.gross_amount as abap.fltp) * 0.03)
      as reduced_amount, //Type CURR

     -- extract the year from a UTC timestamp (YYYYMMDDhhmmssmmmuuun)
    cast( ceil( cast( so.created_at as abap.fltp ) * 0.0000000001 ) - 1 as abap.numc(4) ) as so_creation_year,

    case so.lifecycle_status
      when 'N' then 'New'
      when 'P' then
        case so.billing_status
          when 'P' then 'Payed'
          else 'Open'
        end
      when 'C' then 'Closed'
      else 'Canceled'
    end as status_txt
  }
where so_inv.payment_status <> 'P'
group by  bp.bp_id, bp.company_name, so.created_at, so.node_key, so.gross_amount,
          so.currency_code, so.lifecycle_status, so.billing_status, bp.node_key
  having sum(so.gross_amount) > 10000
```

WHY CDS VIEW

- CDS leverages the concept of Code Pushdown at the database level which results in better performance as the data intense calculations are performed in the database layer



WHY CDS VIEW

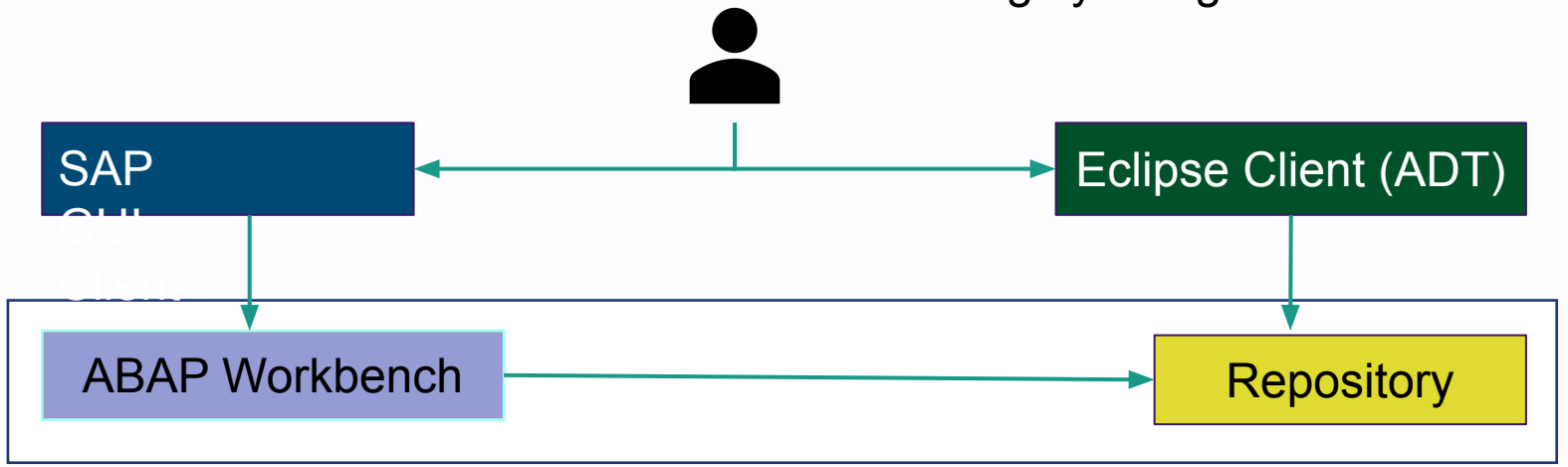
- Hierarchy data models can be created with the help of CDS increases the reusability of the CDS
- By the help of annotations, we can even directly expose the CDS as OData service for accessing and extracting SAP data
- CDS supports various joins, aggregation and numeric functions, string operations
- In CDS Views, calculated fields are possible at runtime
- The concept of DCL (Data Control Language) in CDS helps in fetching only the data for which the user/executer has the authorization to
- CDS Table functions helps in achieving more than just data selection

02

CDS Development

Environment

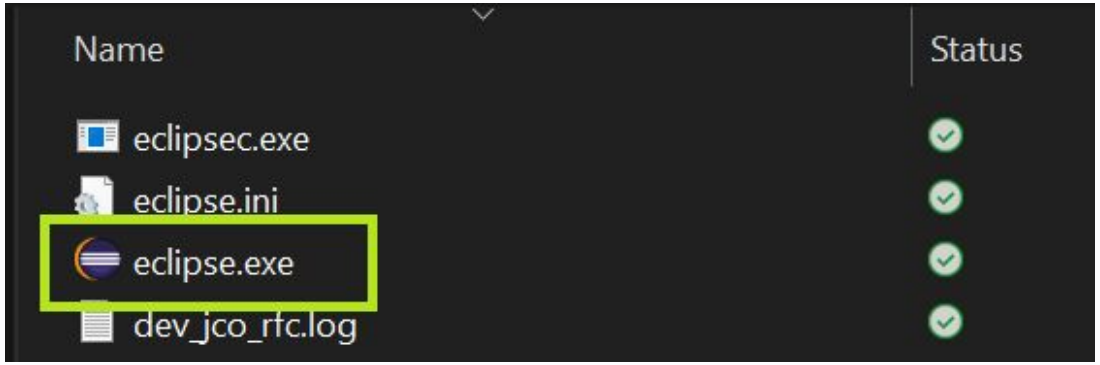
- ABAP Development Tools (ADT) is used for creating CDS Views and cannot be created with the help of SAP GUI
- ADT is integrated in the Eclipse Web IDE
- ADT provides features like refactoring functionality, code completion, auto-insertions and code templates. It also includes Quick Fix feature and is highly navigable.



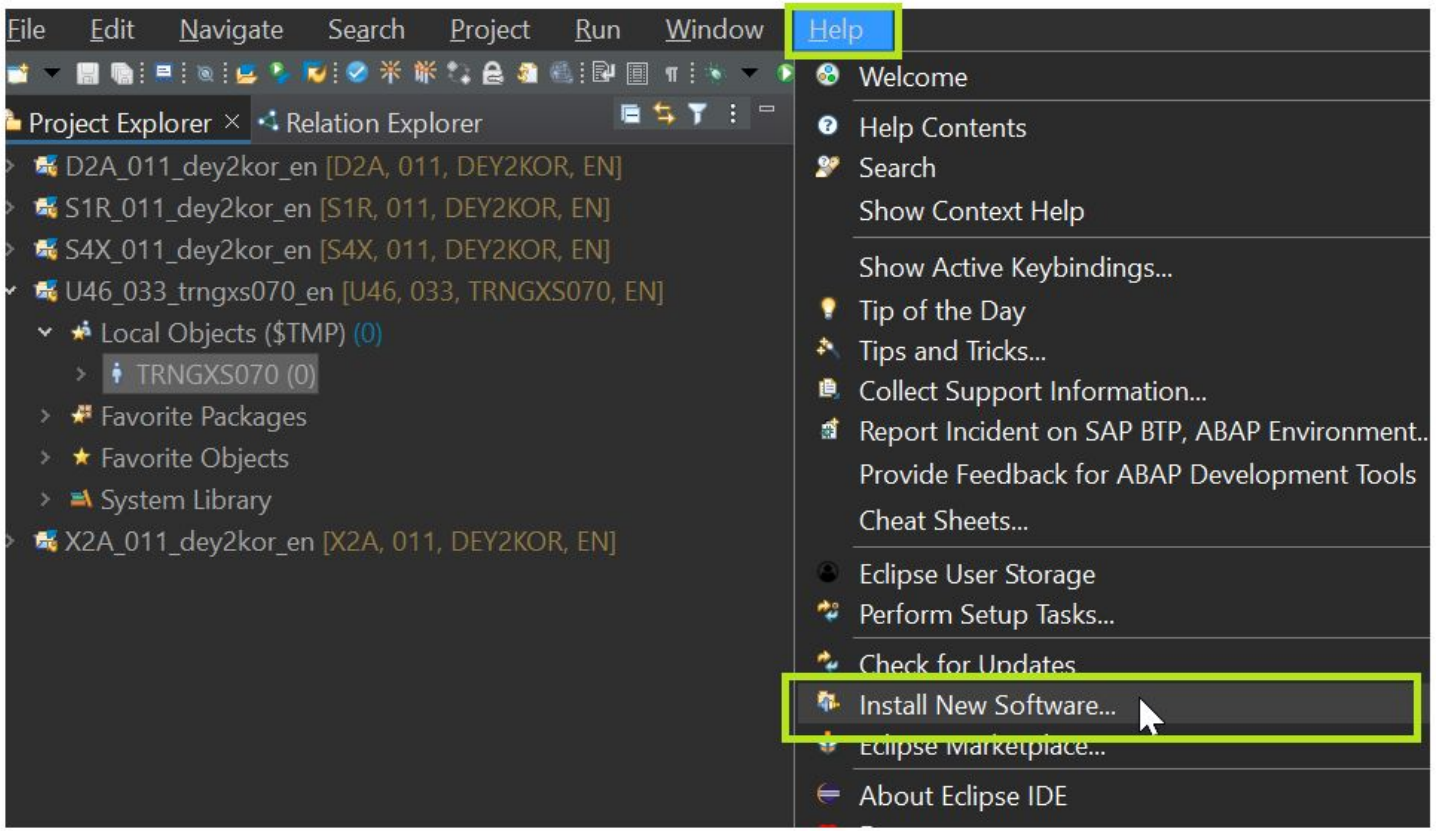
- Steps to follow in order to install Eclipse and ADT in the system

    1. Download the Eclipse from the website https://www.eclipse.org/downloads/

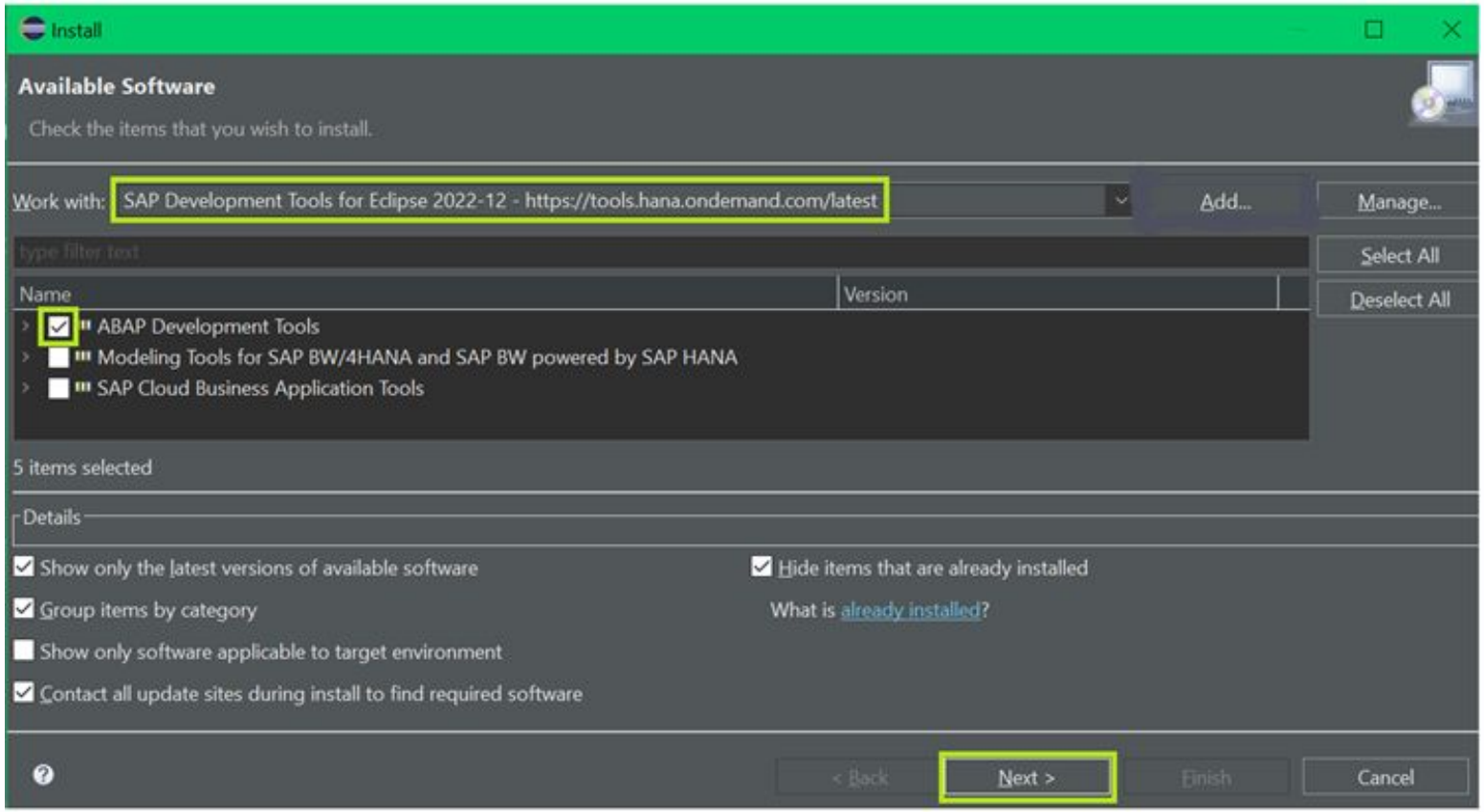
    2. Open the eclipse.exe file after the download is complete



3. In the Menu go to Help -> Install New Software as seen below

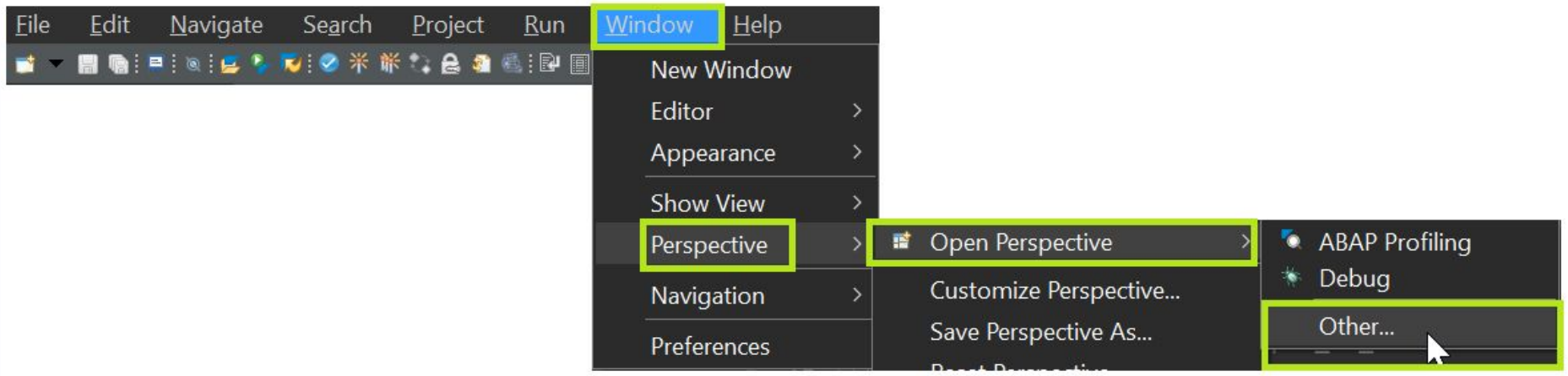


3. In the Work with field, Add the link https://tools.hana.ondemand.com/latest and check the ABAP Development Tools checkbox. Then click on Next until Finish.

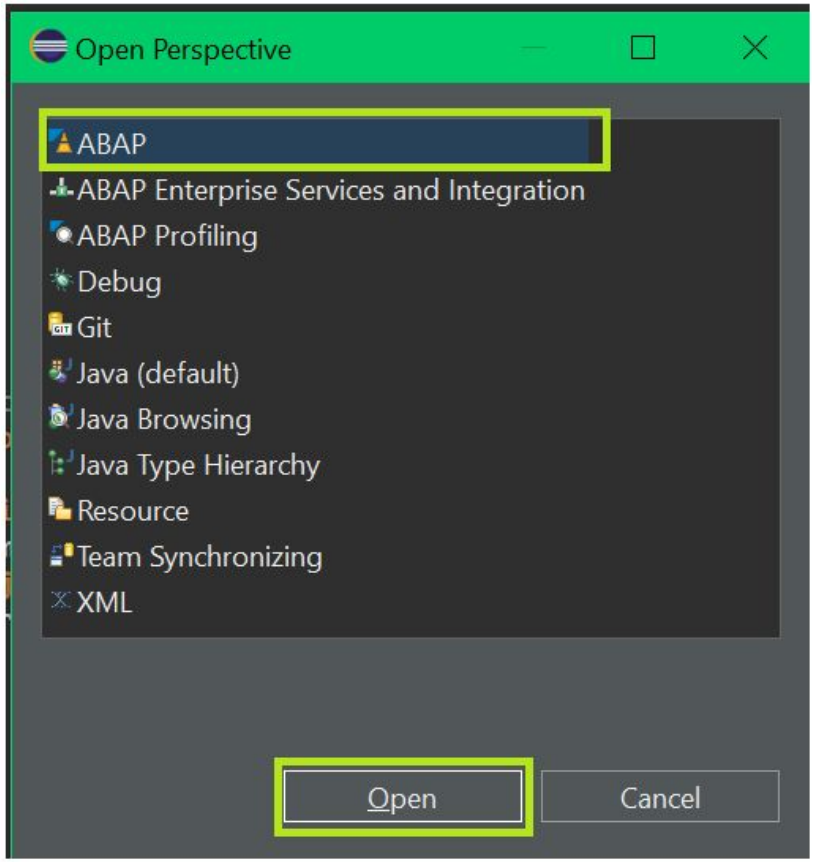


4. Once the download is finished from the link then restart the Eclipse application.

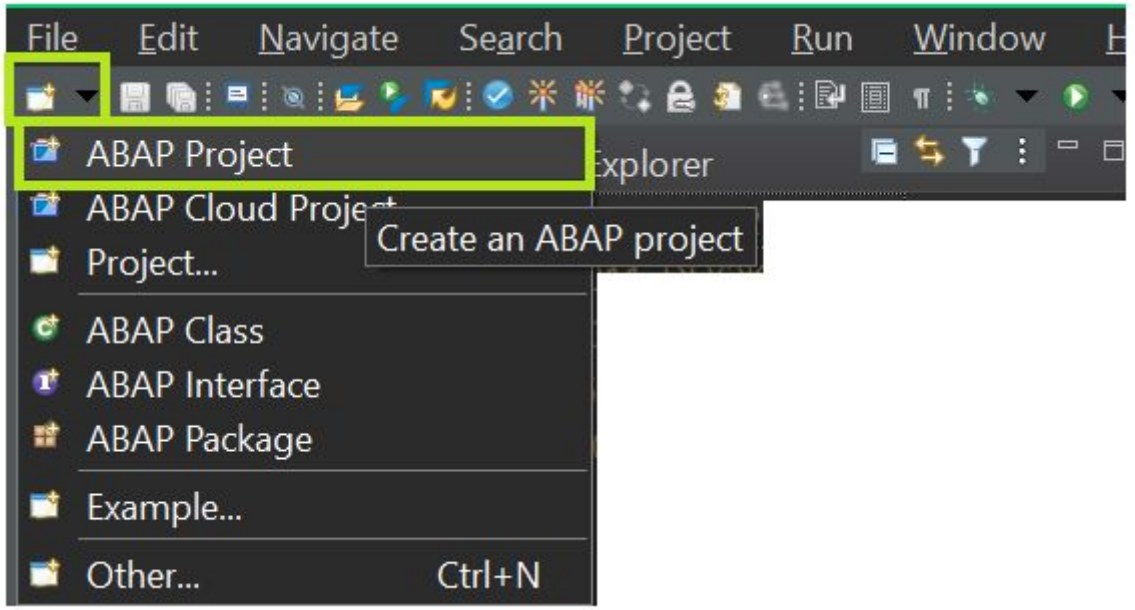
5. No follow the below link from the Menu



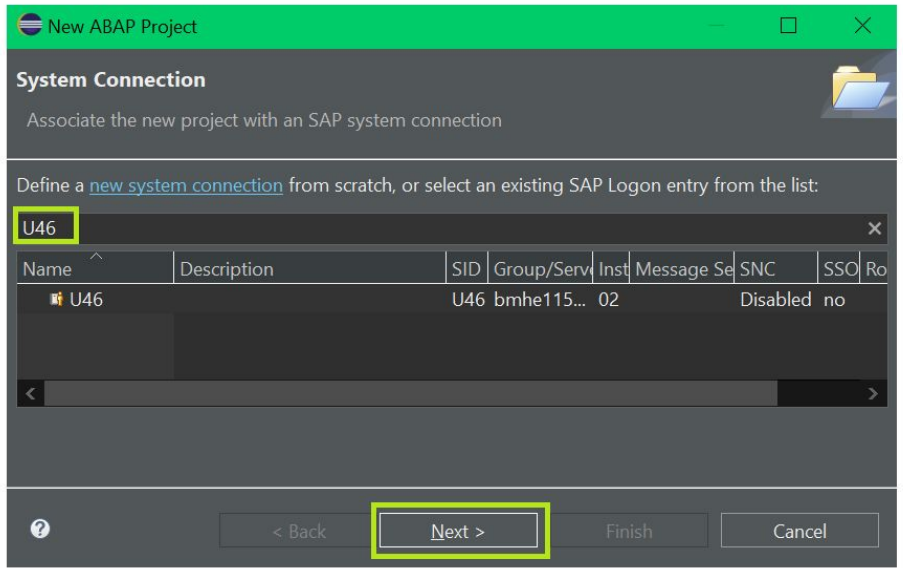
6. Select ABAP and click on OK

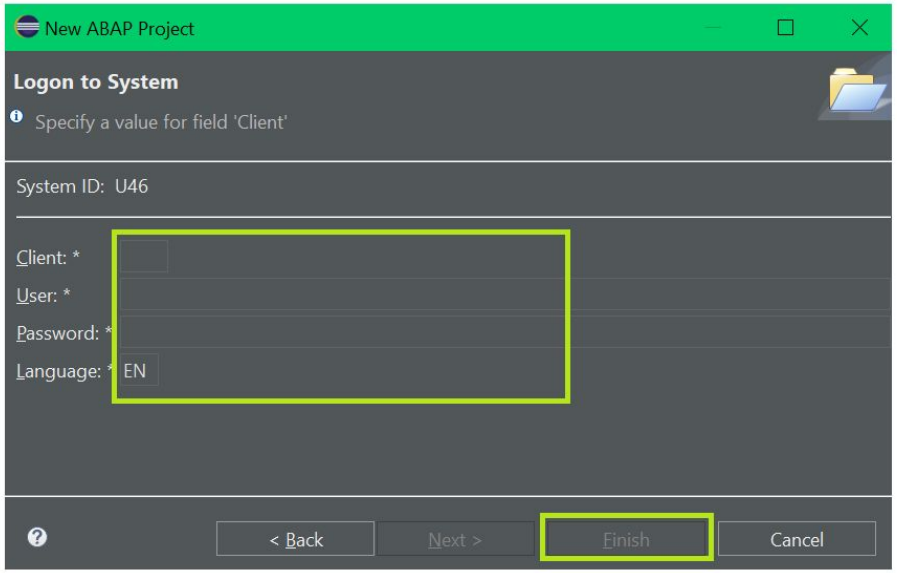


7. To Add the SAP system, click on the icon as shown below and select ABAP Project



7. Select the desired system, click on next and give the necessary details. Once done, click on finish





03

Data Modelling in CDS View and Basic Annotations

- CDS Views are used to select data from the database table(s) and allows us to push down complex logic at the database level
- CDS views can also be used to consume data from another CDS view
- The created CDS views are consumed in ABAP program with the help of SELECT statement like the database tables/views
- With the help of inbuilt functions and annotations the CDS views can be built into an efficient data model
- Annotations begin with @ sign and adds meaning to the sections/line of codes it appears before

- Annotations which are specific to the entire CDS views are written before the ‘define view…’ keyword. For example,

  @AbapCatalog.sqlViewName
  @AbapCatalog.compiler.compareFilter
  @AccessControl.authorizationCheck

- Annotations which are specific to the fields are written before the fields. For example

  @EndUserText.label
  @Semantics.currencyCode
  @Semantics.amount.currencyCode

- The annotation @AbapCatalog.sqlViewName is a mandatory annotation. The SQL view is created with the name provided in this annotation which we can see in the SE11 transaction

Creating the First CDS View - Demo

04

Inbuilt Functions

- Inbuilt functions can be used to manipulate the data being fetched and get the data in a more
  desired format

- SAP has provided multiple inbuilt functions like case, cast, numeric and arithmetic functions which can be directly called from the CDS view definition

- Session variables available, helps us in getting runtime system relevant data which can be readily used to make CDS more flexible and enriched

- For converting currency and quantity as well SAP has provided functions that helps in converting
  the data in real time

- Client handling can be done in CDS views with the help of annotations

CASE Statement

```
define view ZTRNG_CDS_FUNCS
  as select from vbap
{
  key vbeln as SalesDoc,
  key posnr as SalesItem,
      matnr as Product,
      case (matkl)
          when 'YA000' then 'Metals'
          when 'YA002' then 'Plastics'
          when 'YA018' then 'Tools'
          else 'Uncategorised Goods'
      end   as ProductGroup,
```

CAST Operation

- Used to determine type of a calculated field or for converting an existing field into another type

```
define view ZTRNG_CDS_FUNCS
  as select from vbap
{
  key vbeln as SalesDoc,
  key posnr as SalesItem,
      matnr as Product,
      case (matkl)
          when 'YA000' then 'Metals'
          when 'YA002' then 'Plastics'
          when 'YA018' then 'Tools'
          else 'Uncategorised Goods'
      end   as ProductGroup,
      netpr as NetPrice,
      cast( netpr as abap.fltp( 16 ) ) *  0.1 as Discount
```

Numeric and Arithmetic Functions

- Used to carry numeric operations like FLOOR, CEIL, DIV, MOD and arithmetic operations like
  addition, subtraction, multiplication on the data.

```
netpr as NetPrice,
cast( netpr as abap.fltp( 16 ) ) *  0.1 as Discount,
ceil(brgew) as GrossWeight,
vbeaf + 2 as ShippingProcessTime
```

Numeric Functions

FUNCTIONS	USE	RESULT TYPE	EXAMPLE
ABS(arg)	Returns absolute value of arg	Data Type of arg	abs(-2) = 2
CEIL(arg)	Smallest integer number not less than the value of arg	INT4, INT8 (if arg is of type INT8)	CEIL(3.6) = 4 CEIL(-2.8) = -2
DIV(arg1,arg2)	Division without decimal places	Data type arg1, where DEC, CURR and QUAN are implemented after INT4	DIV(15,2) = 6
DIVISION(arg1, arg2, dec)	Division with an additional argument to specify the decimal places	DEC with dec decimal places. The length of the result is the length of arg1 minus the decimal places in arg1 plus the decimal places in arg2 plus dec. This value must not be greater than 31.	DIVISION(15,2,2) = 7.50
FLOOR(arg)	Largest integer number not greater than the value of arg	Data type of arg for the integer types, else DEC without decimal places	FLOOR(3.6) = 3 FLOOR(-2.8) = -3
MOD(arg1, arg2)	Remainder after division	Data type of arg1	MOD(25, 3) = 1
ROUND(arg, pos)	Rounded value of arg	Data type of arg, where INT1 and INT2 are transformed to INT4	ROUND(9.89,1) = 9.9

String Functions

- Used to perform operation on strings and are executed directly at the Database Level

```
        right(charg, 4) as BatchUnit,
        concat( matnr, right(charg, 4) ) as MaterialCode,
        arktx as SalesText,
        upper(arktx) as SalesTextUpper,
        length( matnr ) as ProductIDLength
}
```

Session Variable

- Session variables are global variables of the database with predefined value
- These predefined values are set when the CDS view is used in Open SQL (or) CDS views that are used as data sources in other CDS views
- It is like the SYST table in SAP

Session Variable	ABAP system field
$session.client	sy-mandt
$session.system_date	sy-datum
$session.system_language	sy-langu
$session.user	sy-uname

```
length( 'MATH' ) as ProductLength,
$session.system_date as ExecutionDate,
$session.user as ExecutedBy
```

Client Handling

- In CDS, we use the annotation @ClientHandling to define the client dependency and how the client will be handling

- The below 2 annotations are used to handle client
  1. @ClientHandling.type: #INHERITED / #CLIENT_DEPENDENT / #CLIENT_INDEPENDENT

  2. @ClientHandling.algorithm: #AUTOMATED / #SESSION_VARIABLE / #NONE

- @ClientHandling.type :

  -> ***INHERITED*** - The client dependency is determined by the data sources used in the view. If one of the data sources is client dependent, then the view is client dependent and if all the data sources are client independent, then the view is client independent.

Client Handling

-> ***CLIENT_DEPENDENT*** – The view is client dependent and at least one of the data sources must be client dependent.

-> ***CLIENT_INDEPENDENT*** – The view is client independent, and all the sources must be client independent.

- @ClientHandling.algorithm :

  -> ***AUTOMATED*** – The ON conditions of the view and other clauses are implicitly extended for client columns of the underlying data sources

  -> ***SESSION_VARIABLE*** – Implicit WHERE conditions are added which specifies that select the client that is currently stored in the session variable $session.client

  -> ***NONE*** – Only possible for client independent views.

Currency and Quantity Conversion

- SAP has provided functions which can be used to convert currency and quantity from one unit
  to another as required.

- The function for unit conversion is UNIT_CONVERSION and for currency conversion it is CURRENCY_CONVERSION

- Unit conversion has 3 mandatory parameters and 2 optional parameters to support client verification and error handling

- Currency conversion has 4 mandatory parameters and 6 optional parameters to support fine
  adjustments in the output, client verification and error handling

Currency and Quantity Conversion

Currency

Quantity

Formal Parameter	Optional	Data Type
Quantity	–	QUAN, DEC, INT1, INT2, INT4, FLTP
Source_Unit	–	UNIT
Target_Unit	–	UNIT
Client	X	CLNT
Error_Handling	X	CHAR with length 20

Formal Parameter	Optional	Data Type
Amount	–	CURR
Source_Currency	–	CUKY
Target_Currency	–	CUKY
Exchange_Rate_Date	–	DATS
Exchange_Rate_Type	X	CHAR with length 4
Client	X, –	CLNT
Round	X	CHAR
Decimal_Shift	X	CHAR
Decimal_Shift_Back	X	CHAR
Error_Handling	X	CHAR with length 20

Aggregate Expressions and Group By

- Like classic ABAP, we can also use aggregate functions and group by inside CDS views
- Aggregate functions are used to calculate single values from multiple set of rows fetched the database
- The group by clause helps in grouping the result set by one or more columns

Aggregate Expressions	Descriptions
MIN	Returns the smallest value in the operand
MAX	Returns the largest value in the operand
SUM	Calculates the sum of the values of the operand
AVG	Calculates the average of the values of the operand
COUNT(*)	Returns the number of entries in the result set
COUNT(DISTICT operand)	Returns number of distinct values of operand

Having Clause

- Defines a HAVING condition for the result set of a CDS view after GROUP BY clause is evaluated.
- HAVING condition can only be specified along with the GROUP BY clause.
- HAVING clause removes all the rows from the result set that do not meet the condition specified after HAVING

```
define view ZERPT_CDS_GRP_BY as select from vbap
{
    key vbeln  as SalesDoc,
    min(netwr) as MinAmt,
    max(netwr) as MaxAmt,
    avg(netwr) as AvgAmt,
    count(*)   as ItemCount,
    count( distinct matnr )   as DistinctMaterial

} group by vbeln, pstyv
  having pstyv = 'TAN'
```

Using Inbuilt functions in CDS - Demo

05

Unions, Joins and Associations

Union Views

- Union merges the result sets of multiple SELECT statements of CDS view entities into one result set.
- A prerequisite is that the structures of the results sets are compatible. This means that the results sets must have the same number of elements and that the pairs of elements in each position have a compatible data type.
- Union sets can be a good way of transforming non-standardized database tables into a standardized view of the data.
- Underlying associations must have same ON conditions, cardinalities and target entities.
- If the addition `ALL` is not specified, all duplicate entries are removed from the results set. They are not removed if `ALL` is specified.

MATERIAL	PLANT	STORAGE LOCATION
HD1000	HD1L	100
HD1000	HD1L	200
HD1000	HD2L	100

MATERIAL	PLANT	STORAGE LOCATION
HD1000	HD1L	100
HD2000	HD1L	200
HD2000	HD2L	100

Union

MATERIAL	PLANT	STORAGE LOCATION
HD1000	HD1L	100
HD1000	HD1L	200
HD1000	HD2L	100
HD2000	HD1L	200
HD2000	HD2L	100

Union ALL

MATERIAL	PLANT	STORAGE LOCATION
HD1000	HD1L	100
HD1000	HD1L	100
HD1000	HD1L	200
HD1000	HD2L	100
HD2000	HD1L	200
HD2000	HD2L	100

Joins

- Like classical joins in SQL statement, CDS view also supports joins.
- Joins helps in getting the result from multiple data sources based on the ON condition where we provide the fields name of the data sources
- CDS views support four kinds of joins:
  1. Inner Join
  2. Left Outer Join
  3. Right Outer Join
  4. Cross Join

MATERIAL	PLANT	STORAGE LOCATION
HD1000	HD1L	100
HD1000	HD1L	200
HD1002	HD2L	100

LANGUAGE	DESCRIPTION
EN	Engine Spark
DE	Motorfunke

Cross Join

MATERIAL	PLANT	STORAGE LOCATION	LANGUAGE	DESCRIPTION
HD1000	HD1L	100	EN	Engine Spark
HD1000	HD1L	100	DE	Motorfunke
HD1000	HD1L	200	EN	Engine Spark
HD1000	HD1L	200	DE	Motorfunke
HD1002	HD2L	100	EN	Engine Spark
HD1002	HD2L	100	DE	Motorfunke

Associations

- ASSOCIATIONS are kind of Joins to fetch data from multiple tables on Join conditions, but these are ‘JOINS ON-DEMAND’ (only in case of exposed association)

- JOIN ON-DEMAND means 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 ASSOICATION on other 2 tables will
  not be triggered, and the system would return the results quickly, so it enables high turn-around
  time as compared to regular SQL JOINS.

- Associations are defined with ‘Cardinality’. Syntax : association[<cardinality>]

Associations

- Cardinality concept is not new and holds the same concept with CDS views as well.

- There are 4 types of Cardinality possible based on the data and relationship in the tables joined

    1. 0..1
    2. 0..n or 0..*
    3. 1..0
    4. 1..n or 1..*

Cardinality	Min records of Association Target	Max records of Association Target
[1]	0	1
[0..1]	0	1
[1..1]	1	1
[0..*]	0	Unlimited
[1..*]	1	Unlimited
Not Specified	0	1

Associations

- Associations need to be exposed in the CDS view to use it or get the fields
- Association can be either of types exposed or ad-hoc
- In exposed association, we only specify the association name in the selection list whereas for
  ad-hoc association, the requested fields from the association are also specified in the selection
  li

-EXPOSED Association

```
@AbapCatalog.sqlViewName: 'ZSQL_VIEW_ASSTN'
@AbapCatalog.compiler.compareFilter: true
@AbapCatalog.preserveKey: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS View with Associ
define view ZCDS_VIEW_ASSOCIATIONS as sel
association [1] to spfli as _flights
    on sf.carrid = _flights.carrid {
    //sf
    key sf.carrid,
    sf.connid,
    sf.fldate,
    sf.price,
    sf.seatsocc_b,
    sf.seatsmax_f,
    sf.seatsocc_f,

    _flights // Make association public
}
```

Similar to Join, we need key fields to Associate 2 different tables.

The key field which we used to Associate must be part of the selection

Make Association Public i.e. Expose the association. This will not create any Join beforehand but do it need basis.

-AD-HOC Association:

```
@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_ASSOCIATIONS as select from sflight as sf
association [1] to spfli as _flights
    on sf.carrid = _flights.carrid {
    //sf
    key sf.carrid,
    sf.connid,
    sf.fldate,
    sf.price,
    sf.seatsocc_b,
    sf.seatsmax_f,
    sf.seatsocc_f,

    _flights.airpfrom, // Make association public
    _flights.airpto
}
```

Difference between Joins and Associations

Joins	Associations
Join is always performed even if the fields of the joined tables are not selected	Joins are only performed when fields from the exposed association are requested
Only fields in the selection list can be reuested from the joined data source	In case of exposed association all the fields of the association are availbe for request

CDS with Parameters

ZCDS_VIEW

```
1  @AbapCatalog.sqlViewName: 'ZcdsView'
2  @AbapCatalog.compiler.compareFilter: true
3  @AccessControl.authorizationCheck: #CHECK
4  @EndUserText.label: 'Define View CDS_ENTITY'
5
6  define view zcds_View
7    with parameters
8      carid : abap.char( 3 ),
9      conid : s_conn_id
10   as select from spfli
11 {
12   spfli.carrid,
13   spfli.connid
14 }
15 where
16       spfli.carrid = $parameters.carid
17   and spfli.connid = $parameters.conid
```

ABAP Data type as inline declaration

Data element can also be used

[BWR] ZFLIGHT_02

```
@AbapCatalog.sqlViewName: 'ZSQL_FLIGHT1'
@AbapCatalog.compiler.compareFilter: true
@AbapCatalog.preserveKey: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'flights'
define view ZFLIGHT_02
with parameters  P_CARRID : ABAP.char( 3 ),
                 P_MANDT : ABAP.clnt(3)
as select from spfli {
mandt,
carrid,
cityfrom,
cityto,
fltime as DURATION_MIN,
CONCAT(cityfrom, cityto) as TEST1,
CONCAT_WITH_SPACE(cityfrom, cityto, 3 ) as TEST2,
case
    when fltime < 100 then 'SHORT'
    when fltime < 300 then 'MEDIUM'
    else 'LONG'
end as FLIGHT_DURATION,
fltime*60 as DURATION_SEC
}
where carrid = :P_CARRID and
        mandt = $parameters.P_MANDT;
```

Block A

Block B

Block C

Block D

Block E

Block F

Using CDS Joins and Association and Parameters - Demo

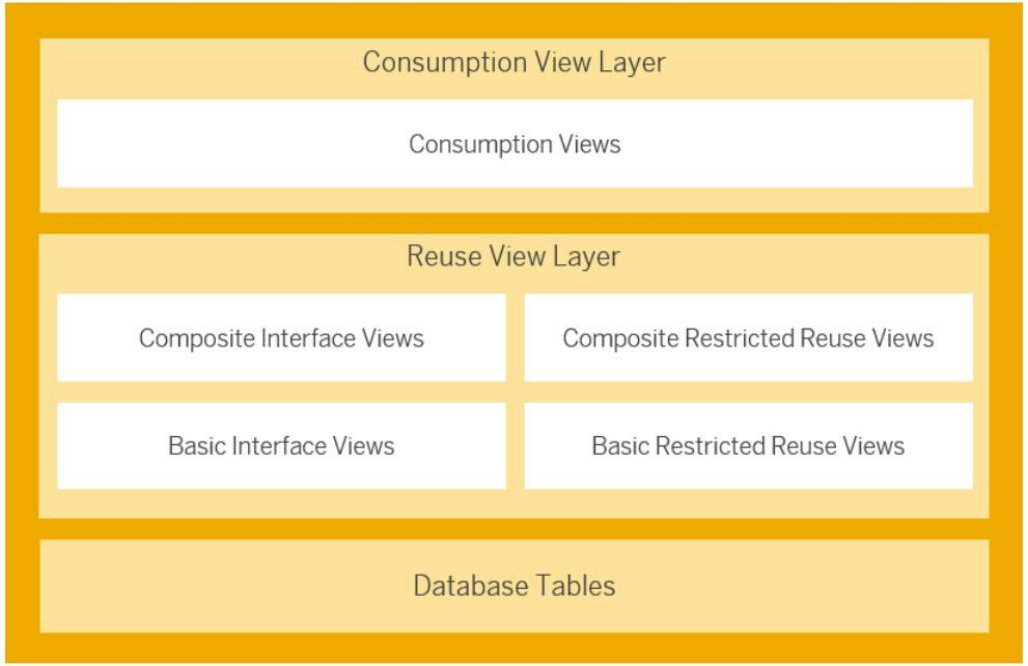
06

Virtual Data Model

(VDM)

Virtual Data Model (VDM)

- VDM provides access to business data and is realized with the help of CDS.
- VDM consists of CDS views and exposes data based with its business semantics making it easier to consume. It forms the central data model of the application data.



Virtual Data Model (VDM)

- VDM structure can be mainly categorized into 3 types:

1. Basic View:

- Describes the lowest layer of the VDM and are the only views that access the database tables directly

- The table fields in the database layer have names that are hard to understand. In the basic interface view layer, they're renamed in terms of business semantics, resulting in more comprehensible field names.

- Basic interface views enrich the data model derived from the underlying database tables with additional metadata, such as annotations and relations between different views.

- Basic interface views have the annotation @VDM.viewType: #BASIC.

Virtual Data Model (VDM)

2. Composite View:

- Are based on basic interface views and may also have association to other composite views

- They don’t access the database tables directly, but only through the basic view layer

- Composite interface views combine multiple basic interface views or another composite interface views to form new semantic entities. These can be used, for example, as analytical cube views

- Composite interface views have the annotation @VDM.viewType: #COMPOSITE

Virtual Data Model (VDM)

3. Consumption View:

- The top layer of the VDM CDS view stack is made up of consumption views

- They're based on reuse views and access the database tables only indirectly through the reuse view layer. These views are designed for a particular purpose with specific requirements, such as use by a particular transactional or analytical app

- They are for consumption by the UI tools – Bex, Lumira, Webi, Analysis for Office etc. to be accessed by business users in the from of a report. These views can consume all other Basic or Composite Views to read data and create a final data set to be fed into the UI tools for reports based on business requirements

- Consumption views have the annotation @VDM.viewType: #CONSUMPTION

Virtual Data Model (VDM)

- More VDM types

1. Restricted reuse:

- Like basic or composite interface views, but they're not intended for reuse by customers and partners

- They may have the annotation @VDM.viewType: #BASIC or @VDM.viewType: #COMPOSITE.

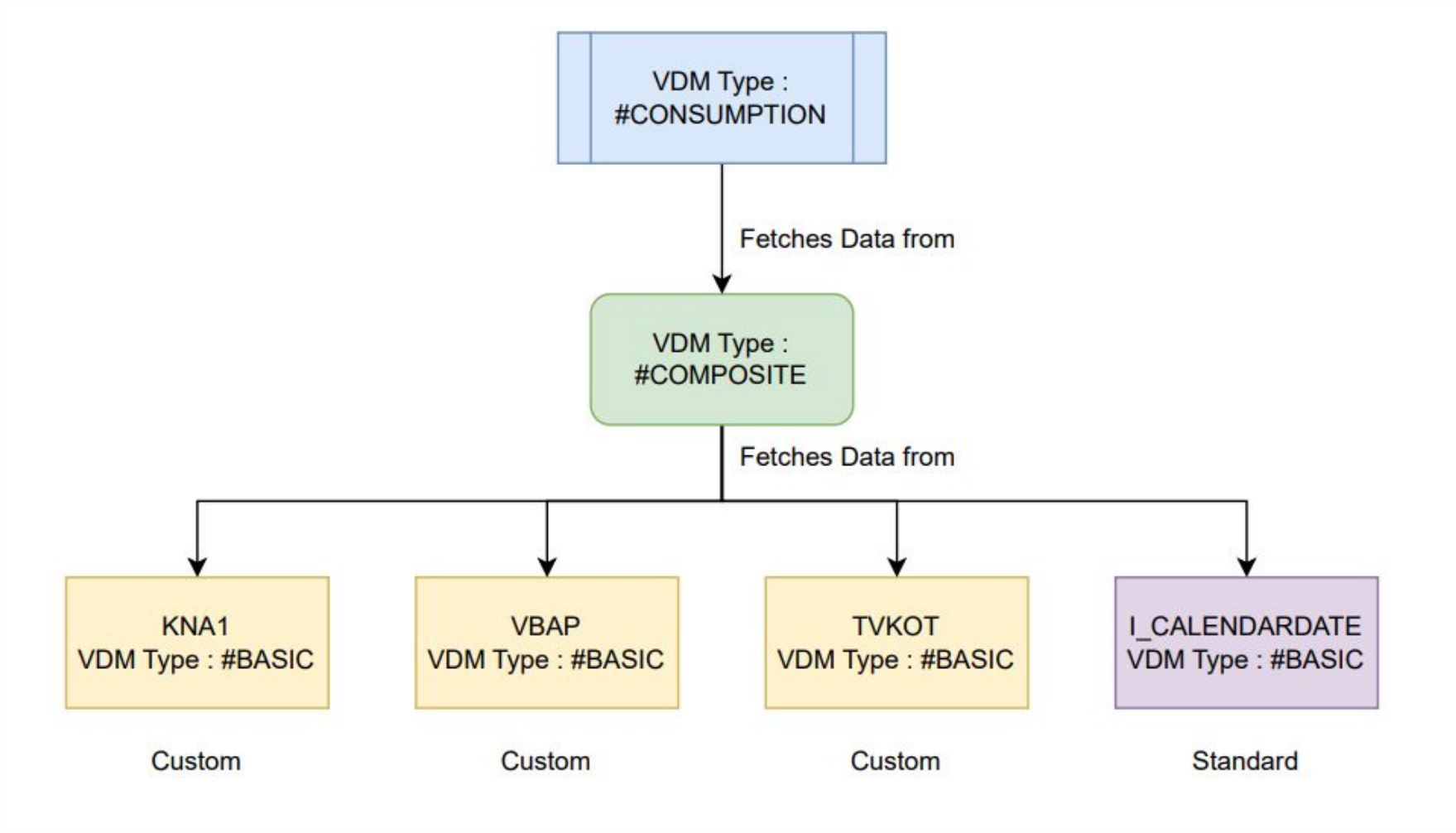
2. Transactional processing:

- These CDS views are mainly designed to not only expose data for reading, but to enable actions such as creating, changing, or deleting instances of a RAP business object.

Virtual Data Model (VDM)

3. Remote API:

- These CDS views are designed to form the basis for external APIs

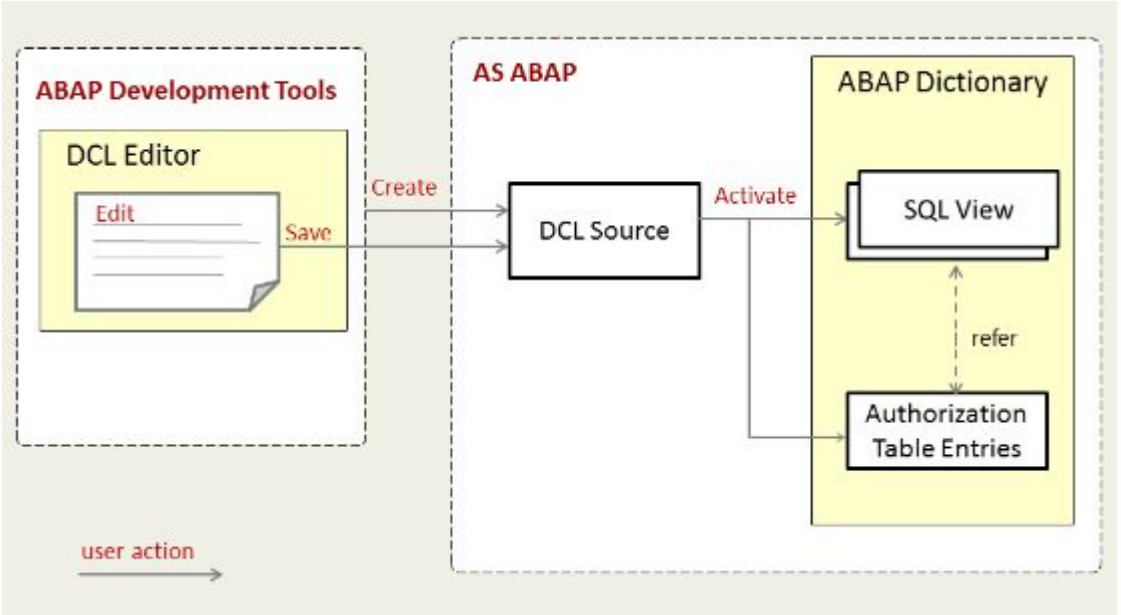

VDM Type :
#CONSUMPTION
Fetches Data from
VDM Type :
#COMPOSITE
Fetches Data from
KNA1
VDM Type : #BASIC
VBAP
VDM Type : #BASIC
TVKOT
VDM Type : #BASIC
I_CALENDARDATE
VDM Type : #BASIC
Custom
Custom
Custom
Standard

07

CDS Development

Access Control

- ABAP Core Data Services (CDS) has its own authorization concept based on a data control language (DCL).
- The authorization concept of ABAP CDS uses conditions defined in CDS and can draw upon classical (PFCG) authorizations to check the authorizations of users.



Authorization Check Types

#CHECK

#MANDATORY

#NOT_REQUIRED

#NOT_ALLOWED

#PRIVILEGED_ONLY

Note: Access control will not be applied to a CDS if a consumption source is another CDS which already has Access Control / DCL assigned.

Example: If CDS_A has DCL created and CDS_B is accessing data from CDS_A then the DCL will not be applied during accessing the data from CDS_A

CDS Views for Asset Management | SAP Help Portal

- **Access Control With Literals**

```
@MappingRole: true
define role demo_cds_role_literal {
  grant select on demo_cds_name
  where carrid = 'LH'; }
```

- **Access Control with PFCG**

```
@MappingRole: true
define role demo_cds_role_pfcg {
  grant select on demo_cds_name
  where (carrid) =
  aspect pfcg_auth (s_carrid, carrid, actvt='03'); }
```

- **Access Control with Generic Aspect**

```
@MappingRole: true
define role demo_cds_role_user {
  grant select on demo_cds_name
    where
      uname ?= aspect user; }
```

- **Access Control Inheritance**

```
@MappingRole: true
define role demo_cds_role {
  grant select on demo_cds_name
              inherit demo_cds_role_super; }
```

08

CDS Development

OData Service Creation

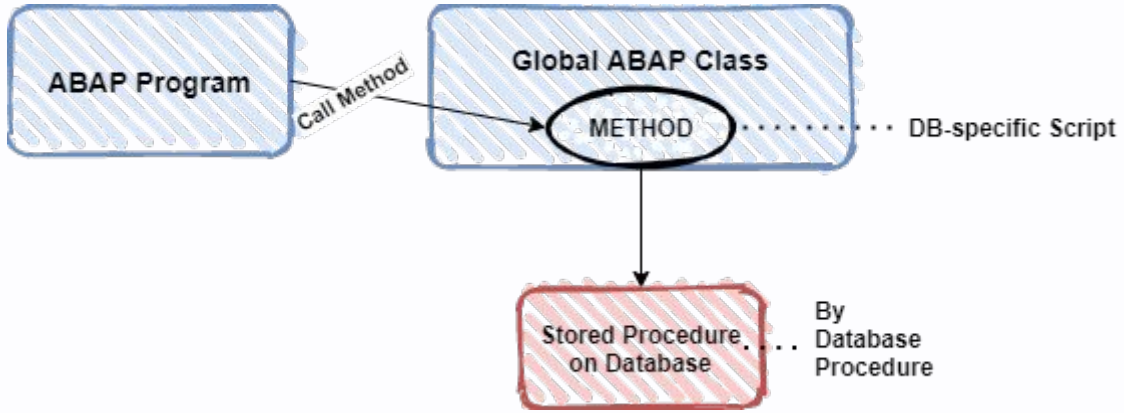
- @ODATA.Publish : True
- The above annotation is used for creating OData service from CDS.
- This approach is useful only when we are exposing one CDS as a service
- IF a service is for CRUD operations then approach should be of Odata service using SADL
- The Process of ODATA via CDS is from SEGW

09

CDS Development

Table Function with AMDP

- ABAP-Managed Database Procedures is one of the recommended patterns for use in ABAP code optimization within the context of ABAP development on SAP HANA.



IF_AMDP_MARKER_HDB

10

CDS Development

UI Annotations

- UI.HeaderInfo : An entity, its title, and an optional short description, the name of its entity in singular and plural form, and optional image URLs for the individual entity.
- UI.identification : Represent an ordered collection of specific data fields that together with headerInfo identifies an entity to an end user.
- UI.lineitem : Represent an ordered collection of data fields that is used to represent data from multiple data instances in a table or a list.
- UI.selectionField : These are usually used in an initial page floorplan as filter bar.

- AccessControl Annotations | SAP Help Portal