INTRODUCTION

Need for CDS view?

SAP technology works on two prime problems in the industry:

- Better user experience- good responsiveness.
- Improving data latency- result should come in matter of time for quick decisions.

To meet its goals SAP introduced the following:

- 2011 → SAP HANA
- 2013 → SAP Fiori

Role for an ABAP developer?

Make use of both SAP HANA & Fiori techs for developing app for S/4HANA

SAP business suite ECC 6.0

client

UI : SAP GUI

Application server

SELECT mantr
matkx
FROM mara
INTO TABLE itab

Open SQL

DB INTERFACE

Native SQL

database

RDBMS: Any DB supported by SAP (Oracle, MS SQL)

SAP business suite 4 SAP HANA

client

UI : SAP Fiori

Latest AS ABAP 7.53. etc.

Enhanced Open SQL,CDS View, AMDPs

database

Purely HANA DB

So what's in it for an ABAP developer?

S/4HANA system uses HANA as its native database to store the data.

Best feature of SAP HANA base is IN MEMORY COMPUTING

In memory computing: storage of memory in main memory instead of slow hard drives.

- This helps the companies to analyse the data in main memory in a matter of seconds or minutes instead of long hours.

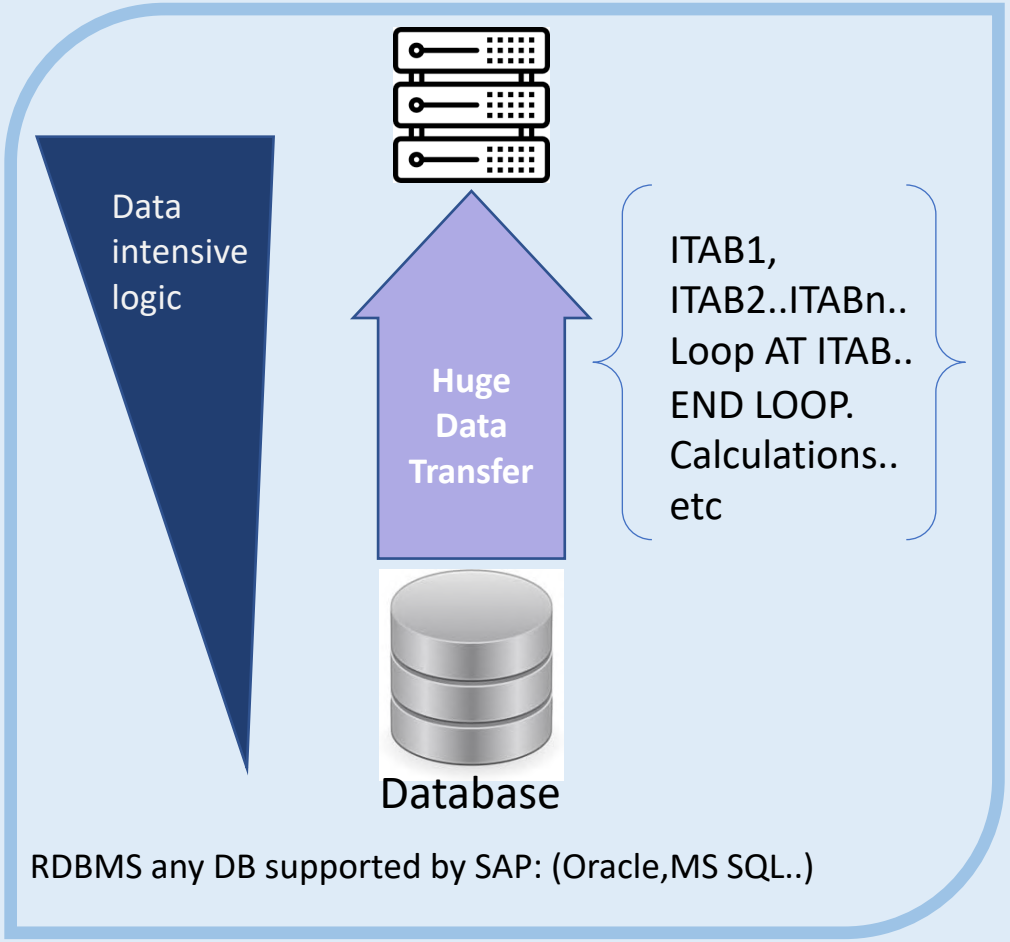
To get the best out of in memory computing, as an ABAP developer you need to take care of two things:

- Perform data intensive calculations at the database level
- Avoid moving unnecessary large volume data into application server.

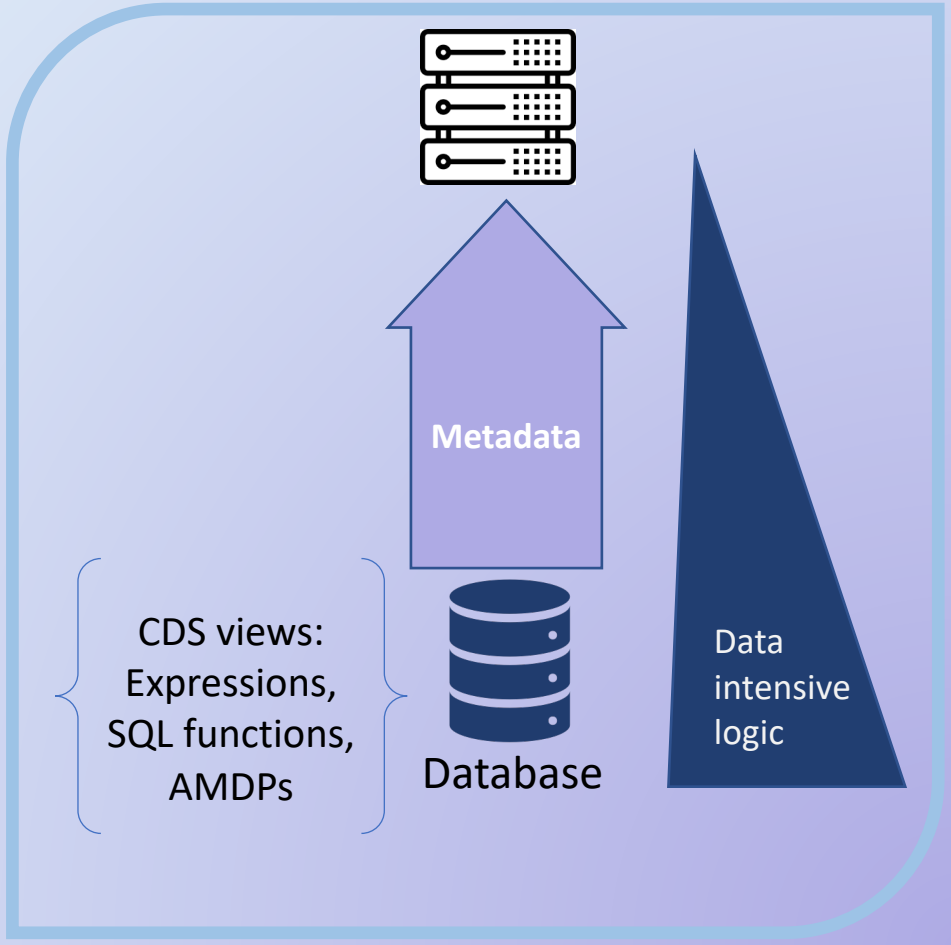
HOW TO ACHIEVE ?

Core data service (CDS) & Enhanced Open SQL : We can perform data intensive logic at DB level

Data to code(before SAP HANA)

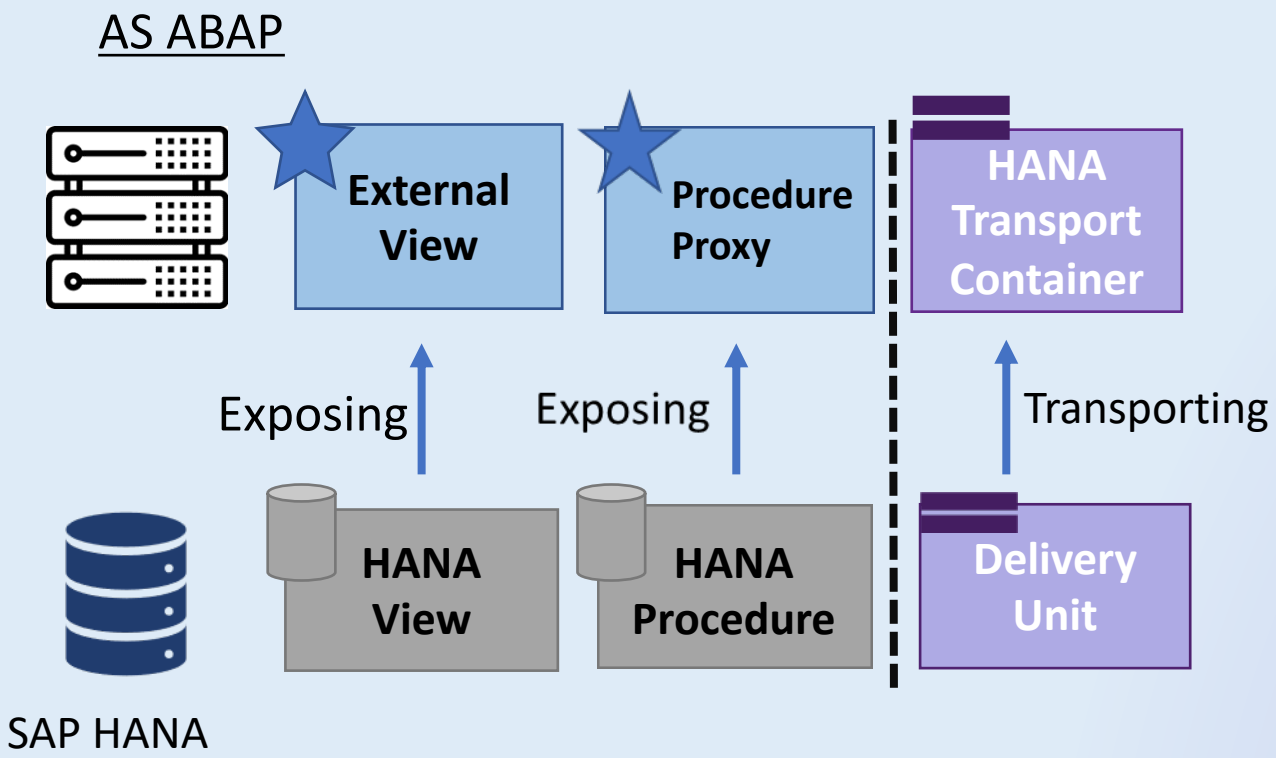


Code to data(with SAP HANA)

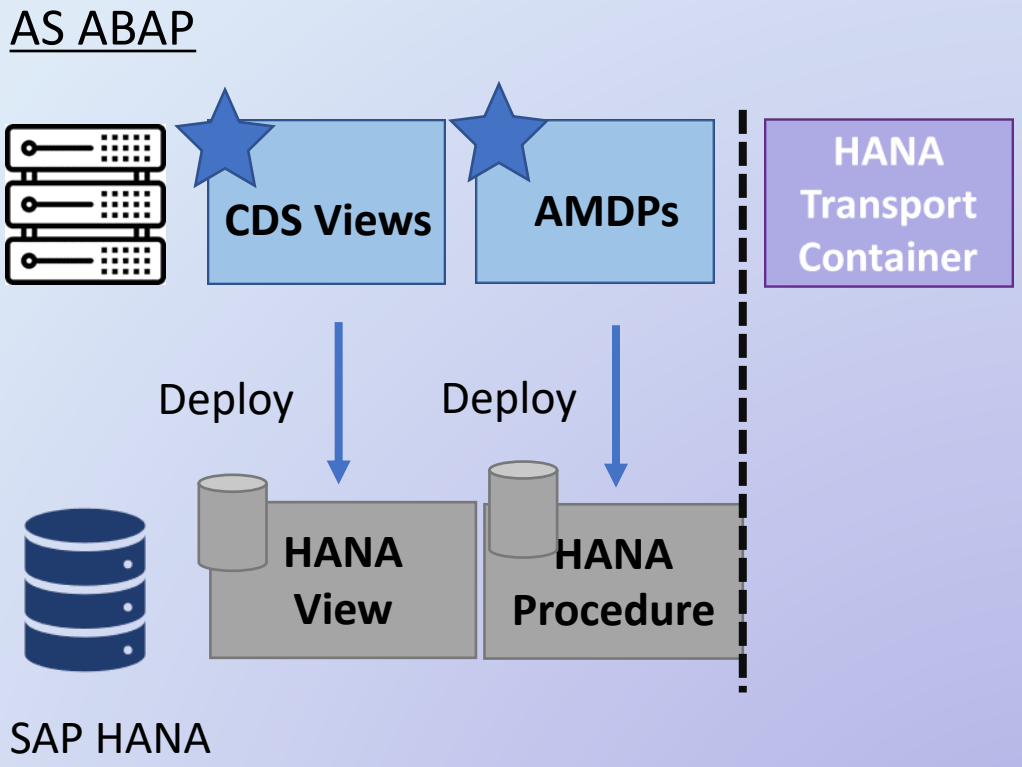


What is the approach to develop S/4HANA?

Bottom up approach(7.40 SPO2 above)



Top Down Approach (Recommended since 7.40 SPO5)



Possible approaches based on AS ABAP releases

AS ABAP RELEASE	How to achieve Code Pushdown
>=7.40 SP05 or above	❑ Enhanced Open SQL ❑ CDS Views ❑ AMDP (ABAP Managed Database Procedure)
Below NW 7.40 SP05	❑ HANA View (Attribute Views, Analytics View, Calculation Views) ❑ Hana Database Procedures Note : Need to create External Views & Database Procedure Proxies in AS ABAP
Below NW 7.40 SPs	❑ HANA Views &Database Procedures can be consumed using Native SQL &ADBC from AS ABAP

CDS(Core Data Service)

Main advantage of CDS

We can leverage the features of SAP HANA. ABAP CDS is open, meaning that you can use it on all database platforms supported by SAP (but recommended with SAP HANA to get best out of it).

Why can’t I just use database views which are available in Data dictionary (SE11)?

1.Database views are very plain.

2.Cannot support application specific logic on the go within the view.

3.Multiple steps involved to expose the view as OData service (SEGW transaction).

4.No support for annotation (we can add through by coding from SEGW& MPC_EXT.

Quick comparison between Normal view & CDS view

	CDS Views in Eclipse ADT	Database View in SE11
Calculation at field level?	✓	✗
Supports RIGHT OUTER JOIN, UNION?	✓	✗
Supports for Annotations to integrate with Fiori apps?	✓	✗
Ease of exposing as Odata Service ,is it simple?	✓	✗
Supports all databases?	✓	✓
Supports SQL functions at field level?	✓	✗

ABAP developer job

Working with SAP HANA (2011) For Data Modelling and Code pushdown

- **Enhance Open SQL**
- **CDS Views**
- **AMDP**
- **QUERIES**

From AS ABAP Using Eclipse ADT

SAP Fiori (2013) to develop Fiori Apps

- **ABAP RESTful Programming**
- **Work with Fiori elements using different annotations in CDS views**
- **No JAVA script required**

Speed up your ABAP development using shortcuts

Edit

Ctrl+Shift+A	Open development object
Ctrl+F2	Check development object
Ctrl+F3	Activate development object
Ctrl+Shift+F3	Activate all inactive objects
Ctrl+Space	Code completion
Ctrl+1	Quick fix proposal
Ctrl+<	Add comment
Ctrl+Shift+<	Remove comment
Shift+F1	Format source aka pretty printer

Help

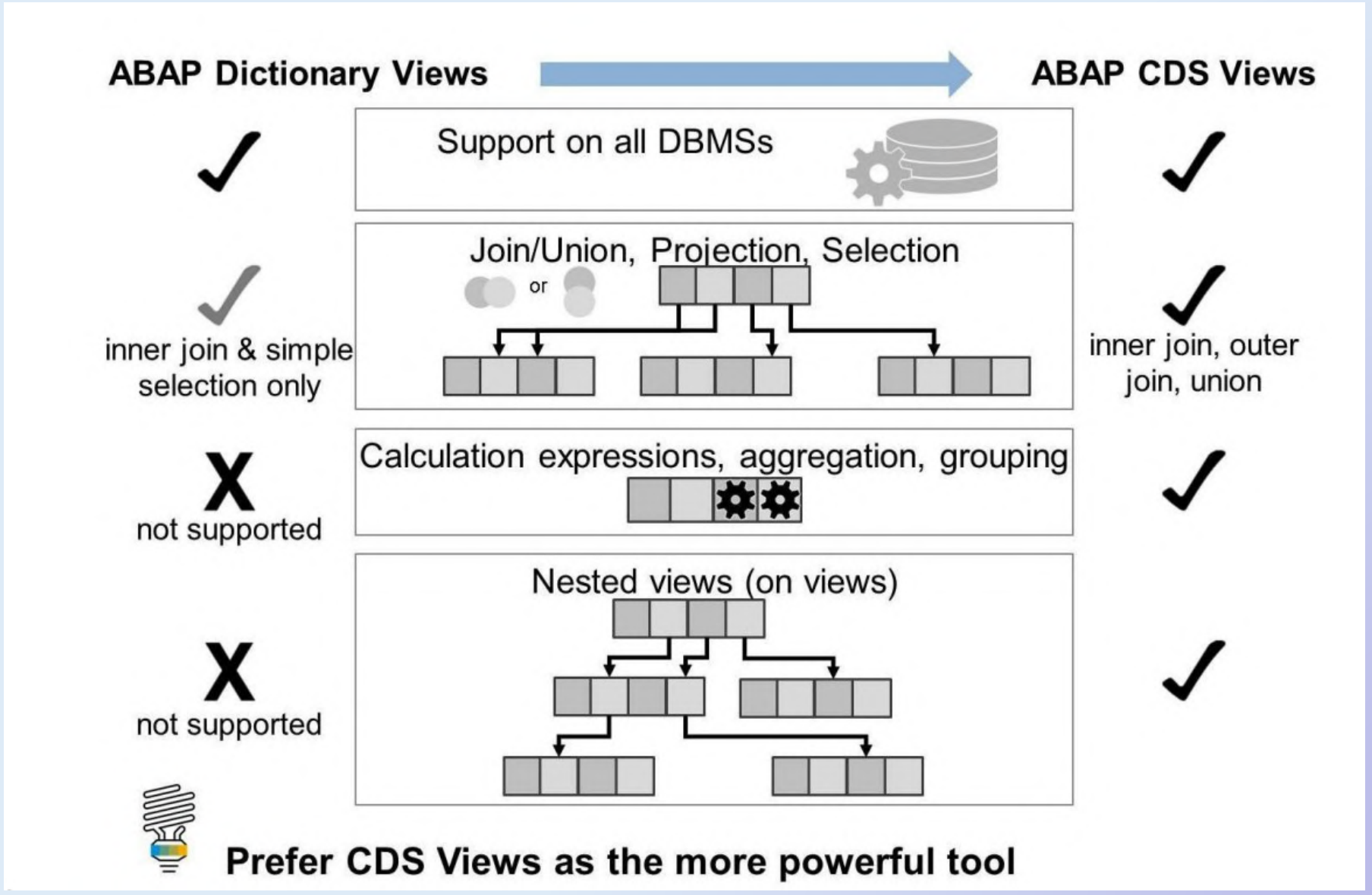
F1	ABAP keyword documentation
F2	Show code element information
Ctrl+3	Search for commands & views
Ctrl+Shift+L	List all keyboard shortcuts

Navigate

F3	Open definition
Alt+Left	Backward history
Alt+Right	Forward history
Ctrl+T	Quick hierarchy
F4	Open Type Hierarchy
Ctrl+O	Quick outline
Ctrl+Shift+G	Where-used list

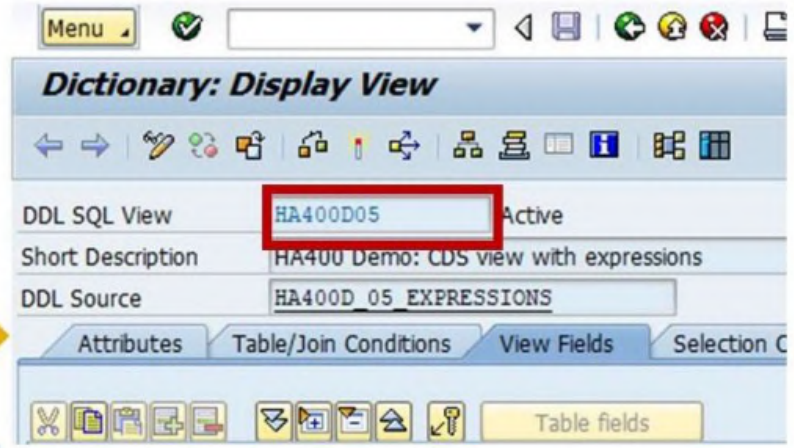
Run, Debug

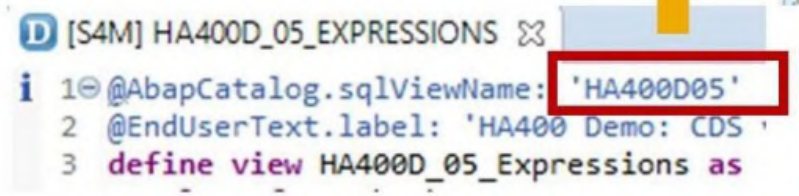
F8	Run current ABAP object
Alt+F8	Select & run ABAP application
Ctrl+Shift+B	Toggle breakpoint
F5, F6, F7, F8	Step into, over, return, resume
Ctrl+Shift+F10	Execute ABAP unit tests
Alt+F9	Profile development object


ABAP Dictionary Views
ABAP CDS Views
Support on all DBMSs
Join/Union, Projection, Selection
or
inner join & simple selection only
inner join, outer join, union
Calculation expressions, aggregation, grouping
not supported
Nested views (on views)
not supported
Prefer CDS Views as the more powerful tool

ABAP Dictionary and DB object correspond to active ABAP CDS View

- ABAP Dictionary view (read-only)
- Database view
- Name specified in DDL source annotation:
  `@AbapCatalog.sqlViewName: 'HA400D05'`






```
CREATE VIEW "HA400D05" AS SELECT
  "SBOOK"."MANDT" AS "MANDT",
  CONCAT(
    RTRIM (
      "SBOOK"."CARRID"
    ),
    RTRIM (
      "SBOOK"."CONNID"
    )
  ) AS "FLIGHTNO",
  "SBOOK"."FLDATE",
  "SBOOK"."BOOKID",
  "SBOOK"."CUSTOMID" AS "CUSTOMER",
  CONCAT(
    RTRIM (
```

Most simple CDS View

```
@AbapCatalog.sqlViewName: 'HA400D01'
define view HA400D_01_Simple as
select * from sbook
```

CDS View as projection and using column alias to rename a column

```
@AbapCatalog.sqlViewName: 'HA400D02'
define view ha400D_02_Field_List as
 select carrid   as airline,
        connid   as flightnumber,
        fldate   as flightdate,
        bookid   as id,
        customid as customer
 from sbook
```

Alternative syntax: column list in curly brackets {…} *after* FROM clause

```
@AbapCatalog.sqlViewName: 'HA400D03'
define view ha400D_03_Brackets as
 select from sbook
{
   carrid   as airline,
   connid   as flightnumber,
   fldate   as flightdate,
   bookid   as id,
   customid as customer
}
```

```
@AbapCatalog.sqlViewName: 'S4D430_CARR'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Demo: Simple Projection'


define view S4D430_Carrier
   as select from scarr
 {
    @EndUserText.label: 'Airline'
    carrid,
    carrname,
    currcode,
    url
 }
```

View Annotations

Element Annotation

- Enrich a view definition with metadata
- Start with character “@“
- May be related to the complete view (view annotations)
  or to individual parts (element annotations, parameter annotations, etc.)
- Are mostly optional
  (exception: `@AbapCatalog.sqlViewName` for DDIC-based views)

Figure 34: New ABAP SQL Syntax in NW 7.40

Classical Syntax (Before NW 7.40 SP05)

```
SELECT carrid connid fldate SUM( paymentsum ) currency
    FROM sflight
    INTO TABLE gt_flights
    WHERE fldate > sy-datum
    GROUP BY carrid connid fldate currency
    ORDER BY carrid connid.
```

New Syntax Rules in NW 7.40

```
SELECT carrid, connid, fldate, SUM( paymentsum ), currency
    FROM sflight
    INTO TABLE @gt_flights
    WHERE fldate > @sy-datum
    GROUP BY carrid, connid, fldate, currency
    ORDER BY carrid, connid.
```

Variables must be escaped with the @ symbol

Column lists are now comma-separated

Figure 41: General Syntax Rules of CDS DDL

- **Allowed Characters**

  Only ASCII-characters.

- **Keywords:**

  All uppercase, all lowercase, or lowercase with uppercase initial letter.
  No mixed uppercase and lowercase.
  Allowed: SELECT, select, Select. Not allowed: SeLect, seleCT.

- **Literals:**

  Numeric literals always in full, with decimal point if necessary.
  Allowed: 1, 2.0, or 0.5. Not allowed: .5, 1,3

  Character literals enclosed in single quotations marks (').
  'LH', '00001'

- **Comments:**

  Explicit end: enclosed by /* and */
  Rest of line: two forward slashes (//)

- **Separators:**

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

- **Protected Words:**

  Certain keywords cannot be used as self-defined names.

Figure 45: Aliases for Tables And Fields

```
define view S4D430_Connection2_Alias as select
            from       spfli as c
            inner join scarr as a
                    on c.carrid = a.carrid

            {
              c.carrid,
              c.connid,
              a.carrname,
              a.currcode as currency,
              c.cityfrom,
              c.cityto,
              airpfrom,
              airpto
            }
```

Alias for table name

Alias for element

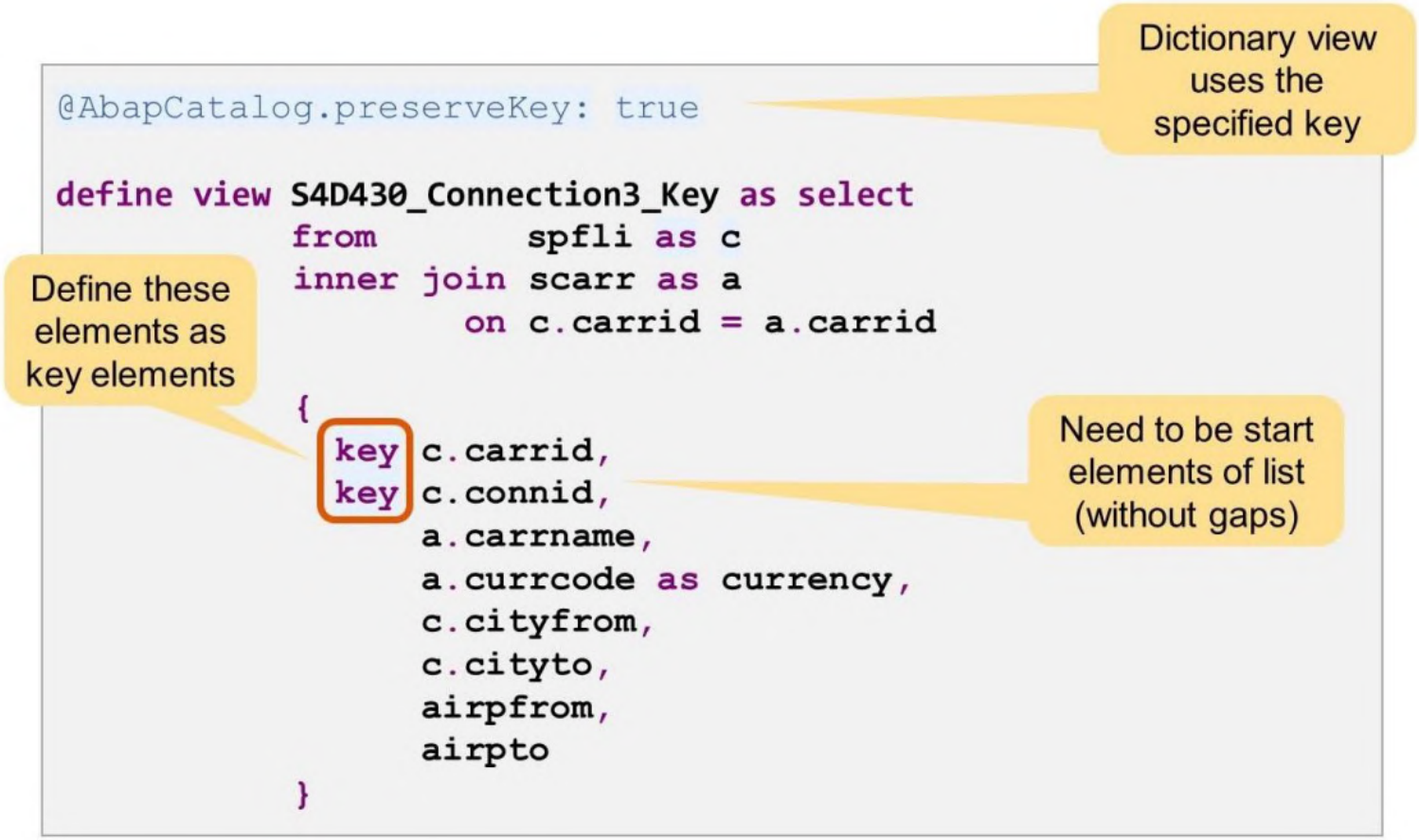
(mandatory for certain expressions)

Figure 46: Selections

```
define view S4D430_Connection4_Selection as select
            from       spfli as c
            inner join scarr as a
                    on c.carrid = a.carrid

            {
              key c.carrid,
              key c.connid,
                  a.carrname,
                  a.currcode as currency,
                  c.cityfrom,
                  c.cityto,
                  airpfrom,
                  airpto
            }
            where c.fltype <> 'X' //exclude charter flights
```

Add WHERE clause to define a pre-selection

Figure 47: Key Definition



Enrich CDS data models with additional metadata

- Annotations begin with @

Different ABAP specific information can be added

- Name of the associated SQL / ABAP Dictionary view - mandatory
- Buffering
- Client dependency
  Default: client-dependent
- Reference information for Amount and Quantity columns
- …

```
@AbapCatalog.sqlViewName: 'HA400D04'
@AbapCatalog.buffering.status:'#ACTIVE'
@AbapCatalog.buffering.type:  '#SINGLE'
@ClientDependent:'true'    //default

define view HA400D_04_Annotations as
  select from sbook
{
 carrid    as airline,
 connid    as flightnumber,
 fldate    as flightdate,
 bookid    as id,
 customid  as customer,

 @Semantics.amount.currencyCode:'CURRENCY'
 loccuram as amount,              //Type CURR

 @Semantics.currencyCode: true
 loccurkey as currency            //Type CUKY
}
```

Figure 48: Classification of SAP Annotations

	ABAP Annotations (Evaluated by ABAP runtime)	Framework-specific (Evaluated by Framework like OData, VDM, UI, Analytics, ...) **

View Annotations	`@AbapCatalog.sqlViewName`
	`@ClientDependent`
	`@EndUserText.label`	

Element Annotations	`@Semantics.unitOfMeasure` `@EndUserText.label`	
Parameter Annotations *	`@Environment.systemField` `@EndUserText.label`	
Extension Annotations *	`@AbapCatalog.sqlViewAppendName`	
Function Annotations *	`@ClientDependent` `@EndUserText.label`	

* See details in the respective lessons

** See the short outlook on those in the last unit

Figure 49: Important ABAP View Annotations for CDS DDIC-based views

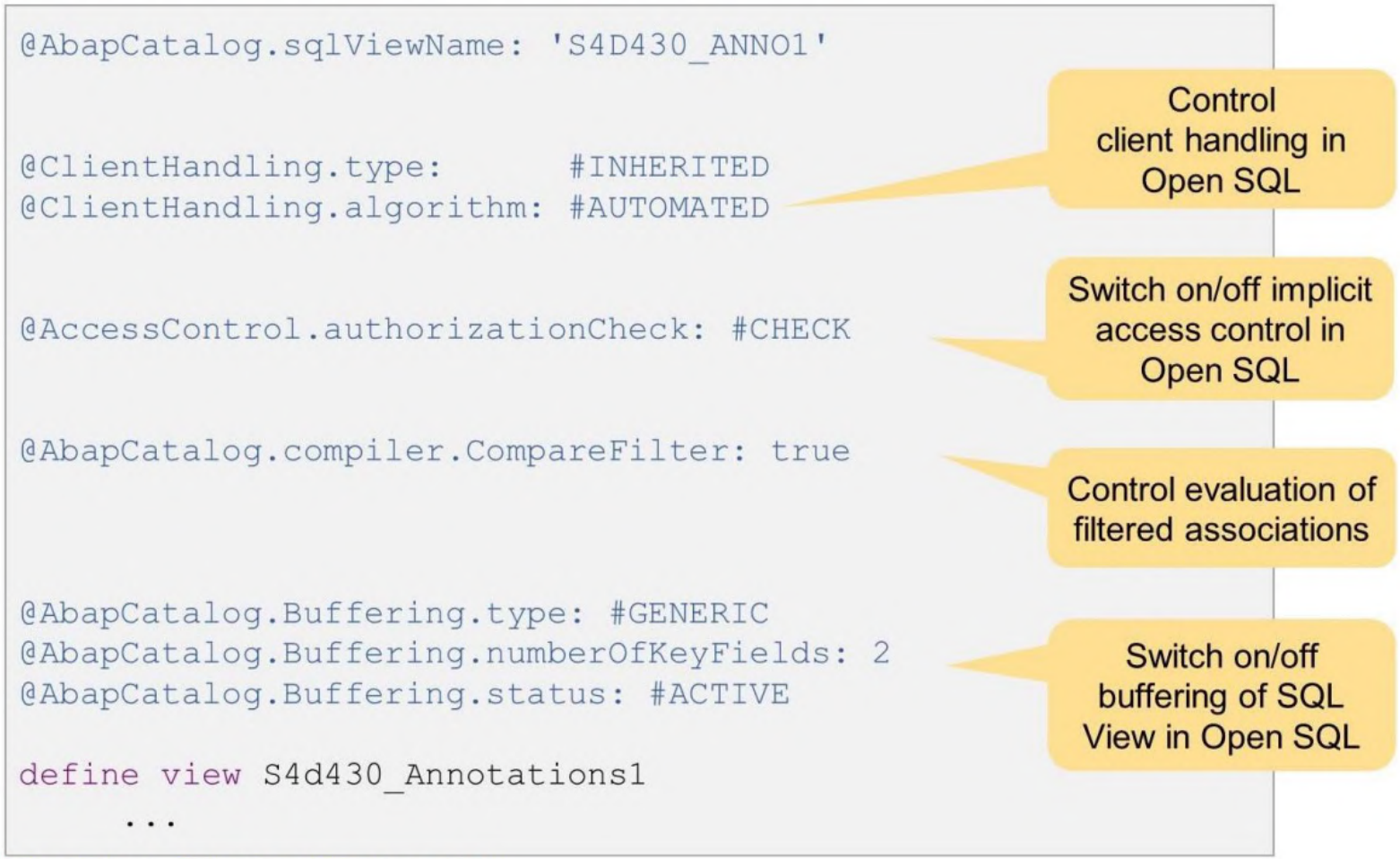


Figure 50: Important ABAP Annotations for View Elements

Semantics for Amount / Currency Code

```
...
@Semantics.amount.currencyCode:'currency'
 price,

@Semantics.currencyCode: true
 currency,
...
```

currency code for field *price* can be found in field *currency*

Semantics for Quantity / UnitOfMeasure

```
...
@Semantics.quantity.unitOfMeasure: 'DistanceID'
distance,

@Semantics.unitOfMeasure: true
distid AS DistanceID,
 ...
```

Unit for field *distance* can be found in field *DistanceID*

Figure 51: Element Annotations after Element

Element Annotation before the Element

```
...
@Semantics.amount.currencyCode:'CURRENCY'
  price,
@Semantics.currencyCode: true
 currency,
...
```

Element annotation after the Element

```
...
price      @<Semantics.amount.currencyCode:'CURRENCY',
currency   @<Semantics.currencyCode: true ,
...
```

Requires „<“ after „@“

Annotation before comma

Figure 52: Grouping of Annotations

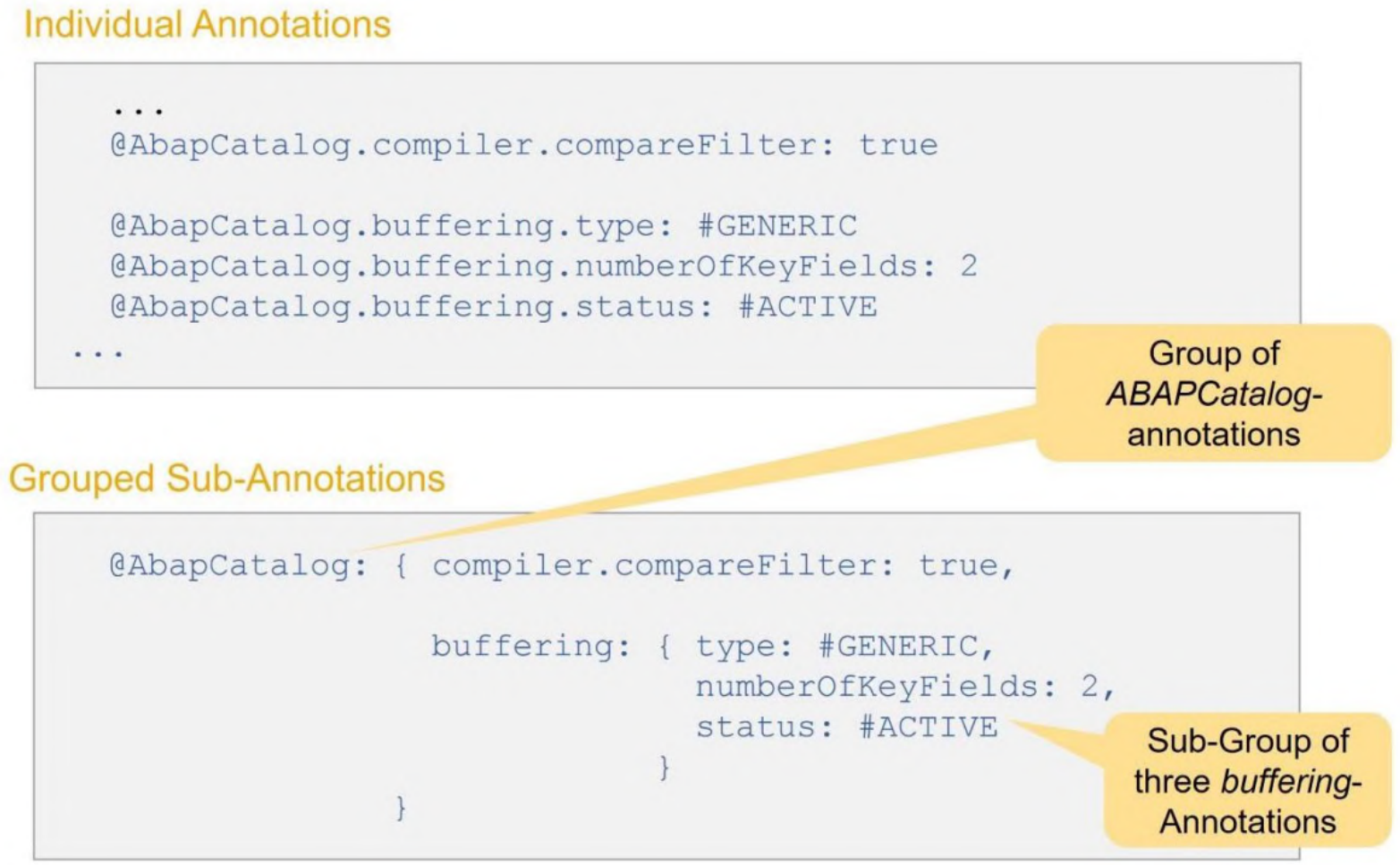


Figure 85: Select List – Support for Expressions in CDS Views

String & arithmetic expressions

- Literals
- Operators: +, - , *, /, unary –
- Built-in functions, e.g. CONCAT

CAST expression

- Length determined at activation time
- No nesting of CAST expressions

CASE expression

Special built-in functions

- Currency and unit conversion
- COALESCE

Alias required for result columns

```
define view HA400D_05_Expressions as
select from sbook {
CONCAT( carrid, connid) as flightno,

fldate, bookid,

CONCAT( CONCAT(
  SUBSTRING(order_date,5,2), '-') ),
  SUBSTRING(order_date,1,4)) as ordermonth,

case smoker
  when 'X' then
        cast(loccuram as abap.fltp)*1.03
  else  cast(loccuram as abap.fltp)*0.98
end as adjusted_amount,

currency_conversion(
   amount             => loccuram,
   source_currency => loccurkey,
   target_currency =>
                cast('EUR' as abap.cuky(5)),
   exchange_rate_date => order_date
  ) as euro_amount   }
```

Figure 57: Literals

Character Literals

```
...
'Hello' as col_char,  //Type CHAR
'32768' as col_numc,  //Type NUMC (only digits)
...
```

Numeric Literals

```
...
32768   as col_int4, //Type INT4
4711    as col_int2, //Type INT2 in Range [-32768,32767]
255     as col_int1, //Type INT1 in Range [0,255]
1.5     as col_fltp, //Type FLTP
...
```

Figure 58: Case Distinctions

Simple Case Distinction

```
CASE operand
        WHEN operand1 THEN result1
       [WHEN operand2 THEN result2]
        ...
       [ELSE resultn]
 END
```

- Comparable to ABAP Statement CASE … WHEN … ENDCASE.
- Result depends on a series of “EQUALS”-comparisons

Complex Case Distinction (= Searched Case)

```
CASE WHEN sql_condition1 THEN result1
    [WHEN sql_condition2 THEN result2]
         ...
    [ELSE resultn]
 END
```

- Comparable to ABAP Statement IF … ELSEIF …. ENDIF.
- Result depends on a sequence of SQL conditions (logical expressions)

Figure 59: Example 1: A Simple Case Expression

```
@AbapCatalog.sqlViewName: 'S4D430_EXPRCASE'
define view S4d430_Expression_Case as select
  from sbook
  {

 ...

// Simple Case

      case class
         when 'Y' then 'Economy'
         when 'C' then 'Business'
         when 'F' then 'First'
       end                                as class_txt
...

  }
```

Figure 60: Example 2: A Complex Case Expression

```
@AbapCatalog.sqlViewName: 'S4D430_EXPRCASE'
define view S4d430_Expression_Case as select
  from sbook
  {

...

// Complex Case
       case
        when class = 'F' then ' '
        when wunit = 'KG' and luggweight > 20 then 'X'
        when wunit = 'LB' and luggweight > 44 then 'X'
        else ' '
       end                                          as excess_luggage1,

...

  }
```

Figure 61: Example 3: Two Nested Case Expressions

```
@AbapCatalog.sqlViewName: 'S4D430_EXPRCASE'
define view S4d430_Expression_Case as select
  from sbook
  {

...

// Nested Case
       case class
         when 'F' then ' '
         else case
                when ( wunit = 'KG' and luggweight > 20 )
                  or ( wunit = 'LB' and luggweight > 44 )
                then 'X'
                else ' '
              end
        end                                          as excess_luggage2
  }
```

Figure 63: Example: Arithmetic Expressions in CDS Views

```
@AbapCatalog.sqlViewName: 'S4D430_EXPRARITH'
define view s4d430_Expression_arithmetic as select
  from sflight
  {
    seatsmax,
    seatsocc,

    seatsmax - seatsocc                as seatsfree,
    seatsocc + seatsocc_b + seatsocc_f as seatsmax_tot,
    2 * price                          as double_price

}
```

Figure 64: Type Conversion with CAST Expression

- **What it looks like:**

```
CAST ( operand AS target_type [PRESERVING TYPE] )
```

- **What it does:**
  - Converts the value of *operand* into *target_type*
- **Many Options for *operand*:**
  - Literal (without a domain prefix)
  - Field of a data source
  - Arithmetic expression
  - Case distinction with CASE
  - Predefined function
  - …
- **Two Options for *target_type*:**
  - A predefined dictionary type, e.g. *abap.int4, abap.char(10), abap.dec(8,2)*
  - Any Dictionary data element, e.g. *S_CARRID* , *BUKRS*
  - Addition PRESERVING TYPE to change semantic attributes, only

Figure 65: Example: Type Conversions With CAST

```
@AbapCatalog.sqlViewName: 'S4D430_EXPRCAST'
define view S4d430_Expression_Cast as select
  from sflight
  {
    '19891109'                                          as col_char,

    cast('19891109' as abap.int4)                       as col_int4,
    cast('19891109' as abap.dec(16,2))                  as col_dec,
    cast('19891109' as abap.fltp)                       as col_fltp,

    cast('19891109' as abap.dats)                       as col_dats,
    cast('19891109' as s_date)                          as col_ddic,
    cast('19891109' as s_customer preserving type)      as col_cust,

    cast(seatsocc as abap.fltp) / cast(seatsmax as abap.fltp)
                                                        as ratio
 }
```

col_char	col_int4	col_dec	col_fltp	col_dats	col_ddic	ratio
19891109	19.891.109	19891109.00	1.9891109000000000E+07	1989-11-09	1989-11-09	9.7402597402597402E-01
19891109	19.891.109	19891109.00	1.9891109000000000E+07	1989-11-09	1989-11-09	9.6623376623376622E-01

Figure 88: CDS Views in ABAP Programming

CDS views in ABAP Programming

- CDS view name as ABAP data type
- CDS view name in the FROM clause of Open SQL SELECT statements
- New Open SQL Syntax mandatory
- CDS views not allowed in INSERT, UPDATE, MODIFY, DELETE

```
DATA lt_book TYPE STANDARD TABLE
                  OF HA400D_02_Field_List.

SELECT *
  FROM HA400D_02_Field_List
  INTO TABLE @lt_book
 WHERE customer = '931'.
```

Figure 86: Support for Joins, WHERE, Aggregations, Grouping, and Filtering in CDS Views

Like in Open SQL:

JOINs supported

- **INNER**, **LEFT OUTER**, **RIGHT OUTER** join
- Before or after the field list
- No SELECT * together with joins

Aggregations supported

WHERE clause supported

Grouping, Filtering supported

- All columns not aggregated must be listed in **GROUP BY**
- **HAVING** clause filters resulting groups, aggregate functions supported

```
define view HA400D_06_Complex as
  select from sbook as book
    inner join scustom as customer
            on book.bookid = customer.id
{
    book.carrid,
    book.loccurkey as currcode,

    sum(book.loccuram) as total_amount,
    max(book.loccuram) as max_amount,
    min(book.loccuram) as min_amount,

    customer.region,
    customer.discount
}
  where     customer.country = 'US'

  group by book.carrid, book.loccurkey,
           customer.region,
           customer.discount
  having sum(book.loccuram) > 100
```

Figure 84: Reminder: Inner Join

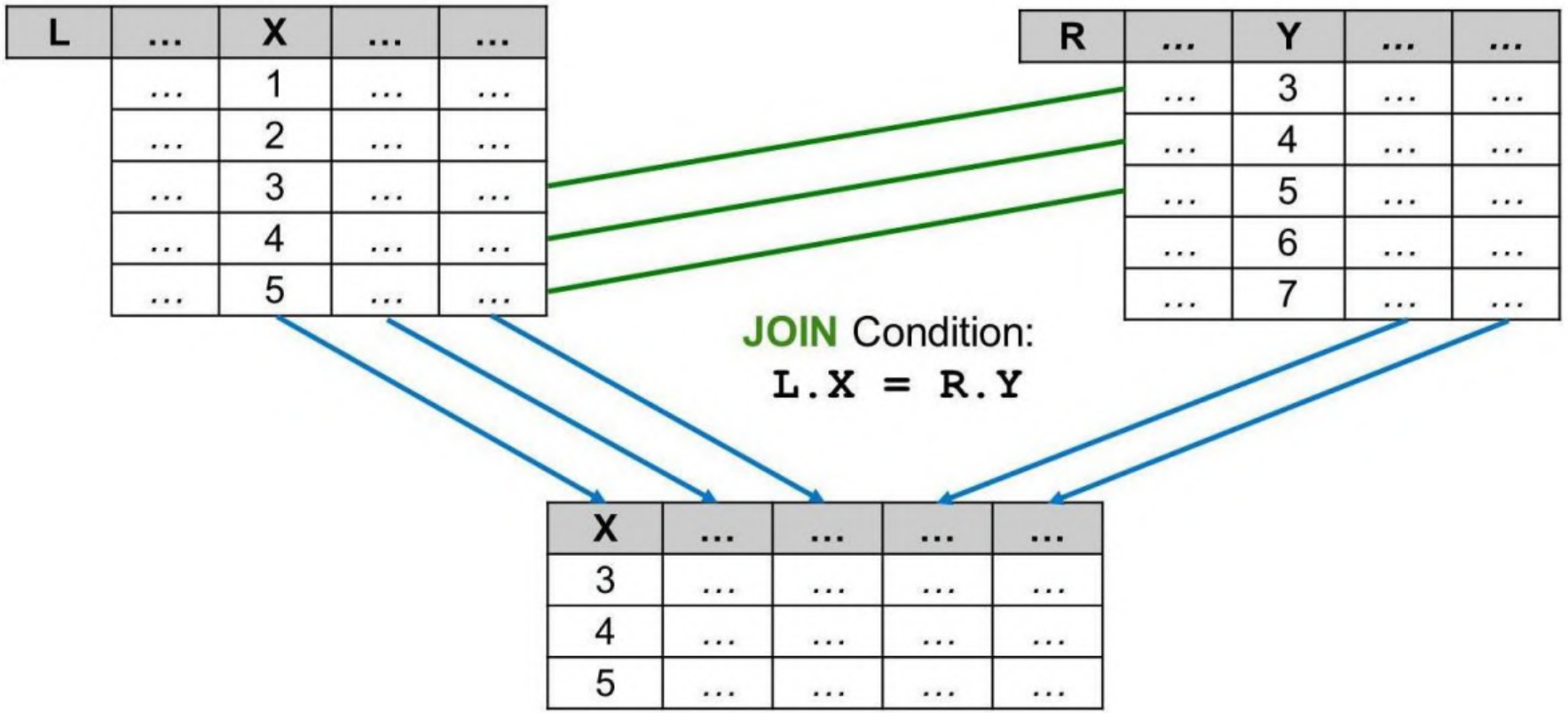


Figure 44: Inner Join in DDL View Definition

Figure 85: Types of Outer Joins

- **Left Outer Join**
  - Special treatment for left table
  - Result set may contain entries of **left** table without matching entries in right table
  - Supported in classical Open SQL
- **Right Outer Join**
  - Special treatment for right table
  - Result set may contain entries of **right** table without matching entries in left table
  - Supported in Open SQL (as off NW 7.40 SP05) and ABAP CDS
- **Full Outer Join**
  - Special treatment for both tables
  - Result set may contain entries of **left** table without matching entries in right table
  - Result set may contain entries of **right** table without matching entries in left table
  - Not yet supported in Open SQL and ABAP CDS

Figure 86: Left Outer Join

Figure 87: Right Outer Join

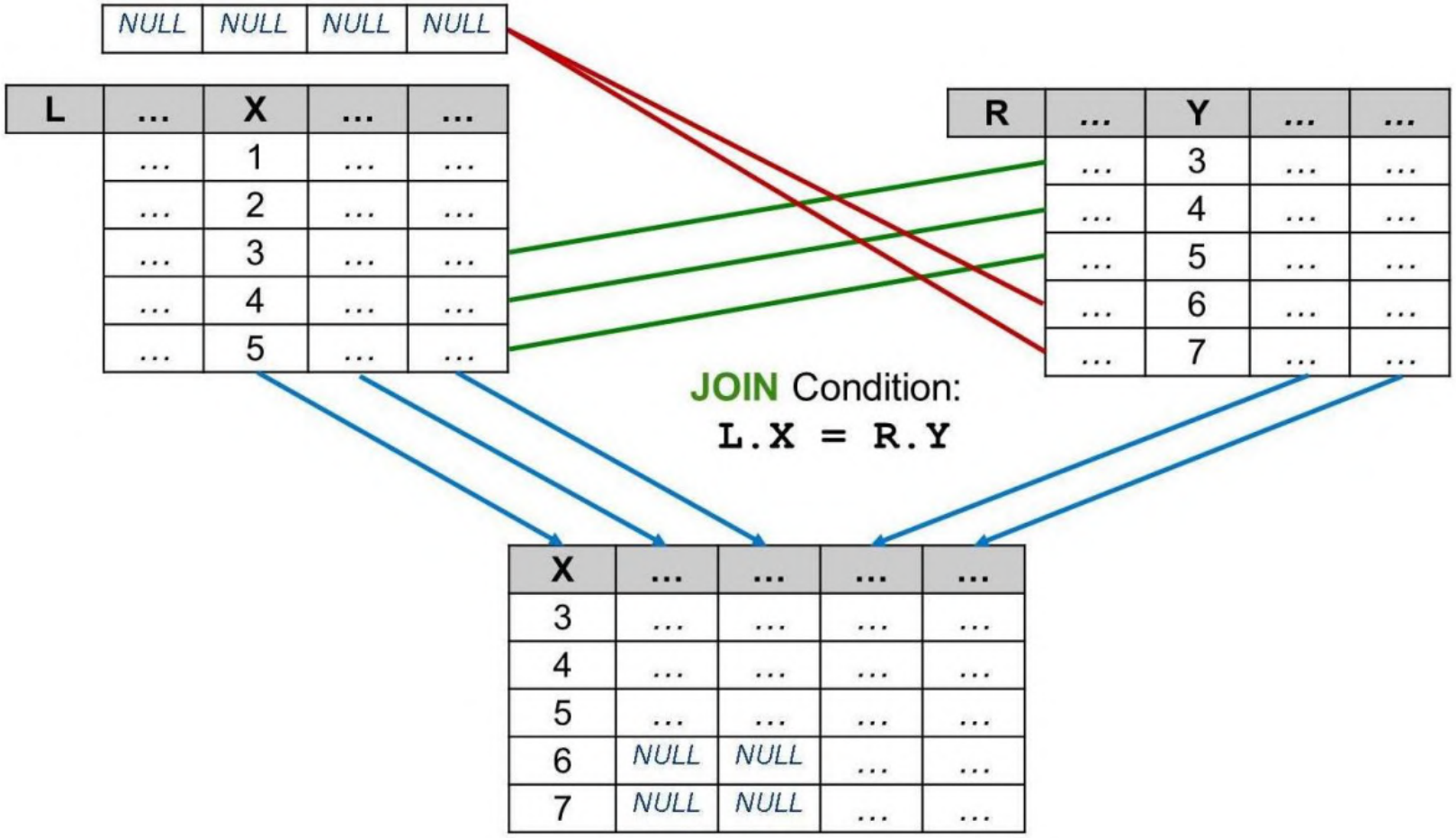


Figure 89: Example: Right Outer Join

```
@AbapCatalog.sqlViewName: 'S4D430_JOINRIGHT'
define view S4d430_Join_Right_Outer as select
       from  spfli as c right outer join scarr as a
         on c.carrid = a.carrid
      {
        key a.carrid,
            a.carrname,
            c.connid,
            c.cityfrom,
            c.cityto
}
```

Raw Data

Filter pattern | 36 rows retrieved - 62 ms

carrid	carrname	connid	cityfrom	cityto
AA	American Airlines	0017	NEW YORK	SAN FRANCISCO
AA	American Airlines	0064	SAN FRANCISCO	NEW YORK
AB	Air Berlin	0000		
AC	Air Canada	0000		
AF	Air France	0000		
AZ	Alitalia	0555	ROME	FRANKFURT

No flight connections in table SPFLI for Carriers “AB“, “AC“ and “AF“

Figure 94: Example: Nested Join Expression

```
@AbapCatalog.sqlViewName: 'S4D430_JOINNEST'
define view S4d430_Join_Nested as select
       from scarr as a
       left outer join (
                         sairport as p
                         left outer join scounter as c
                                      on p.id = c.airport
                       )
                    on a.carrid = c.carrid
     {
       a.carrid   as carrier_id,
       p.id       as airport_id,
       c.countnum as counter_number
     }
```

Join in parantheses is evaluated first

Figure 88: Full Outer Join (not supported in ABAP CDS)

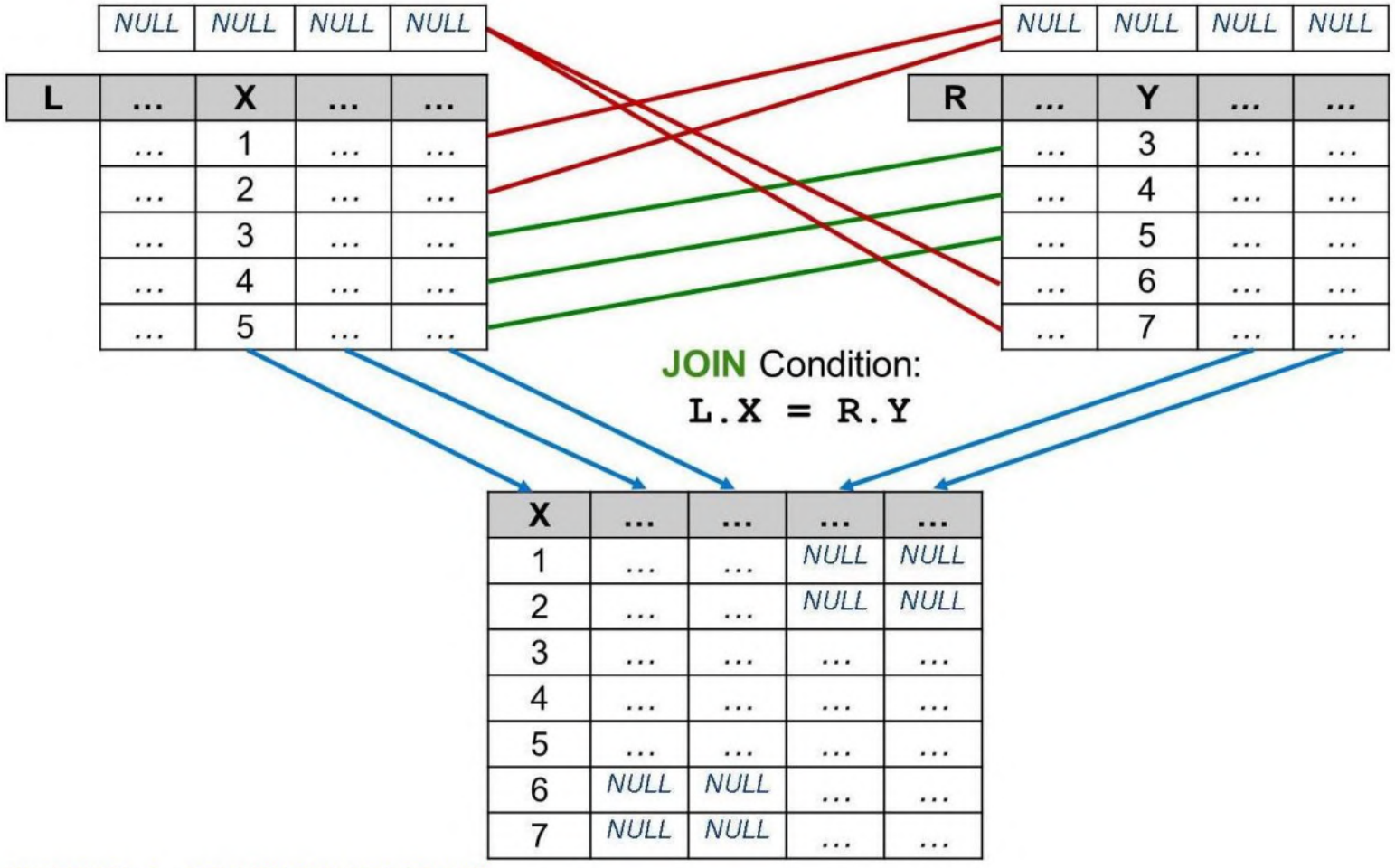


Figure 90: Cross Join (As off Release 7.51)

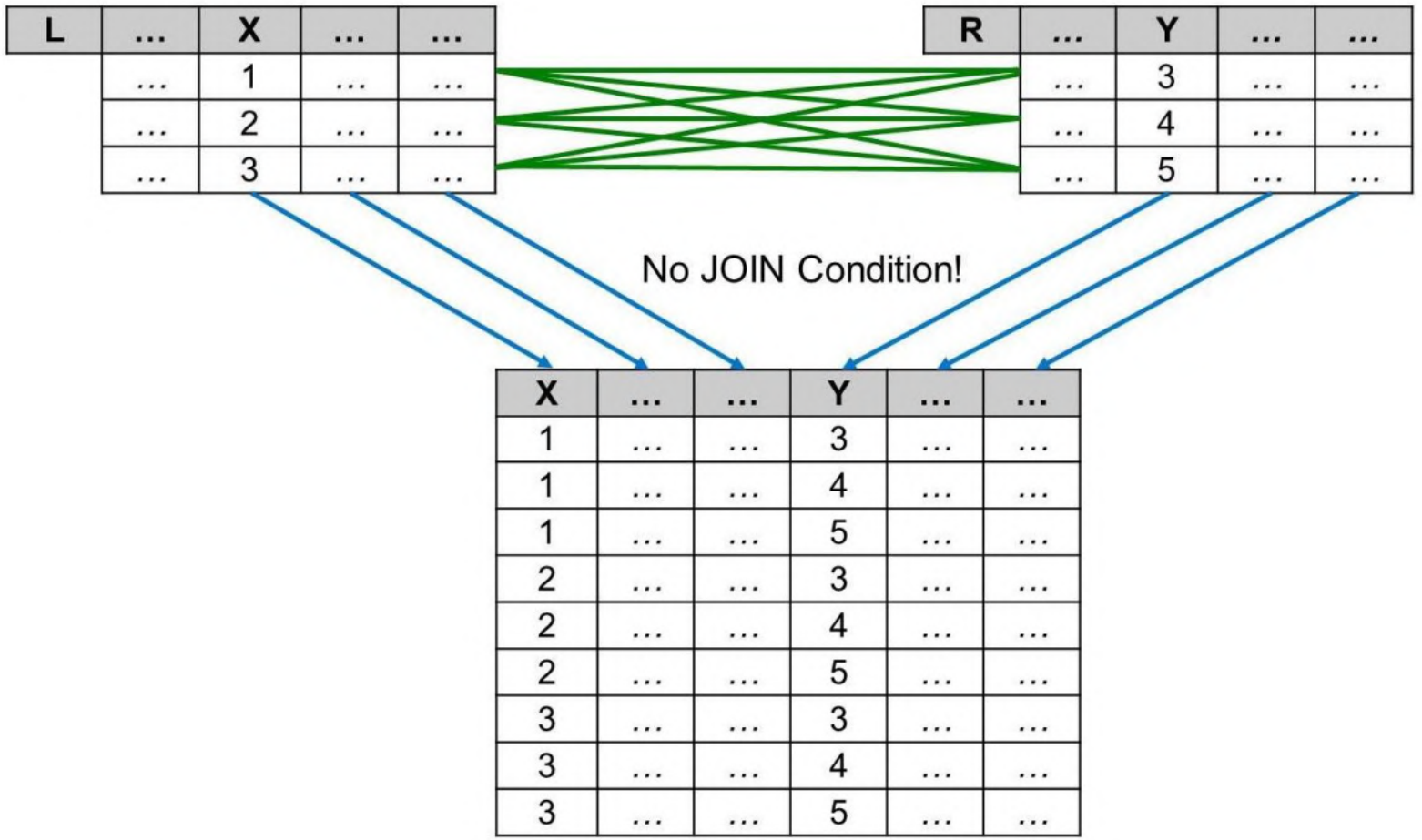


Figure 91: Example: Cross Join

```
@AbapCatalog.sqlViewName: 'S4D430_JOINCROSS'
define view S4D430_JOIN_CROSS as select
       from  spfli as c cross join scarr as a
      {
        key a.carrid as carrid_scarr,
            a.carrname,
            c.carrid as carrid_spfli,
            c.connid,
            c.cityfrom,
            c.cityto
}
```

S4D430_JOIN_CROSS

Raw Data

Filter pattern | 100 rows retrieved - 3 ms (partial result)

carrid_scarr	carrname	carrid_spfli	connid	cityfrom	cityto
AA	American Airlines	AZ	0555	ROME	FRANKFURT
AA	American Airlines	LH	2402	FRANKFURT	BERLIN
AA	American Airlines	UA	0941	FRANKFURT	SAN FRAN...
AA	American Airlines	AZ	0789	TOKYO	ROME
AA	American Airlines	LH	0402	FRANKFURT	NEW YORK
AA	American Airlines	QF	0005	SINGAPORE	FRANKFURT

Figure 96: UNION

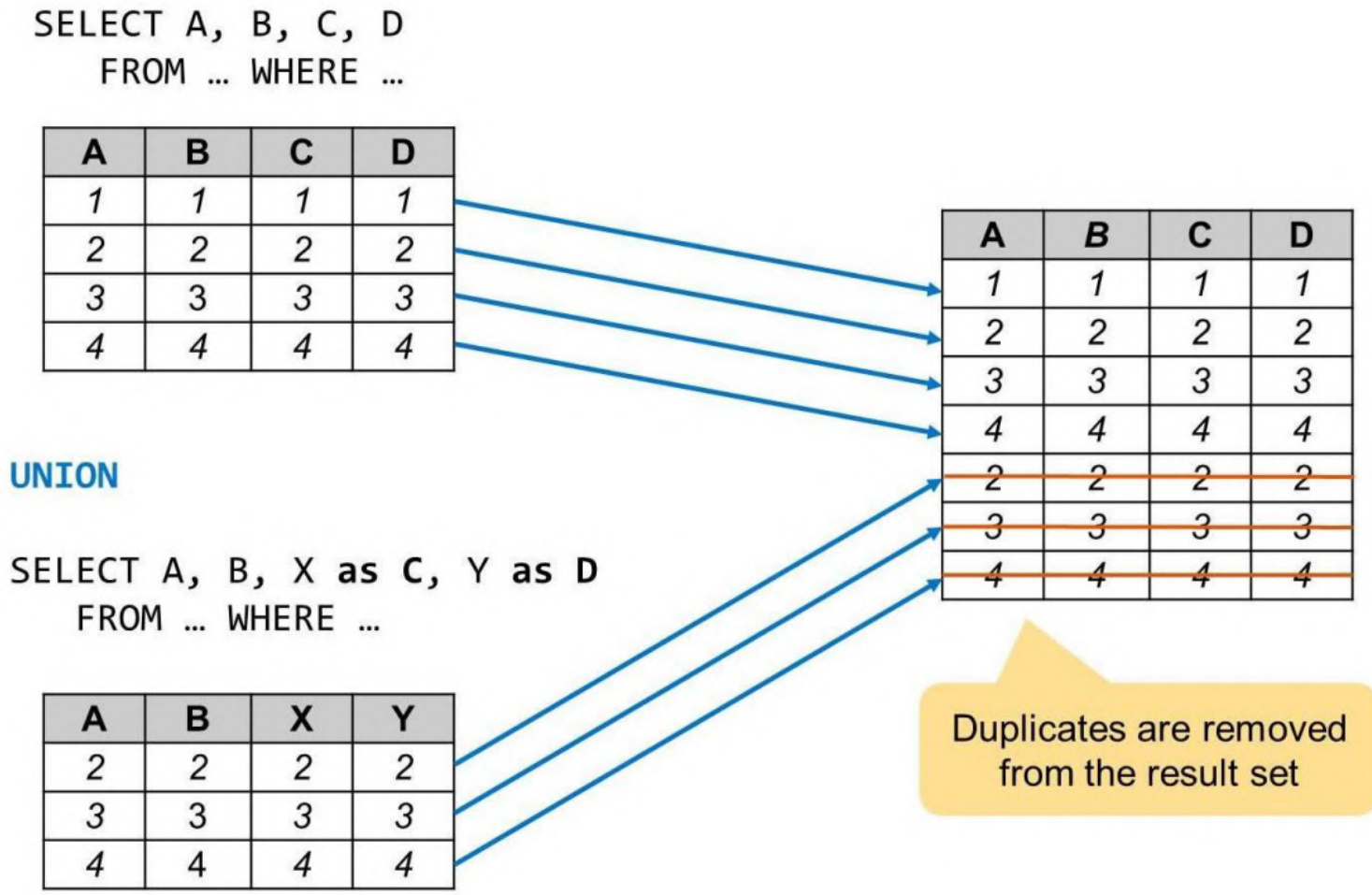


Figure 98: Example: UNION

```
@AbapCatalog.sqlViewName: 'S4D430_UNIO'
define view S4d430_Union as
select from scustom
          {
            key id,
            key 'Customer'       as type,
                name,
                city,
                country
           }
union
select from stravelag
          {
            key agencynum        as id,
                'Agency  '       as type,
                name,
                city,
                country
     }
```

Key definition in second select is ignored

Alias *id* is needed to have identical element names

Additional blanks needed to have compatible types

Figure 87: Unions

Merge the results of queries using keyword UNION

- Column lists of the queries must
  - have equal number of columns
  - Be of compatible types in matching order
- **UNION** implies a distinct semantic
- **UNION ALL** does not include distinct

```
define view HA400D_07_Union as
select from scustom
 {  id,                          }
    form, name,                  }
    street,                      }
    postcode, city,              } First query
    country,                     }
    postbox, custtype            }
} where postbox <> ' '
union
select from scustom
 {  id,                          }
    form, name,                  }
    street,                      }
    postcode, city,              } Second query
    country,                     }
    postbox, custtype            }
} where custtype = 'B'
```

Figure 99: UNION ALL

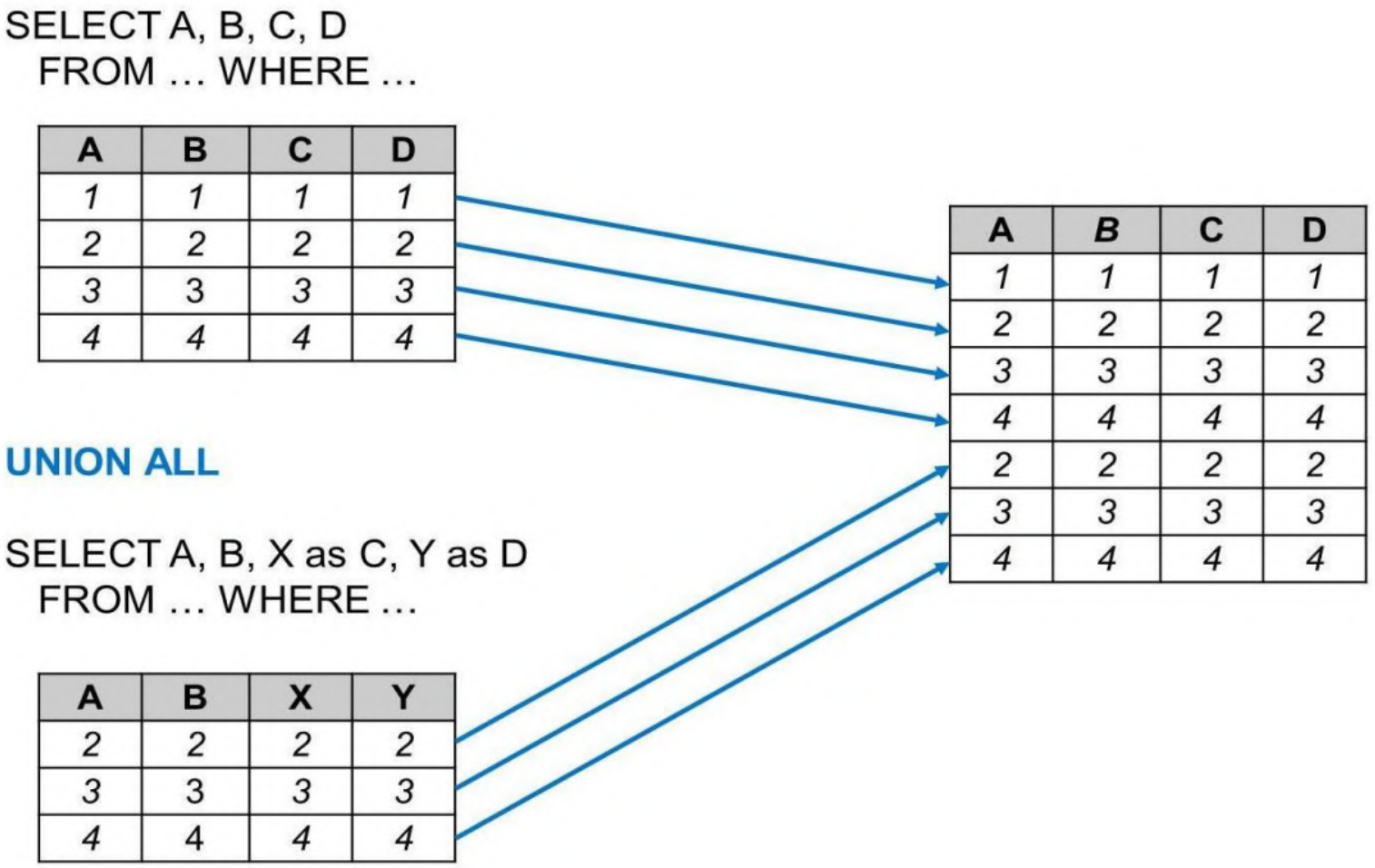
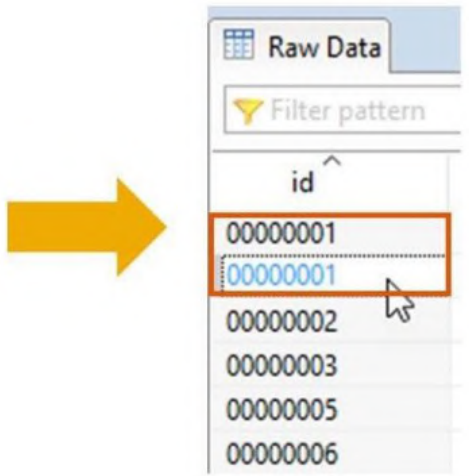


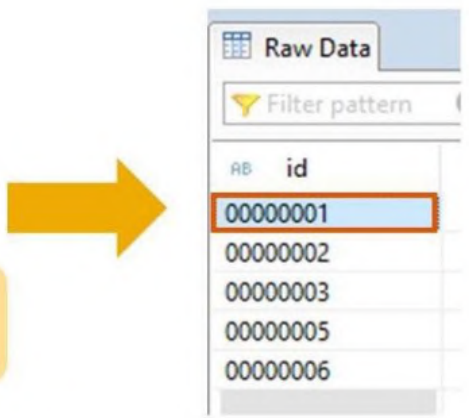
Figure 100: Example: UNION vs UNION ALL

```
@AbapCatalog.sqlViewName: 'S4D430_UNIOALL1'
define view S4d430_Union_All_1 as
select from scustom
          { key id }
      where city = 'Walldorf'
union all
select from sbook
          { key customid as id }
      where agencynum = '00000100'
```




```
@AbapCatalog.sqlViewName: 'S4D430_UNIOALL2'
define view S4d430_Union_All_2 as
select from scustom
          { key id }
      where city = 'Walldorf'
union
select from sbook
          { key customid as id }
      where agencynum = '00000100'
```

Same definition without ALL



Lesson Agenda

Learn About:

- Arithmetic Functions
- String Functions
- Currency and Unit Conversion
- Calculations with Dates

Do:

- Use Built-in Functions in a CDS View

: Numeric Functions (1)

Built-in Functions for Calculations

- **div(*arg1,arg2*)**
  - Input: Only integer values
    (INT1, INT2, INT4, INT8 or DEC, CURR, QUAN with decimals = 0)
  - Result type: type of *arg1*
  - Result always rounded off to next integer value
- **mod(*arg1,arg2*)**
  - Input: Only integer types (INT1, INT2, INT4, INT8)
  - Result type: type of *arg1*
  - Result can be negative
- **division(*arg1, arg2, dec*)**
  - Input: Integer values, values with fixed decimal
    (Types INT1, INT2, INT4, INT8 and DEC, CURR, QUAN with any number of decimals)
  - Result type: DEC with *dec* decimal places, length depends on type of *arg1*
  - Result is always rounded to *dec* decimals ((commercial rounding)

Numeric Functions (2)

Built-in Rounding Functions

- **abs(*arg*)**
  - returns the absolut value of *arg*

```
abs(  1.5) =  1.5
abs( -1.5) =  1.5
```

- **floor(arg)**
  - rounds to the next lower integer
    i.e. towards zero if arg > 0, away from zero if arg < 0

```
floor(  1.5) =   1
floor( -1.5) =  -2
```

- **ceil(*arg*)**
  - rounds to the next higher integer
    i.e. away from zero if arg > 0, towards zero if arg < 0

```
ceil(  1.5) =   2
ceil( -1.5) =  -1
```

- **round(*arg*,*pos*)**
  - pos > 0: Round *arg* to *pos* decimal places
  - *pos* < 0: Round *arg* to position

```
round(3.1514,  2) = 3.15
round(273.15,-1) =  270
```

String Processing Functions(1)

Some Built-in Functions for String Processing (NW 7.40)

- **concat(*arg1*,*arg2*)**
  - Returns result of type CHAR or SSTRING (depending on types of *arg1* and *arg2*)
  - All trailing blanks are removed
  - Corresponds to ABAP statement **`CONCATENATE`** without addition **`SEPARATED BY`**
- **replace(*arg1*, *arg2*, *arg3*)**
  - Result type depends on type of *arg1*
  - corresponds to ABAP statement

    ```
    REPLACE ALL OCCURENCES OF arg2 IN arg1 WITH arg3.
    ```
- **substring(*arg*,*pos*,*len*)**
  - Result type depends on type of *arg*
  - Similar to ABAP function substring( ) or direct substring access:

    ```
    dobj[+off][(len)]
    ```

    Important difference: *pos* denotes the position, not the offset!

Some Built-in Functions for String Processing (NW 7.50)

- **concat_with_space(*arg1,arg2,count*)**
  - Like concat( ) but with *count* spaces between *arg1* and *arg2*
  - Similar to ABAP statement **`CONCATENATE`** with addition **`SEPARATED BY ` `**
- **length(*arg*)**
  - Returns result of type INT4
  - Trailing blanks do not count
  - Corresponds to ABAP built-in function **`numofchar( )`**
- **left(*arg,n*) and right(*arg,n*)**
  - Similar to substring( ), but returns first or last *n* characters of *arg*
  - left( ) corresponds to ABAP expression **`arg(n)`**

Built-in Functions for String Processing (NW 7.51)

- **lower(*arg*)**
  - Result type identical to type of *arg*
  - NUMC, DATS and TIMS not allowed as input type
  - Corresponds to ABAP statement **`TRANSLATE arg TO LOWER CASE`**
- **upper(*arg*)**
  - Result type identical to type of *arg*
  - NUMC, DATS and TIMS not allowed as input type
  - Corresponds to ABAP statement **`TRANSLATE arg TO UPPER CASE`**

Example: String Processing in a CDS View

```
@AbapCatalog.sqlViewName: 'S4D430_FUNCCHAR,
define view s4d430_Function_string as select
  from scarr
  {
    carrid,                                  // => 'LH '
    carrname,                                // => 'Lufthansa'
    concat(carrid,carrname)                  // => 'LHLufthansa'
        as col_concat,
    concat_with_space(carrid,carrname,3)     // => 'LH   Lufthansa'
        as col_concat_space,
    replace(carrname,'hans','fritz')         // => 'Luftfritza'
        as col_replace,
    substring(carrname,5,4)                  // => 'hans'
        as col_substring,
    length(carrid) as col_length             // => 2 (trailing blank!)
   }
   where carrname = 'Lufthansa'
```

Currency and Unit Conversions

Built-in functions

- `Unit_Conversion(p1 => a1, p2 => a2, ... )`
  - Returns result of type *abap.quan*
  - Converts a quantity in source unit into a value in target unit
  - Rules maintained in transaction CUNI and stored in database table T006
- `Currency_Conversion(p1 => a1, p2 => a2, ... )`
  - Returns result of type *abap.curr*
  - Converts an amount in source currency into a value in target currency
  - Based on the exchange rate valid on a target date
  - Rules maintained in transaction OB08 and stored in database tables TCUR... .

General Remarks

- Parameter assignment with operator “=>”
- Comma-separated parameters
- Optional parameter *error_handling* with default value "FAIL_ON_ERROR“
  Other options: "SET_TO_NULL“ and "KEEP_UNCONVERTED“
- Result may depend on the database (different rounding rules)

Example: Unit Conversion From Kilometers to Miles

```
@AbapCatalog.sqlViewName: 'S4D430_FUNCCONV1'
define view S4d430_Function_Convertion1 as select
  from spfli
     { carrid,
       connid,
       @Semantics.quantity.unitOfMeasure: 'DISTID'
       Unit_Conversion( quantity    => distance,
                        source_unit => distid,
                        target_unit => cast('MI' as abap.unit)
                      ) as distance,
       @Semantics.unitOfMeasure: true
       cast('MI' as abap.unit) as distid
     }
```

Casting of literal 'MI' into expected type

Parameter assignment with operator „=>“

Example: Currency Conversion

```
@AbapCatalog.sqlViewName: 'S4D430_FUNCCONV2'
define view s4d430_Function_Conversion2 as select
  from sflight
     { carrid,
       connid,
       fldate,
       @Semantics.amount.currencyCode:'CURRENCY'
       currency_conversion(
             amount              => price,
             source_currency     => currency,
             round               => 'X',
             target_currency     => cast( 'USD' as abap.cuky),
             exchange_rate_type => 'M',
             exchange_rate_date => fldate,
             error_handling      => 'SET_TO_NULL'
             ) as price,
       @Semantics.currencyCode: true
       cast('USD' as abap.cuky) as currency
      }
```

Casting of literal 'USD' into expected type

Behaviour in Case of error

Calculations with Dates

Built-in functions, new in NW 7.50

- **dats_is_valid(*date*)**
  - Returns result of type INT4
  - Returns 1 if *date* contains a valid date, 0 otherwise
- **dats_days_between(*date1*,*date2*)**
  - Returns result of type INT4
  - Calculates number of days between two dates (corresponds to *date2* – *date1* in ABAP)
- **dats_add_days(*date*,*count*,*on_error*)**
  - Returns result of type DATS
  - Adds count days to the given date (corresponds to *date* + *count* in ABAP)
- **dats_add_months(*date*,*count*,*on_error*)**
  - Returns result of type DATS
  - Adds *count* months to the given date (no simple equivalent in ABAP)

General Remarks:

- All dates in format YYYYMMDD (technical format on database)
- Possible values for *on_error*: 'FAIL', 'NULL', 'INITIAL', 'UNCHANGED'

Example: Calculations with Dates

```
 @AbapCatalog.sqlViewName: ,S4D430_FUNCDAYS'
define view S4D430_Function_Days as select
       from sbook
          {
            carrid,
            connid,
            fldate,
            bookid,

            dats_days_between(order_date, fldate) as days_ahead,

            dats_add_days( order_date, 14, 'FAIL' ) as due_date

          }
```

„Subtract“ two fields of type DATE

What happens in case of an error? (Here: Raise exception)

Lesson Agenda

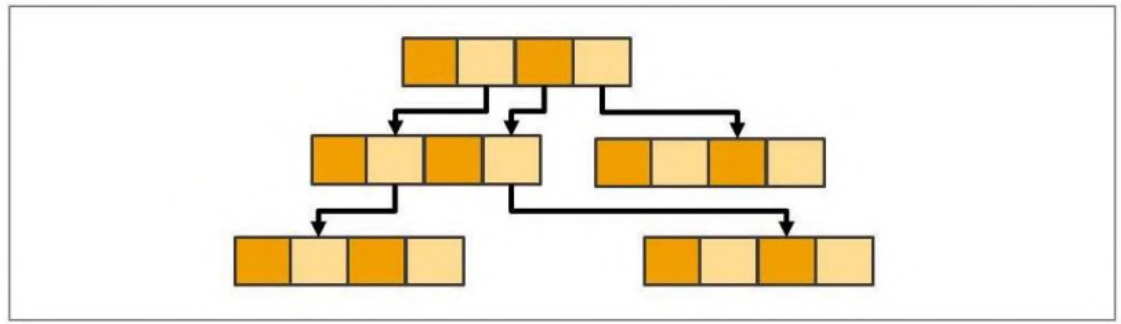
Learn About:

- CDS Views Based on Other CDS Views
- Advantages and Pitfalls
- Recommendations

Do:

- Define a CDS View based on other CDS Views

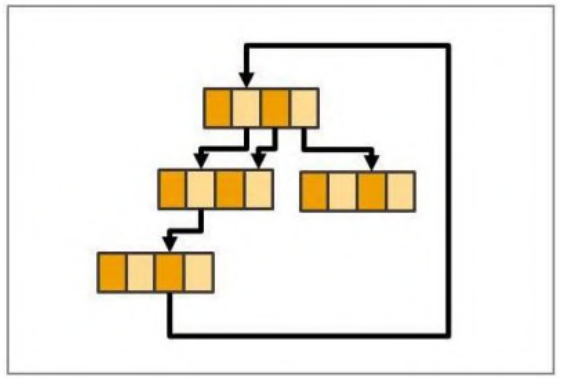
Nested Views (View on View Concept)



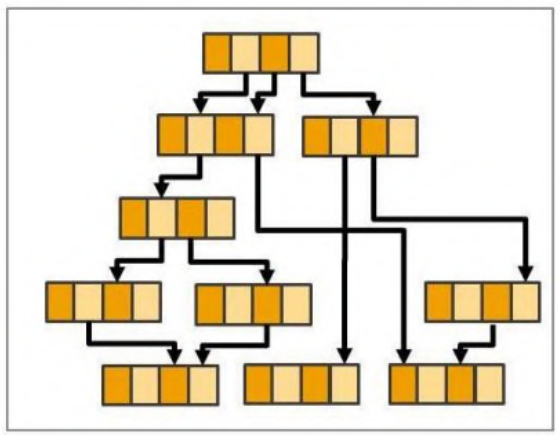
- **CDS Views as data source of other CDS Views**
  - Join of CDS Views and join of CDS View with DB table supported
  - No technical limit of nesting depth and complexity
- **Improved Re-use**
  - e.g. calculation done in one basis view that is used in several other views instead of coding (and maintaining) identical expressions
- **Improved Readability**
  - e.g. join of 2 views that contain joins instead of a complicated join of many tables

Pitfalls of Nested Views

- **Unintended Circular References**
  - Eventually, all views have to be based on tables
  - Circular references are not allowed
  - At the moment, circular references lead to timeouts during activation



- **Too many Nesting Levels**
  - increase the likelihood for recursive references
  - deteriorate the readability of the model
  - increase the likelihood for redundancies



Recommendations for Nested Views

- Group your views into levels (preferably 3 levels)
- Only access views that are assigned to a lower level

Lesson Agenda

Learn About:

- Aggregate Expressions
- Restrictions in ABAP CDS

Do:

- Define a CDS View with Aggregation

Aggregate Functions

- **MIN(*operand*) and MAX(*operand*)**
  - Returns the smallest/greatest value in *operand*
- **SUM(*operand*)**
  - Calculates the sum of the values of *operand*
- **AVG(*operand*)**
  - Calculates the average value of the values of *operand*
- **COUNT(*)**
  - Returns the number of entries in the result set
- **COUNT(DISTINCT *operand*)**
  - Returns the number of distinct values of *operand*

Operand can be
- a field of the data source
- a literal
- a case distinction

Example: Aggregate Functions

```
@AbapCatalog.sqlViewName: 'S4D430AGGREGATE'
define view S4d430_Aggregates as select
     from sflight
  {
   min( seatsocc )                      as col_min,
   max( seatsocc )                      as col_max,
   sum( seatsocc )                      as col_sum,

   avg( seatsocc )                      as col_avg,
   avg( seatsocc as abap.dec(16,2) )    as col_avg_conv,

   count(*)                             as col_count,
   count(distinct planetype)            as col_cnt_dist,

   cast( sum( 1 ) as abap.int4 )        as col_literal,
   sum(
     case
       when seatsocc > seatsmax
       then cast( 1 as abap.int4 )
       else 0
     end )                              as col_overbooked
}
```

Alias name mandatory for aggregate expressions

Result type of avg( ) is FLTP – change it with addition as

Sum of literal 1 (Same result as count(*))

Cast required to avoid overflow

Sum of a case distinction (Count overbooked flights)

Example: Aggregate Functions with GROUP BY

```
@AbapCatalog.sqlViewName: 'S4D430_AGGRGRP1'
define view S4D430_Aggregates_Group_By_1 as select
  from sflight
  {
    carrid,
    connid,
    count(*)                    as col_count,
    avg( seatsocc )             as col_avg,
}
group by carrid, connid
```

Field list consists not only of aggregate expressions

GROUP BY needed for all fields in the field list

```
@AbapCatalog.sqlViewName: 'S4D430_AGGRGRP2'
define view S4D430_Aggregates_Group_By_2 as select
  from sflight
  {
    concat_with_space(carrid, connid, 1) as ID,
    count(*)                             as col_count,
    avg( seatsocc )                      as col_avg,
}
group by carrid, connid
```

Field list contains expression or function

Arguments of the expression or function listed individually

Important Restrictions for Aggregate Functions in CDS

- **Argument of aggregate function**
  - only fields, literals, case distinctions
  - no other expressions or functions (e.g. avg (seatsmax – seatsocc))
  - use „View on View“ to aggregate calculated results
- **Argument of function SUM**
  - has to be numeric
  - INT8, DFL16, DFL34 are not supported
- **NULL values**
  - Are not considered in the aggregation
- **CDS View Extensions (see later)**
  - CDS Views with aggregates cannot be extended before Rel. 7.51

Lesson Objectives

After completing this lesson, you will be able to:

- Define CDS Views with Input Parameters
- Provide Values for Input Parameters of a CDS View
- Specify a Default Value for an Input Parameter

Lesson Agenda

Learn About:

- Input Parameter Declaration
- Template Define View With Parameters
- Access to Input Parameters
- Input Parameters in Data Preview
- Input Parameters in CDS DDL
- Input Parameters in ABAP SQL
- Default Value for Parameters in ABAP SQL
- Session Variables

Examples for the Need of Input Parameters

- **Input for SQL expressions and functions, e.g.**
  - a discount factor when calculating a price
  - the separation character when concatenating text
  - the target currency for a currency conversion
- **Selection of Data to Enter an Aggregation, e.g.**
  - date up to which revenue shall enter a summation
- **Mandatory Selection Criteria, e.g.**
  - language key
  - user name

Example: View Definition with Input Parameters

```
@AbapCatalog.sqlViewName: 'S4D430_PARAM'
define view S4d430_Parameter
  with parameters
         parameter1: abap.char(10),

         parameter2: s_carr_id,

        @EndUserText.label: 'Discount'
        @EndUserText.quickInfo: 'Discount Factor for Price '
         factor:     abap.dec(3, 2),

         separator:  abap.char(1)
        @<EndUserText.label: 'Separation Character'

as select
     from sflight
        {
         ...
        }
```

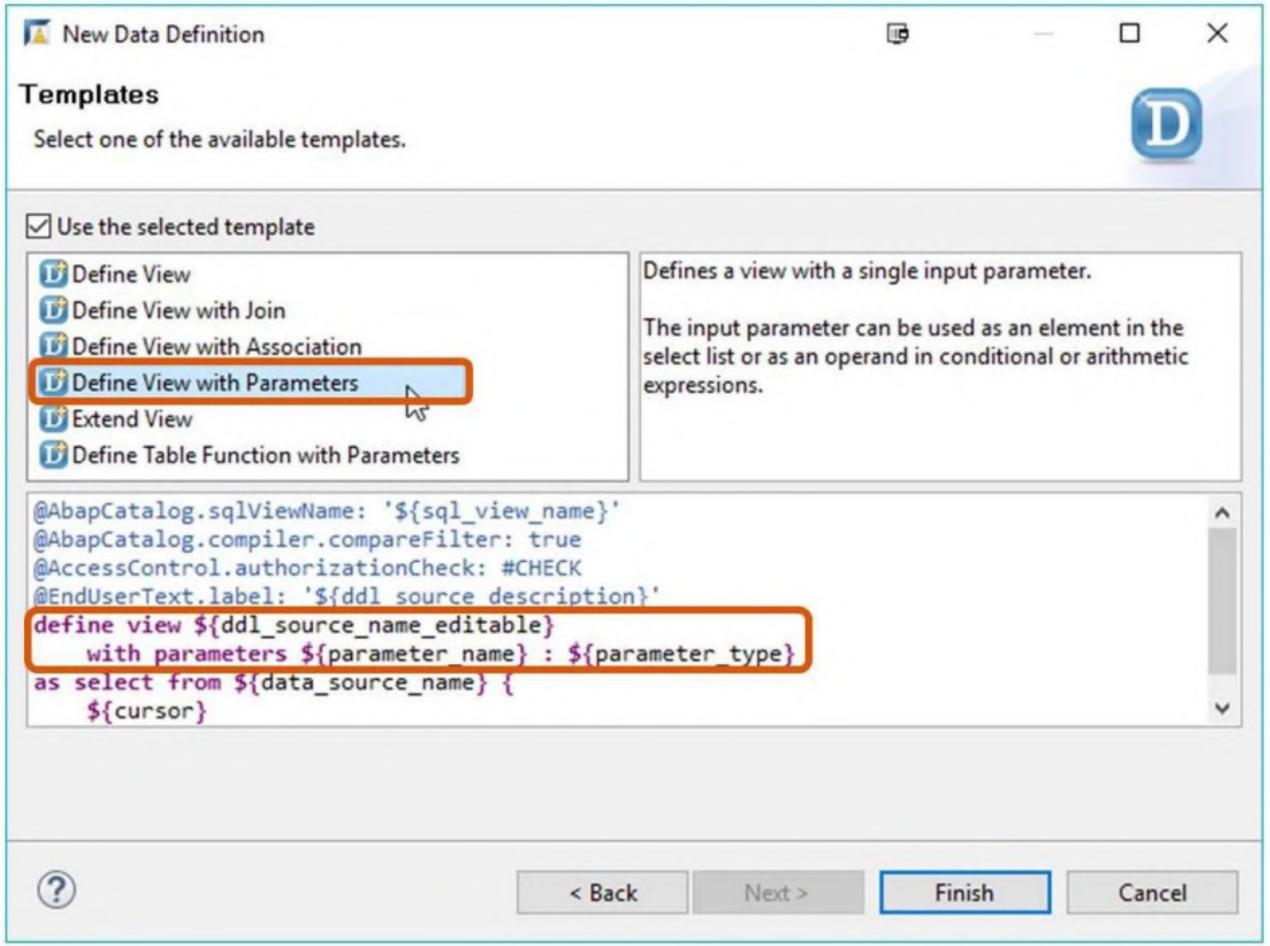
Addition
WITH PARAMETERS

Predefined Dictionary
type or data element

EndUserText-Annotations
before the parameter ..

… or after the parameter

Template Define View With Parameters



Example: How to Access Input Parameters

```
@AbapCatalog.sqlViewName: 'S4D430_PARAM'
define view S4d430_Parameter
  with parameters
            ...
            parameter1: abap.char(10),
            parameter2: s_carr_id

as select
     from sflight
        { ...
          $parameters.parameter1 as param1,

          :parameter2            as param2,
}
```

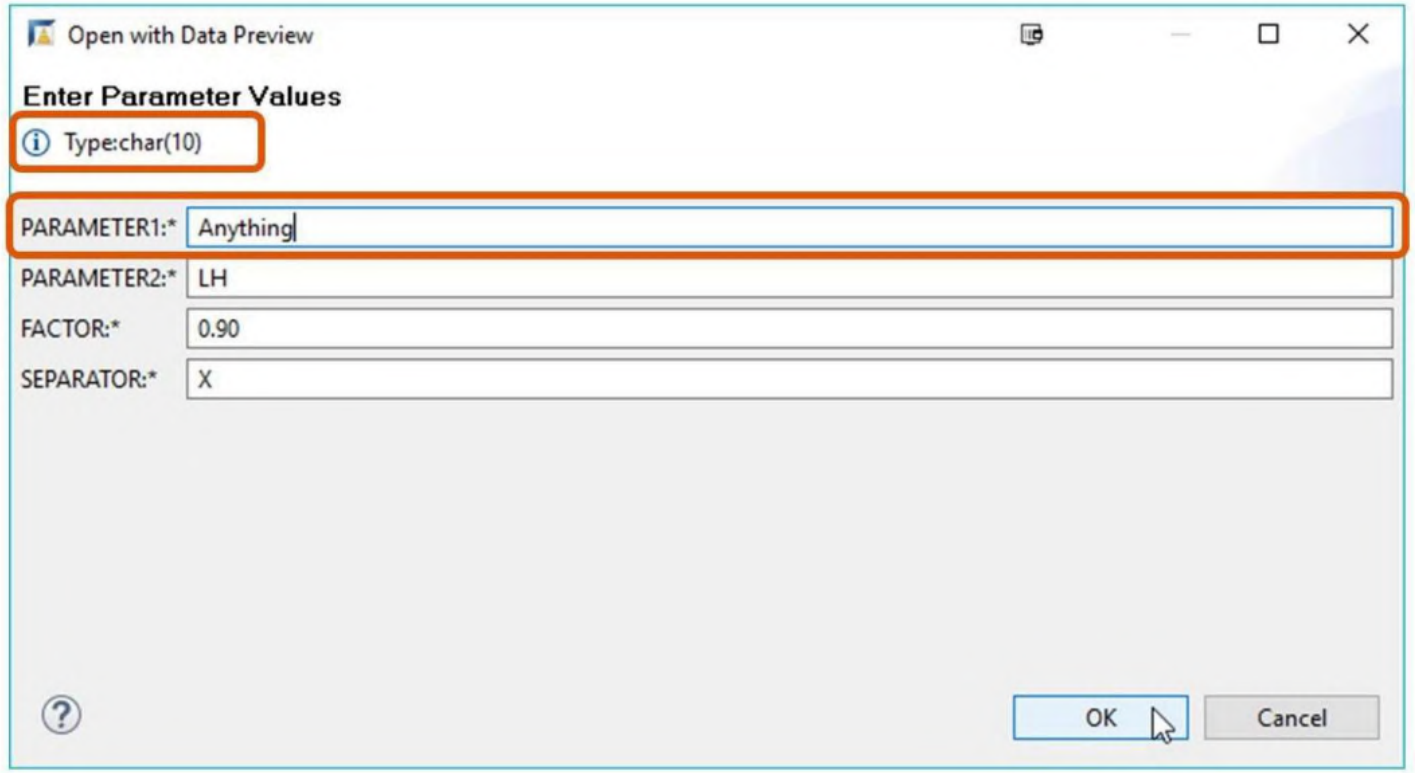
Prefixed with „*$parameters.*“ (recommended)

Prefixed with „:“ (not recommended)

Where to Access Input Parameters

- **In the Element List**
  - as standalone element
- **In Expressions**
  - as operand in an expression
  - as input for a predefined SQL function
  - directly after case in a case distinction
- **In the WHERE clause or HAVING clause**
  - Right side of a condition
- **In the ON condition of a Join**
  - Right side of a condition

: Input Parameters in Data Preview



: Example: Use Input Parameters in CDS DDL

```
@AbapCatalog.sqlViewName: 'S4D430_USEPARAM'
define view S4d430_Use_Parameters
  with parameters
       factor : abap.dec( 3, 2 )

as select
    from S4d430_Parameter( parameter1: 'Any Text',
                           parameter2: 'LH',
                           factor:     $parameters.factor,
                           separator:   'X'
                         )
      {
        ...
      }
```

CDS View with Parameters as data source

Parameter passing in parantheses „()“

This parameter is „forwarded“

Example: CDS View With Parameters in ABAP SQL

```
SELECT FROM s4d430_parameter(
                  parameter1 = @( `Carrier ` && pa_car ),
                  factor     = '0.93',
                  parameter2 = @pa_car,
                  separator  = 'X'  )
     FIELDS *
      WHERE carrid = @pa_car
       INTO TABLE @gt_data.
```

Assignment with „=“ instead of „:“

Sequence of parameters not fixed

Literals, host variables or host expressions

Restricted support in Release 7.40

Before Release 7.50:

- Input parameters only supported on SAP HANA database
- Check of system capability needed:

```
DATA gt_feature_set TYPE cl_abap_dbfeatures=>features_set_t.

INSERT cl_abap_dbfeatures=>views_with_parameters
       INTO TABLE gt_feature_set.

IF cl_abap_dbfeatures=>use_features( gt_feature_set )
   = abap_false.
...
ENDIF.
```

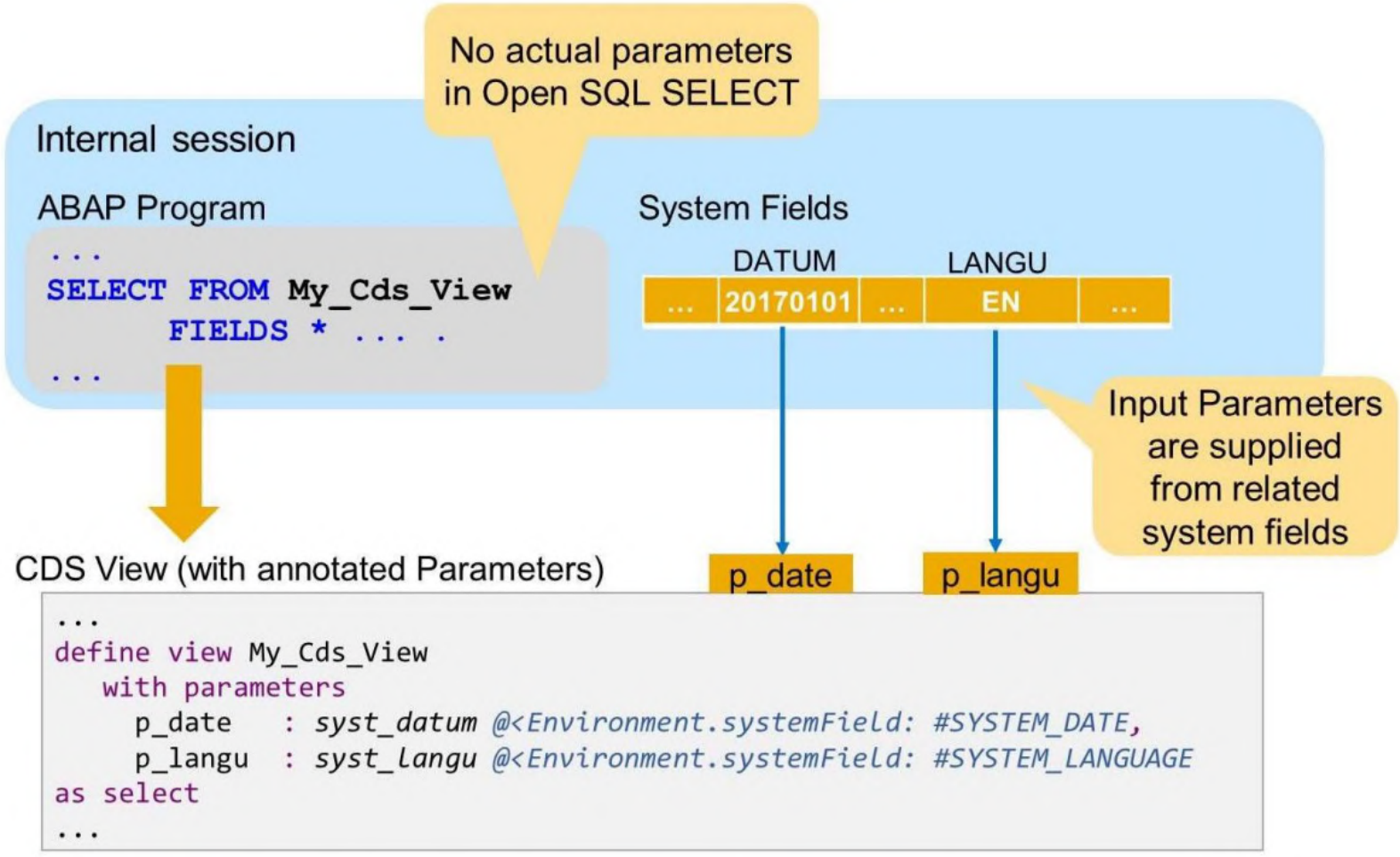
As of Release 7.50:

- Input parameters supported on all DB platforms
- Syntax above obsolete (warning from syntax check)

As of Release 7.52:

- Constant *views_with_parameters* removed from class CL_ABAP_DBFEATURES
- Syntax above causes syntax error!

Optional Parameters for CDS Views



: Rules for Annotation Environment.systemField

- **7 different values**

Value for Annotation systemField	Related ABAP system field
#CLIENT	*sy-mandt*
#SYSTEM_DATE	*sy-datum*
#SYSTEM_TIME	*sy-uzeit*
#SYSTEM_LANGUAGE	*sy-langu*
#USER	*sy-uname*
#USER_DATE	*sy-datlo*
#USER_TIMEZONE	*sy-zonlo*

- **Strictly once per parameter**
- **Has an effect only in Open SQL**
  (when using the view in another CDS view, the annotation is ignored)
- **Special rule for *#CLIENT:***
  Explicit Actual value not allowed

Session Variables

Session Variables in CDS ABAP

- **Are global variables of the Database**
- **Correspond to system fields of the ABAP runtime**

Session Variable	Related ABAP system field
$session.client	*sy-mandt*
$session.system_date	*sy-datum*
$session.system_language	*sy-langu*
$session.user	*sy-uname*
$session.user_date	*sy-datlo*
$session.user_timezone	*sy-zonlo*

- **Are set to their values when the view is used in ABAP SQL**
- **Have undefined content if the CDS View is not used in ABAP SQL**

To access ABAP system fields, it is generally preferable to use annotated input parameters

Example for Using Session Variables in ABAP CDS

```
@AbapCatalog.sqlViewName: 'S4D430_SESVAR'

define view S4D430_Session_Variables
   as select from scarr
  left outer join tcurt
               on scarr.currcode = tcurt.waers
              and tcurt.spras    = $session.system_language

                {

                  key scarr.carrid,
                      scarr.carrname,
                      scarr.currcode,
                      tcurt.ltext

                }
```

Client Session Variable

- In ABAP if you want to access the runtime information of a current session we use system variables. Similarly we can also do the same in CDS Views using Session variables.
- We can use these Variables within the data selection of the CDS Views or at the where condition directly.
- Client Handling: Usually in ABAP whenever we are querying the data using open SQL, the client- specific data is automatically handled by the current client of the ABAP Session. Meanwhile, the data will be fetched from the current client.
- If you have to fetch data specific to the client, we use CLIENT SPECIFIED as apart of the query. Just like this →

EG: SELECT * FROM vbak CLIENT SPECIFIED
INTO TABLE @dara(it_vbak)
WHERE client ='200'.

Associations in CDS View

Association in simple terminology:

An association documents the relationship between a source view and target view.

```
EG: define view Zdemo_SalesItems
As select from SNWD_SO_I
association (0..1) to SNWD_PD
as_Products on $projection.Product
=_Products.Product
{
… ,
_Products
}
```

EXPLANATION

- As per the specified ON condition , data records of the source view and target view are linked to each other if their values for field Product are same.
- One more thing we need to observe here is cardinality of the association . Depending whether there is no or single matching record of product, the cardinality is defined here as [0...1].
- Which means there can be no or single matching record of the product can exist.
- In addition we can expose the associations to the outer world (consumer of the CDS View)

- Apart from data provisioning, published association also carry the metadata information for frameworks. These frameworks can interpret and derive many functions from this information.
- Association work as LEFT OUTER by default , we can change this behaviour if needed.

EG: ***define view Zdemo_SalesItem as select from SNWD_SO_I Association [0..1] to SNWD_PD as _Products on Sportjection.Product =_Products.Product { … , _Products }***

Cardinality	Minimum	Maximum
[1]	0	1
[0..1]	0	1
[1..1]	1	1
[0..*]	0	unlimited
[1..*]	1	unlimited

Lesson Objectives

After completing this lesson, you will be able to:

- Define Views with Associations
- Expose Associations
- Use Exposed Associations in Path Expressions
- Understand Filtered Associations

Lesson Agenda

Learn About:

- Associations Principle
- Syntax of Associations
- Associations and Cardinality
- Syntax of Exposed Associations
- Template Define View With Association
- Exposed Associations in Data Preview
- Path Expressions in CDS DDL
- Path Expressions in ABAP SQL
- Syntax of Filtered Associations
- Cardinality in Filtered Associations
- Annotation compareFilter

Do:

- Define a CDS View with Associations
- Use a CDS View with Associations

: What Are Associations?

- **A High-value Wrapping of the Syntax for Joins**
  - Associations define relationships (they not just join data sources)
  - Associations are translated into joins on database level
- **Easier to Read**
  - Associations may contain additional **semantic information** (e.g. cardinality)
  - **Path expressions** reveal the underlying data model and origin of data
  - **Filter** conditions are easier to read than WHERE conditions or CASE-expressions
- **Improved Performance**
  - **Exposed associations** are only evaluated when needed ("JOIN on Demand")

Example: Association Instead of Join

View Definition with Association

```
@AbapCatalog.sqlViewName: 'S4D430_ASSO1'
define view S4d430_Association_1 as select
  from spfli association to scarr
    on spfli.carrid = scarr.carrid
   { key carrid,
     key connid,
         scarr.carrname
   }
```

Association target

Name of data source mandatory for all fields from association target

View Definition with Join

```
@AbapCatalog.sqlViewName: 'S4D430_JOININN'
define view S4d430_JOIN_INNER as select
  from spfli inner join scarr
    on spfli.carrid = scarr.carrid
     { key spfli.carrid,
       key connid,
           carrname
     }
```

Name of data source only mandatory for non-unique field names

Technical Realization of Associations

View Definition with Association

```
@AbapCatalog.sqlViewName: 'S4D430_ASSO1'
define view S4d430_Association_1 as select
  from spfli association to scarr
    on spfli.carrid = scarr.carrid
   { key carrid,
     key connid,
         scarr.carrname
   }
```

Corresponding SQL Create Statement

```
CREATE VIEW "S4D430_ASSO1" AS SELECT
  "SPFLI"."MANDT" AS "MANDT",
  "SPFLI"."CARRID",
  "SPFLI"."CONNID",
  "SPFLI"."CITYFROM",
  "SPFLI"."CITYTO",
  "=A0"."CARRNAME"
FROM "SPFLI" "SPFLI" LEFT OUTER JOIN "SCARR" "=A0" ON (
  "SPFLI"."MANDT" = "=A0"."MANDT" AND
  "SPFLI"."CARRID" = "=A0"."CARRID"
)
```

Example: Additional Semantics

```
@AbapCatalog.sqlViewName: 'S4D430_ASSO3'
define view S4d430_Association_3 as select

from spfli as c
     association[1..1] to scarr as _Carrier
       on c.carrid = _Carrier.carrid
   {
      key c.carrid as CarrierID,
      key c.connid,
          c.cityfrom,
          c.cityto,
          _Carrier.carrname
   }
```

Cardinality (exactly one carrier related to a connection)

Name of the association

Replaced with Alias in SQL Create Statement

Extract From the SQL Create Statement

```
  "=A0"."CARRNAME"
FROM "SPFLI" "C" LEFT OUTER JOIN "SCARR" "=A0" ON (
  "C"."MANDT" = "=A0"."MANDT" AND
  "C"."CARRID" = "=A0"."CARRID"
)
```

On-Condition with $Projection

```
@AbapCatalog.sqlViewName: 'S4D430_ASSO3'

define view S4D430_Association_3 as select
  from spfli as c
     association[1..1] to scarr as _Carrier

           on $projection.CarrierID = _Carrier.carrid
   {
      key c.carrid as CarrierID,
      key c.connid,
          c.cityfrom,
          c.cityto,
          _Carrier.carrname
   }
```

Use $projection on the left hand side of the ON-clause

If the field has an alias, the alias has to be used after $projection

Some Rules Regarding Cardinality

- **Cardinality is optional**
  - Default cardinality is [0..1]
- **Minimum value is optional**
  - Default Value for mimimum is 0

    [1] means [0..1]

    [4] means [0..4]

    [*] means [0..*]
- **Forbidden Values**
  - Minimum can not be *
  - Maximum can not be 0
- **Syntax Check for Maximum > 1**
  - Association cannot be used in a WHERE condition (syntax error)
  - Association should not be used outside aggregate expressions (syntax warning)

Example: Syntax Warning for Cardinality [*]

```
@AbapCatalog.sqlViewName: 'S4D430_ASSOWARN'
define view S4d430_Association_WARNING as select
  from spfli as c
    association[*] to sflight as _Flights     // short for [0..*]
                   on $projection.carrid = _Flights.carrid
                  and $projection.connid = _Flights.connid
   {
     key carrid,
     key connid,
         _Flights.fldate
}
```

Access to association with cardinality [*] increases result set

Result without field fldate

Raw Data

Filter pattern 26 rc

carrid	connid
AA	0017

… and with field fldate

Raw Data

Filter pattern 100 rows retrieved

carrid	connid	fldate
AA	0017	2015-11-18
AA	0017	2015-12-16
AA	0017	2016-01-13
AA	0017	2016-02-10
AA	0017	2016-03-09
AA	0017	2016-04-06

Example: Exposed Association

```
@AbapCatalog.sqlViewName: 'S4D430_ASSOEXP'
define view S4d430_Association_Exposed as select
 from spfli as c
     association[1] to scarr as _Carrier
                    on $projection.carrid = _Carrier.carrid
     association[*] to sflight as _Flights
                    on $projection.carrid = _Flights.carrid
                   and $projection.connid = _Flights.connid

{
      key c.carrid,
      key c.connid,
          c.cityfrom,
          c.cityto,

          _Carrier,
          _Flights
   }
```

Prerequisite: all fields used in the ON condition also in element list

Association target visible to consumer of this CDS View

Difference Between Exposed and Ad-hoc Associations

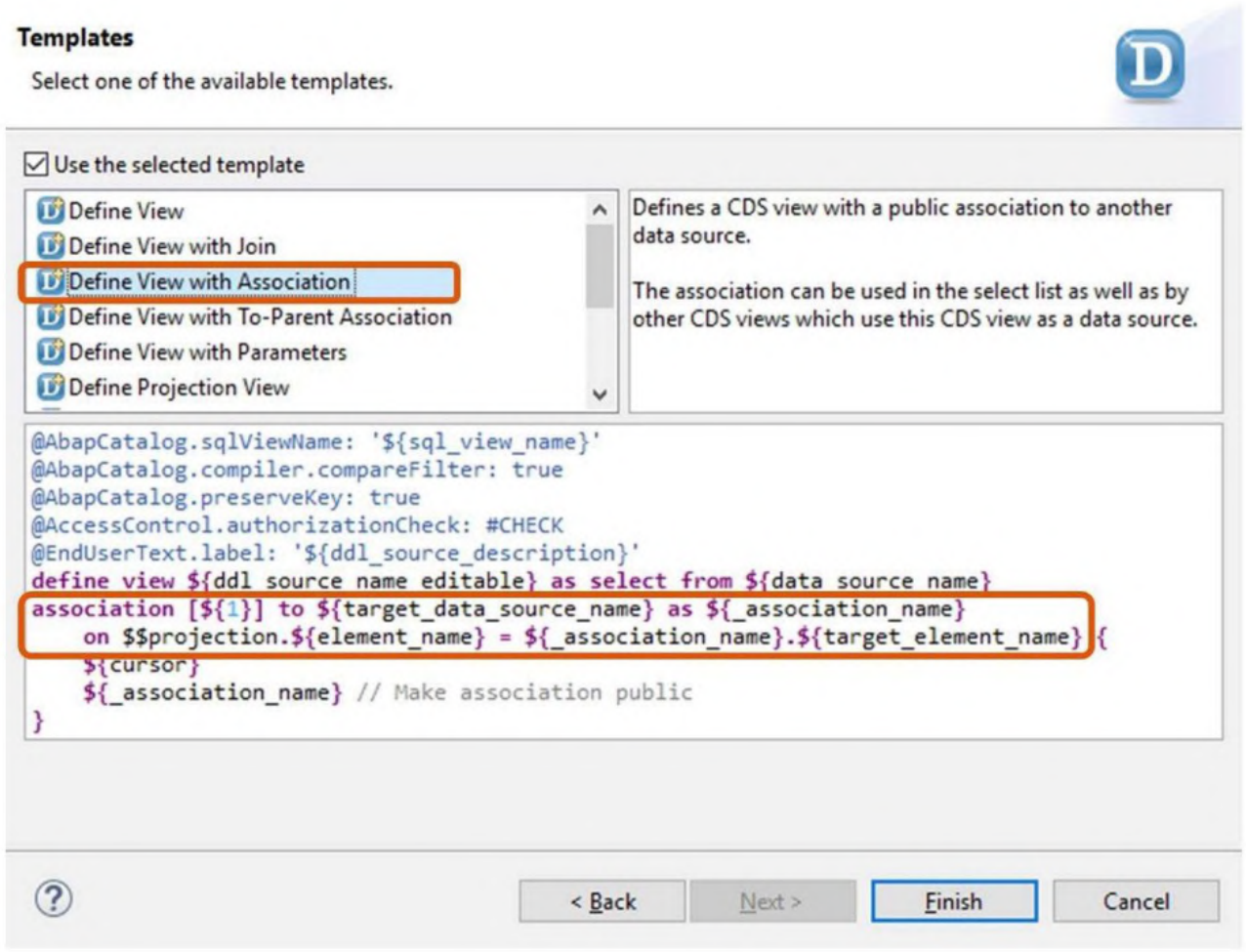
```
@AbapCatalog.sqlViewName: 'S4D430_ASSOEXP'
define view S4d430_Association_Exposed as select
 from spfli as c
     association[1] to scarr as _Carrier
                    on $projection.carrid = _Carrier.carrid
     association[*] to sflight as _Flights
                    on $projection.carrid = _Flights.carrid
                   and $projection.connid = _Flights.connid
   {
      key c.carrid,
      key c.connid,
          c.cityfrom,
          c.cityto,
          _Carrier,
          _Flights
   }
```

No warning if association with cardinality [*] is exposed

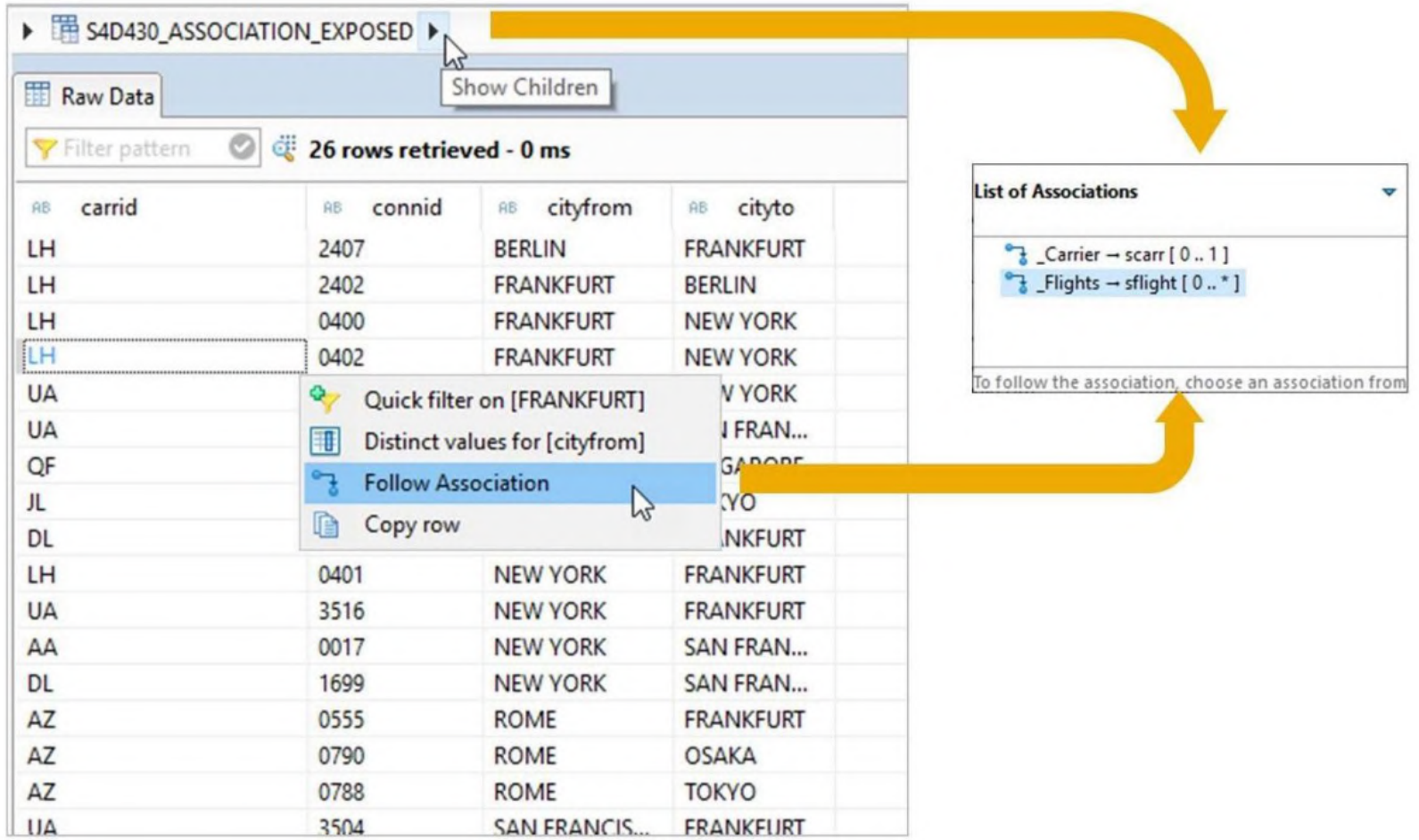
Exposed Associations are not (yet) joined on database level

```
CREATE VIEW "S4D430_A    XP" AS SELECT
  "C"."MANDT" AS "MA    ,
  "C"."CARRID",
  "C"."CONNID",
  "C"."CITYFROM",
  "C"."CITYTO"
FROM "SPFLI" "C"
```

Template Define View with Association



Exposed Associations in Data Preview



Example: Use Exposed Associations in a View Definition

```
@AbapCatalog.sqlViewName: 'S4D430_PATHEXPR1'
define view S4d430_Path_Expressions_1 as select
 from S4d430_Association_Exposed as c
    {
      key carrid,
      key connid,

          c._Carrier.carrname as carrier_name,

          _Flights
    }
```

Read from CDS View that exposes Associations

Read field carrname from target of association _carrier

Propagate association _flights (Do not use it directly)

```
CREATE VIEW "S4D430_PATHEXPR1" AS SELECT
  "C"."MANDT" AS "MANDT",
  "C"."CARRID",
  "C"."CONNID",
  "=A0"."CARRNAME" AS "CARRIER_NAME"
FROM "S4D430_ASSOEXP" "C" LEFT OUTER JOIN "SCARR" "=A0" ON (
  "C"."MANDT" = "=A0"."MANDT" AND
  "C"."CARRID" = "=A0"."CARRID"
)
```

Join of SCARR, but no join of SFLIGHT

Example: Propagation of Associations

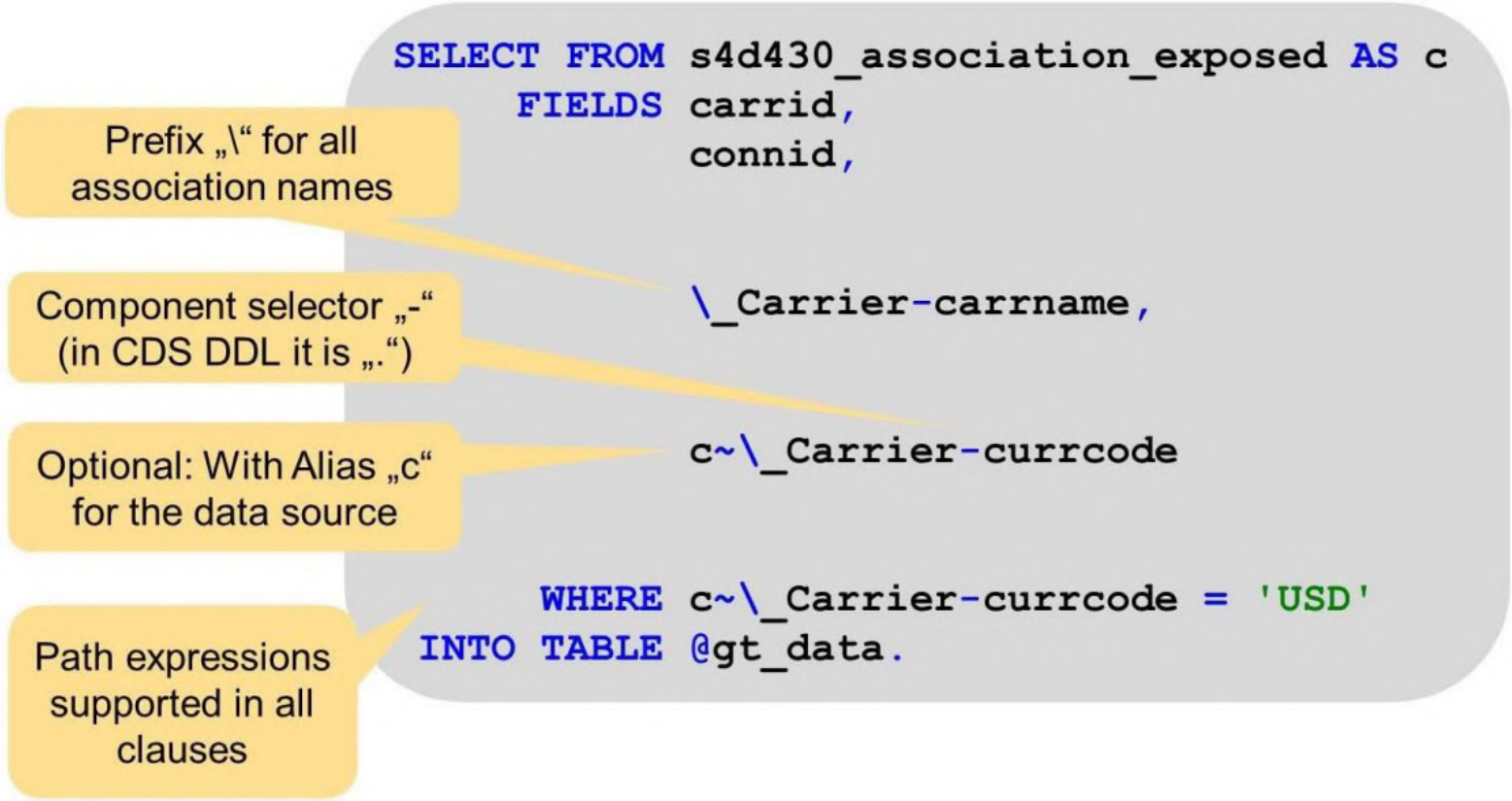
```
@AbapCatalog.sqlViewName: 'S4D430_PATHEXPR2,
define view S4d430_Path_Expressions_2 as select
 from S4d430_Path_Expressions_1 as c
    { key carrid,
      key connid,
          carrier_name,

          sum(c._Flights.seatsocc) as occupied_total
    }
group by carrid,
         connid,
         carrier_name
```

Read field seatsocc from target of association _flights

```
CREATE VIEW "S4D430_PATHEXPR2" AS SELECT
  "C"."MANDT" AS "MANDT",
  "C"."CARRID",
  "C"."CONNID",
  "C"."CARRIER_NAME",
  SUM(
    "=A0"."SEATSOCC"
  ) AS "SEATS_TOTAL"
FROM "S4D430_PATHEXPR1" "C" LEFT OUTER JOIN "SFLIGHT" "=A0" ON (
  "C"."CARRID" = "=A0"."CARRID" AND
  "C"."CONNID" = "=A0"."CONNID" AND
  "C"."MANDT" = "=A0"."MANDT"
)
```

Now also table SFLIGHT is joined

Example: Use Exposed Associations in ABAP SQL



Filtered Associations – The Principle

Data Definition

```
...
from t1 association to t2 as _asso
        on <on-condition>
    {
     ...
     _asso[<filter-condition>].field,
     ...
    }
```

SQL View Definition

```
...

FROM T1 LEFT OUTER JOIN T2
   ON <on-condition>
  AND <filter-condition>

...
```

: Example: Filter Condition in Path Expression

```
@AbapCatalog.sqlViewName: ,S4D430_ASSOFILT'
define view S4d430_Association_FILTERED
  with parameters
     language : syst_langu @<Environment.systemField: #SYSTEM_LANGUAGE

as select
 from scarr as a
      association to tcurt as _Currency
      on $projection.currcode = _Currency.waers
 //                  and _currency.spras = $parameters.language
    {
     carrid,
     carrname,
     currcode,
     $parameters.language as lang,

     _Currency[spras = $parameters.language].ktext as currency_name
    }
```

Do not restrict to one language in the ON condition

But use a filter condition in the Path Expression

Rules for Filter Conditions

- **Allowed Operators**

  =, <, >, <=, >=, <>

- **Left-hand Side**
  - Must be a field of association target
- **Right-hand Side**
  - field of association target
  - literal
  - parameter
  - session variable

Example: Cardinality in Filtered Associations

```
@AbapCatalog.sqlViewName: 'S4D430_FILTCARD'
@EndUserText.label: 'Demo: Cardinality in Filtered Asso.'
define view S4D430_Asso_Filter_Cardinality
  with parameters
     language : syst_langu @<Environment.systemField: #SYSTEM_LANGUAGE
as select
 from scarr as a
      association[*] to tcurt as _Currency
      on $projection.currcode = _Currency.waers
    {
     carrid,
     carrname,
     currcode,
     $parameters.language as lang,

     _Currency[1:spras = $parameters.language].ktext as currency_name
    }
```

„Foreign key relation has cardinality 0..* “

„But filter condition reduces this to 0..1 “

Annotation
AbapCatalog.compiler.compareFilter

```
...
from t1 association to t2 as _asso
        on <on-condition>
    {...
     _asso[<filter-condition>].field1,
     _asso[<filter-condition>].field2,
     ... }
```

With Value *False*

```
...
FROM T1 LEFT OUTER JOIN T2 AS A0
   ON <on-condition>
  AND <filter-condition>
   LEFT OUTER JOIN T2 AS A1
 ON <on-condition>
  AND <filter-condition>
...
```

With Value *True*

```
...
FROM T1 LEFT OUTER JOIN T2
   ON <on-condition>
  AND <filter-condition>

...
```

Example with compareFilter: False

```
@AbapCatalog.sqlViewName: 'SD430_FILTCOMP'

@AbapCatalog.compiler.compareFilter: false

define view S4D430_Asso_Filter_Compare
    with parameters
     language : syst_langu @<Environment.systemField: #SYSTEM_LANGUAGE
as select
 from scarr as a
      association to tcurt as _Currency
      on $projection.currcode = _Currency.waers
    {
     carrid,
     carrname,
     currcode,
     $parameters.language as lang,

     _Currency[spras = $parameters.language].ktext as currency_name,
     _Currency[spras = $parameters.language].ltext as curr_description
    }
```

compareFilter is switched off

Two separate joins for the identical paths

Lesson Objectives

After completing this lesson, you will be able to:

- Understand the Concept of CDS View Enhancements
- Enhance a CDS View
- Understand Metadata Extensions

Lesson Agenda

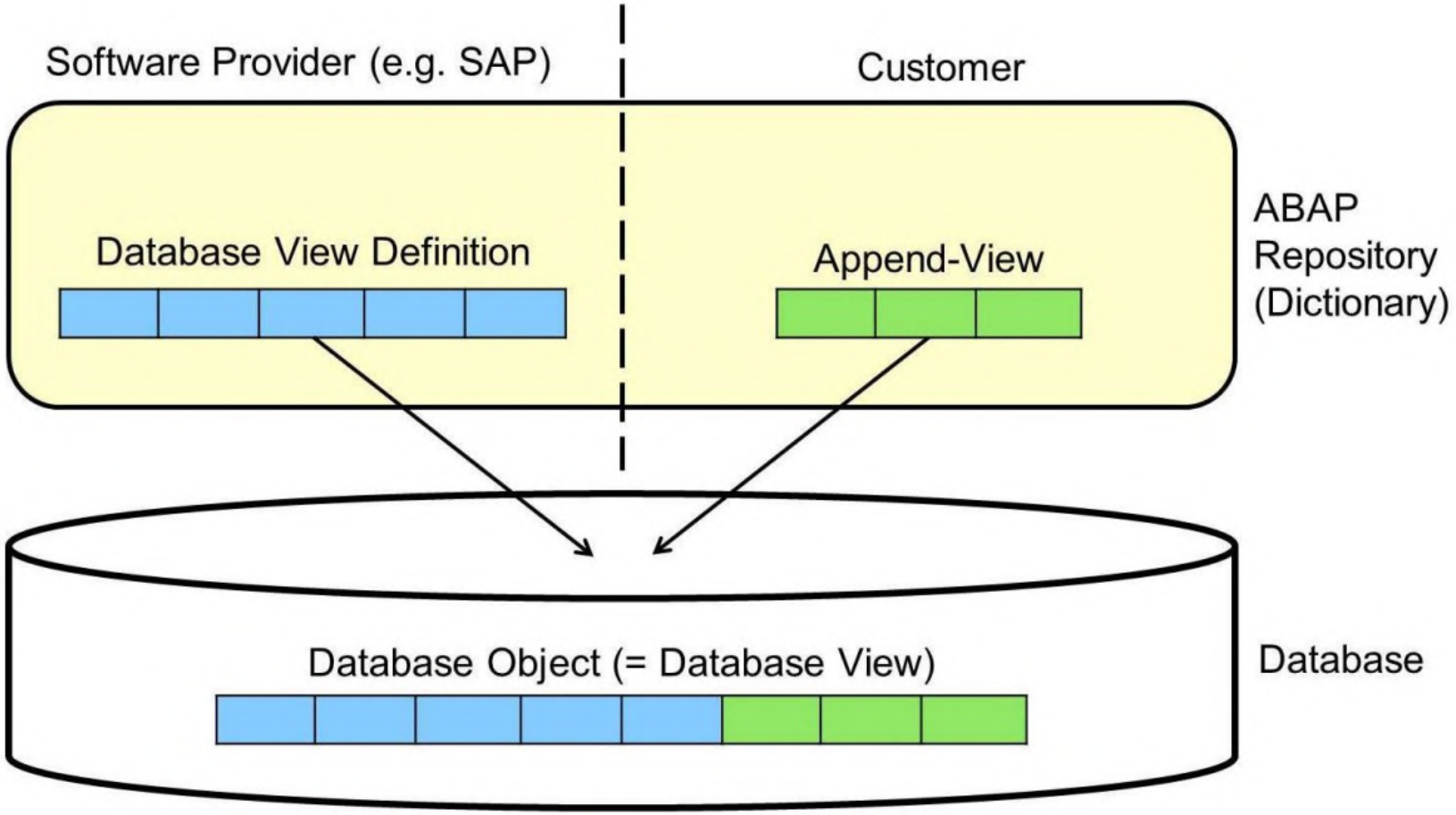
Learn About:

- Reminder - Append View
- CDS View Extensions
- Template Extend View
- Recommendations and Restrictions for View Extensions
- Annotation viewEnhancementCategory
- Navigation from Extension Target to View Extension
- Metadata Extensions

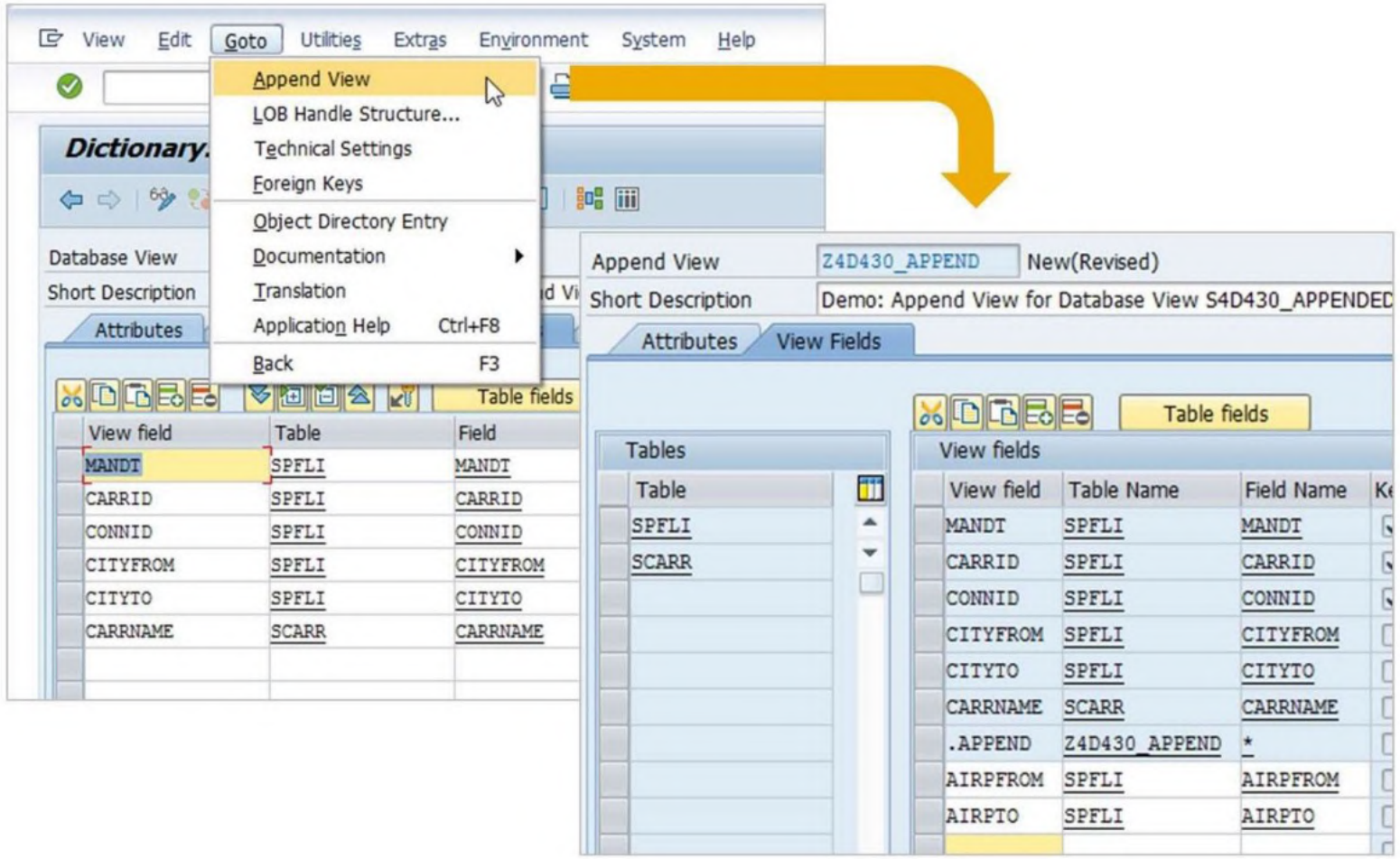
Do:

- Enhance a CDS View

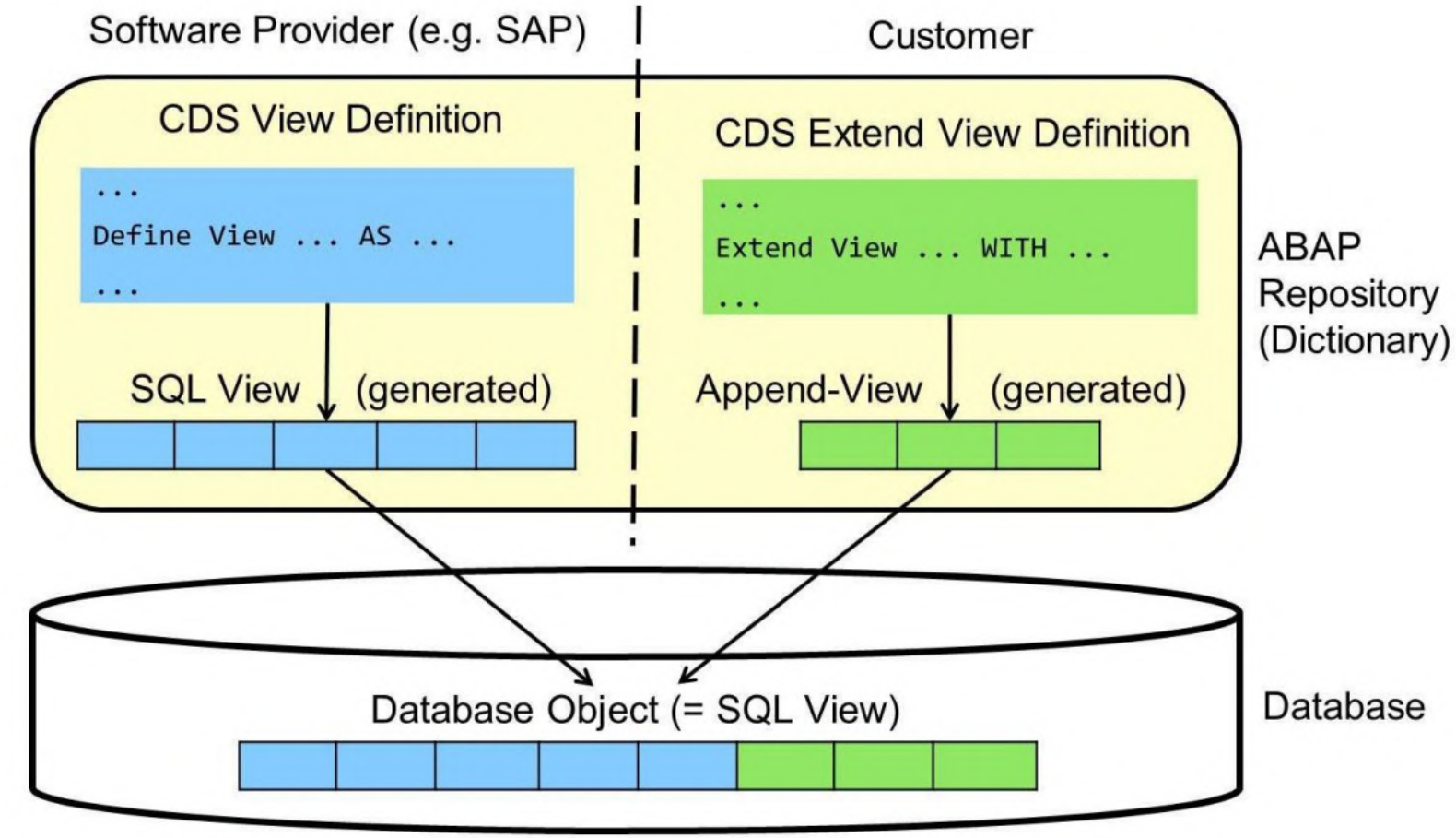
Reminder: Append Views for Classical Database Views



Example: Creating a Classical Append View



CDS View Extensions



Example: CDS View and CDS View Extension

Data Definition with View Extension (Created by Customer)

```
@AbapCatalog.sqlViewAppendName: 'Z4D430_EXTENDA'
extend view S4d430_Extend_Target_A
       with Z4d430_Extend_View_A
     { c.airpfrom,
       c.airpto
     }
```

Points at an existing CDS View

Data Definition with View Definition (Created by Software Provider)

```
@AbapCatalog.sqlViewName: 'S4D430_EXTTARGA'
define view S4d430_Extend_Target_A as select
  from spfli as c
     inner join scarr as a
             on c.carrid = a.carrid
     { key c.carrid,
       key c.connid,
           c.cityfrom,
           c.cityto,
           a.carrname
     }
```

: Syntax of View Extensions

- **Element List**
  - literals, fields, expressions, functions
  - Input parameters
  - Path expressions
  - Aggregates (as of 7.51, only if target already contains aggregates)
- **Associations**
  - View Extension can add associations
- **GROUP BY Clause (as of release 7.51)**
  - Only if target view contains a GROUP BY clause
  - Mandatory for new elements that are not aggregates
- **UNION(ALL) Statements (as of release 7.51)**
  - Mandatory if target view contains UNION (ALL) statements
  - Same number of UNION(ALL) Statements required

: Example: CDS View Extension with Association

Data Definition with View Extension

```
@AbapCatalog.sqlViewAppendName: 'Z4D430_EXTENDB'
extend view S4d430_Extend_Target_B
       with Z4d430_Extend_View_B
        association to sairport as _From
                    on $projection.airpfrom = _From.id
        association to sairport as _To
                    on $projection.airpto = _To.id
    {
      c.airpfrom,
      _From.name as airpfrom_name,
      c.countryfr,
      c.airpto,
      _To.name as airpto_name,
      c.countryto
    }
```

Add associations to target view

Use elements of the associated data sources

Example: CDS View Extension with GROUP BY and UNION

Data Definition with View Extension (Created by Customer)

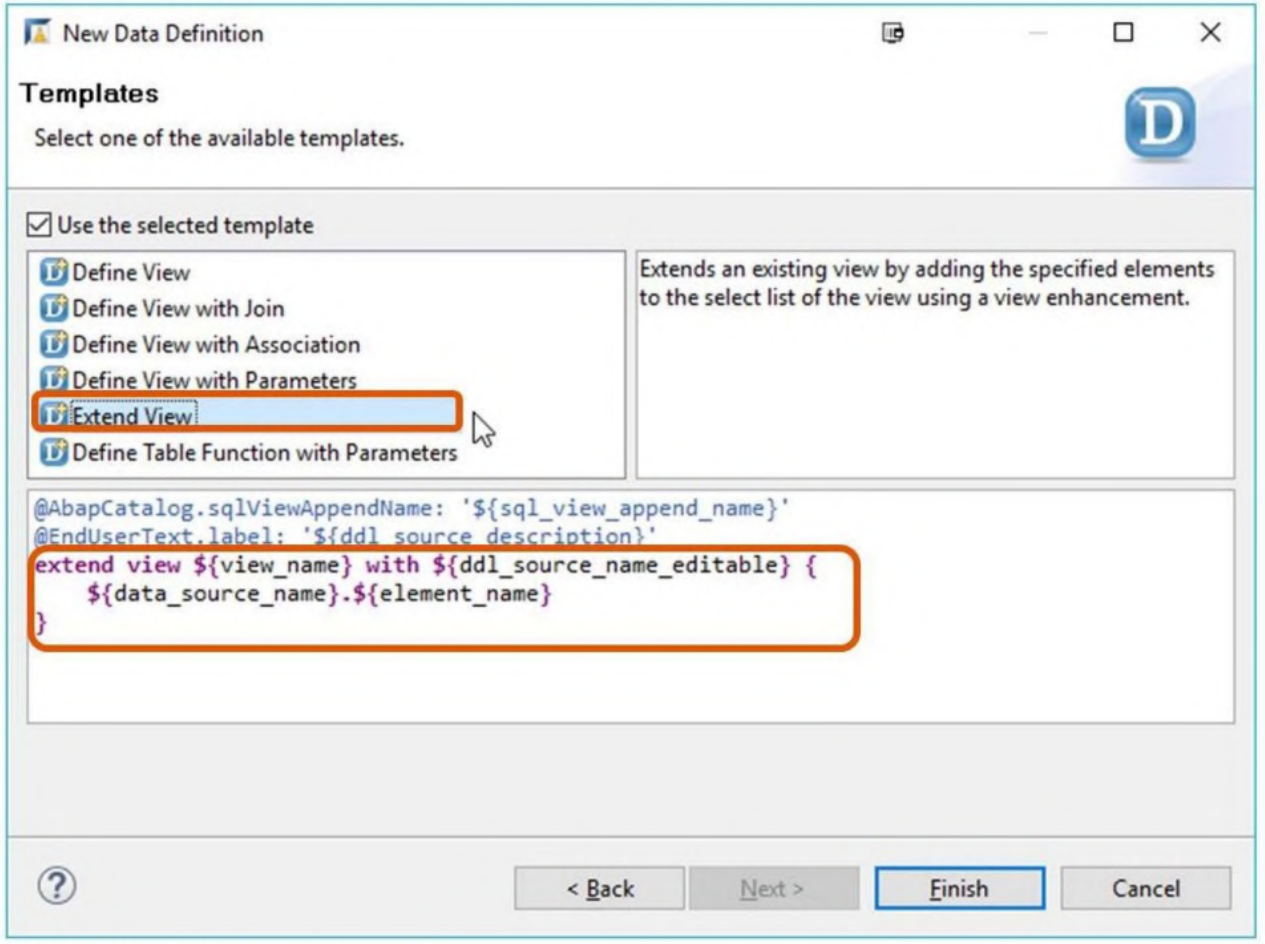
```
@AbapCatalog.sqlViewAppendName: 'S4D430_EXTENDC'
extend view S4D430_Extend_Target_C
       with S4D430_EXTEND_VIEW_C

   {   city,
       country,
       max( _Booking.fldate ) as last_booking
     }
group by city,
         country

union

     {  city,
        country,
       max( _Booking.fldate ) as last_booking
     }
 group by city,
          country
```

Additional aggregations in element list

Extension of the GROUP BY clause

Key word UNION, followed by extension for the other SELECT

Template Extend View



Recommendations

- **Naming**
  - (Customer) Namespace for repository objects (Data Definition and SQL Append View)
  - Uncommon names for additional elements and associations (to avoid later conflicts)
  - Identical Name for Data Definition and CDS Entity (besides upper/lower case)

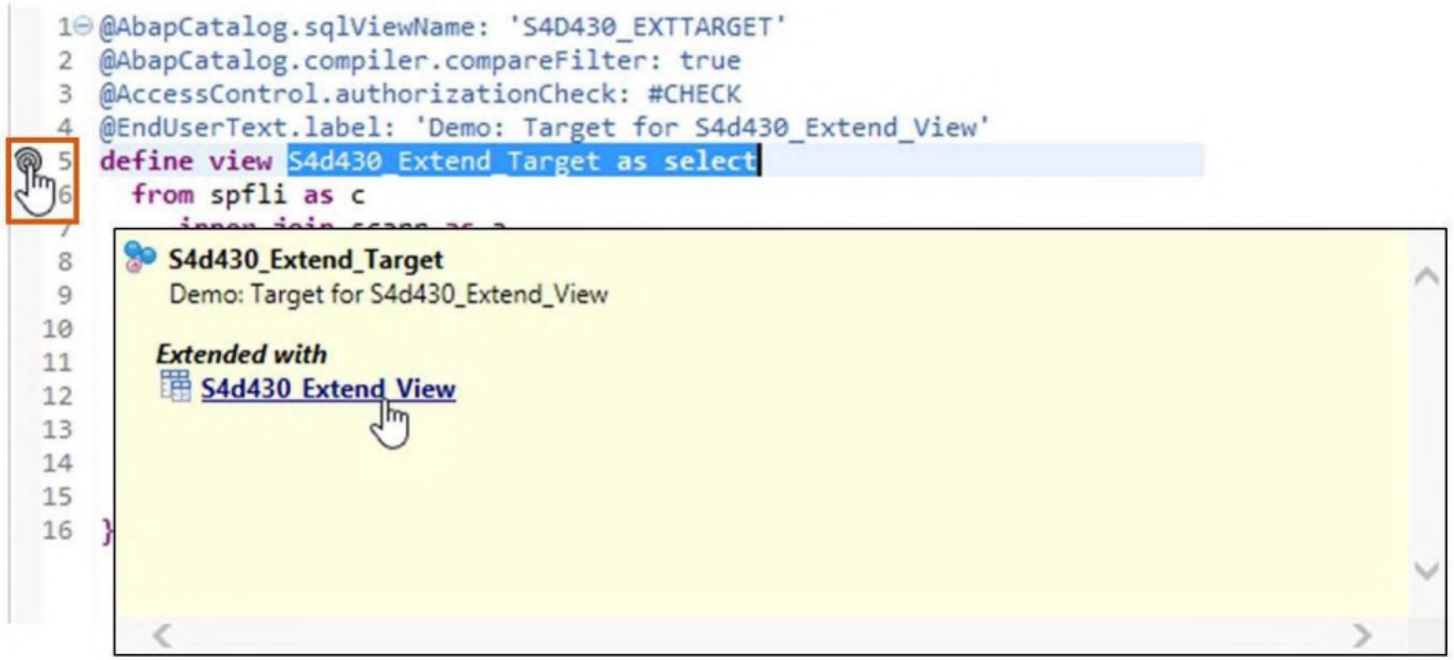
Restrictions

- **No Renaming After Transport**
  - Name of CDS View Extend and SQL Append View can not be changed after first transport of Data Definition
- **No Additional Key Fields**
  - Prefix KEY is not allowed in CDS View Extends

: Restricting Extensibility (as of Release 7.51)

- **Annotation *AbapCatalog.viewEnhancementCategory[ ]***
  - Restricts the extensibility of a CDS View
  - Comma-separated list of values in square brackets
- **Possible Values**
  - #NONE
  - #PROJECTION_LIST
  - #GROUP BY
  - #UNION
- **Allowed Combinations**
  - #NONE cannot be combined with other values
  - #GROUP BY and #UNION require #PROJECTION_LIST
- **Default Value**
  - #PROJECTION_LIST

Extended View in DLL Source Editor



Example: CDS View and Metadata Extension

Metadata Extension

```
@Metadata.layer: #CUSTOMER

annotate view S4D430_Metadata_Ext_Target with
{
  @EndUserText.label: 'Layer CUSTOMER'
    col_5;
}
```

Mandatory annotation specifies the extension layer

Points at an existing CDS View

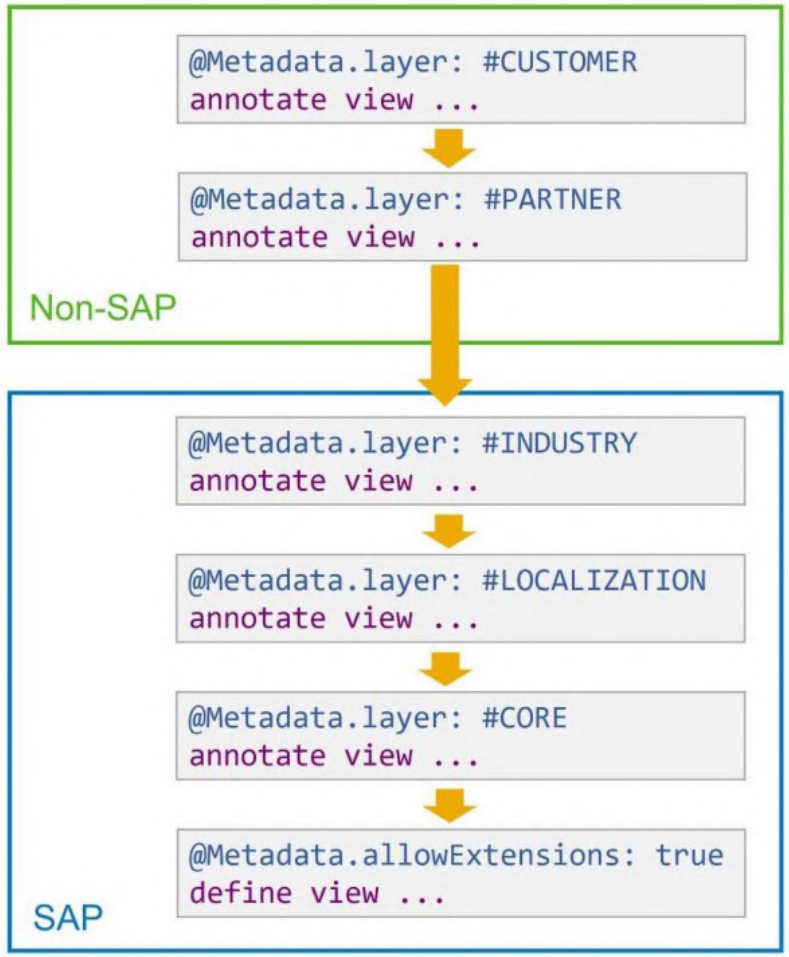
Data Definition

```
@AbapCatalog.sqlViewName: 'S4D430_MDE'

@Metadata.allowExtensions: true

define view S4D430_Metadata_Ext_Target as select
       from scarr
       {
         ...
         @EndUserText.label: 'No Extension'
         'Column5' as col_5
       }
```

Pre-requisite for metadata extension

Metadata Extension Layers

- **Several Matadata Extensions for same CDS View**
- **Priority defined through annotation @Metadata.layer**
- **Values:**
  - #Customer
  - #Partner
  - #Industry
  - #Localization
  - #Core
- **Highest Priority: #Customer**



Lesson Objectives

After completing this lesson, you will be able to:

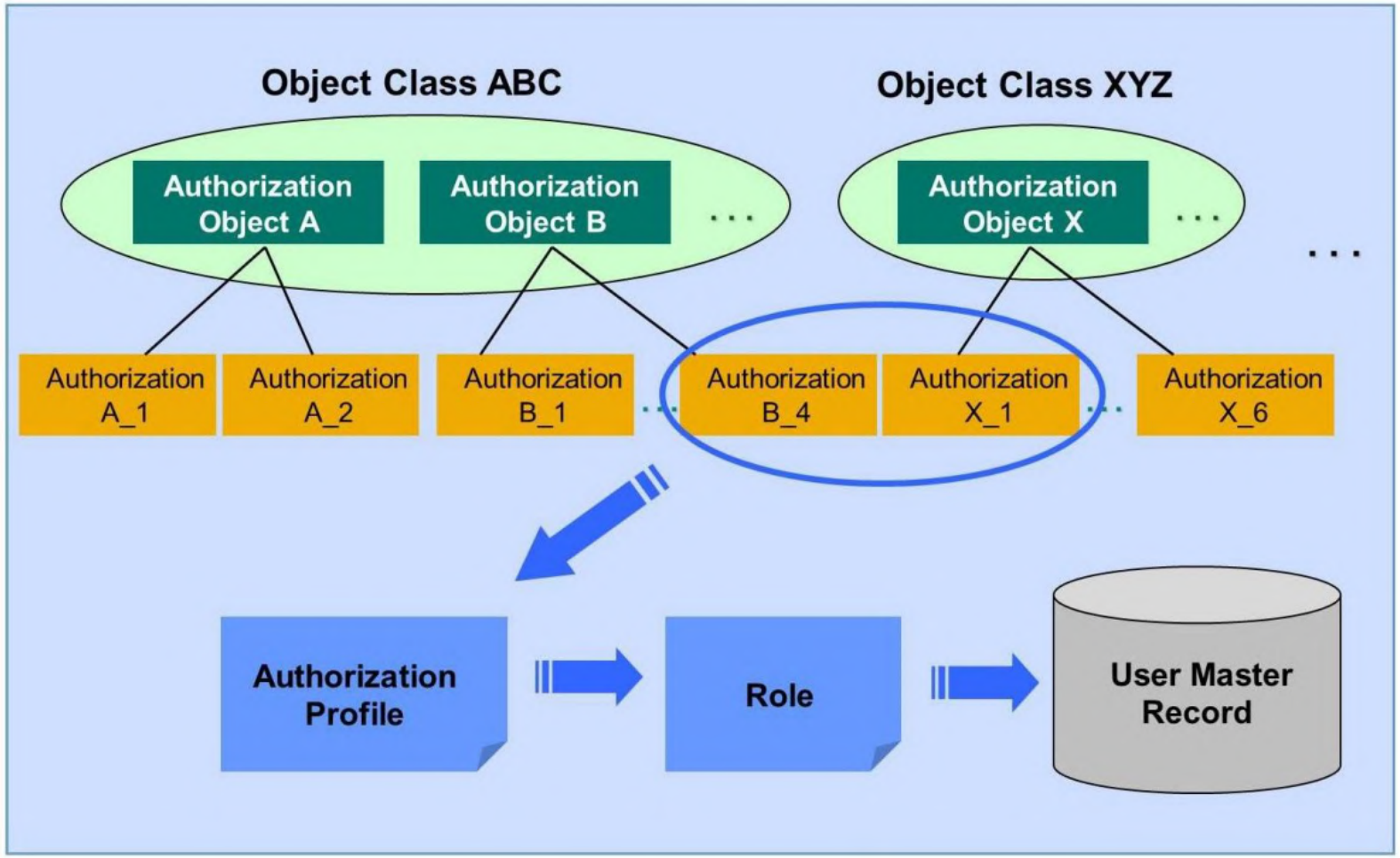
- Understand the Concept of CDS DCL
- Create a Access Control

Lesson Agenda

Learn About:

- Reminder - Classical Approach to Authorization Checks
- The Need for Authorization Checks on Database Level
- Repository Object Access Control
- Create a New Access Control
- CDS DCL Syntax
- Access Conditions in Access Controls
- Templates for Access Controls
- Authorization-related Annotations

: Reminder: Authorization Objects, Profiles, and Roles



Reminder: Implicit and Explicit Authorization Checks

Implicit Authorization Checks

- **Performed by the ABAP Framework**
  - Start of a transaction
  - Start of a Web Dynpro application
  - Start of a report
  - Call of an RFC function module
  - Access to table content with data browser or maintenance dialogue

Explicit Authorization Checks

- **Defined in the ABAP Coding (AUTHORITY-CHECK statement)**
  - Access to specific functionality
  - Access to specific data

: Known Problems of the Classical Approach

- **General Problems:**
  - No visible link between data and authorization objects
  - Developer can forget (or ignore) necessary checks
  - Users with debugging rights can bypass authorization checks

- **Problems related to Code-to-Data:**
  - All authorization checks on application level
  - Some calculations cannot be done on the database because they require an authorization check between steps

Example: Classical Approach to Authorization Checks

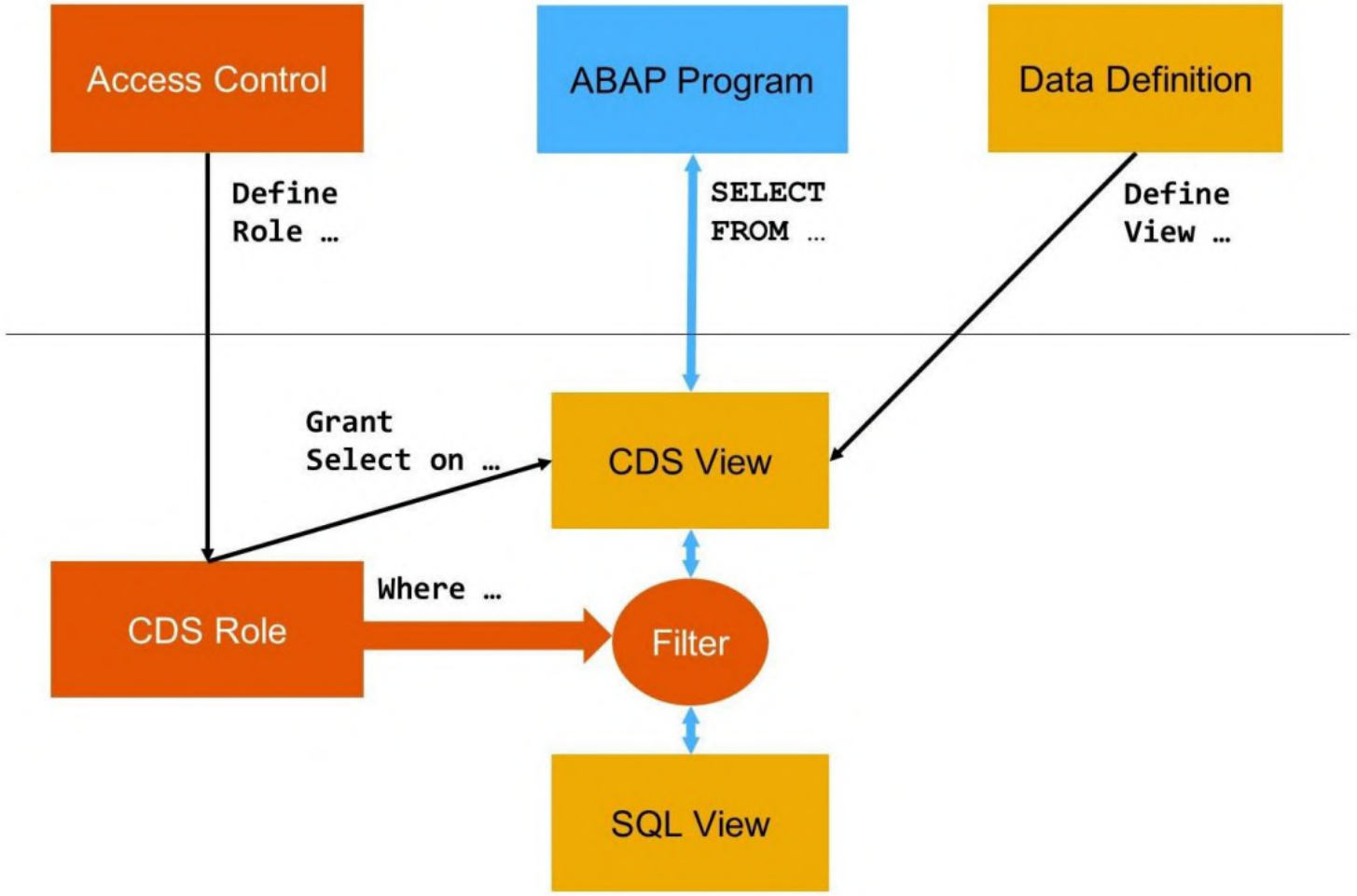
```
DATA gt_connections TYPE TABLE OF s4d430_access_control_a.
DATA gs_connection LIKE LINE OF gt_connections.

PARAMETERS pa_citfr TYPE s_from_cit DEFAULT 'FRANKFURT'.

SELECT FROM s4d430_access_control_a
     FIELDS carrid, connid, cityfrom, cityto
     WHERE  cityfrom = @pa_citfr
     INTO TABLE @gt_connections.

LOOP AT gt_connections INTO gs_connection.

  AUTHORITY-CHECK OBJECT 'S_CARRID'
           ID 'CARRID' FIELD gs_connection-carrid
           ID 'ACTVT' FIELD '03'.
  IF sy-subrc <> 0.
    DELETE gt_connections INDEX sy-tabix.
  ENDIF.

ENDLOOP.
```

Read all connections departing from given city

Check Authorization for each connection

Remove connections from the result set

Access Control on Database – How It Works



Example: DLC Source

```
@EndUserText.label: 'Demo: Authorization Check'
@MappingRole: true
define role S4d430_Role_B
 {
  grant select
      on S4d430_Access_Control_B
   where carrid <> 'AZ';
  }
```

S4D430_ACCESS_CONTROL_A

Raw Data

Filter pattern — 26 rows retrieved - 0 ms

carrid	connid	cityfrom	cityto
AA	0017	NEW YORK	SAN FRANCISCO
AA	0064	SAN FRANCIS...	NEW YORK
AZ	0555	ROME	FRANKFURT
AZ	0788	ROME	TOKYO
AZ	0789	TOKYO	ROME
AZ	0790	ROME	OSAKA
DL	0106	NEW YORK	FRANKFURT
DL	1699	NEW YORK	SAN FRANCISCO

S4D430_ACCESS_CONTROL_B

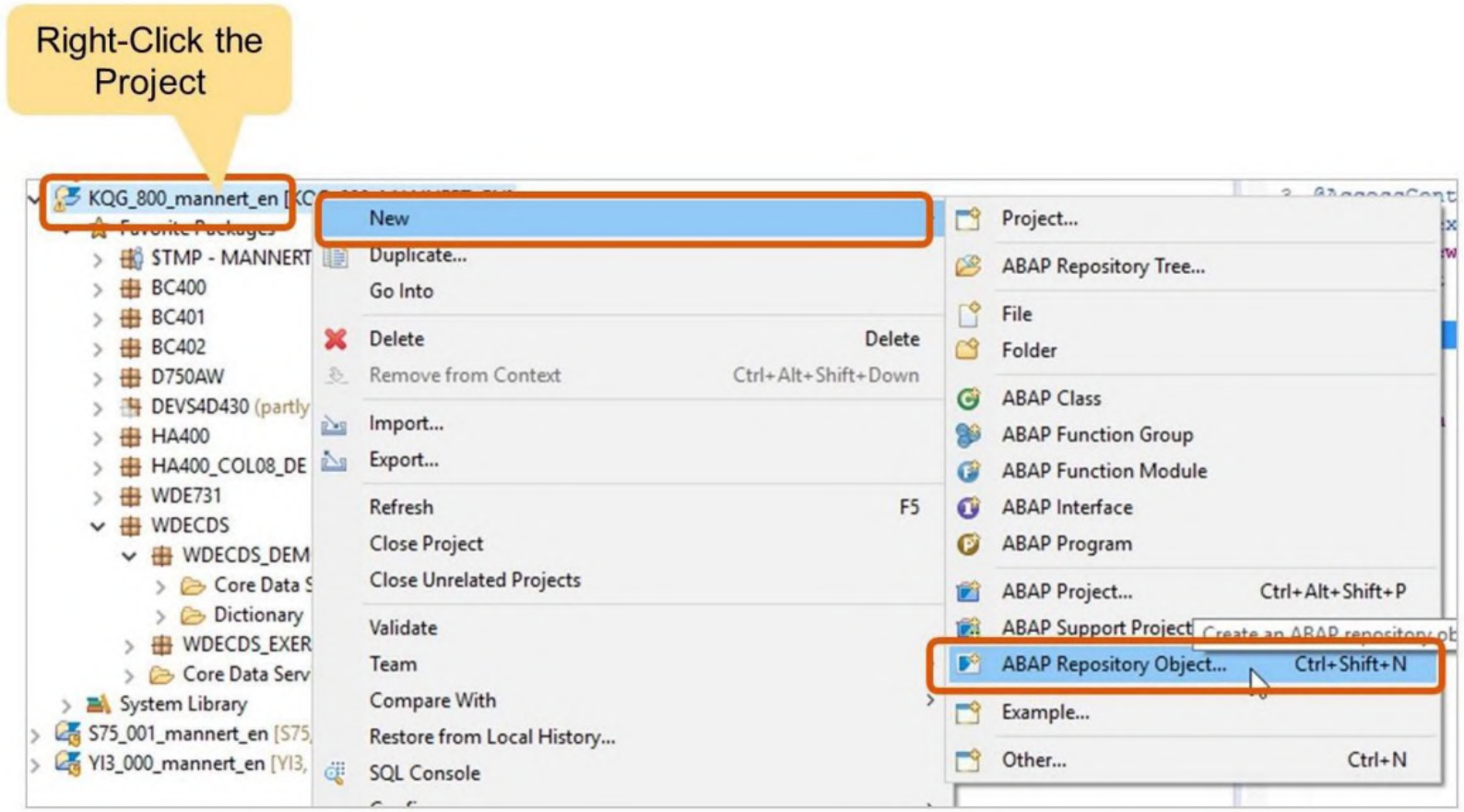
Raw Data

Filter pattern — 22 rows retrieved - 0 ms

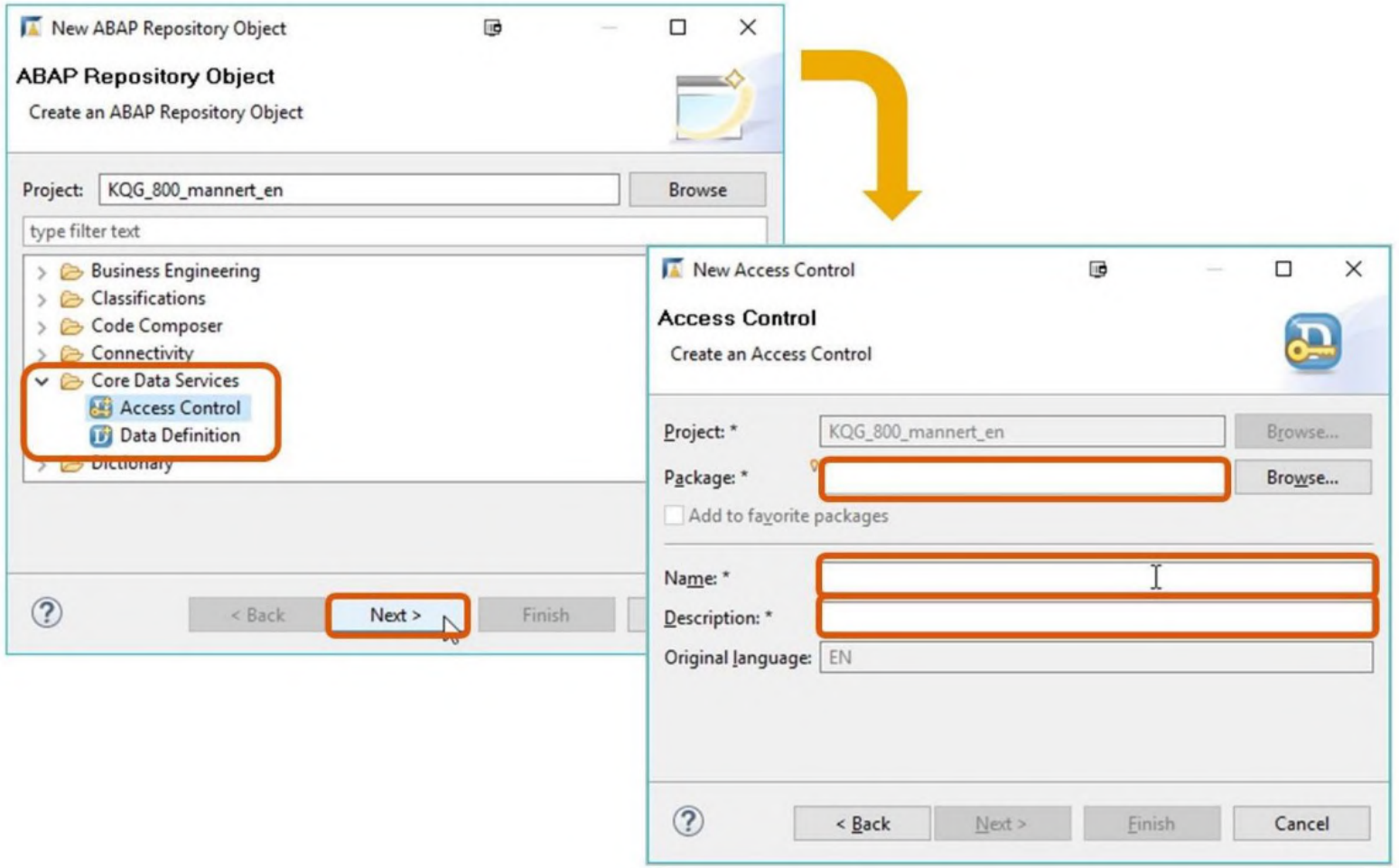
carrid	connid	cityfrom	cityto
AA	0017	NEW YORK	SAN FRANCISCO
AA	0064	SAN FRANCIS...	NEW YORK
DL	0106	NEW YORK	FRANKFURT
DL	1699	NEW YORK	SAN FRANCISCO
DL	1984	SAN FRANCIS...	NEW YORK

Flight connections with carrid = 'AZ' removed from result set

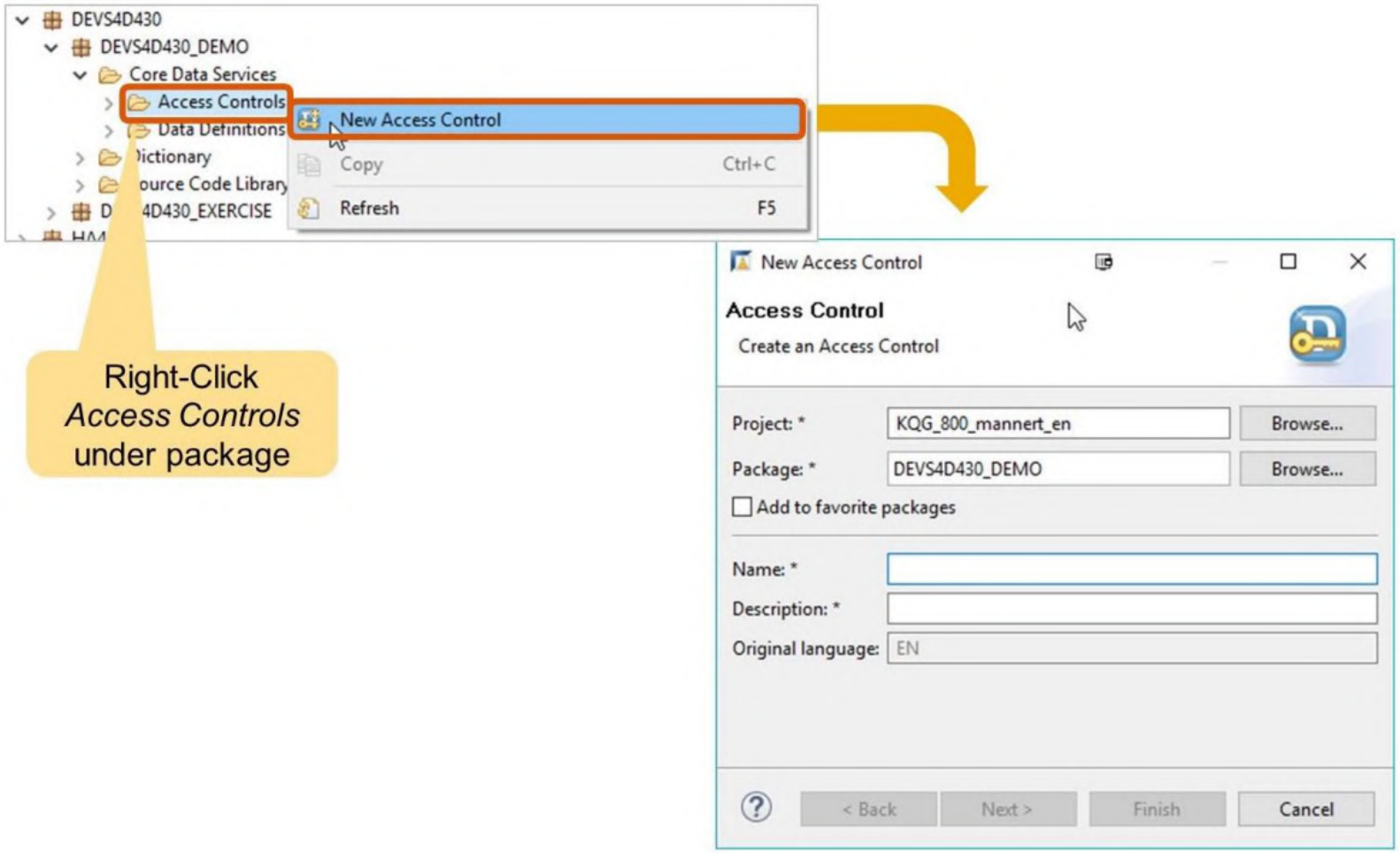
Create New Access Control (1)



Create New Access Control (2)



Shortcut (If Package Already Contains a Access Control)



ABAP CDS DCL Syntax

- **Same Basic Rules as for ABAP CDS DDL Syntax**
  - Case of key words and names
  - Syntax for literals, comments, statement delimiters
- **Statement DEFINE ROLE**

```
[DEFINE] ROLE <role_name>
  {
   GRANT SELECT ON <cds_entity1> WHERE <cond1> [AND|OR <cond2>] ...;
  [GRANT SELECT ON <cds_entity2> ...;]
   ...
  }
```

Key word DEFINE is optional

One or more GRANT statements between „{“ and „}”

At least one condition per GRANT statement

Delimiter „;“ mandatory after each GRANT statement

Naming Rules and Recommendations

Access Control name:

- Max. 30 Characters
- Unique throughout the system (Customer namespace!)
- Always Upper-Case

CDS Role name:

- Max 30 Characters
- Unique throughout the system (Customer namespace!)
- Not Case-sensitive
- Can be different from Access Control name (not recommended!)

Types of Conditions in DCL

- **Literal Conditions**

```
... WHERE <field name> = <literal value> ...
```

  - Use values specified literally to restrict access
  - Allow declaration of new authorizations

- **PFCG conditions**

```
... WHERE (<field1>, <field2>, ...) =
          aspect pfcg_auth( <authorization object>,
                            <authorization field 1>,
                            <authorization field 2>,
                            ...
                          ) ...
```

  - Evaluate authorizations in user master record
  - Based on authorization objects
  - Fit into SAP authorization concept

Example: PFCG Conditions

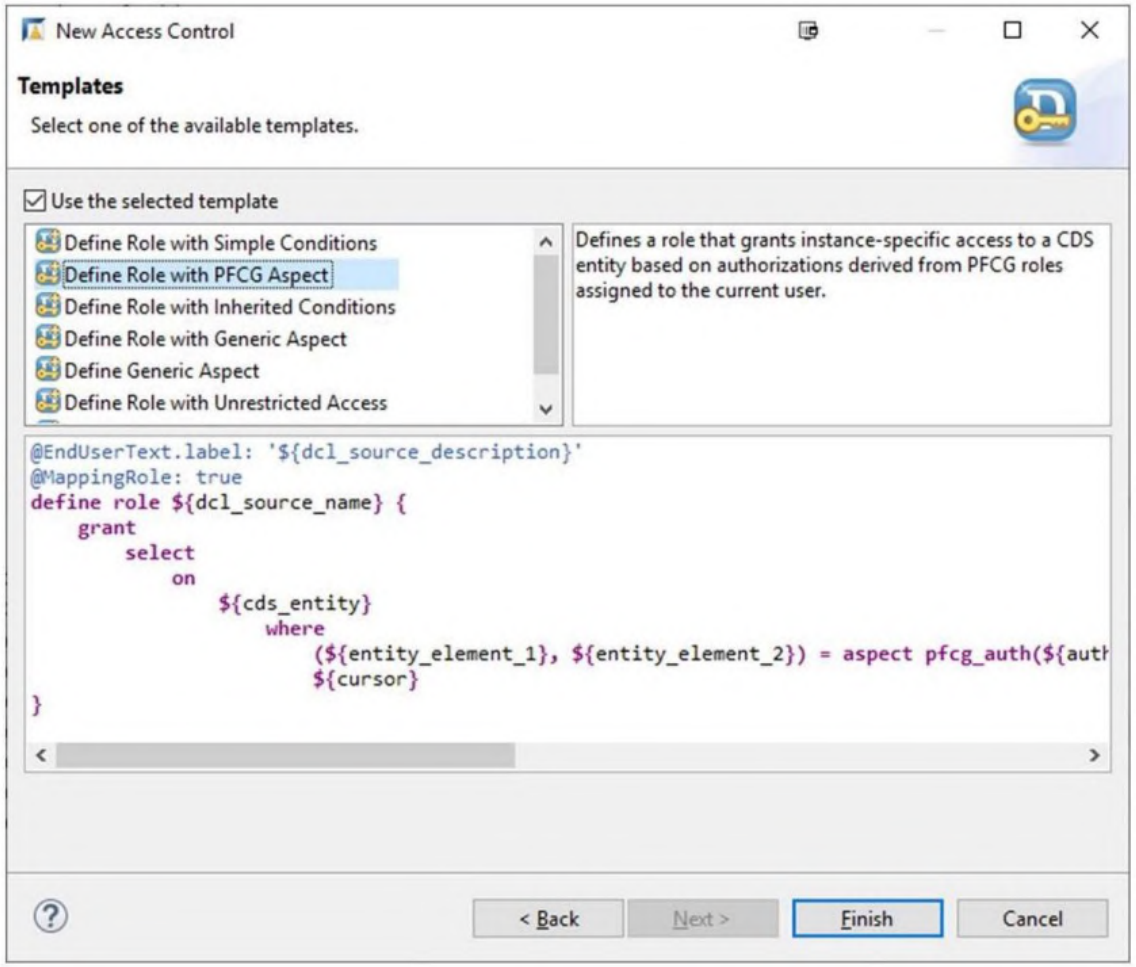
```
@EndUserText.label: 'Demo: Authorization Check With Link to User Profile'
@MappingRole: true
define role S4d430_Role_C
  { grant select
              on S4d430_Access_Control_C
           where (carrid)           = aspect pfcg_auth( S_CARRID,
                                                        CARRID,
                                                        ACTVT = '03'
                                                      )
             and (carrid, counter) = aspect pfcg_auth( S_COUNTER,
                                                        CARRID,
                                                        COUNTNUM,
                                                        ACTVT = '03'
                                                      );
  }
```

Left-hand side:
One or more comma-separated view fields in brackets

Right-hand side:
Key word ASPECT and function pfcg_auth( )

Arguments of pfcg_auth:
One authorization object with its fields

Templates for Access Controls



Annotations in Access Controls

- **@EndUserText.label:**
  - Translatable short text for role (same as in Data Definition)
- **@MappingRole:**
  - Value **`true:`** Role is implicitly assigned to all users
  - Value **`false:`** Currently not supported by ABAP CDS

Annotations in Data Definitions

- **@AccessControl.authorizationCheck:**
  - **`#CHECK:`** Perform authorization check. Syntax warning if no role is assigned
  - **`#NOT_REQUIRED:`** Like **`#CHECK`** but suppress syntax warning
  - **`#NOT_ALLOWED:`** No authorization check. Syntax warning if role is assigned

Lesson Objectives

After completing this lesson, you will be able to:

- Understand the AMDP Framework
- Define CDS Table Functions
- Read Data From CDS Table Functions

Lesson Agenda

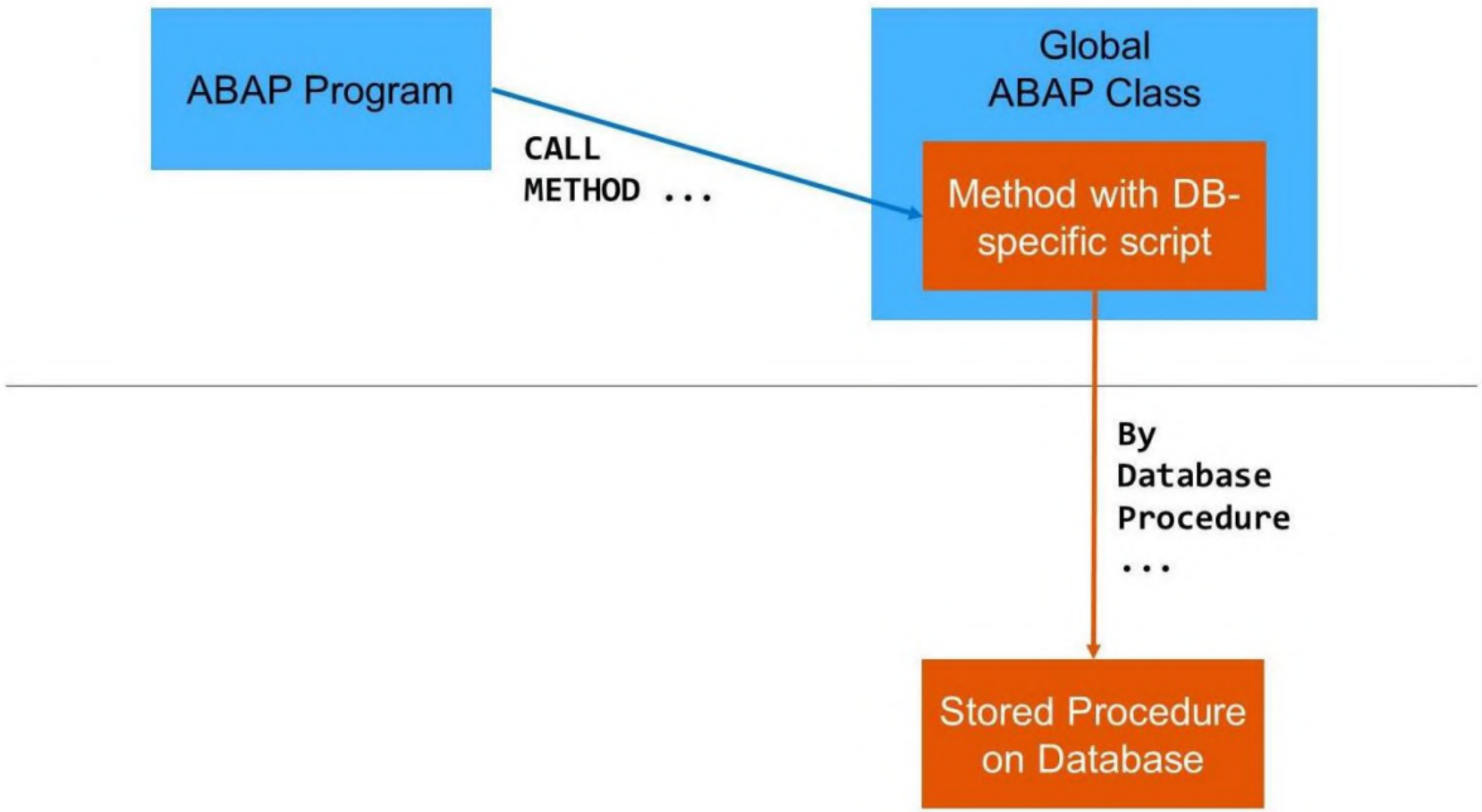
Learn About:

- Basic Features of AMDP
- AMDP Functions
- AMDP Functions for CDS Table Functions
- Template Define Table Function With Parameters
- Creating a CDS Table Function
- CDS Table Functions in ABAP SQL

Do:

- Define and Use a CDS Table Function

: Basic idea of ABAP Managed Database Procedures



: Example: Definition of an AMDP Method

```
CLASS cl_s4d430_amdp_demo DEFINITION.

PUBLIC SECTION.

   INTERFACES if_amdp_marker_hdb.

   ...

   TYPES ty_t_customers TYPE STANDARD TABLE OF ....


    CLASS-METHODS procedure
      IMPORTING
        VALUE(iv_name)    TYPE s_custname DEFAULT 'Schwarz'
        VALUE(iv_mandt)   TYPE mandt
      EXPORTING
        VALUE(et_normal) TYPE ty_t_customers
        VALUE(et_fuzzy)   TYPE ty_t_customers.

ENDCLASS.
```

Implementation of this interface mandatory

Parameters passed by value

Example: Implementation of an AMDP Method

```
CLASS cl_s4d430_amdp_demo IMPLEMENTATION.

  METHOD procedure BY DATABASE PROCEDURE
                          FOR HDB LANGUAGE SQLSCRIPT
                       OPTIONS READ-ONLY
                         USING scustom.

    et_normal = select id, name
                  from scustom
                 where mandt = :iv_mandt and name = :iv_name;

    et_fuzzy = select id, name
                 from scustom
                where mandt = :iv_mandt
                  and contains( name, :iv_name, fuzzy)
                order by score( ) desc;

  ENDMETHOD.

ENDCLASS.
```

contains() and score() are SAP HANA specific functions (not available in Open SQL or CDS DDL)

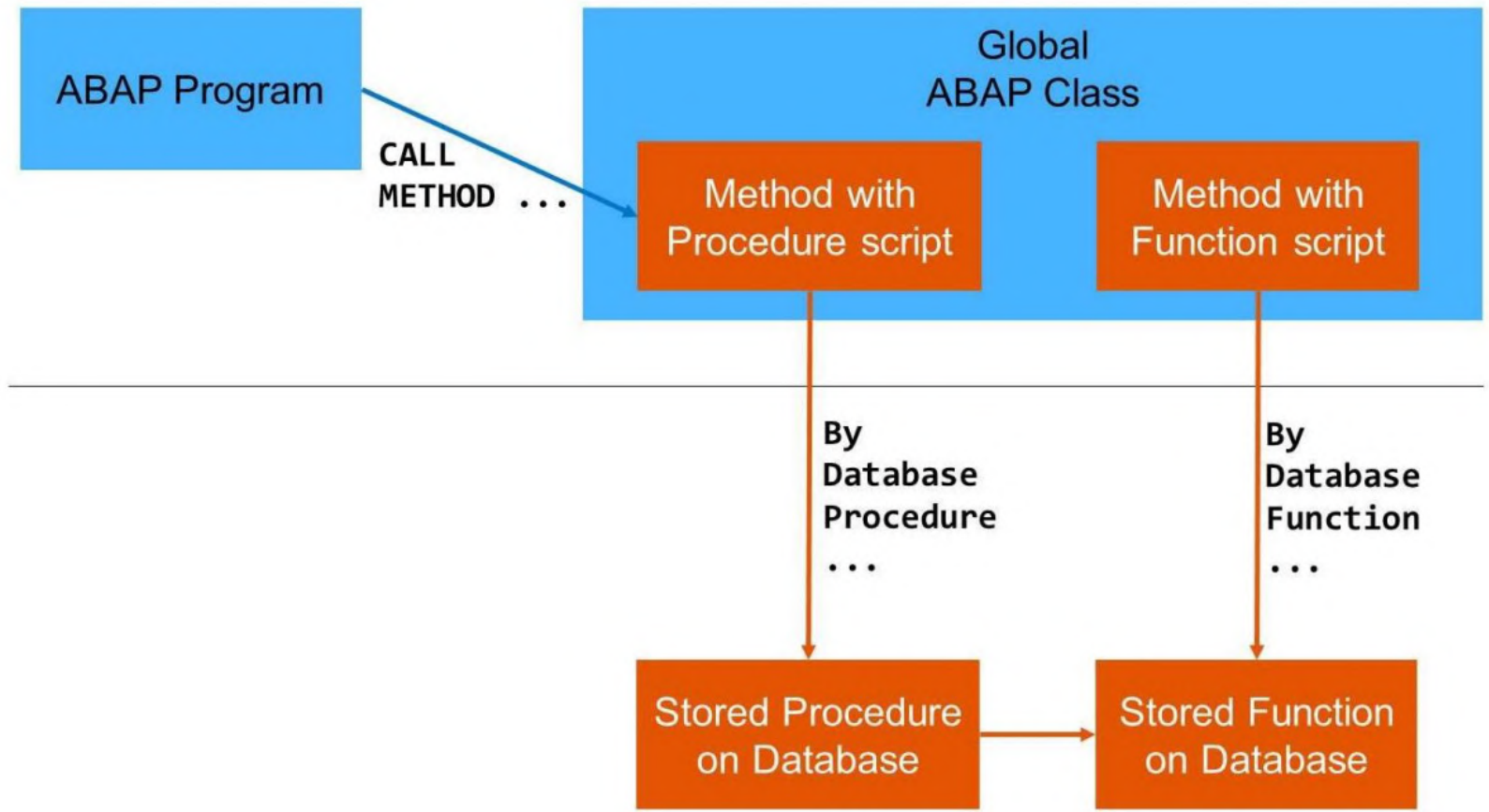
```
CLASS zamdp_demo_learning DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.
    TYPES: tt_mara TYPE TABLE OF mara.
    INTERFACES: if_amdp_marker_hdb.
    METHODS: get_matrl_detail
      IMPORTING VALUE(imp_matnr) TYPE mara-matnr
      EXPORTING VALUE(expt_mara) TYPE tt_mara.
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.


CLASS zamdp_demo_learning IMPLEMENTATION.
  METHOD get_matrl_detail BY DATABASE PROCEDURE FOR HDB LANGUAGE SQLSCRIPT OPTIONS READ-ONLY USING mara.
    expt_mara = SELECT * FROM MARA WHERE matnr = :imp_matnr and mandt = session_context( 'CLIENT' );
  ENDMETHOD.
ENDCLASS.
```

```
*&---------------------------------------------------------------------*
*& Report zreport_amdp_consumption_learn
*&---------------------------------------------------------------------*
*&
*&---------------------------------------------------------------------*
REPORT zreport_amdp_consumption_learn.
PARAMETERS: p_matnr TYPE matnr.
DATA(lr_new) = NEW zamdp_demo_learning(  ).
"Calling AMDP Method
lr_new->get_matrl_detail(
  EXPORTING
    imp_matnr = p_matnr
  IMPORTING
    expt_mara = DATA(lt_data)
).

TRY.
    CALL METHOD cl_salv_table=>factory
      IMPORTING
        r_salv_table = DATA(lr_salv)
      CHANGING
        t_table      = lt_data.

  CATCH cx_salv_msg.
ENDTRY.
lr_salv->display(  ).
```

: Prerequisites and Limitations

- **Class Definition**
  - Implementation of Interface IF_AMDP_MARKER_HDB
- **Method Definition**
  - VALUE( ) addition for all parameters
  - Only scalar or table types for parameters (no structures)
- **Supported Databases and Languages**
  - Only SAP HANA (at present)
  - Only SAP HANA SQL Script (at present)
- **Use of Dictionary Objects**
  - Transparent Tables and Dictionary Views
  - CDS Views only via their CDS SQL Views
  - Need to be listed after USING

: AMDP Functions



Example: Definition of an AMDP Function

```
CLASS cl_s4d430_amdp_demo DEFINITION.

PUBLIC SECTION.

   INTERFACES if_amdp_marker_hdb.
   ...
   TYPES ty_t_customers TYPE STANDARD TABLE OF ....

   ...

   CLASS-METHODS function
      IMPORTING
        VALUE(iv_name)      TYPE s_custname DEFAULT 'Schwarz'
        VALUE(iv_mandt)     TYPE mandt
      RETURNING
        VALUE(et_customers) TYPE ty_t_customers.

ENDCLASS.
```

Same pre-requisites as for AMDP procedures

This method cannot be called in ABAP

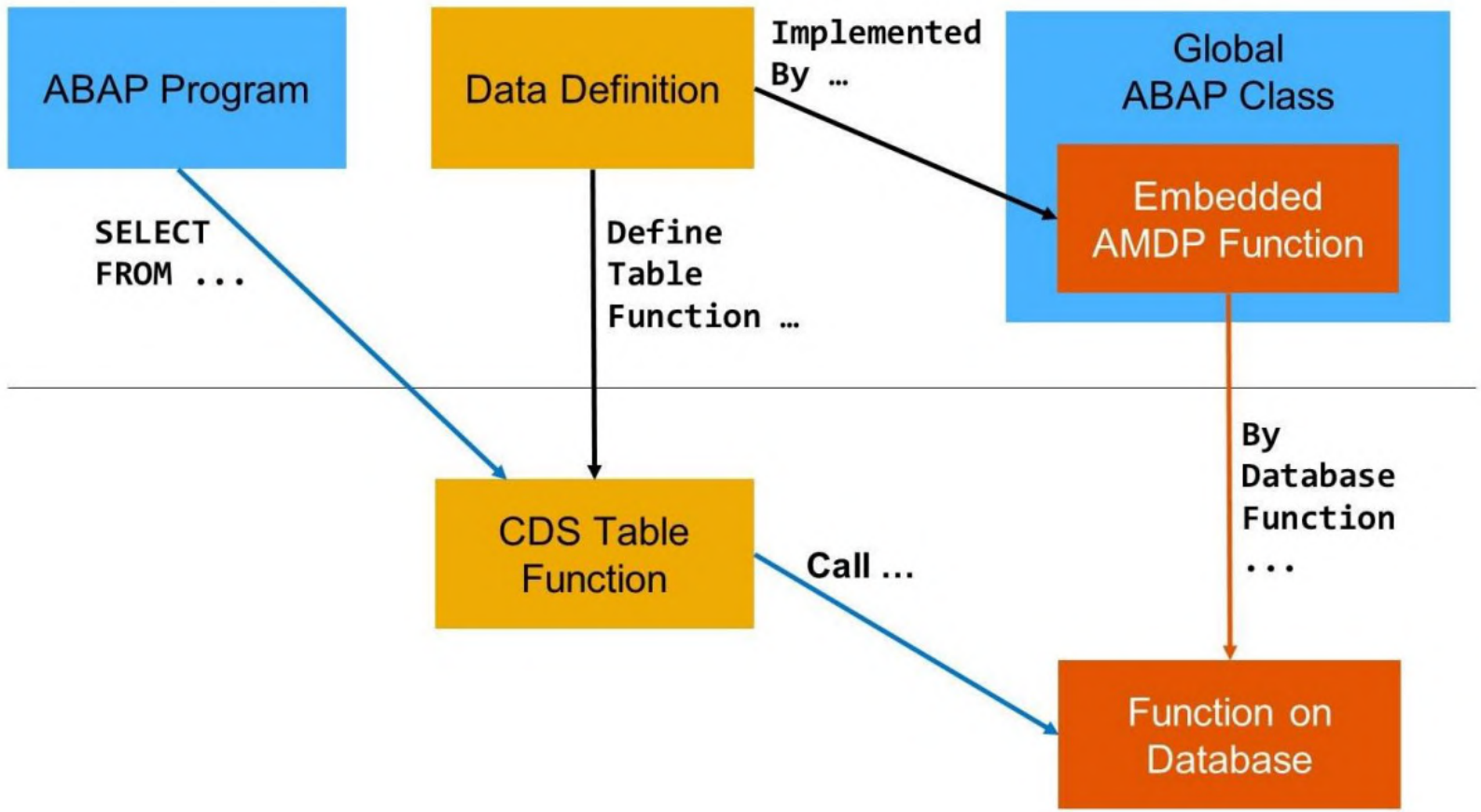
Exactly one returning parameter

Example: Implementation of an AMDP Function

```
CLASS cl_s4d430_amdp_demo IMPLEMENTATION.

  METHOD function BY DATABASE FUNCTION
                         FOR HDB LANGUAGE SQLSCRIPT
                     OPTIONS READ-ONLY
                       USING scustom.

    return select id, name
             from scustom
            where mandt = :iv_mandt
              and contains( name, :iv_name, fuzzy )
            order by score( ) desc;

  ENDMETHOD.

ENDCLASS.
```

Return value is a table with columns ID and NAME

: The Basic Idea of CDS Table Functions



Example: Definition of a CDS Table Function

```
@EndUserText.label: 'Demo: Table Function'
define table function S4d430_Table_Function
    with parameters
                name_in : s_custname

    returns {
                mandt    : mandt;
                id       : s_customer;
                name     : s_custname;
            }

   implemented by method CL_S4D430_AMDP_FOR_CDS=>FOR_TABLE_FUNC;
```

Parameters are passed to import parameters of AMDP function

Field list of table function (= line type of AMDP function‘s return value)

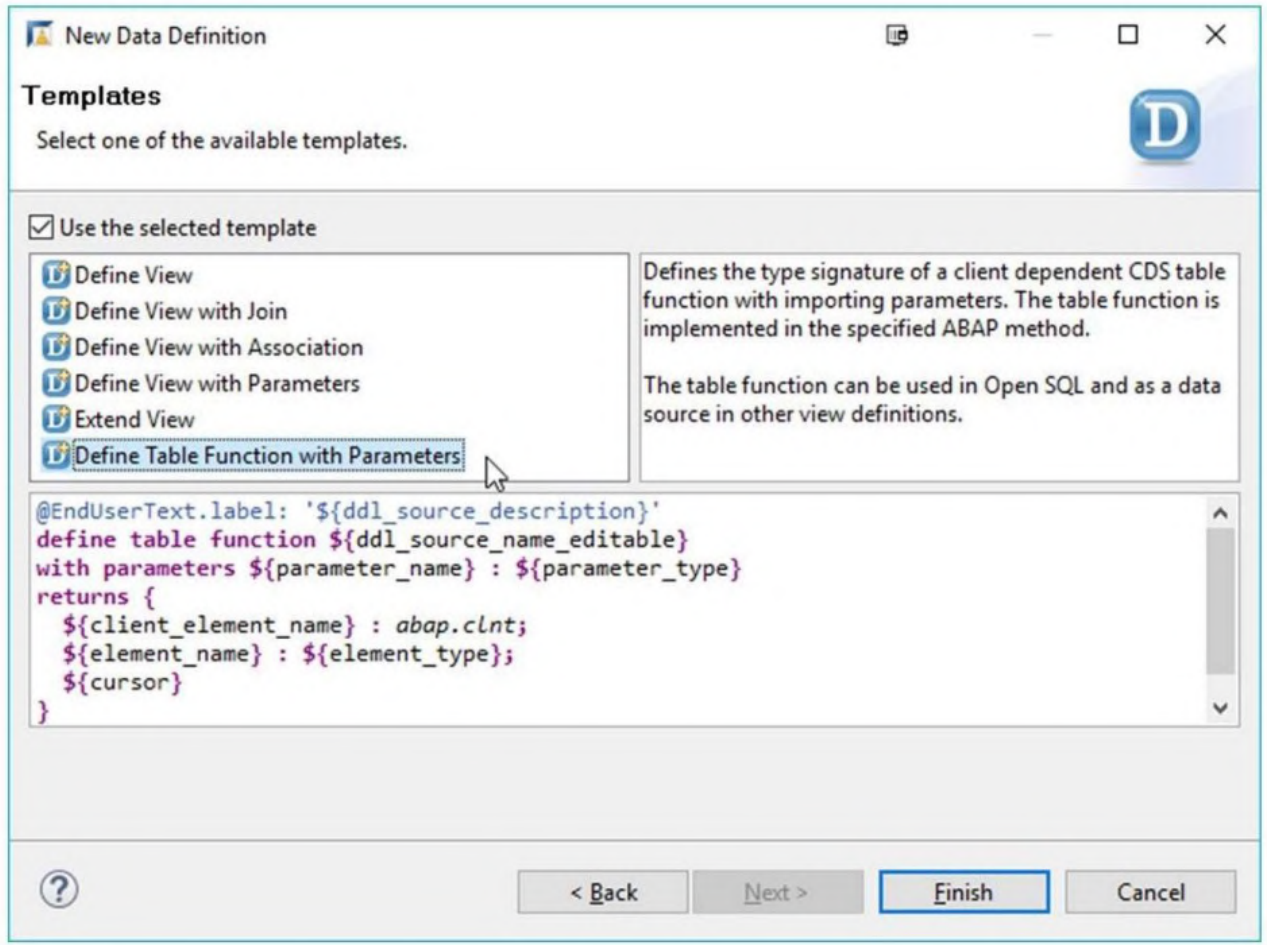
Name of AMDP class and AMDP method

Example: Definition of an AMDP Function for CDS

```
CLASS cl_s4d430_amdp_for_cds DEFINITION.

PUBLIC SECTION.

   INTERFACES if_amdp_marker_hdb.

   ...

   TYPES ty_t_customers TYPE STANDARD TABLE OF ....

   ...

     CLASS-METHODS for_table_func
        FOR TABLE FUNCTION s4d430_table_function.


ENDCLASS.
```

No parameter definition – signature is taken from CDS table function

Template Define Table Function with Parameters



CDS Table Function: Recommended Sequence

- **Create Data Definition for CDS Table Function**
  - Use Template *DefineTableFunctionWithParameters*
  - Define parameters (optional) and return value (don't forget the client field!)
  - Leave the information after IMPLEMENTED BY as it is (not checked by syntax check)
  - Activate the Data Definition
- **Create AMDP Class With AMDP Method**
  - Create a global class and implement interface IF_AMDP_MARKER_HDB
  - Define a public static method with addition FOR TABLE FUNCTION …
  - Implement the method as an AMDP function
  - Activate the global class
- **Complete the Definition of the CDS Table Function**
  - Enter the AMDP class and AMDP method after IMPLEMENTED BY
  - Activate the Data Definition
  - Test the CDS table function in data preview

Use of CDS Table Function in ABAP SQL

Use CDS Table function like any other CDS Views

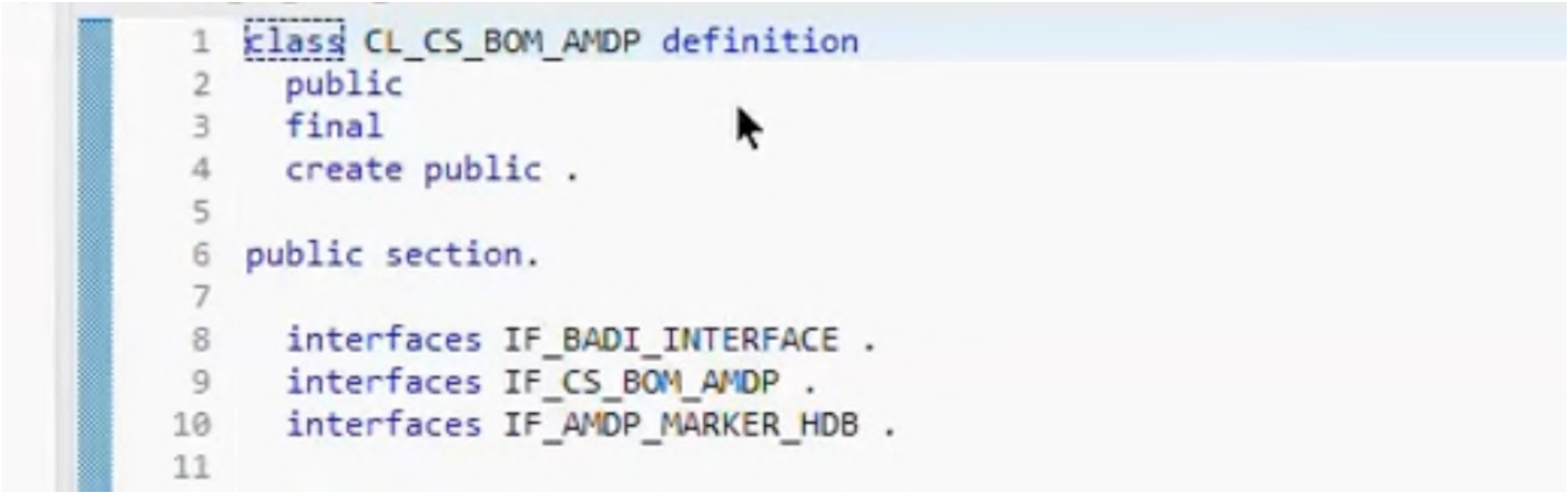
```
DATA gt_data TYPE TABLE OF s4d430_table_function.

PARAMETERS pa_nam TYPE s_custname LOWER CASE DEFAULT 'schwarz'.

SELECT FROM s4d430_table_function( name_in = @pa_nam )
     FIELDS id, name
 INTO TABLE @gt_data.
```

Possible runtime error if AMDP not supported on current system:

Category	ABAP programming error
Runtime Errors	SAPSQL_UNSUPPORTED_FEATURE
Except.	CX_SY_SQL_UNSUPPORTED_FEATURE
ABAP Program	S4D430_USE_TABLE_FUNCTION
Application Component	CA
Date and Time	28.07.2017 14:38:41

Short Text

Unsupported database extension.



Reference Class for SQLSCRIPT syntax

❖ Important Points to be considered while handling AMDP:-

Both AMDP Method and Non-AMDP methods can co-exist in the same AMDP class.

You cannot edit an AMDP class from SAP GUI. It will throw an error message.

Implementing the Interface IF_AMDP_MARKER_HDB doesn't add any interface methods, it simply flags the code as an AMDP Class.

When defining your exporting Table Types within AMDP Class; the "TYPE TABLE OF" statements are not allowed. Statements in SQLSCRIPT are terminated with ';' and not '.'.

We can perform SELECT'S on our generated Internal Tables as if they were DB tables and we can store the results in Internal Objects.

AMDP methods called inside an AMDP body have to be declared with their full names i.e. the class they belong to along with the method name and they must be addressed with double quotes and in upper case.

Debugging an AMDP:-

An AMDP can be debugged in a fully integrated way using the ADT in Eclipse.

```
et_output_data = SELECT a.mblnr, a.mjahr, a.zeile as line_item,a.header_counter as matdoc_counter,
                 a.zzgrnstat as incorrect_grn_stat, b.correct_grn_stat, c.log_grn_status
                 FROM MATDOC AS A inner join
                 (SELECT mblnr, mjahr, header_counter, zzgrnstat as correct_grn_stat FROM
                 MATDOC WHERE header_counter = 1 AND record_type = 'MDOC' AND zzgrnstat <> ''
                 AND mandt = :imp_client ) AS b ON a.mblnr = b.mblnr
                 AND a.zzgrnstat <> b.correct_grn_stat
                 LEFT OUTER JOIN
                 ( SELECT grn_num, fiscal_year, new_status AS log_grn_status FROM
                   (SELECT grn_num, fiscal_year, new_status, zdate, ztime,
                   ROW_NUMBER ( ) OVER ( PARTITION BY grn_num, fiscal_year ORDER BY zdate DESC,ztime DESC ) rownum
                   FROM ZVIM_APPR_LOG WHERE mandt = :imp_client)
                  WHERE rownum = 1 ) AS c ON a.mblnr = c.grn_num and a.mjahr = c.fiscal_year
                 WHERE a.header_counter = 0 AND a.record_type = 'MDOC'
                 AND a.mandt = :imp_client ORDER BY a.mjahr DESC,a.mblnr ASC;
```

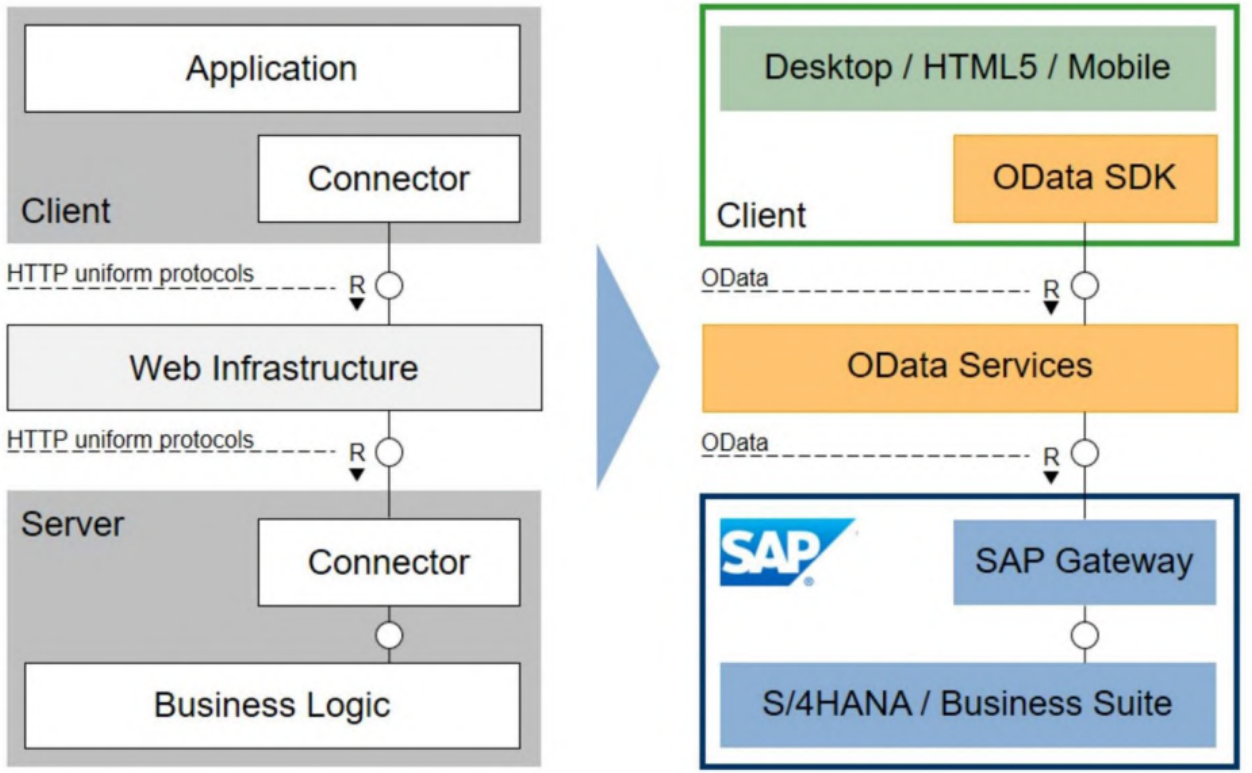
Example showing SQLSCRIPT flexibility:

Understanding OData

- Explain the OData standard

The OData Protocol

Exchanging data between multiple different server systems and end-user items needs a standardized exchange infrastructure. One of the possible solutions is the OData protocol (OData).



Architecture of the World Wide Web

The OData Protocol

SAP annotations enhance OData by adding additional information, like dictionary labels.

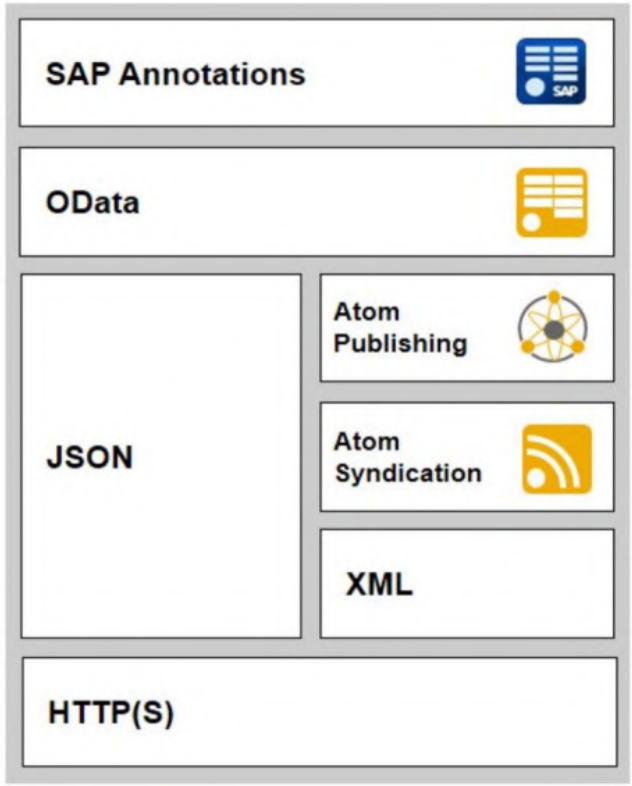
OData defines the standard to transfer data using already established technologies. It is also called “ODBC for the web”.

JSON (Java Script Object Notation) defines the data format for the transferred data.

XML in combination with Atom Publishing and Atom Syndication is the alternative to JSON supported also in older clients.

HTTP(S) is the network protocol for communication.

OData is an open standard originally developed by Microsoft but now managed by the Oasis Organisation. It is based on the Atom Publishing and Atom Syndication standards, which in turn are based on XML and HTTP(S). The objective of the OData protocol is to provide a vendor-neutral web-based API that fully complies with the design principles of Representational State Transfer (REST). OData is also extensible. This allows SAP to supplement the data types used by OData with extra information from the ABAP Data Dictionary.



Atom and JSON format comparison

Because OData uses the HTTP protocol, any web browser can be used to start exploring OData. As of today, OData supports the Atom and the JSON formats.

JSON has significantly less protocol overhead than the Atom Publishing protocol which is used by default. JSON can also easily be consumed by JavaScript and thus by SAPUI5.

Atom

JSON

CRUD Operations

One of the main features of OData is that it uses the existing HTTP verbs GET, PUT, POST and DELETE against addressable resources identified in the URI. Conceptually, OData is a way of performing database-style create, read, update and delete operations on resources by using HTTP verbs.

Operation on resource	HTTP verb
Create	POST
Read	GET
Update	PUT
Delete	DELETE

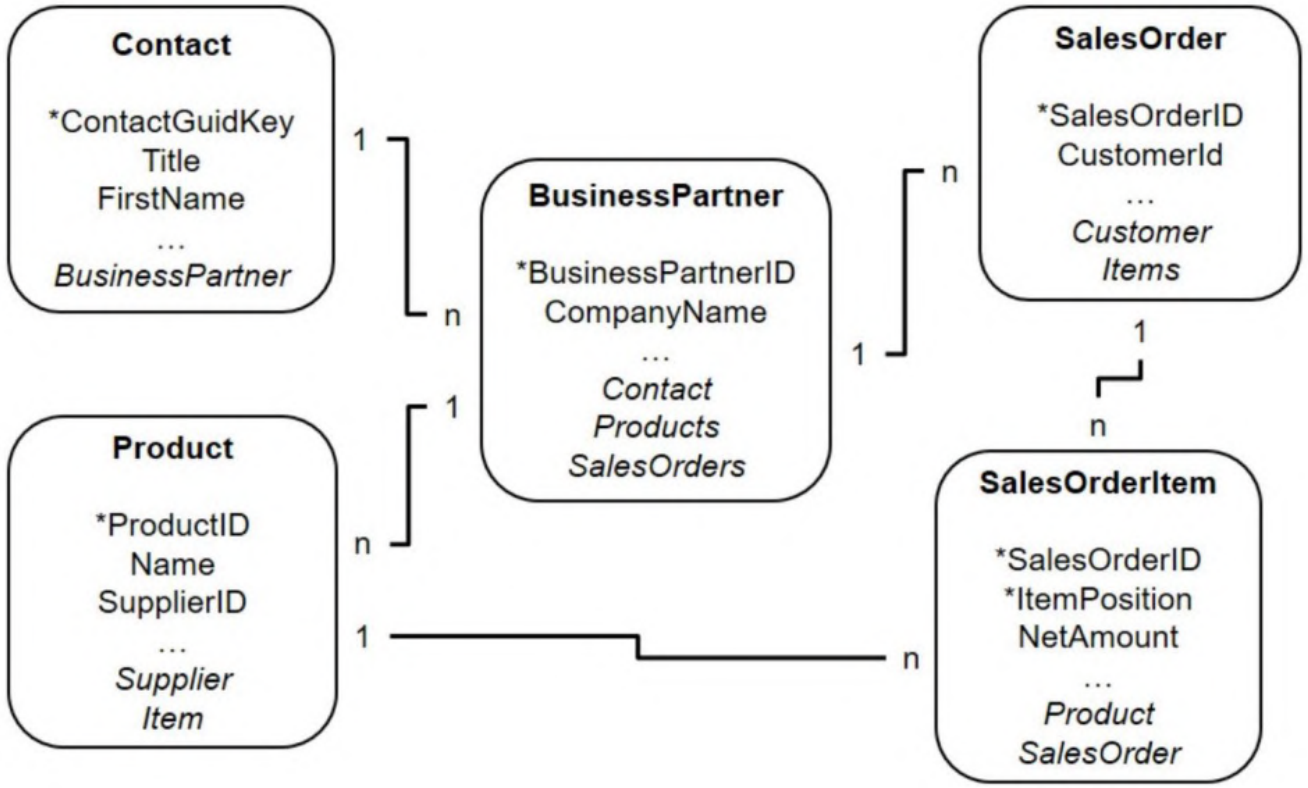
Service Document

To consume an OData service for reading, you just need a browser and the OData service URI. This leads to the service document. To get data from the service, add the name of an entity of the service to the base URI. You will get a list of entities of that type, which could be the content of a database table.


```
▼<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="de"
 xml:base="http://                                /sap/opu/odata/IWBEP/GWSAMPLE_BASIC/">
 ▼<app:workspace>
    <atom:title type="text">Data</atom:title>
  ▼<app:collection sap:content-version="1" href="BusinessPartnerSet">
     <atom:title type="text">BusinessPartnerSet</atom:title>
     <sap:member-title>BusinessPartner</sap:member-title>
   </app:collection>
  ▼<app:collection sap:content-version="1" href="ProductSet">
     <atom:title type="text">ProductSet</atom:title>
     <sap:member-title>Product</sap:member-title>
   </app:collection>
  ▼<app:collection sap:updatable="false" sap:content-version="1" href="SalesOrderSet">
     <atom:title type="text">SalesOrderSet</atom:title>
     <sap:member-title>SalesOrder</sap:member-title>
   </app:collection>
  ▼<app:collection sap:content-version="1" href="SalesOrderLineItemSet">
     <atom:title type="text">SalesOrderLineItemSet</atom:title>
     <sap:member-title>SalesOrderLineItem</sap:member-title>
   </app:collection>
  ▼<app:collection sap:content-version="1" href="ContactSet">
     <atom:title type="text">ContactSet</atom:title>
     <sap:member-title>Contact</sap:member-title>
   </app:collection>
 </app:workspace>
 <atom:link rel="self"
 href="http://                                /sap/opu/odata/IWBEP/GWSAMPLE_BASIC/"/>
 <atom:link rel="latest-version"
 href="http://                                /sap/opu/odata/IWBEP/GWSAMPLE_BASIC/"/>
</app:service>
```

Entity Data Model

The entities of an OData service are defined in an entity data model (EDM). Entity types define properties and navigations to other entity types. These associations define relation constraints based on key properties of the entity types. A navigation property is then used to actively navigate between entities during runtime. For every entity type at least one entity set is defined. These are shown in the service document and used to request data from the OData service.



Service Metadata Document


```
▼<edmx:Edmx xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns:sap="http://www.sap.com/Protocols/SAPData" Version="1.0">
  ▼<edmx:DataServices m:DataServiceVersion="2.0">
    ▼<Schema xmlns="http://schemas.microsoft.com/ado/2008/09/edm" Namespace="/IWBEP/GWSAMPLE_BASIC" xml:lang="de" sap:schema-version="1">
      ▶<EntityType Name="BusinessPartner" sap:content-version="1">...</EntityType>
      ▼<EntityType Name="Product" sap:content-version="1">
        ▼<Key>
          <PropertyRef Name="ProductID"/>
        </Key>
        <Property Name="ProductID" Type="Edm.String" Nullable="false" MaxLength="10" sap:label="Produkt-ID" sap:updatable="false"/>
        <Property Name="TypeCode" Type="Edm.String" Nullable="false" MaxLength="2" sap:label="Typcode"/>
        <Property Name="Category" Type="Edm.String" Nullable="false" MaxLength="40" sap:label="Kategorie"/>
        <Property Name="Name" Type="Edm.String" Nullable="false" MaxLength="255" sap:sortable="false" sap:filterable="false"/>
        <Property Name="NameLanguage" Type="Edm.String" MaxLength="2" sap:label="Sprache" sap:creatable="false" sap:updatable="false"
        sap:sortable="false" sap:filterable="false"/>
        <Property Name="Description" Type="Edm.String" MaxLength="255" sap:sortable="false" sap:filterable="false"/>
        <Property Name="DescriptionLanguage" Type="Edm.String" MaxLength="2" sap:label="Sprache" sap:creatable="false" sap:updatable="false"
        sap:sortable="false" sap:filterable="false"/>
        <Property Name="SupplierID" Type="Edm.String" Nullable="false" MaxLength="10" sap:label="Geschäftspartner-ID"/>
        <Property Name="SupplierName" Type="Edm.String" MaxLength="80" sap:label="Firma" sap:creatable="false" sap:updatable="false"/>
        <Property Name="TaxTarifCode" Type="Edm.Byte" Nullable="false" sap:label="Steuertarifcode"/>
        <Property Name="MeasureUnit" Type="Edm.String" Nullable="false" MaxLength="3" sap:label="Maßeinheit" sap:semantics="unit-of-measure"/>
        <Property Name="WeightMeasure" Type="Edm.Decimal" Precision="13" Scale="3" sap:unit="WeightUnit" sap:label="Gewicht"/>
        <Property Name="WeightUnit" Type="Edm.String" MaxLength="3" sap:label="Maßeinheit" sap:semantics="unit-of-measure"/>
        <Property Name="CurrencyCode" Type="Edm.String" Nullable="false" MaxLength="5" sap:label="Währungscode" sap:semantics="currency-code"/>
        <Property Name="Price" Type="Edm.Decimal" Precision="16" Scale="3" sap:unit="CurrencyCode" sap:label="Preis"/>
        <Property Name="Width" Type="Edm.Decimal" Precision="13" Scale="3" sap:unit="DimUnit" sap:label="Maßangaben"/>
        <Property Name="Depth" Type="Edm.Decimal" Precision="13" Scale="3" sap:unit="DimUnit" sap:label="Maßangaben"/>
        <Property Name="Height" Type="Edm.Decimal" Precision="13" Scale="3" sap:unit="DimUnit" sap:label="Maßangaben"/>
        <Property Name="DimUnit" Type="Edm.String" MaxLength="3" sap:label="Maßeinheit" sap:semantics="unit-of-measure"/>
        <Property Name="CreatedAt" Type="Edm.DateTime" Precision="7" sap:label="Zeitstempel" sap:creatable="false" sap:updatable="false"/>
        <Property Name="ChangedAt" Type="Edm.DateTime" Precision="7" ConcurrencyMode="Fixed" sap:label="Zeitstempel" sap:creatable="false"
        sap:updatable="false"/>
        <NavigationProperty Name="ToSalesOrderLineItems" Relationship="/IWBEP/GWSAMPLE_BASIC.Assoc_Product_SalesOrderLineItems"
        FromRole="FromRole_Assoc_Product_SalesOrderLineItems" ToRole="ToRole_Assoc_Product_SalesOrderLineItems"/>
        <NavigationProperty Name="ToSupplier" Relationship="/IWBEP/GWSAMPLE_BASIC.Assoc_BusinessPartner_Products"
        FromRole="ToRole_Assoc_BusinessPartner_Products" ToRole="FromRole_Assoc_BusinessPartner_Products"/>
      </EntityType>
      ▶<EntityType Name="SalesOrder" sap:content-version="1">...</EntityType>
      ▶<EntityType Name="SalesOrderLineItem" sap:content-version="1">...</EntityType>
      ▶<EntityType Name="Contact" sap:content-version="1">...</EntityType>
      ...
```

Request using URI Options

An OData request consists of server address, service URI, entity set, and additional query options. There are many query options with many combinations possible.

Server Address	http(s)://<server>:<port>
+	
Service URI	/sap/opu/odata/IWBEP/ GWSAMPLE_BASIC
+	
Entity Set	/ProductSet
+	
Query Options	?$format=json
	&$top=3
	&$select=ProductID, Category,Name, SupplierName, CurrencyCode,Price

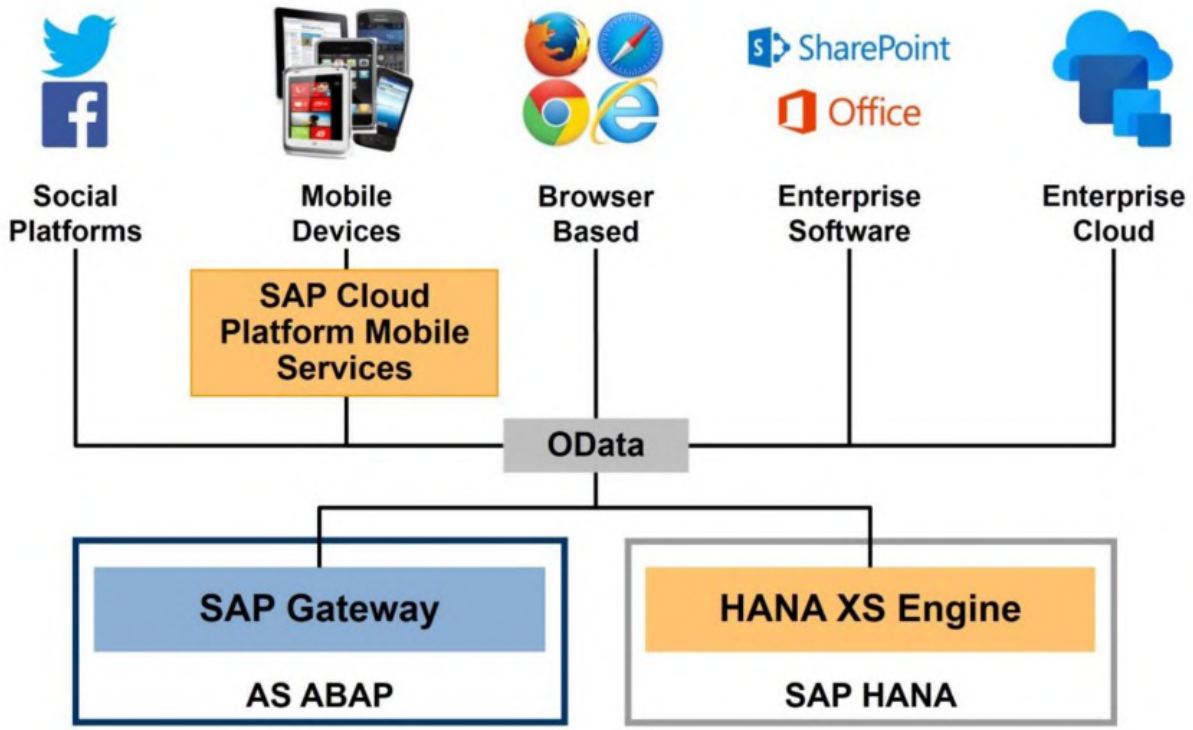
=

Exploring SAP Gateway

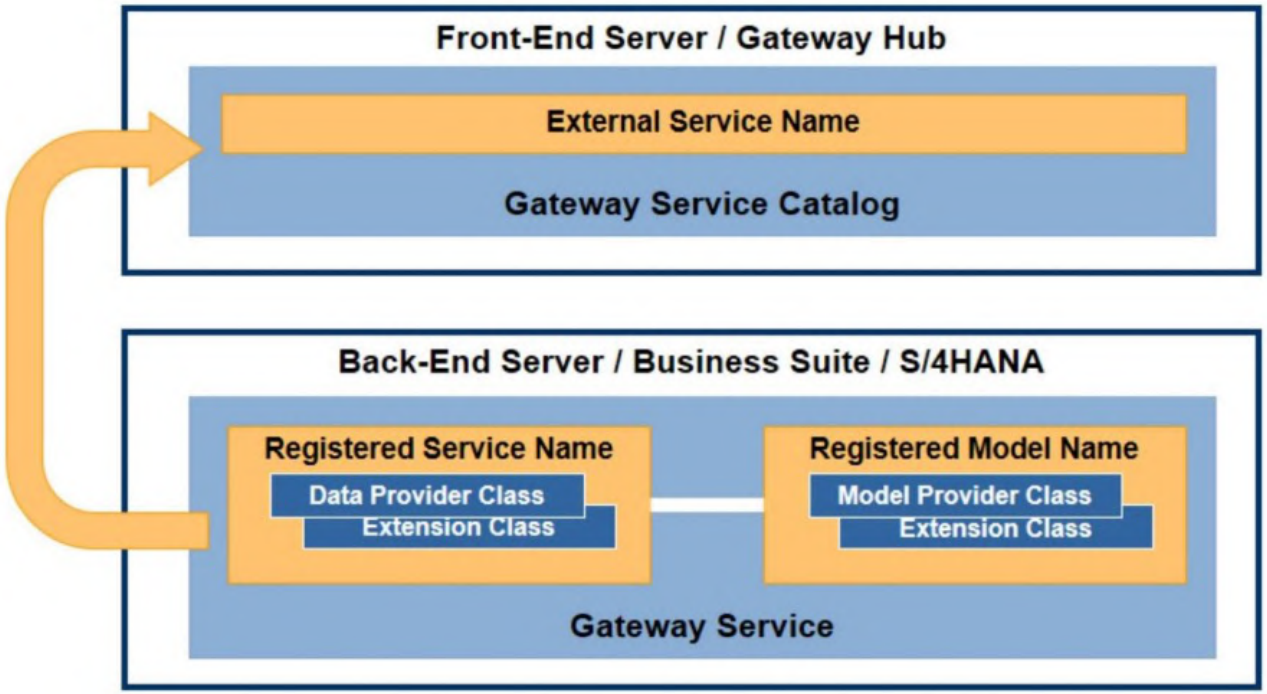
- Describe SAP Gateway service architecture
- Identify SAP Gateway service components
- Operate central SAP Gateway transactions

SAP Gateway – Introduction

SAP Gateway (formerly known as SAP NetWeaver Gateway) provides a single entry point to access business data of ABAP-based systems such as the SAP Business Suite or SAP S/4HANA. The SAP HANA eXtended Services (XS) Engine has the same role in SAP HANA. This business data can be shared among multiple environments and platforms.

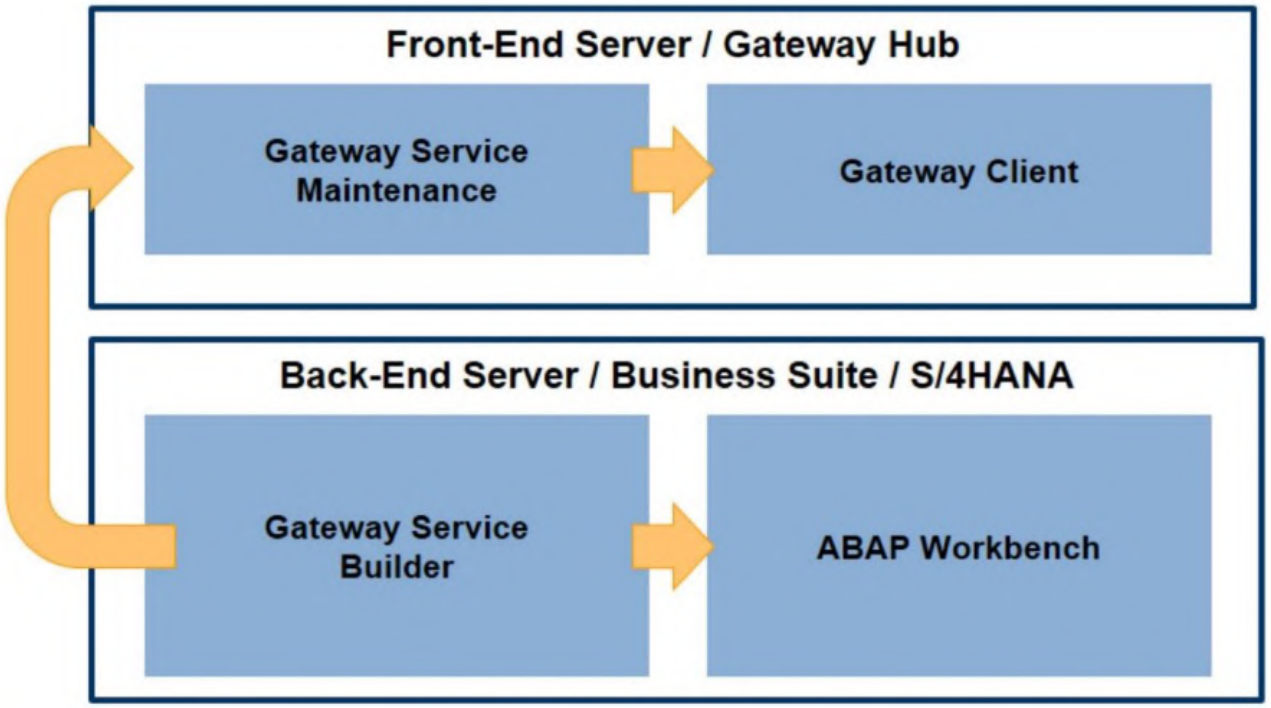


SAP Gateway – Architecture



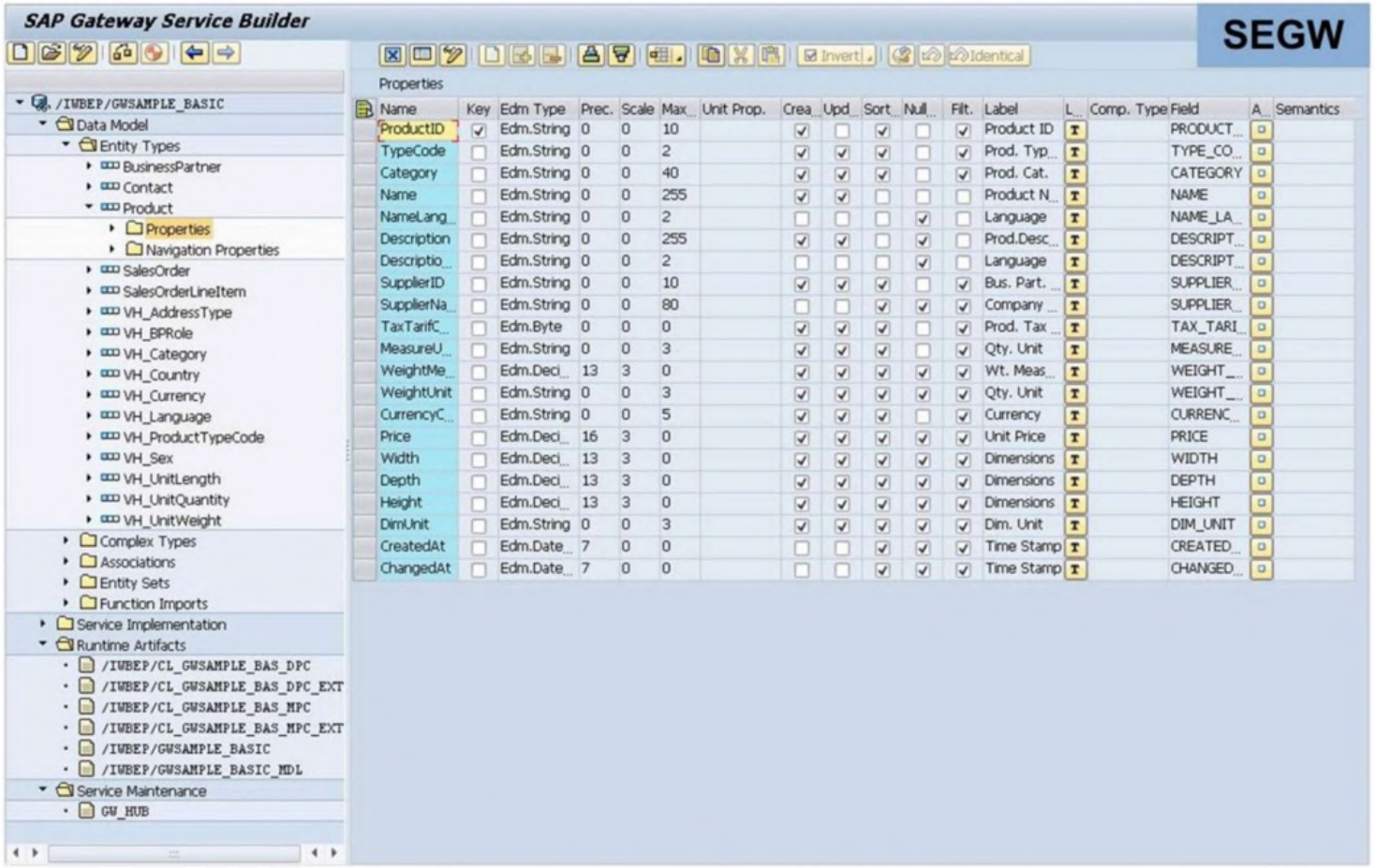
SAP Gateway Server Architecture

SAP Gateway – Architecture



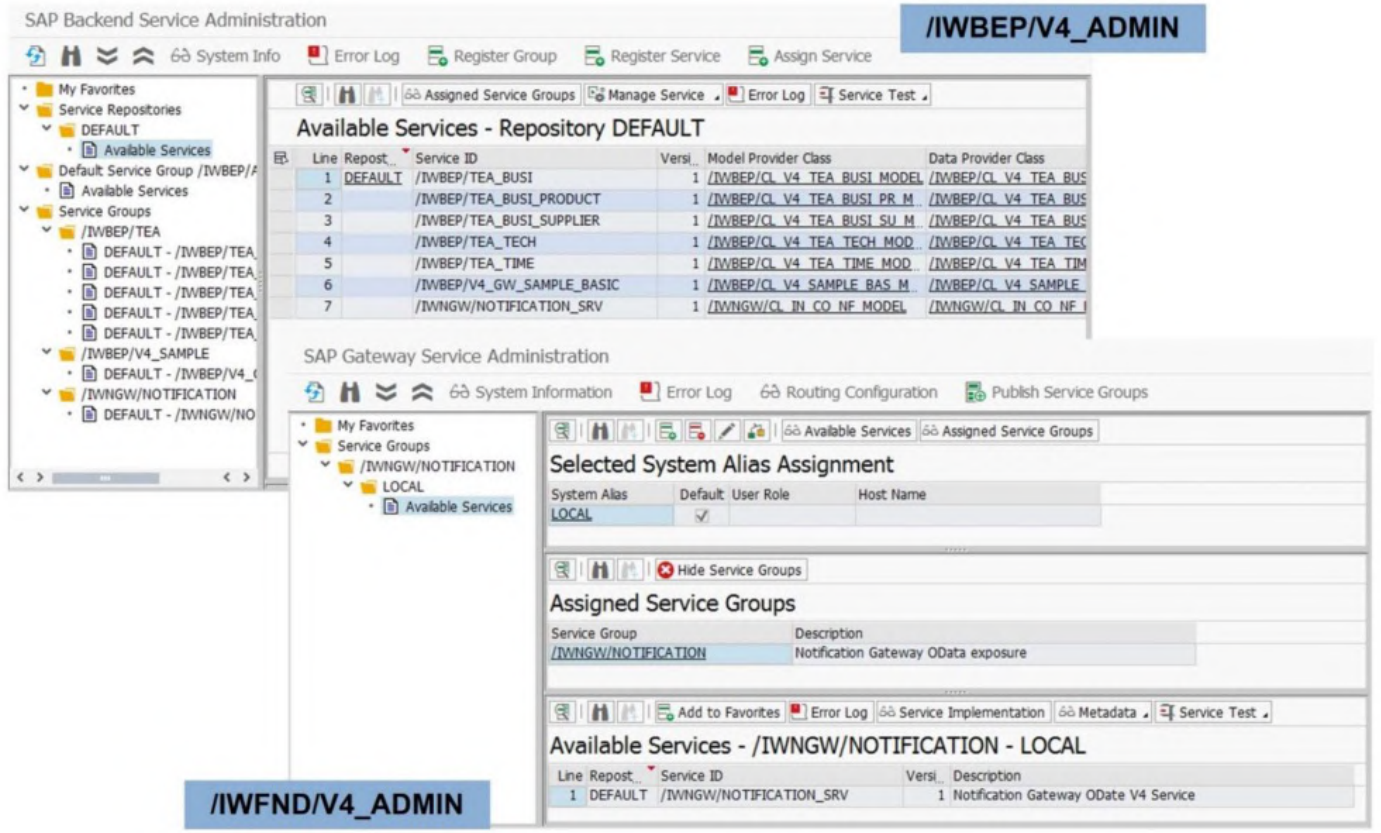
SAP Gateway ABAP Transactions

SAP Gateway – Architecture



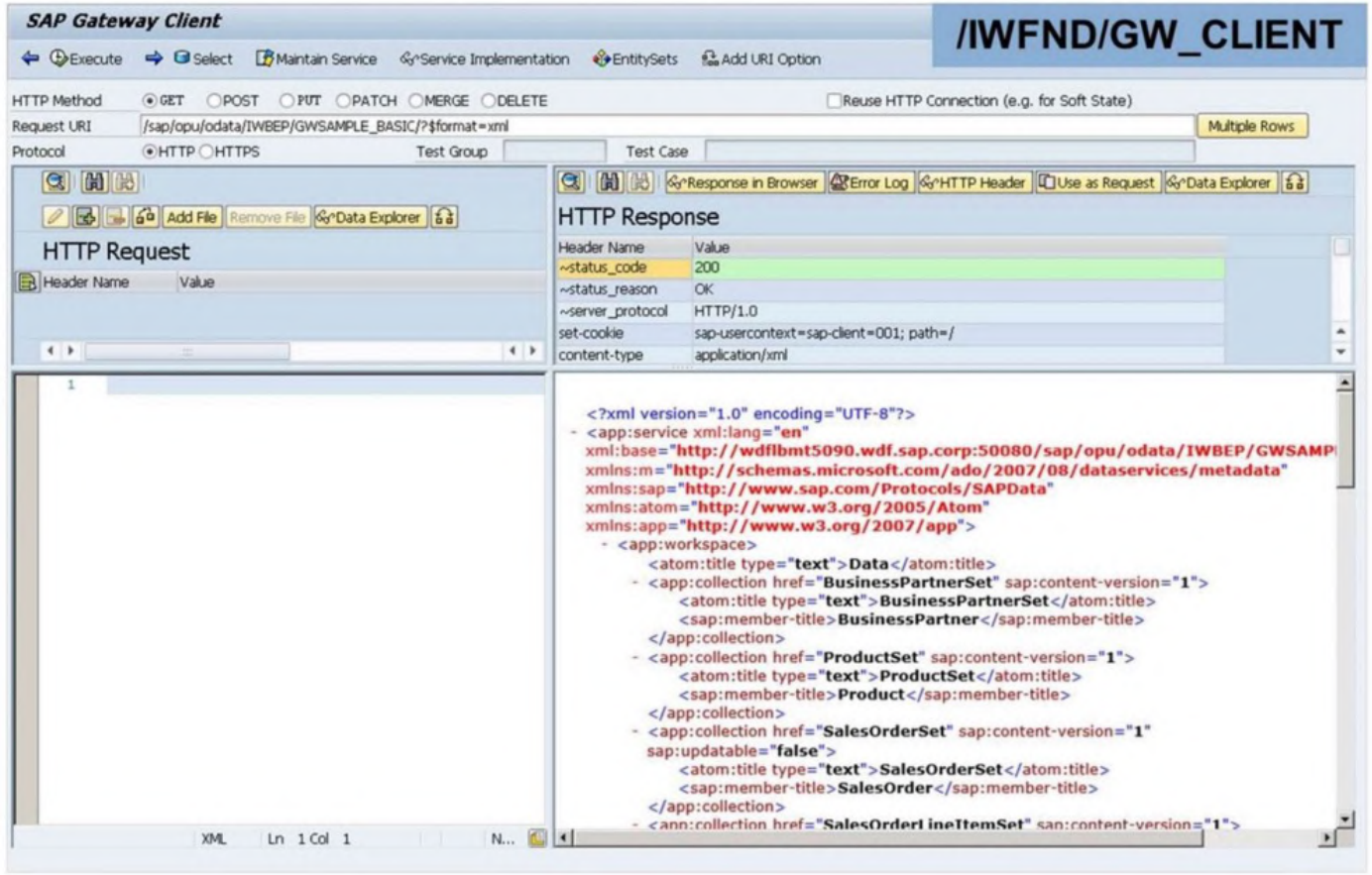
SAP Gateway Service Builder

SAP Gateway – Architecture



SAP Gateway Service Administration OData

SAP Gateway – Architecture



SAP Gateway Client

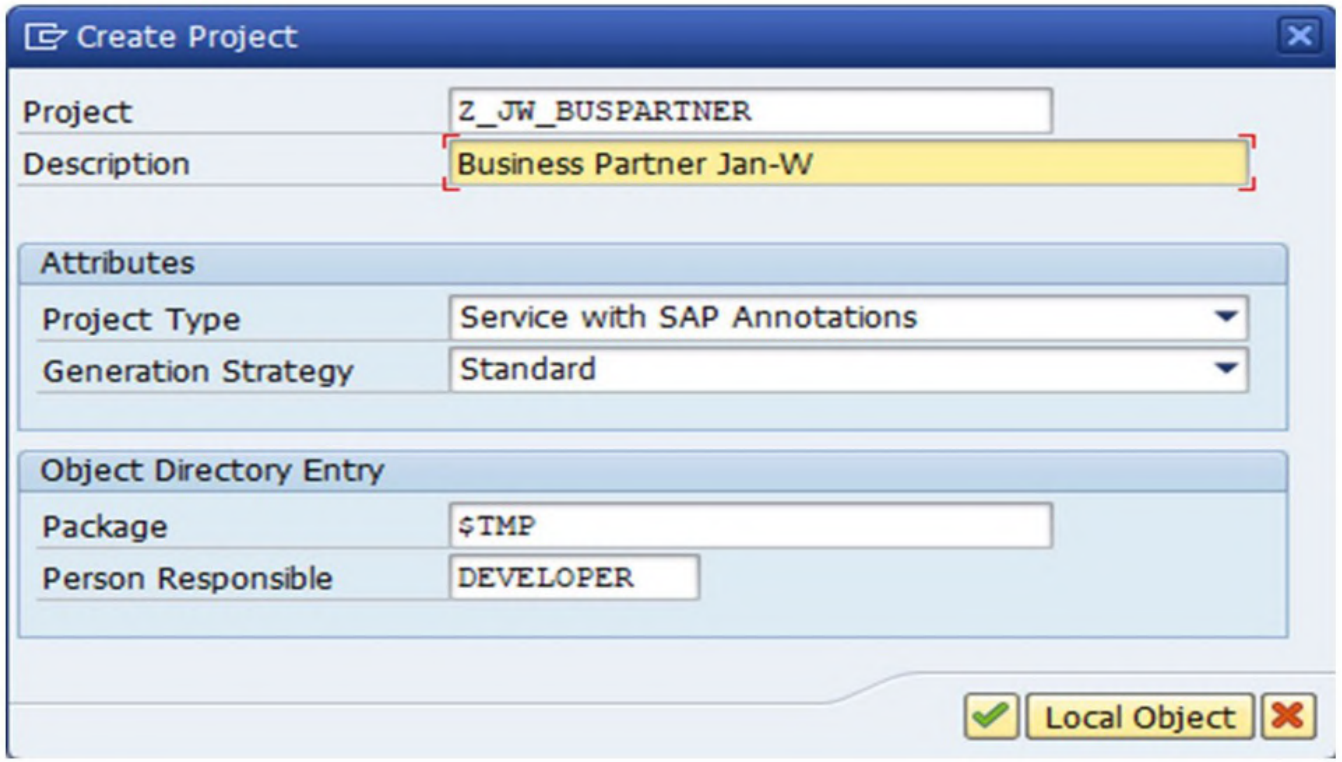
Lesson 4: Odata Services

- Defining Odata Services
- Testing Odata Services

Defining OData Services

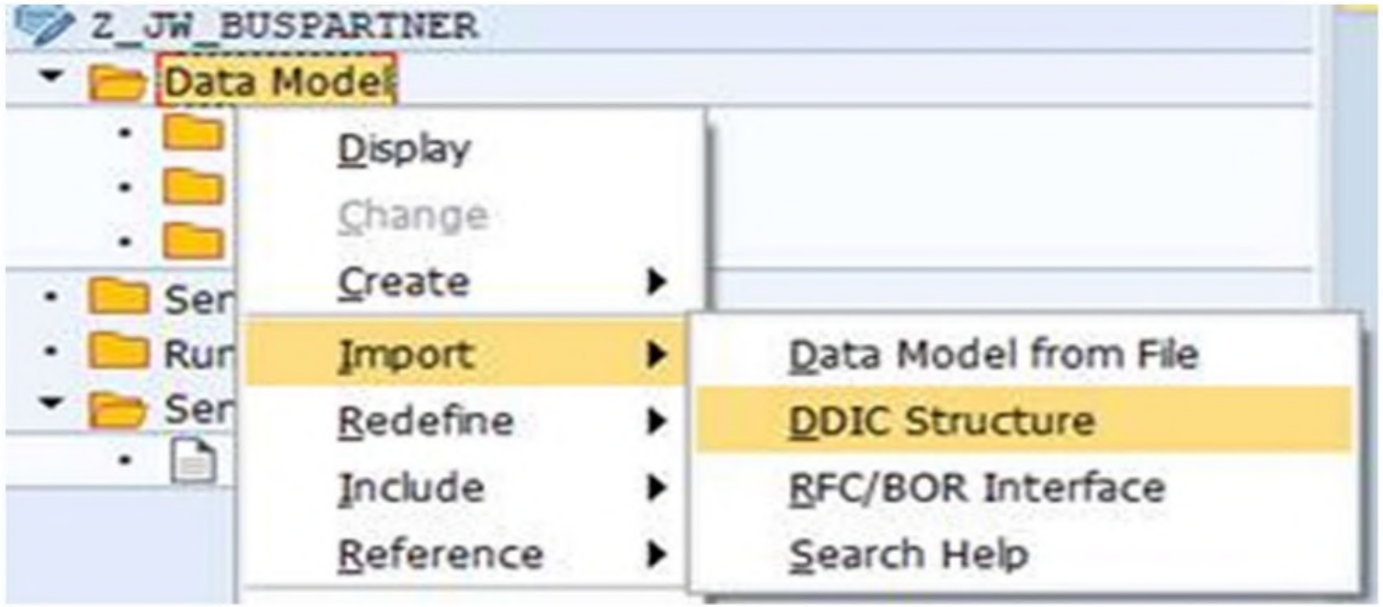
Step 1

In transaction ‘SEGW’ you can define your OData service. Let’s start by creating a read service (HTTP “GET” operation) for the Business Partner entity. Click on create and fill in the requested parameters. An empty OData project appears.



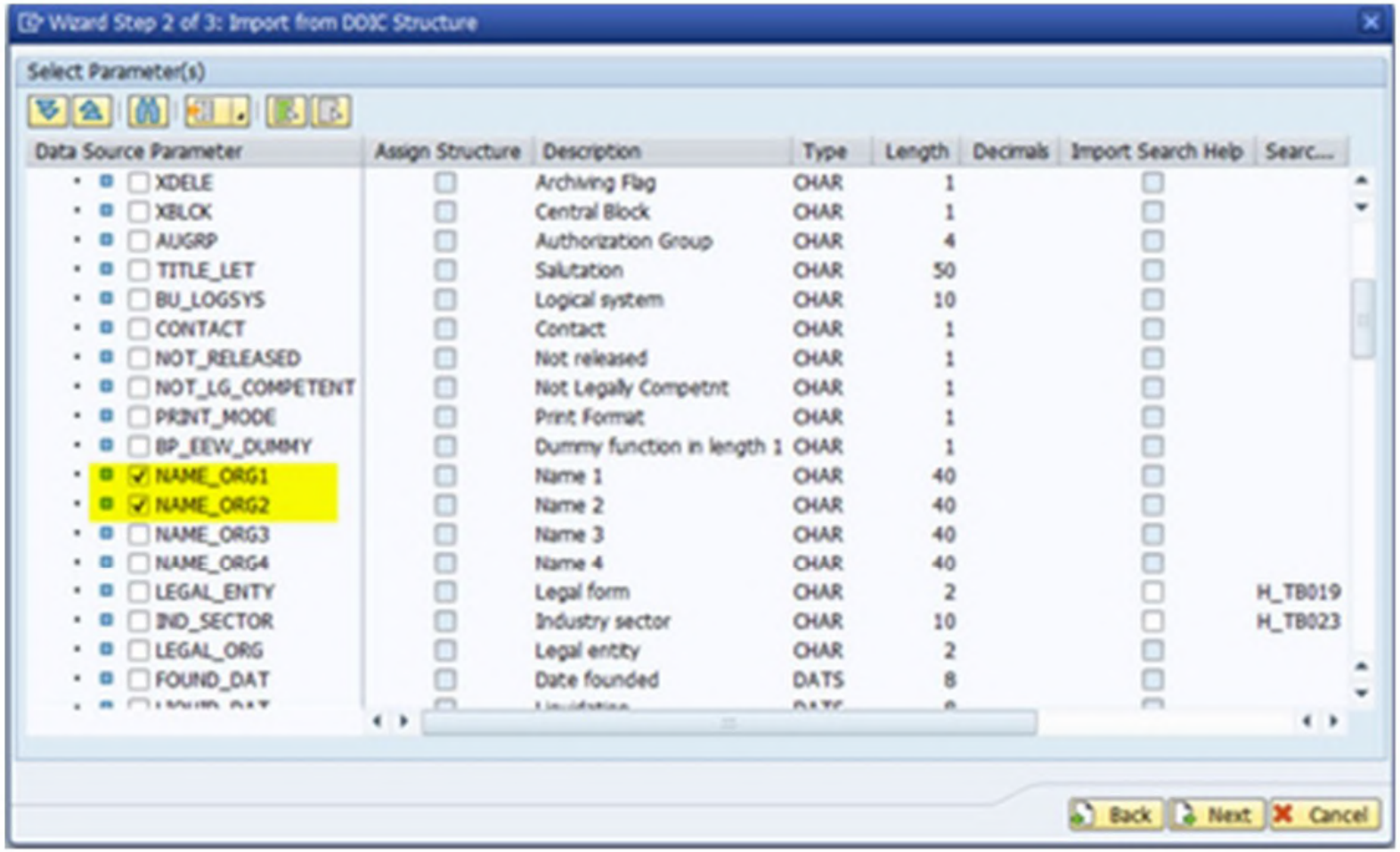
Defining OData Services

Step 2



Defining OData Services

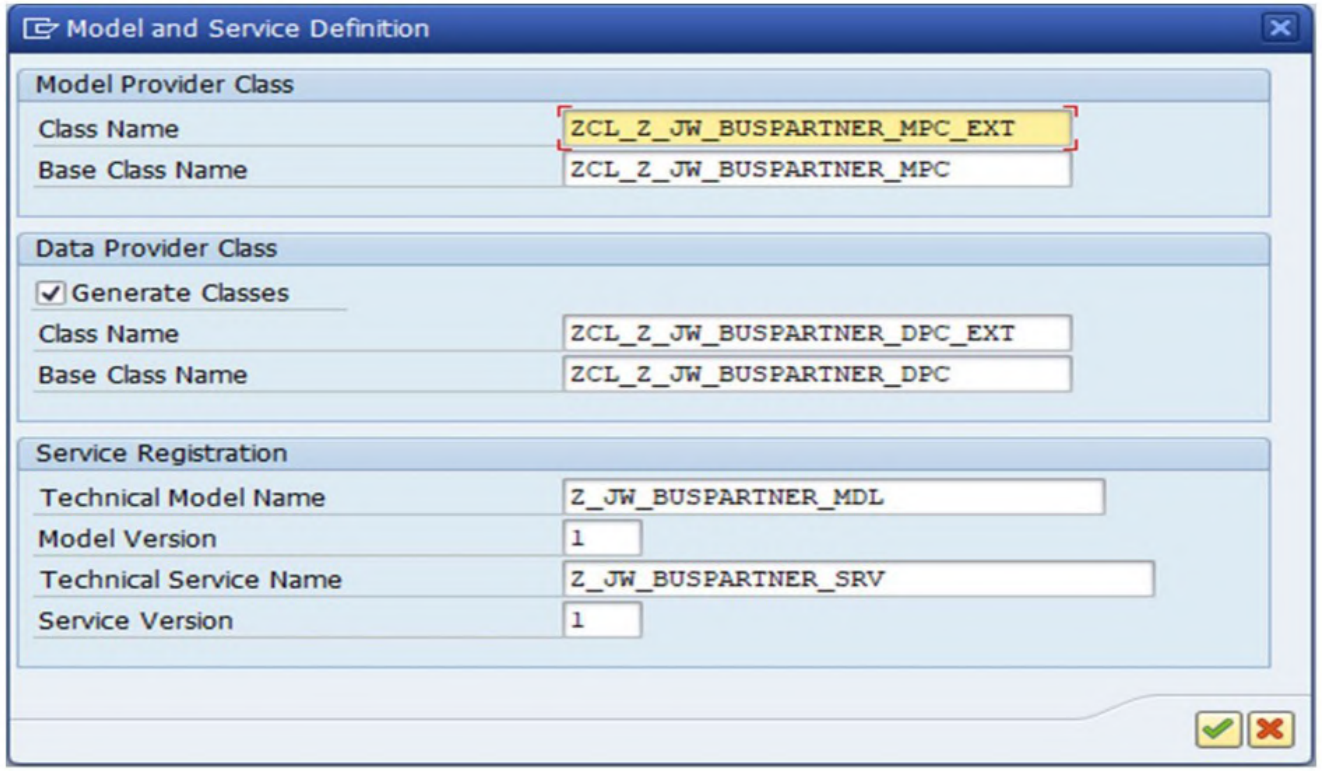
Step 3



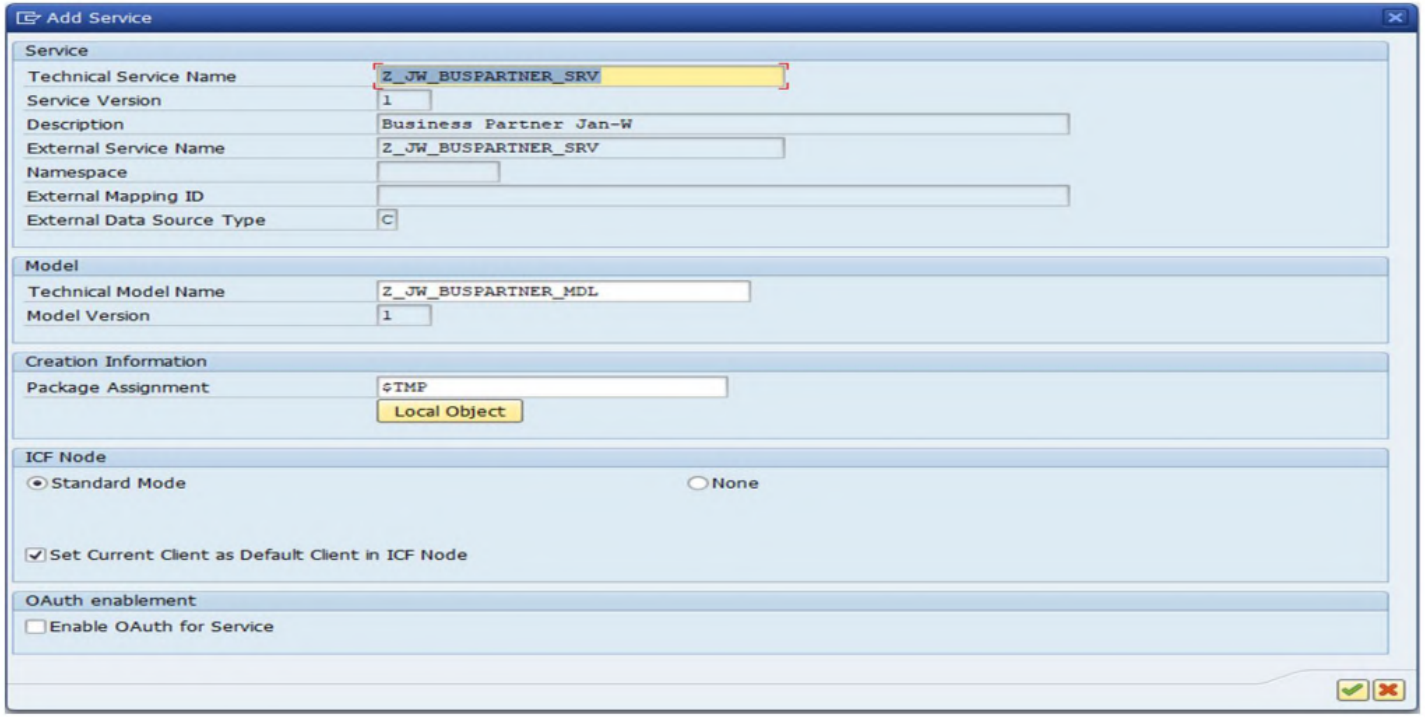
Defining OData Services

Step 4

After generating the first part of the OData service is created! Best practice is to leave the suggested names as they are, but you can change the names of the classes. Later on, we'll use the classes with the "*_DPC_EXT*" and "*_MPC_EXT*" for implementing our own logic.



Registering it through SICF



OData with CDS Views

CDS Views

Development flow:

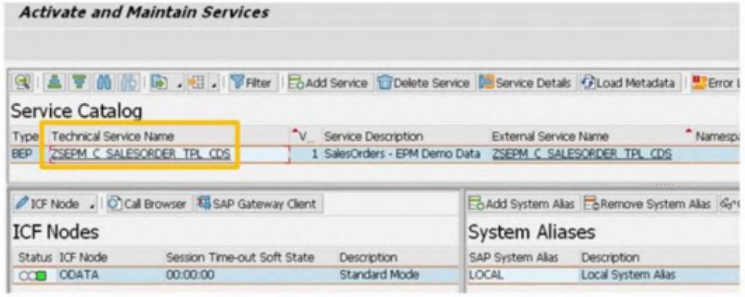
1. Create CDS view (for example Zsepm_C_Salesorder_Tpl) with annotation:

@OData.publish: true

2. Service is generated in the SAP Business Suite back-end system.
(Zsepm_C_Salesorder_Tpl_CDS)

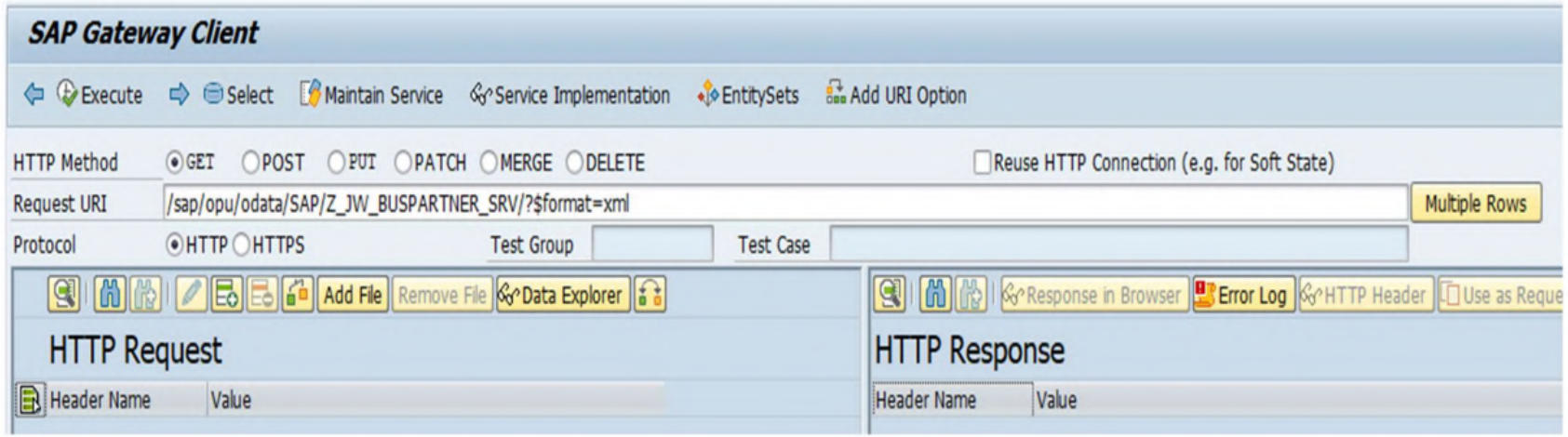
3. Service can be published in the SAP Gateway Server.

```
@AbapCatalog.sqlViewName: 'ZSEPM_ISOE_TPL'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'SalesOrders - EPM Demo Data'
@OData.publish: true
define view Zsepm_C_Salesorder_Tpl
as select from SEPM_I_SalesOrder_E
{ …
```



Use transaction code **/n/iwfnd/maint_service** to publish and maintain service

Testing OData services in the SAP GUI



Lesson Objectives

After completing this lesson, you will be able to:

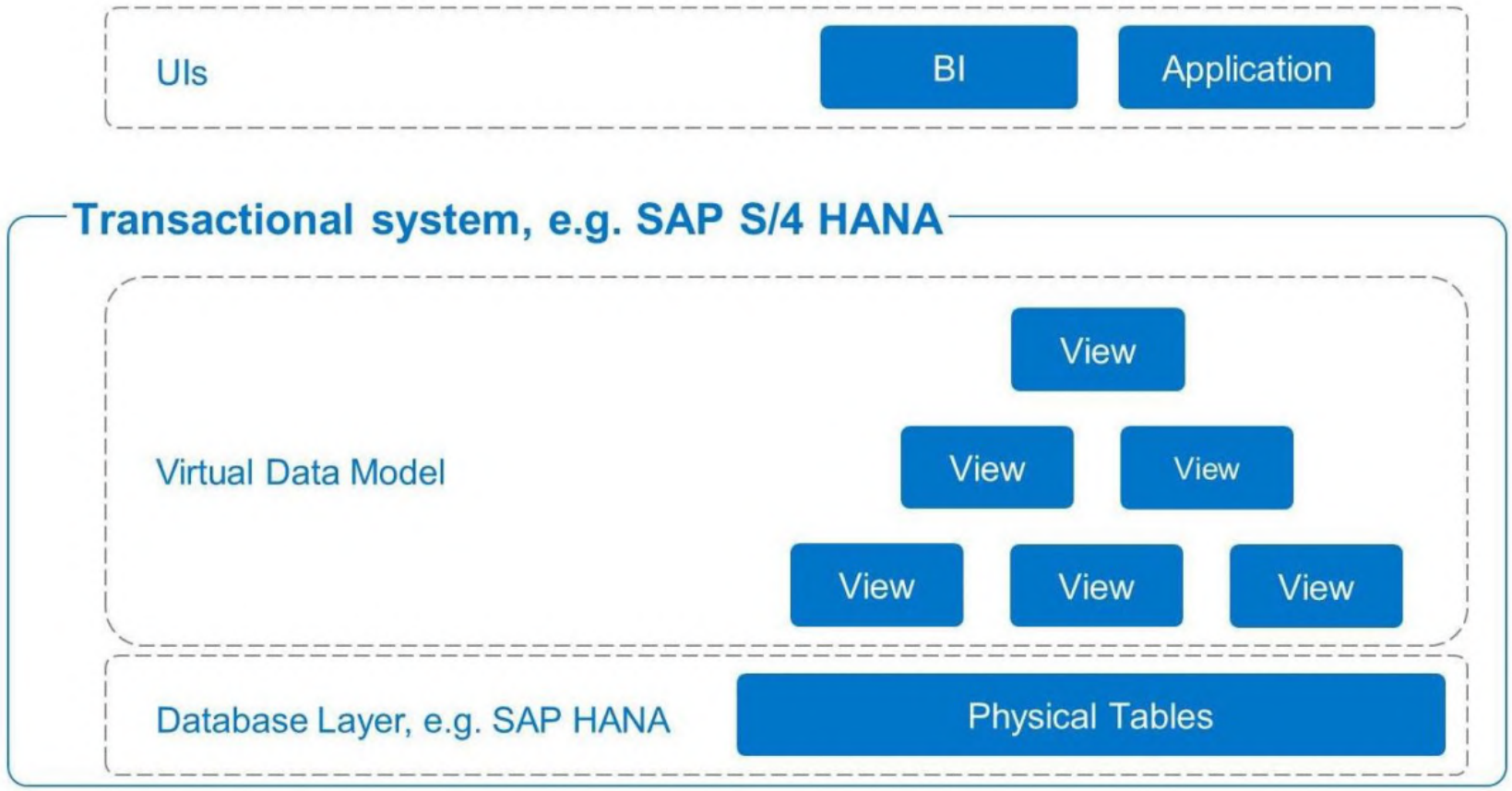
- Understand Virtual Data Models (VDM)

Lesson Agenda

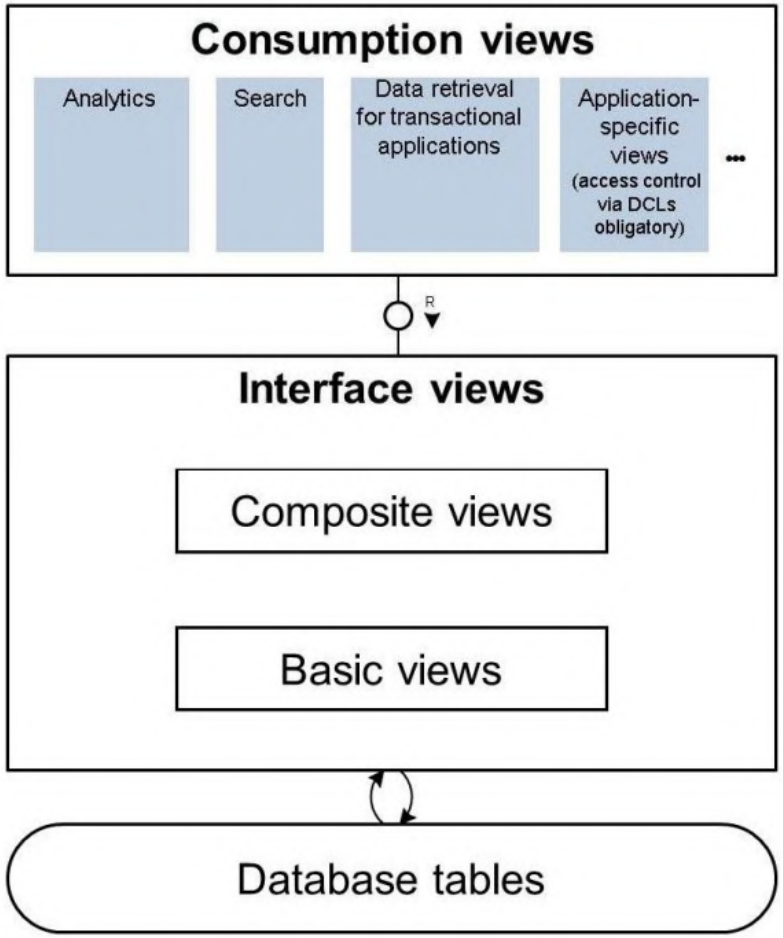
Learn About:

- Data Modeling with CDS Views
- Data Model-Related Annotations

Virtual Data Models



: The Layers in the Virtual Data Model (VDM)

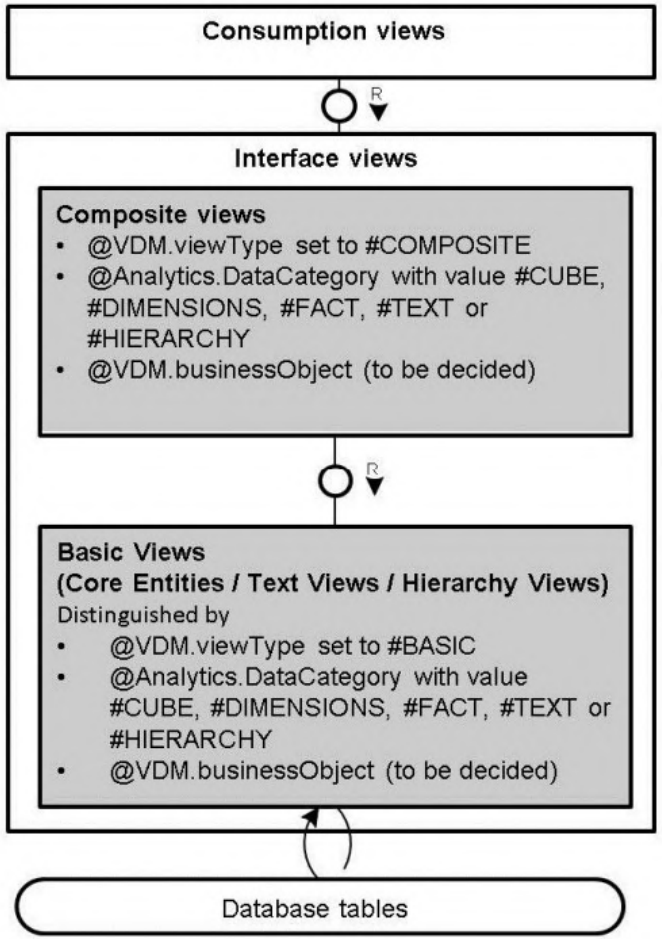


- Views from various consumption domains (e.g. Analytics, Search or …) which consume interface views and create queries on top to consume interface views

- Expose the public and stable VDM interface for consumers.
- Provide business semantics as a easily consumable and reusable data model.
- Domain-specific views (e.g. Analytics, Search or …)
- Instance Authorizations via DCLs
- Executed via ABAP SQL Interface

- Suite database model „as it is“

Interface Views Divided into Basic and Composite Views



Composite views are built selecting from the basic views and exploring the associations between basic views. They can be specific to a consumption domain or reusable in many.

Basic views form the low-redundancy model on top of the Suite database tables. Core entity views contain the associations to other core entity views.

Data Model-Related Annotations

- ***@VDM.viewType***
  - Admissible Values: *#BASIC, #COMPOSITE, #CONSUMPTION, #EXTENSION*
  - In VDM, all CDS views shall be annotated explicitly with a VDM view type
- ***@VDM.private***
  - Admissible Values: #blank (=true), true or false
  - Only private views need to be annotated explicitly
- **@ObjectModel.dataCategory**
  - Admissible Values: #TEXT, #HIERARCHY
- **@Analytics.dataCategory**
  - Admissible Values: #FACT, #CUBE, #DIMENSION