

ZS4HCDS

S/4HANA Development Fundamentals

SAP Development Learning
September , 2021

INTERNAL

Best Practice – Ground Rules

- Turn off email, phones, instant messaging tools and clear other distractions away from your training area.
- Close all the unnecessary browser windows/applications to save internet bandwidth.
- Participate and prepare to be called on by name.
- Raise your hand if you have an immediate question or comment.
- Be patient in waiting for a response to your Chat messages.
- If you leave the program, please use the coffee mug emoticon to let the trainer know when you leave/return.

Course agenda – 3 days

Day 1

- Introduction & Warm-up
- CDS Introduction
- ABAP Development Tools
- Lunch Break/Hands-on
- Mentimeter Quiz
- Annotations
- Basic SQL features in CDS
 - Expressions
 - Functions
 - Aggregations
 - Joins
 - Union

Day 2

- Day 1 Recap by participants
- Input parameters
- Associations
- Lunch Break/Hands-on
- Mentimeter Quiz
- View Extension
- Metadata Extension
- AMDP and CDS Table Function

Day 3

- Day 2 Recap by participants
- Data Control Language
- Restful Abap Programming Model
- Lunch Break/Hands-on
- Mentimeter Quiz
- Virtual Data Modeling
- CDS Test Double Framework

Unit 1 – Introduction to Core Data Services

Lesson 1 – Motivation and Definition

- **Structured Query Language (SQL)**
 - Access to database management systems (DBMS)
 - Largely standardized (SQL-92)
 - Three sublanguages: DML, DDL, DCL
- **Data Manipulation Language (DML)**
 - Read and change data in DB tables
 - Statements SELECT, INSERT, UPDATE, DELETE ...
- **Data Definition Language (DDL)**
 - Create and maintain DB tables and views
 - Statements CREATE, ALTER, DROP, ...
- **Data Control Language (DCL)**
 - Define authorizations and transaction control
 - Statements GRANT, REVOKE, ...

- **DML in ABAP**

- Open SQL, integrated in programming language
- DB-independent syntax
- DB-independent behavior
- Reduced feature set compared to SQL-92

- **DDL in ABAP**

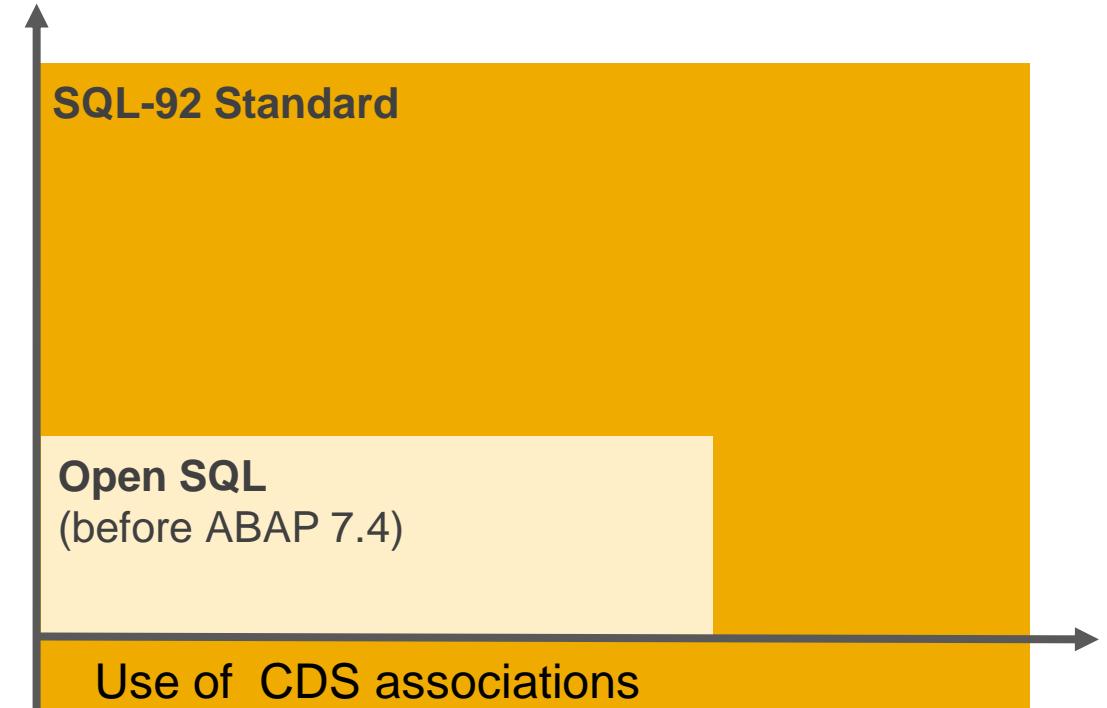
- ABAP Dictionary
- DB-independent tool
- Form-based editing of table definitions and views

- **DCL in ABAP**

- Not needed in ABAP
- Authorization and transaction control never on DB level
- Always done in application (AUTHORITY-CHECK, COMMIT WORK, ...)

Limitations of Open SQL Before Release 7.4

Limitations in ABAP < 7.4 SP05



- **DB-independent Definition of Database View**
 - Defined in ABAP Dictionary (form based tool)
 - Created on database (CREATE VIEW statement)
- **Very limited support of SQL features**
 - Only inner joins with Equi-join
 - Only simple projections (no calculated fields, no conversions, functions, ...)
 - Only simple selections (using literal values)
 - Only DB tables as source (View-on-View not supported)
- **Some ABAP specialities**
 - Buffering
 - Labels (through data elements)
 - Search helps and foreign key relations (inherited from underlying tables)

- **DB-independent Definition of Database View**
 - Defined in ABAP Dictionary (form based tool)
 - Created on database (CREATE VIEW statement)
- **Very limited support of SQL features**
 - Only inner joins with Equi-join
 - Only simple projections (no calculated fields, no conversions, functions, ...)
 - Only simple selections (using literal values)
 - Only DB tables as source (View-on-View not supported)
- **Some ABAP specialities**
 - Buffering
 - Labels (through data elements)
 - Search helps and foreign key relations (inherited from underlying tables)

Limitations of ABAP Dictionary Views

ABAP Dictionary Views

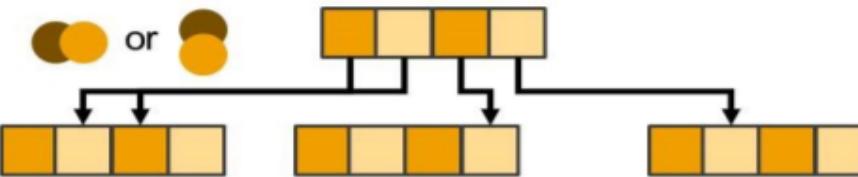


Support on all DBMSs



inner join & simple
selection only

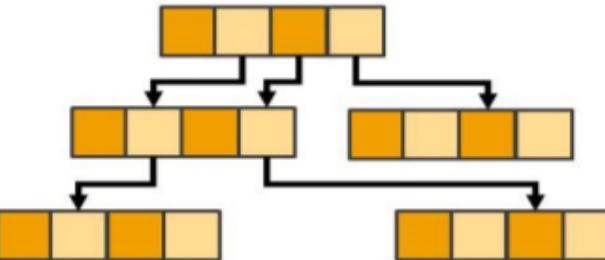
Join/Union, Projection, Selection



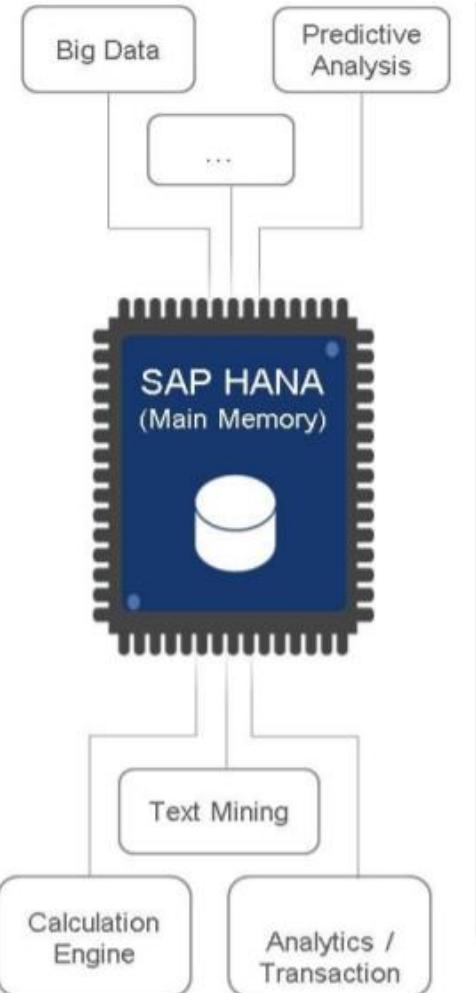
Calculation expressions, aggregation, grouping



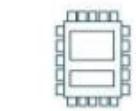
Nested views (on views)



The SAP HANA Platform



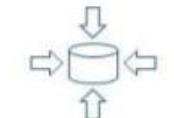
SAP HANA takes advantage of the latest technology developments



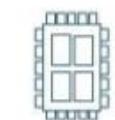
100% In-Memory computing
OLTP & OLAP in real-time



Column and row storage

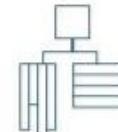


5-50x Compression
Based on column storage



10-10,000x Acceleration
Massive parallelization

And changed the way of developing and executing **applications**



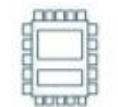
No

Aggregates
On-the-fly data models without duplicates



Less

Indices
Flexible and fast retrieval of the dataset



Less

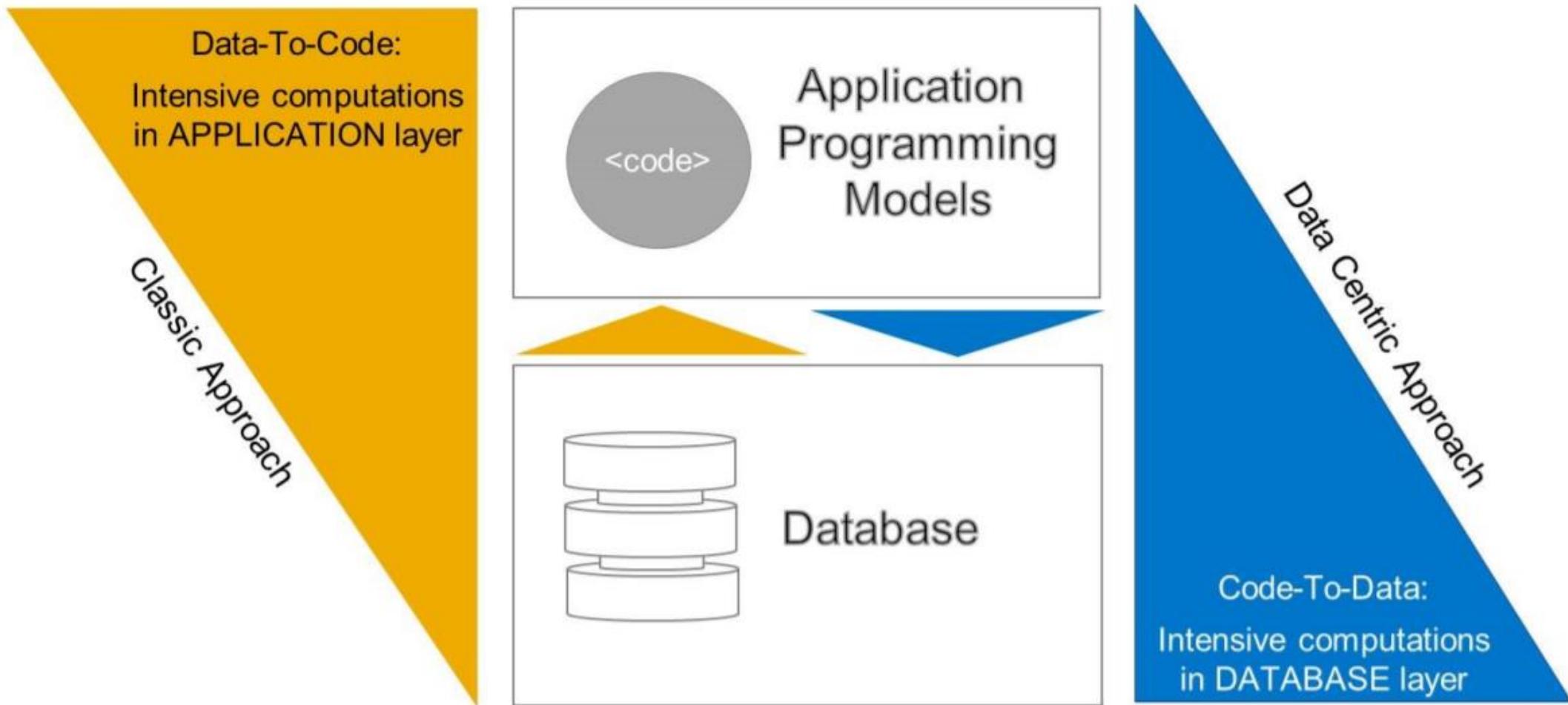
Code lines
Less complexity in data models and code



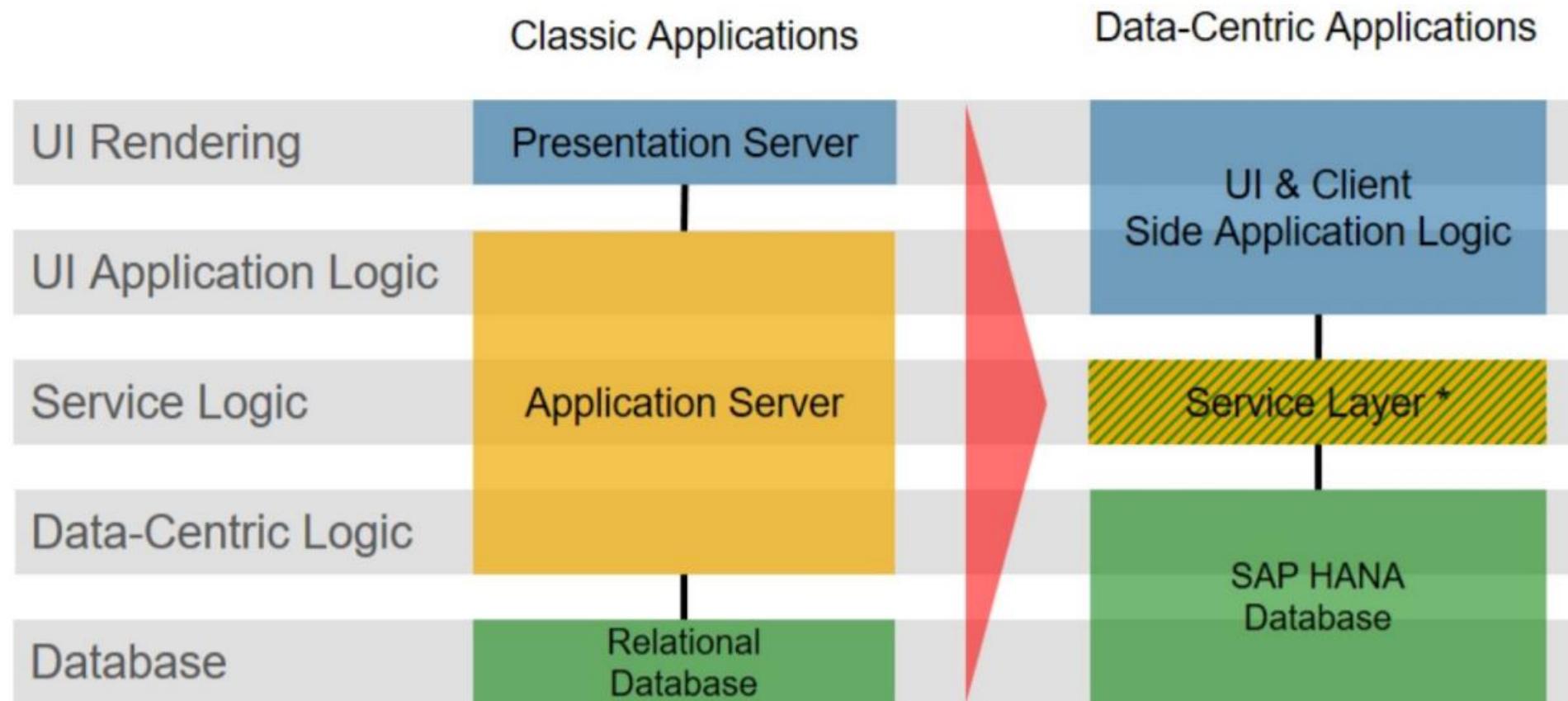
Partitioning

Mass data analysis

Paradigm changes in application programming

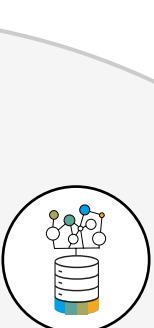


Transformation of Application Design with SAP HANA

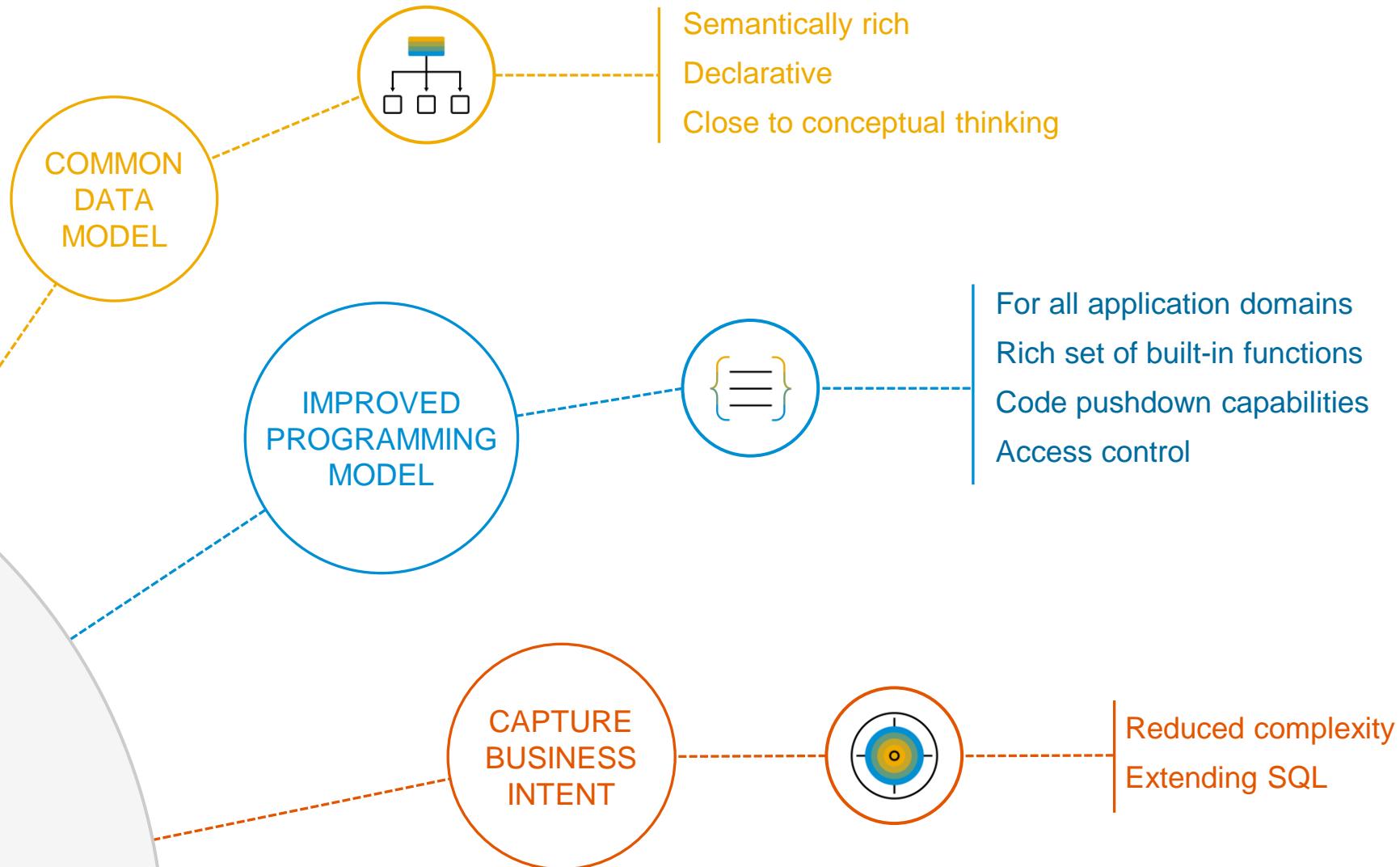


ABAP Core Data Services (CDS)

Next-generation data modeling



CORE
DATA
SERVICES



Core Data Services – A Family of Domain Specific Languages

The **Core Data Services (CDS)** are a collection of domain-specific languages and services for defining and consuming semantically rich data models.

DDL



Data Definition Language

Model and retrieve data on a semantic level higher than SQL

Extends native SQL means for increased productivity

QL



Query Language

Consume CDS entities via Open SQL in ABAP

Fully transparent SQL extensions

DCL



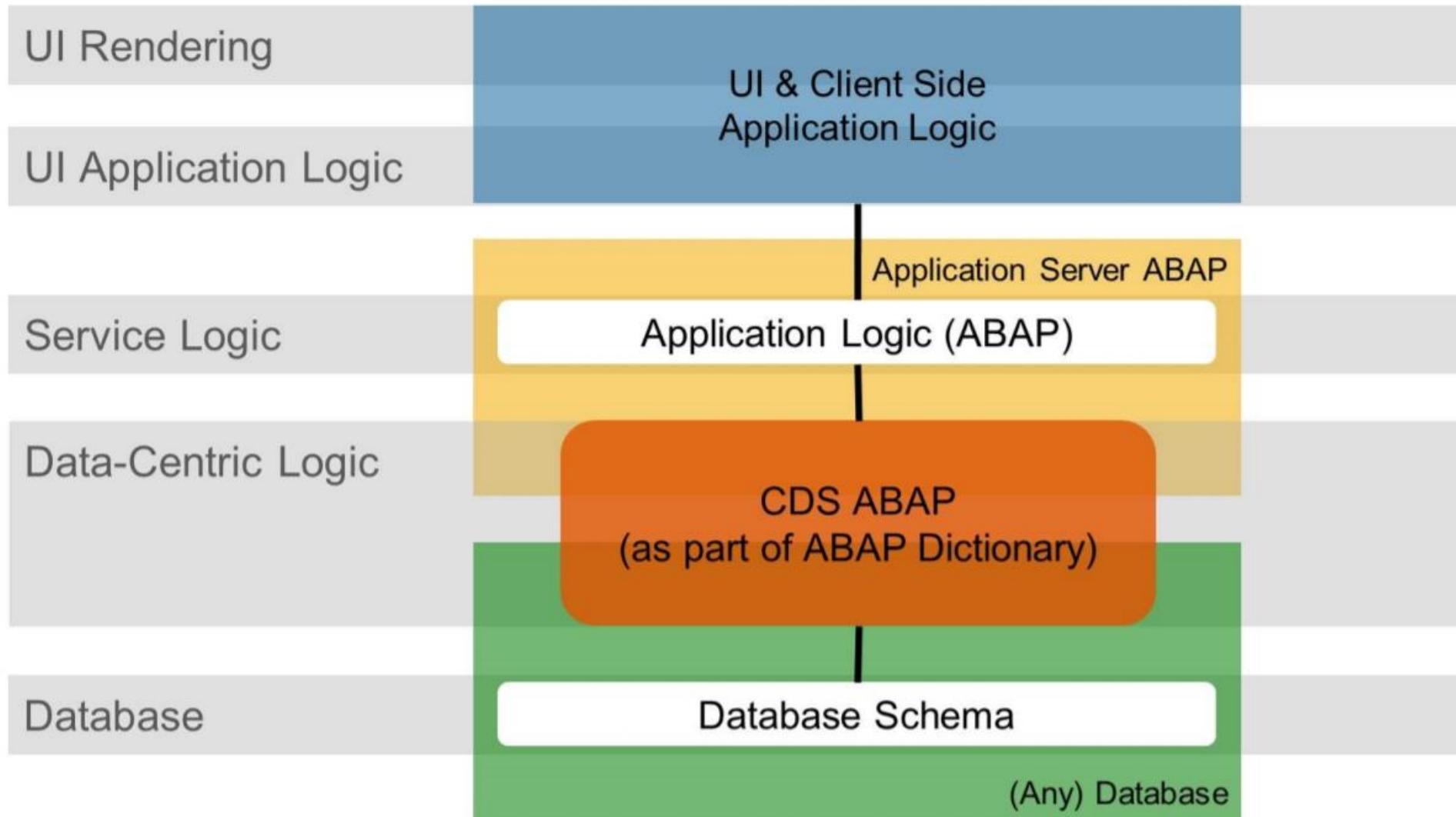
Data Control Language

Define authorizations for CDS views

Modelled and declarative approach

Integrates with classic authorization concepts

Core Data Services on ABAP



- **Database Independent**
 - Use CDS ABAP with any database supported by SAP
- **Advanced View Building**
 - CDS Views provide much more SQL features than classical Dictionary Views
- **Annotations to add Semantic Information**
 - Add end user texts, currency keys, buffer settings, ...
 - Add semantic information for consumers (analytics, OData, SAP UI5, ...)
- **Implicit Authorization Checks**
 - Define authorization rules for CDS Objects
- **Associations instead of Joins**
 - Define relations between CDS Objects that will be translated into joins
- **CDS Table Functions**
 - Views based on scripted coding (Currently only supported for SAP HANA)

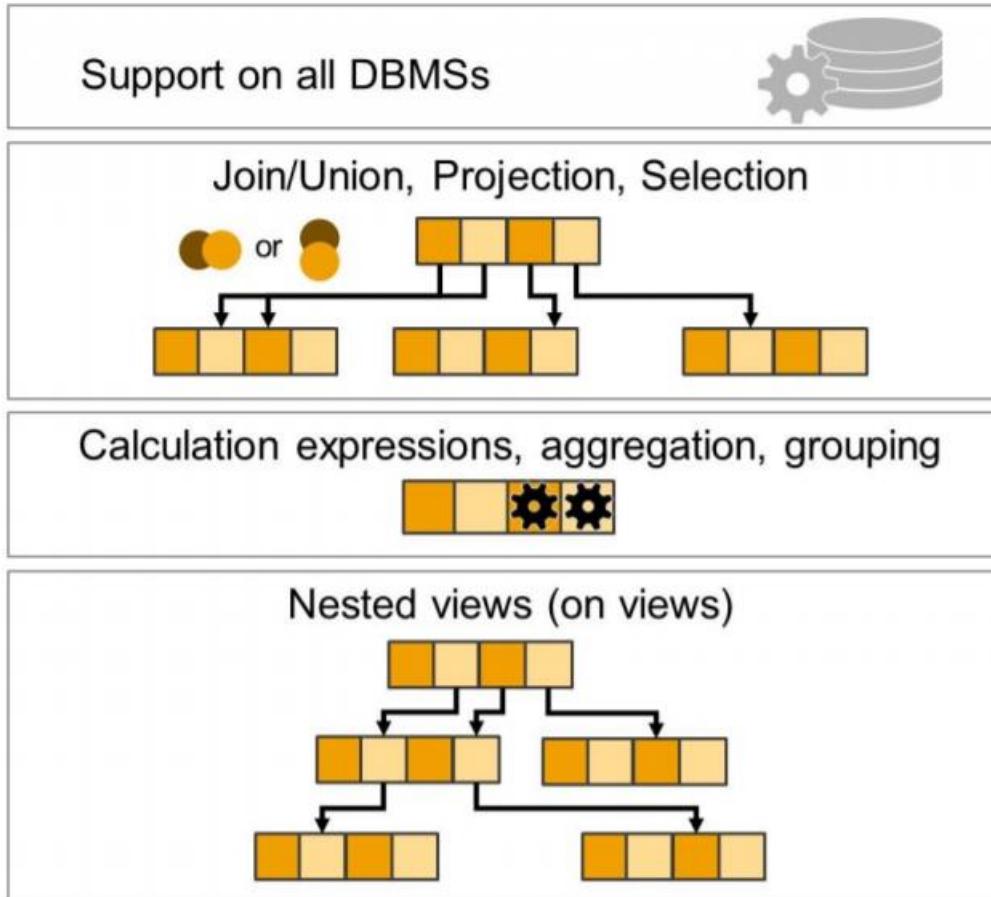
Advanced view building ABAP CDS

ABAP Dictionary Views

ABAP CDS Views

✓
✓
inner join & simple selection only

✗
not supported

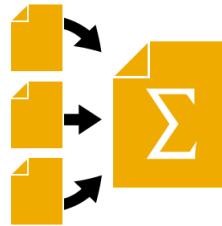


✓
✓
inner join, outer join, union

✓

✓

Lesson 1 – Motivation and Definition Summary



You should now be able to:

- Describe the motivation for Core Data Services
- Define Core Data Services

Unit 1 – Introduction to Core Data Services

Lesson 2 – Basic Syntax of CDS Views

DDL Source: CDS View and SQL View

```
@AbapCatalog.sqlViewName: 'S4D430_CARR'  
define view S4D430_Carrier as select carrid,  
carrname,  
currcode,  
url  
from scarr
```

Name of the
CDS View

FROM clause

Annotation:
Name of the
SQL View

Field list (comma
separated)

Variants for specifying the Element List

Standard SQL (After keyword SELECT)

```
@AbapCatalog.sqlViewName: 'S4D430_CARR'  
define view S4D430_Carrier as select carrid,  
carrname,  
currcode,  
url  
from scarr
```

CDS DDL Style (After FROM-Clause, enclosed in „{“ and „}“)

```
@AbapCatalog.sqlViewName: 'S4D430_CARR'  
define view S4D430_Carrier as  
    select from scarr  
    { carrid,  
      carrname,  
      currcode,  
      url }
```

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)

Key Elements

```
@AbapCatalog.preserveKey: true

define view S4D430_Connection3_Key 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
}
```

Define these elements as key elements

Create database object with the specified key

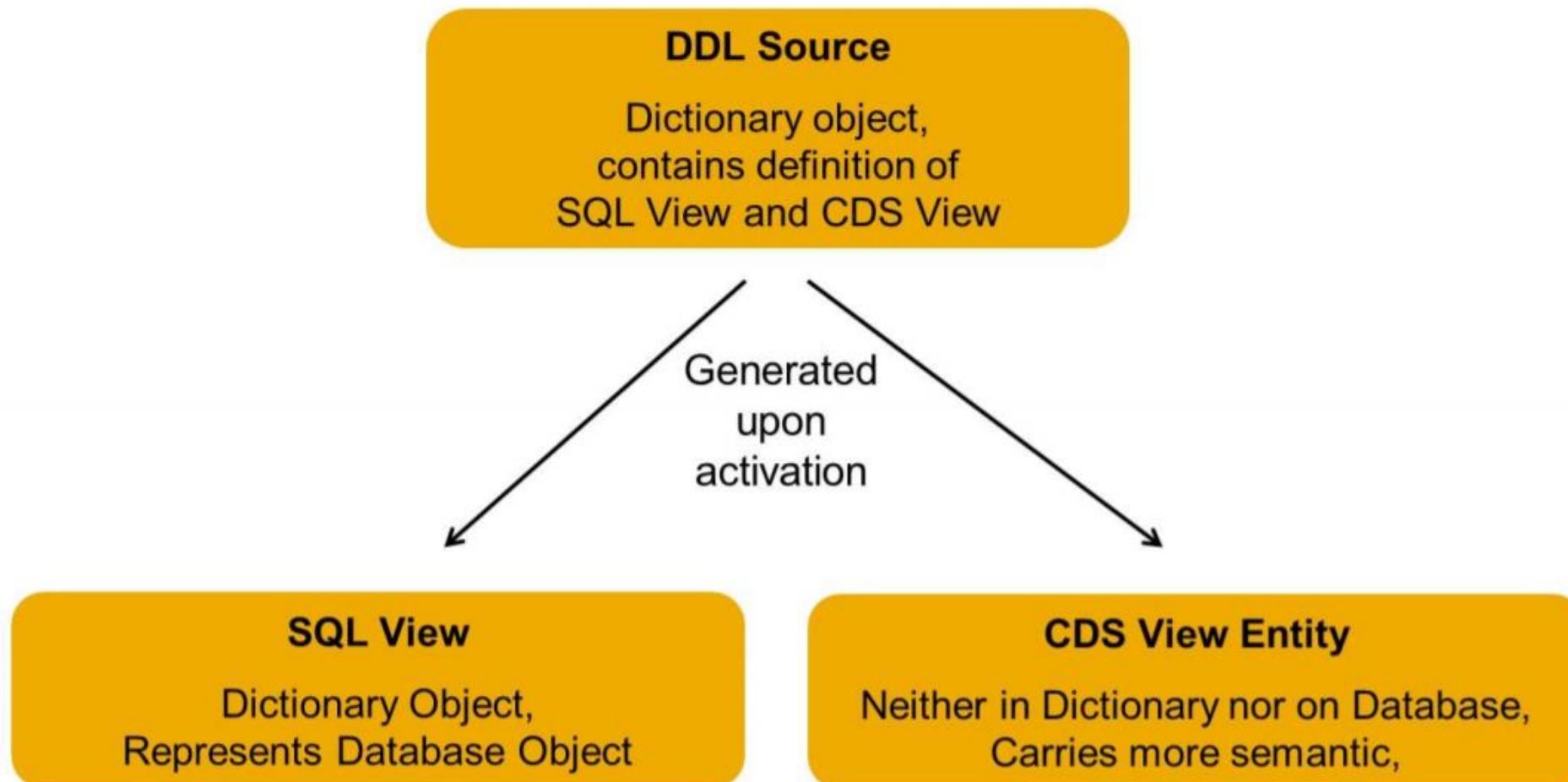
Need to be start elements of list (without gaps)

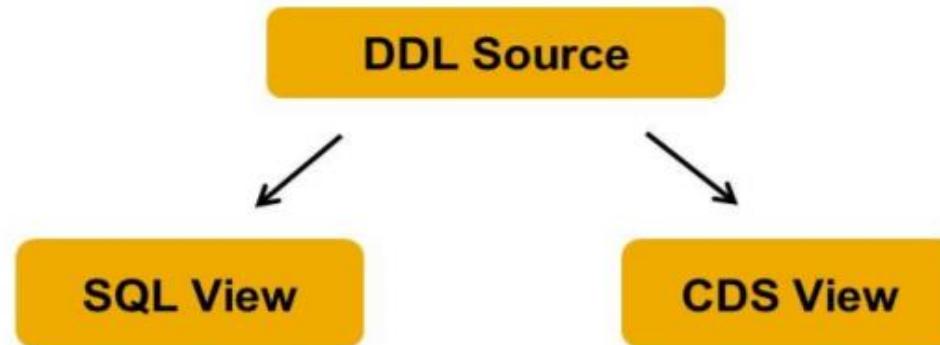
WHERE

- You can use compound WHERE clauses.
- You can reference the same column multiple times.

```
@AbapCatalog.sqlViewName:'S4HCDS_SEL_V07'
define view s4hcds_sel_v7 as select from sflight as f
{
    f.carrid as ID,
    f.connid as Connection,
    f.planetype as PlaneType,
    f.seatsmax as MaxSeats
}
where
seatsmax <= 330 and seatsmax > 100 or planetype = 'A340-600'
```

One DDL Source - Define Two Objects





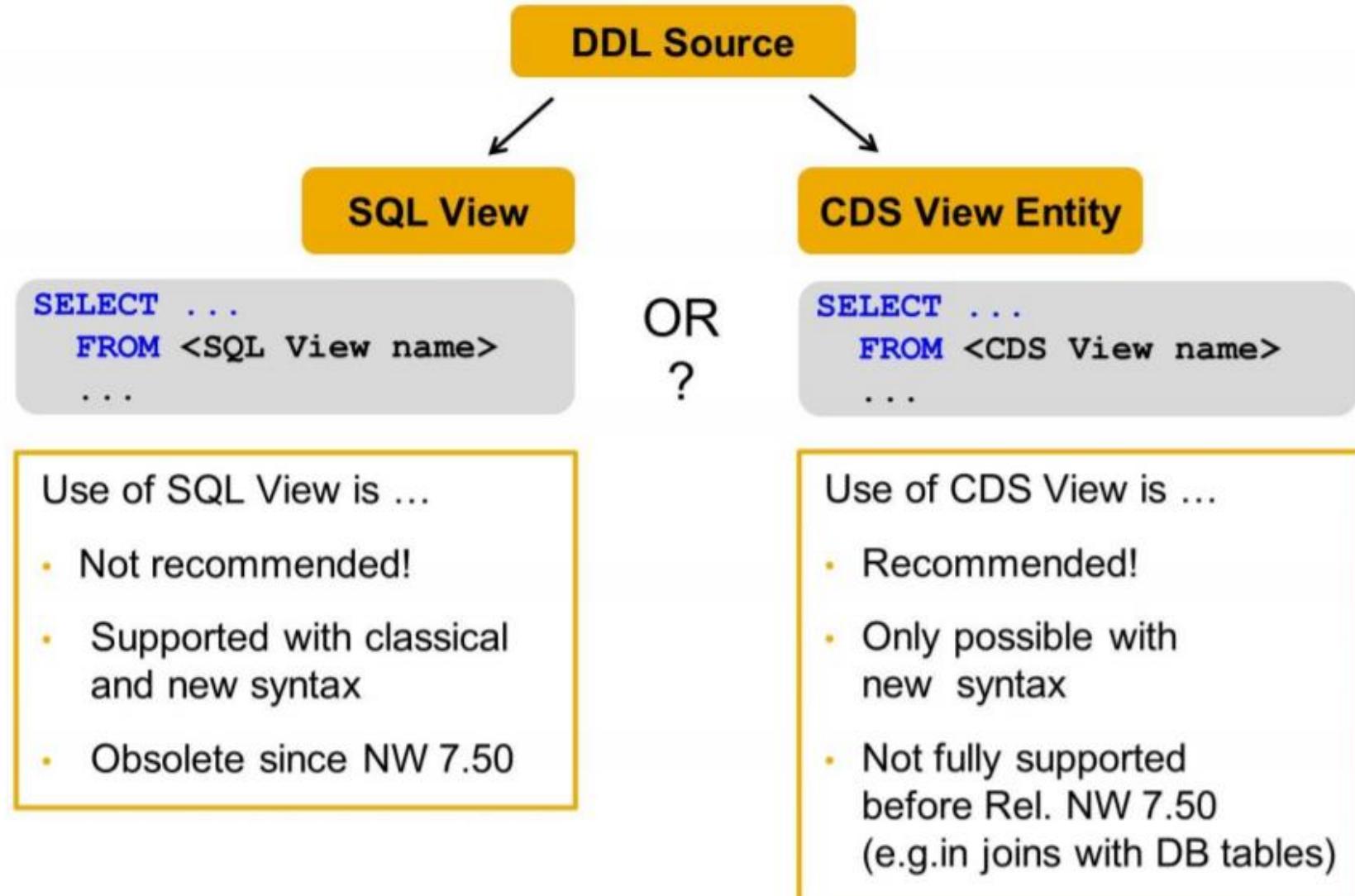
SQL View ...

- Repository Object (Dictionary)
- Represents “real” database object
- Is an ABAP data type
(structure, with client field)
- No additional semantics

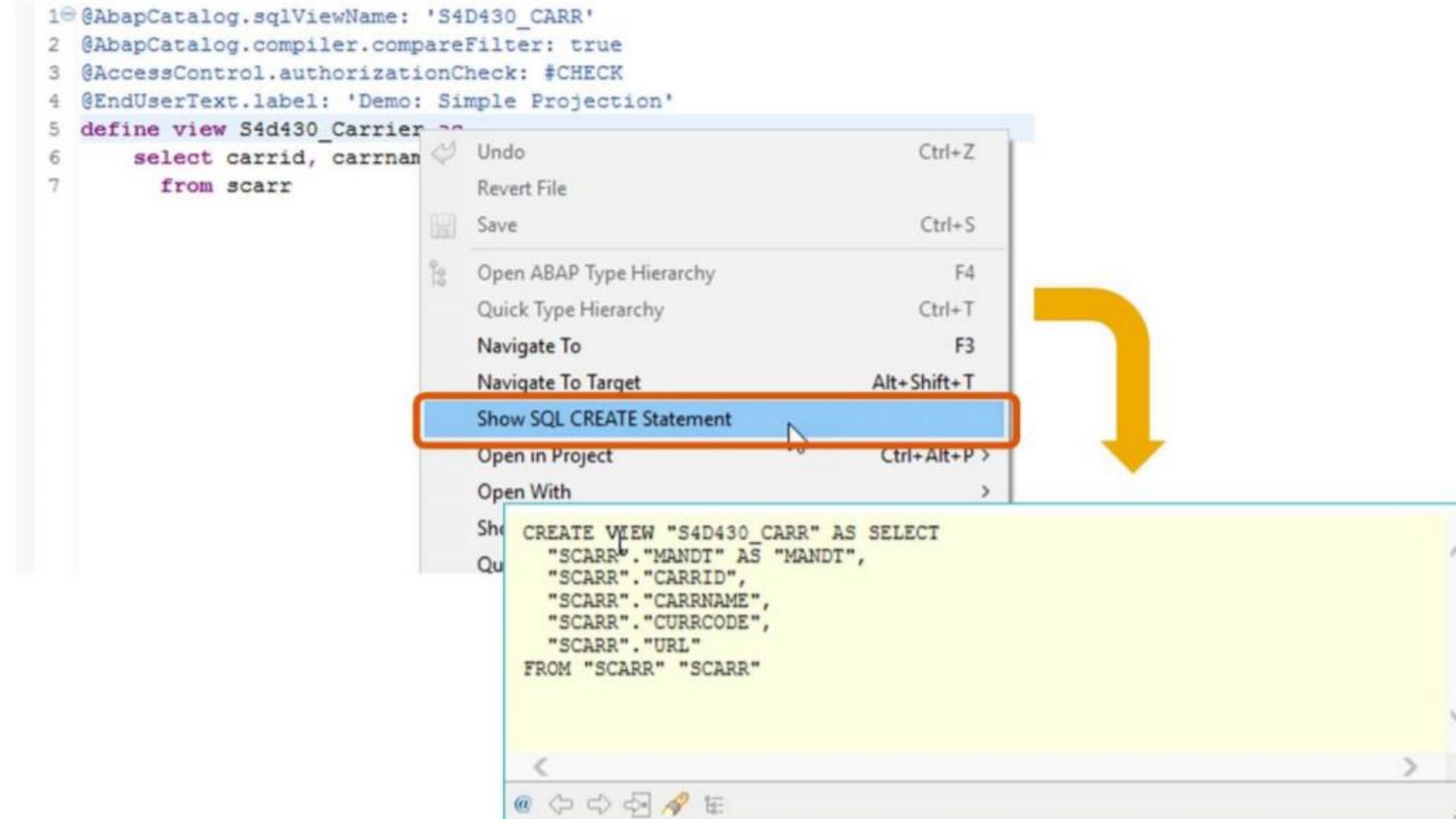
CDS View ...

- Not found in repository
- Not known to database
- Is an ABAP data type
(structure, without a client field)
- Additional semantics (Annotations)

SQL View vs CDS View



Display SQL Create Statement



Dependency Analyzer

The screenshot shows the SAP Studio interface with the 'Dependency Analyzer' feature highlighted. A context menu is open over a CDS View named 'S4D430_CARRIER'. The 'Open With' option is selected and highlighted with a red box. A large yellow arrow points from this menu down to a SQL Dependency Tree table below. The table displays dependencies between objects. The 'SCARR' row in the tree is also highlighted with a red box.

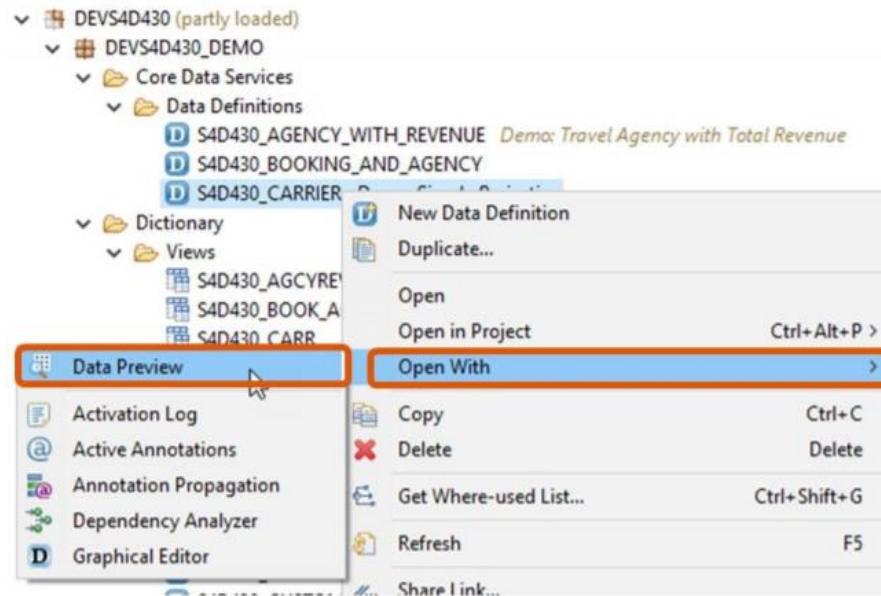
Context menu options:

- New Data Definition
- Duplicate...
- Open
- Open in Project Ctrl+Alt+P >
- Open With** (highlighted)
- Copy Ctrl+C
- Delete Delete
- Get Where-used List... Ctrl+Shift+G
- Refresh F5
- Share Link...

SQL Dependency Tree table data:

SQL Name	SQL Relation	Object Type	Entity Name	Database Object
S4D430_CARRIER	From	CDS View (STOB)	S4d430_Carrier	Information not available
SCARR		Database Table (TABL)		Information not available

Data Preview for CDS views



The screenshot shows the SAP Studio interface with the 'S4D430_CARRIER' CDS view selected. The main area displays a table titled 'Raw Data' with 18 rows retrieved in 15 ms. The columns are 'carrid', 'carrname', 'currcode', and 'url'. The data includes entries for American Airlines, Air Berlin, Air Canada, Air France, Alitalia, British Airways, Continental Airlines, and Delta Airlines. The 'SQL Console' and 'Number of Entries' buttons are highlighted with orange boxes and arrows pointing to them from the left menu. Below the table, a SQL query is shown in the 'SQL Console':

```
1 SELECT
2   S4D430_CARRIER~CARRID,
3   S4D430_CARRIER~CARRNAME,
4   S4D430_CARRIER~CURRCODE,
5   S4D430_CARRIER~URL
6 FROM
7   S4D430_CARRIER
8
```

A modal dialog box titled 'Number of Entries' displays the message: 'Number of entries that meet the selection criteria: 18'. The 'OK' button is visible at the bottom right of the dialog.

Unit 1 – Introduction to Core Data Services

Lesson 3 – Annotations

Classification of SAP Annotations

	ABAP Annotations (Evaluated by ABAP runtime)	Framework-specific (Evaluated by Framework like OData, VDM, UI, Analytics, ...) **
View Annotations	<code>@AbapCatalog.sqlViewName</code> <code>@ClientDependent</code> <code>@EndUserText.label</code>	
Element Annotations	<code>@Semantics.unitOfMeasure</code> <code>@EndUserText.label</code>	
Parameter Annotations *	<code>@Environment.systemField</code> <code>@EndUserText.label</code>	
Extension Annotations *	<code>@AbapCatalog.sqlViewAppendName</code>	
Function Annotations *	<code>@ClientDependent</code> <code>@EndUserText.label</code>	

Important ABAP View Annotations

```
@AbapCatalog.sqlViewName: 'S4D430_ANNO1'  
  
@ClientHandling.type:      #INHERITED  
@ClientHandling.algorithm: #AUTOMATED  
  
@AccessControl.authorizationCheck: #CHECK  
  
@AbapCatalog.compiler.CompareFilter: true  
  
@AbapCatalog.Buffering.type: #GENERIC  
@AbapCatalog.Buffering.numberOfKeyFields: 1  
@AbapCatalog.Buffering.status: #ACTIVE  
  
define view S4d430_Annotations1  
...
```

Control
client handling in
Open SQL

Switch on/off implicit
access control in
Open SQL

Control evaluation of
filtered associations

Switch on/off
buffering of SQL
View in Open SQL

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*

Element Annotations after Element

Element Annotation before the Element

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

Requires „<“
after „@“

Element annotation after the Element

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

Annotation
before comma

ABAP Annotations for Translatable Texts

View Annotation

```
...  
@EndUserText.label: 'Demo: Simple View Definition'  
...
```

Maximum of
60 Characters

Element Annotations

```
...  
@EndUserText.label: 'Discount'  
@EndUserText.quickInfo: 'Customer Specific Discount'  
c.discount,  
...
```

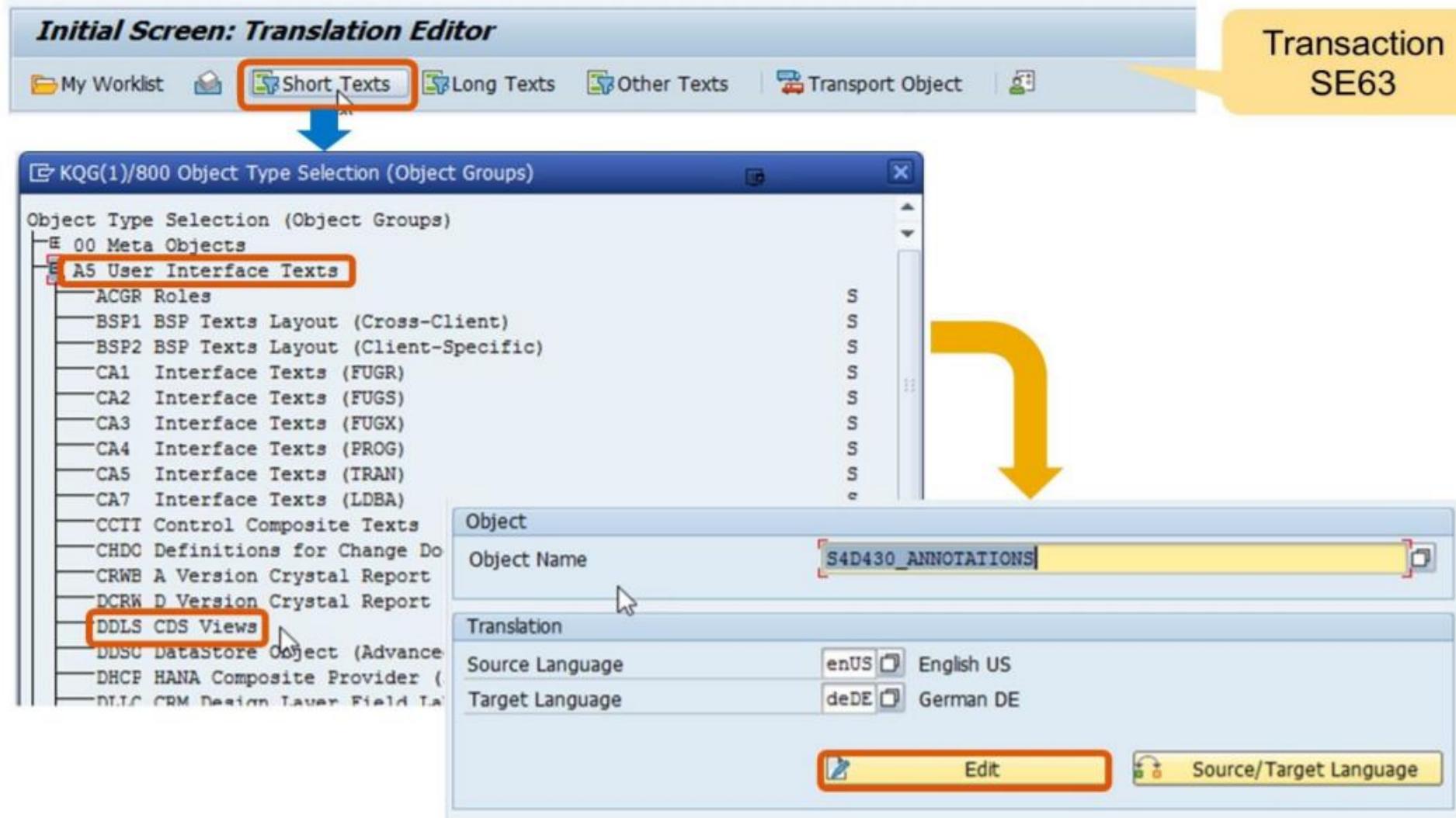
Maximum of
60 Characters

Before or after
the element

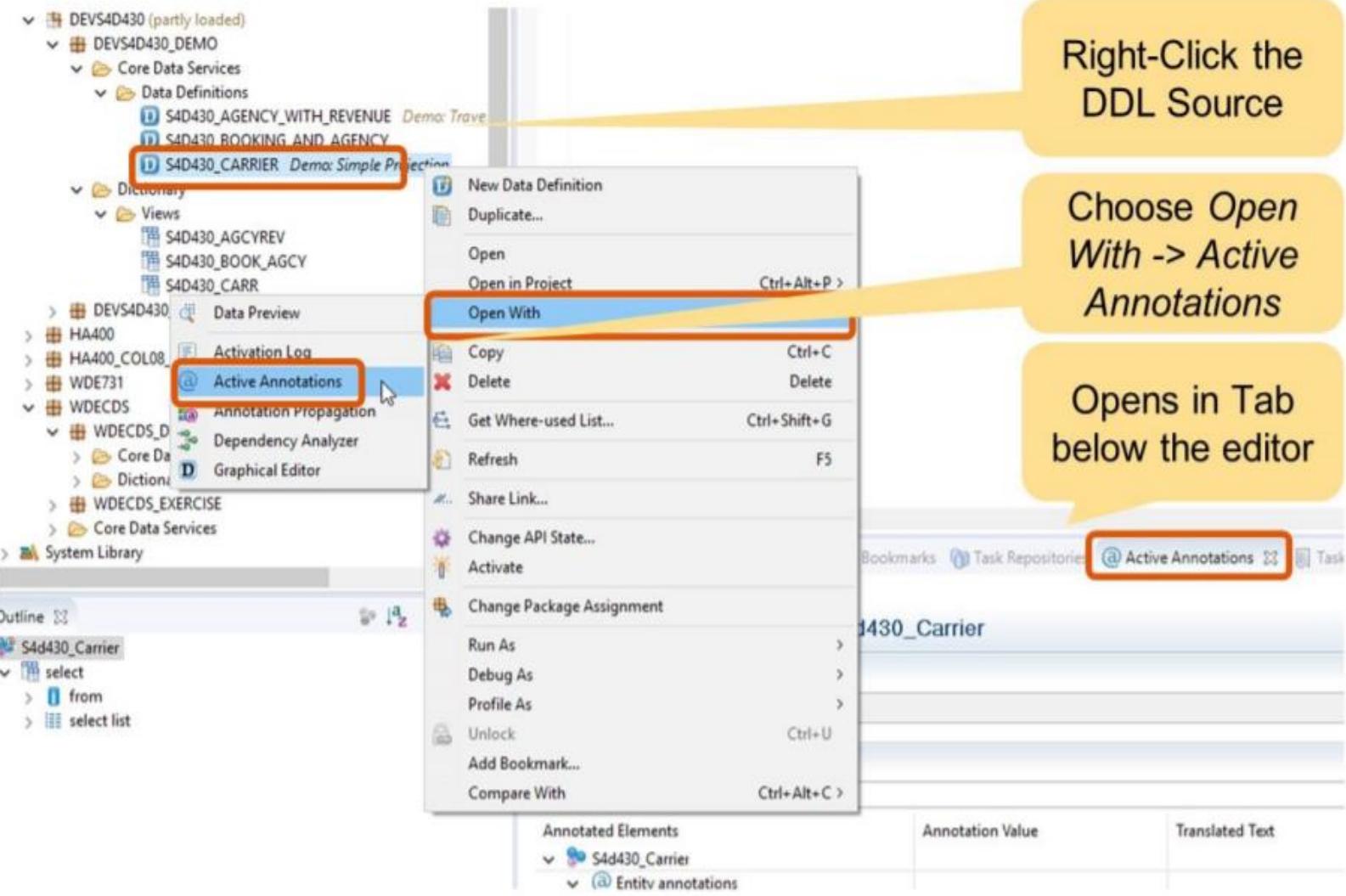
Note:

Element text annotations are required if no text is inherited from data source (i.e. table definition or other view)

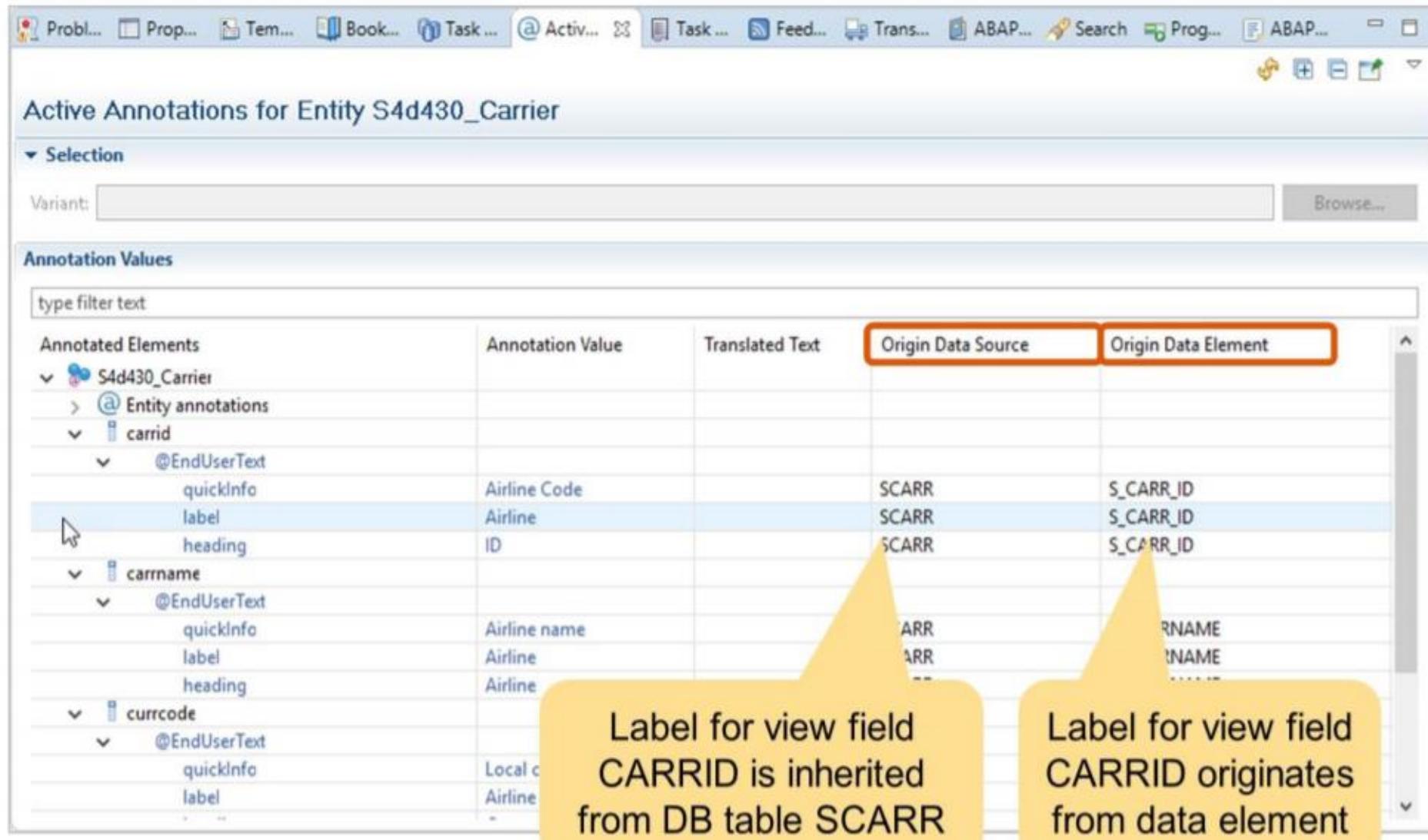
Translation of CDS Views



Active Annotations of CDS Views



Active Annotations of CDS Views



Annotated Elements	Annotation Value	Translated Text	Origin Data Source	Origin Data Element
↳ S4d430_Carrier				
> @Entity annotations				
↳ carrid				
↳ @EndUserText				
quickInfo	Airline Code		SCARR	S_CARR_ID
label	Airline		SCARR	S_CARR_ID
heading	ID		SCARR	S_CARR_ID
↳ carrname				
↳ @EndUserText				
quickInfo	Airline name		ARR	RNAME
label	Airline		ARR	TNAME
heading	Airline		--	-----
↳ currcode				
↳ @EndUserText				
quickInfo	Local c			
label	Airline			

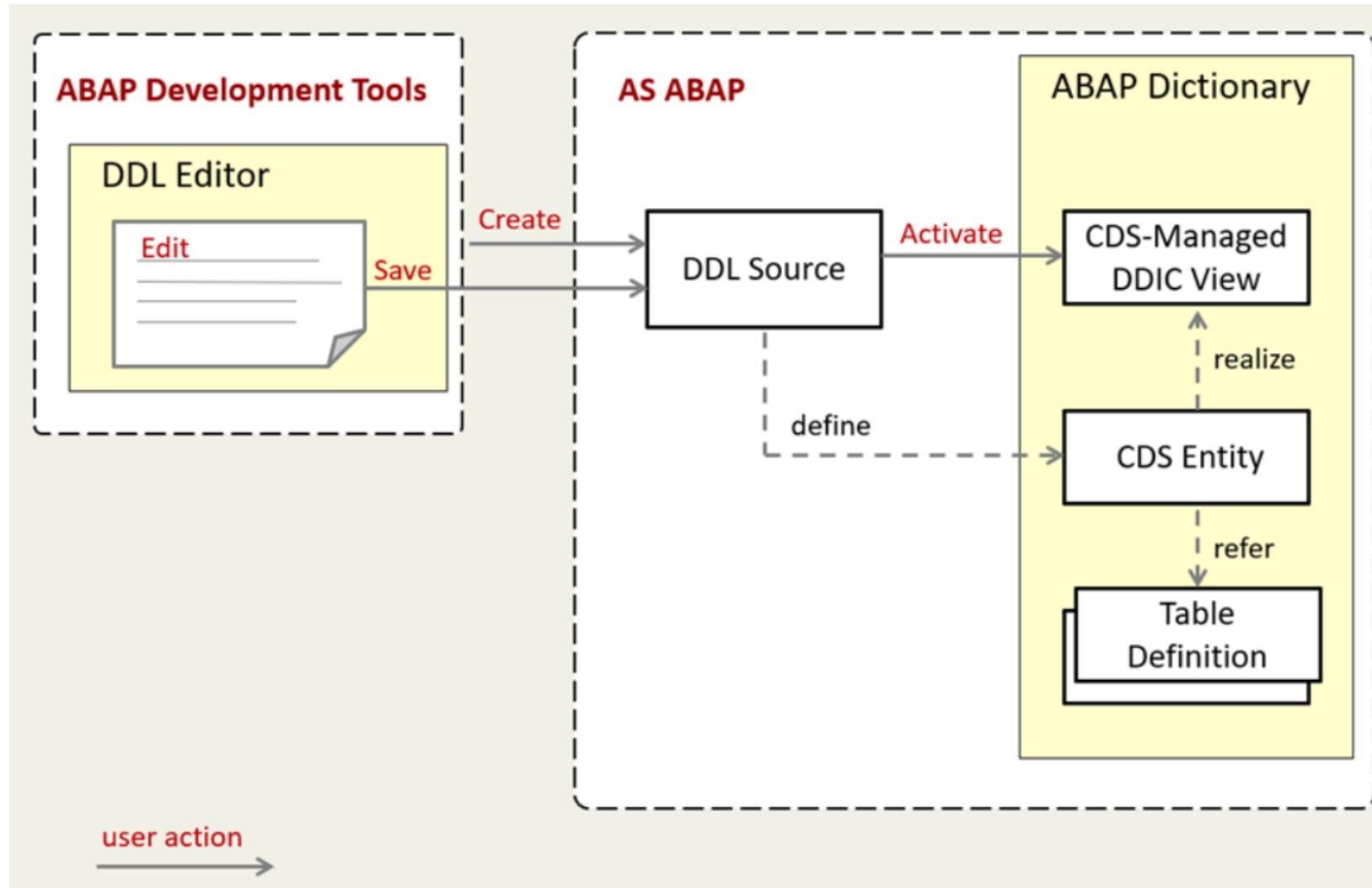
Label for view field
CARRID is inherited
from DB table SCARR

Label for view field
CARRID originates
from data element

NEW Feature : CDS View Entities AKA ‘CDS view V2’

```
@define view entity I_SampleViewEntity as select from ...  
{  
    ...  
}
```

CDS view V1



Important Differences

Description	CDS DDIC-Based Views	CDS View Entities
Optimized CDS activation	The CDS entity, CDS DDIC-based view, and the database view need to be activated.	<ul style="list-style-type: none"> Only the CDS entity and the respective database view on SAP HANA need to be activated. Consequently, the performance of the activation runs faster. Simplified handling of CDS extends
Syntax and annotation checks	Simple checks	Stricter checks that indicate critical situations more explicitly
Client handling	<ul style="list-style-type: none"> Based on different algorithms Controlled using the ClientHandling.type and ClientHandling.algorithm annotations 	<ul style="list-style-type: none"> Automatized for complete transparency No annotation allowed for client handling
Database object	View / table function	View
CDS-managed DDIC view in ABAP Dictionary	Creation upon activation	No CDS-managed DDIC view is generated upon activation
Naming convention	Three names for the CDS object, CDS entity, and the CDS-managed DDIC view on SAP HANA	One name for the CDS object, CDS entity, and database view on SAP HANA

[CDS View Entities - ABAP for HANA - Wiki@SAP](#)

[CDS Documentation: V2 Differences - ABAP
Dictionary - Wiki@SAP](#)

Unit 1 – Introduction to Core Data Services

Lesson 4 – SQL Features in ABAP CDS

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)

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
    ...
}
```

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,
  ...
}
```

Four arithmetic operators (+, -, *, /) and brackets

Three Types of Expressions

- **Integer Expression**

- All Operands have integer type (INT1, INT2, INT4, INT8)

- **Decimal Expression**

- At least one operand has decimal type (DEC, CURR, QUAN)
- No operand of type FLTP

- **Floating Point Expression**

- All operands have type FLTP

Important Restrictions

- **If one operand has type FLTP, all operands require type FLTP**
- **Operator "/" only allowed in floating point expressions**

```
@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
}
```

Type Conversion with Cast Expression

```
CAST ( operand AS target_type [PRESERVE 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 PRESERVE TYPE to change semantic attributes, only

Example: Type Conversion with Cast Expression

```
@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	1.989110900000000E+07	1989-11-09	1989-11-09	9.7402597402597402E-01
19891109	19.891.109	19891109.00	1.989110900000000E+07	1989-11-09	1989-11-09	9.6623376623376622E-01
10001100	10.001.100	10001100.00	1.000110000000000E+07	1000 11 00	1000 11 00	0.6623376623376622E-01

Numeric Functions

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 to 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

Built-in Rounding Functions

▪ **abs(arg)**

- returns the absolute value of arg

▪ **floor(arg)**

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

▪ **ceil(arg)**

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

▪ **round(arg,pos)**

- pos > 0: Round arg to pos decimal places
- pos < 0: Round arg to position

abs(1.5) = 1.5

abs(-1.5) = 1.5

floor(1.5) = 1

floor(-1.5) = -2

ceil(1.5) = 2

ceil(-1.5) = -1

round(3.1514, 2) = 3.15

round(273.15,-1) = 270

String Processing Functions

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!

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)**

String Processing Functions

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**

```
@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'
```

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: Currency and Unit Conversions

```
@AbapCatalog.sqlViewName: 'S4D430_FUNCNV2'
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

```
@AbapCatalog.sqlViewName: 'S4D430_FUNCNV1'
define view S4d430_Function_Conversion1 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 „=>“

- **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: Calculation with Dates

```
@AbapCatalog.sqlViewName: ,S4D430_FUNC DAYS'  
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)

Exercise 01: Enhance the CDS view with Expressions



In this exercise you will:

- Use CASE expression to define the column more descriptive
- Use CAST expression in floating point calculation
- Use CONCAT function to combine 2 strings
- Use currency_conversion special function to convert the currency
- See Student Manual, Exercise 1 Tasks 1-5

Time: 15 min

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

Aggregate Functions - Example

```
@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)

Aggregate Functions – Group by

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

Field list contains expression or function

Arguments of the expression or function listed individually

Exercise 02: CDS view with Aggregate Expressions

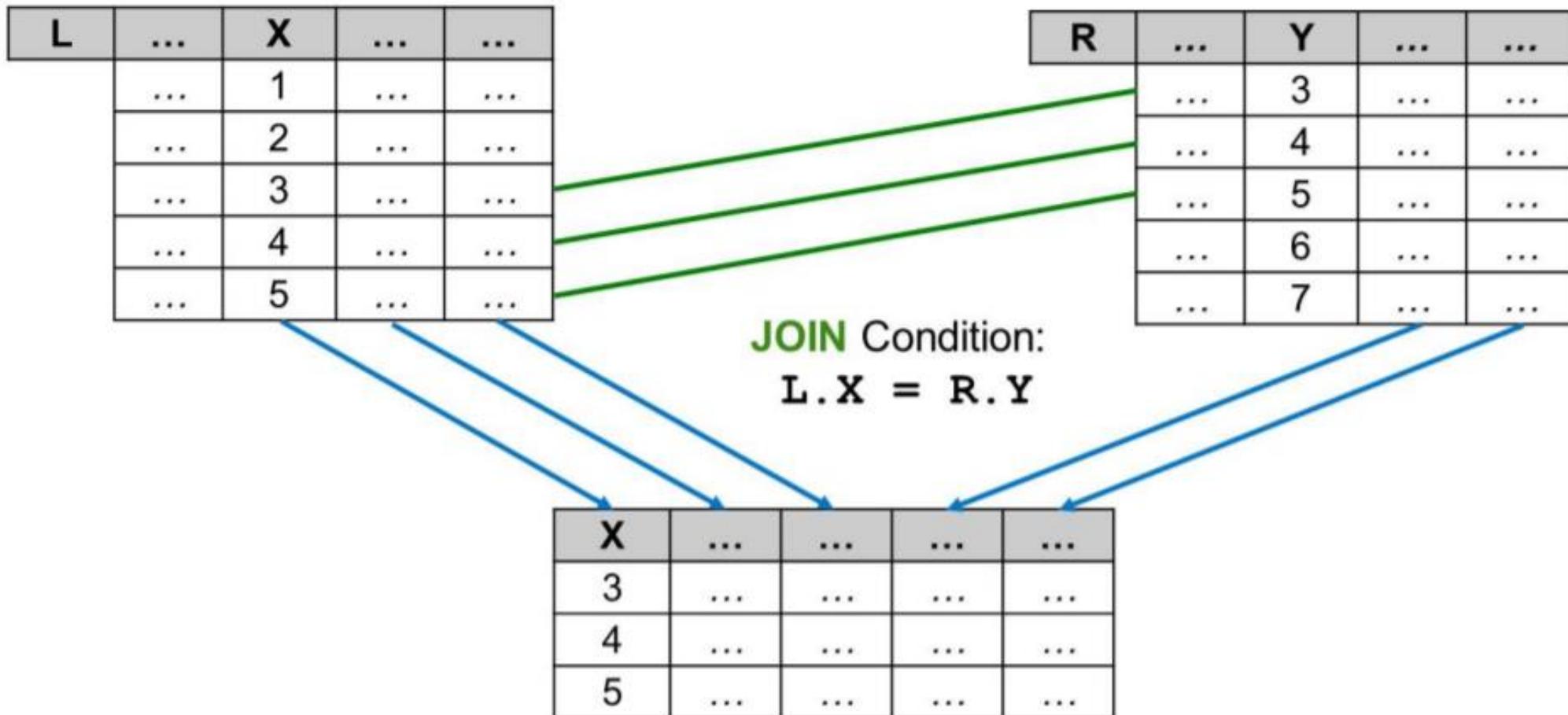


In this exercise you will:

- Use COUNT aggregation function to find the total number of bookings
- Use MIN and MAX functions to find the minimum and maximum booking amount
- Use HAVING condition to restrict the result groups
- See Student Manual, Exercise 2 Tasks 1-4

Time: 15 min

Inner Join



- **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

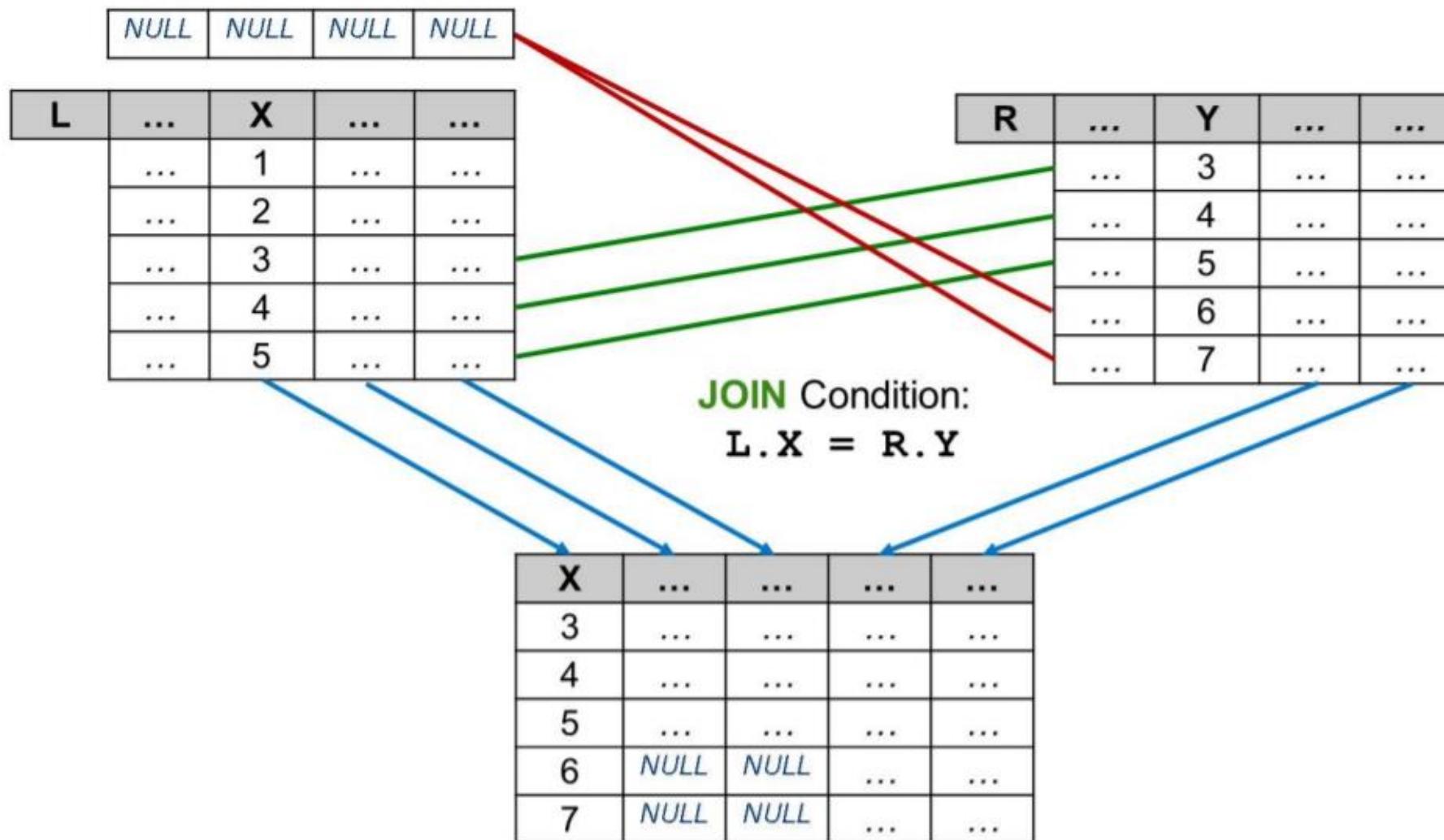
- **Full Outer Join**

- 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

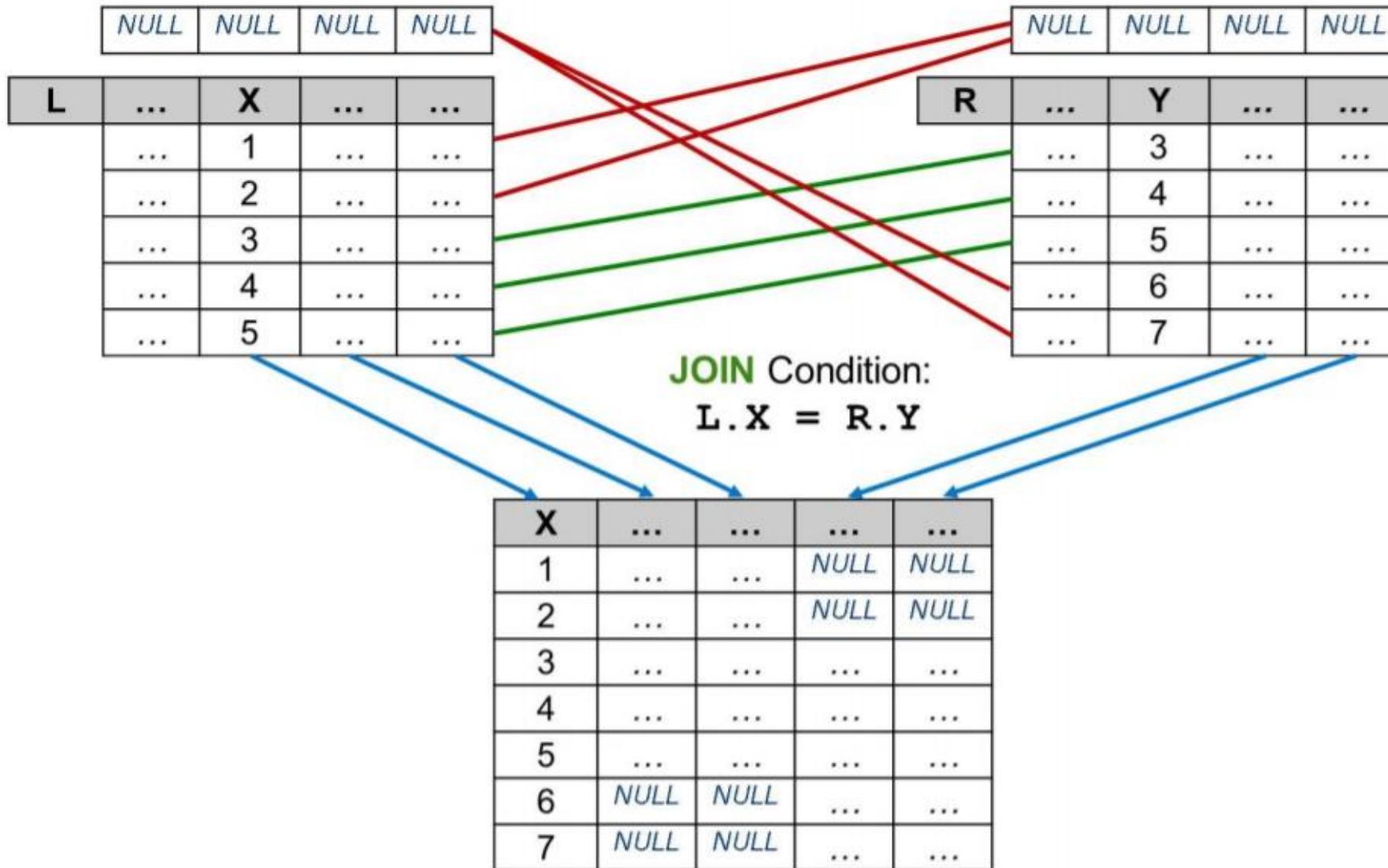
Left Outer Join



Right Outer Join



Full Outer Join (not supported in ABAP CDS)



Right Outer Join - Example

```
@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
    }
```

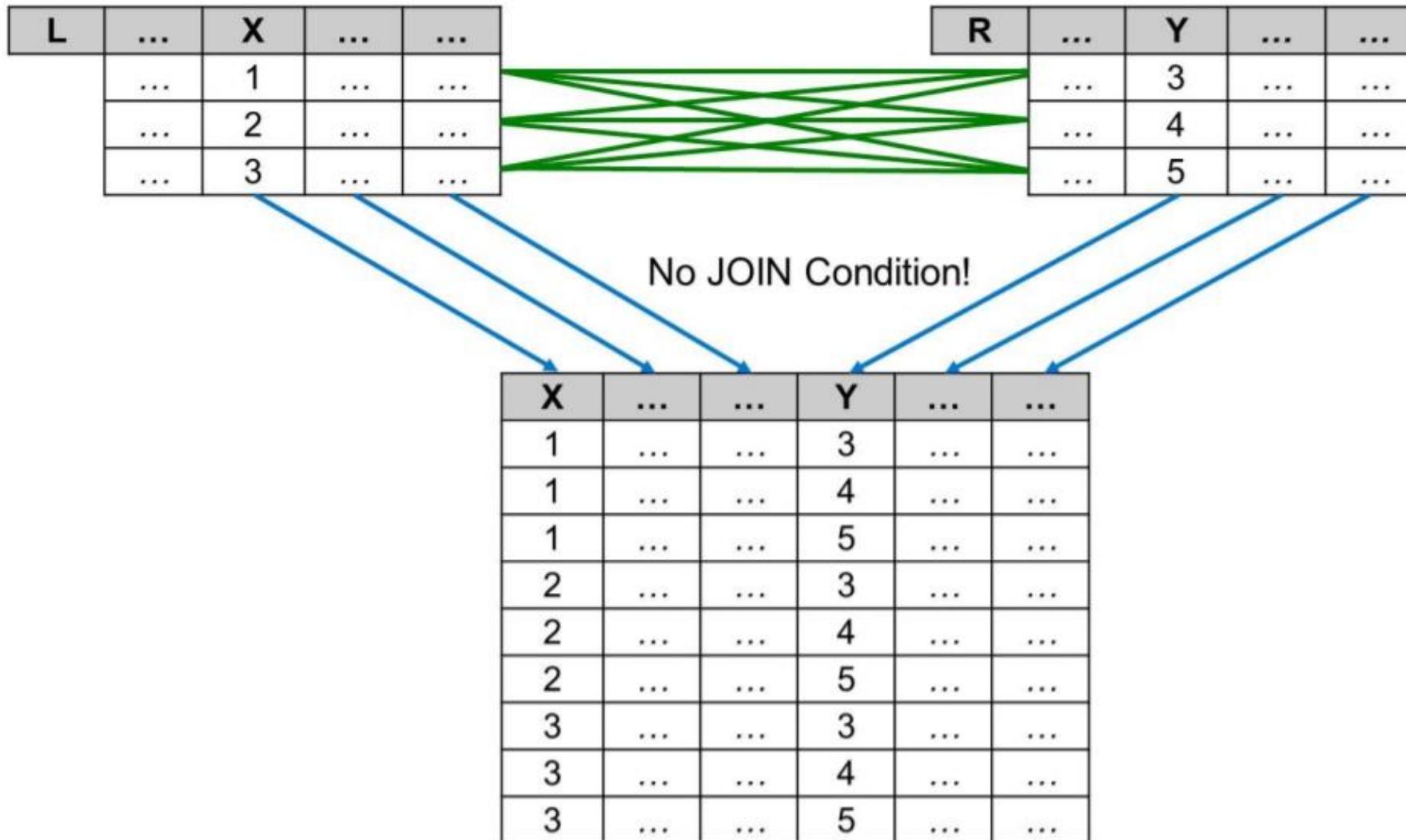
Raw Data

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
			PARIS	TOKYO

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

Cross Join (as of Release 7.51)



Cross Join - Example

```
@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

Nested Join Expressions – Order of Evaluation

Explicit (with parentheses) or Implicit (no parentheses)

- Parentheses are recommended (makes the code easier to read)

Implicit Order of Evaluation

- INNER and OUTER JOINS: Arrangement of the ON conditions
- CROSS JOINS: from left to right

```
@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
}
```

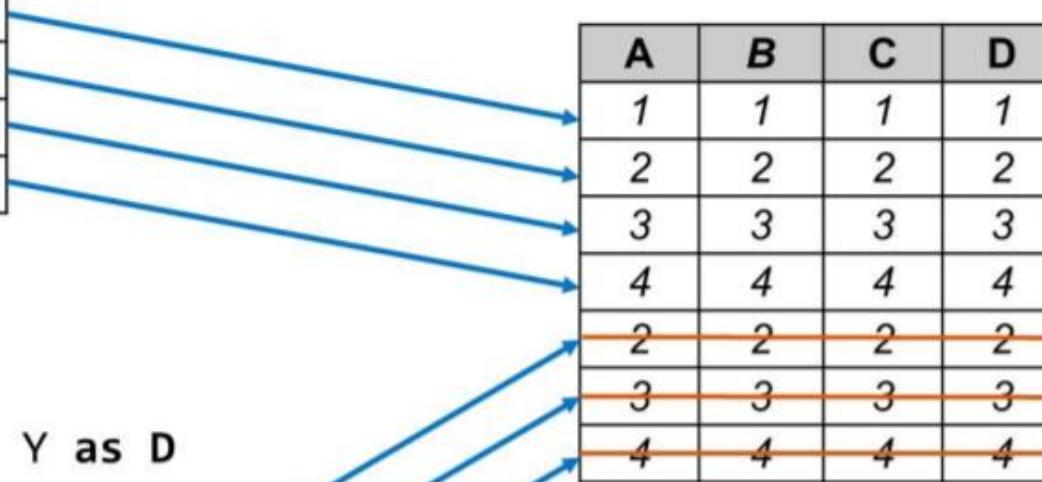
Join in parentheses
is evaluated first

```
@AbapCatalog.sqlViewName: 'S4D430_JOINIMPL'
define view S4d430_Join_Nested_Implicite 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
}
```

No parentheses:
Sequence of
ON-conditions rules

SELECT A, B, C, D
FROM ... WHERE ...

A	B	C	D
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4



UNION

SELECT A, B, X as C, Y as D
FROM ... WHERE ...

A	B	X	Y
2	2	2	2
3	3	3	3
4	4	4	4

Duplicates are removed
from the result set

Prerequisites

- **Compatible Structure of Result Sets**
 - Same number of elements
 - Elements in same position have compatible data type
 - Elements in same position have identical name/alias

Properties of Result Set

- **Element Names**
 - Are identical in both SELECT statements
- **Key Definition**
 - taken from first SELECT statement
- **Element Type**
 - taken from first SELECT statement

UNION - Example

```
@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
    'Agency'           as id,
    as type,
        name,
        city,
        country
}
```

Key definition in second select is ignored

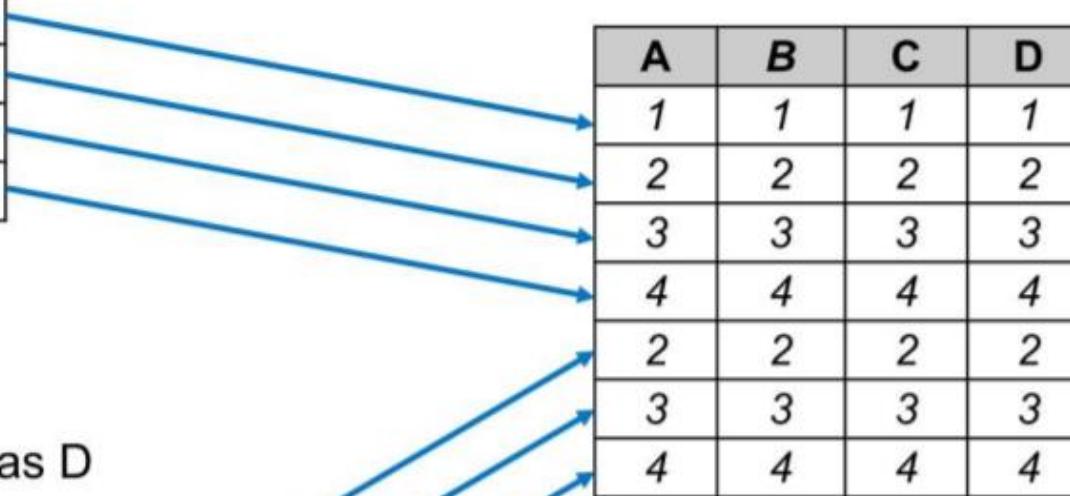
Alias *id* is needed to have identical element names

Additional blanks needed to have compatible types

UNION ALL

SELECT A, B, C, D
FROM ... WHERE ...

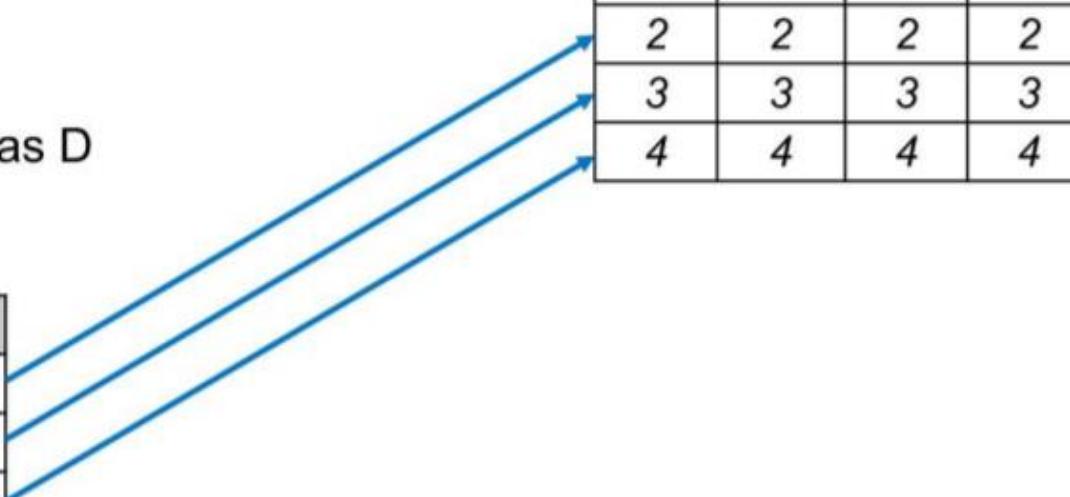
A	B	C	D
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4



UNION ALL

SELECT A, B, X as C, Y as D
FROM ... WHERE ...

A	B	X	Y
2	2	2	2
3	3	3	3
4	4	4	4



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'
```



Raw Data	
Filter pattern	
	id
	00000001
	00000001
	00000002
	00000003
	00000005
	00000006

```
@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



Raw Data	
Filter pattern	
AB	id
	00000001
	00000001
	00000002
	00000003
	00000005
	00000006

Exercise 03: CDS view with Left Outer Join

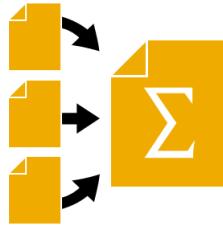


In this exercise you will:

- Use LEFT OUTER Join to display all the customer information with their total booking amount.
- See Student Manual, Exercise 3 Tasks 1-2

Time: 10 min

Lesson 4 – SQL Features in ABAP CDS Summary



You should now be able to:

- Understand SQL Expressions and Built-in-Functions
- Understand Join Types
- Understand UNION and UNION ALL

Unit 2 – Advanced Concepts

Lesson 1 – Input Parameter

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

View with Input Parameters

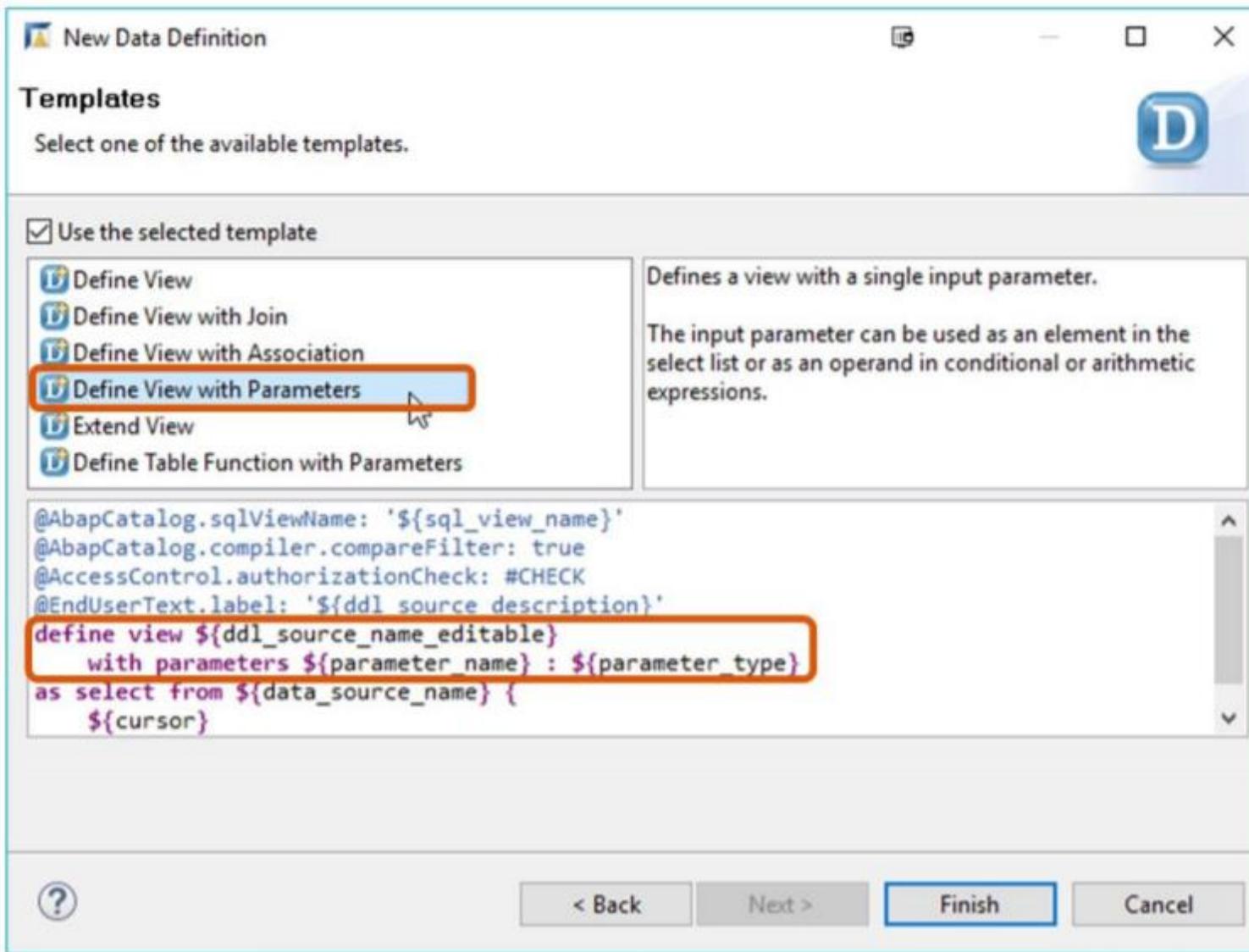
```
@AbapCatalog.sqlViewName: 'S4D430_PARAM'  
define view S4d430_Parameter  
with parameters  
    parameter1: abap.char(10),  
    parameter2: s_carr_id,  
@EndUserText.label: 'Discount'  
@EndUserText.quickInfo: 'Discount Factor for Price '  
    factor: abap.dec(3, 2),  
    separator: abap.char(1)  
@<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 for View with Input Parameters



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**

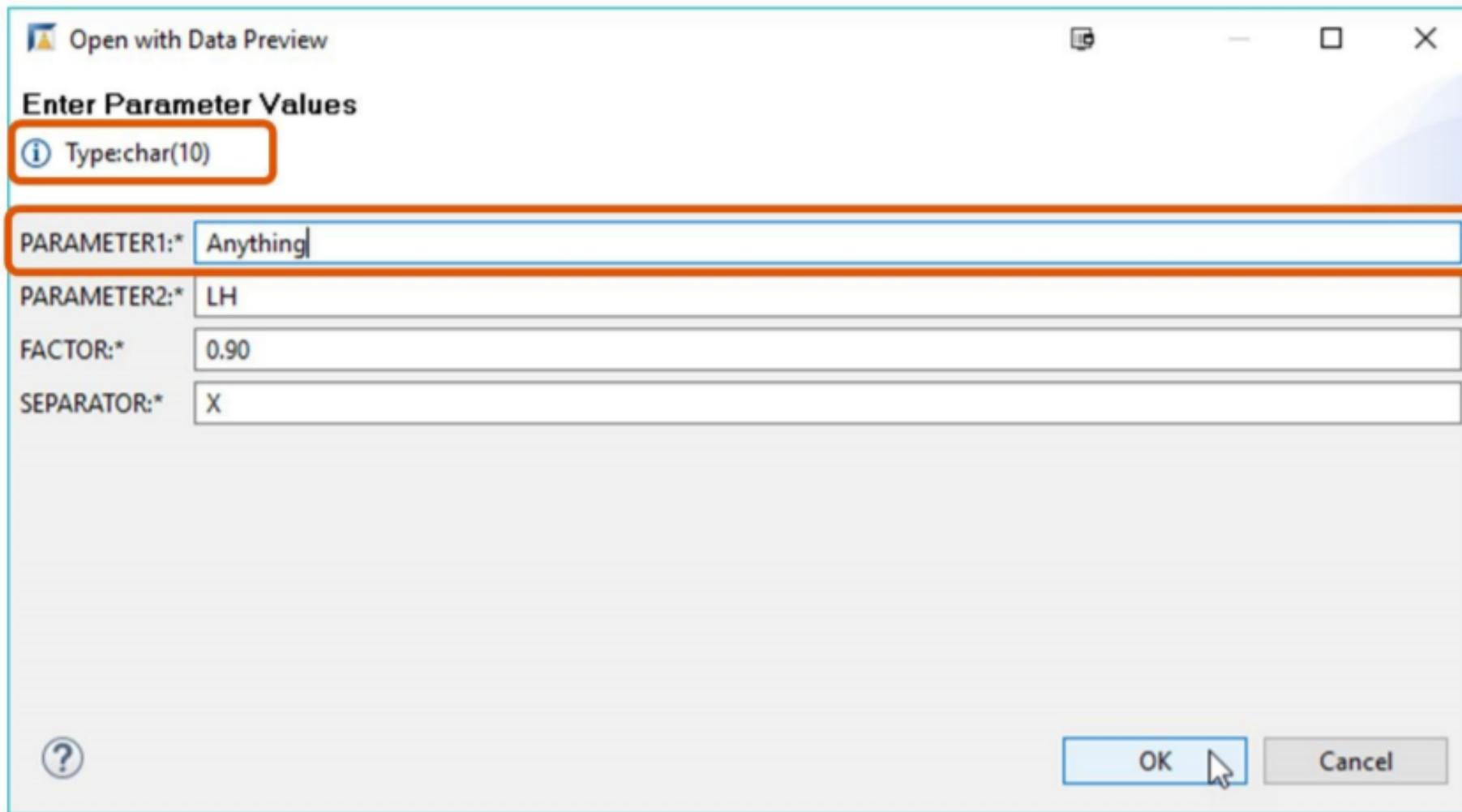
- Right side of a condition

```
@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)

Input Parameters in Data Preview



Forwarding Input Parameters to another CDS

```
@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 parentheses „()“

This parameter is
„forwarded“

Consume CDS View With Parameters in Open SQL

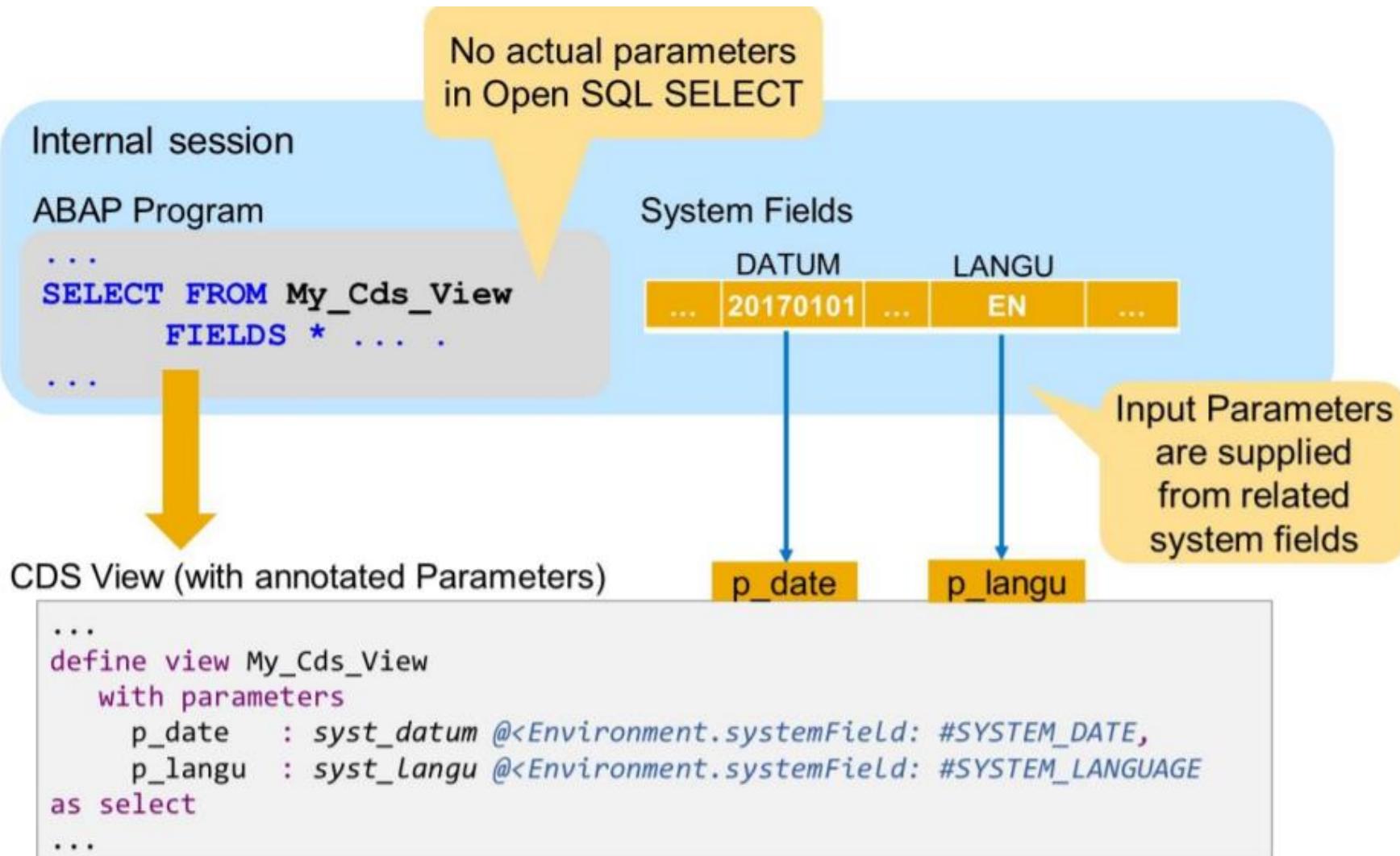
```
SELECT FROM s4d430_parameter(
    parameter1 = @(`Carrier` && pa_car),
    factor      = '0.93',
    parameter2 = @pa_car,
    separator   = 'X')
FIELDS *
WHERE carrid = @pa_car
INTO TABLE @gt_data.
```

Assignment with „=“
instead of „:“

Sequence of parameters
not fixed

Literals, host variables
or host expressions

Optional Parameters for CDS Views



Rules for Annotation Environment.systemField

- 5 different values

Value for Annotation systemField	Related ABAP system field
#CLIENT	<i>sy-mandt</i>
#SYSTEM_DATE	<i>sy-datum</i>
#SYSTEM_TIME	<i>sy-uzeit</i>
#SYSTEM_LANGUAGE	<i>sy-langu</i>
#USER	<i>sy-uname</i>

- 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

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

Session Variable	Related ABAP system field
<code>\$session.client</code>	<code>sy-mandt</code>
<code>\$session.system_date</code>	<code>sy-datum</code>
<code>\$session.system_language</code>	<code>sy-langu</code>
<code>\$session.user</code>	<code>sy-uname</code>

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

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

    }
```

Exercise 5: CDS View with Input Parameters



In this exercise you will:

- Create a CDS view with Input parameters
- Assign the input parameter to System field using Environment Annotation
- Use the input parameter in the filter path expression
- Write an ABAP program to consume the CDS view with parameter using Open SQL
- See Student Manual, Exercise 5 Tasks 1-5

Time: 15 min

Unit 2 – Advanced Concepts

Lesson 2 – Association

- **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”)

Template for View with Association

Templates

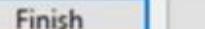
Select one of the available templates.

Use the selected template

 **Define View with Association**

The association can be used in the select list as well as by other CDS views which use this CDS view as a data source.

```
@AbapCatalog.sqlViewName: '${sql_view_name}'  
@AbapCatalog.compiler.compareFilter: true  
@AbapCatalog.preserveKey: true  
@AccessControl.authorizationCheck: #CHECK  
@EndUserText.label: '${ddl_source_description}'  
define view ${ddl_source_name editable} as select from ${data_source_name}  
association ${_1} to ${target_data_source_name} as ${_association_name}  
on $$projection.${element_name} = ${_association_name}.${target_element_name} {  
    ${cursor}  
    ${_association_name} // Make association public  
}
```

Association Instead of Join

View Definition with Association

```
@AbapCatalog.sqlViewName: 'S4D430_ASS01'  
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_ASS01'  
define view S4d430_Association_1 as select  
  from spfli association to scarr  
    on spfli.carrid = scarr.carrid  
  { key carrid,  
    key connid,  
    scarr.carrname  
 }
```

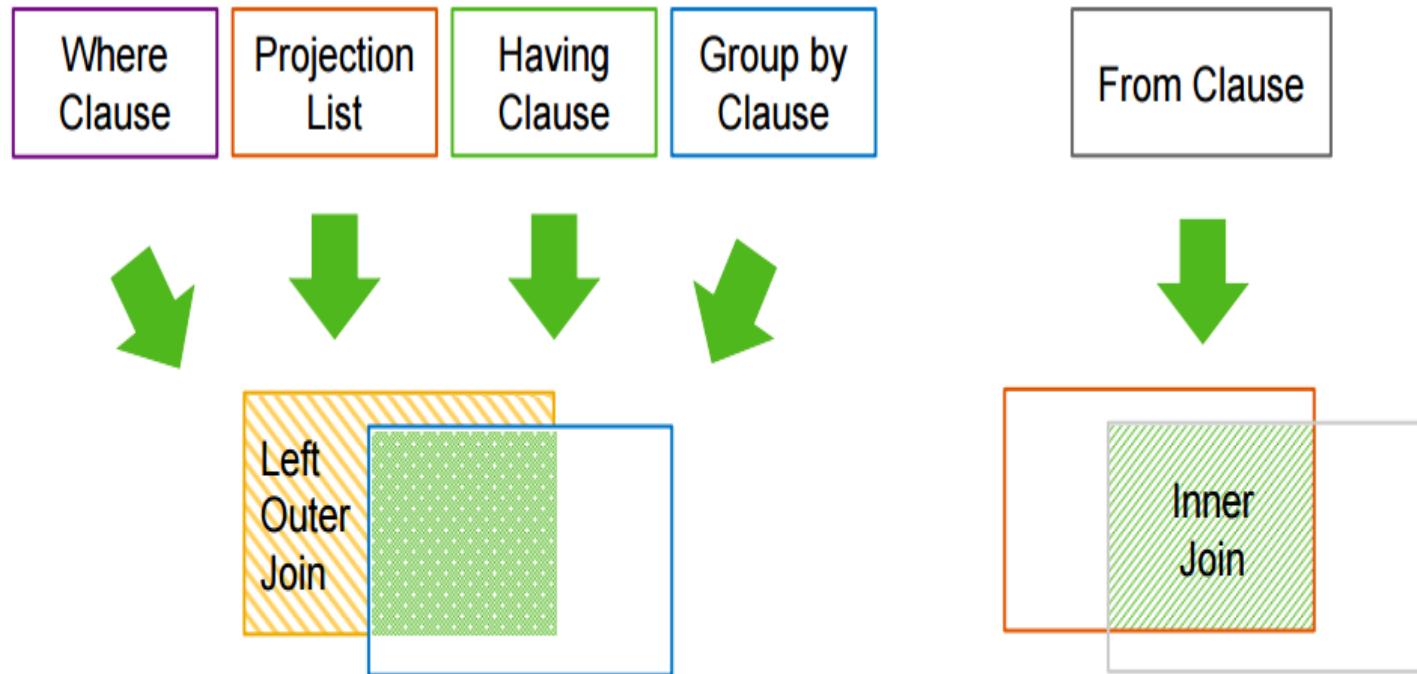
- Default join type can be changed to Inner Join

scarr[inner].carrname

Corresponding SQL Create Statement

```
CREATE VIEW "S4D430_ASS01" 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"  
)
```

CDS Associations – Translation into Joins



Used (!) associations are implicitly translated into SQL joins

Reuse of generated joins when semantically identical

Additional Semantics in Association

```
@AbapCatalog.sqlViewName: 'S4D430_ASS03'  
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

```
"=AU"."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_ASS03'

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)

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	
<input type="button" value="Filter pattern"/> <input checked="" type="checkbox"/> <input type="button" value="26 rc"/>	
carrid	connid
AA	0017

... and with field fldate

Raw Data		
Filter pattern		
	carrid	connid
	AA	0017

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

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  
    "C"."MANDT" AS "MA  
    "C"."CARRID",  
    "C"."CONNID",  
    "C"."CITYFROM",  
    "C"."CITYTO"  
FROM "SPFLI" "C"
```

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_ASSEXP" "C" LEFT OUTER JOIN "SCARR" "=A0" ON (
    "C"."MANDT" = "=A0"."MANDT" AND
    "C"."CARRID" = "=A0"."CARRID"
)

```

Join of SCARR,
but no join of SFLIGHT

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

Exposed Associations in Data Preview

The screenshot shows the SAP Data Preview interface for the dataset 'S4D430_ASSOCIATION_EXPOSED'. The main view displays a table with four columns: 'carrid' (values LH, UA, QF, JL, DL), 'connid' (values 2407, 2402, 0400, 0402, 0401, 3516, 0017, 1699, 0555, 0790, 0788, 3504), 'cityfrom' (values BERLIN, FRANKFURT, FRANKFURT, FRANKFURT, NEW YORK, NEW YORK, NEW YORK, NEW YORK, ROME, ROME, SAN FRANCIS...), and 'cityto' (values FRANKFURT, BERLIN, BERLIN, NEW YORK, NEW YORK, V YORK, J FRAN..., SINGAPORE, TOKYO, FRANKFURT, FRANKFURT, SAN FRAN..., SAN FRAN..., FRANKFURT, OSAKA, TOKYO, FRANKFURT). A context menu is open over the row with connid 0402, showing options: 'Quick filter on [FRANKFURT]', 'Distinct values for [cityfrom]', 'Follow Association', and 'Copy row'. The 'Follow Association' option is highlighted with a blue background and a cursor icon. A large yellow arrow points from this menu item to a separate 'List of Associations' panel on the right. This panel lists two associations: '_Carrier → scarr [0 .. 1]' and '_Flights → sflight [0 .. *]'. Below the list is a note: 'To follow the association, choose an association from'.

carrid	connid	cityfrom	cityto
LH	2407	BERLIN	FRANKFURT
LH	2402	FRANKFURT	BERLIN
LH	0400	FRANKFURT	NEW YORK
LH	0402	FRANKFURT	NEW YORK
UA		Quick filter on [FRANKFURT]	V YORK
UA		Distinct values for [cityfrom]	J FRAN...
QF		Follow Association	SINGAPORE
JL			TOKYO
DL			FRANKFURT
LH	0401	NEW YORK	FRANKFURT
UA	3516	NEW YORK	FRANKFURT
AA	0017	NEW YORK	SAN FRAN...
DL	1699	NEW YORK	SAN FRAN...
AZ	0555	ROME	FRANKFURT
AZ	0790	ROME	OSAKA
AZ	0788	ROME	TOKYO
IIA	3504	SAN FRANCIS...	FRANKFURT

Use Exposed Associations in Open SQL

```
SELECT FROM s4d430_association_exposed AS c
FIELDS carrid,
connid,
\Carrier-carrname,
c~\Carrier-currcode
WHERE c~\Carrier-currcode = 'USD'
INTO TABLE @gt_data.
```

Prefix „\“ for all association names

Component selector „-“
(in CDS DDL it is „..“)

Optional: With Alias „c“ for the data source

Path expressions supported in all clauses

DDL Source

```
...
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>
...
```



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

Filtered Associations

Filtered Association

Associations can be enhanced by adding a supplementary filter

- Define association of cardinality “to-N”
- Use filter expressions in square brackets
- Usage of attribute 1:
(Declaration of monovalency)

```
define view S4hcds_Para_Demo2 as
  select from scustom as customer
    association[0..*] to sbook as _bookings
      on customer.id = _bookings.customerid
  { customer.id,
    customer.name,
    // Exposed Association with filter
    _bookings[ class = 'C' ] as _business_flights,
    // Exposed Association without filter
    _bookings }
```

```
define view S4hcds_Para_Demo3 as
  select from S4hcds_Para_Demo2
  { id,
    name,
    // Path expressions with filter condition and 1:
    _bookings[1: fldate = '20150101'].carrid
      as ny_booking_carrid,
    _bookings[1: fldate = '20150101'].connid
      as ny_booking_connid
  }
```

- **Allowed Operators**

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

- **Left-hand Side**

- Must be a field of association target

- **Right-hand Side**

- field of association target
 - literal
 - parameter
 - session variable

Cardinality in Filtered Associations

```
@AbapCatalog.sqlViewName: 'S4D430_FILTCARD'  
@EndUserText.label: 'Demo: Cardinality in Filtered Asso.'  
define view S4D430_Acco_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>
...
...
```

Exercise 4: CDS View with Association



In this exercise you will:

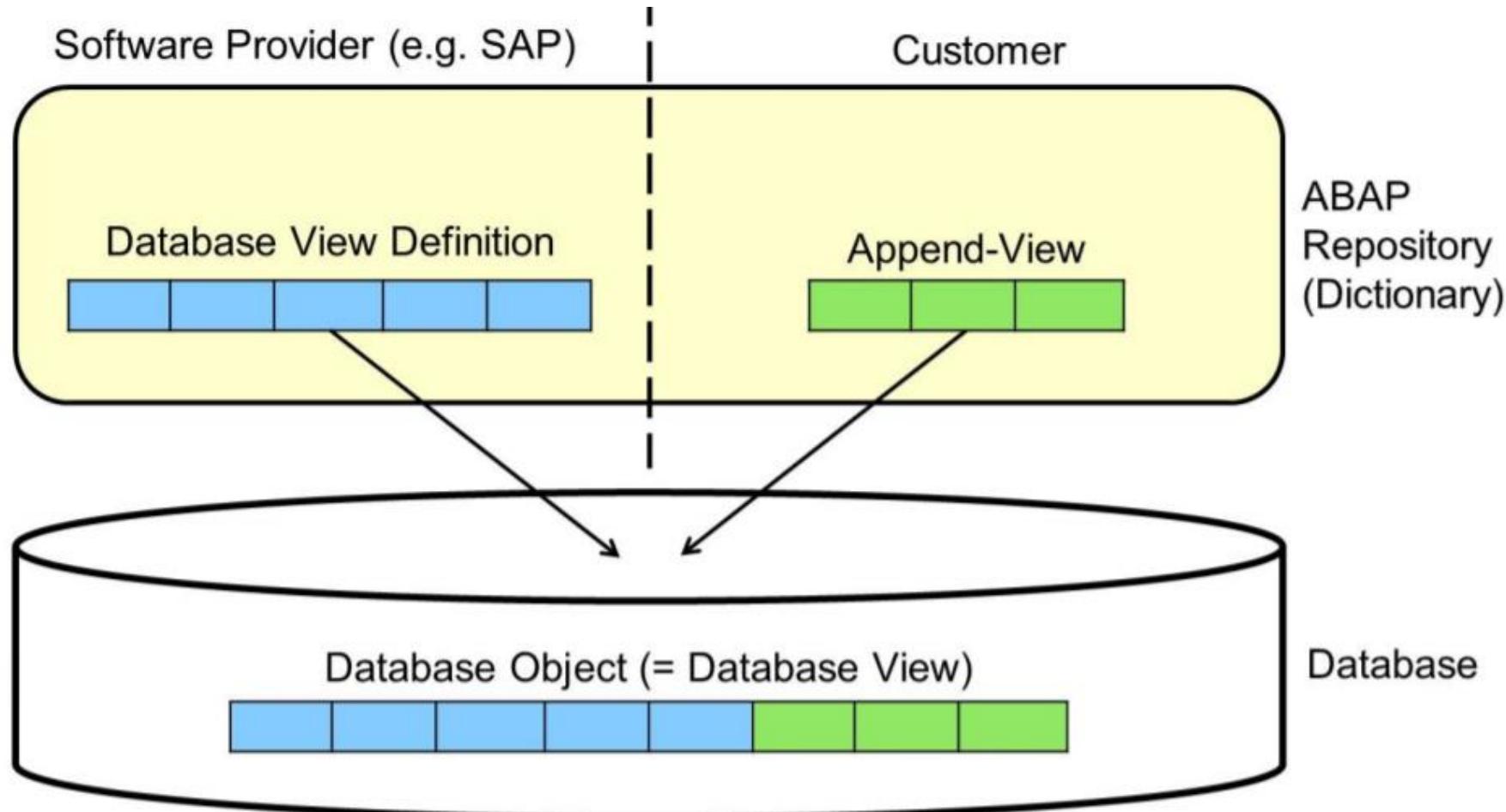
- Use association to define relationship with two tables
- Perform aggregations on the fields from the association target table
- Expose an association to consume from outside
- Consume the exposed association using path expression in Open SQL
- See Student Manual, Exercise 4 Tasks 1-5

Time: 10 min

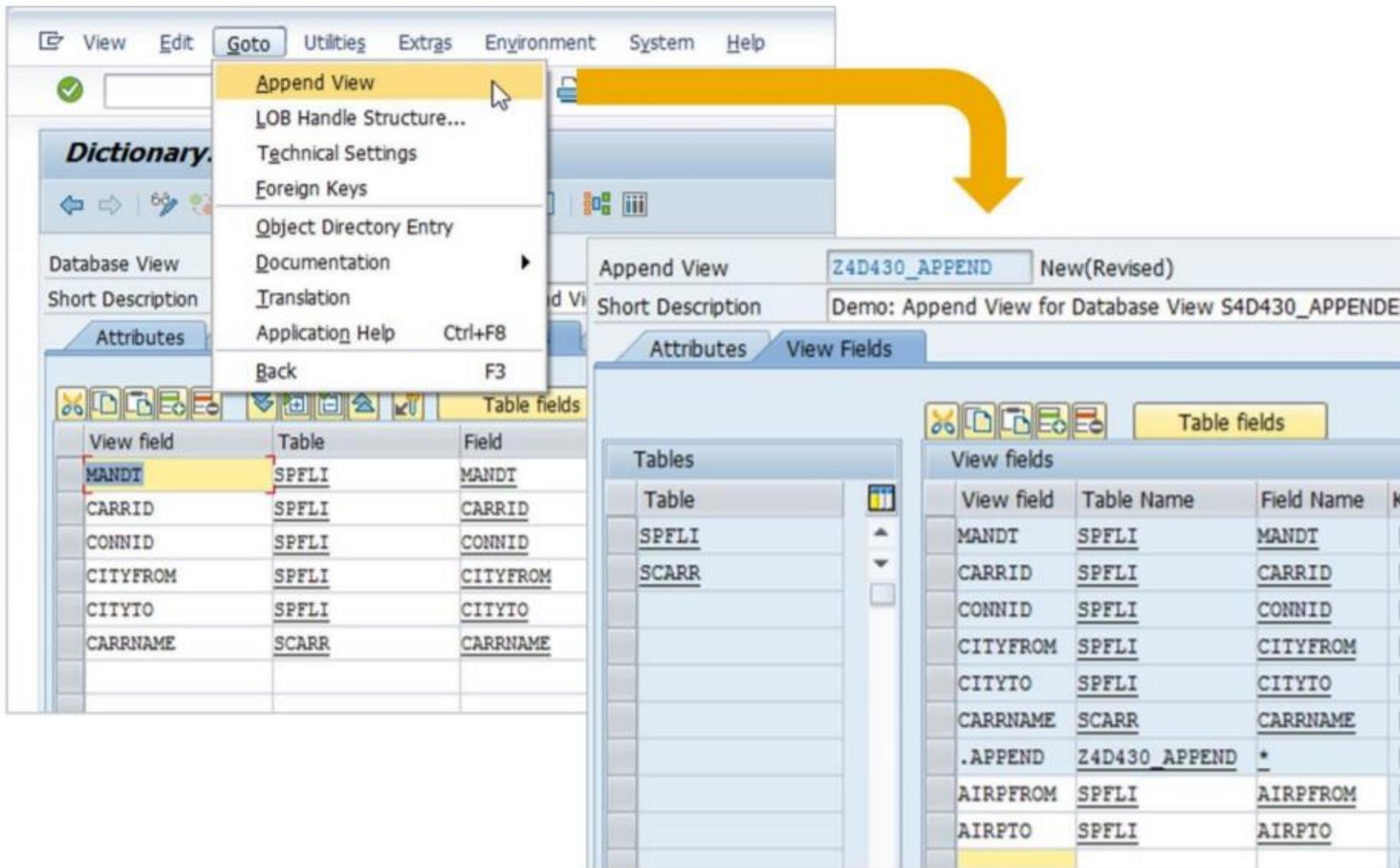
Unit 2 – Advanced Concepts

Lesson 3 – CDS View Extensions

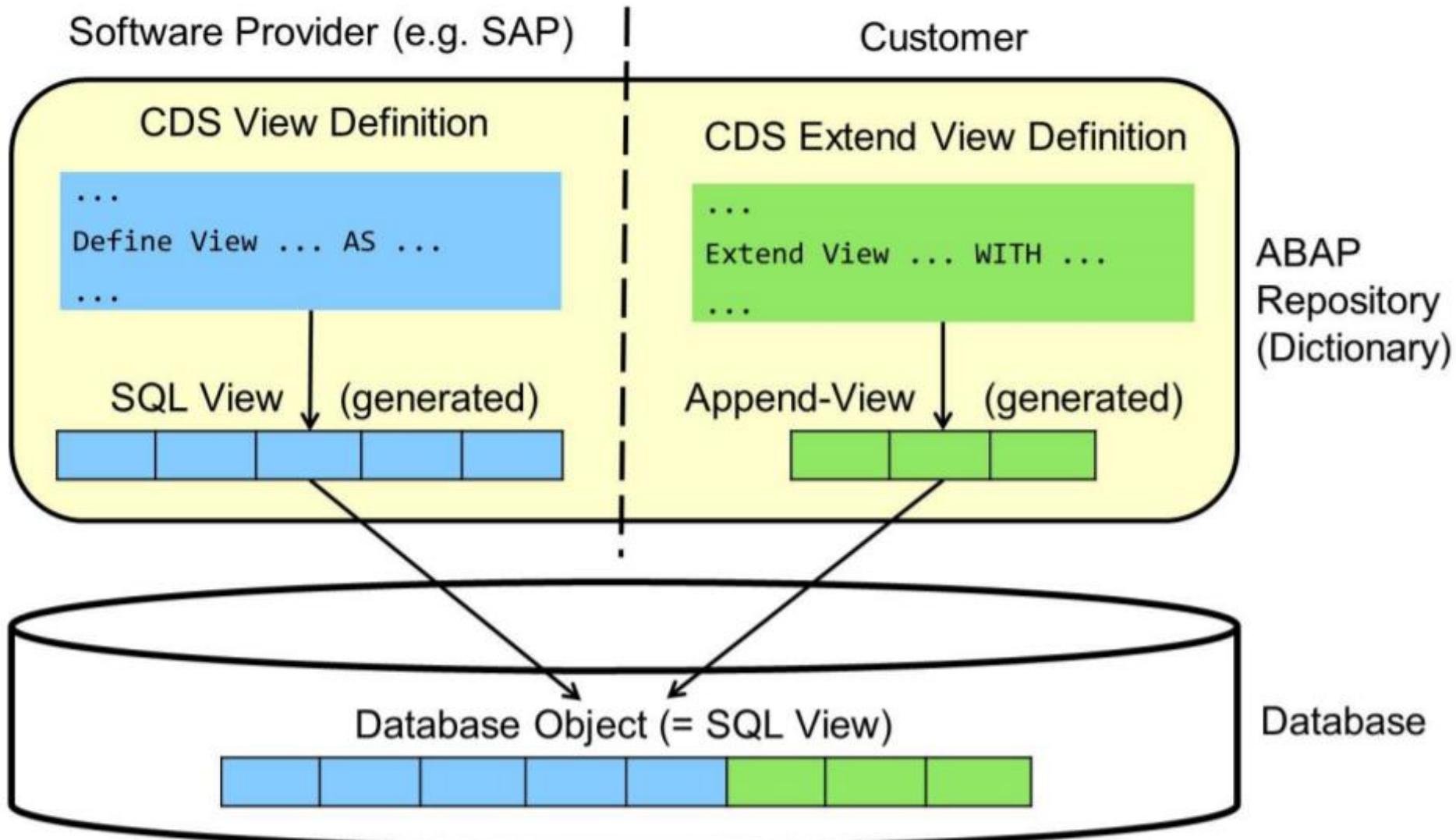
Append Views for Classical Database Views



Creating a Classical Append View



CDS View Extensions



DDL Source 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  
    }
```

DDL Source 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

What can you do in Extension

- **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

DDL Source 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

CDS View Extension with GROUP BY and UNION

DDL Source With View Extension (Created by Customer)

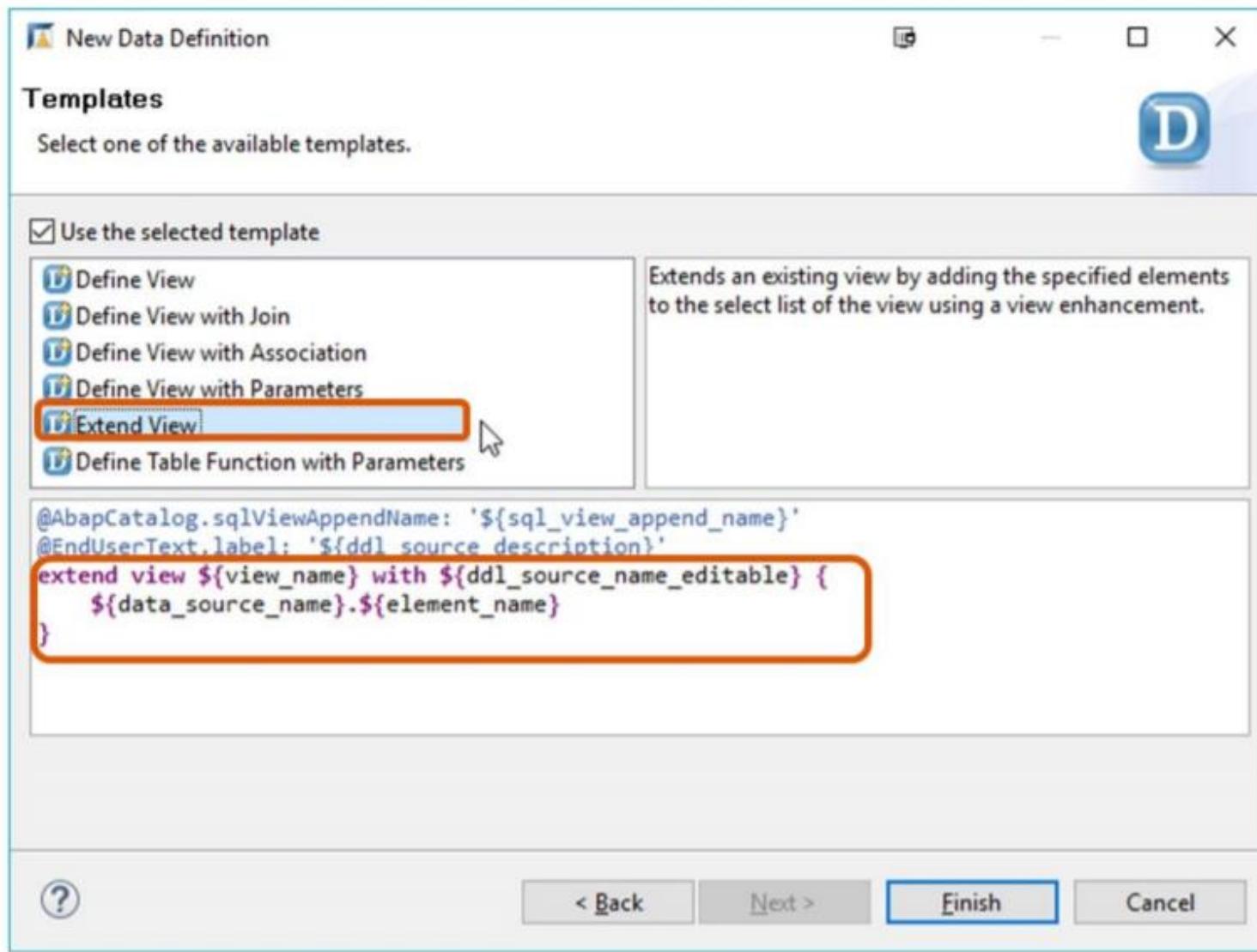
```
@AbapCatalog.sqlViewAppendName: 'S4D430_EXTENDC'  
extend view S4D430_Extend_Target_C  
with S4D430_EXTEND_VIEW_C  
  
{ city,  
  country,  
  max( _Booking.fldate ) as last_booking  
}  
group by city,  
        country  
  
union  
  
{ city,  
  country,  
  max( _Booking.fldate ) as last_booking  
}  
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 for Extend View



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**

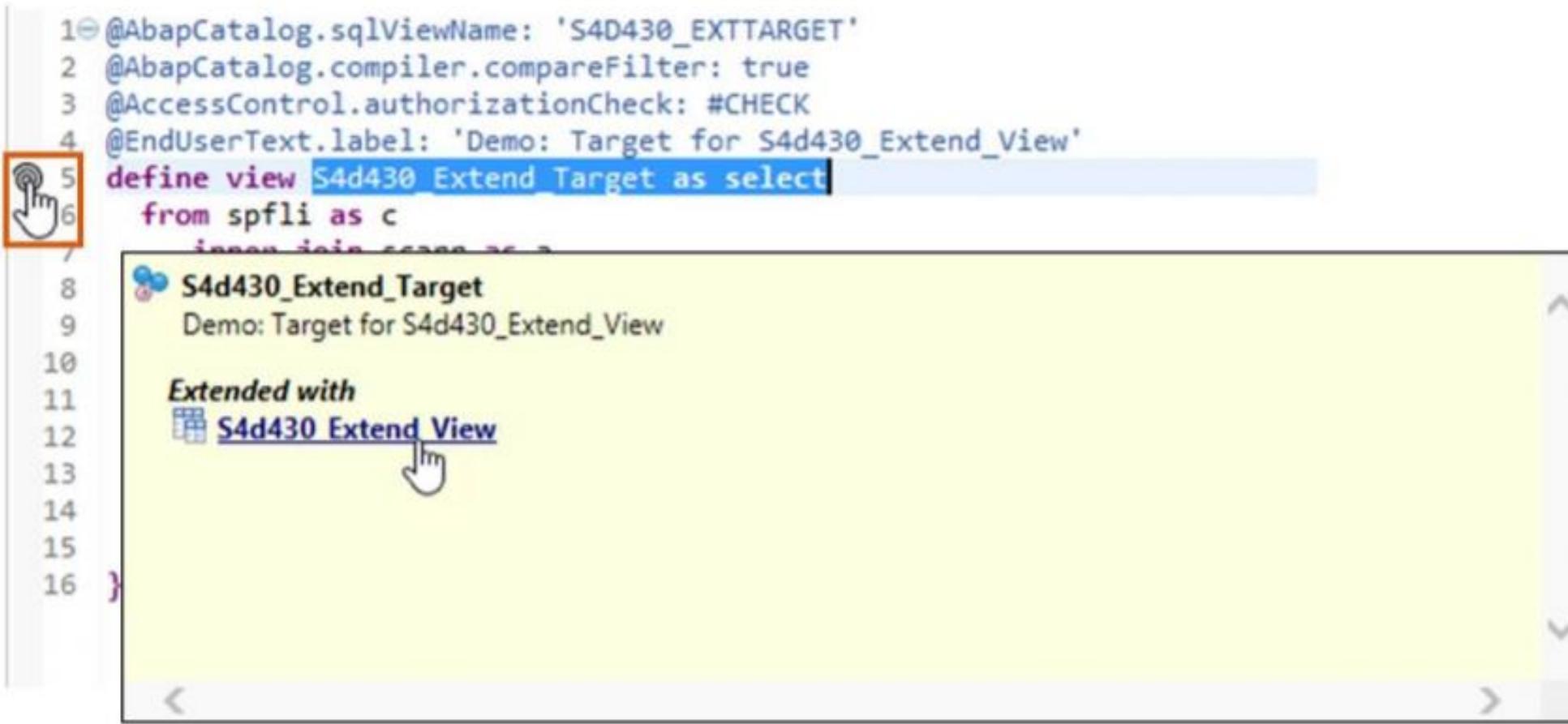
- #NONE cannot be combined with other values
 - #GROUP BY and #UNION require #PROJECTION_LIST

- **Default Value**

- #PROJECTION_LIST

Extended View in DLL Source Editor

```
1 @AbapCatalog.sqlViewName: 'S4D430_EXTTARGET'  
2 @AbapCatalog.compiler.compareFilter: true  
3 @AccessControl.authorizationCheck: #CHECK  
4 @EndUserText.label: 'Demo: Target for S4d430_Extend_View'  
5 define view S4d430_Extend_Target as select  
6   from spfli as c  
7     inner join scana as a  
8       S4d430_Extend_Target  
9         Demo: Target for S4d430_Extend_View  
10  
11       Extended with  
12         S4d430_Extend_View  
13  
14  
15  
16 }
```



Exercise 6: Extend CDS Views



In this exercise you will:

- Extend an existing CDS view to add additional fields to the projection list
- See Student Manual, Exercise 6 Tasks 1-2

Time: 10 min

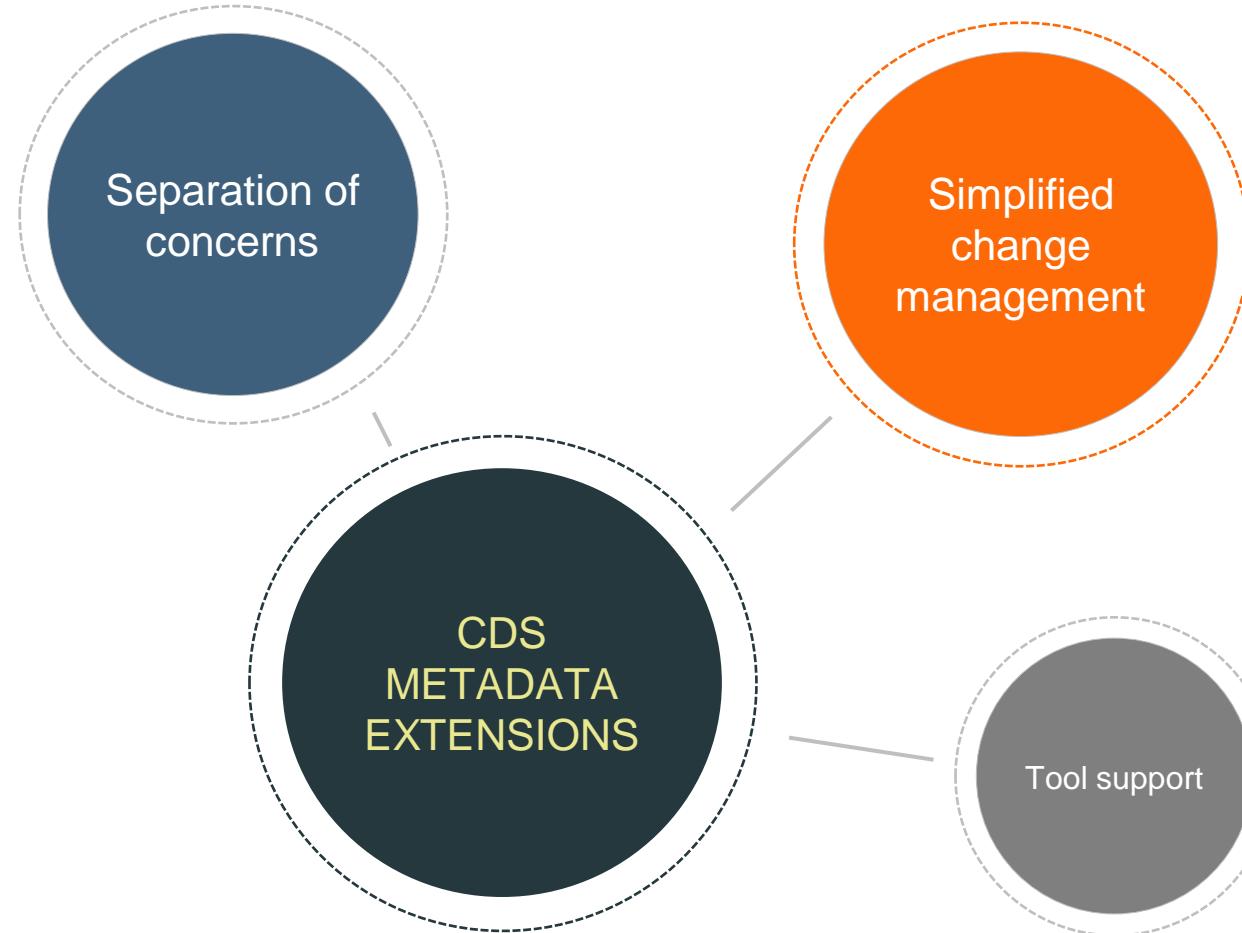
Unit 2 – Advanced Concepts

Lesson 4 – Metadata Extension

CDS metadata extensions (MDE)

Keep view definition distinct from domain specific annotations

Use one view definition with various sets of metadata (e.g. SAP, partner, customer or industry specific)



Change annotations without modifying underlying CDS view
No need to change original (SAP) views when metadata changes

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

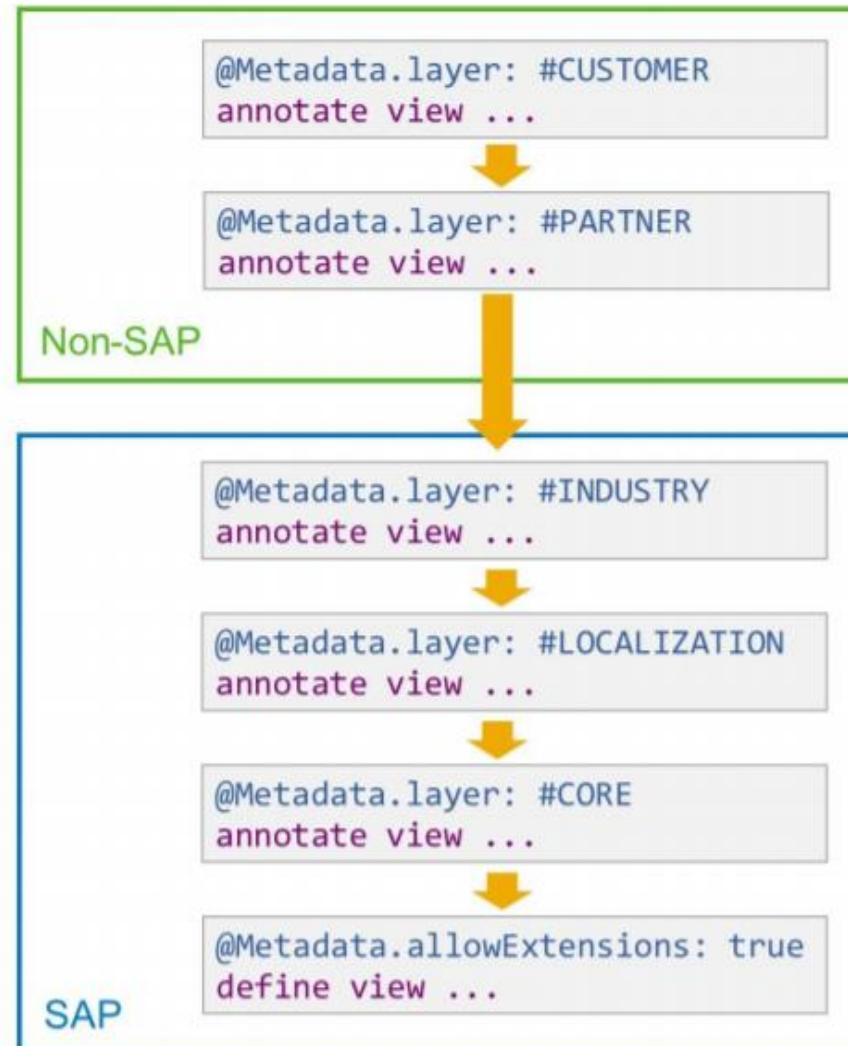
DDL Source With View Definition

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



Metadata Extensions for a CDS view

Metadata extensions of a given CDS entity can be accessed by clicking the corresponding extension symbol next to "define" statement.

```
①2 define view ZJH_Product_2
  ZJH_Product_2 (View)
  test

Metadata extended with
  ZJH_PRODUCT
  ZJH_PRODUCT_PARTNER
```

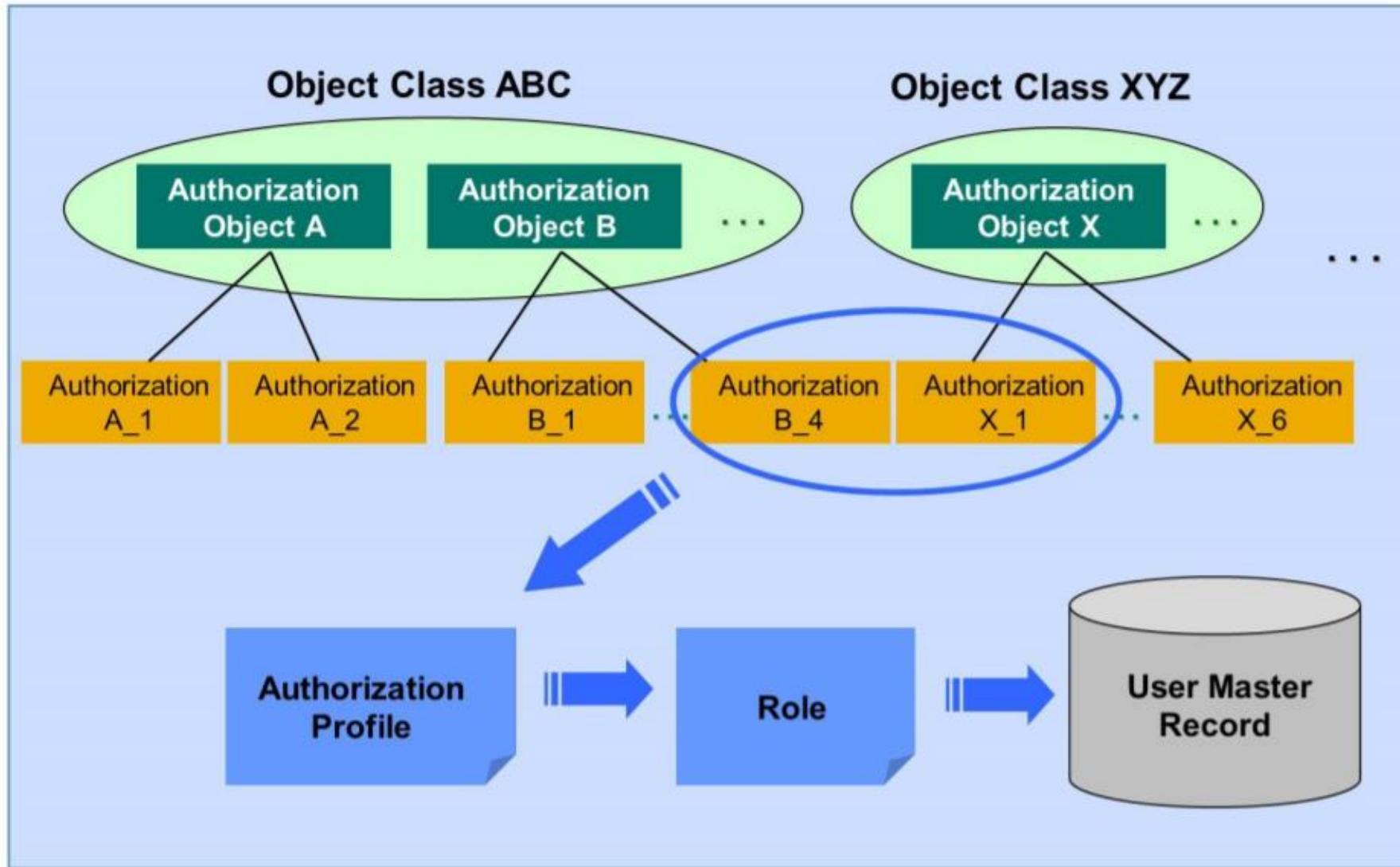
In order to get an overview about the effect of a metadata extension and the origin of an active annotation, use the ADT context menu function "Open With => Active Annotations"

Annotated Elements	Annotation Value	Translated Text	Origin Data Source	Origin Data Element
lineItem[1].importance	#MEDIUM		ZJH_PRODUCT_PARTNER (Metadata Extension)	
@ObjectModel.text.association	'_Text'		SEPM_I_Product_E (CDS View)	
@EndUserText.quickInfo		EPM: Product ID	SNWD_PD (Database Table)	SNWD_PRODUCT_ID
label		Product ID	SNWD_PD (Database Table)	SNWD_PRODUCT_ID
heading		Product ID	SNWD_PD (Database Table)	SNWD_PRODUCT_ID
Product2.@UI.lineItem[1].importance	#LOW		ZJH_PRODUCT (Metadata Extension)	

Unit 2 – Advanced Concepts

Lesson 5 – Metadata Extension

Authorization Objects, Profiles, and Roles



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

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

- **General Problems:**

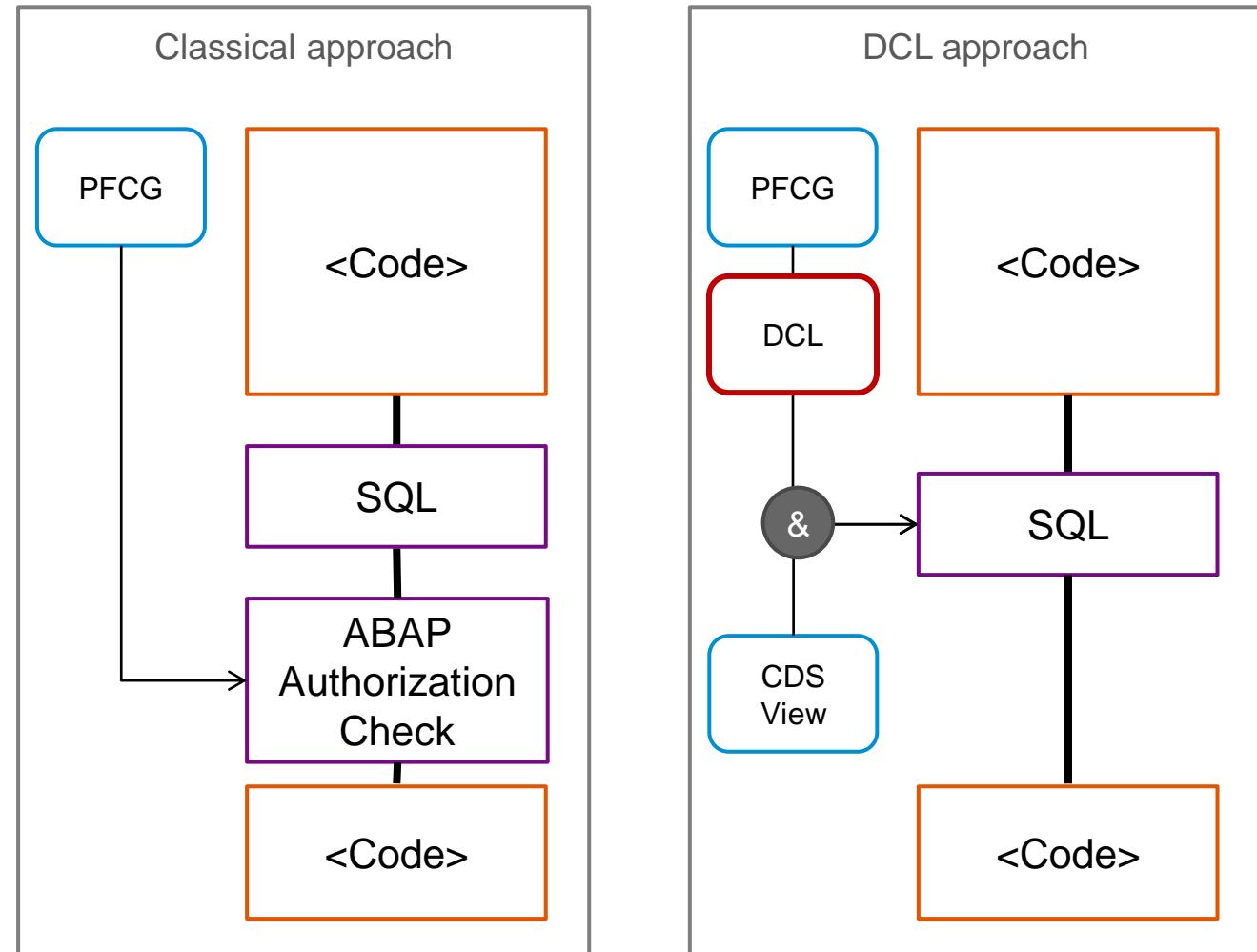
- No visible link between data and authorization objects
- Developer can forget (or ignore) necessary checks
- Users with debugging rights can bypass authorization checks

- **Problems related to Code-to-Data:**

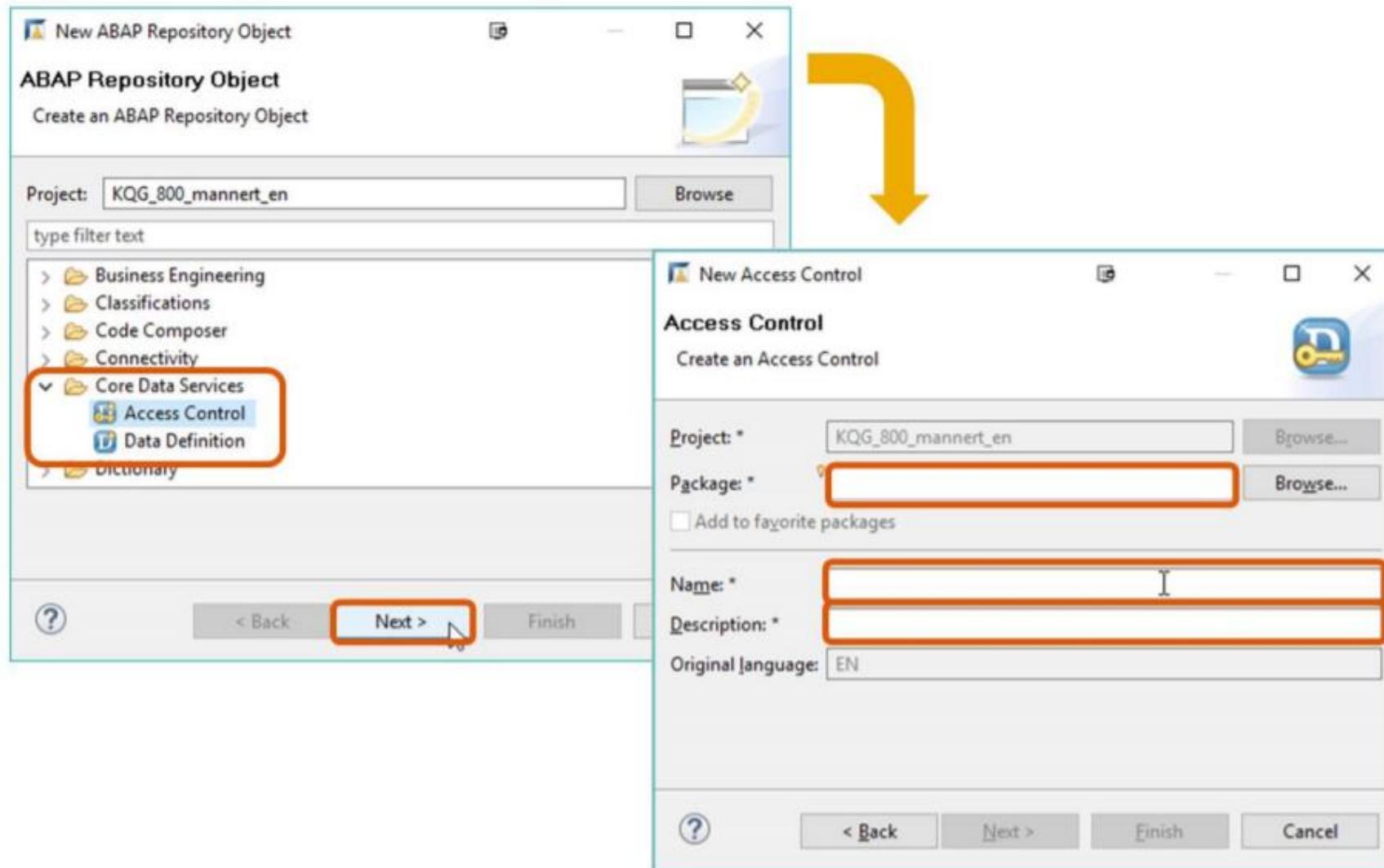
- All authorization checks on application level
- Some calculations cannot be done on the database because they require an authorization check between steps

Data Control Language Overview

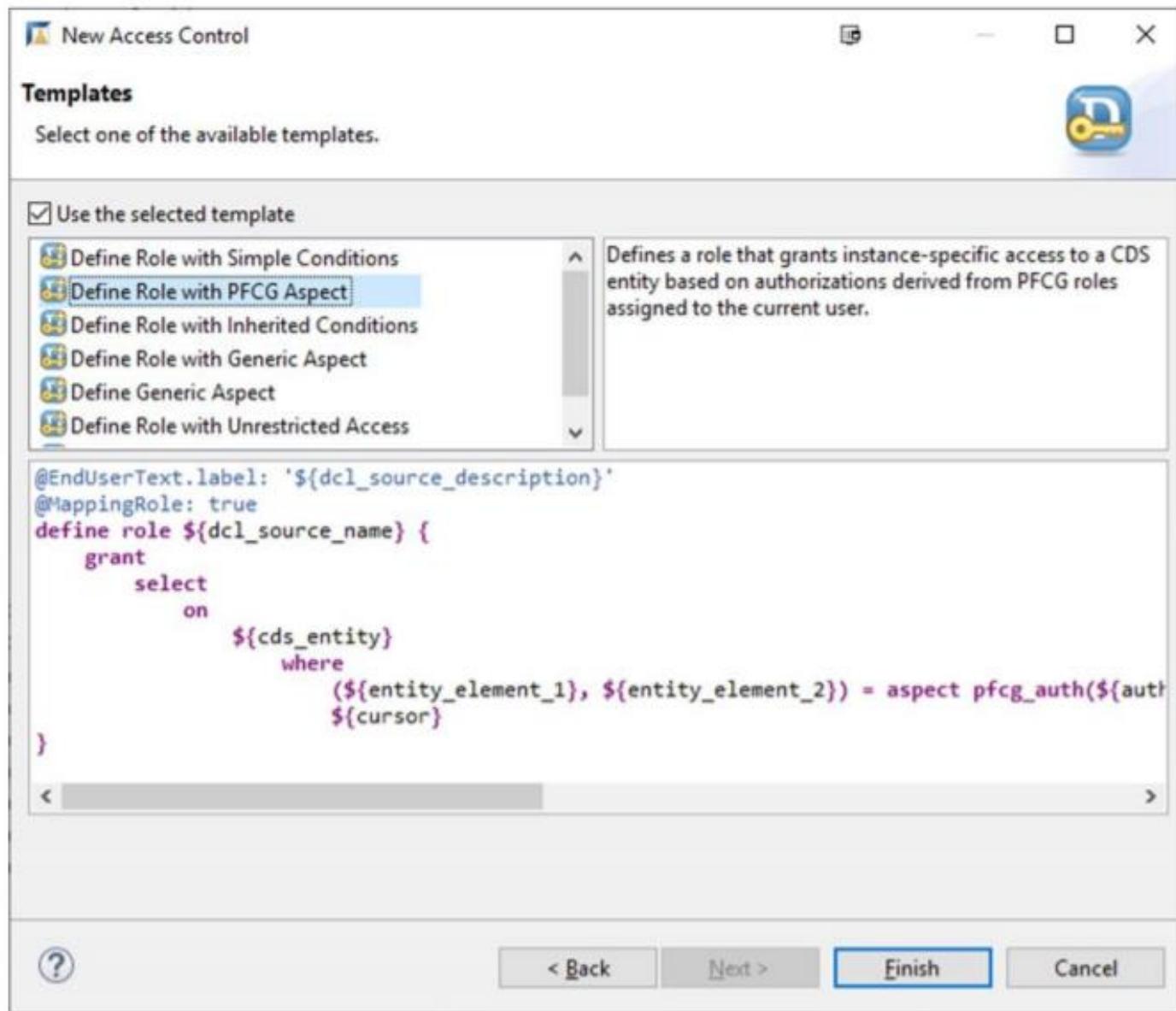
- Declarative approach instead of coded
- Based on CDS modeling objects and therefore part of the data-model
- Authorizations are also pushed down to DB by extending the Open SQL SELECT statement
- Authorizations are only defined once and automatically (re-) used everywhere



Create New DCL Source



Templates for Access Controls



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

Key word DEFINE
is optional

- Statement **DEFINE ROLE**

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

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

At least one
condition per
GRANT statement

Delimiter „;“ mandatory
after each
GRANT statement

- **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



An aspect can be used to define user-dependent values which can be compared to one or more columns of the data model

- PFCG_AUTH aspect is a pre-delivered aspect that links the object model not to a user attribute but to his ABAP PFCG authorizations
- USER aspect can be used to compare the current logged on user with column of the date model

```
@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'  
                                                 );  
}
```

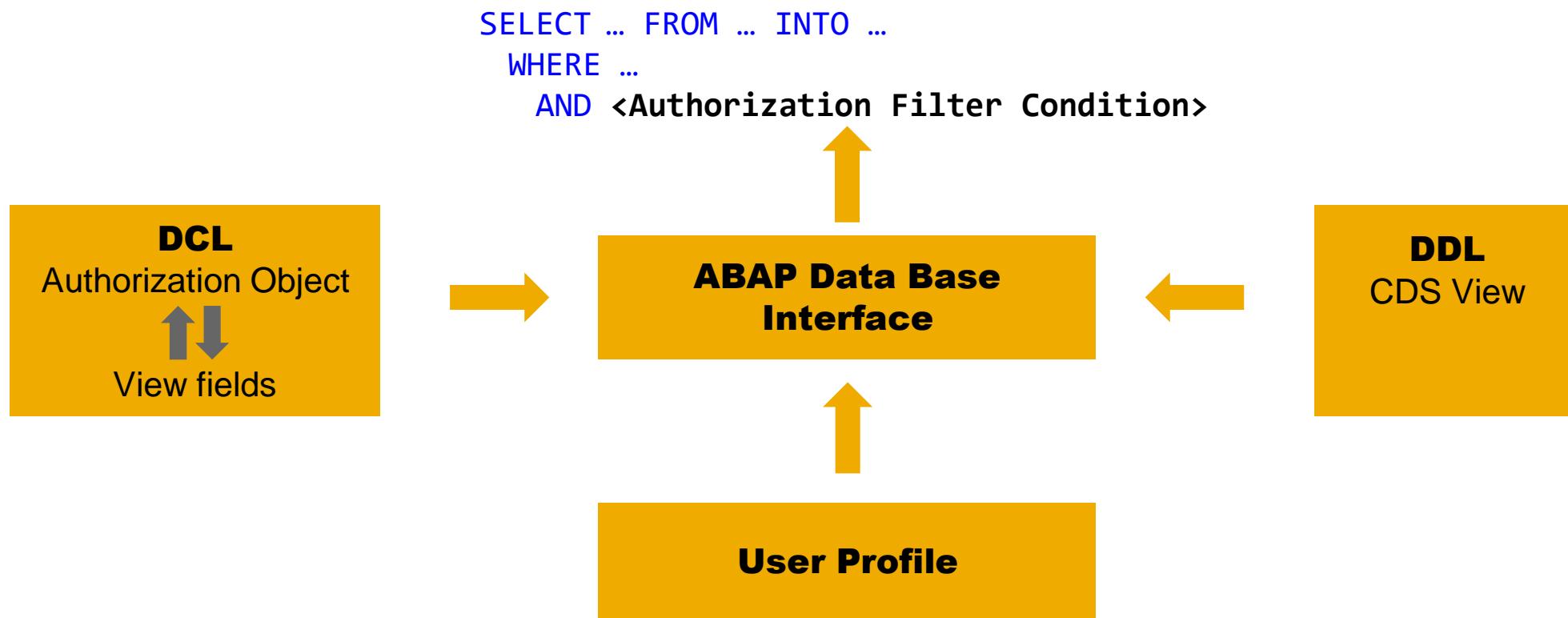
Left-hand side:
One or more comma-separated
view fields in brackets

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

Arguments of pfcg_auth:
One authorization object
with its fields

Runtime Behavior

During runtime the SELECT statement for a CDS view will automatically be enhanced with the authorization filter.



Annotations in DCL Sources

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

Annotations in DDL Sources

- **@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

Most Important Annotations For Access Control Objects

Annotations in DCL Sources

- `@EndUserText.label`: Translatable short text of role
- `@MappingRole`: The role is assigned to all users automatically regardless of the client. Only `true` supported

```
@EndUserText.label: 'Auto assigned mapping role ...'  
@MappingRole: true
```

```
define role <role name> { ... }
```

Annotations in DDL Sources

- `@AccessControl.authorizationCheck`: Defines access control when Open SQL is used to access the CDS view:
 - `#CHECK`: Perform access control, syntax check warning if no role is assigned
 - `#NOT_REQUIRED`: Like `#CHECK`, but suppress syntax check warning
 - `#NOT_ALLOWED`: Perform no access control, syntax check warning if DCL source is assigned
 - `#PRIVILEGE_ONLY`: Perform When DCL0 is assigned to CDS, Privileged only can access the data

```
...  
@AccessControl.authorizationCheck: #NOT_REQUIRED
```

```
...
```

```
define view <view name> as ...
```

Example of a DCL Role

Authorization Object

Object	v_VBAK_AAT
Text	Sales Document: Authorization for Sales Document Types
Class	SD Sales and Distribution
Author	SAP
Authorization fields	
Authorization Field	Short Description...
AUART	Sales Document Type
ACTVT	Activity

Object	v_VBAK_VKO
Text	Sales Document: Authorization for Sales Areas
Class	SD Sales and Distribution
Author	SAP
Authorization fields	
Authorization Field	Short Description...
VKORG	Sales Organization
VTWEG	Distribution Channel
SPART	Division
ACTVT	Activity

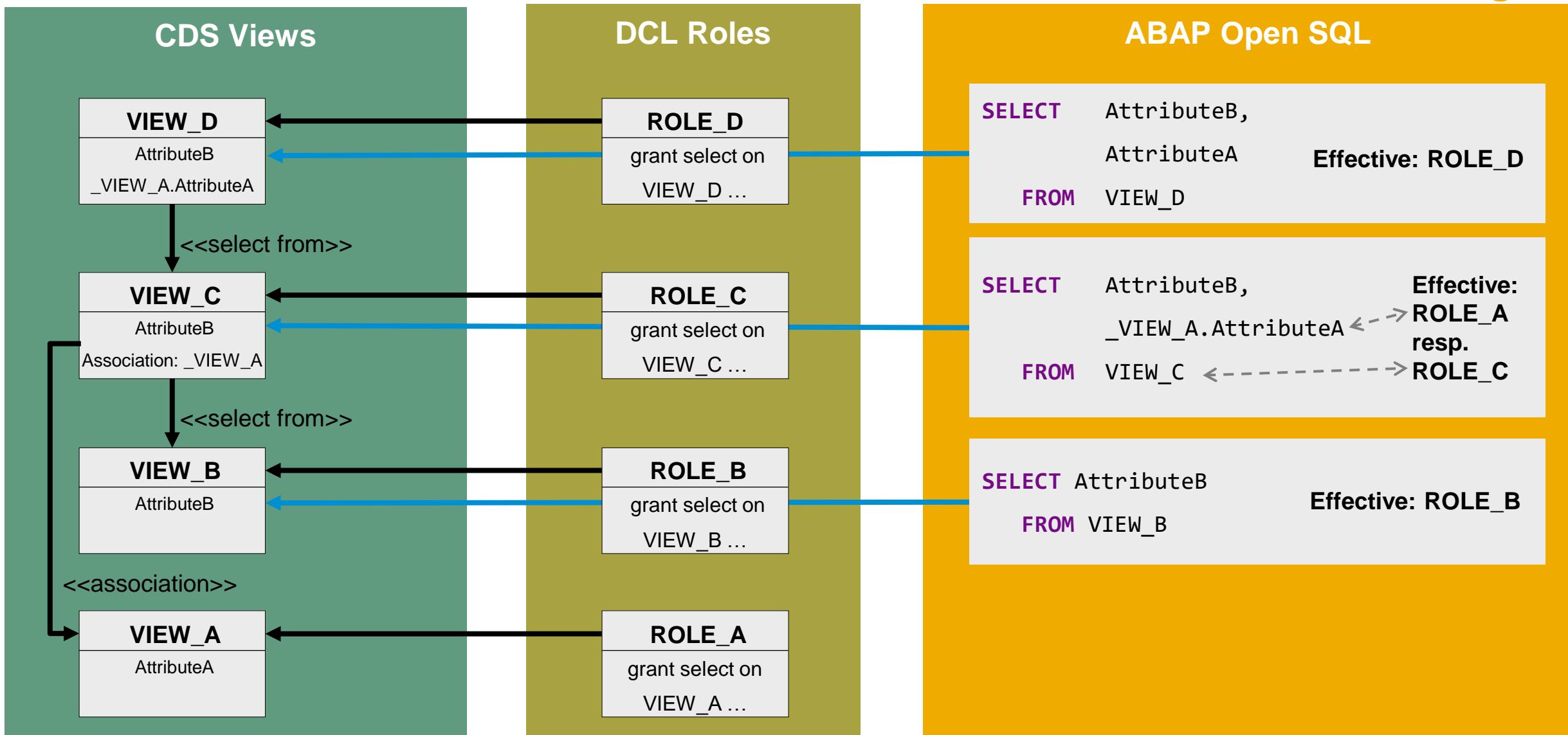
DCL Role

```
@MappingRole: true
define role I_SalesOrder {
    grant select on I_SalesOrder
    where
        ( SalesOrderType
            = aspect pfcg_auth
            ( v_vbak_aat,
                auart,
                actvt = '03' )
        )
    and
        ( SalesOrganization,
            DistributionChannel,
            OrganizationDivision
            = aspect pfcg_auth
            ( v_vbak_vko,
                vkorg,
                vtweg,
                spart,
                actvt = '03' );
}
```

CDS View

```
@AccessControl:{ authorizationCheck:#CHECK}
define view I_SalesOrder
as select from ...
{
    key SalesOrder,
    ...
    SalesOrderType,
    ...
    SalesOrganization,
    DistributionChannel,
    OrganizationDivision,
    ...
}
```

Which DCL Roles are Evaluated When?



Testing for DCL Transaction SACM

The screenshot shows the SAP Access Control Management interface with the following sections:

- Access Control Management - Quicklinks**: Includes links for DCL Documents, ABAP Artifacts, Tools, Designtime, Admintime, and Runtime.
- Designtime**: Sub-links for DCL Documents (read-only mode), ABAP Artifacts (Aggregates + Aspects, Roles (simple-view), Access Conditions, Authorization Views), and Tools (Show Aspect-Proposalvalues of an User, DDL Infocenter).
- Admintime**: Sub-links for User-Role-Assignments (overview) and User-Role-Assignments (per user).
- Runtime**: Sub-link for ACM Runtime Tool (for SELECT).

ACM-Tool to show DCL-authorizations on SELECT

Merge authorizations into an SQL SELECT statement

Variant: 0: No merging is done at all
 Variant: 1,2,4: Merging is done by ACM-internal coding (in ABAP)
 Variant: 8: Merging is done by SSDL (in ABAP)
Variant: 16: Merging is done by Open-SQL (in the Kernel) (non-productive coding)

User, whose authorizations are checked and applied

- Own user (logged-in user): LUCIUS
- ACMTST testuser: ACMTST_UP1 ... (password will be created automatically)
- Other user: LUCIUSTESTER** Password: *****

How to execute the SELECT statement

- Execute method: IF ACMTOOL EXECSQL~SELECT of class CL_ACMTOOL_EX02_SO
- SELECT * FROM <CDS-Entity>**: Z_Test_SlsOrdFlfmtSalesOrderQ

View-Parameters

Entity-/View-Parameters: **(p_language = 'E')**
 for example: (PTAX = 19, PLANGUAGE = 'E')

Annotations:

- Select variant 16 for test with open SQL
- Test with your logged-in user or execute with different user.
- Enter the name of the CDS view.
- Enter the name value pairs for the parameters used in the view.

Testing for DCL Result in Transaction SACM

SELECT for User: LUCIUSTESTER - Variant:16, Execute-type(2)

SQL-Trace (for more information, see transaction: ST05)

```

SELECT
  "TSOFLSORDQ" . *
FROM
  /* Z_TEST_SLSORDFLMTSALESORDER */ "TSOFLSORDQ" ("P_LANGUAGE" => 'E') "TSOFLSORDQ"
INNER JOIN /* ACM_AUTHVIEW_70750ECA07 */ "ACMAUT70750E
  CAA7" ("P_LANGUAGE" => 'E') "V0" ON "TSOFLSORDQ" . "MANDT" = "V0" . "MANDT" AND (
  "V0" . "BILLINGBLOCKREASONDESCRIPTION" = "TSOFLSORDQ" .
  "BILLINGBLOCKREASONDESCRIPTION" IS NULL AND
  "TSOFLSORDQ" . "BILLINGBLOCKREASONDESCRIPTION" IS NULL AND
  "V0" . "BILLINGDOCUMENTDATE" = "TSOFLSORDQ" . "BILLINGDOCUMENTDATE" OR "V0" .
  "BILLINGDOCUMENTDATE" IS NULL AND "TSOFLSORDQ" . "BILL
  INGDOCUMENTDATE" IS NULL AND ("V0" . "BILLTOPARTY" = "TSOFLSORDQ" . "BILLTOPARTY" OR
  "V0" . "BILLTOPARTY" IS NULL AND "TSOFLSORDQ" . "BIL
  LTOPARTY" IS NULL) AND ("V0" . "BILLTOPARTYNAME" = "TSOFLSORDQ" . "BILLTOPARTYNAME"
  OR "V0" . "BILLTOPARTYNAME" IS NULL AND "TSOFLSORDQ" .
  "BILLTOPARTYNAME" IS NULL) AND ("V0" . "COMPLETEDDELIVERYISDEFINED" = "TSOFLSORDQ" .
  "COMPLETEDDELIVERYISDEFINED" OR "V0" . "COMPLETEDDELIVER
  YISDEFINED" IS NULL AND "TSOFLSORDQ" . "COMPLETEDDELIVERYISDEFINED" IS NULL) AND (
  "V0" . "CREATEDBYUSER" = "TSOFLSORDQ" . "CREATEDBYUSER" O
  R "V0" . "CREATEDBYUSER" IS NULL AND "TSOFLSORDQ" . "CREATEDBYUSER" IS NULL) AND (
  "V0" . "CREATIONDATE" = "TSOFLSORDQ" . "CREATIONDATE" OR
  "V0" . "CREATIONDATE" IS NULL AND "TSOFLSORDQ" . "CREATIONDATE" IS NULL) AND ("V0" .
  "CREATIONTIME" = "TSOFLSORDQ" . "CREATIONTIME" OR "V
  0" . "CREATIONTIME" IS NULL AND "TSOFLSORDQ" . "CREATIONTIME" IS NULL) AND ("V0" .
  "CUSTOMERPAYMENTTERMS" = "TSOFLSORDQ" . "CUSTOMERPAYMEN
  TTERMS" OR "V0" . "CUSTOMERPAYMENTTERMS" IS NULL AND "TSOFLSORDQ" .
  "CUSTOMERPAYMENTTERMS" IS NULL) AND ("V0" . "CUSTOMERPAYMENTTERMSNAME"
  = "TSOFLSORDQ" . "CUSTOMERPAYMENTTERMSNAME" OR "V0" . "CUSTOMERPAYMENTTERMSNAME" IS
  NULL AND "TSOFLSORDQ" . "CUSTOMERPAYMENTTERMSNAME" IS
  NULL) AND ("V0" . "DELIVERYBLOCKREASON" = "TSOFLSORDQ" . "DELIVERYBLOCKREASON" OR
  "V0" . "DELIVERYBLOCKREASON" IS NULL AND "TSOFLSORDQ" . "D
  ELIVERYBLOCKREASON" IS NULL) AND ("V0" . "DELIVERYBLOCKREASONTXT" = "TSOFLSORDQ" .
  "DELIVERYBLOCKREASONTXT" OR "V0" . "DELIVERYBLOCKREASO
  
```

Matching DCL-Roles Show all PFCG-AuthORIZATIONS to all matching roles

DCL-Role	M	Rule	Entity/Resource	Con...	#A...
Z_TEST_SLSORDFLMTSALESORDER	X	R-0000	Z_TEST_SLSORDFLMTSALE...	C-00	1
Z_TEST_SLSORDFLMTSALESORDER	X	R-0000	Z_TEST_SLSORDFLMTSALE...	C-01	1

Used DCL role

PFCG-AuthORIZATIONS related to all matching DCL-Roles

Object	Authorization	Field	From value	To value
V_VBAK_AAT	T-Q864007500	AUART	TA	
V_VBAK_VKO		SPART	*	
		VKORG	*	
		VTWEG	*	

Used authorizations of the user

Results - Number of returned entries from the database: 356

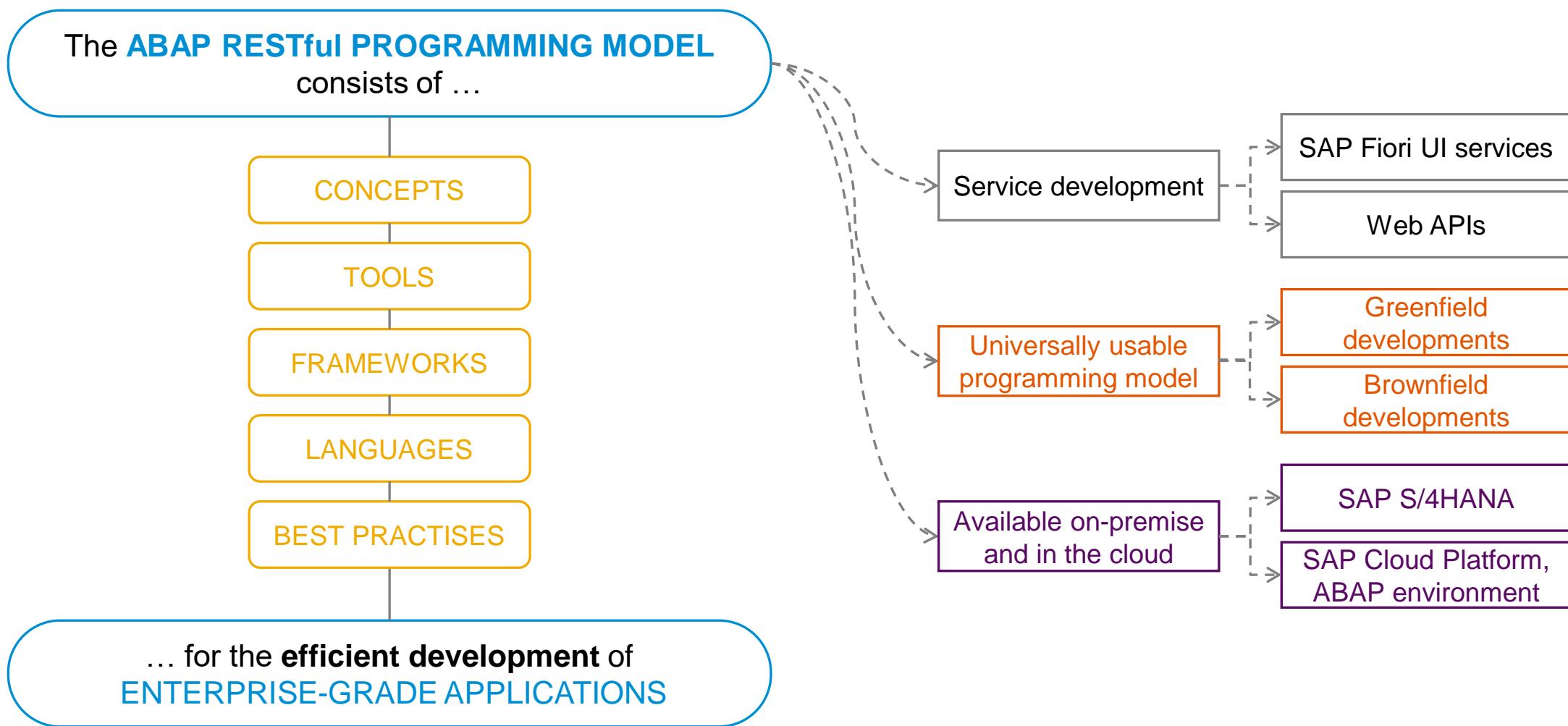
Cl...	Sales Doc.	Sal...	Description	Changed on	Created on	Created by	Time	Sal..	Name	D	Name	D	Name	Sal..	Sal..	Partne
1	OR	Standard Order		19.06.2014	WEISSENBER...	20:42:03	0001	Sales Org. 001	01	Distribtn Channel 01	01	Product Division 01	0001	001		
2	OR	Standard Order		23.06.2014	OPSTESTER	07:35:31	0001	Sales Org. 001	01	Distribtn Channel 01	01	Product Division 01	0001	001		
3	OR	Standard Order		23.06.2014	OPSTESTER	08:02:03	0001	Sales Org. 001	01	Distribtn Channel 01	01	Product Division 01	0001	001		
4	OR	Standard Order		23.06.2014	OPSTESTER	10:47:05	0001	Sales Org. 001	01	Distribtn Channel 01	01	Product Division 01	0001	001		
5	OR	Standard Order		23.06.2014	OPSTESTER	11:50:44	0001	Sales Org. 001	01	Distribtn Channel 01	01	Product Division 01	0001	001		
6	OR	Standard Order		23.06.2014	OPSTESTER	11:56:44	0001	Sales Org. 001	01	Distribtn Channel 01	01	Product Division 01	0001	001		
7	OR	Standard Order		23.06.2014	OPSTESTER	11:58:04	0001	Sales Org. 001	01	Distribtn Channel 01	01	Product Division 01	0001	001		
8	OR	Standard Order		23.06.2014	OPSTESTER	13:56:39	0001	Sales Org. 001	01	Distribtn Channel 01	01	Product Division 01	0001	001		

Unit 2 – Advanced Concepts

Lesson 6 – Consumption of CDS Views in SAP Fiori

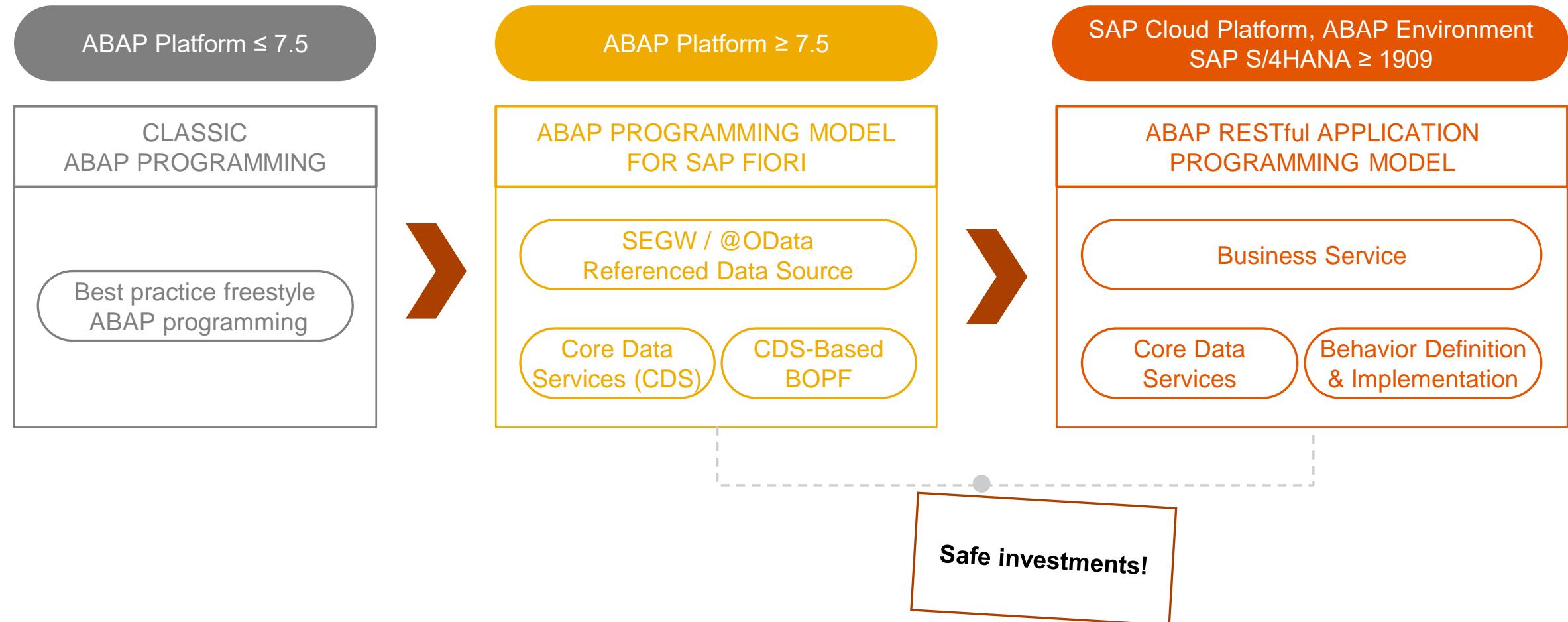
The Big Picture

The ABAP RESTful Application Programming Model (RAP)



The Big Picture

Evolution towards the ABAP RESTful Application Programming Model (RAP)



Architecture Overview

The big picture

SERVICE CONSUMPTION



BUSINESS SERVICES PROVISIONING



DATA MODELING & BEHAVIOR

SAP FIORI UI
Consume OData UI services

WEB API
Consume OData Web APIs

SERVICE BINDING – Bind to protocol version and scenario

SERVICE DEFINITION – Define scope to be exposed

BUSINESS OBJECT PROJECTION

- CDS: Projection views
- BDEF: Behavior projection
- ABAP: Behavior implementation

BUSINESS OBJECTS

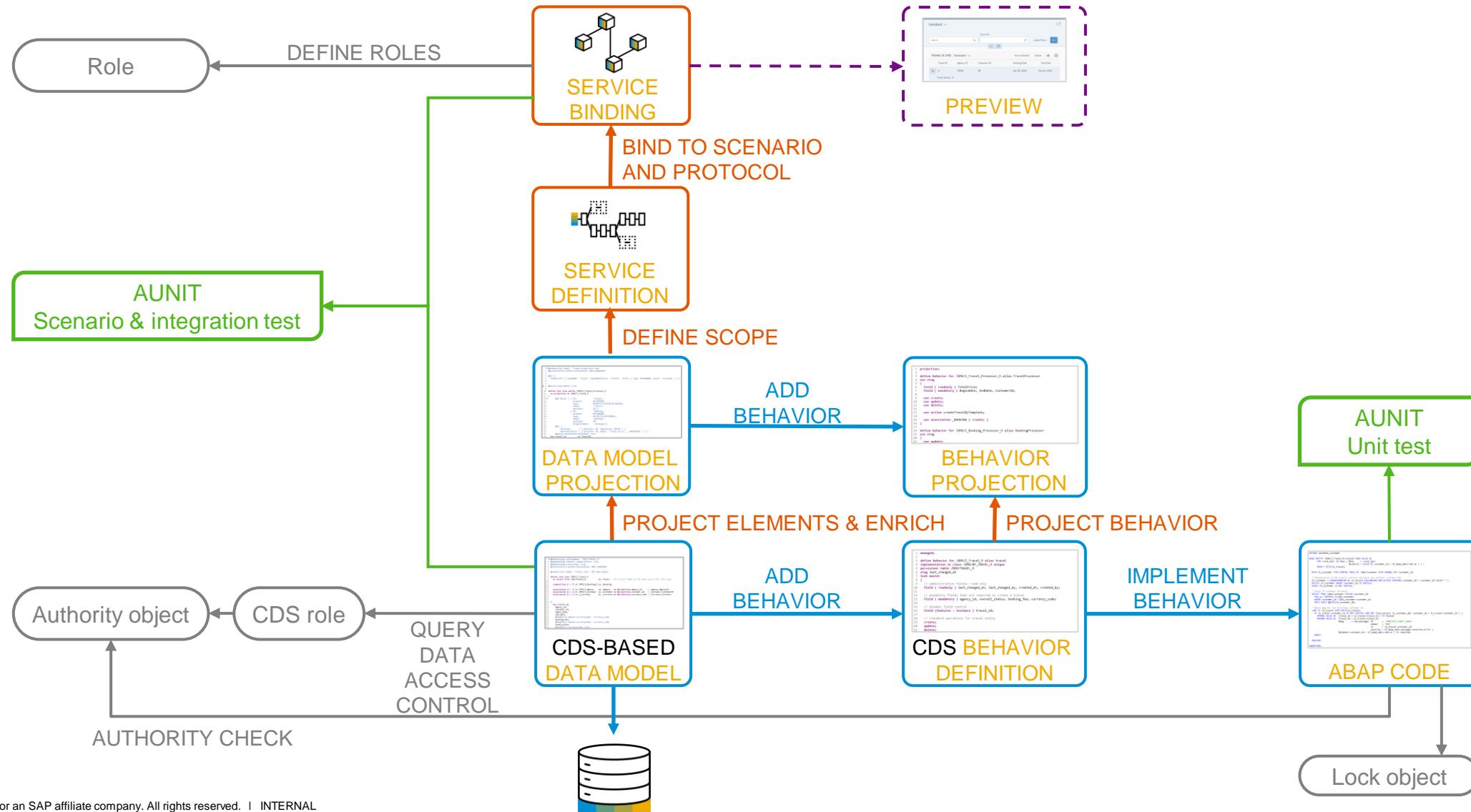
- CDS: Data modeling
- BDEF: Behavior definition
- ABAP: Behavior implementation

QUERIES

- CDS: Data modeling

Architecture Overview

Development flow



UI-Specific Annotations

```
@AbapCatalog.sqlViewName: 'S4D430_ODFIORI'  
@OData.publish: true  
  
@UI.headerInfo: { typeName: 'Flight Booking',  
                  typeNamePlural: 'Flight Bookings'}  
  
@Search.searchable: true  
  
define view S4d430_OData_And_Fiori as select  
    from sbook  
    association[1] to scustom as _customer  
        on customid = _customer.id  
    {  
        @UI.lineItem: { position: 10, importance: #HIGH }  
        @UI.selectionField.position: 30  
    key carrid,  
    ...  
}
```

Set a Caption for the List Display

Enable a Search Dialog

Display this Column and Set the Position

Display this field in the selection dialog

Preview of an Application in ADT

ZFET_SOLUTIONS (23) *Fiori Elements Training Solutions*

- > Authorizations (2)
- > Business Services (4)
 - > Service Bindings (2)
 - > ZFET_SLSORDER_SB_SOL *Sales Order - Service Binding*
 - > ZFET_SLSORDERITEMAG_SB_SOL *Service Binding for ALP*
 - > Service Definitions (2)
- > Core Data Services (11)
- > Dictionary (5)
- > Source Code Library (1)



[RA8] ZFET_SLSORDER_SB_SOL

Service Binding: ZFET_SLSORDER_SB_SOL

General Information
This section describes general information about this service binding
Binding Type: OData V2 - UI

Service Versions
Define service versions associated with the service binding
type filter text

Versi...	Service Definition
000	ZFET_SLSORDER_SD_SOL

Add... Remove

Service Version Details
View information on selected service version
Local Service Endpoint: Published Unpublish

Service Information
Service URL: /sap/opu/odata/sap/ZFET_SLSORDER_SB_SOL
type filter text

Entity Set and Association

- OverallStatus
- SalesOrItemAG
- SalesOrderItem
 - to_SalesOrder
- SalesOrder
 - to_AggItem
 - to_Item
 - to_OverallStatus

Preview...

Console Problems
0 items

Description Resource Path

UI-Specific Annotations – The Outcome

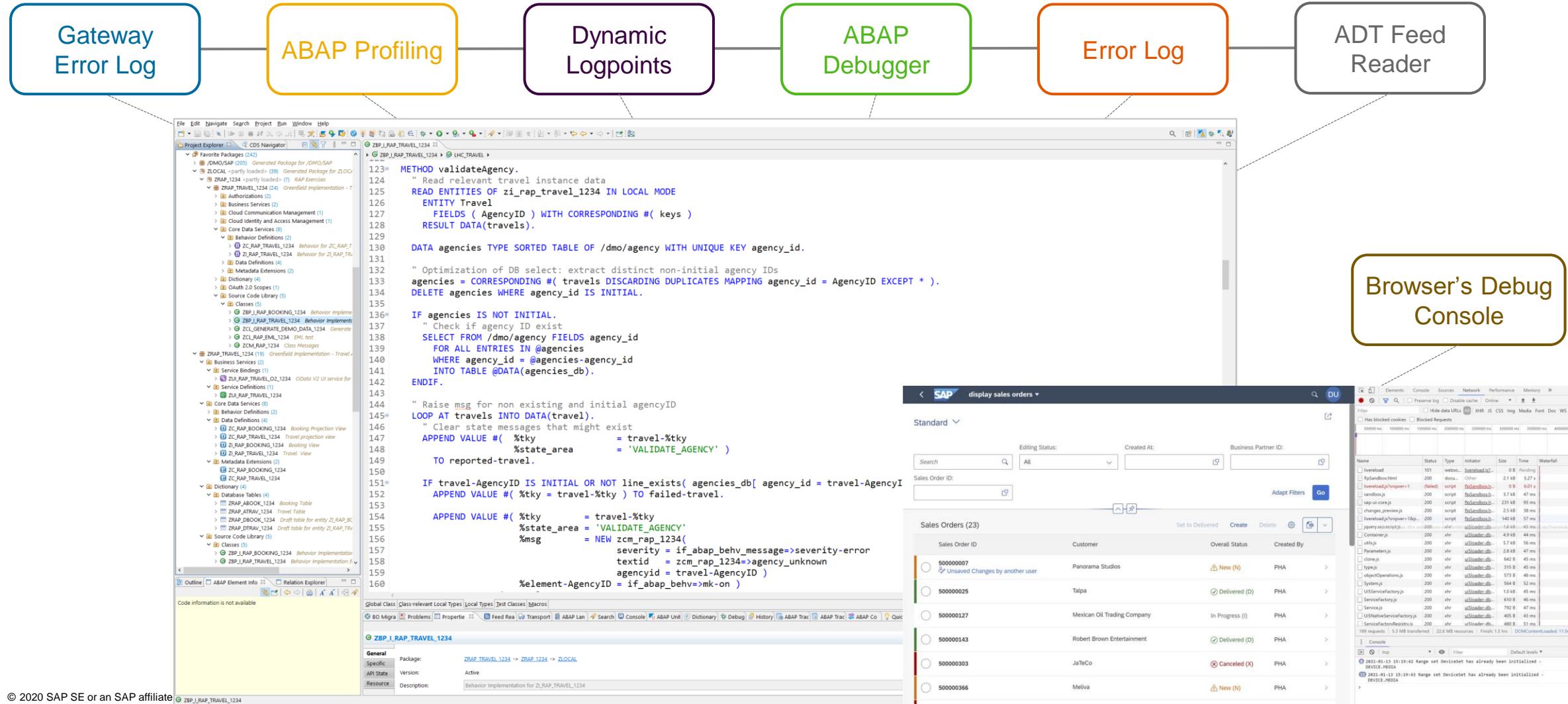
The screenshot shows an SAP Fiori application interface for flight bookings. At the top, there is a search bar labeled "Search" with a magnifying glass icon, followed by three dropdown fields for "Country", "City", and "Airline". Below the search bar are buttons for "Hide Filters" and "Adapt Filters". A "Go" button is located at the bottom right of the filter area. The main content area displays a table titled "Flight Bookings (100,110) | Standard". The table has columns for Airline, Connection number, Flight Date, Booking number, Customer name, price, and Payment method. The first four columns are highlighted with a light blue background. The "Airline" column is explicitly labeled in the table header. The table contains four rows of booking data. A yellow callout bubble points to the search bar with the text "Search Function Switched on". Another callout bubble points to the "Airline" dropdown with the text "Selection via airline (3rd selection field)". A third callout bubble points to the "Airline" column in the table with the text "Column Airline is visible in the list display".

Airline	Conn...	Flight Date	Booking number	Customer name	price	Payme...	
AA	0017	2/23/17	00000001	Johann Buehler	719.00	USD	USD
AA	0017	2/23/17	00000002	Christoph Ryan	761.30	USD	USD
AA	0017	2/23/17	00000003	Christine D'Oultrement	761.30	USD	USD
AA	0017	2/23/17	00000004	Anna Dumbach	761.30	USD	USD

Troubleshooting Tools

Troubleshooting Your SAP Fiori App

ADT troubleshooting tools for ABAP



Troubleshooting Your SAP Fiori App

ADT troubleshooting tools for ABAP CDS

Dictionary Log

Data Preview

Activation
Graph

Annotation
Propagation

```
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'Sales Order Header - Consumption View'

@VDM.viewType: #CONSUMPTION

@Search.searchable: true

@Metadata.allowExtensions: true

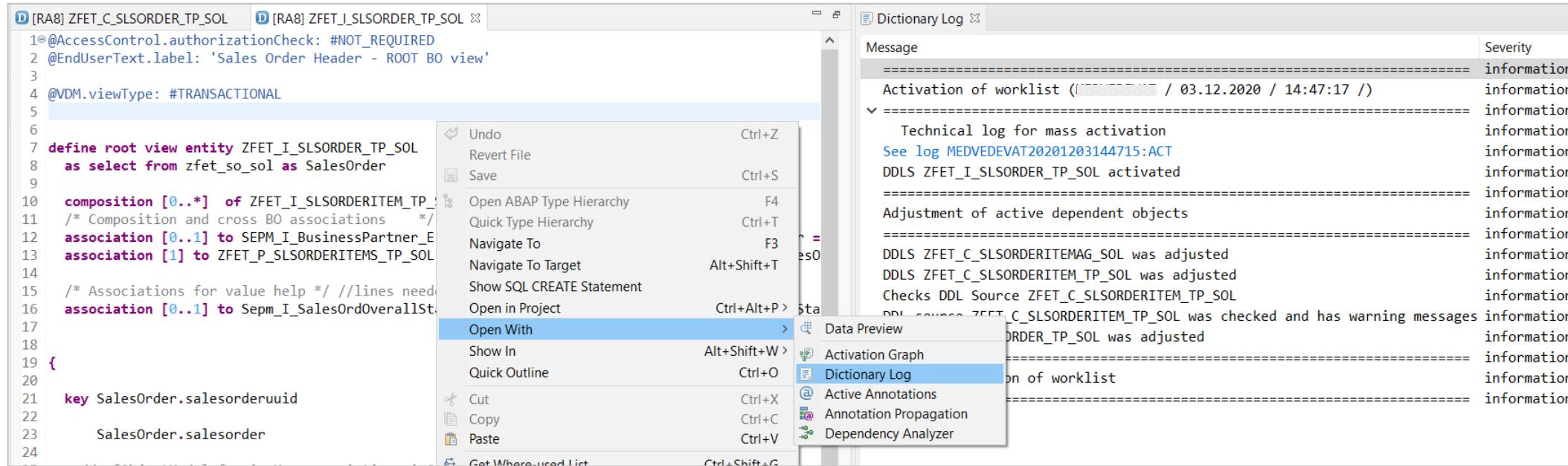
@UI.lineItem: [{criticality: 'Criticality'}]
define root view entity ZFET_C_SLSORDER_TP_SOL
as projection on ZFET_I_SLSORDER_TP_SOL as SalesOrder

association [0..*] to ZFET_C_SLSORDERITEMAG_SOL as _AggrItem on _AggrItem.SalesOrderUUID = SalesOrder.SalesOrderUUID
{
    key SalesOrder.SalesOrderUUID,
    @Search.defaultSearchElement: true
    @Search.fuzzinessThreshold: 0.9
    SalesOrder.SalesOrder,
    @Consumption.semanticObject: 'Customer'
    //ExerciseTag: SmartLinkNavigation
    @ObjectModel.text.element: ['CompanyName']
    SalesOrder.Customer,
    @ObjectModel.text.element: ['OverallStatusText']
    SalesOrder.OverallStatus,
    SalesOrder.Criticality, //line needed for ExerciseTag: EmphasizeOverallStatus
    //lines needed for ExerciseTag: AddAdminDataAsFormGroup
    SalesOrder.Created_at, //line needed for ExerciseTag: AddFieldsToListReportTable
    SalesOrder.Created_by_loginname, //line needed for ExerciseTag: AddFieldsToListReportTable
    SalesOrder.LocalLastChangedAt,
    SalesOrder.LocalLastChangedBy,
    //lines needed for ExerciseTag: AddDataPointsToObjectHeader
    @EndUserText.label: 'Sum of Quantities'
```

Active
Annotations

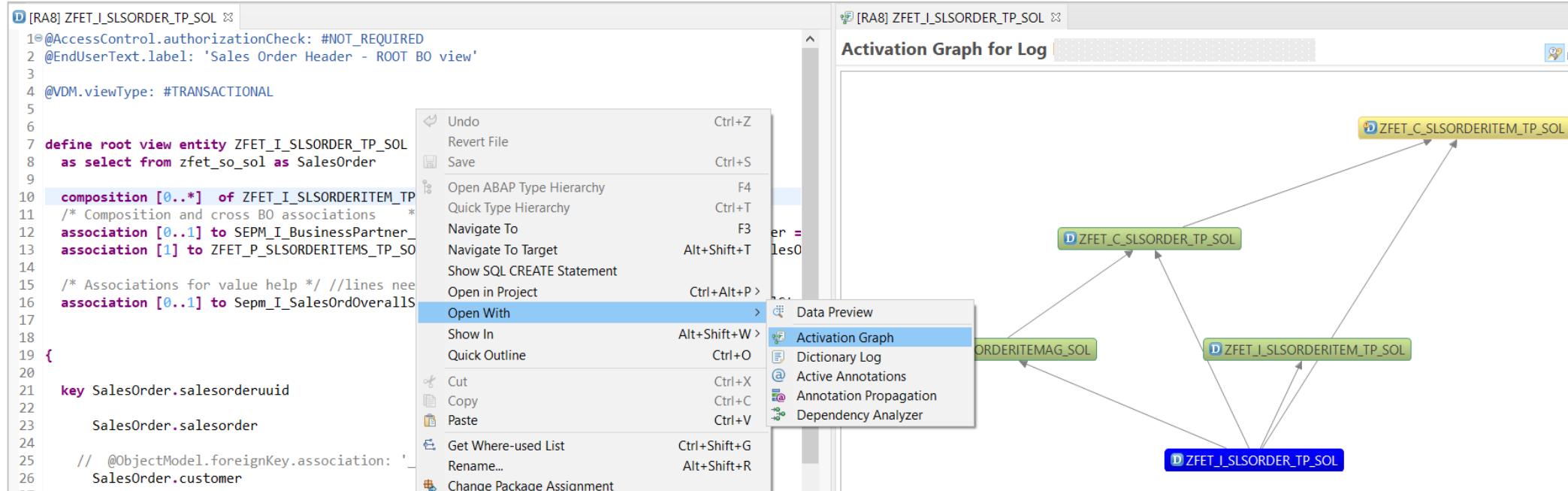
Troubleshooting Your SAP Fiori App

ADT troubleshooting tools for ABAP CDS: Dictionary Log



Troubleshooting Your SAP Fiori App

ADT troubleshooting tools for ABAP CDS: Activation Graph



Troubleshooting Your SAP Fiori App

ADT troubleshooting tools for ABAP CDS: Annotation Propagation

The screenshot shows the SAP Studio interface with two code editors and a central annotation propagation tool.

Code Editor 1:

```

4
5 @VDM.viewType: #CONSUMPTION
6
7 @Search.searchable: true
8
9 @Metadata.allowExtensions: true
10
11
12 @UI.lineItem: [{criticality: 'Criticality'}]
13 define root view entity ZFET_C_SLSORDER_TP_SOL
14   as projection on ZFET_I_SLSORDER_TP_SOL as SalesOrder
15
16   association [0..*] to ZFET_C_SLSORDERITEMAG_SOL as _AggrItem on _AggrItem.SalesOrderUUID = S
17 {
  ...
}
  
```

Code Editor 2:

Annotation Propagation for Entity ZFET_C_SLSORDER_TP_SOL

Annotation Propagation Dialog:

- Selection: Annotations For: (empty), Annotation Key: (empty)
- Value Propagation: A table showing annotation details:

Origin Source	Annotation For	Annotation Key	Annotation Value	Layer	Variant	Entity Name
ZFET_C_SLSORDER_TP_SOL	ZFET_C_SLSORDER_TP_SOL	UI.headerInfo.description.value	'OverallStatus'	CORE		ZFET_C_SLSORDER_TP_SOL
ZFET_C_SLSORDER_TP_SOL	ZFET_C_SLSORDER_TP_SOL	UI.headerInfo.title.type	#STANDARD	CORE		ZFET_C_SLSORDER_TP_SOL
ZFET_C_SLSORDER_TP_SOL	ZFET_C_SLSORDER_TP_SOL	UI.headerInfo.title.value	'SalesOrder'	CORE		ZFET_C_SLSORDER_TP_SOL
ZFET_C_SLSORDER_TP_SOL	ZFET_C_SLSORDER_TP_SOL	UI.headerInfo.typeName	'Sales Order'	CORE		ZFET_C_SLSORDER_TP_SOL
ZFET_C_SLSORDER_TP_SOL	ZFET_C_SLSORDER_TP_SOL	UI.headerInfo.typeNamePlural	'Sales Orders'	CORE		ZFET_C_SLSORDER_TP_SOL
ZFET_C_SLSORDER_TP_SOL	ZFET_C_SLSORDER_TP_SOL	AccessControl.authorizationCheck	#NOT_REQUIRED			ZFET_C_SLSORDER_TP_SOL
ZFET_C_SLSORDER_TP_SOL	ZFET_C_SLSORDER_TP_SOL	EndUserText.label	'Sales Order Header - Consumption View'			ZFET_C_SLSORDER_TP_SOL
ZFET_C_SLSORDER_TP_SOL	ZFET_C_SLSORDER_TP_SOL	Metadata.allowExtensions	true			ZFET_C_SLSORDER_TP_SOL
ZFET_C_SLSORDER_TP_SOL	ZFET_C_SLSORDER_TP_SOL	Search.searchable	true			ZFET_C_SLSORDER_TP_SOL
ZFET_C_SLSORDER_TP_SOL	ZFET_C_SLSORDER_TP_SOL	UI.lineItem[1].criticality	'Criticality'			ZFET_C_SLSORDER_TP_SOL
ZFET_C_SLSORDER_TP_SOL	ZFET_C_SLSORDER_TP_SOL	VDM.viewType	#CONSUMPTION			ZFET_C_SLSORDER_TP_SOL

Troubleshooting Your SAP Fiori App

ADT troubleshooting tools for ABAP CDS: Active Annotations

The screenshot shows the SAP ADT interface with two main windows:

- Left Window:** A code editor for the file `ZFET_C_SLSORDER_TP_SOL.cds`. The code defines a CDS view with annotations like `@AccessControl.authorizationCheck: #NOT_REQUIRED`, `@EndUserText.label: 'Sales Order Header - Consumption View'`, and `@VDM.viewType: #CONSUMPTION`.
- Right Window:** An "Active Annotations" view titled "Active Annotations for Entity ZFET_C_SLSORDER_TP_SOL". It lists annotations for the entity, such as `@UI.headerInfo.description` and `@UI.headerInfo.title`, along with their values and origin data sources.
- Context Menu:** A context menu is open over the annotation `@Search.defaultSearchElement: true`. The "Open With" option is highlighted, and a submenu shows "Data Preview", "Activation Graph", "Dictionary Log", "Active Annotations" (which is also highlighted), and other options like "Annotation Propagation" and "Dependency Analyzer".

Troubleshooting Your SAP Fiori App

ADT troubleshooting tools for ABAP CDS – ADT Feed Reader

The screenshot shows the SAP ADT Feed Reader interface. The left pane displays a hierarchical feed entry structure under 'Project / Feed Query / Feed Entry'. The main pane lists 'Runtime Errors for objects I am responsible for' with the following details:

Object	Date	Time	Author
Syntax error in program "Z14_TR_PR_I======"	27.01.2021	17:58:32	TKS-*
Syntax error in program "Z14_TR_PR_I======"	27.01.2021	17:58:31	TKS-*
Syntax error in program "Z14_TR_PR_I======"	27.01.2021	17:57:26	TKS-*
Syntax error in program "Z14_TR_PR_I======"	27.01.2021	17:57:24	TKS-*
Syntax error in program "ZDD26_C_TRAININGREQ=====	27.01.2021	17:46:10	TKS-*
Syntax error in program "ZDD26_C_TRAININGREQ=====	27.01.2021	17:46:10	TKS-*
Syntax error in program "ZDD26_C_TRAININGREQ=====	27.01.2021	17:45:35	TKS-*
Syntax error in program "ZDD26_C_TRAININGREQ=====	27.01.2021	17:45:35	TKS-*
Syntax error in program "ZDD26_C_TRAININGREQ=====	27.01.2021	17:45:23	TKS-*
Syntax error in program "ZDD26_C_TRAININGREQ=====	27.01.2021	17:45:23	TKS-*

The right pane provides summary information for the feed entry:

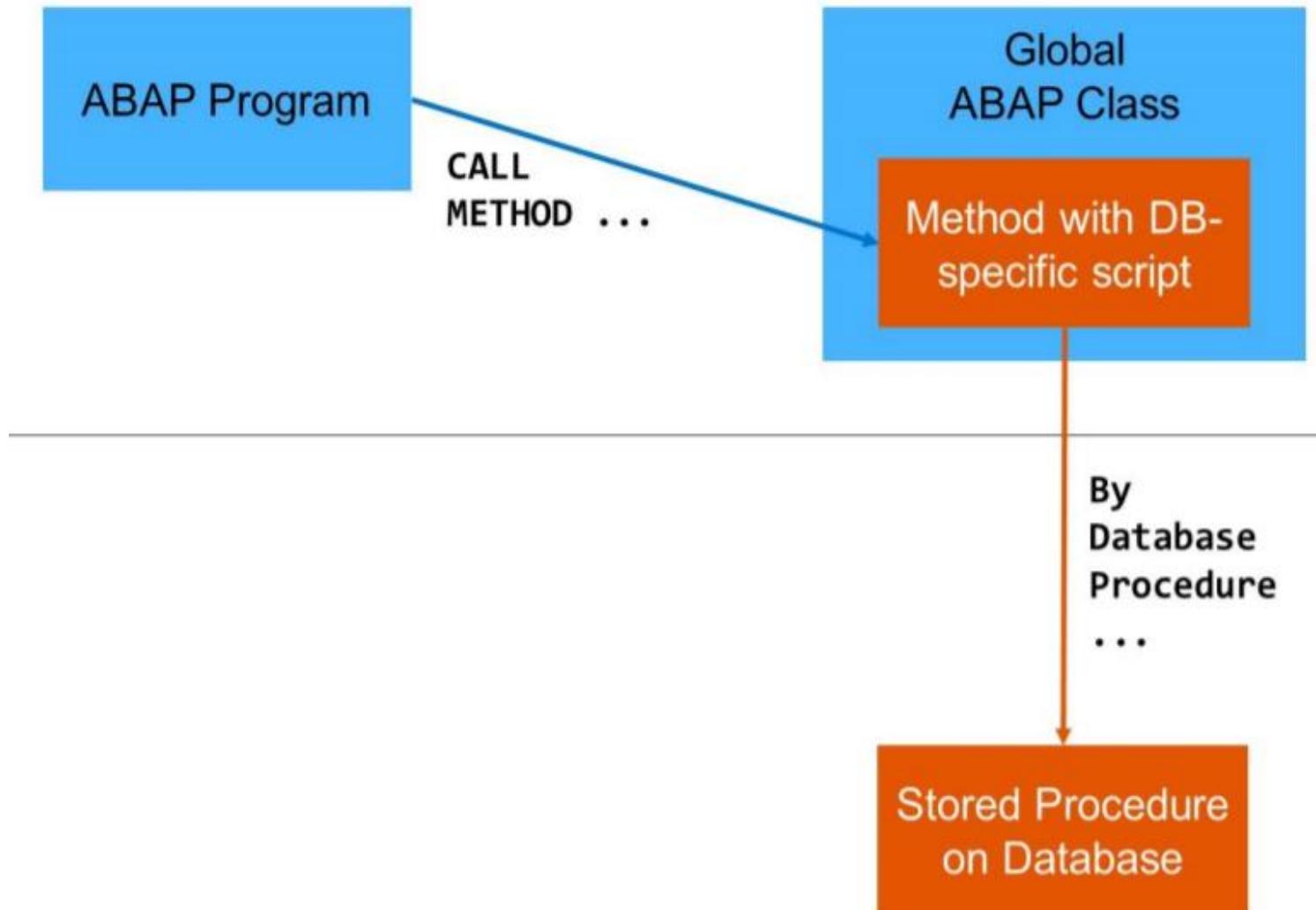
- RA8_100_..._en
- Refresh interval: 5 Minutes
- Refreshes on communication failures
- Page size: 50
- Displays notification messages
- Filter: and(equals(responsible,))
- Last refresh: 27-Jan-2021, 6:42:10 PM

Demo of [Analyze Issue in ADT](#)

Unit 3 – HANA Native features in CDS

Lesson 1 – ABAP Managed Database Procedure

Basic idea of AMDP



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

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 Definition**

- Implementation of Interface IF_AMDP_MARKER_HDB

- **Method Definition**

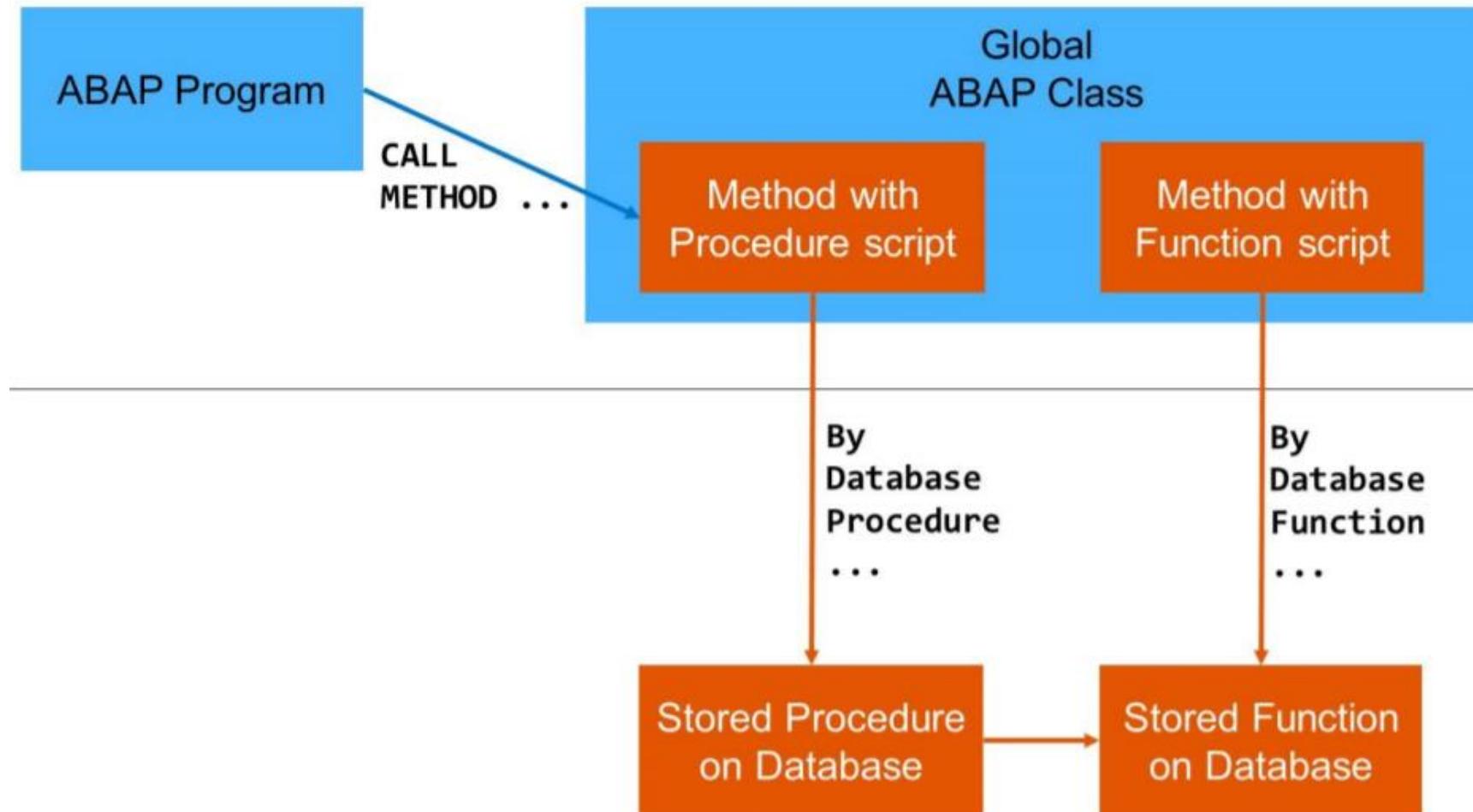
- VALUE() addition for all parameters
 - Only scalar or table types for parameters (no structures)

- **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



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

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;
```

Return value is a table with
columns ID and NAME

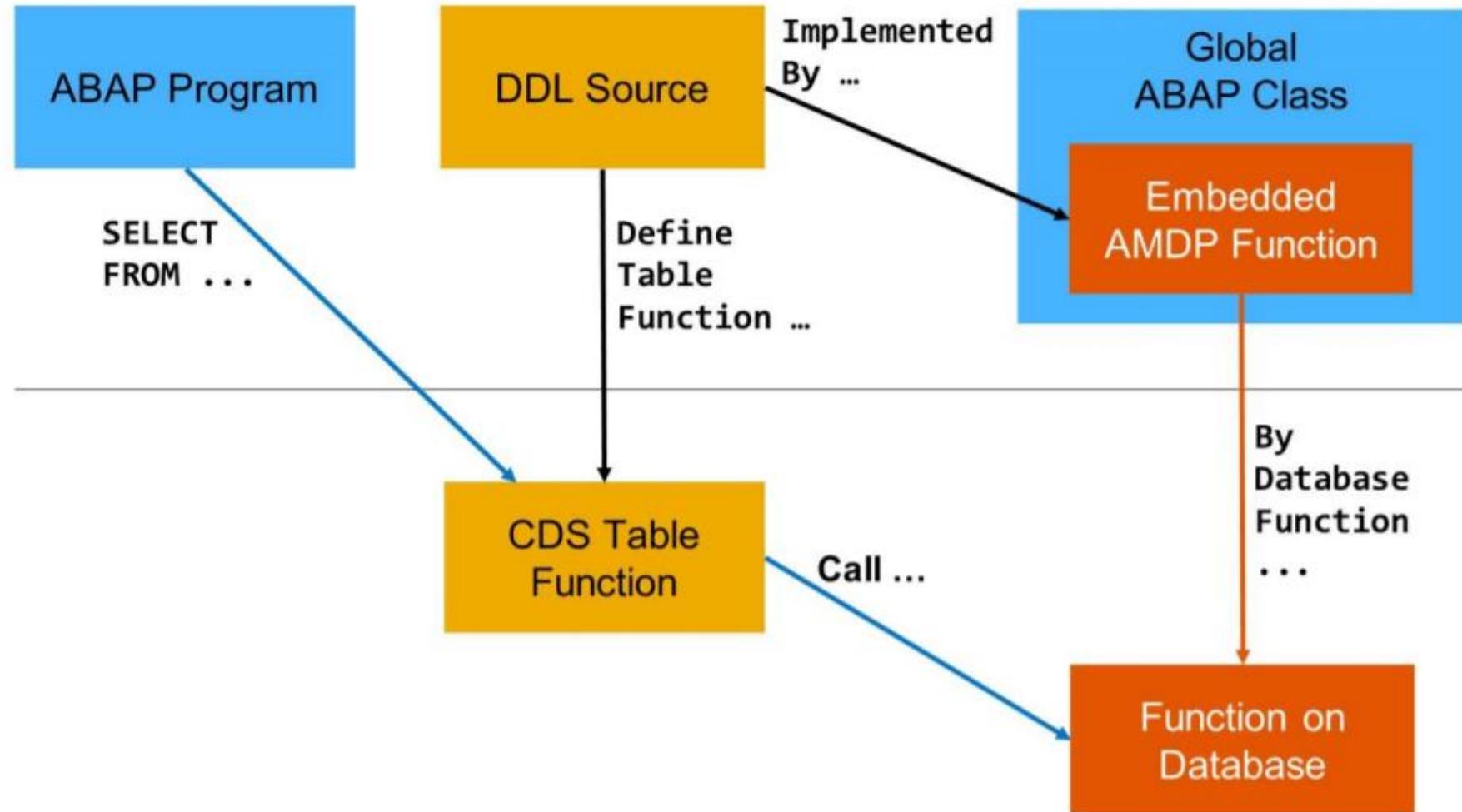
```
ENDMETHOD.
```

```
ENDCLASS.
```

Unit 3 – HANA Native features in CDS

Lesson 2 – CDS Table Function

The Basic Idea of CDS Table Functions



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)

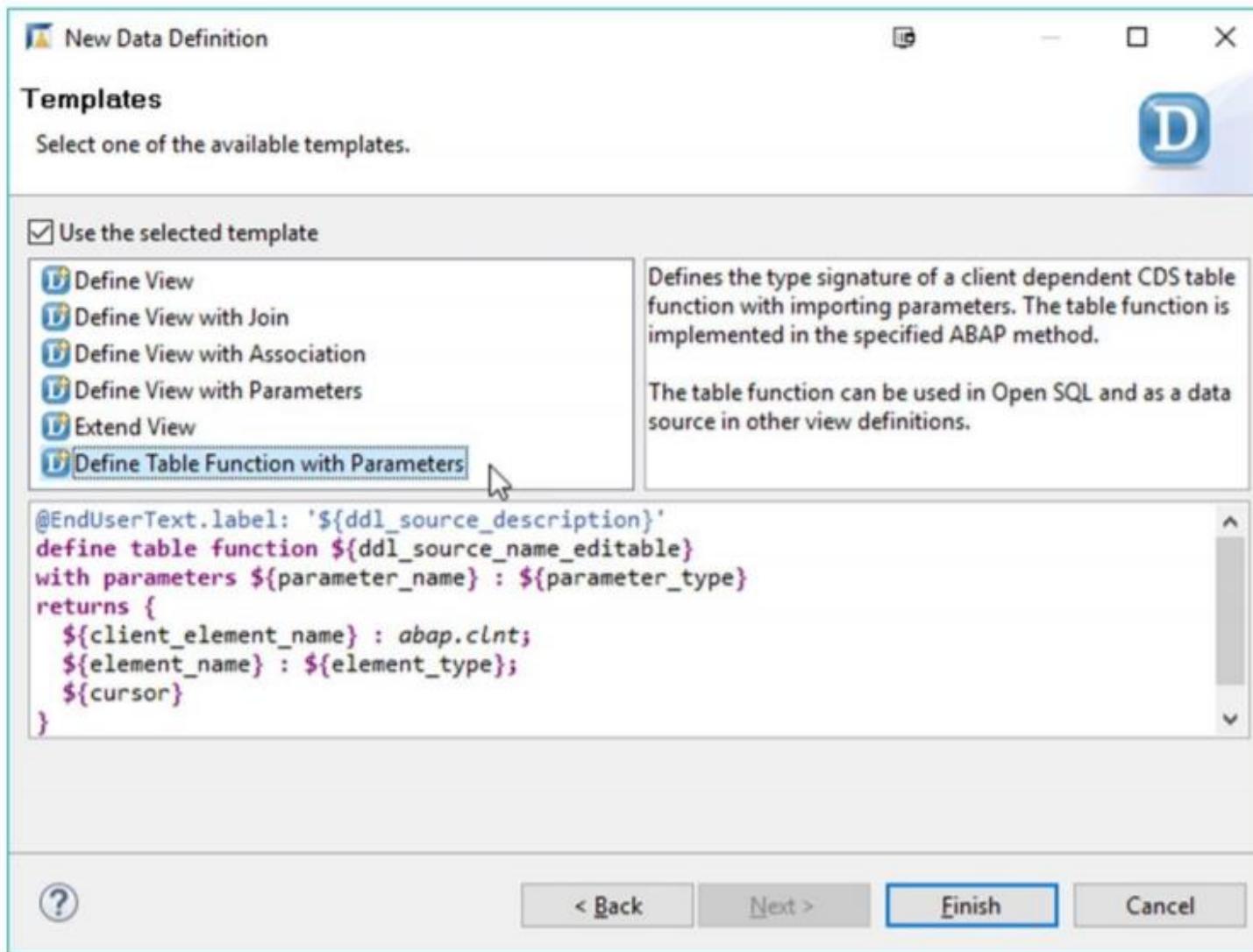
Name of AMDP class
and AMDP method

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 for Define Table Function with Parameters



- **Create DDL Source 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 DDL Source
- **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 DDL Source
 - Test the CDS table function in data preview

Use of CDS Table Function in Open 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.

Check of System Capabilities Needed

Syntax check issues a warning (direct or indirect use of AMDP!)

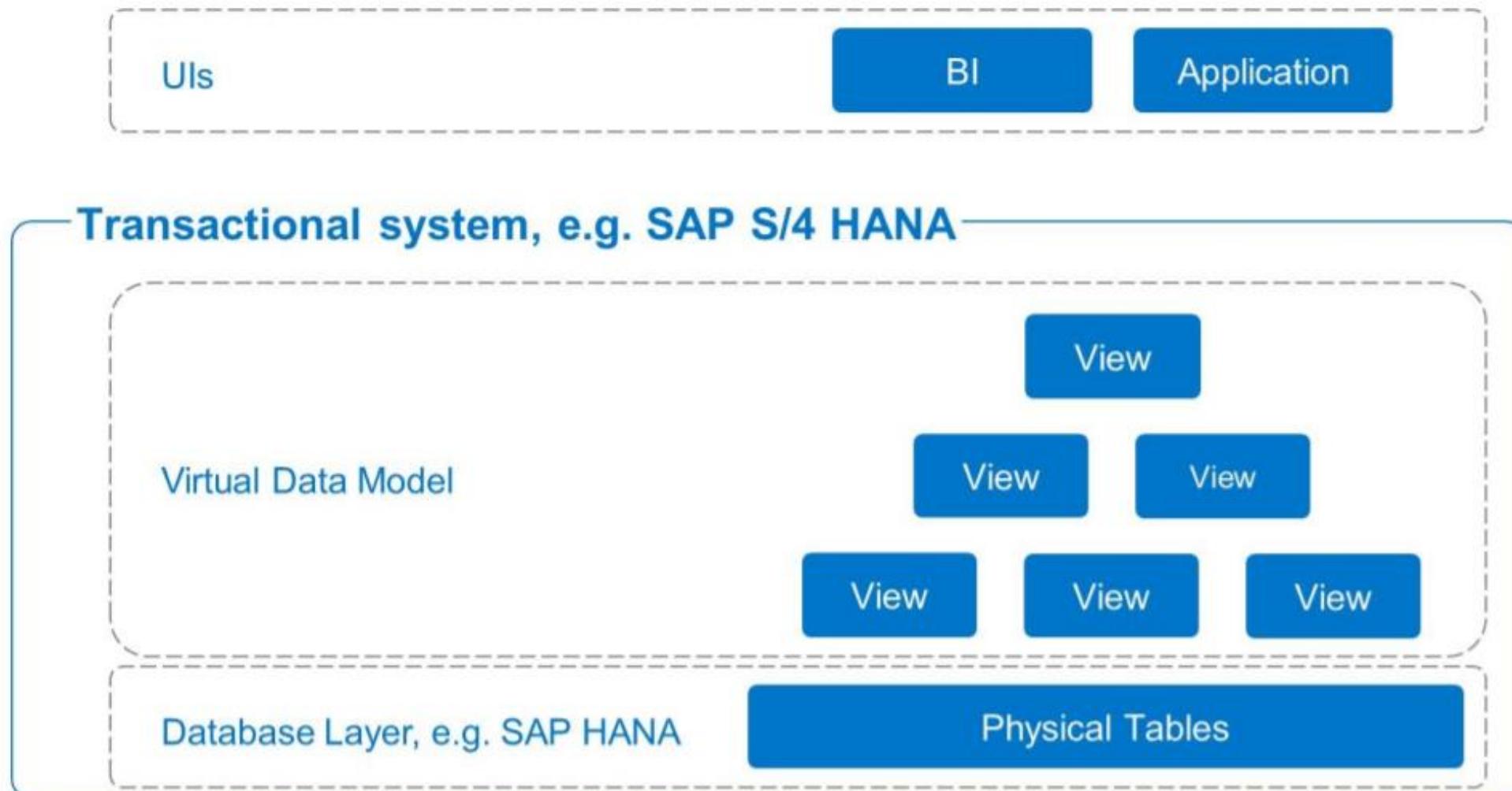


The database feature "AMDP_TABLE_FUNCTION" is used here (read the long text).

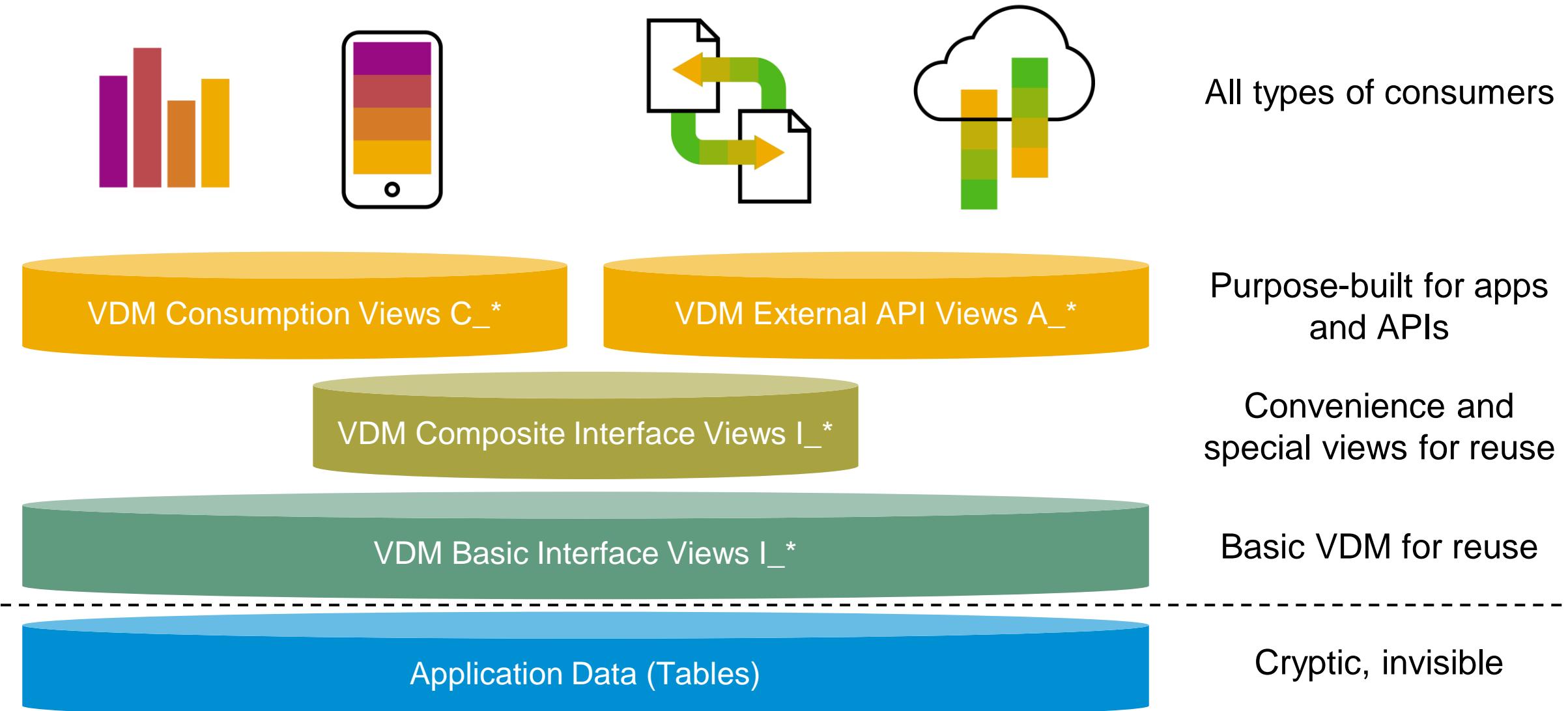
Check system capability and provide alternative coding

```
DATA gt_feature_set TYPE cl_abap_dbfeatures=>features_set_t.  
  
INSERT cl_abap_dbfeatures=>amdp_table_functions  
      INTO TABLE gt_feature_set.  
  
IF cl_abap_dbfeatures=>use_features( gt_feature_set )  
  = abap_false.  
  ...  
ENDIF.
```

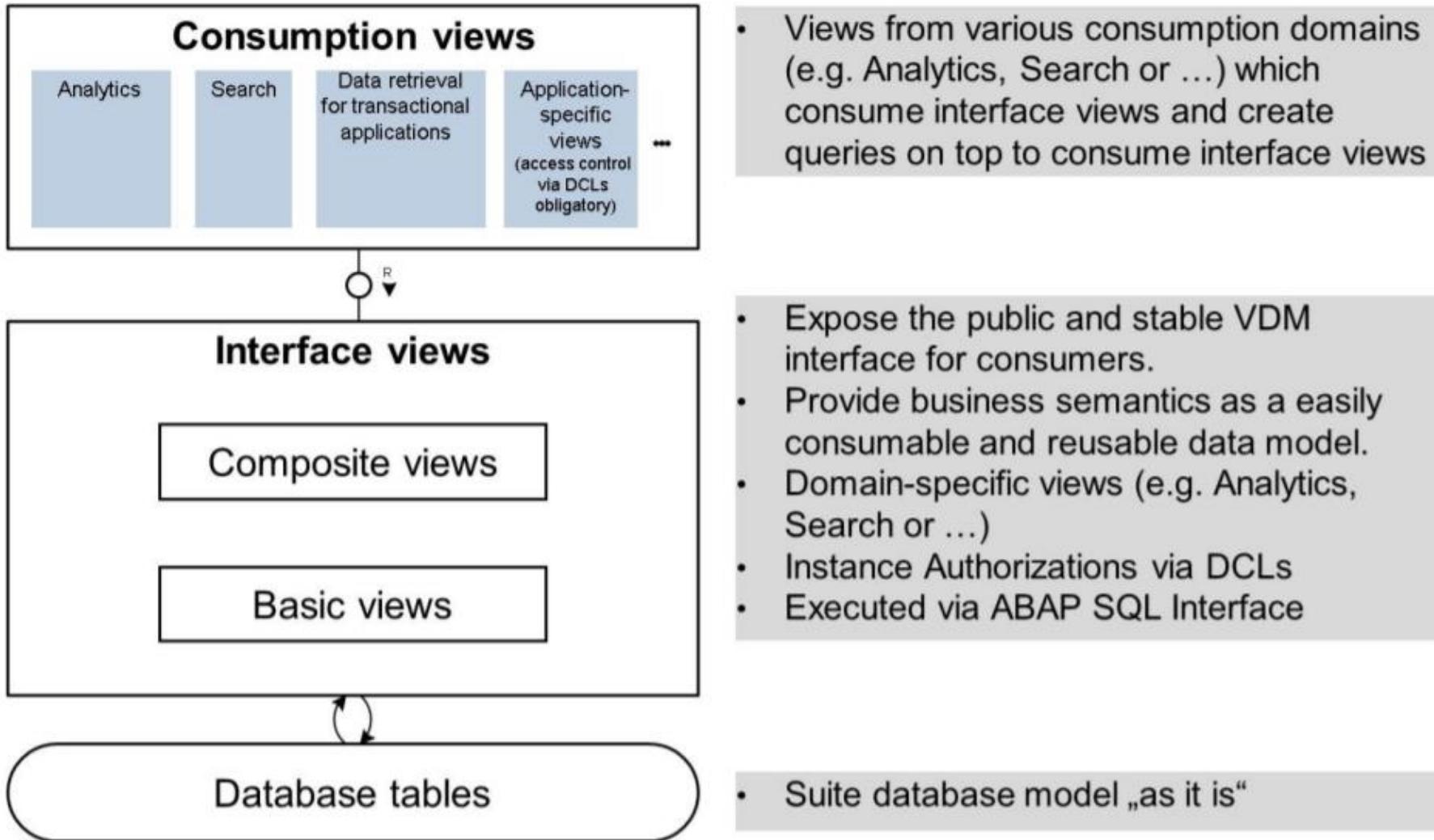
Unit 4 – Virtual Data Model



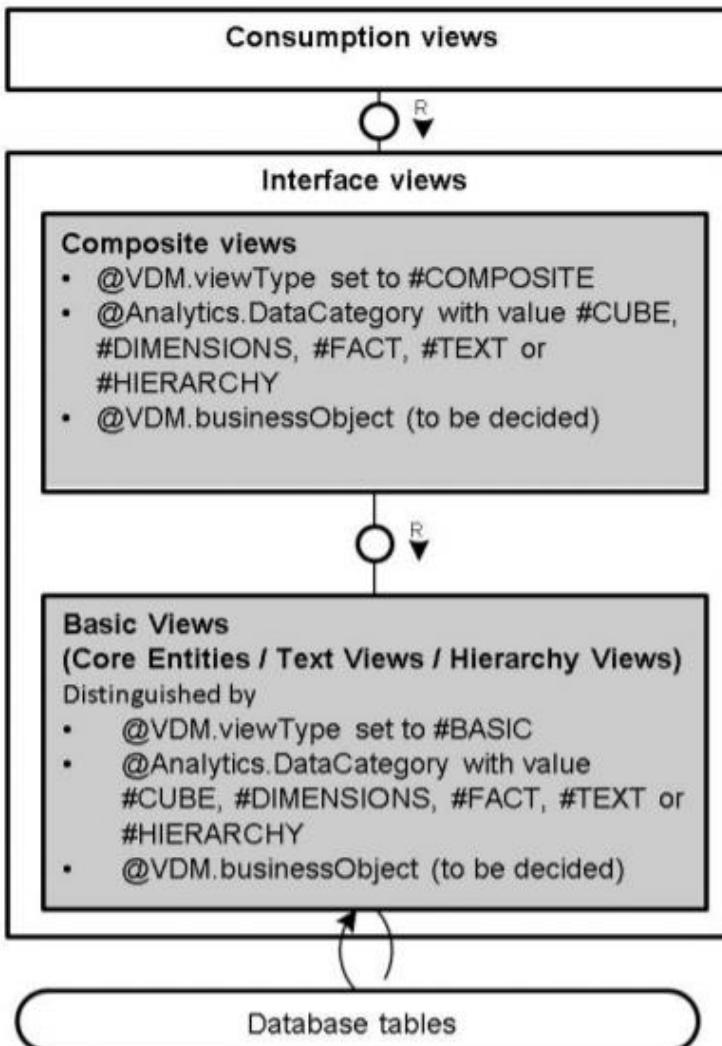
Stack of VDM Views



The Layers in the Virtual Data Model



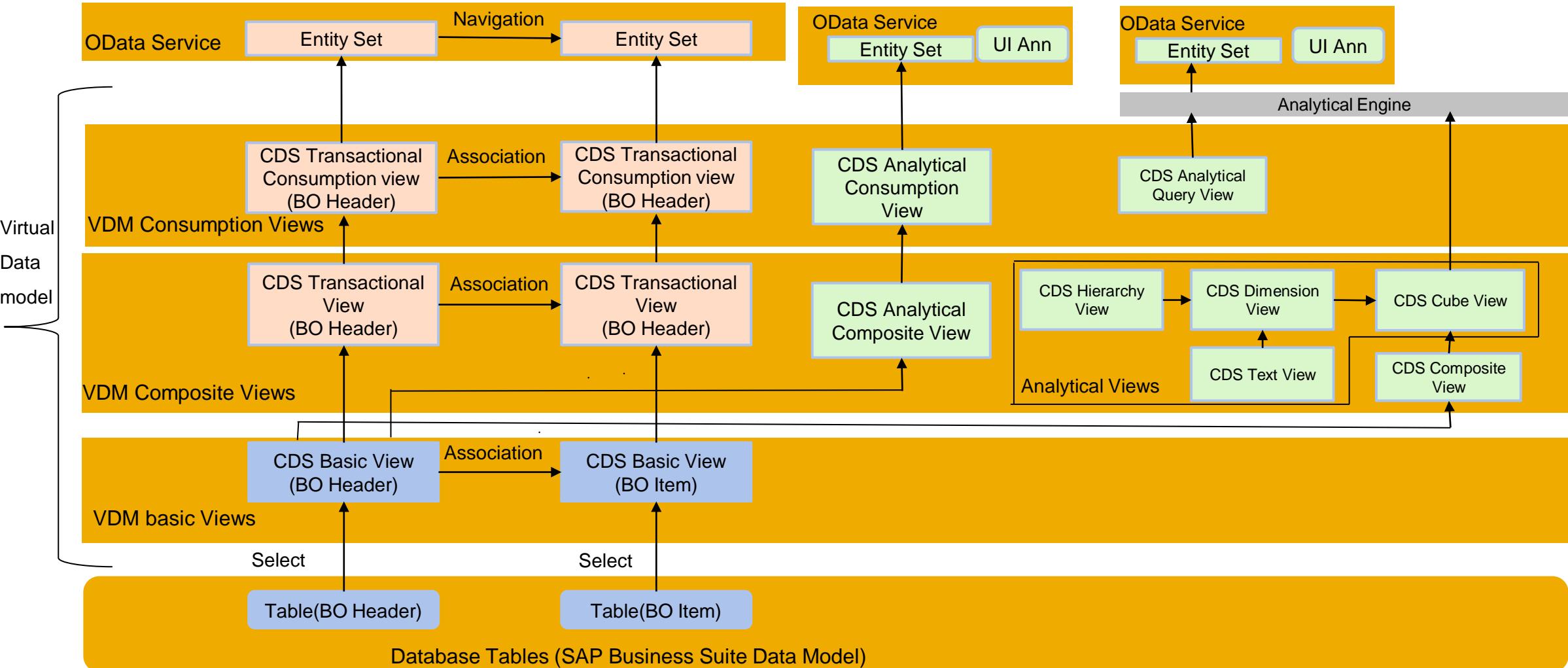
Interface Views Divided into Basic and Composite Views



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.

VDM Layering Architecture

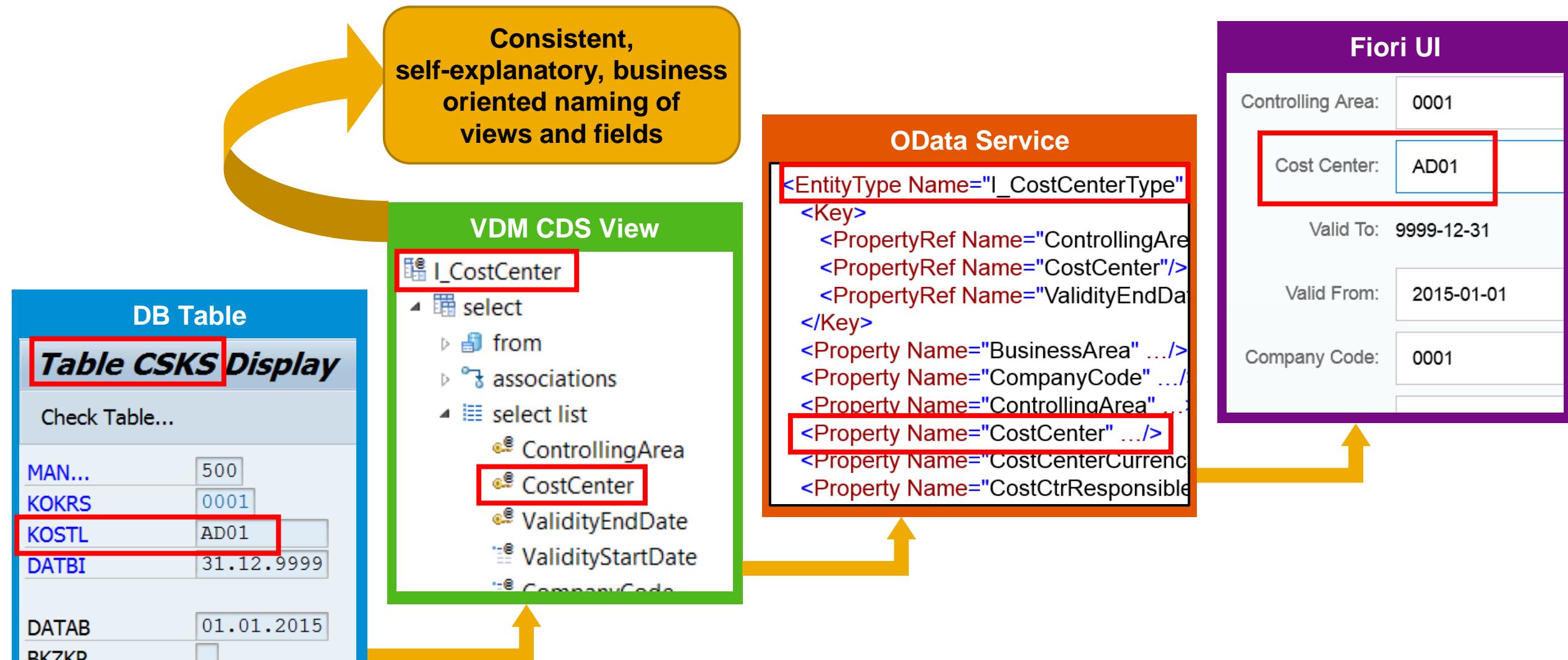


VDM CDS View Layering

Naming Conventions

VDM View Type	Private	CDS Name Prefix	SQL Name Prefix	Global Field Name Checks
#CONSUMPTION		C_	C	X
#CONSUMPTION	X	P_	P	
#COMPOSITE		I_	I	X
#COMPOSITE	X	P_	P	
#BASIC		I_	I	X
#BASIC	X	P_	P	
#EXTENSION		E_	E	X

Names and meta data are propagated from CDS via OData to Fiori UIs



- **@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

VDM CDS Development Guideline

In order to ensure consistency and homogeneity of VDM CDS views in S/4HANA a comprehensive set of rules has been compiled which is verified by ATC checks known as

VDM CDS Development Guideline

The screenshot shows a navigation sidebar on the left with links like 'Learning Corner', 'Tools', 'Concepts', and 'CDS Modeling Guidelines'. Under 'CDS Modeling Guidelines', there's a section for 'VDM CDS Development Guideline' containing topics such as 'VDM CDS Annotations', 'VDM CDS Guideline Details', 'VDM CDS Compositional Hierarchy', etc. Other sections include 'Naming Rules' and 'Important Information'. The main content area displays the 'VDM CDS Development Guideline' page, which includes a diagram illustrating the relationships between different types of VDM views: Consumption View, Interface View, Composite View, Basic View, and Extension Include View. The 'Important Information' box states: 'This guideline applies to VDM views developed for S/4HANA (e.g. in the software development kit) and the reserved VDM CDS View prefixes ('I_','P_','C_','E_','X_') and the VDM standard terms (Details)'.

VDM CDS ATC Checks

- All VDM ATC checks are based on the metadata captured in the source file of the CDS views and role definitions (DCLS).
- They are restricted to the CDS views using the view annotations @VDM...
- Additionally CDS model names, DDLS file names as well as DCLS file names and role names beginning with the reserved VDM prefixes 'I_ ', 'E_ ', 'P_ ' or 'C_ ' are included in the check scope.
- The ATC checks are executed on active objects only.
- ATC Check-ID: [POC_ANNOTA](#) (metadata checks) and [VDMCDSSEC](#) (security checks).

In order to reasonably execute the checks, the annotations @VDM.viewType, @VDM.viewExtension and @VDM.private have to be set correctly

More details can be found here:

<https://wiki.wdf.sap.corp/wiki/display/SuiteCDS/VDM+CDS+ATC+Checks>

Unit 4 – Test Double Framework

Why do we need unit test framework for “Code Pushdown”-Scenarios ? Complexity of S/4 CDS Model (1)

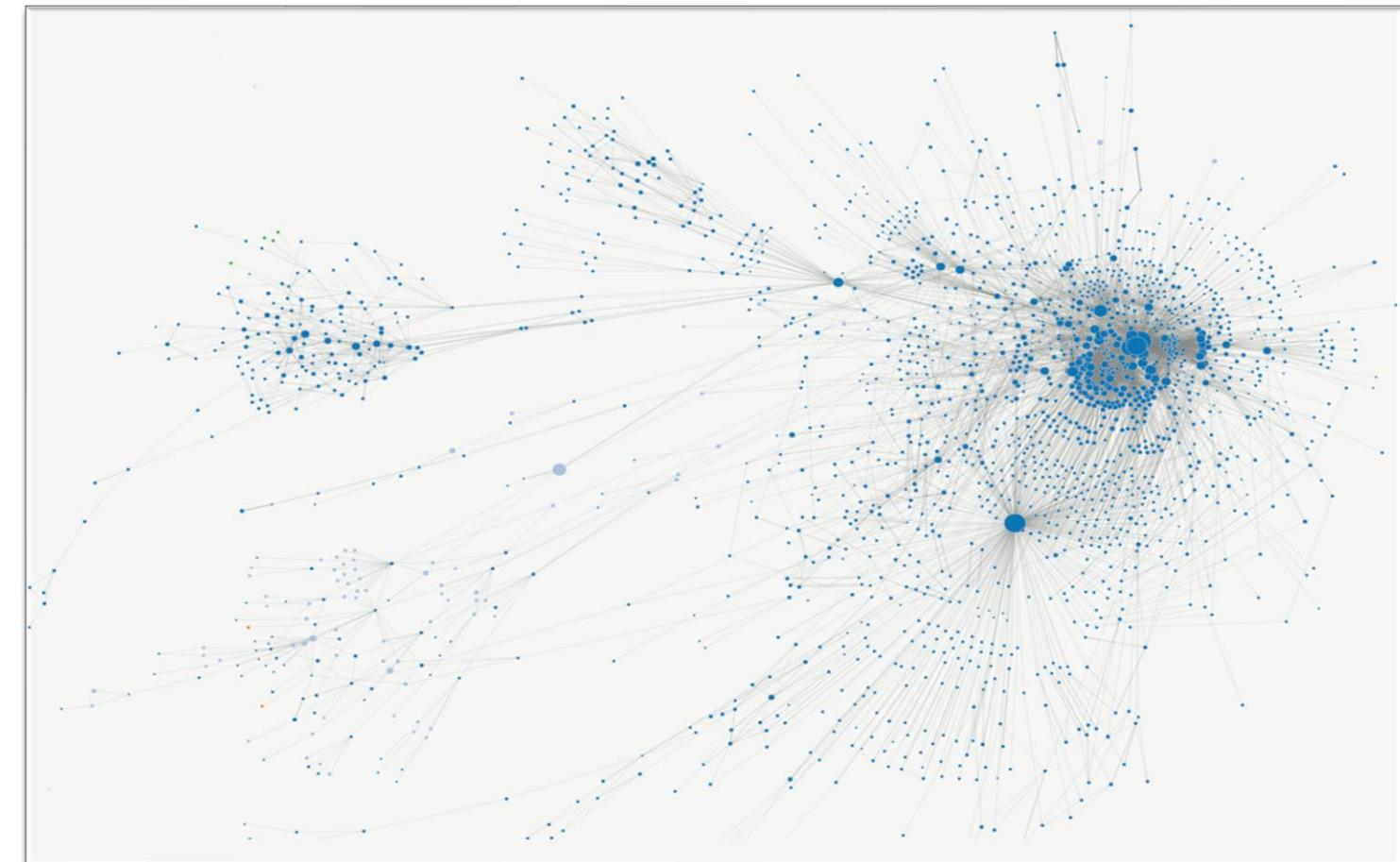
GRAPH REPRESENTATION OF A CDS DATA MODEL
USING THE ASSOCIATION AS EDGES

S/4Hana Development System^[1]

Number of CDS Views: 12.239

Number of Associations 22.847

Lines of Code in CDS Views: 724.656



^[1] on 2016-04-26

Why do we need Testing for “Code Pushdown”-Scenarios ?

Complexity of S/4 CDS Model (2)

TOP Views (with most “basis objects”)

Entity	# Basis Objects
VFCLM_BAM_DDL_BSM_MD5	5.114
C_PURORDREFPURCONITM	3.055
P_ASSOCPURORDITMPURCONITM	3.054
C_PURGINFORECORDFS	3.008
VFCLM_BAM_DDL_BSM_MD4	2.555
VFCLM_BAM_DDL_BSM_MD3	2.553
C_PURCHASINGCATEGORYFS	1.603
C_PURORDBYSUPPLIERINVOICE	1.569
C_CONTRACTITEMFS	1.534
C_PURREQUISITIONITEMFS	1.527
C_PURREQUISITIONSFS	1.519

TOP Views (with most tables)

Entity	#Tables
VFCLM_BAM_DDL_BSM_MD5	4.598
VFCLM_BAM_DDL_BSM_MD4	2.298
VFCLM_BAM_DDL_BSM_MD3	2.298
C_PURORDREFPURCONITM	1.619
P_ASSOCPURORDITMPURCONITM	1.619
C_PURGINFORECORDFS	1.600
C_PURCHASINGCATEGORYFS	838
C_PURORDBYSUPPLIERINVOICE	827
C_CONTRACTITEMFS	813
C_PURREQUISITIONITEMFS	811
I_SUPPLIERCONTACTCARD	807

→ 55 Views exceed “ANYDB limit” of 248 tables

Complete List:



Microsoft Excel
Worksheet

Complete List:



Microsoft Excel
Worksheet

Why do we need Testing for “Code Pushdown”-Scenarios ?

Complexity of S/4 CDS Model (3)

```
[D] [ER9] FCLM_BAM_BSM_MD5 ✎
i 1@AbapCatalog.sqlViewName: 'VFCLM_BAM_BSMMDS'
2 @EndUserText.label: 'Fifth Master Data View for BSM'
3 define view VFCLM_BAM_DDL_BSM_MD5
4   with parameters
5     pDate : abap.dats,
6     pLang : abap.Lang
7   as
8     select from V...
9       left outer join mdtable
10      on mdtable...
11      and dd07t...
12      and dd07t...
13     left outer join countta
14      on countta...
15     left outer join spras
16      on spras...
17      and land1
18   {
19     key mdtable...
20     mdtable.acc...
21     mdtable.bukr...
22     mdtable.azda...
23     mdtable.days...
24     success_count...
25     notmoni_count...
```

Complexity Metrics

Used Datasources	
Number of tables:	4598
Number of views:	501
Number of views with parameter:	15
Number of table functions:	0

SQL Operations	
Number of association definitions:	0
Number of JOIN operations:	135
Number of UNION operations:	4462
Number of GROUP BY clauses:	43

Performance Related Function Calls and Operations	
Number of function calls:	501
Number of explicit type CAST operations:	110
Number of CASE expressions:	18

```
[D] [ER9] C_PURORDREFPURCONITM ✎
i 1@EndUserText.label: 'Purchase Contract Item' //same as
2 @VDM.viewType: #CONSUMPTION
3 @AccessControl.authorizationCheck: #CHECK
4 @AbapCatalog.sqlViewName: 'CPOREFTOPCON'
5 define view C_PurOrdRefPurConItm
6   as select distinct from P_AsocPurOrdItmPurConItm
7
8   association [0..*] to
9     on $projection.Pur...
10    $projection.Pur...
11
12 { key P_AsocPurOrdI...
13   key P_AsocPurOrdI...
14   key P_AsocPurOrdI...
15
16   @Semantics.uni...
17   OrderQuantityU...
18
19   @Semantics.qua...
20   ReleasedQuantit...
21
22   @Semantics.qua...
23   TargetQuantity...
24
25   @ObjectModel.t...
```

Complexity Metrics

Used Datasources	
Number of tables:	1619
Number of views:	1316
Number of views with parameter:	80
Number of table functions:	40

SQL Operations	
Number of association definitions:	1711
Number of JOIN operations:	1526
Number of UNION operations:	132
Number of GROUP BY clauses:	42

Performance Related Function Calls and Operations	
Number of function calls:	1316
Number of explicit type CAST operations:	943
Number of CASE expressions:	426

Why do we need Testing for “Code Pushdown”-Scenarios ?

Here is the Answer...

Resulting Questions (based on such complex CDS Models)

- How to ensure consistency across the different layers of the View stack ?
- How to avoid regressions ?
- How to provide supportability ?

Potential Answers

- Use Debugging
 - SQL doesn't offer debugging “by definition”
- Use (Unit) Testing
 - the only “technically feasible” option to ensure the above qualities!

Recommendation for Test Automation Tools S/4 HANA

Test Type	Purpose	Tools	Examples
System Tests	E2E scenario test of complete stack	<u>START/eCATT</u>	Order to Cash
Integration Tests	Integration test of multiple components	UI: <u>START/eCATT</u> , Non-UI: <u>eCATT oData Assistant</u> , eCATT	Fiori App ⇄ backend communication
Unit/Component Tests	Test of single units & components build out of multiple units	Frontend: {QUnit/OPA5/mockserver} Backend: {ABAP Unit/TDF}, Avalon (CDS)	oData service, ABAP API, Fiori App (isolated)
Static Code Checks	Find possible issues via static code scans	Frontend: ESLint Backend: ATC	Disallow Unused Variables

CDS Test Double Framework addresses this block

manual & automated

automated

Unit Testing

Unit testing is a software methodology to test portions of code, to ensure they are fit for the intended purpose. These portions of code can be described as smallest testable element of an application.

In Object Oriented programming a method or a class can be considered as unit. With code pushdown a CDS view or a database procedure can be considered as unit.

Dependencies

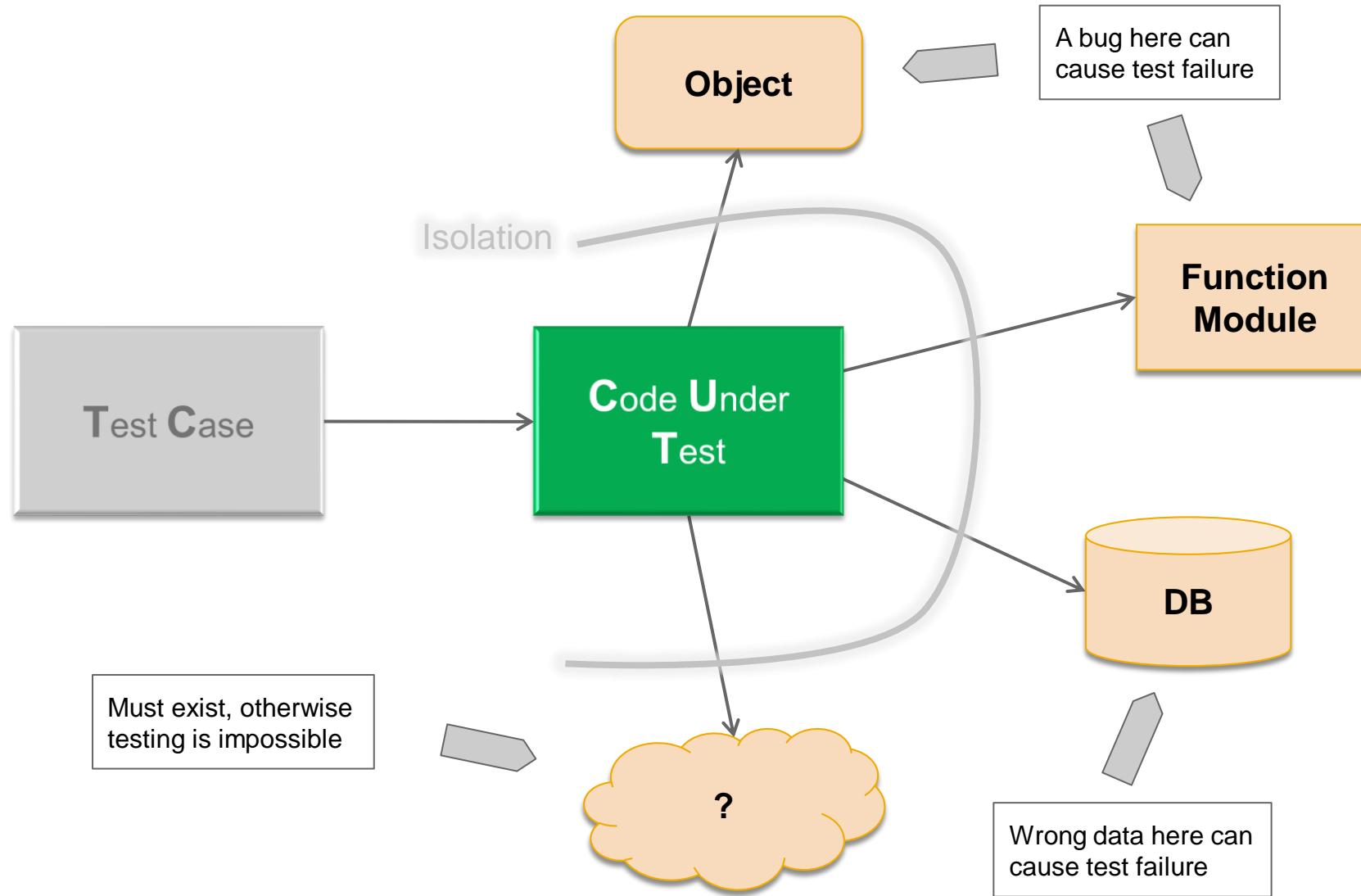
The portion of code that shall be tested is also called Code Under Test (CUT). Typically the CUT invokes other code or relies on data sources like a database table. Such dependencies are also called Depended-on Component (DOC).



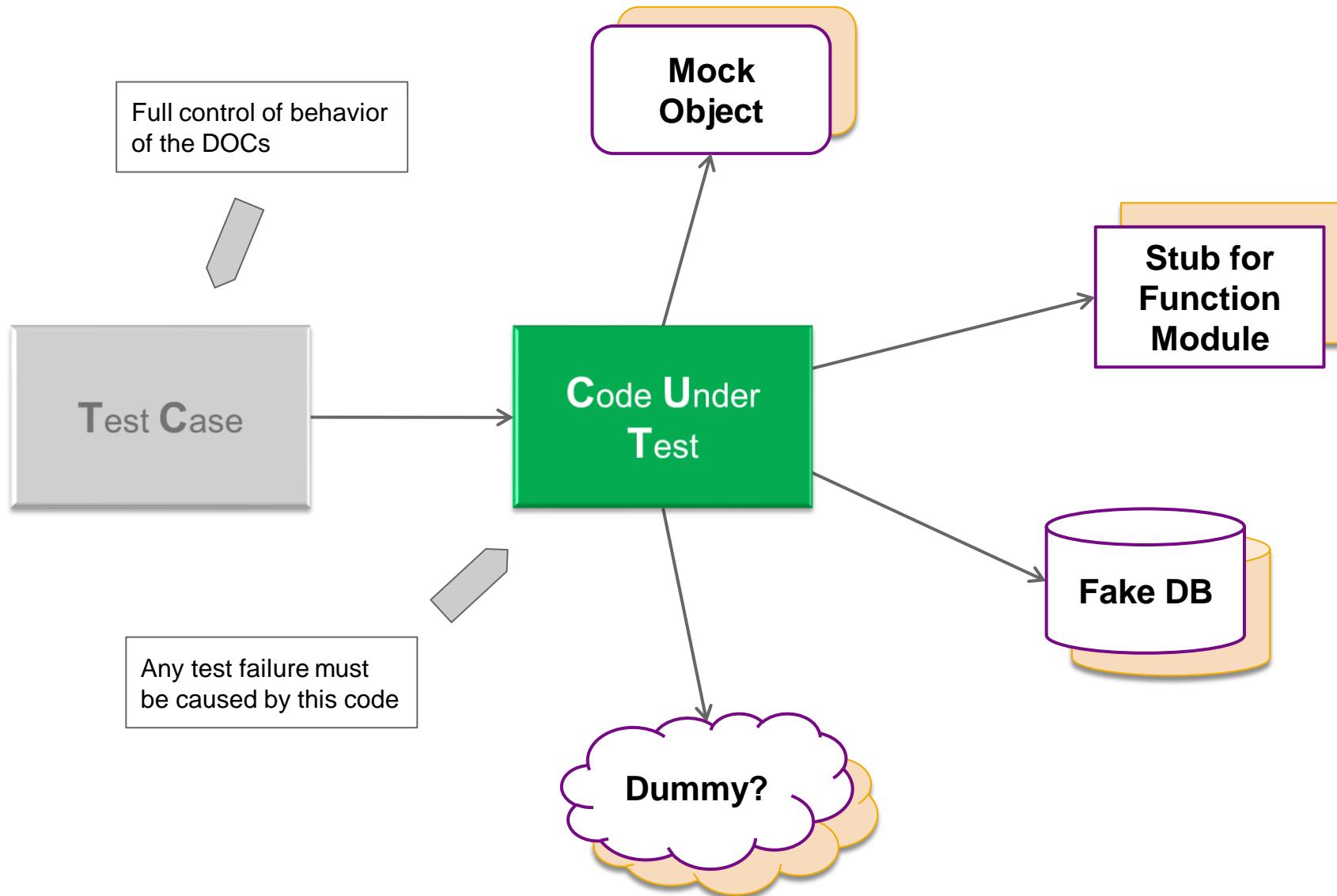
The dependency to database table content can prevent a successful unit test execution in case the assumed content is not available however.

⇒ In order to achieve stable unit tests it is required to isolate the CUT from any test hindering DOC.

Testing without Isolation

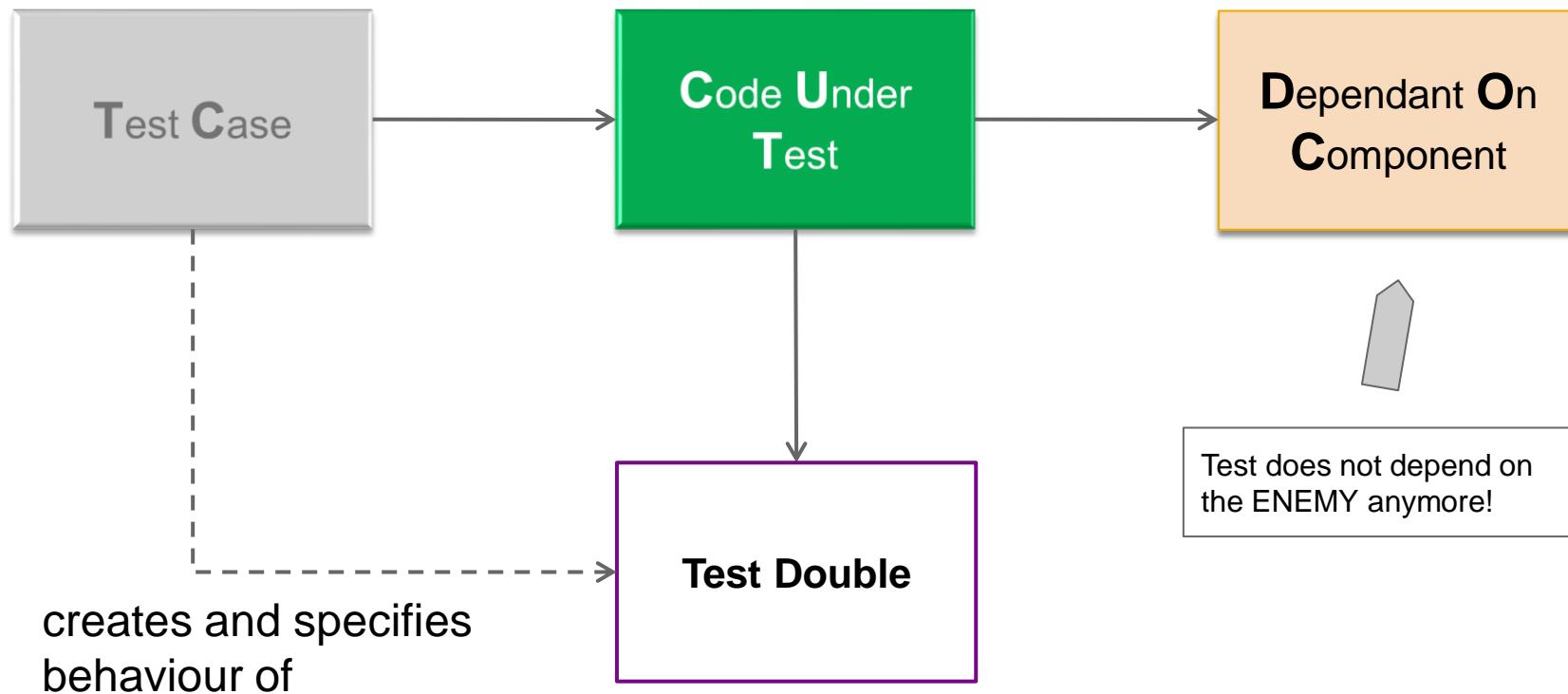


Testing with Isolation (Test Doubles)

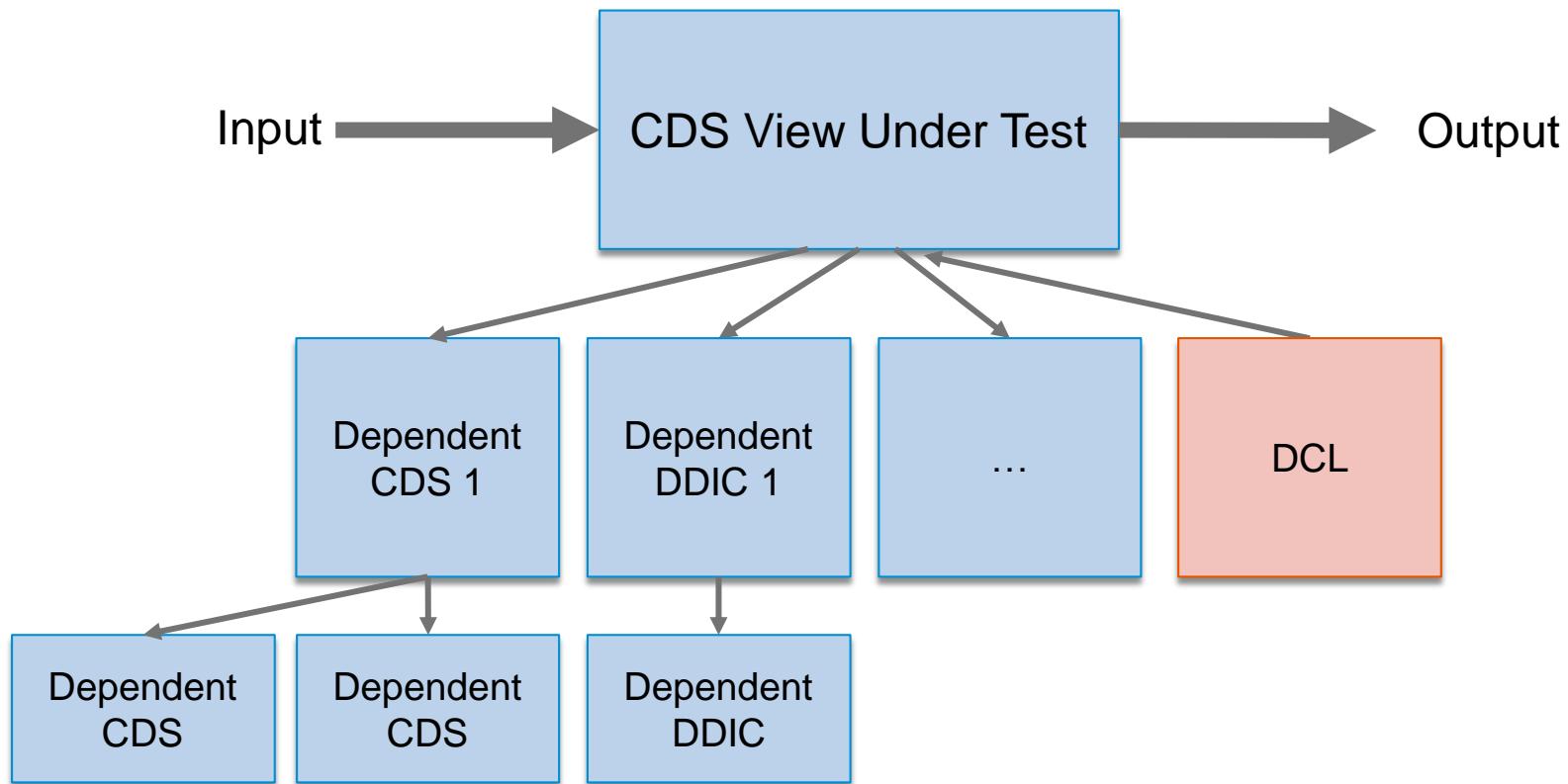


Dependency Injection – Test Doubles

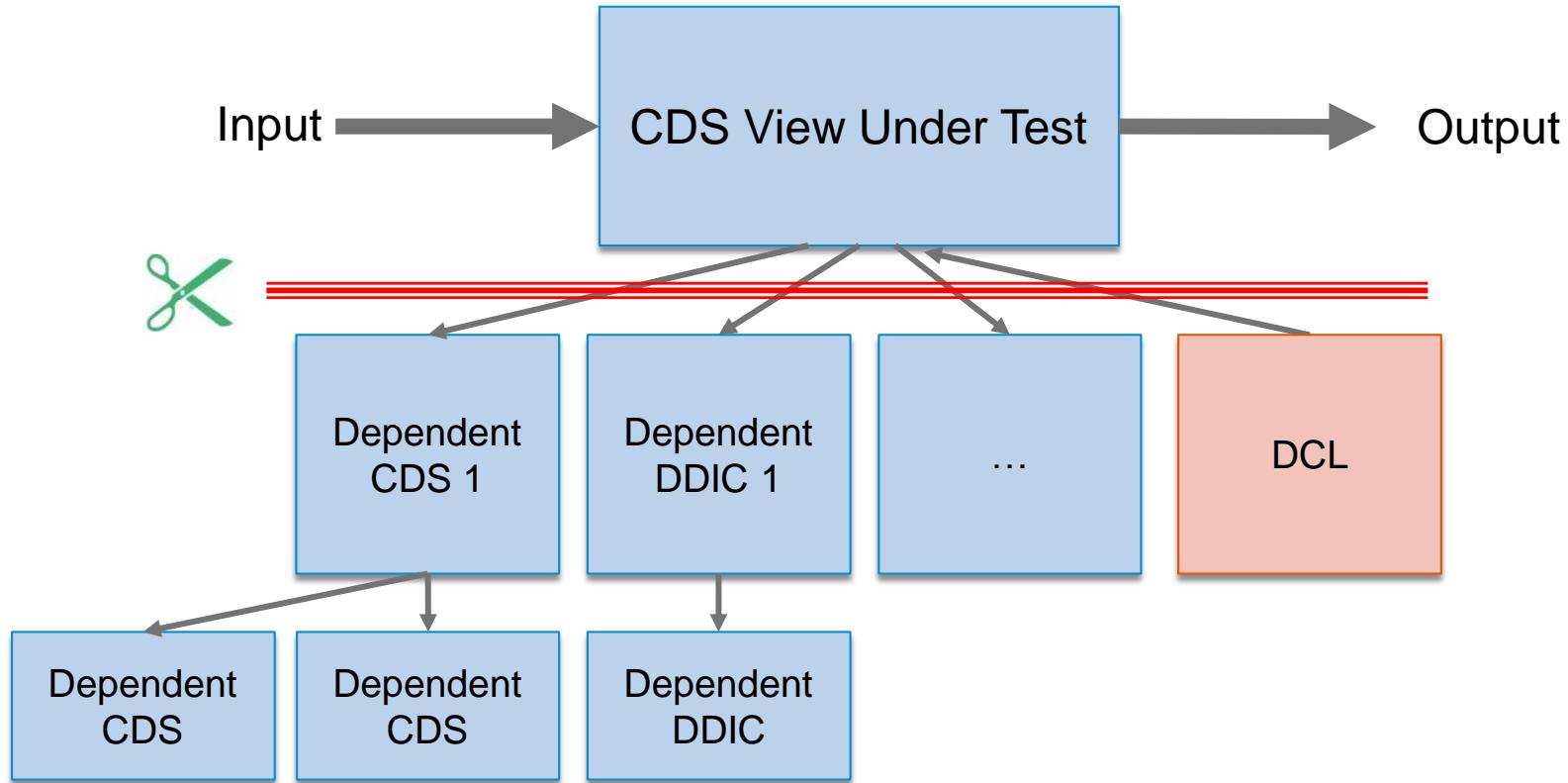
A common practice is to replace a DOC with test unfriendly behavior by a substitute with test friendly behavior. Such a substitute is also called Test Double.



CDS Under Test

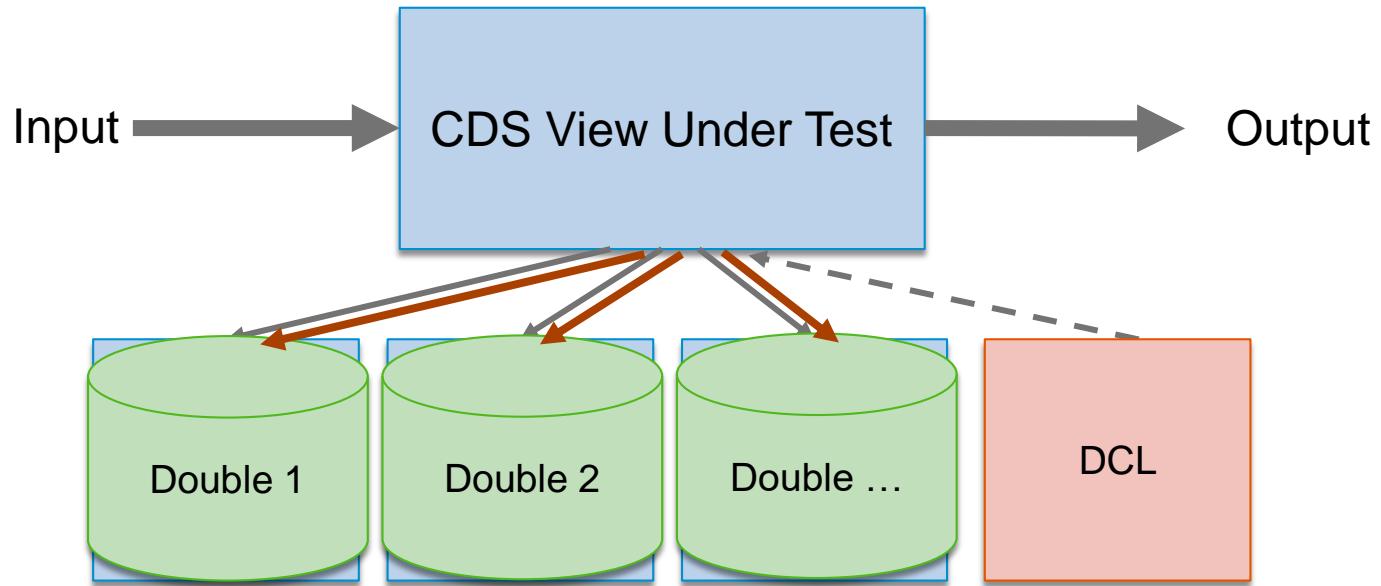


CDS Test Double Framework



Identify dependencies (DOCs) to CDS Under Test

CDS Test Double Framework



Create Doubles for all the first level dependencies (DOCs) of CDS Under test and insert test data into these doubles

Writing Unit Tests – Demo

- **Migration from old to new Avalon:**
 - How to migrate?
 - <https://wiki.wdf.sap.corp/wiki/display/NWCUIAMIndia/Migrating+from+Avalon+to+CDS+Test+Double+Framework>
- Migration Efforts – very less!
- **More general information about the CDS test double framework is available here:**
 - How To Guide: <https://wiki.wdf.sap.corp/wiki/display/NWCUIAMIndia/CDS+Test+Double+Framework>
 - JAM Group: https://jam4.sapjam.com/groups/about_page/w6o8TCeK3emJLDBLeEqu6h

Useful links

- **Jam Group links for CDS Test Doubles:**
 - https://jam4.sapjam.com/groups/w6o8TCeK3emJLDBLeEqu6h/overview_page/dbpUfbEiwvbG5SkIHHditm
 - <https://jam4.sapjam.com/blogs/show/nZfHhotsq91LUELb0WtiTt>
- **Wiki Page:**
 - <https://wiki.wdf.sap.corp/wiki/display/NWCUIAMIndia/CDS+Test+Double+Framework>
- **Info session Recording CDS Test Double Framework**
 - <https://wiki.wdf.sap.corp/wiki/display/SuiteCDS/Online+Trainings#OnlineTrainings-CDSViewComplexity>

Useful links – Cont.

- **Blogs:**
- <https://blogs.sap.com/2016/10/19/introduction-cds-test-double-framework-write-unit-tests-abap-cds-entities/>
-
- **Demo Examples can be find in this package:**
- SABP_UNIT_DOUBLE_CDS_DEMO

Contact Person for CDS Test Doubles:

Sunil Bandameedapalli

AS ABAP Release and S/4 Release mapping

ABAP Release for CE	CE Release	ABAP Release for OP	OP Release
7.76	1905	-	-
7.75	1902	-	-
7.74	1811	-	-
7.73	1808	7.53	1809
7.72	1805	-	-
7.71	1802	-	-
7.70	1711	-	-

ABAP Release for CE	CE Release	ABAP Release for OP	OP Release
7.69	1708	7.52	1709
7.68	1705	-	-
7.67	1702	-	-
7.66	1611	-	-
7.65	1608	7.51	1610
7.64	1605	-	-
7.63	1603	-	-
7.62	1511	-	-
7.62	1511	7.50	1511
7.61	1506	-	-
7.60	1503	-	-

CDS Hierarchy - Syntax

```
[DEFINE] HIERARCHY cds_entity
  [parameter list]
  AS PARENT CHILD HIERARCHY(
    SOURCE cds_view
    CHILD TO PARENT ASSOCIATION _hierarchy_assoc
    [/ DIRECTORY _directory_assoc FILTER BY cond_expr]
    START WHERE cond_expr
    SIBLINGS ORDER BY field1 [ASCENDING/DESCENDING][,
      field2 [ASCENDING/DESCENDING], ...]
    [/ DEPTH depth]
    [/ NODETYPE node_type]
    [/ MULTIPLE PARENTS {NOT ALLOWED}/LEAVES/ALLOWED]
    [/ ORPHANS ERROR/IGNORE/ROOT]
    [/ CYCLES ERROR/BREAKUP]
    [/ GENERATE SPANTREE] )
  { element list }
```

CDS Hierarchies

Available from 7.73 SAP
Development Learning

Hierarchy Definition

```
define hierarchy DEMO_CDS_SIMPLE_TREE
  with parameters
    p_id : abap.int4
  as parent child hierarchy(
    source
      DEMO_CDS_SIMPLE_TREE_SOURCE
      child to parent association _tree
      start where
        id = :p_id
      siblings order by
        id ascending
    )
    {
      id,
      parent,
      name
    }
```

View Definition

```
@AbapCatalog.sqlViewName: 'DEMOTREESOURCE'
@AccessControl.authorizationCheck: #NOT_REQUIRED
define view DEMO_CDS_SIMPLE_TREE_SOURCE
  as select from
    demo_simple_tree
    association [1..1] to
    DEMO_CDS_SIMPLE_TREE_SOURCE as _tree on
      $projection.parent = _tree.id
    {
      _tree,
      id,
      parent_id as parent,
      name
    }
```

Two Ways of Database Integration

Even though the Core Data Services Specification is language and platform agnostic,
the CDS implementations are not.

ABAP CDS

- Same Semantic on all DBs
- Initial Focus on View building
- Integration into ABAP Dictionary

HANA CDS

- HANA only
(HANA specific features available)
- Initial Focus on Building Models from Scratch

Further References

ABAP Application Infrastructure

<https://wiki.wdf.sap.corp/wiki/display/A4H/ABAP+Application+Infrastructure>

HANA Core Data Services

<https://wiki.wdf.sap.corp/wiki/display/ngdb/HANA+Core+Data+Services>

Domain-specific Annotations in CDS

<https://wiki.wdf.sap.corp/wiki/download/attachments/1153423523/Domain-specific%20Annotations.docx?version=1&modificationDate=1441273752000&api=v2>

Fiori Architecture Blueprints

<https://wiki.wdf.sap.corp/wiki/display/fioritech/Fiori+Tech+Programming+Model>

Development Guidelines

<https://wiki.wdf.sap.corp/wiki/display/fioritech/Development+Guideline+Portal>

SAP UI5

<http://scn.sap.com/community/developer-center/front-end>

Further References

SAP Fiori and developing UIs using this technology

[Fiori Curriculum Jam.](#)

[programming guide](#) and an

["Build SAP Fiori UI - End-2-End-Sample"](#)

Open Data protocol (short: OData)

<https://wiki.wdf.sap.corp/wiki/display/NWA/OData+self-paced+training>

Gateway

Customer course [GW100](#).

HANA SQL(Script)

[ZHA150 SQL Basics for HANA](#), Unit 10.

Connecting Customer systems using ABAP in Eclipse

<https://wiki.wdf.sap.corp/wiki/display/aie/AiE+and+customer+systems+-+How+to>