ZS4HCDS
S/4HANA Development Fundamentals

SAP Development Learning
May, 2020

Course agenda – 3 days

SAP Development Learning

Day 1

- CDS Introduction
- ABAP Development Tools
- Annotations
- Basic SQL features in CDS
  - Expressions
  - Functions
  - Aggregations
  - Joins
  - Union

Day 2

- Input parameters
- Associations
- View Extension
- Metadata Extension

Day 3

- Data Control Language
- Fiori Programming Model
- AMDP and CDS Table Function
- Virtual Data Modeling

SAP Development Learning

Unit 1 – Introduction to Core Data Services

Lesson 1 – Motivation and Definition

SQL Overview

SAP Development Learning

- **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, ...

ABAP SQL (aka Open SQL)

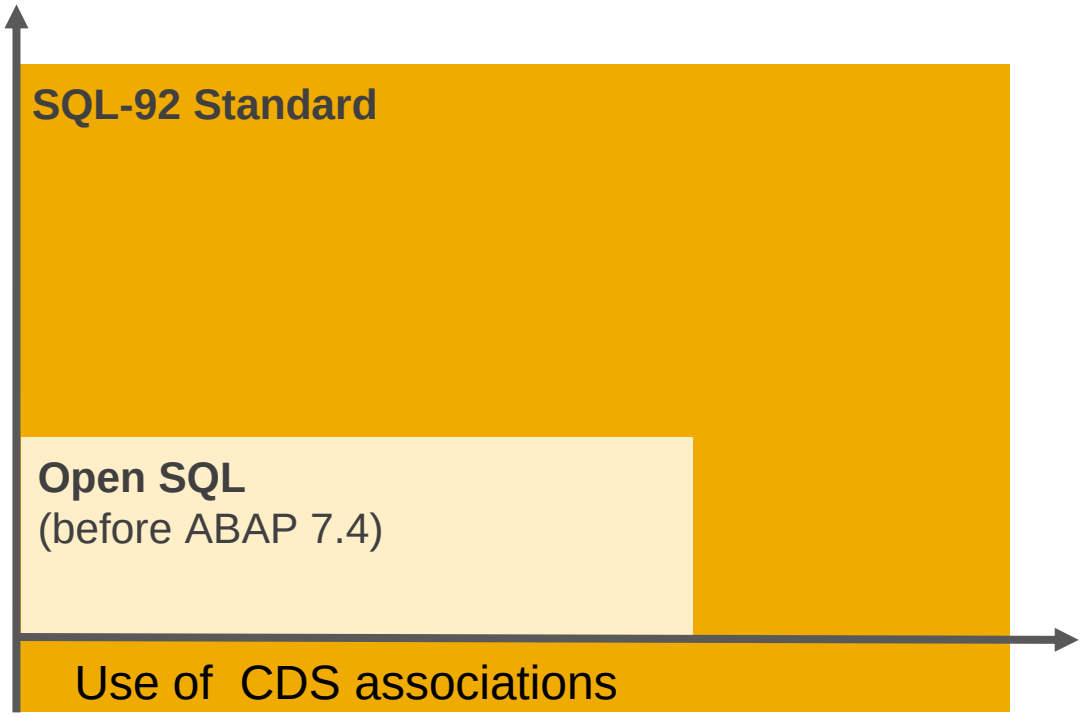
SAP Development Learning

- **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

SAP Development Learning

Limitations in ABAP < 7.4 SP05



ABAP Dictionary Views

SAP Development Learning

- **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)

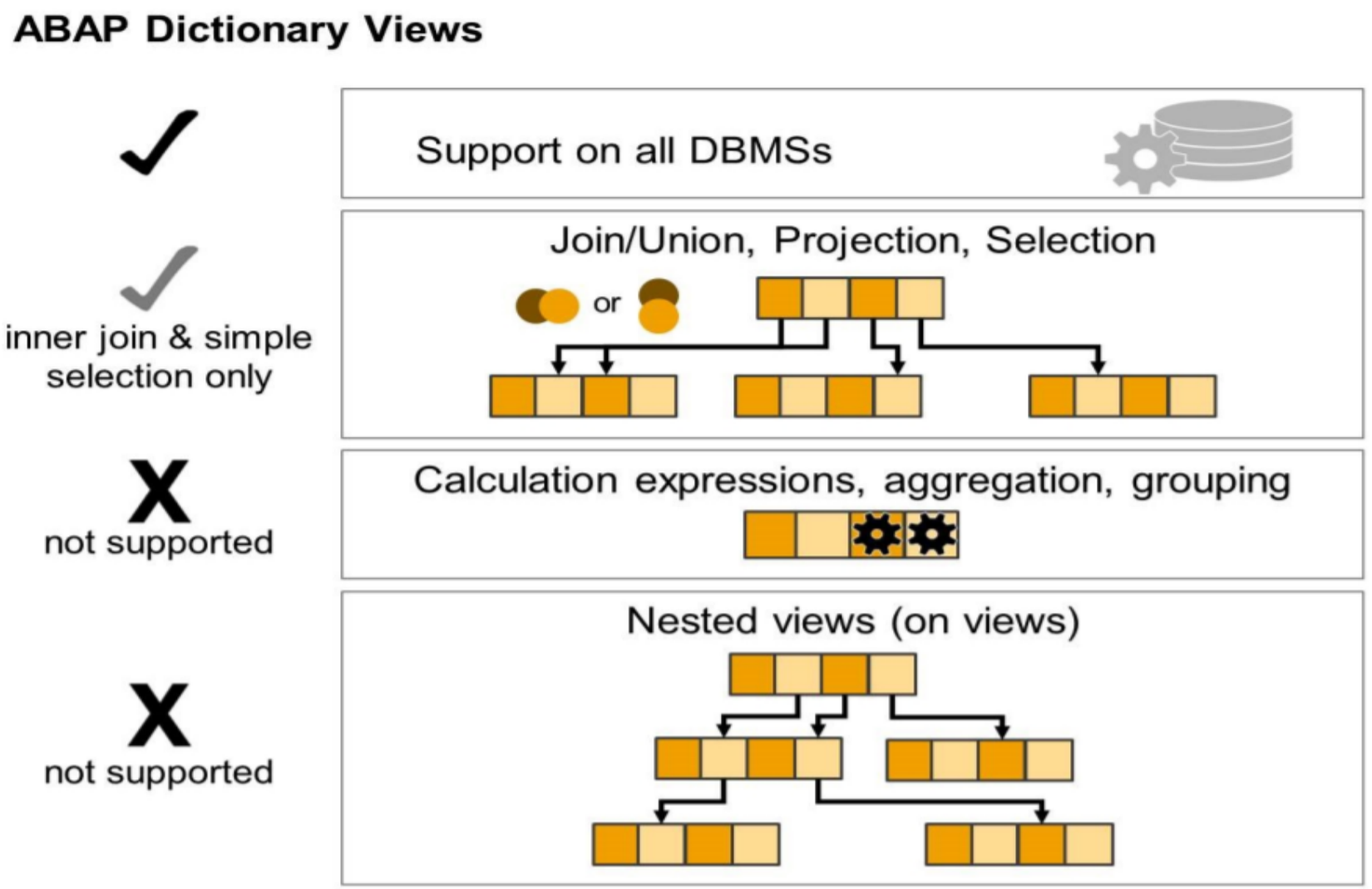
Limitations of ABAP Dictionary Views

SAP Development Learning

- **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)

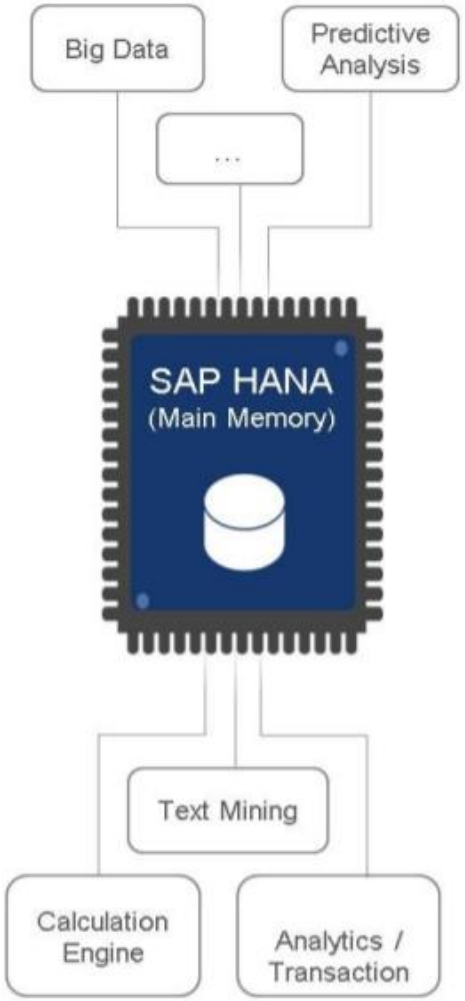
Limitations of ABAP Dictionary Views

SAP Development Learning



The SAP HANA Platform

SAP Development Learning



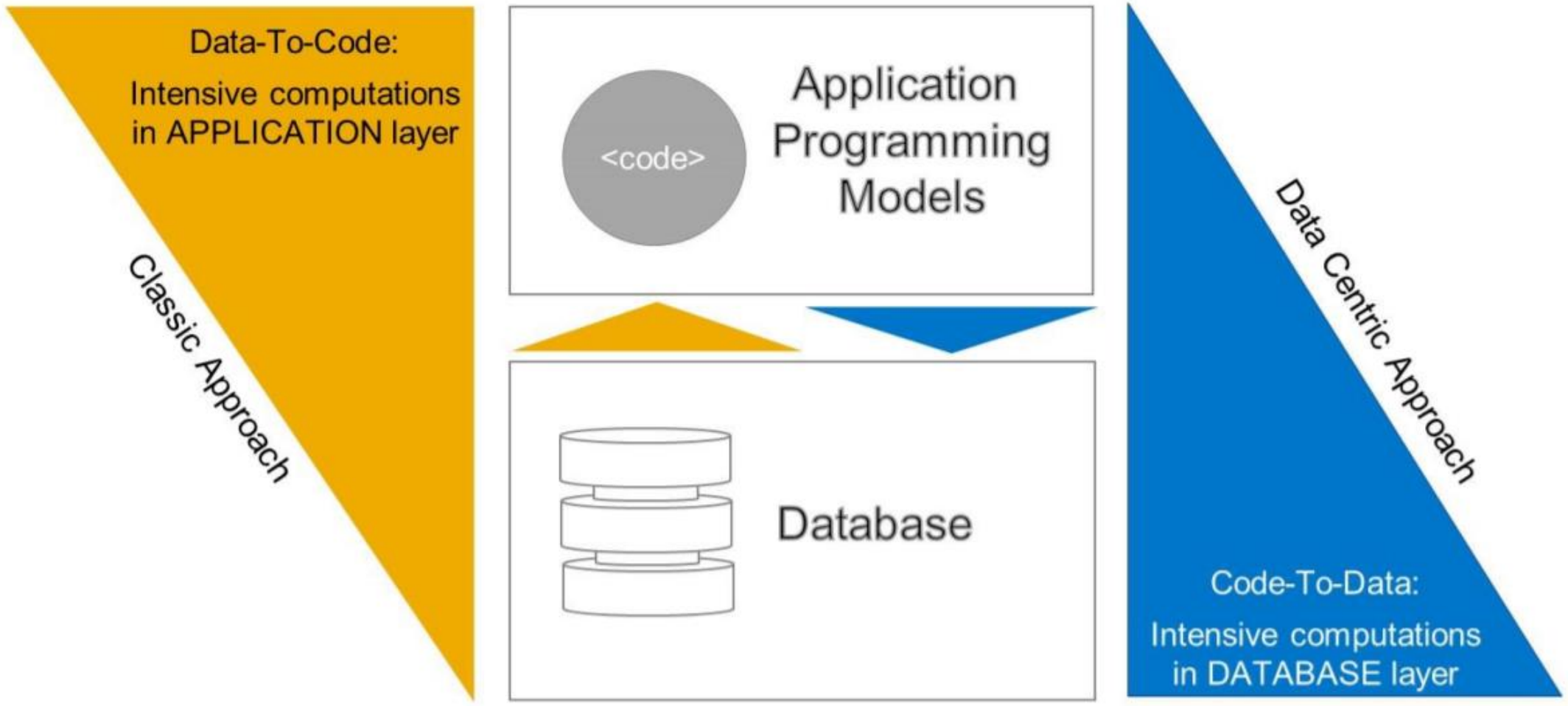
SAP HANA takes advantage of the latest technology developments

And changed the way of developing and executing applications

100%	In-Memory computing OLTP & OLAP in real-time		No	Aggregates On-the-fly data models without duplicates
	Column and row storage		Less	Indices Flexible and fast retrieval of the dataset
5-50x	Compression Based on column storage		Less	Code lines Less complexity in data models and code
10-10,000x	Acceleration Massive parallelization		Parti-tioning	Mass data analysis

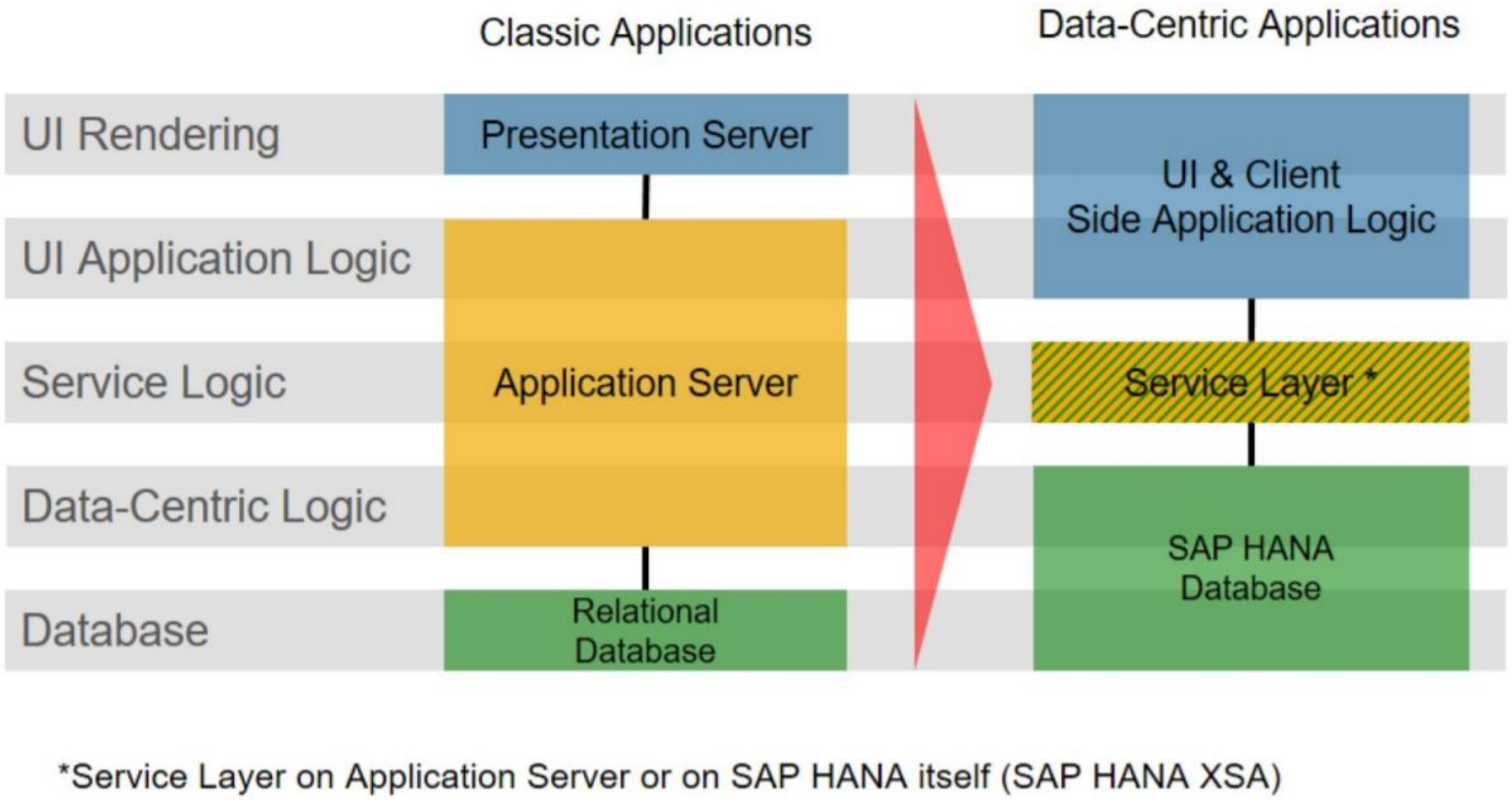
Paradigm changes in application programming

SAP Development Learning



Transformation of Application Design with SAP HANA

SAP Development Learning



*Service Layer on Application Server or on SAP HANA itself (SAP HANA XSA)

Core Data Services – A Family of Domain Specific Languages

SAP Development Learning

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

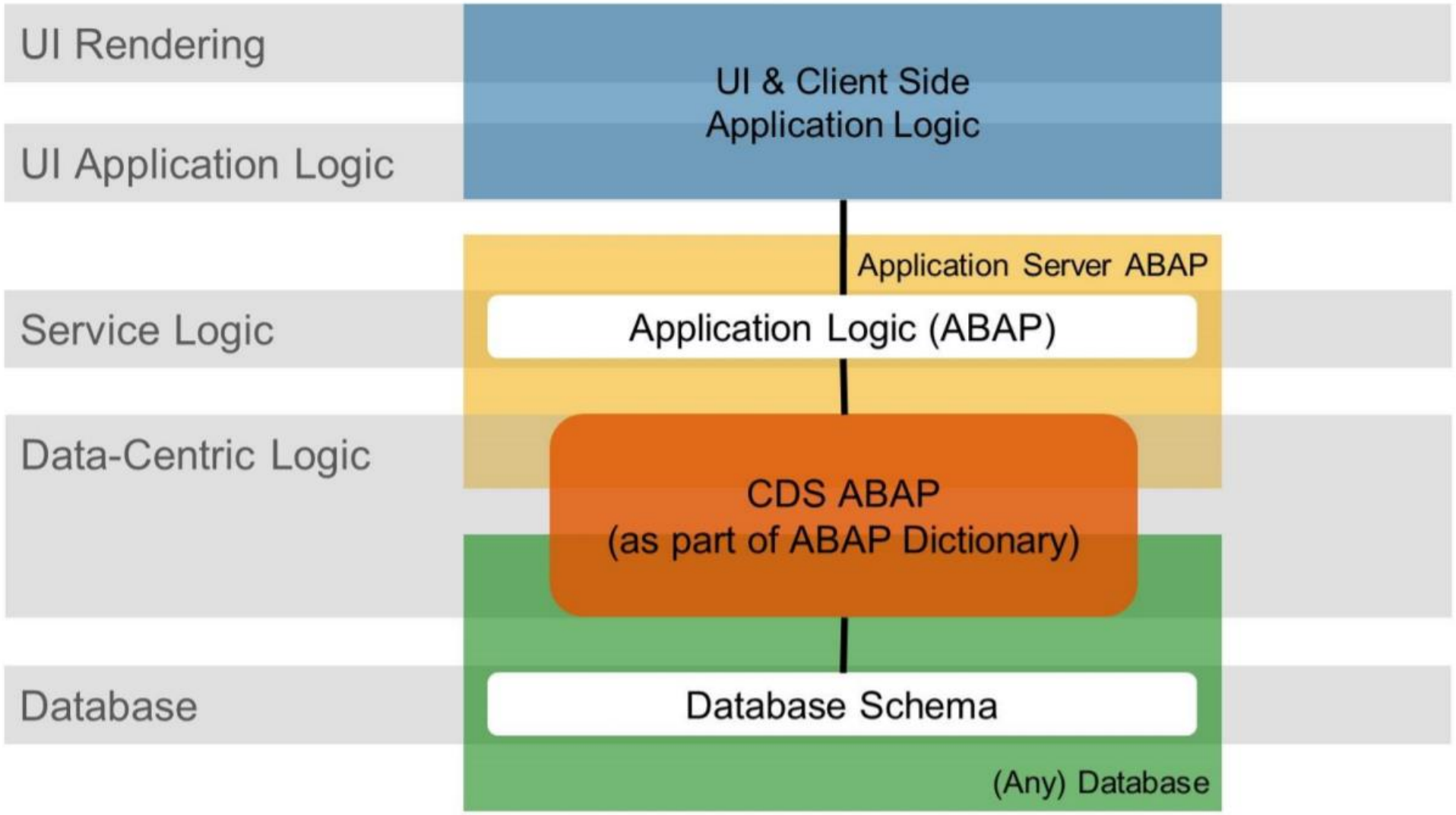
Define authorizations for CDS views

Modelled and declarative approach

Integrates with classic authorization concepts

Core Data Services on ABAP

SAP Development Learning



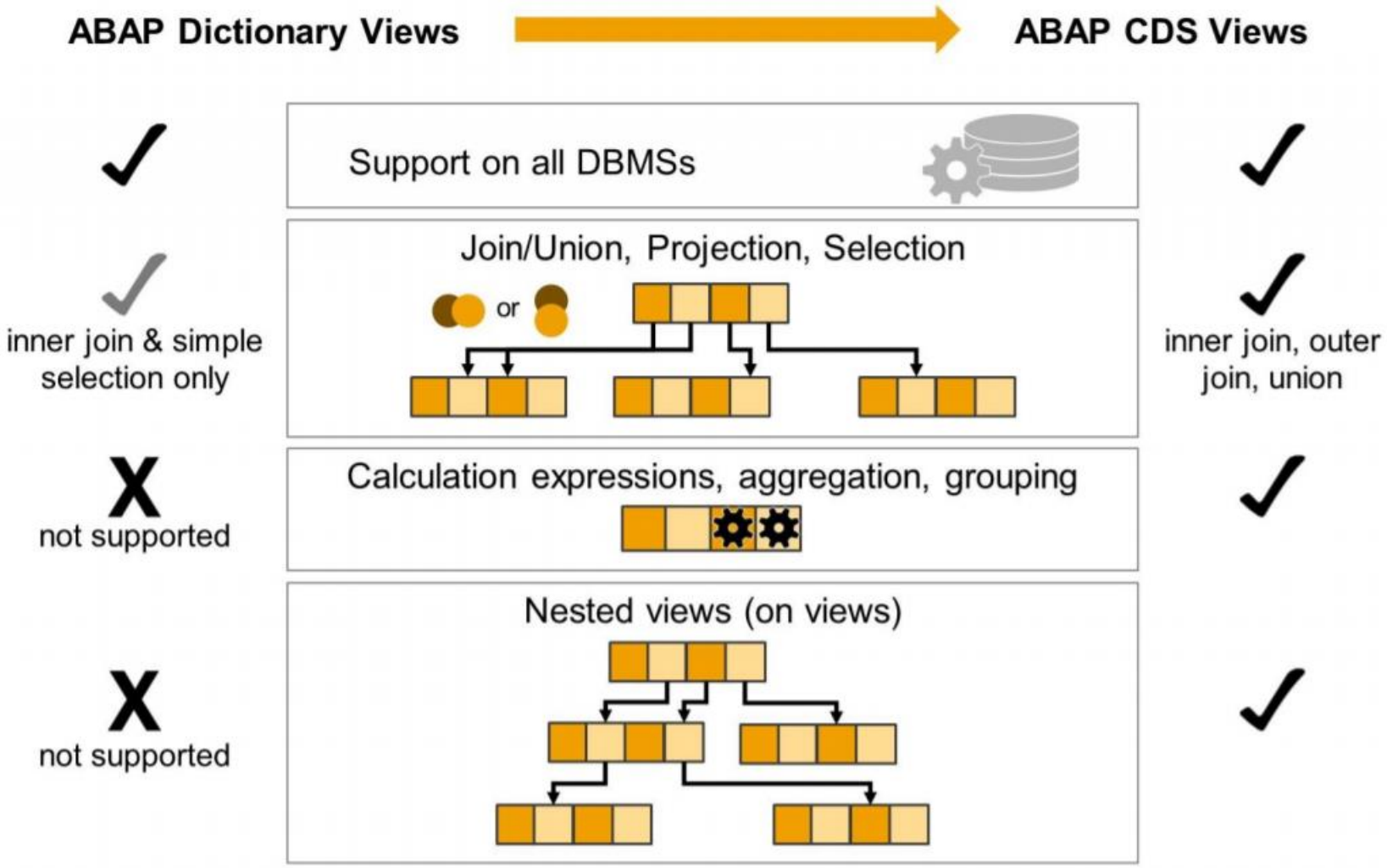
Basic features of ABAP CDS

SAP Development Learning

- **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)

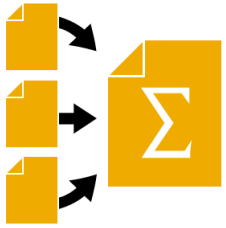
Advanced view building ABAP CDS

SAP Development Learning



Lesson 1 – Motivation and Definition
Summary

SAP Development Learning



You should now be able to:

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

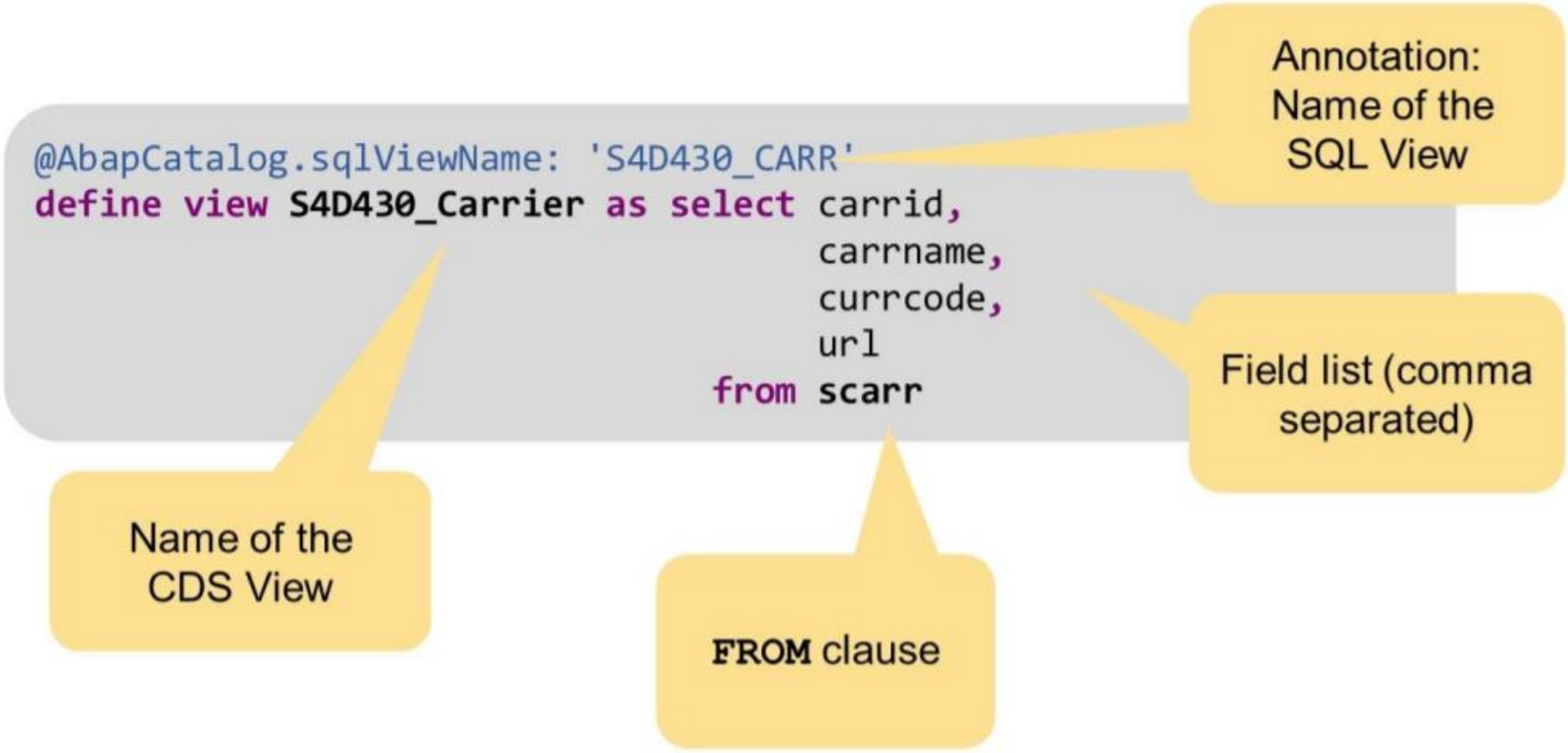
SAP Development Learning

Unit 1 – Introduction to Core Data Services

Lesson 2 – Basic Syntax of CDS Views

DDL Source: CDS View and SQL View

SAP Development Learning



Variants for specifying the Element List

SAP Development Learning

Standard SQL (After keyword SELECT)

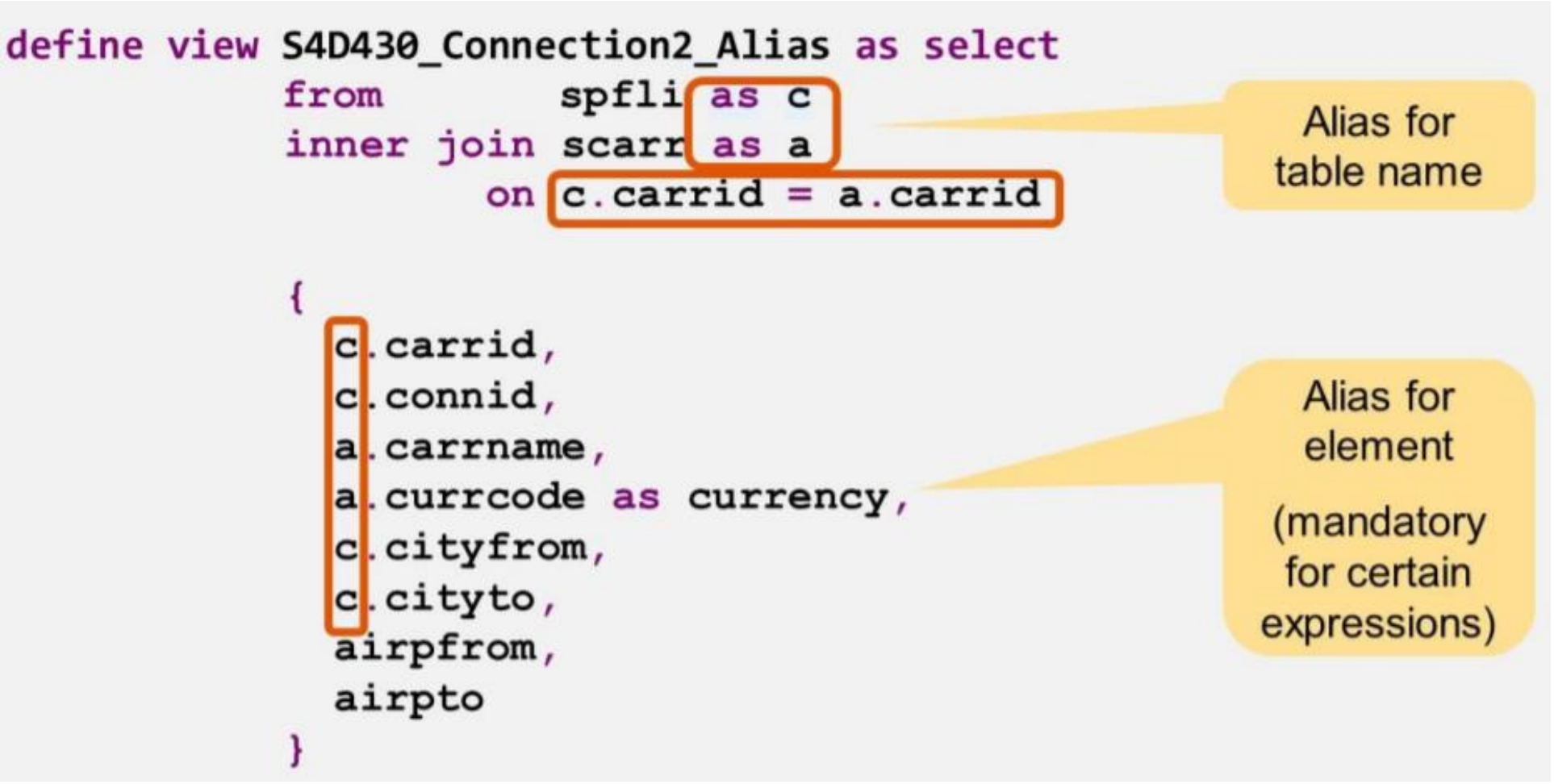
```
@AbapCatalog.sqlViewName: 'S4D430_CARR'
define view S4D430_Carrier as select carrid,
                                     carrname,
                                     currcode,
                                     url
                                from scarr
```

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 }
```

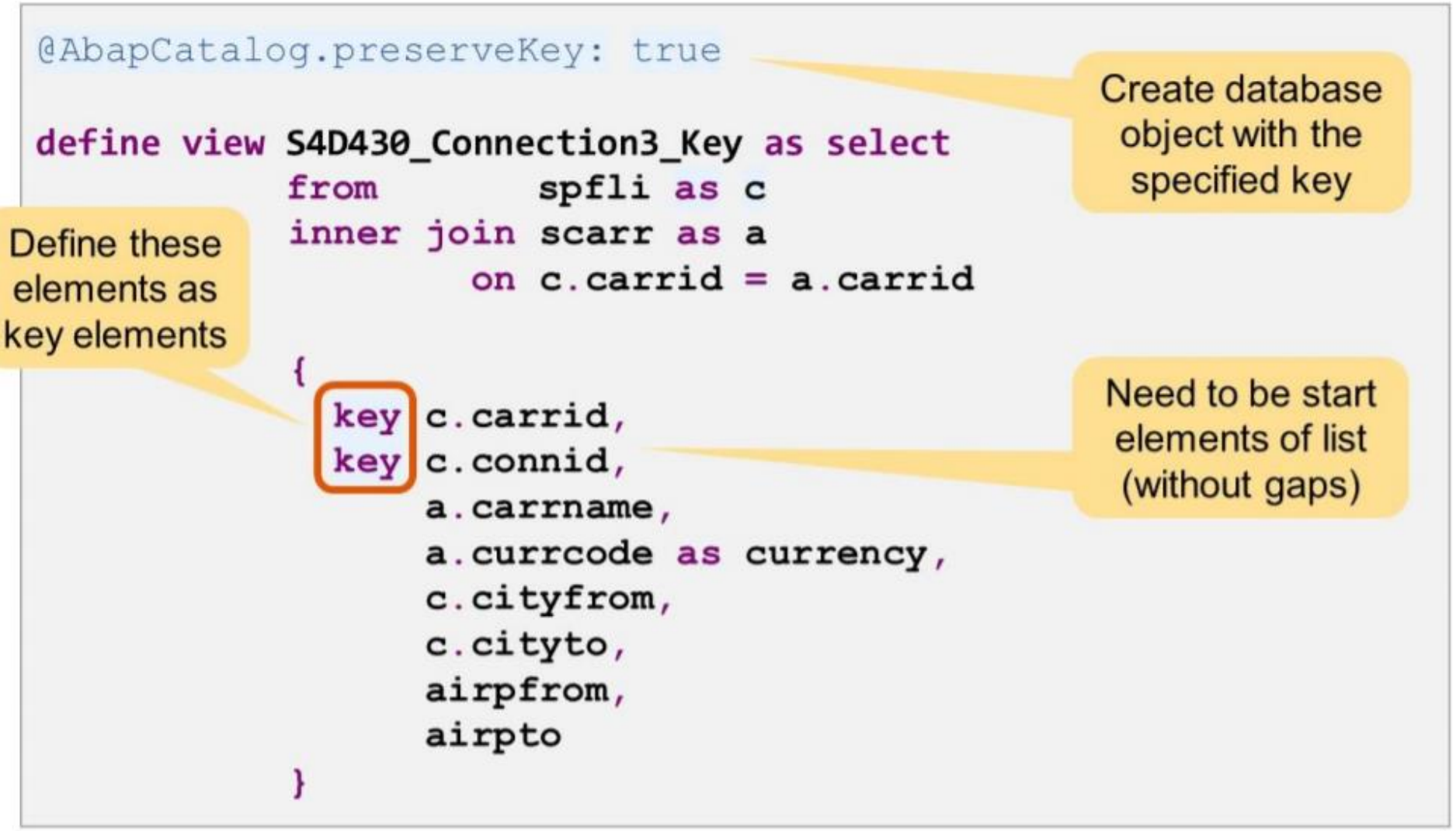
Aliases for Tables and Fields

SAP Development Learning



Key Elements

SAP Development Learning



WHERE

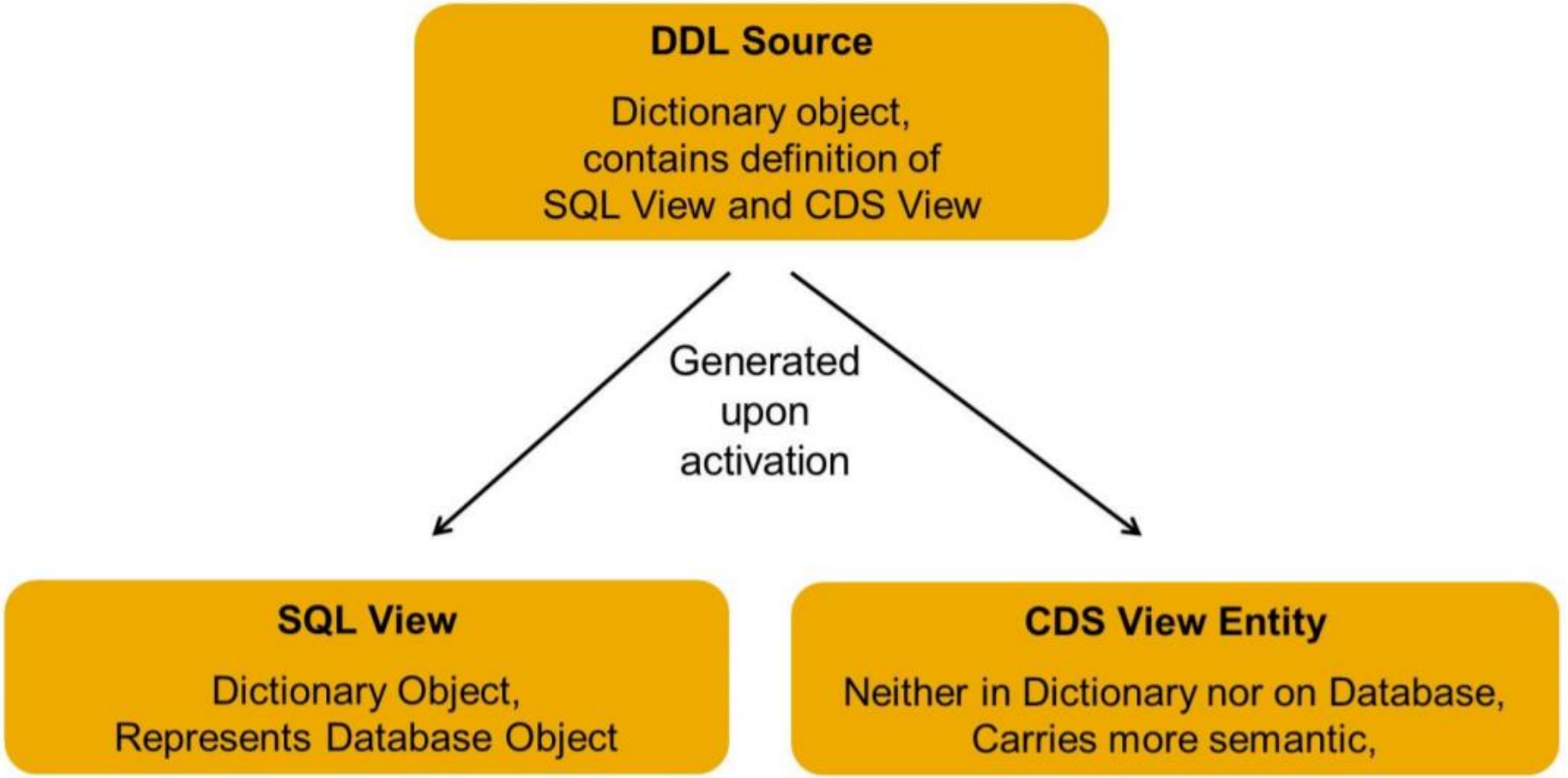
SAP Development Learning

- 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'
```

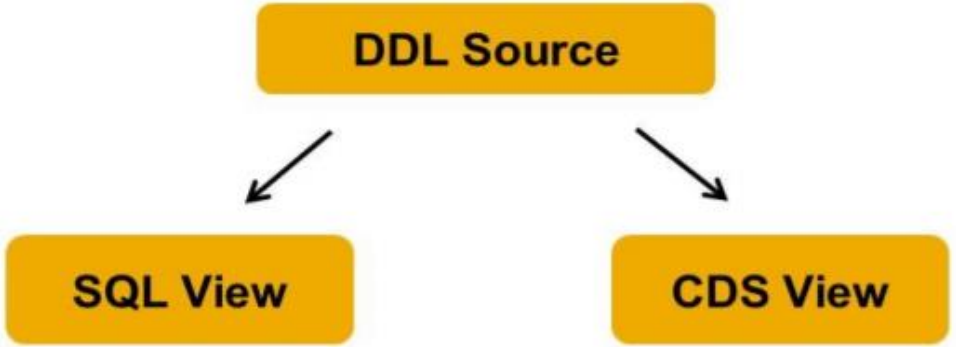
One DDL Source - Define Two Objects

SAP Development Learning



SQL View vs CDS View

SAP Development Learning



SQL View …

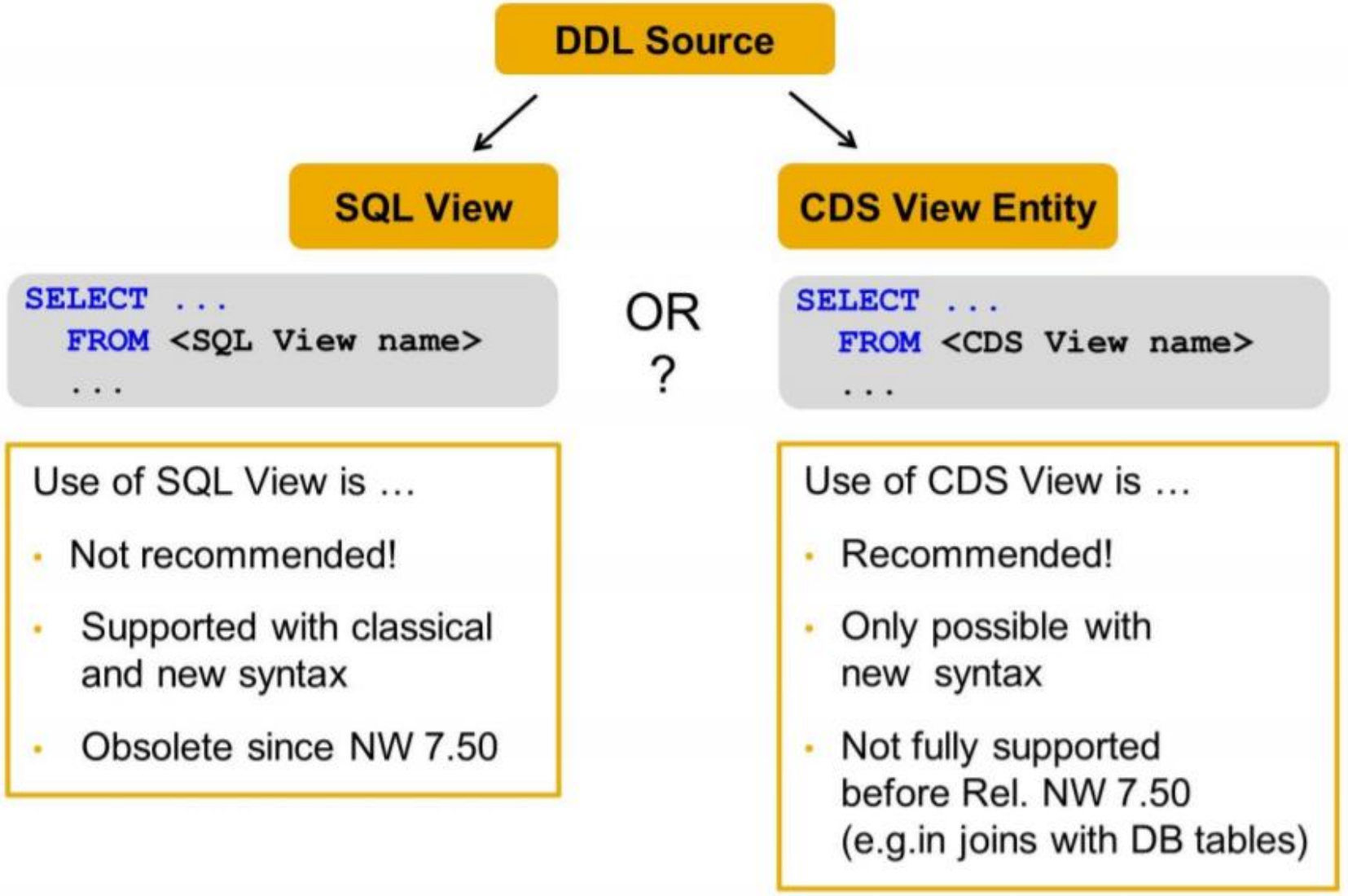
- Repository Object (Dictionary)
- Represents “real“ database object
- Is an ABAP data type (structure, with client field)
- No additional semantics

CDS View …

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

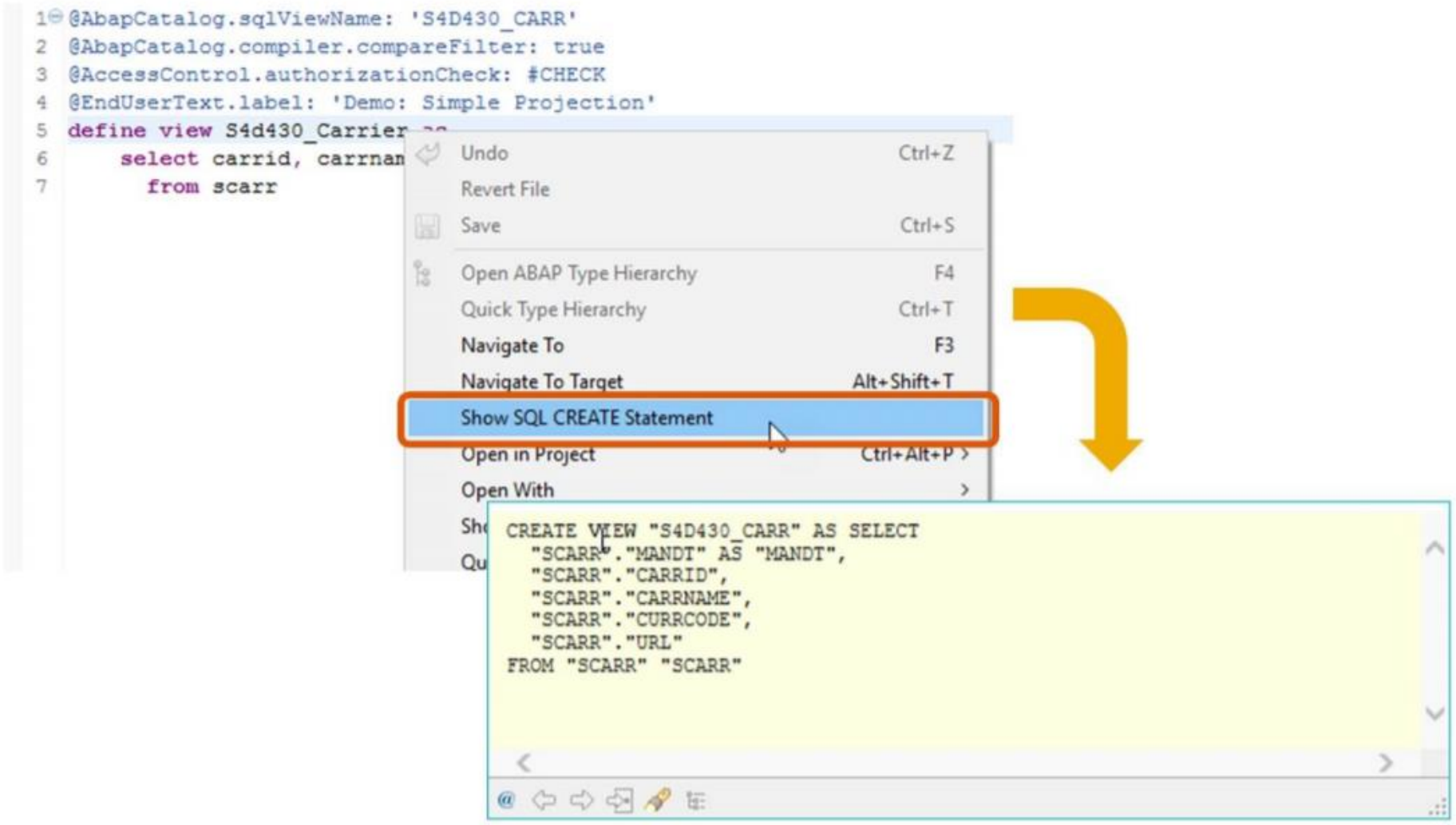
SQL View vs CDS View

SAP Development Learning



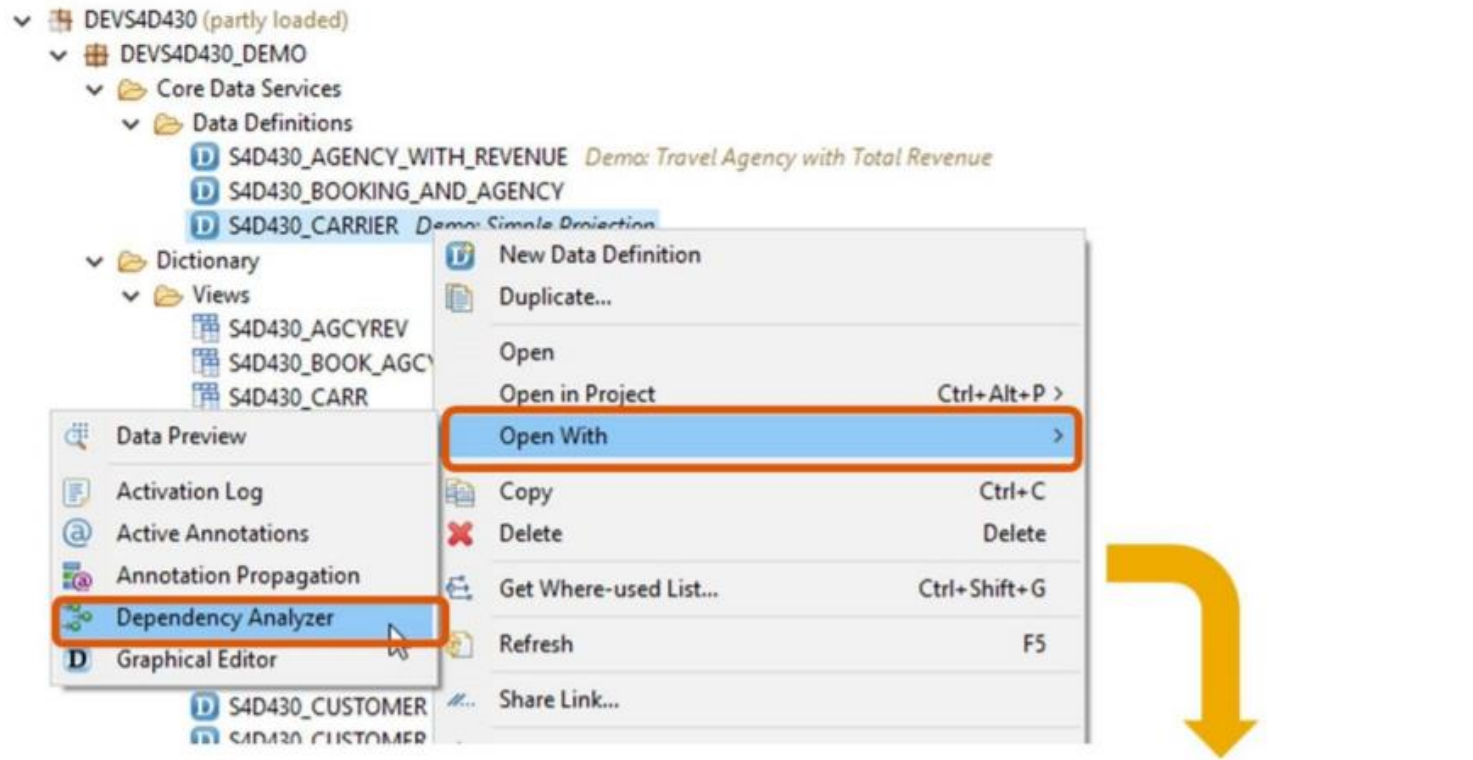
Display SQL Create Statement

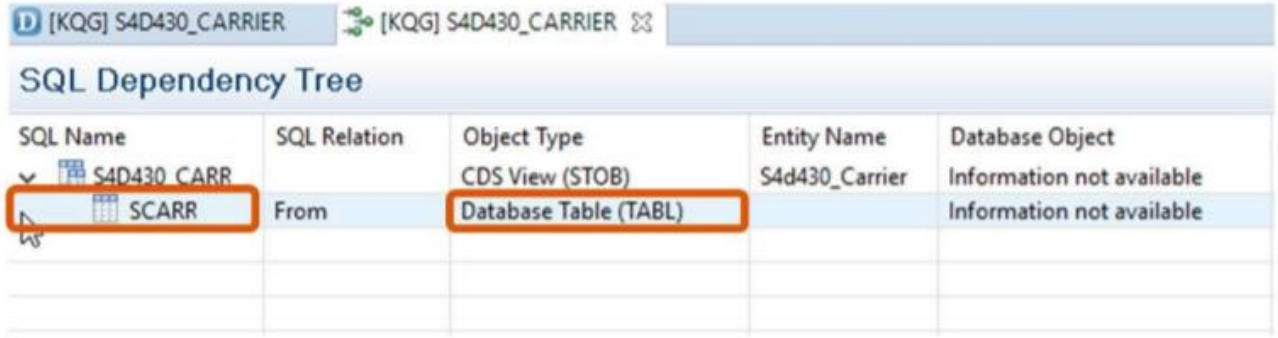
SAP Development Learning



Dependency Analyzer

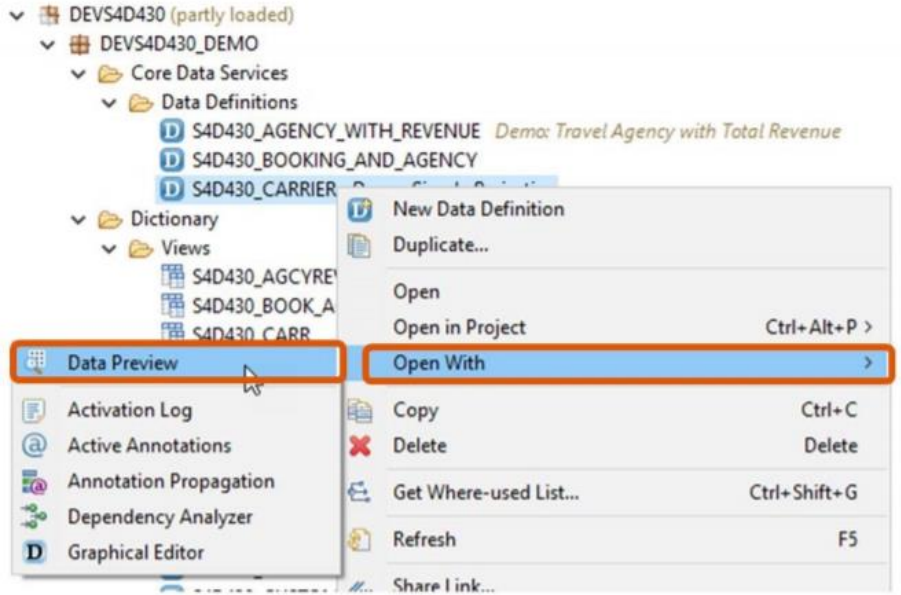
SAP Development Learning

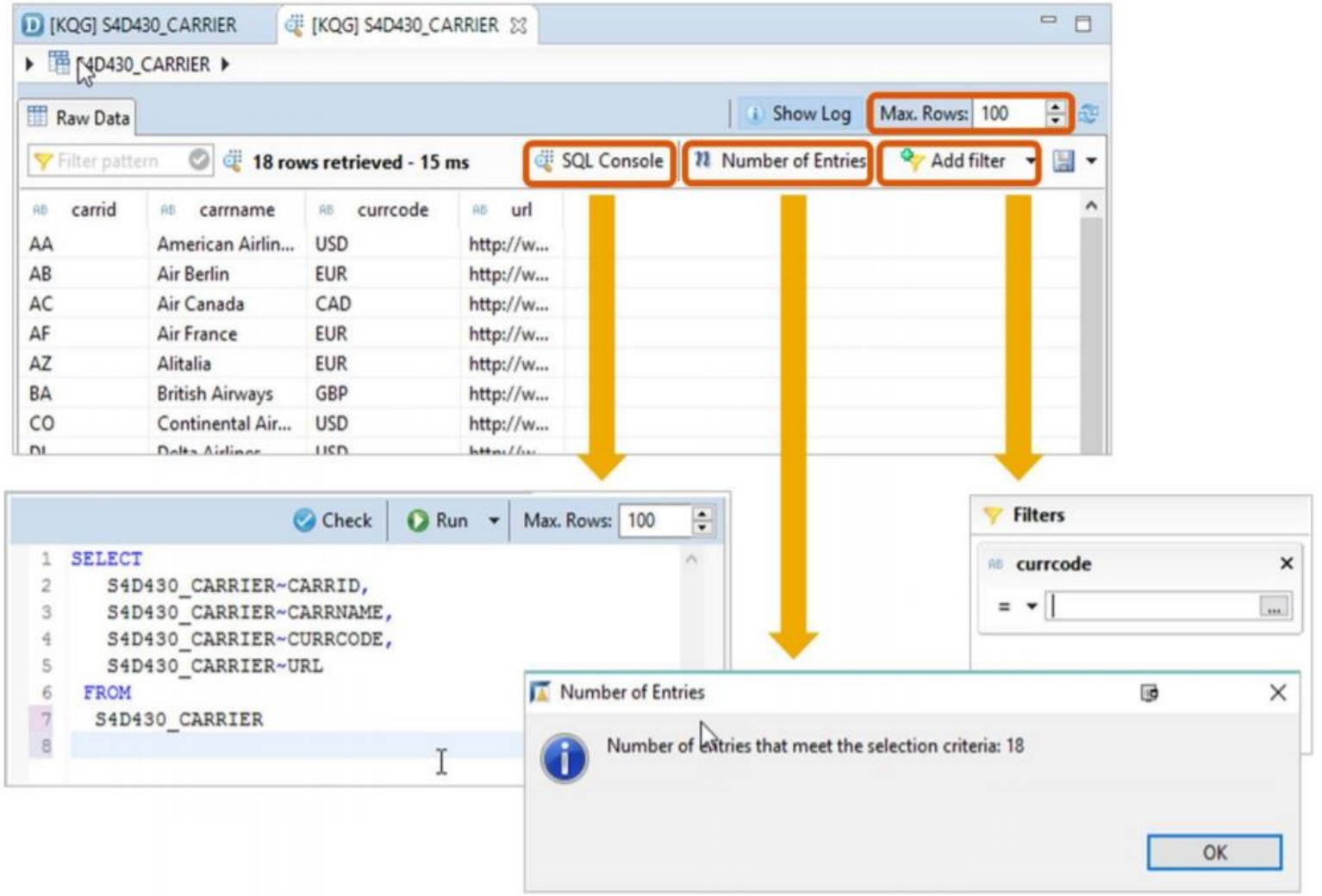




Data Preview for CDS views

SAP Development Learning





SAP Development Learning

Unit 1 – Introduction to Core Data Services

Lesson 3 – Annotations

Classification of SAP Annotations

SAP Development Learning

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

Important ABAP View Annotations

SAP Development Learning

```
@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

SAP Development Learning

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

SAP Development Learning

Element Annotation before the Element

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

Element annotation after the Element

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

Requires „<“ after „@“

Annotation before comma

ABAP Annotations for Translatable Texts

SAP Development Learning

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

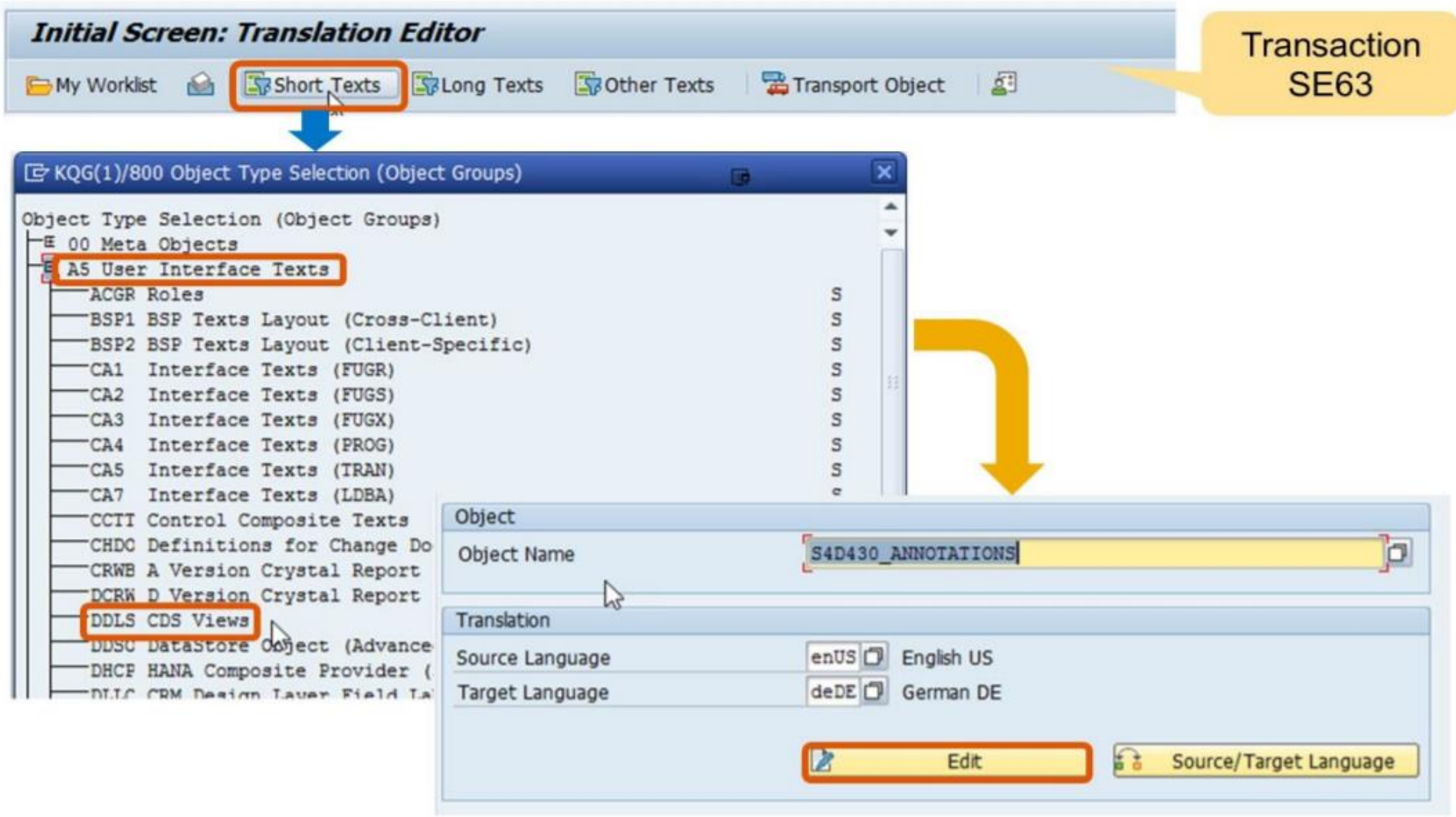
Before or after the element

Note:

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

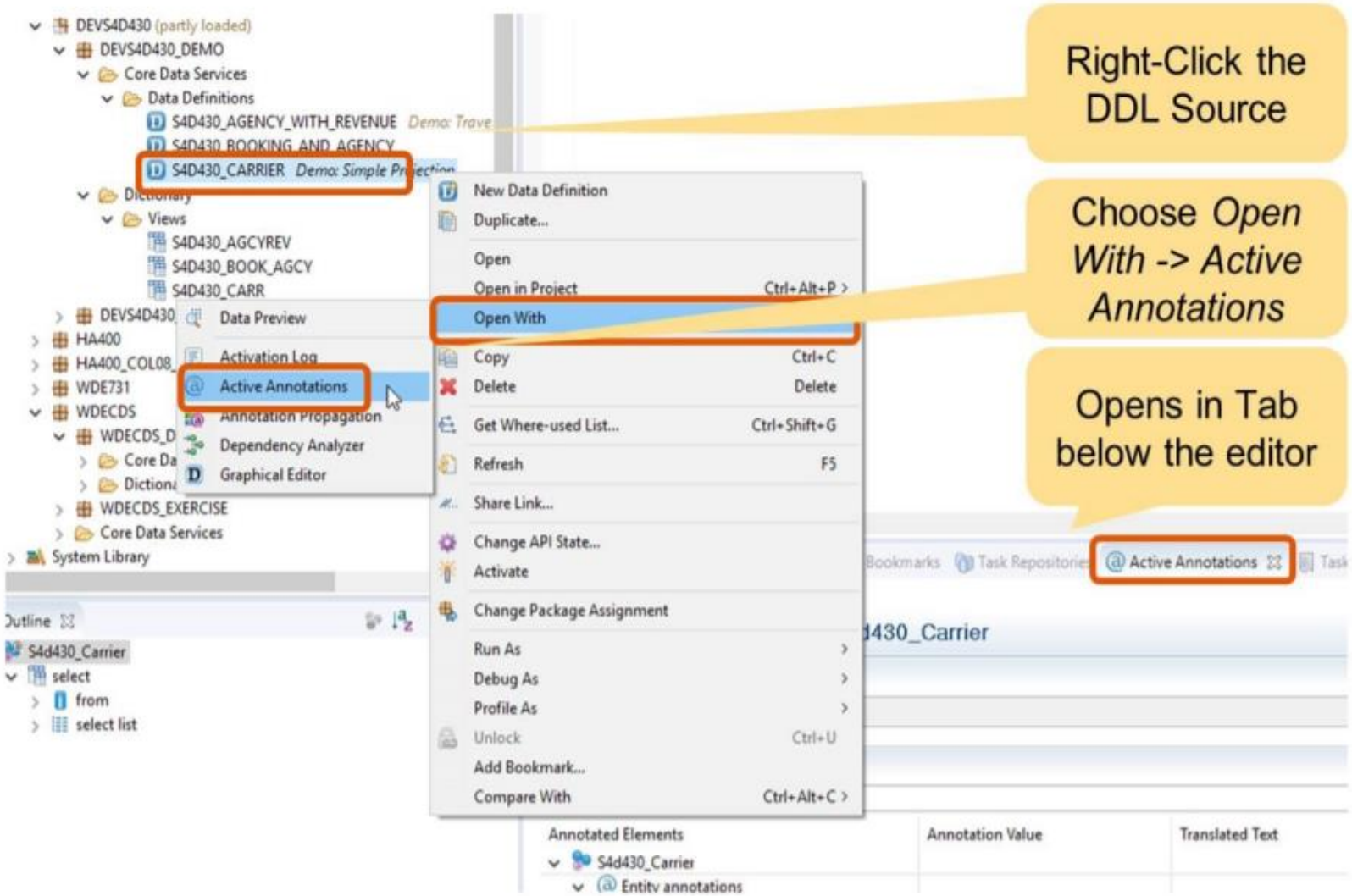
Translation of CDS Views

SAP Development Learning



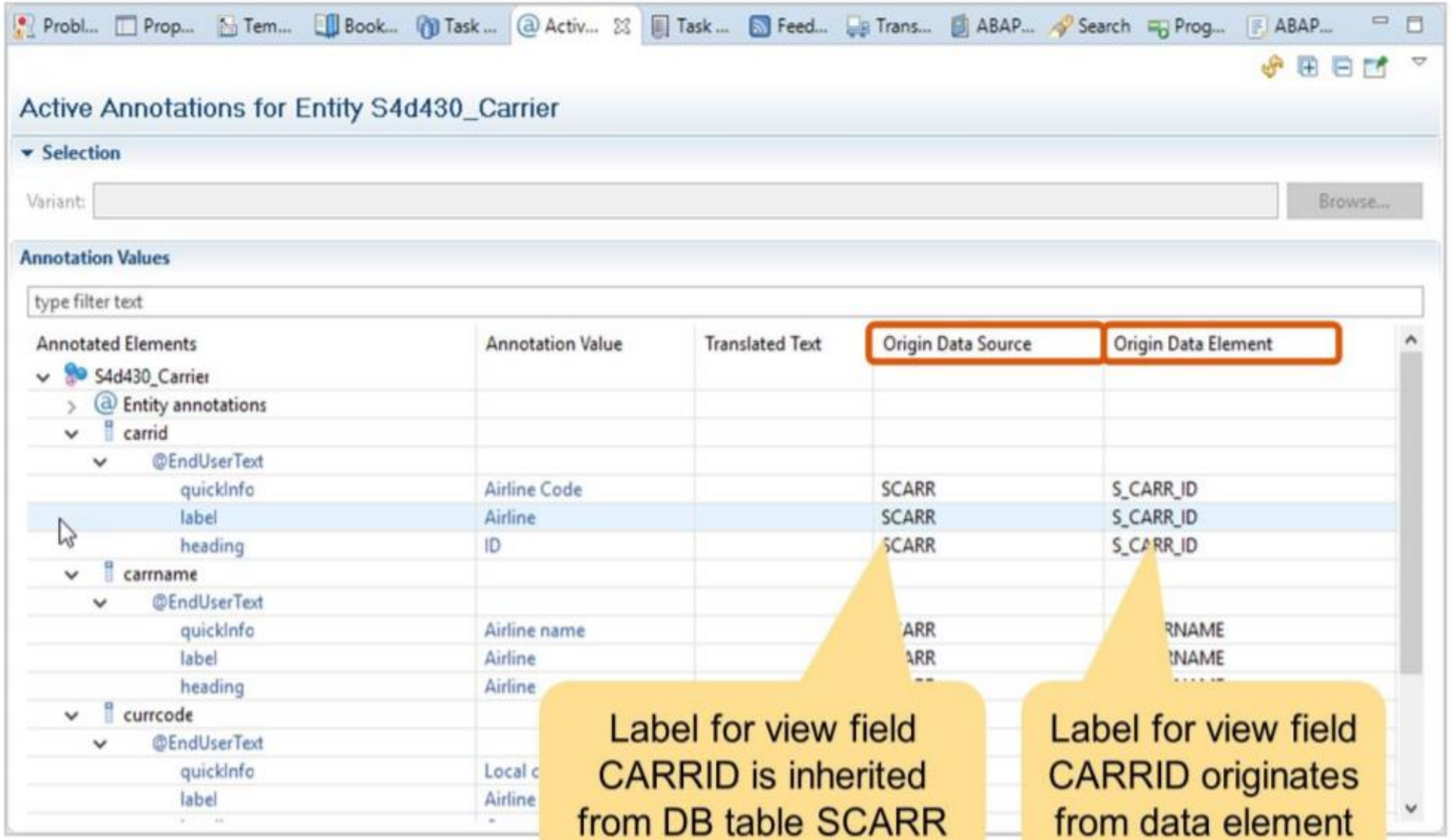
Active Annotations of CDS Views

SAP Development Learning



Active Annotations of CDS Views

SAP Development Learning



SAP Development Learning

Unit 1 – Introduction to Core Data Services

Lesson 4 – SQL Features in ABAP CDS

CASE Expressions

SAP Development Learning

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

SAP Development Learning

```
@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

SAP Development Learning

```
@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,

...

  }
```

Arithmetic Expressions

SAP Development Learning

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

SAP Development Learning

```
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

SAP Development Learning

```
@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	1.98911090000000000E+07	1989-11-09	1989-11-09	9.7402597402597402E-01
19891109	19.891.109	19891109.00	1.98911090000000000E+07	1989-11-09	1989-11-09	9.6623376623376622E-01

Numeric Functions

SAP Development Learning

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 absolut value of *arg*

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

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

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

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

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

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

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

String Processing Functions

SAP Development Learning

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

SAP Development Learning

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'
```

Currency and Unit Conversions

SAP Development Learning

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

SAP Development Learning

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

Casting of literal 'USD' into expected type

Behaviour in Case of error

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

Casting of literal 'MI' into expected type

Parameter assignment with operator „=>“

Calculation with Dates

SAP Development Learning

- **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

SAP Development Learning

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

           dats_days_between(order_date, fldate) as days_ahead,

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

         }
```

„Subtract“ two fields of type DATE

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

Exercise 01: Enhance the CDS view with Expressions

SAP Development Learning

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

SAP Development Learning

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

SAP Development Learning

```
@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

SAP Development Learning

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

Field list consists not only of aggregate expressions

GROUP BY needed for all fields in the field list

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

Field list contains expression or function

Arguments of the expression or function listed individually

Exercise 02: CDS view with Aggregate Expressions

SAP Development Learning

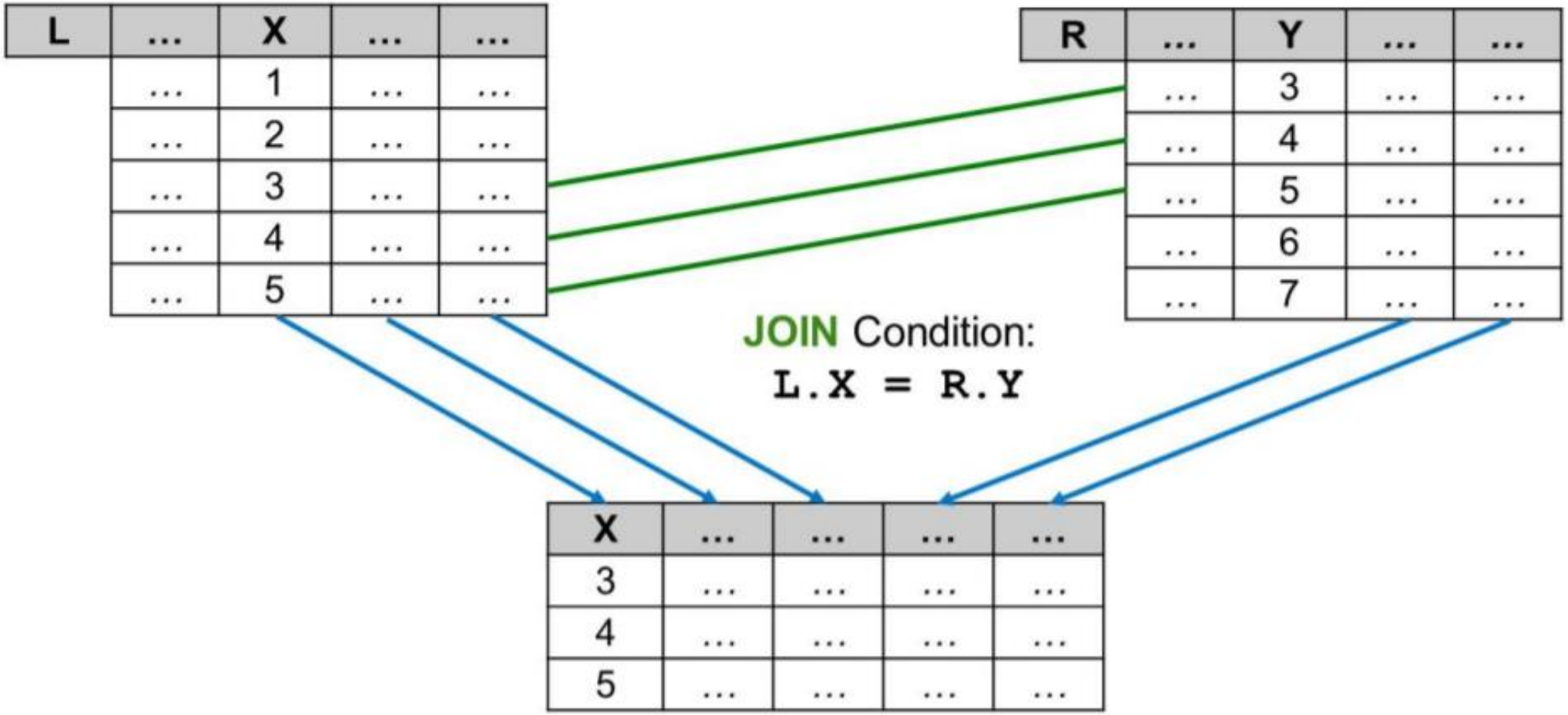
In this exercise you will:

- 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

SAP Development Learning



Types of Outer Joins

SAP Development Learning

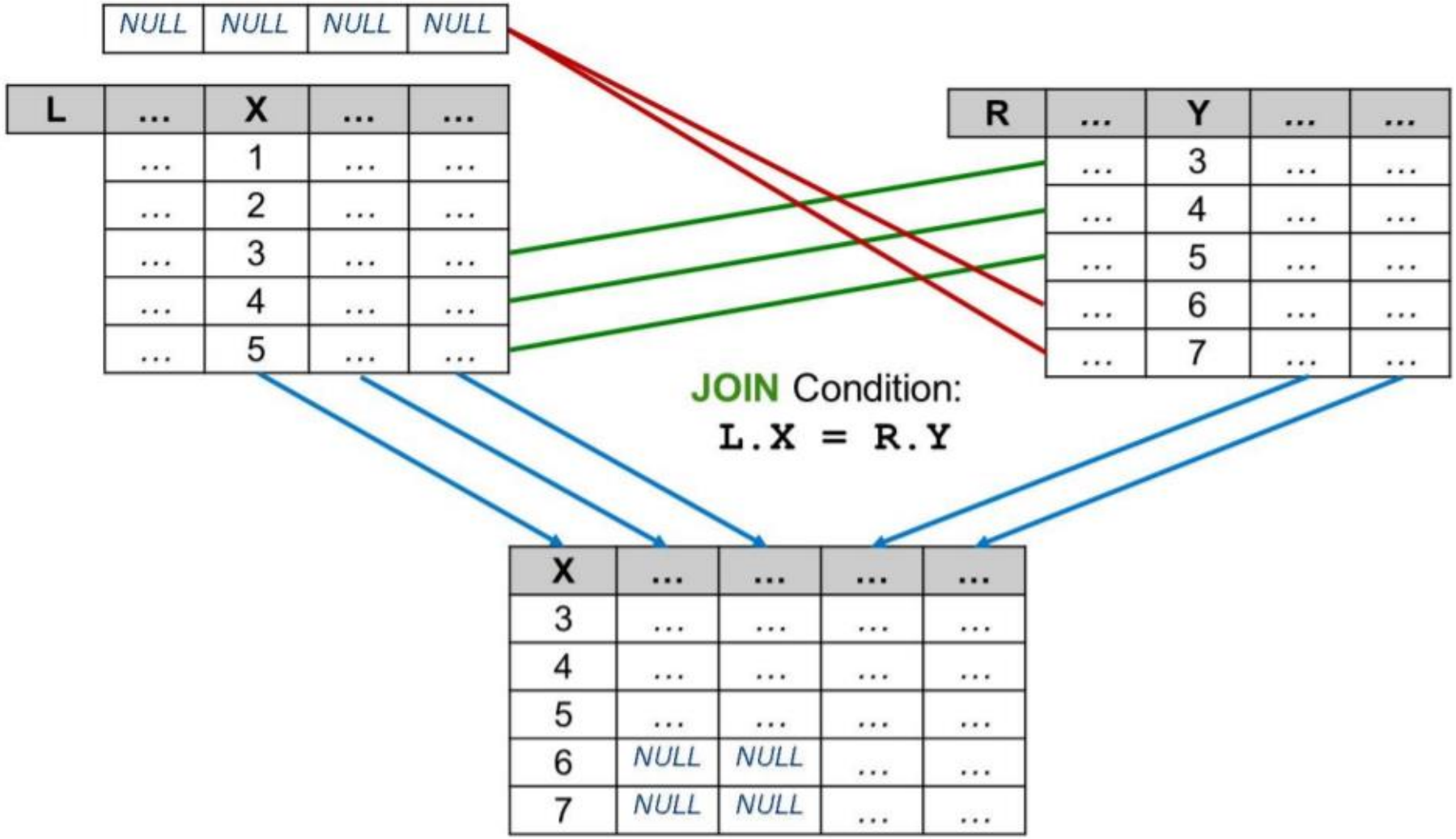
- **Left Outer Join**
  - Special treatment for left table
  - Result set may contain entries of **left** table without matching entries in right table
  - Supported in classical Open SQL
- **Right Outer Join**
  - Special treatment for right table
  - Result set may contain entries of **right** table 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

SAP Development Learning

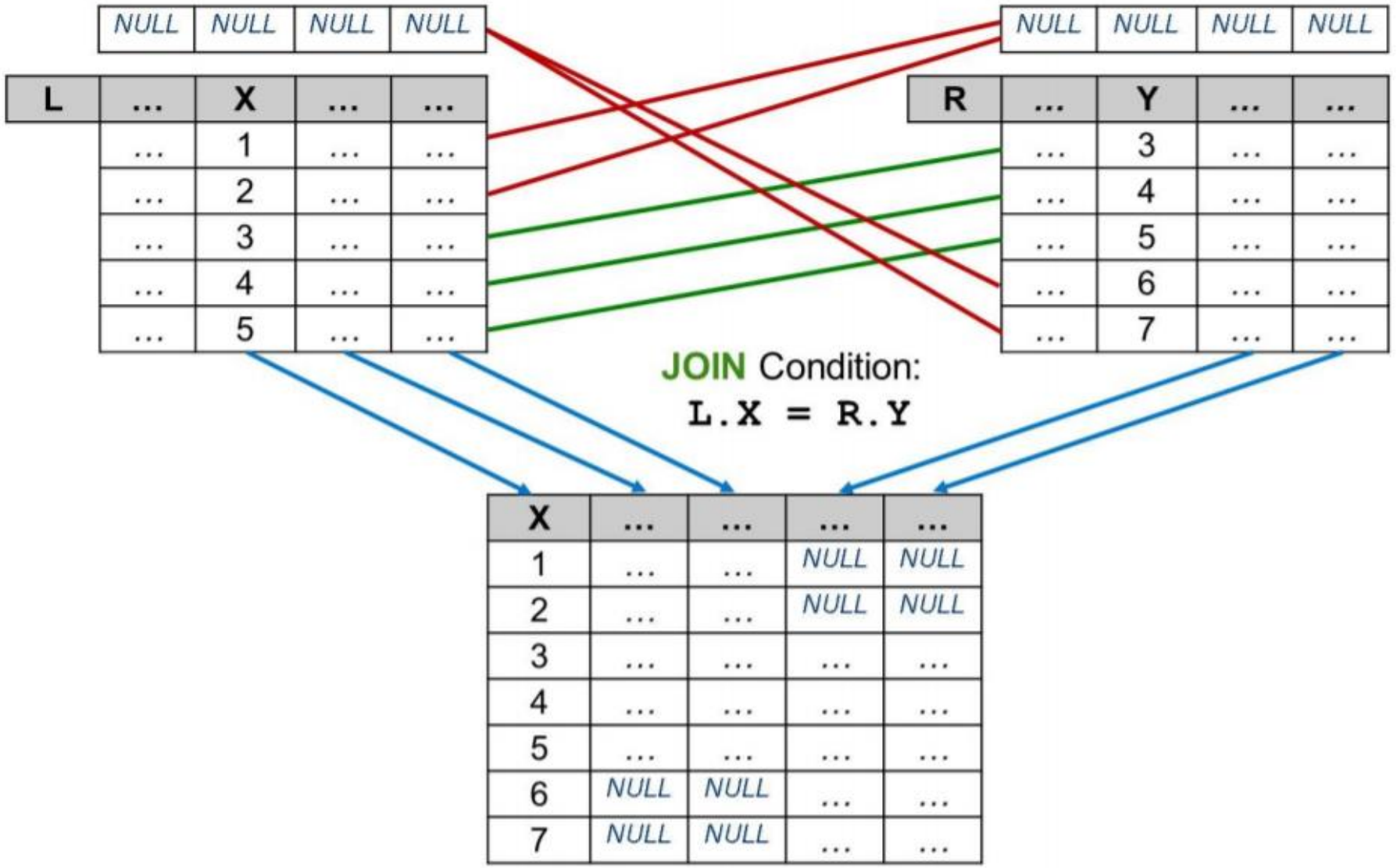
Right Outer Join

SAP Development Learning



Full Outer Join (not supported in ABAP CDS)

SAP Development Learning



Right Outer Join - Example

SAP Development Learning

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

Raw Data

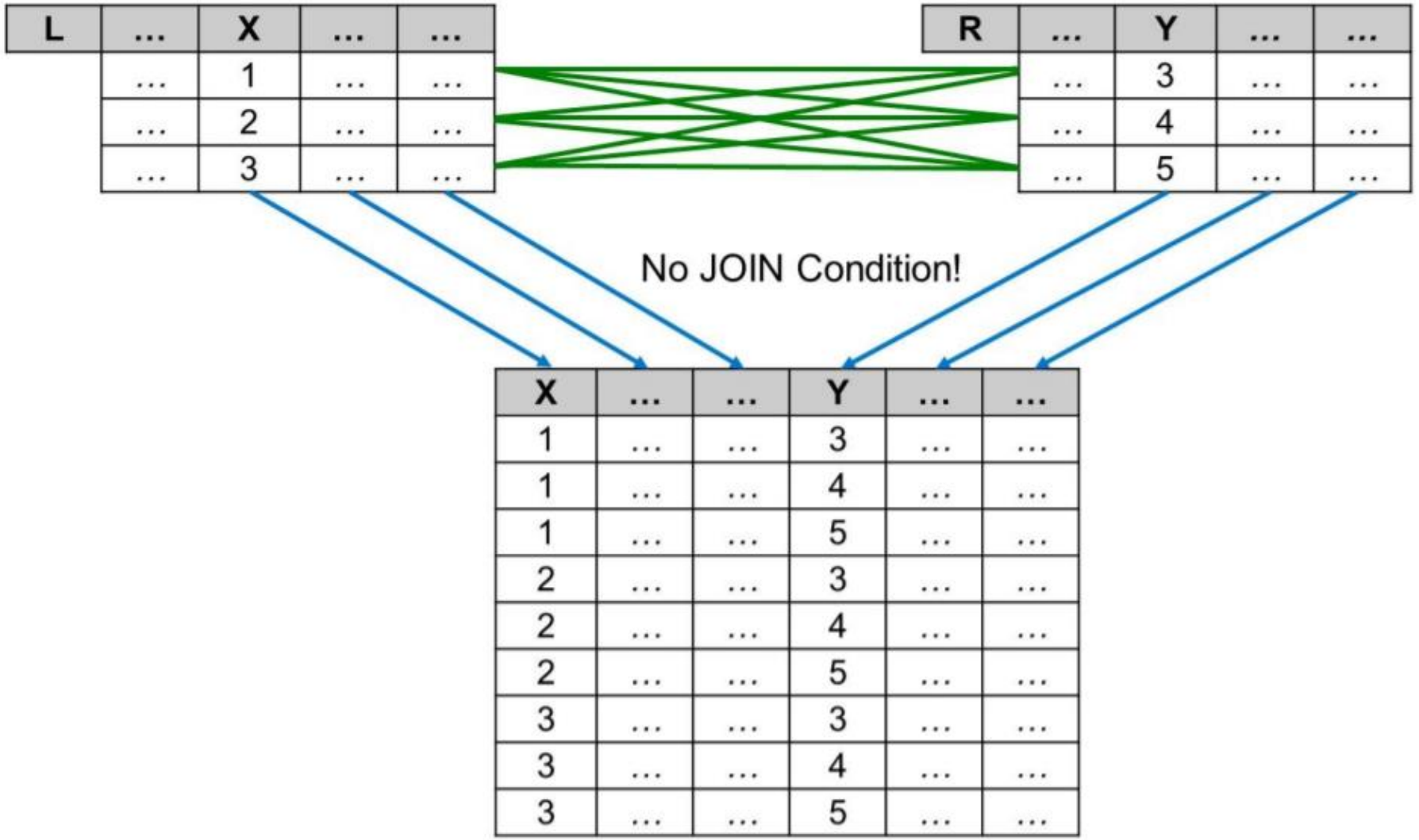
Filter pattern | 36 rows retrieved - 62 ms

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

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

Cross Join (as of Release 7.51)

SAP Development Learning



Cross Join - Example

SAP Development Learning

```
@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

SAP Development Learning

Explicit (with parantheses) or Implicit (no parantheses)

- 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
     }
```

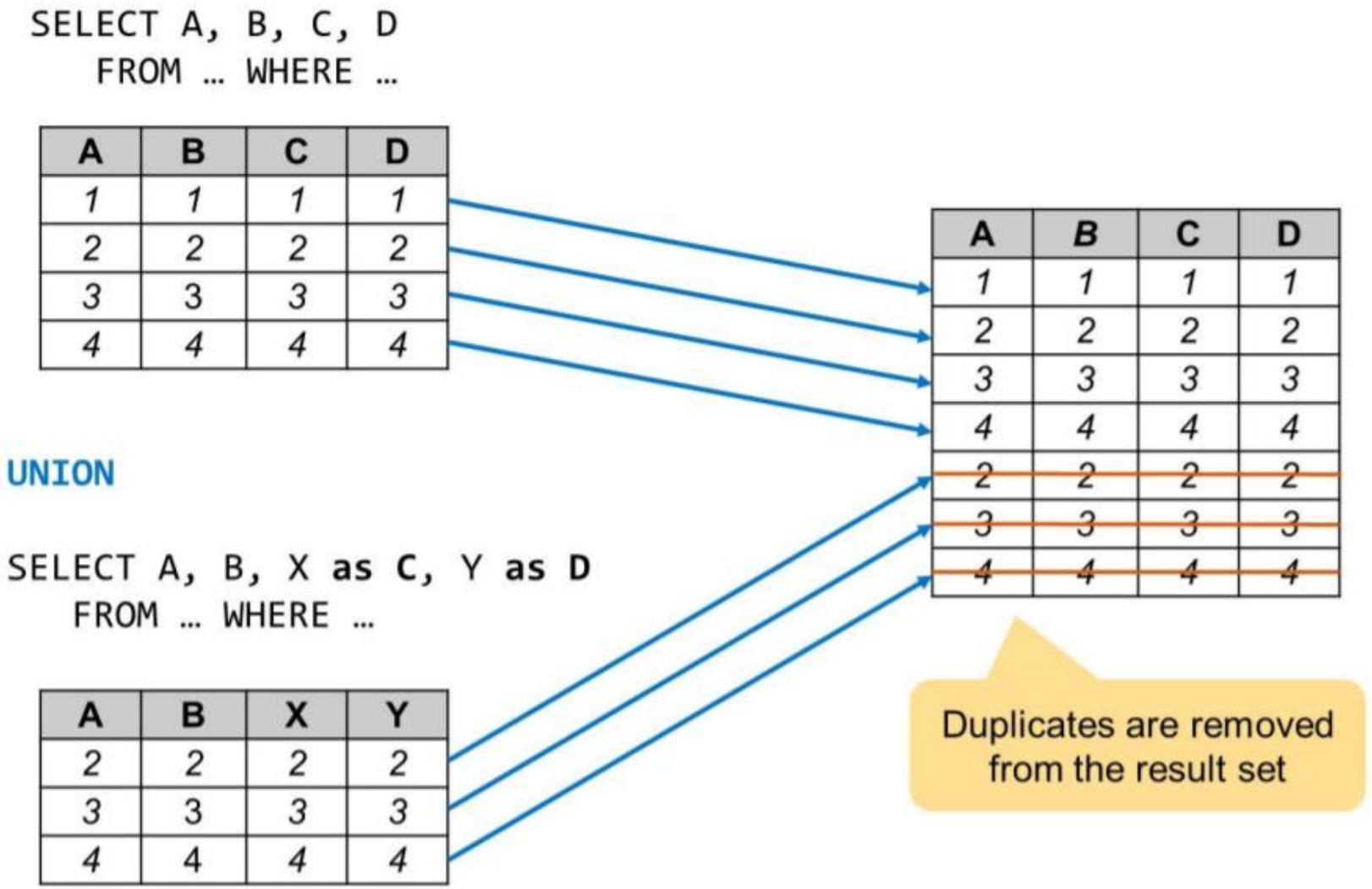
Join in parantheses is evaluated first

```
@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 parantheses: Sequence of ON-conditions rules

UNION

SAP Development Learning



UNION: Prerequisites and Result Set

SAP Development Learning

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

SAP Development Learning

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

Key definition in second select is ignored

Alias *id* is needed to have identical element names

Additional blanks needed to have compatible types

UNION ALL

SAP Development Learning

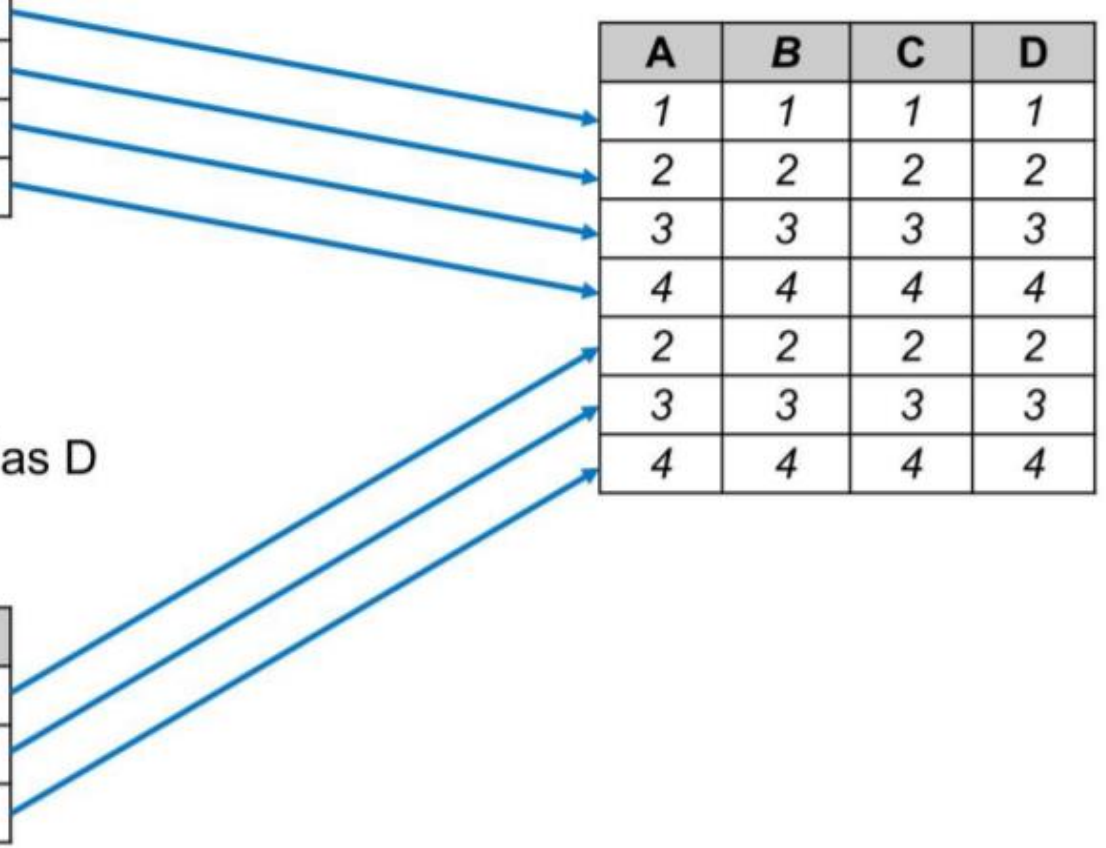
SELECT A, B, C, D
FROM … WHERE …

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

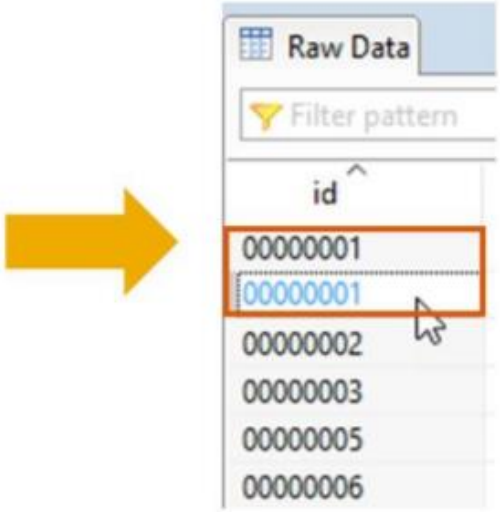


A	B	C	D
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
2	2	2	2
3	3	3	3
4	4	4	4

Example: UNION vs UNION ALL

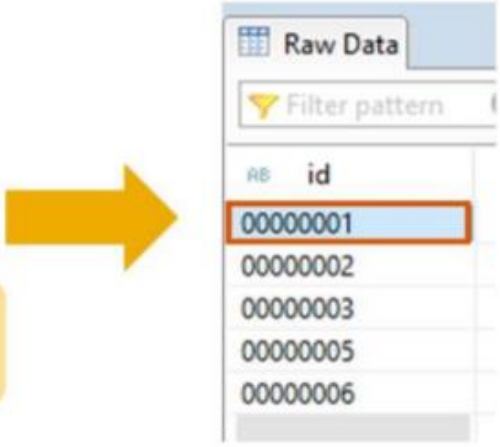
SAP Development Learning

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




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

Same definition without ALL



Exercise 03: CDS view with Left Outer Join

SAP Development Learning

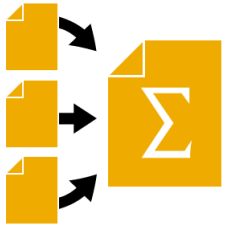
In this exercise you will:

- Use LEFT OUTER Join to display the 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

SAP Development Learning



You should now be able to:

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

SAP Development Learning

Unit 2 – Advanced Concepts

Lesson 1 – Input Parameter

Need for Input Parameters

SAP Development Learning

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

SAP Development Learning

```
@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

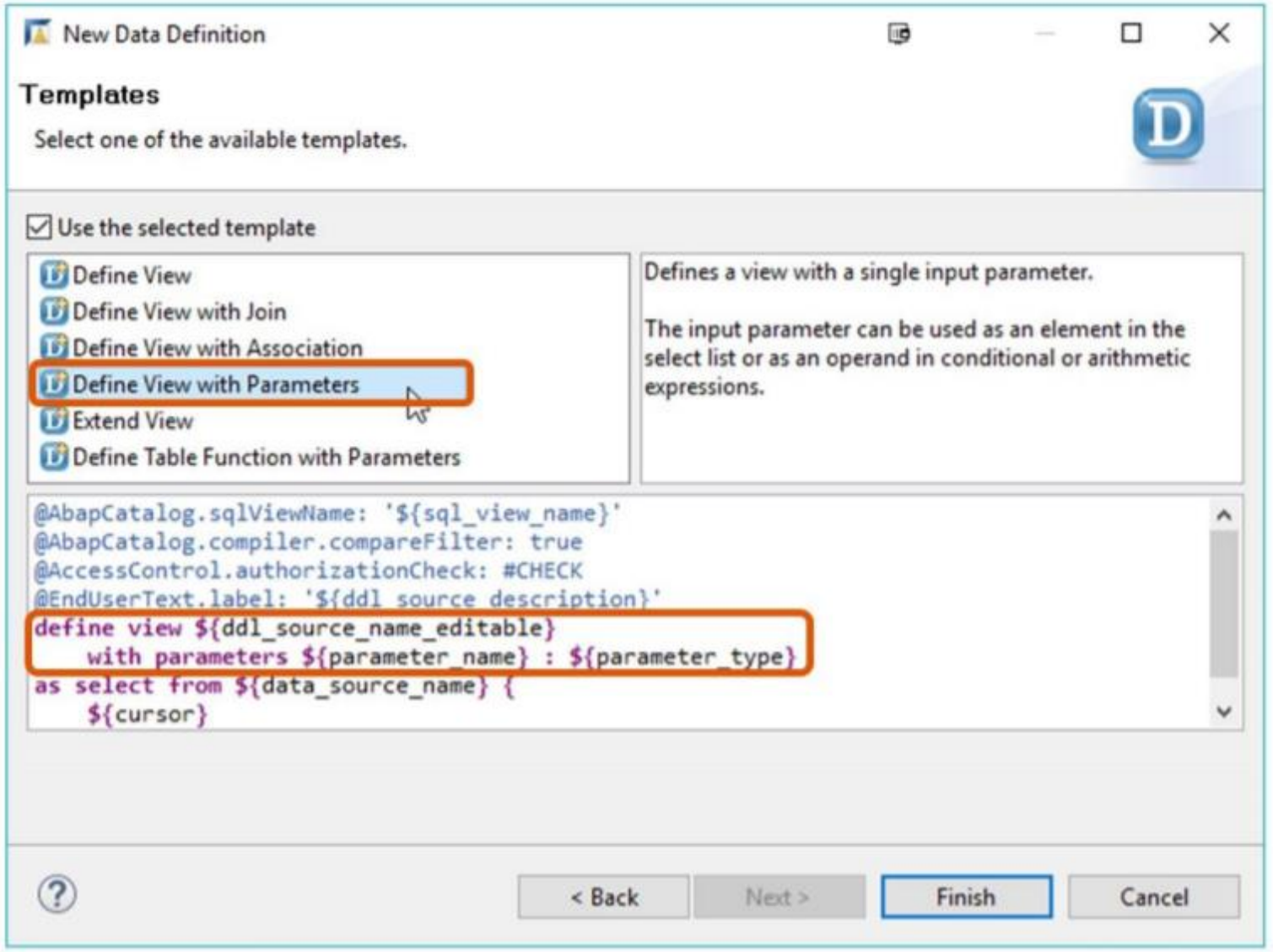
Predefined Dictionary
type or data element

EndUserText-Annotations
before the parameter ..

… or after the parameter

Template for View with Input Parameters

SAP Development Learning



Access Input Parameters

SAP Development Learning

- **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

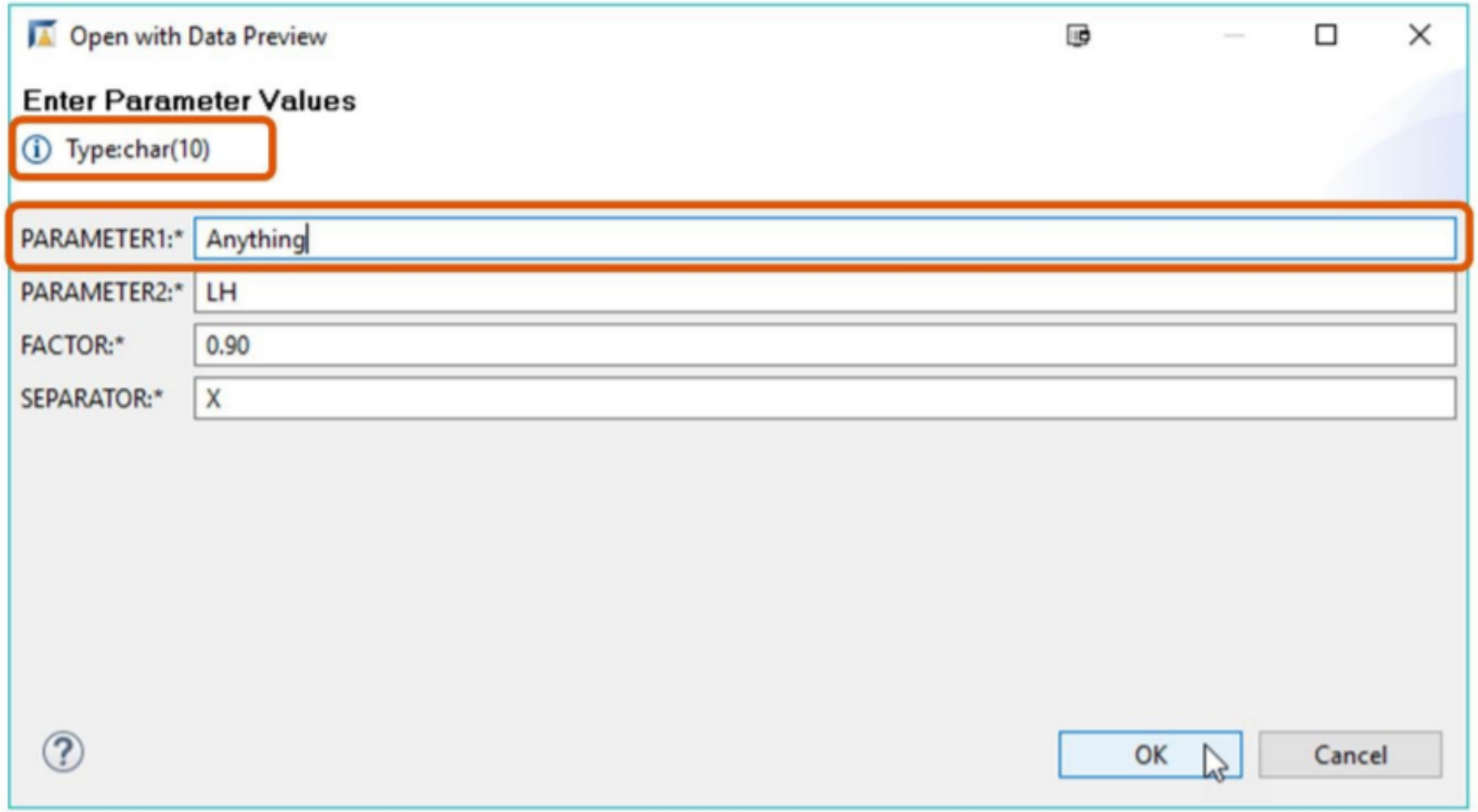
```
@AbapCatalog.sqlViewName: 'S4D430_PARAM'
define view S4d430_Parameter
  with parameters
            ...
            parameter1: abap.char(10),
            parameter2: s_carr_id

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

          :parameter2            as param2,
}
```

Prefixed with „$parameters.“ (recommended)

Prefixed with „:“ (not recommended)

Input Parameters in Data Preview

SAP Development Learning



Forwarding Input Parameters to another CDS

SAP Development Learning

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

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

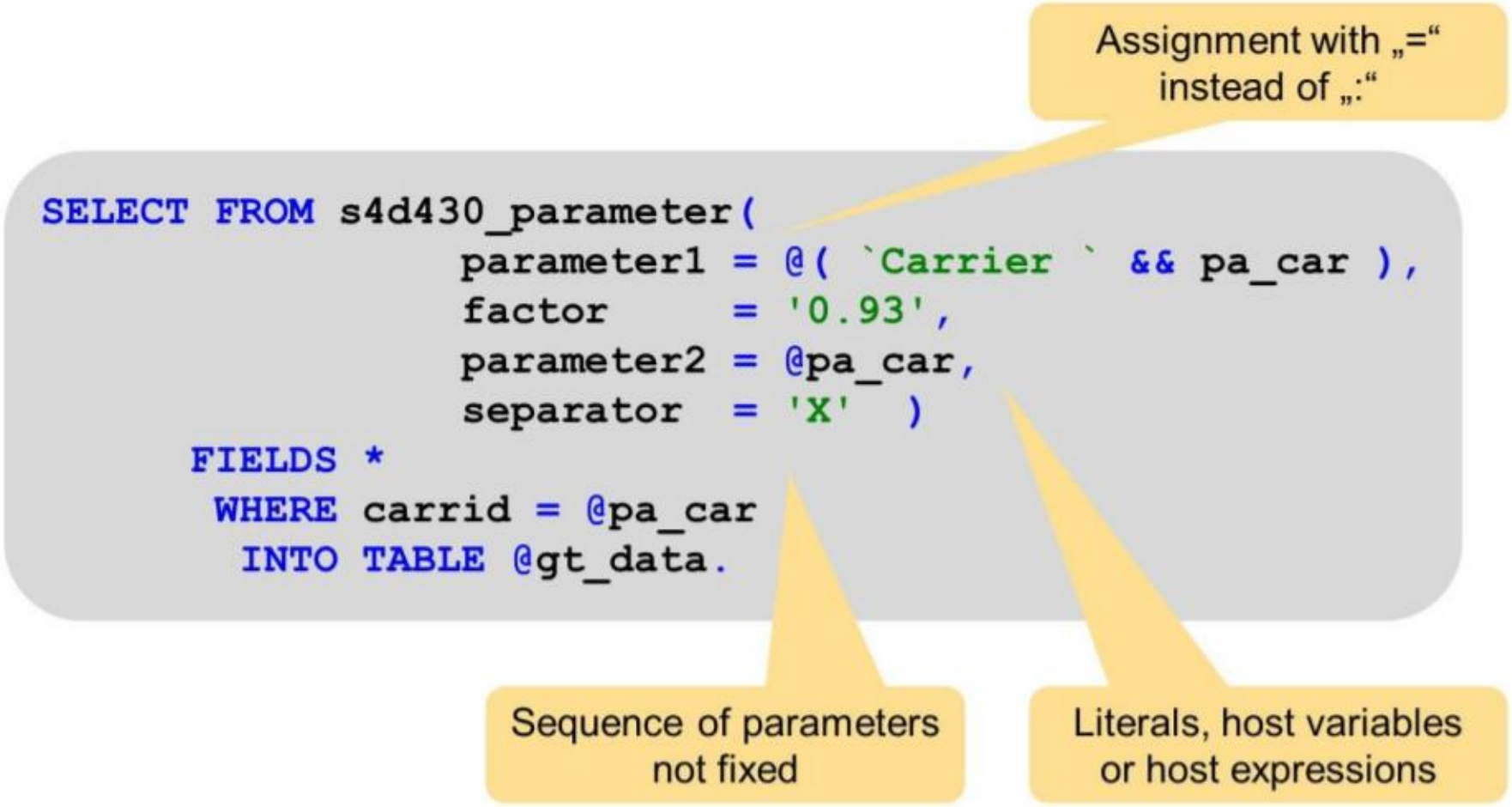
CDS View with Parameters as data source

Parameter passing in parantheses „()“

This parameter is „forwarded“

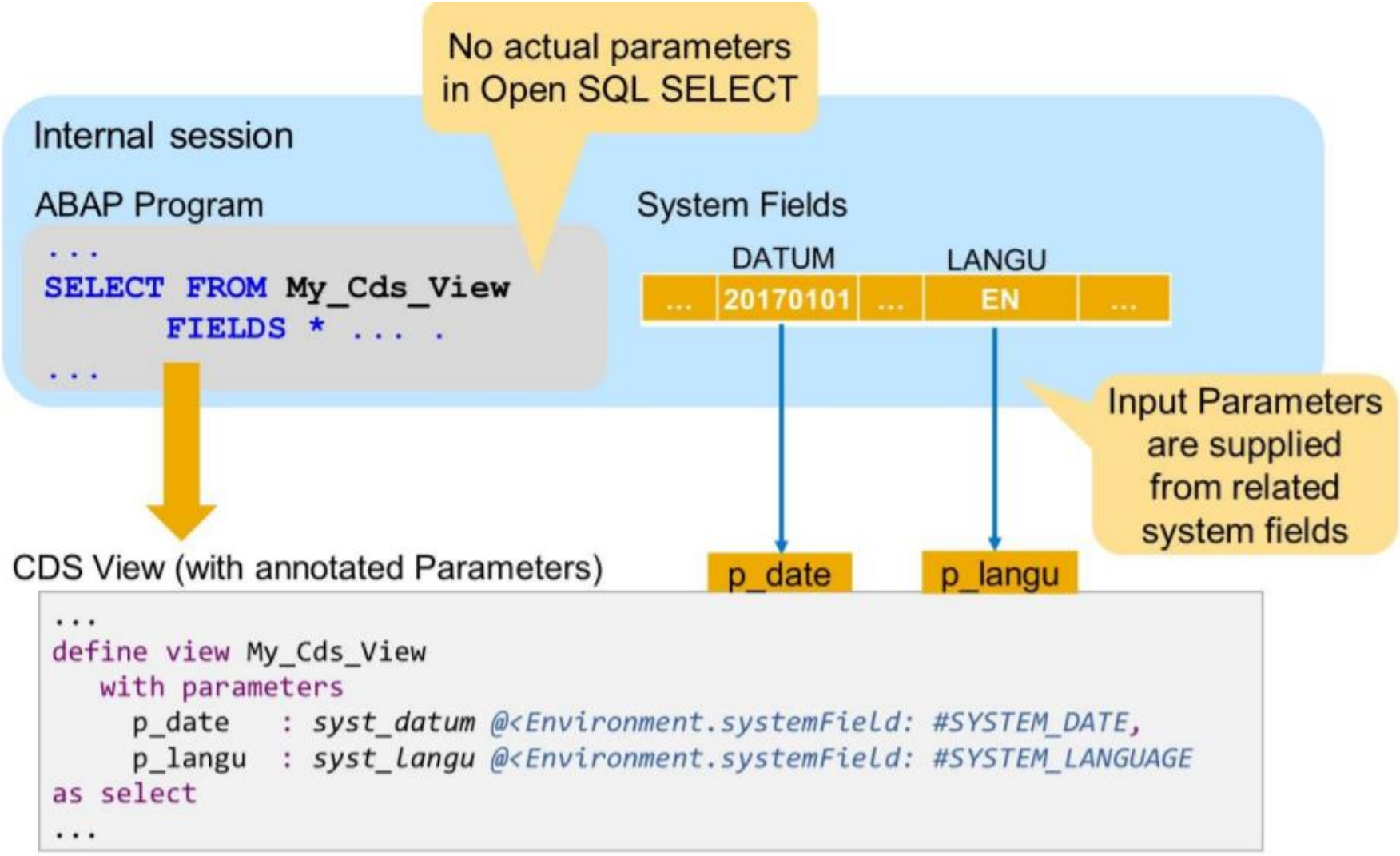
Consume CDS View With Parameters in Open SQL

SAP Development Learning



Optional Parameters for CDS Views

SAP Development Learning



Rules for Annotation Environment.systemField

SAP Development Learning

- 5 different values

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

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

Session Variables

SAP Development Learning

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

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

- 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

SAP Development Learning

```
@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

SAP Development Learning

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

SAP Development Learning

Unit 2 – Advanced Concepts

Lesson 2 – Association

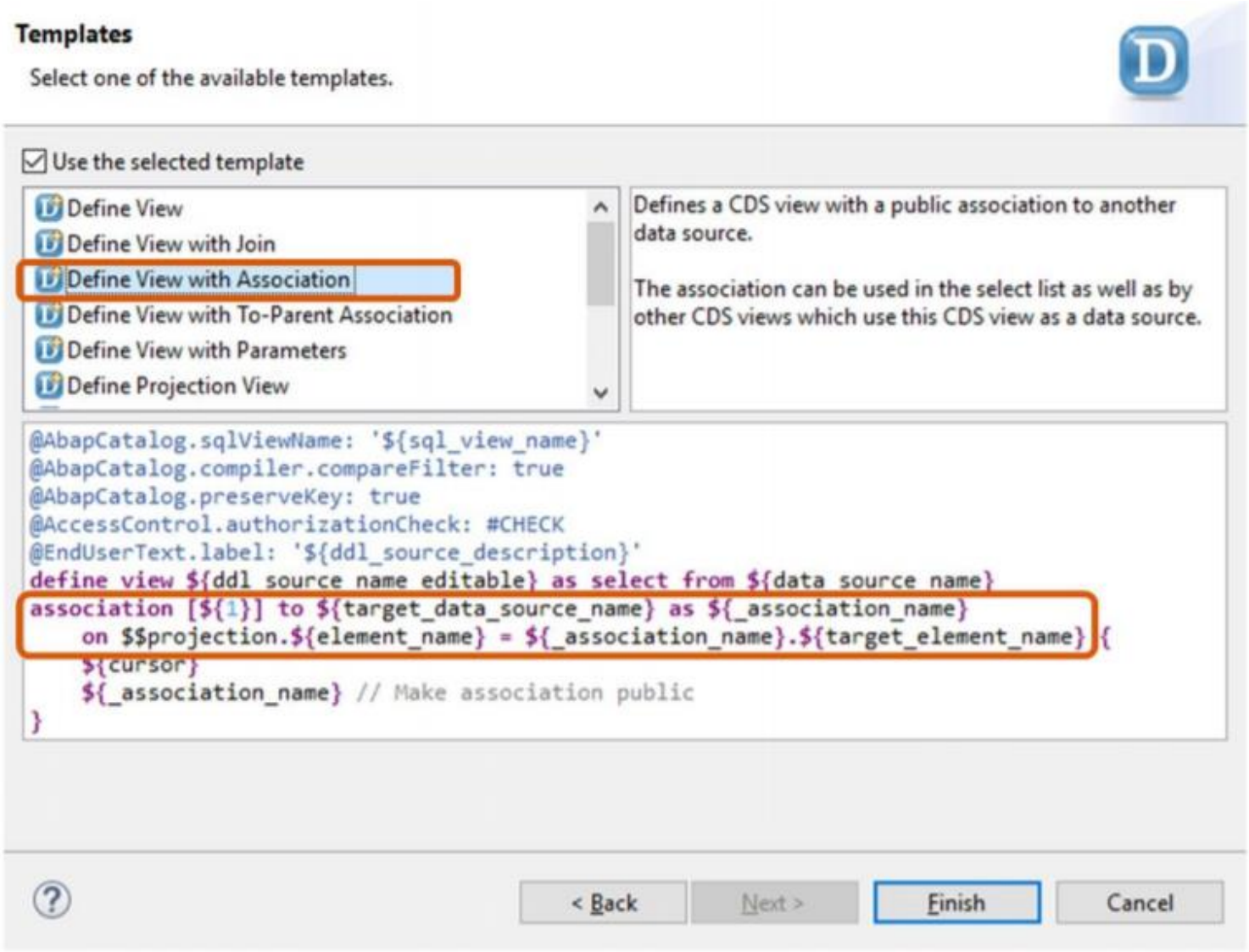
What Are Associations?

SAP Development Learning

- **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

SAP Development Learning



Association Instead of Join

SAP Development Learning

View Definition with Association

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

Association target

Name of data source mandatory for all fields from association target

View Definition with Join

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

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

Technical Realization of Associations

SAP Development Learning

View Definition with Association

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

Corresponding SQL Create Statement

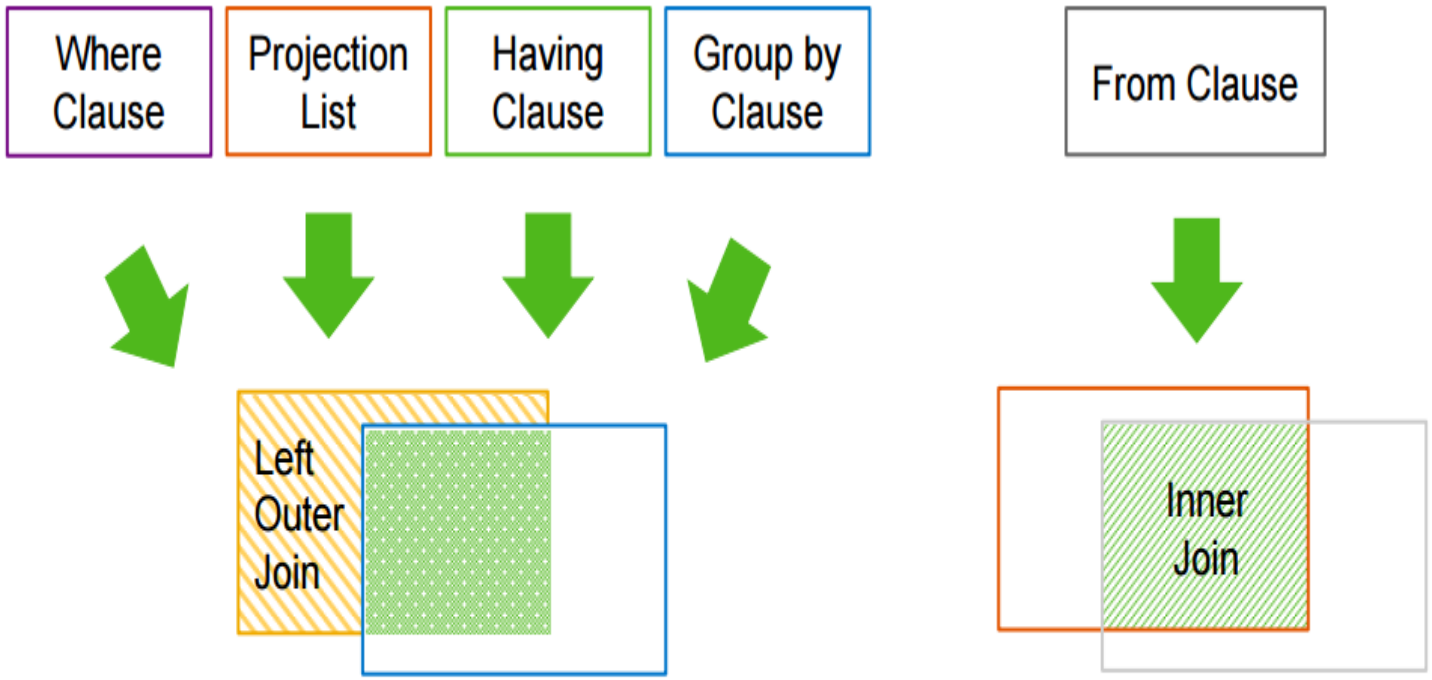
```
CREATE VIEW "S4D430_ASSO1" AS SELECT
  "SPFLI"."MANDT" AS "MANDT",
  "SPFLI"."CARRID",
  "SPFLI"."CONNID",
  "SPFLI"."CITYFROM",
  "SPFLI"."CITYTO",
  "=A0"."CARRNAME"
FROM "SPFLI" "SPFLI" LEFT OUTER JOIN "SCARR" "=A0" ON (
  "SPFLI"."MANDT" = "=A0"."MANDT" AND
  "SPFLI"."CARRID" = "=A0"."CARRID"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

- Default join type can be changed to Inner Join

  scarr[inner].carrname

CDS Associations – Translation into Joins

SAP Development Learning



Used (!) associations are implicitly translated into SQL joins

Reuse of generated joins when semantically identical

Additional Semantics in Association

SAP Development Learning

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

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

Cardinality (exactly one carrier related to a connection)

Name of the association

Replaced with Alias in SQL Create Statement

Extract From the SQL Create Statement

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

On-Condition with $Projection

SAP Development Learning

```
@AbapCatalog.sqlViewName: 'S4D430_ASSO3'

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

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

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

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

Some Rules Regarding Cardinality

SAP Development Learning

- **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 [*]

SAP Development Learning

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

Access to association with cardinality [*] increases result set

Result without field fldate

Raw Data

Filter pattern 26 rc

carrid	connid
AA	0017

… and with field fldate

Raw Data

Filter pattern 100 rows retrieved

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

Exposed Association

SAP Development Learning

```
@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

SAP Development Learning

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

No warning if association with cardinality [*] is exposed

Exposed Associations are not (yet) joined on database level

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

Use Exposed Associations in a View Definition

SAP Development Learning

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

          c._Carrier.carrname as carrier_name,

          _Flights
    }
```

Read from CDS View that exposes Associations

Read field carrname from target of association _carrier

Propagate association _flights (Do not use it directly)

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

Join of SCARR, but no join of SFLIGHT

Propagation of Associations

SAP Development Learning

```
@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
```

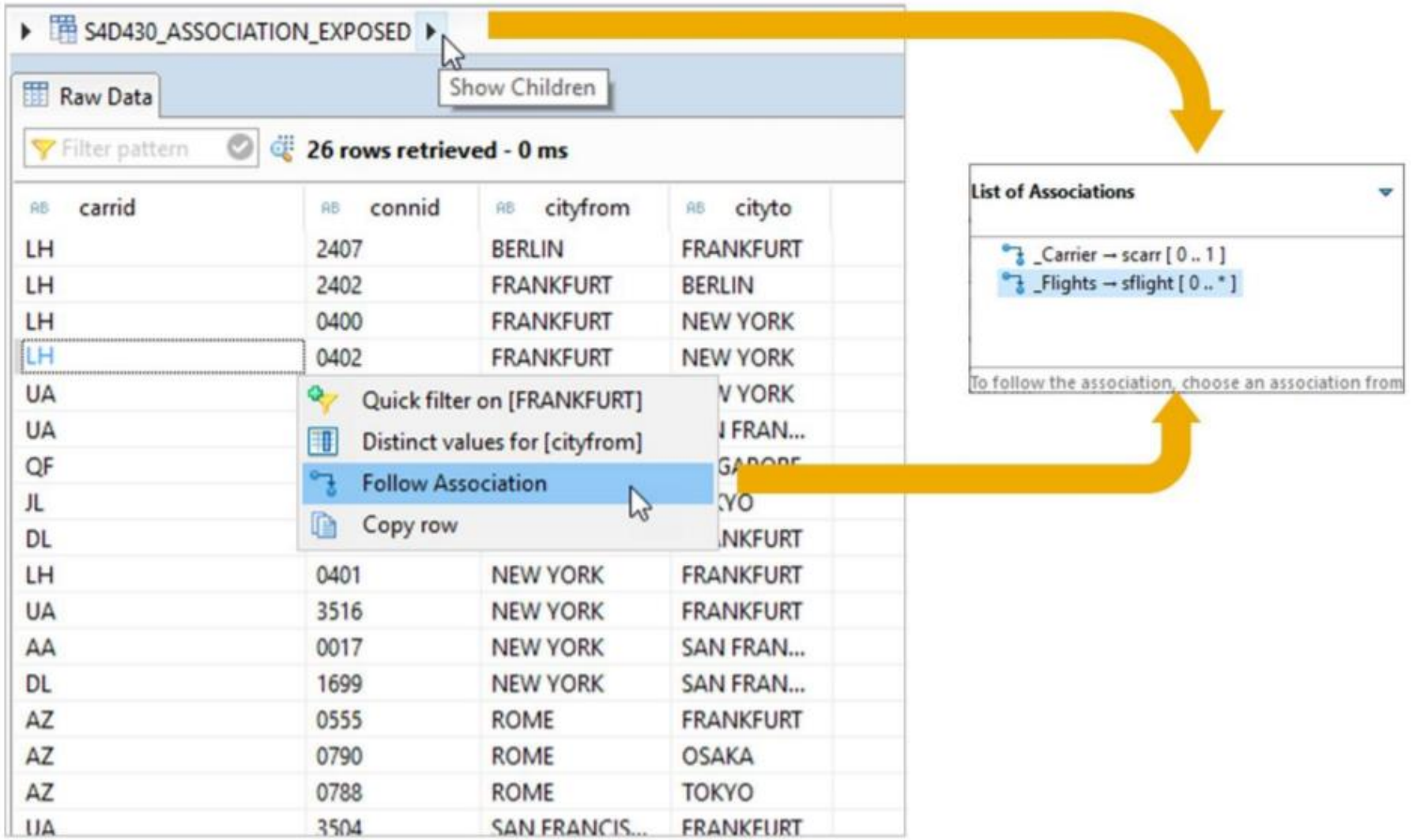
Read field seatsocc from target of association _flights

```
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

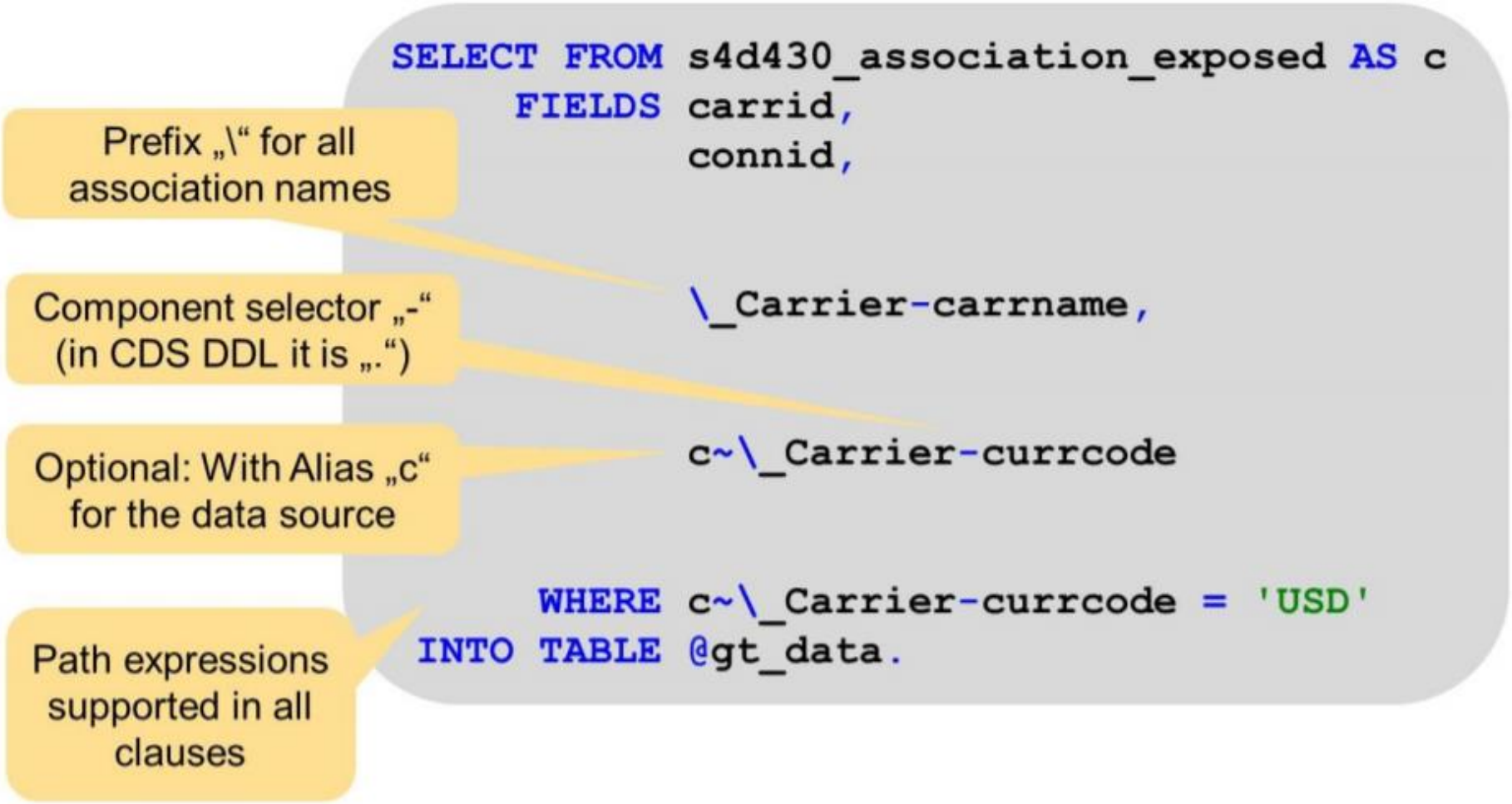
Exposed Associations in Data Preview

SAP Development Learning



Use Exposed Associations in Open SQL

SAP Development Learning



Filter Condition

SAP Development Learning

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

SAP Development Learning

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

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

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

Do not restrict to one language in the ON condition

But use a filter condition in the Path Expression

Rules for Filter Conditions

SAP Development Learning

- **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

SAP Development Learning

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

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

„Foreign key relation has cardinality 0..* “

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

Annotation AbapCatalog.compiler.compareFilter

SAP Development Learning

```
...
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

SAP Development Learning

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

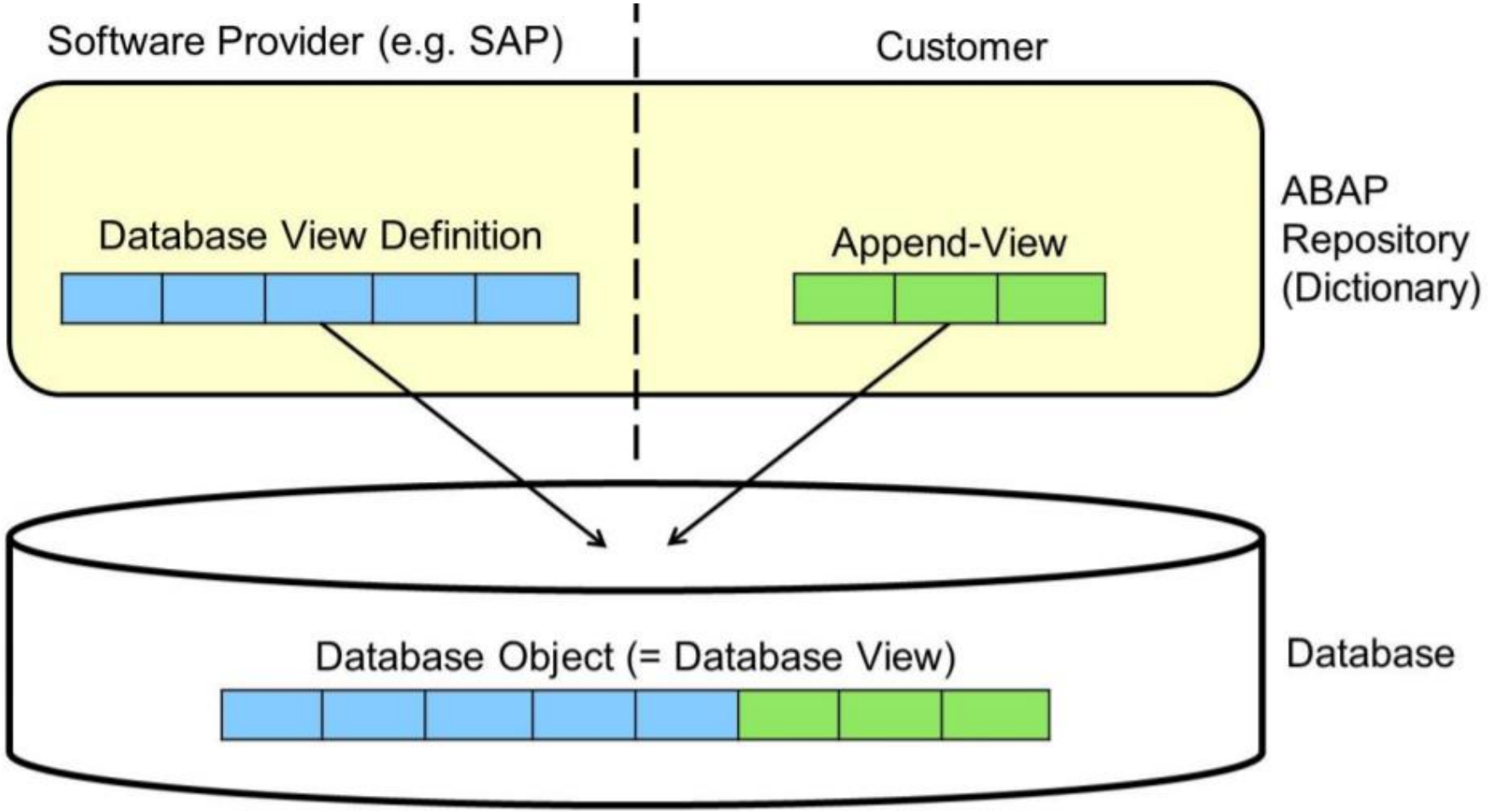
SAP Development Learning

Unit 2 – Advanced Concepts

Lesson 3 – CDS View Extensions

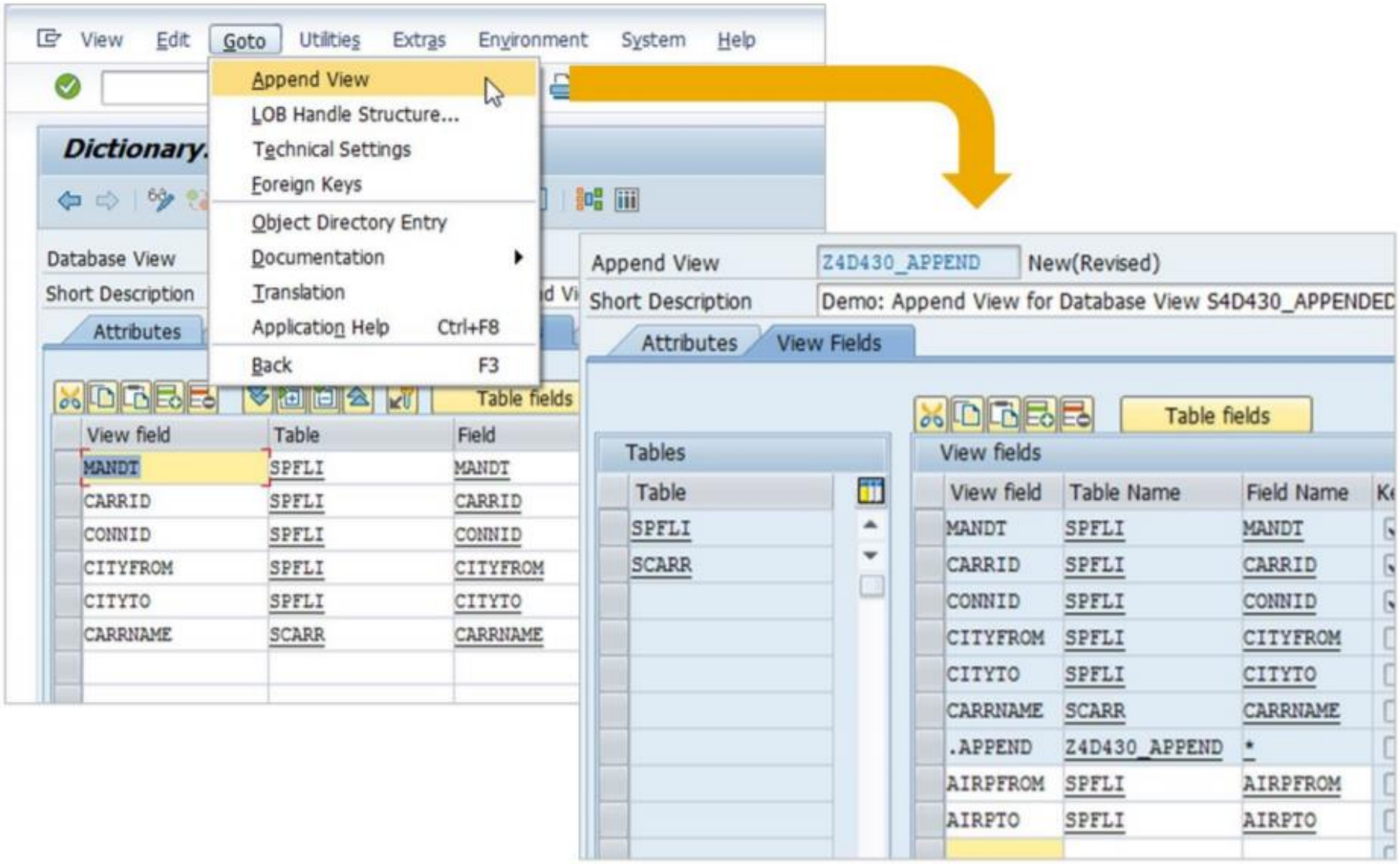
Append Views for Classical Database Views

SAP Development Learning



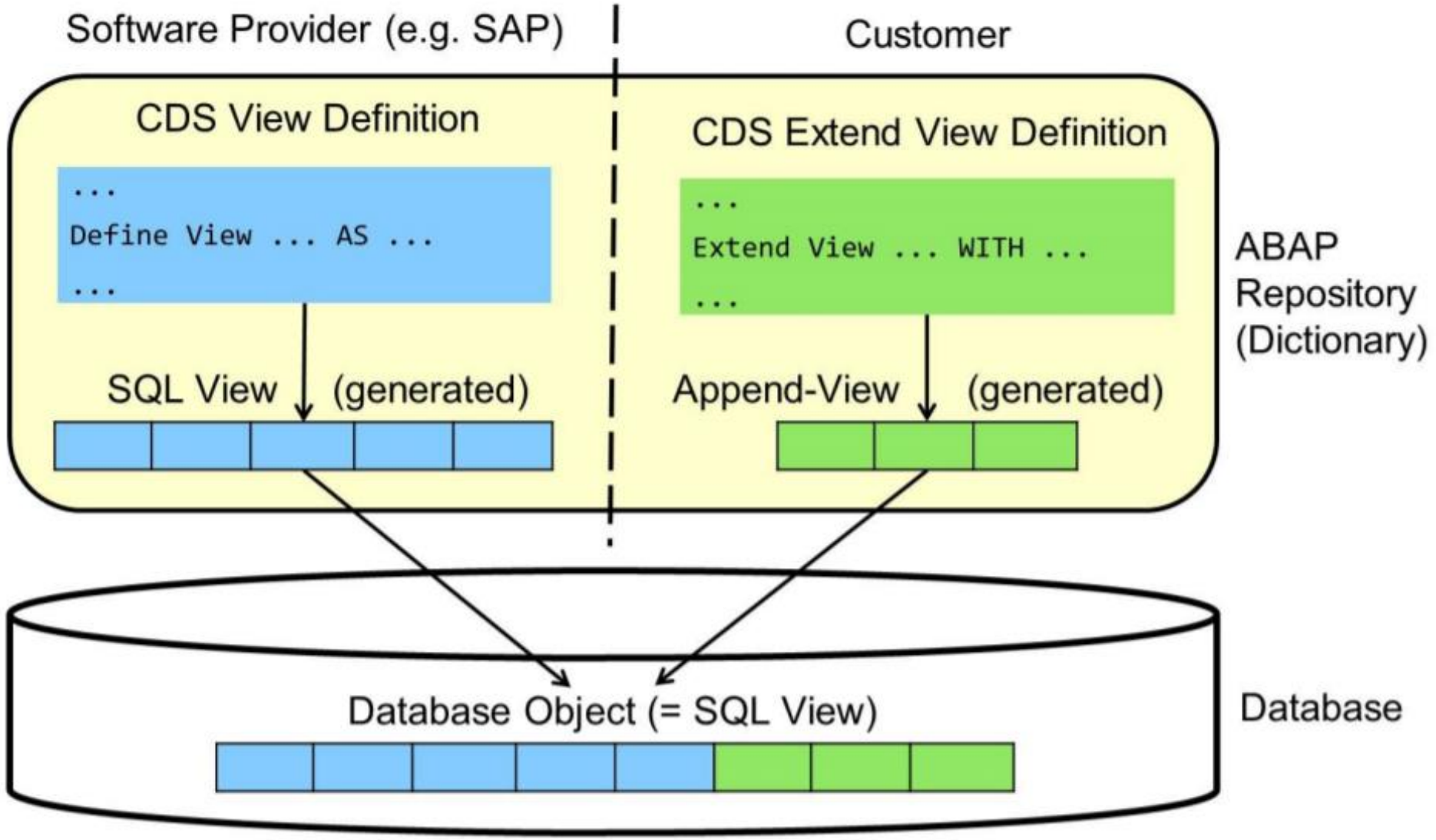
Creating a Classical Append View

SAP Development Learning



CDS View Extensions

SAP Development Learning



CDS View and its Extension

SAP Development Learning

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

SAP Development Learning

- **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

CDS View Extension with Association

SAP Development Learning

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

SAP Development Learning

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
```

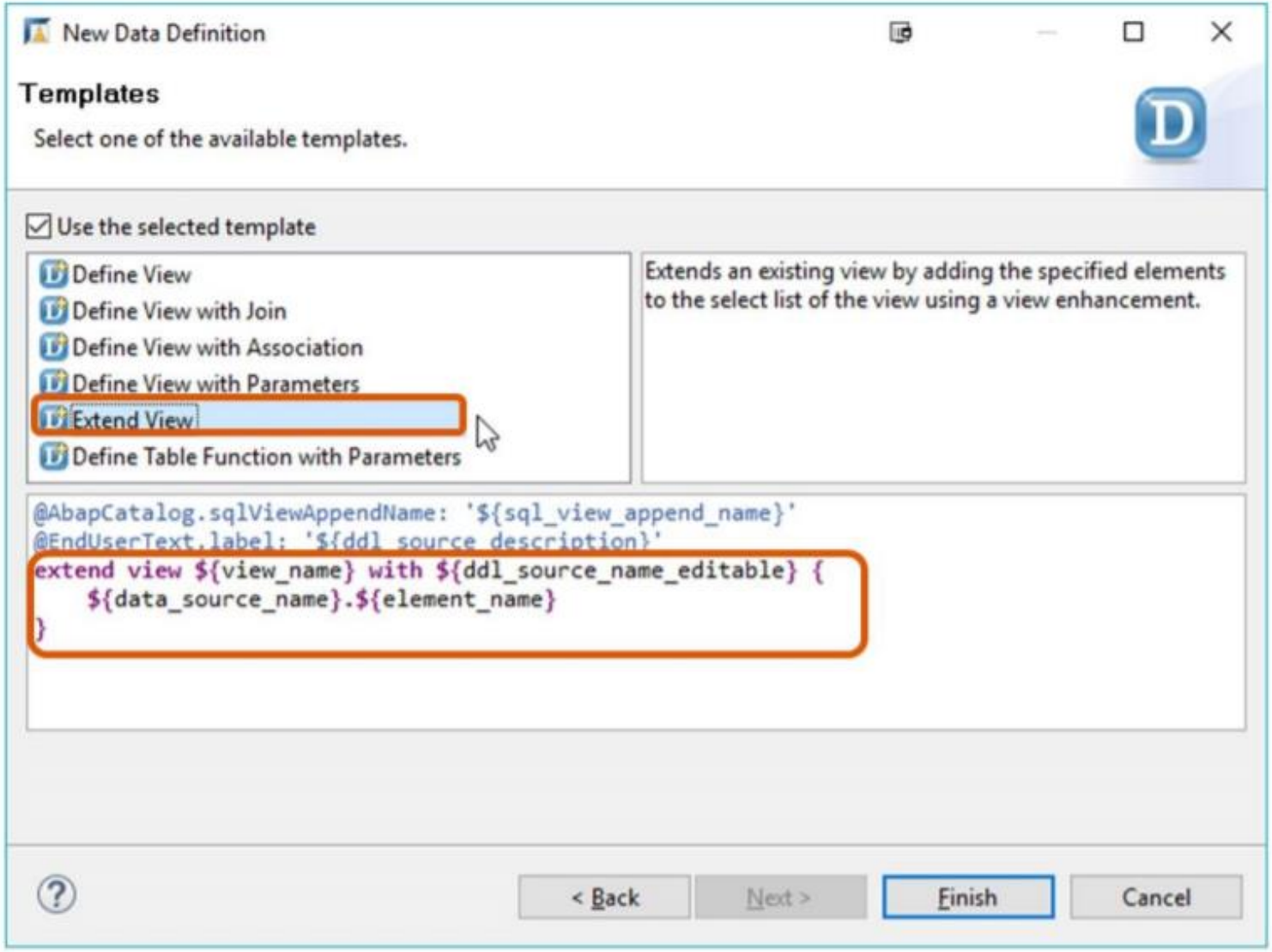
Additional aggregations in element list

Extension of the GROUP BY clause

Key word UNION, followed by extension for the other SELECT

Template for Extend View

SAP Development Learning



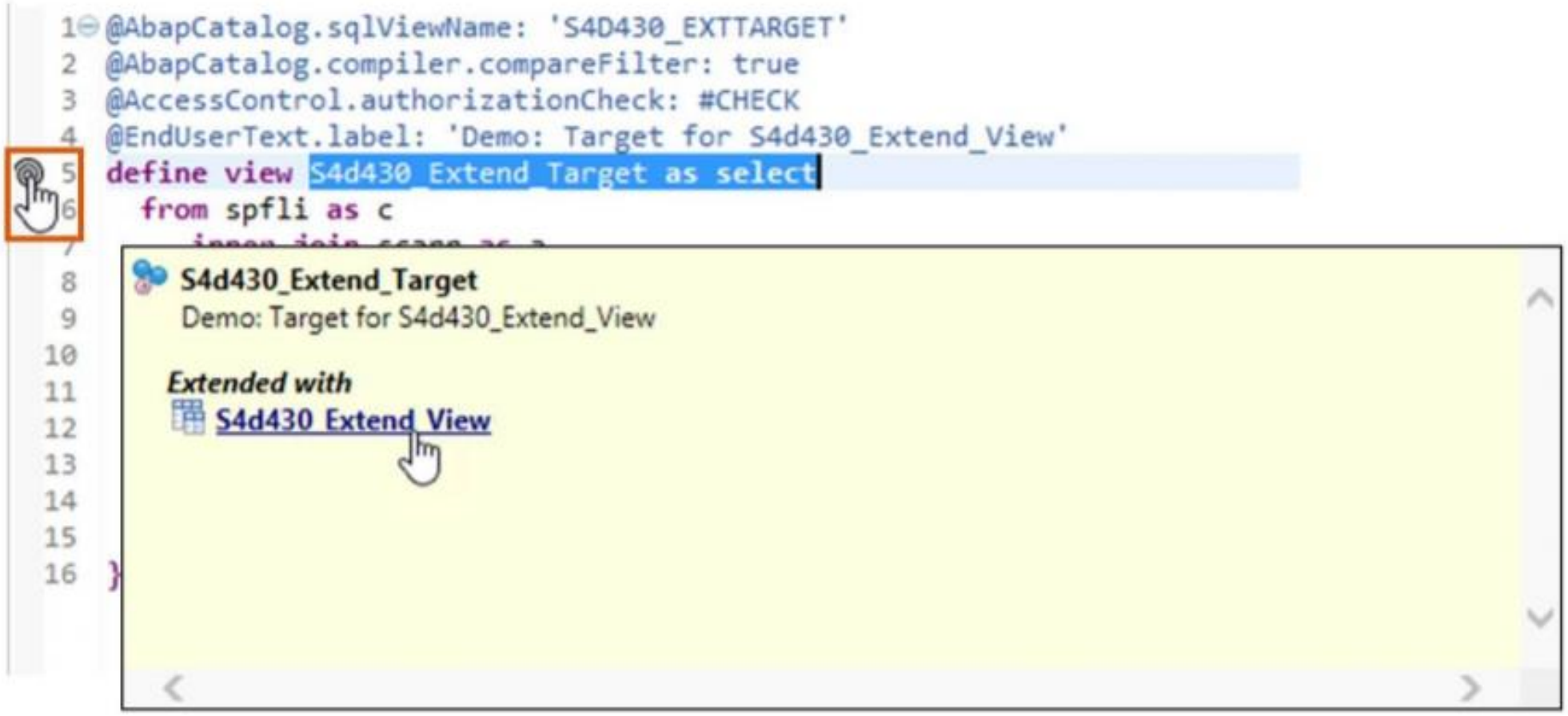
Restricting Extensibility (as of Release 7.51)

SAP Development Learning

- **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

SAP Development Learning



Exercise 6: Extend CDS Views

SAP Development Learning

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

SAP Development Learning

Unit 2 – Advanced Concepts

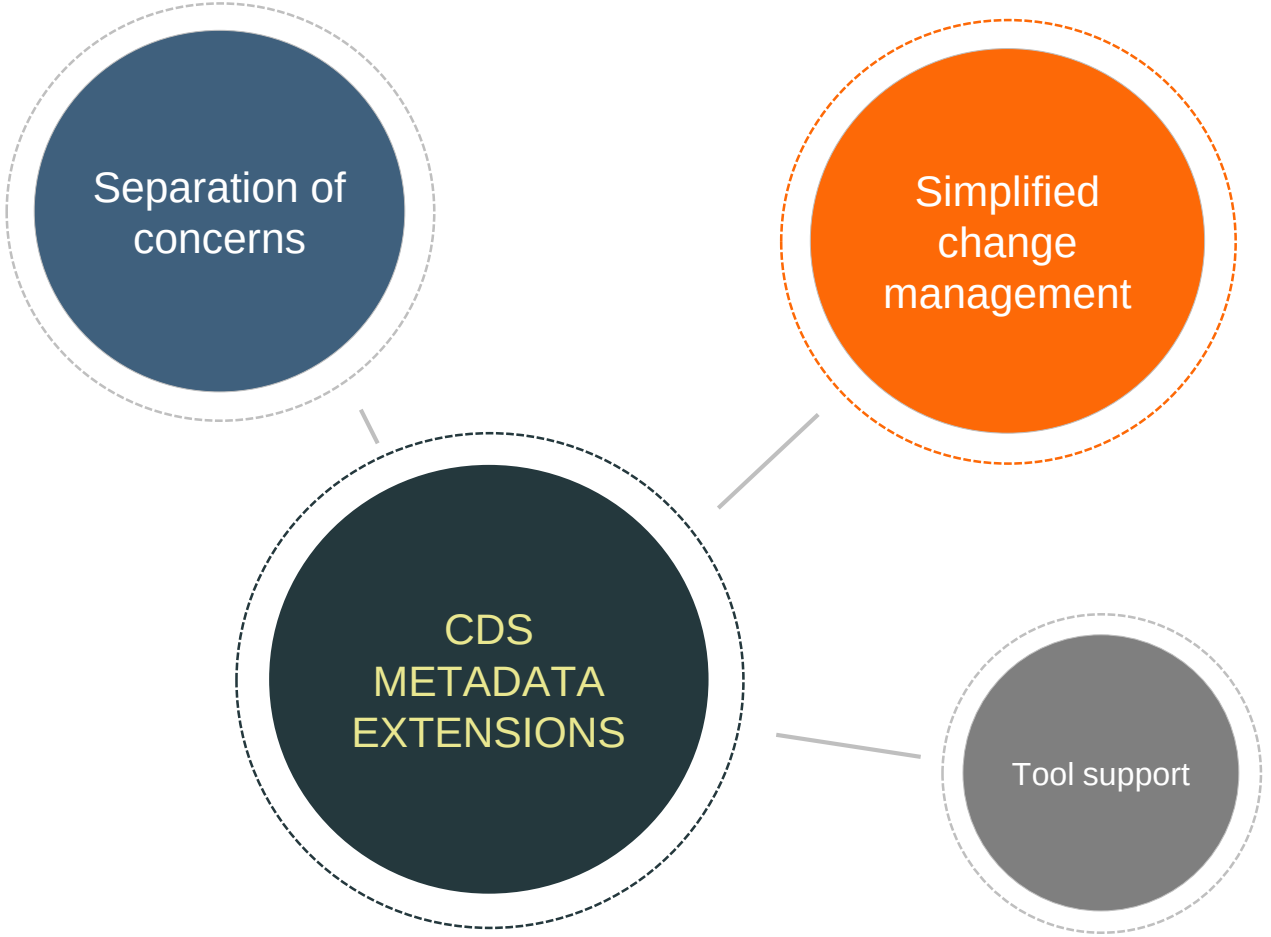
Lesson 4 – Metadata Extension

CDS metadata extensions (MDE)

SAP Development Learning

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

Wizard for extracting existing metadata from CDS view to CDS MDE

CDS View and Metadata Extension

SAP Development Learning

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

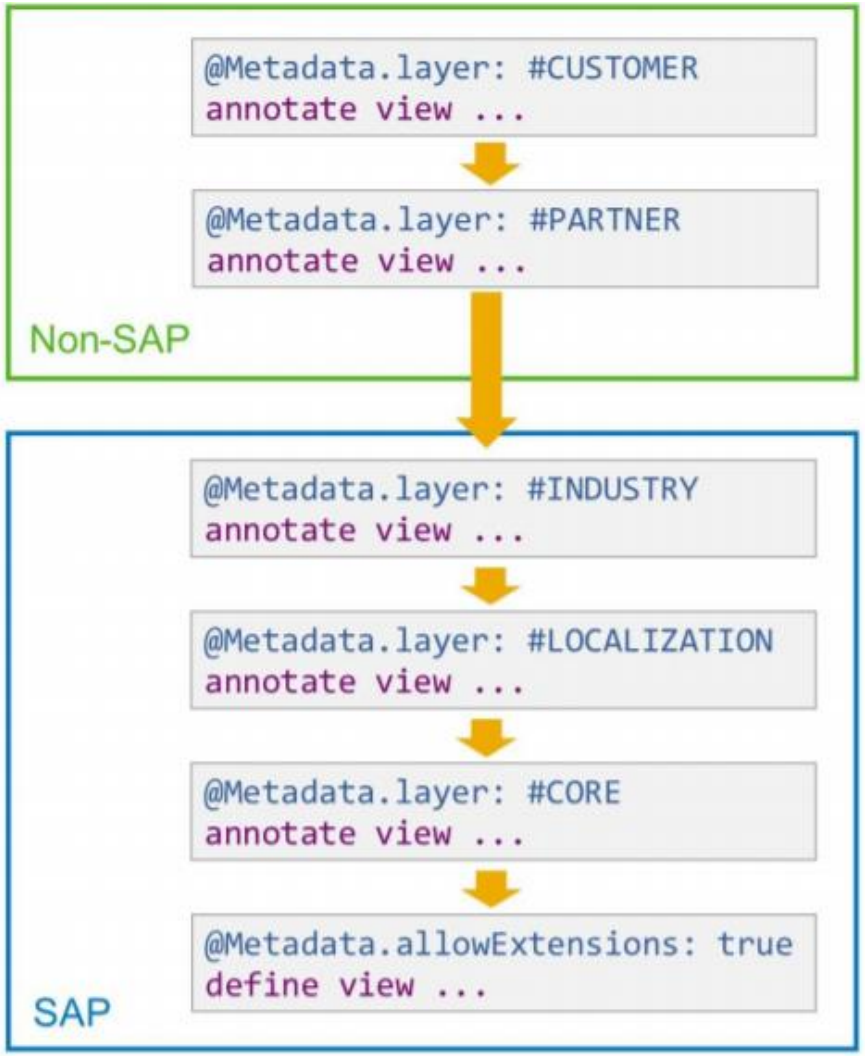
```
@AbapCatalog.sqlViewName: 'S4D430_MDE'

@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

SAP Development Learning

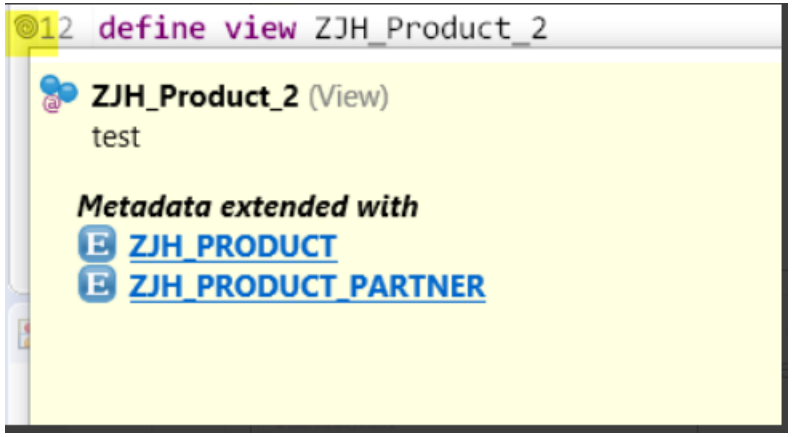
- **Several Matadata Extensions for same CDS View**
- **Priority defined through annotation @Metadata.layer**
- **Values:**
  - #Customer
  - #Partner
  - #Industry
  - #Localization
  - #Core
- **Highest Priority: #Customer**



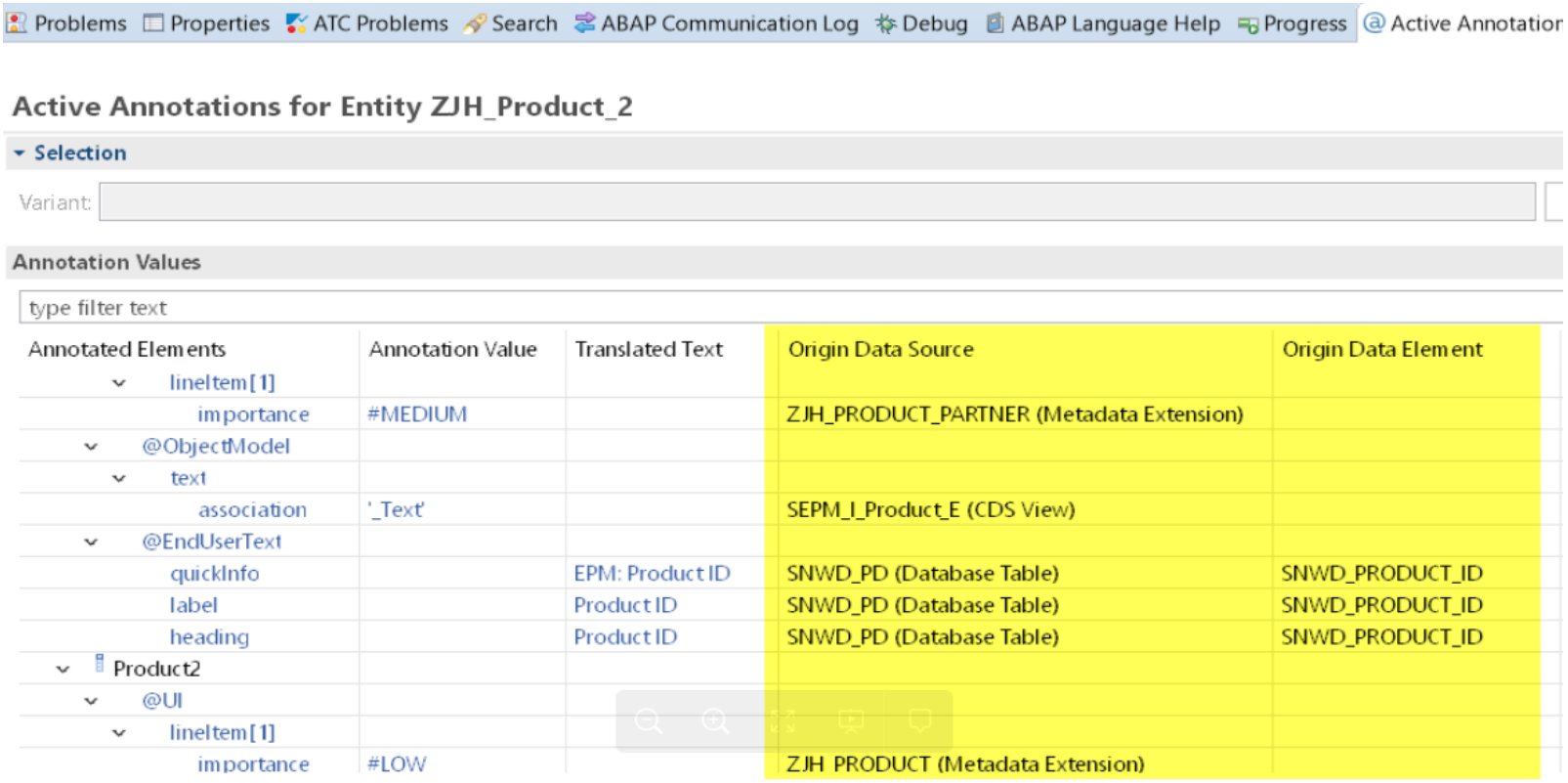
Metadata Extensions for a CDS view

SAP Development Learning

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



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)	

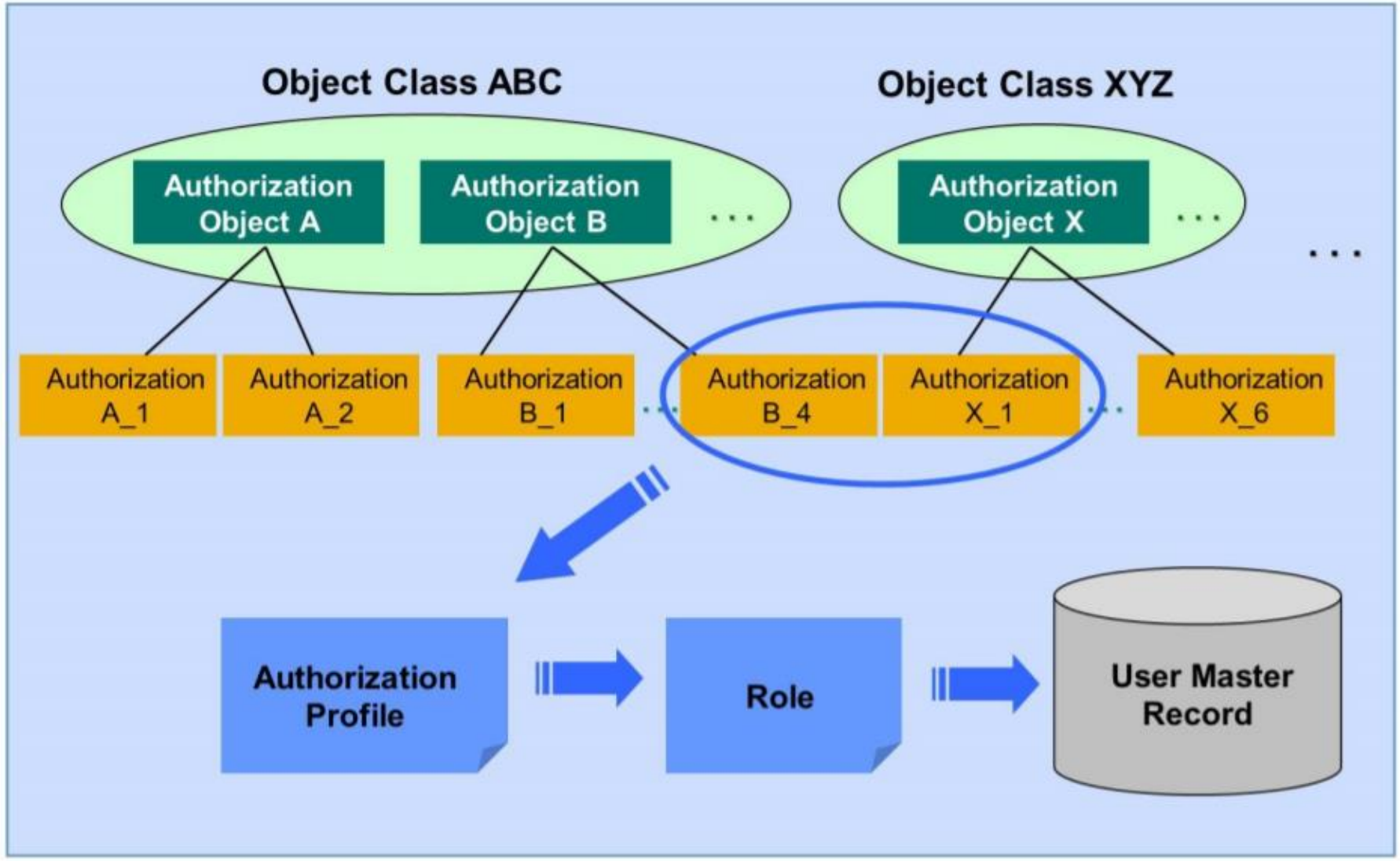
SAP Development Learning

Unit 2 – Advanced Concepts

Lesson 5 – Metadata Extension

Authorization Objects, Profiles, and Roles

SAP Development Learning



Implicit and Explicit Authorization Checks

SAP Development Learning

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

SAP Development Learning

```
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

Known Problems of the Classical Approach

SAP Development Learning

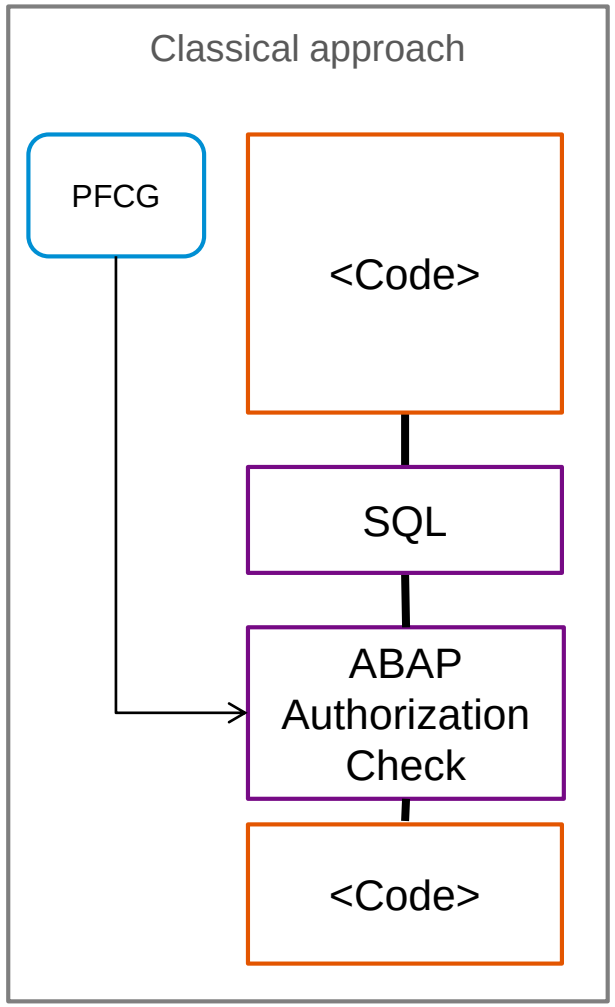
- **General Problems:**
  - No visible link between data and authorization objects
  - Developer can forget (or ignore) necessary checks
  - Users with debugging rights can bypass authorization checks

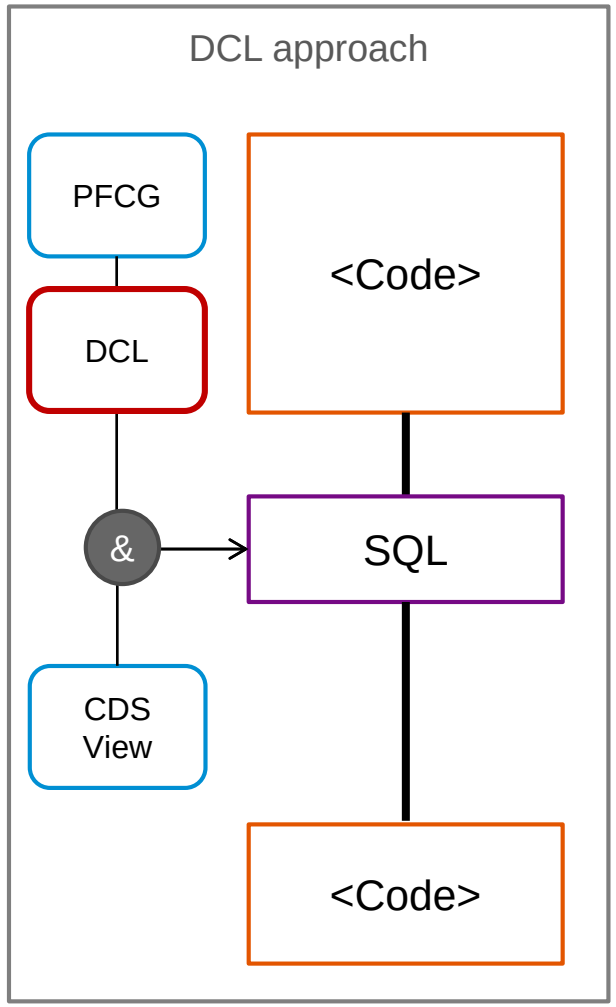
- **Problems related to Code-to-Data:**
  - All authorization checks on application level
  - Some calculations cannot be done on the database because they require an authorization check between steps

Data Control Language Overview

SAP Development Learning

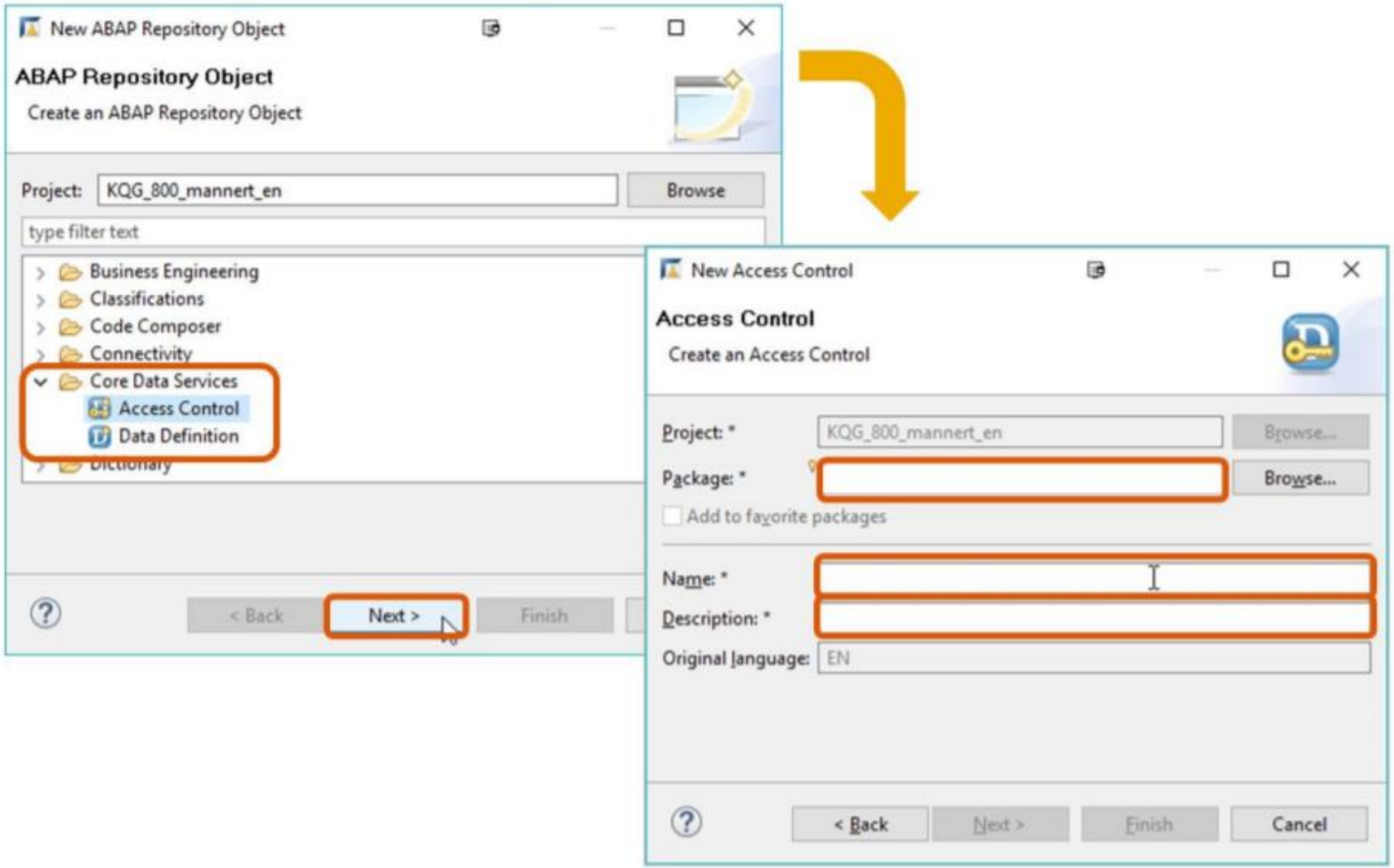
- Declarative approach instead of coded
- Based on CDS modeling objects and therefore part of the data-model
- Authorizations are also pushed down to DB by extending the Open SQL SELECT statement
- Authorizations are only defined once and automatically (re-) used everywhere





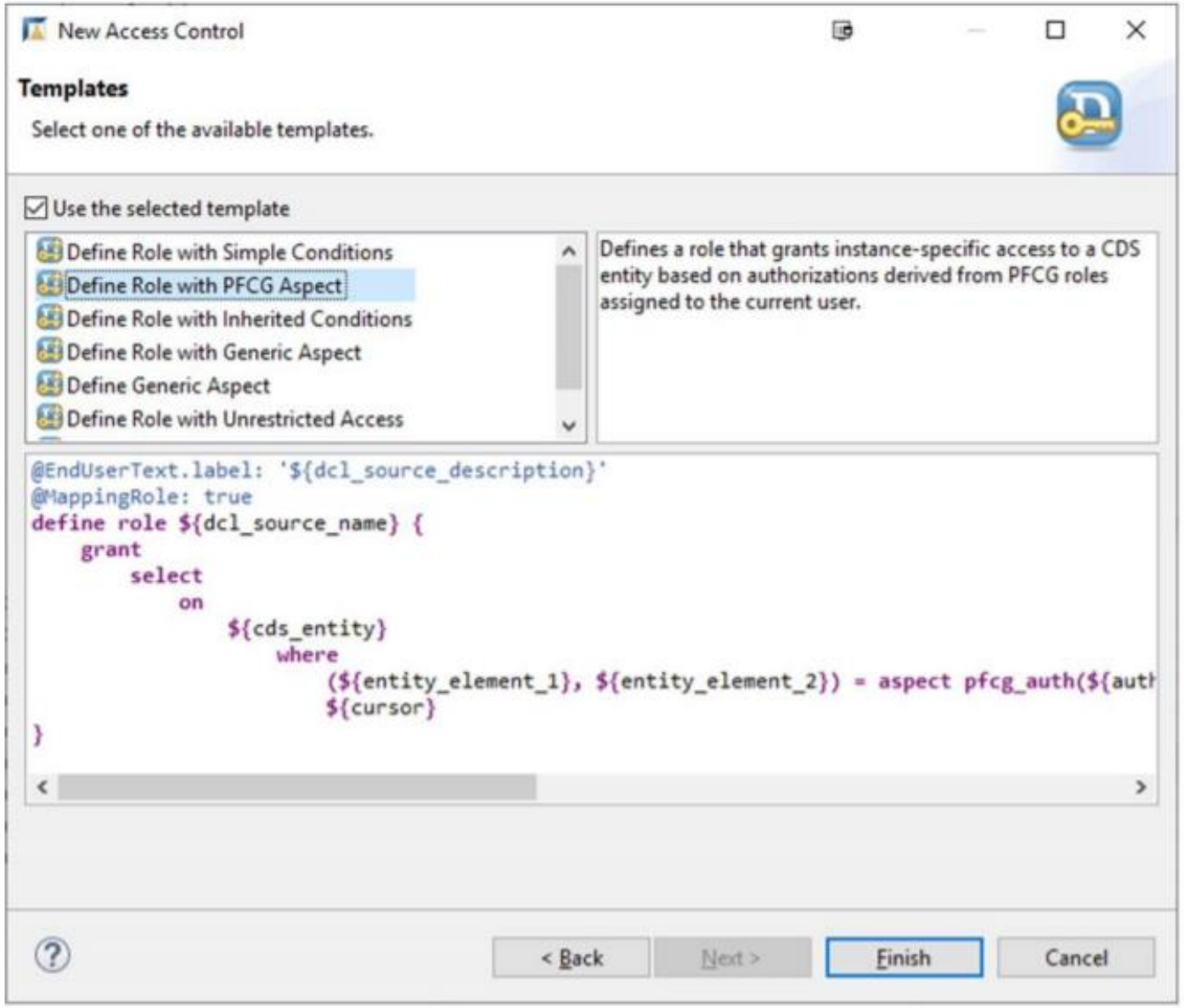
Create New DCL Source

SAP Development Learning



Templates for Access Controls

SAP Development Learning



ABAP CDS DCL Syntax

SAP Development Learning

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

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

Key word DEFINE is optional

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

At least one condition per GRANT statement

Delimiter „;“ mandatory after each GRANT statement

Types of Conditions in DCL

SAP Development Learning

- **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

PFCG Conditions

SAP Development Learning

```
@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

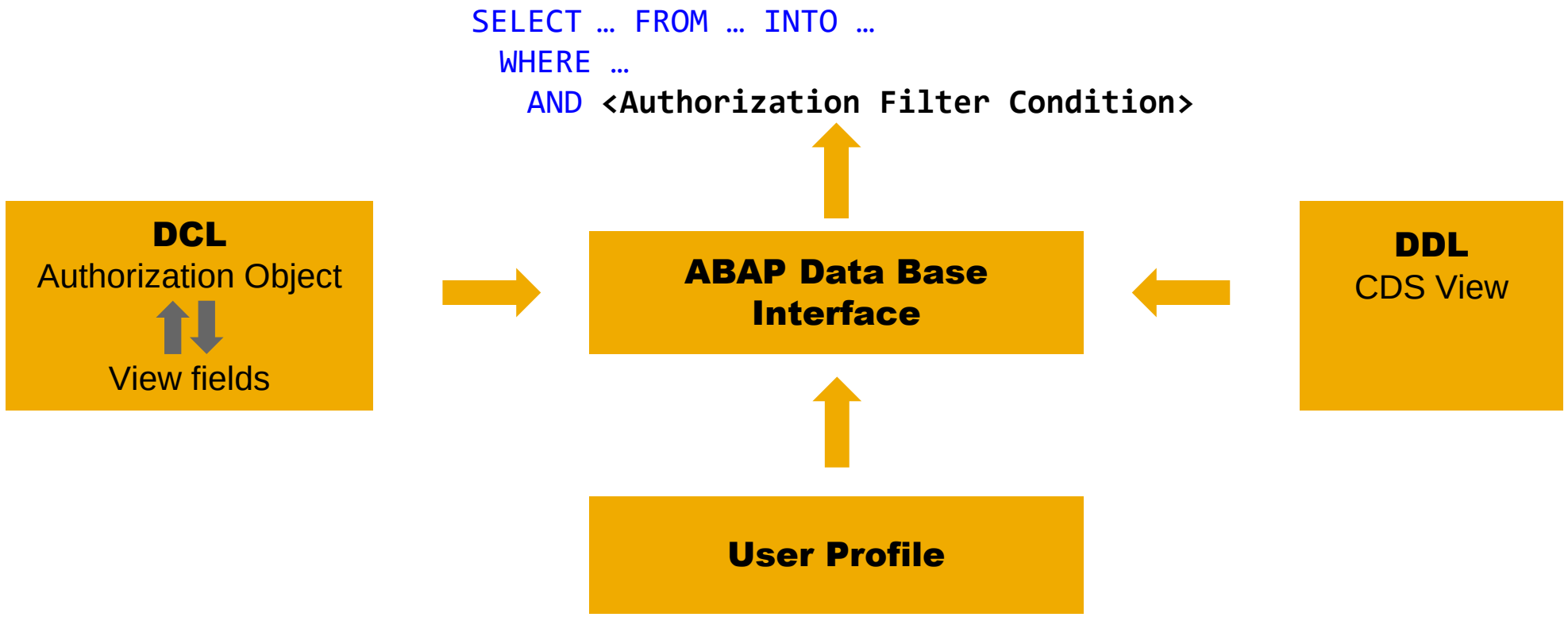
Right-hand side:
Key word ASPECT and function pfcg_auth( )

Arguments of pfcg_auth:
One authorization object with its fields

Runtime Behavior

SAP Development Learning

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



Authorization-related Annotations

SAP Development Learning

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

Example of a DCL Role

SAP Development Learning

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

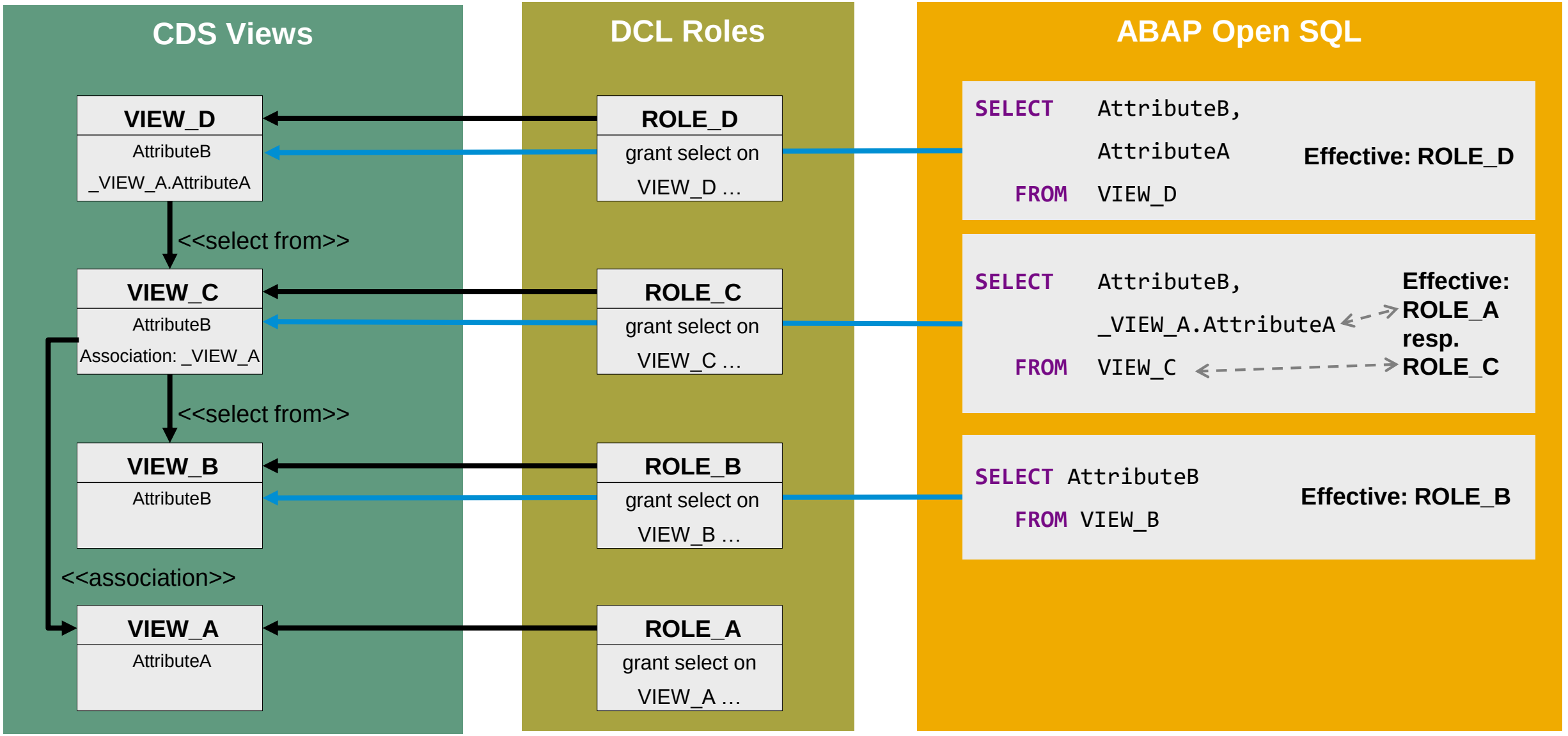
```
@MappingRole: true
define role I_SalesOrder {
    grant select on I_SalesOrder
    where
        ( SalesOrderType )
    = aspect pfcg_auth
        ( v_vbak_aat,
          auart,
          actvt = '03' )
    and
        (SalesOrganization,
         DistributionChannel,
         OrganizationDivision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

SAP Development Learning



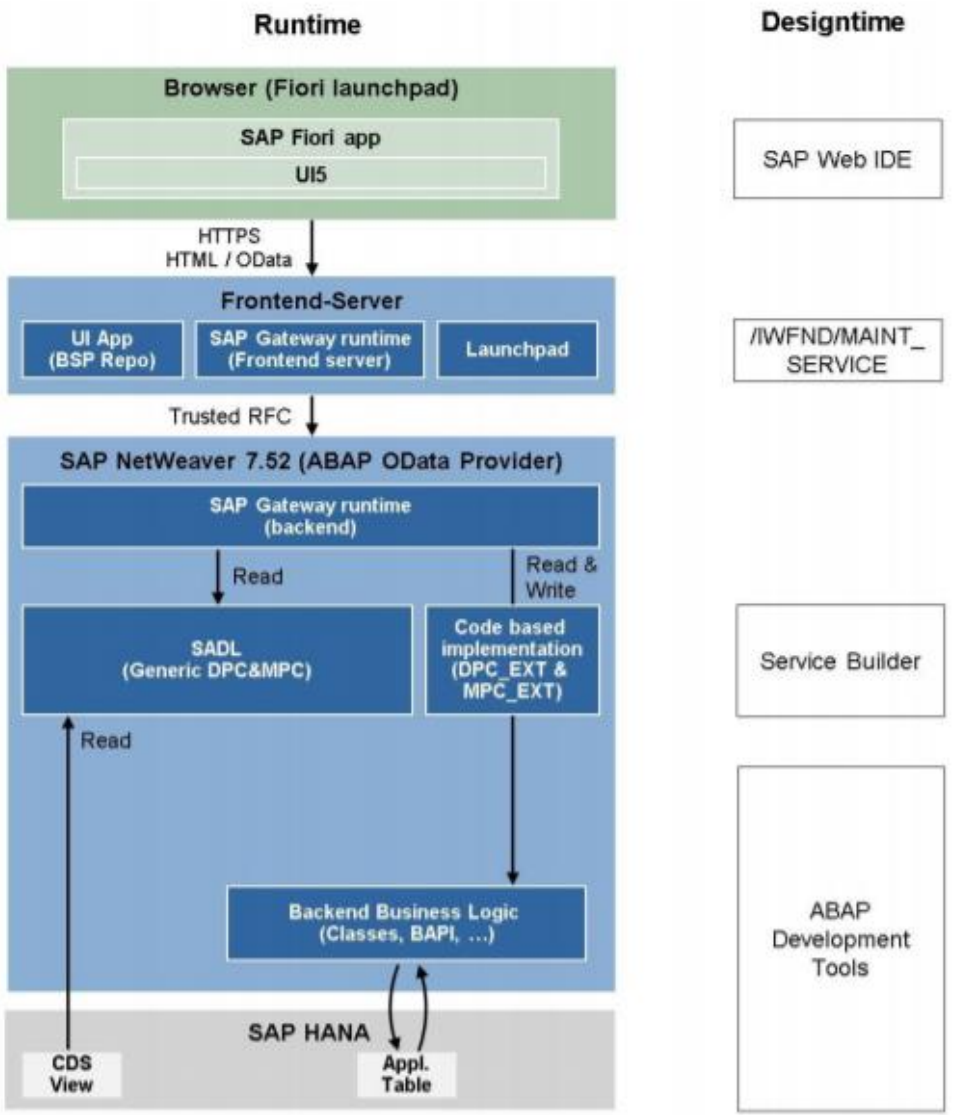
SAP Development Learning

Unit 2 – Advanced Concepts

Lesson 6 – Consumption of CDS Views in SAP Fiori

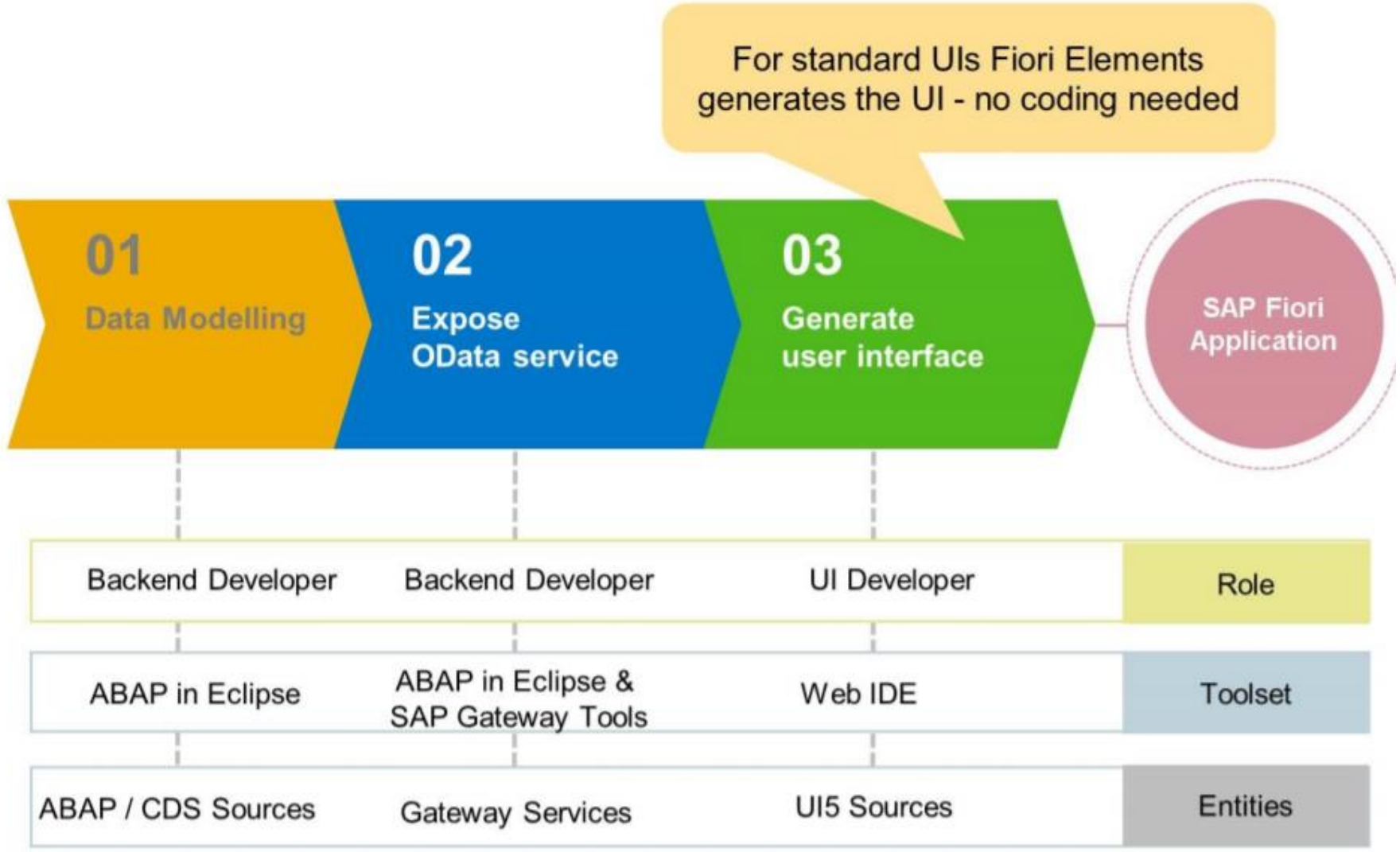
SAP Fiori Programming Model

SAP Development Learning



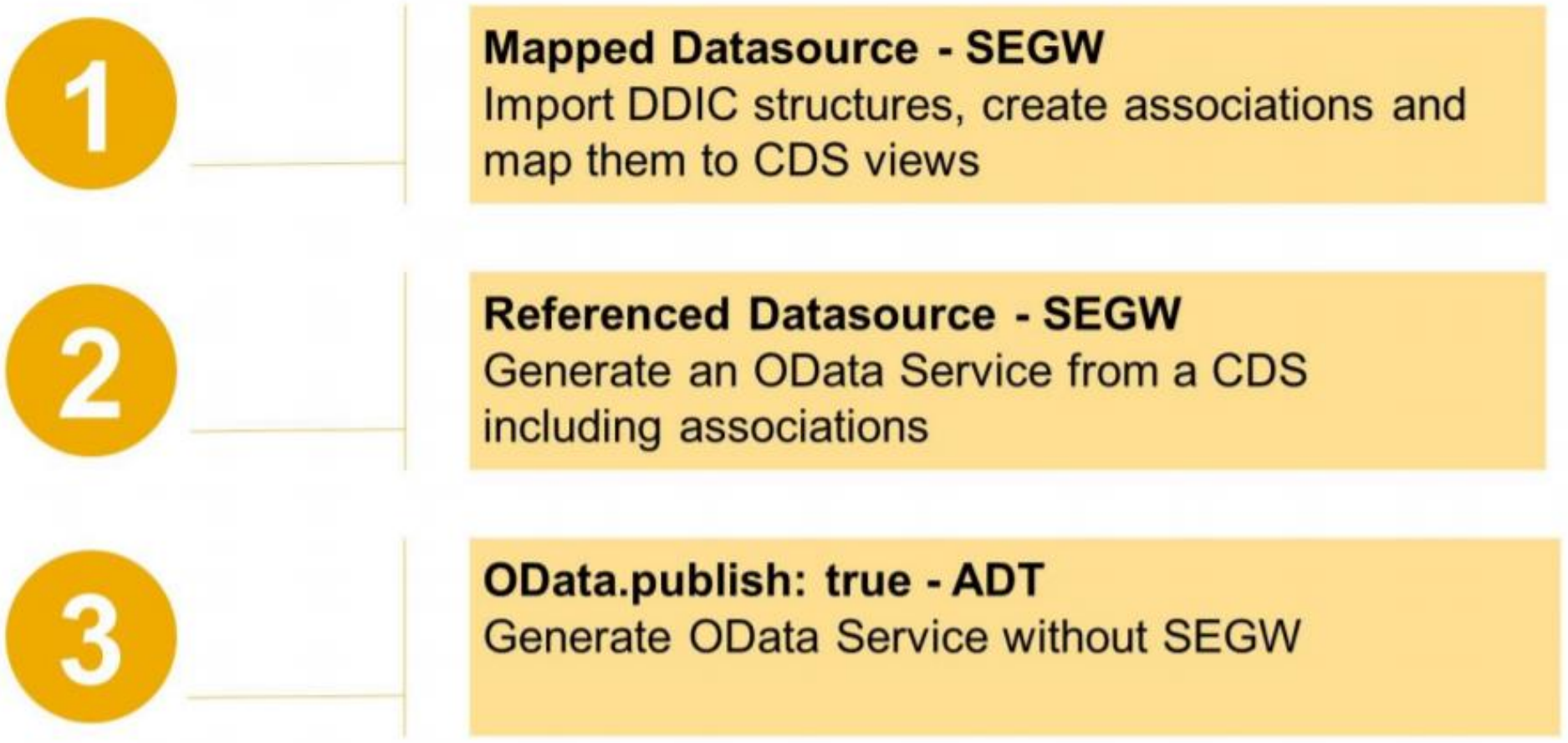
Key steps and Development Tools

SAP Development Learning



OData Service Development with CDS

SAP Development Learning



OData.publish: true

SAP Development Learning

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

Transaction: /IWFND/MAINT_SERVICES

UI-Specific Annotations

SAP Development Learning

```
@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

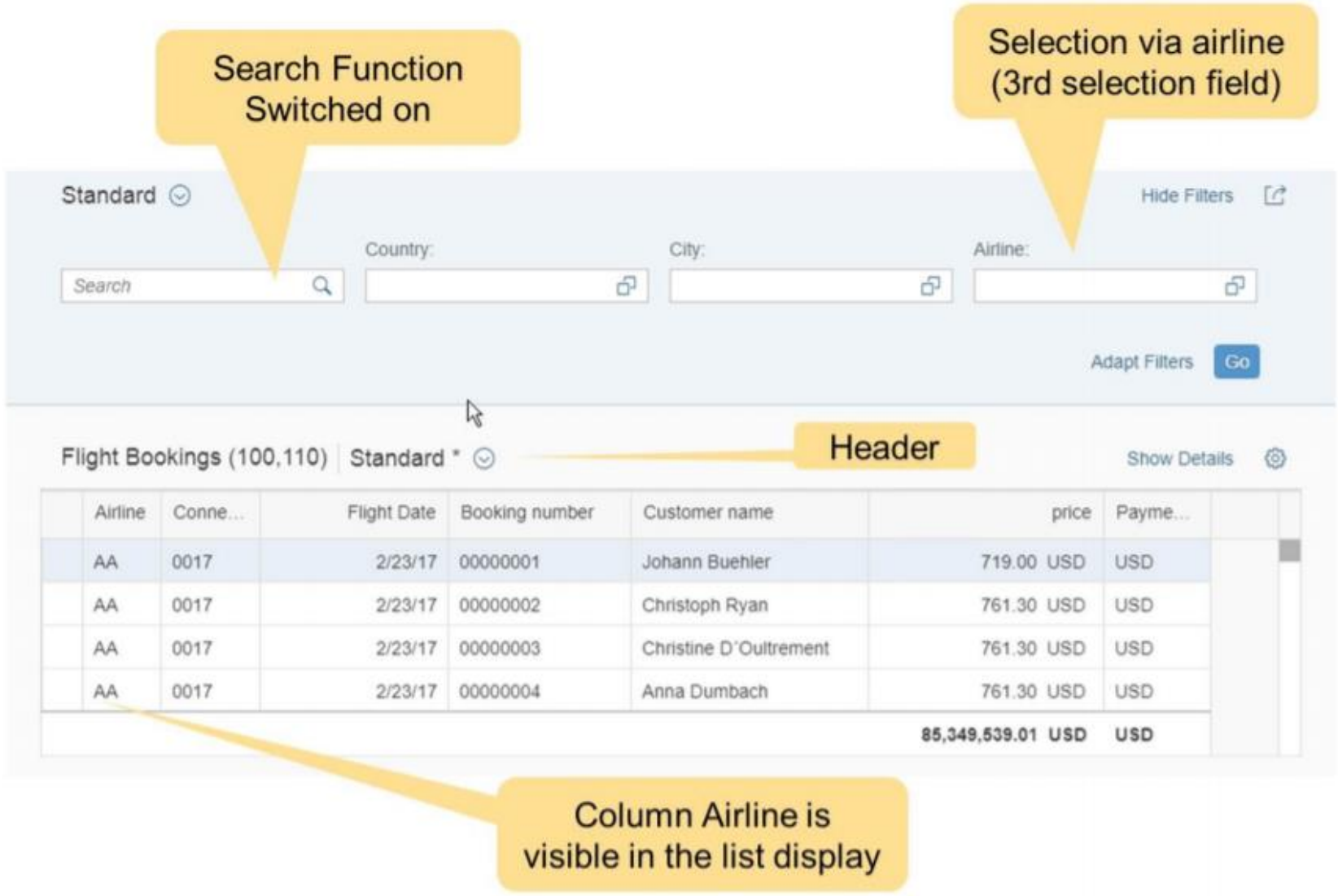
Enable a Search Dialog

Display this Column and Set the Position

Display this field in the selection dialog

UI-Specific Annotations – The Outcome

SAP Development Learning



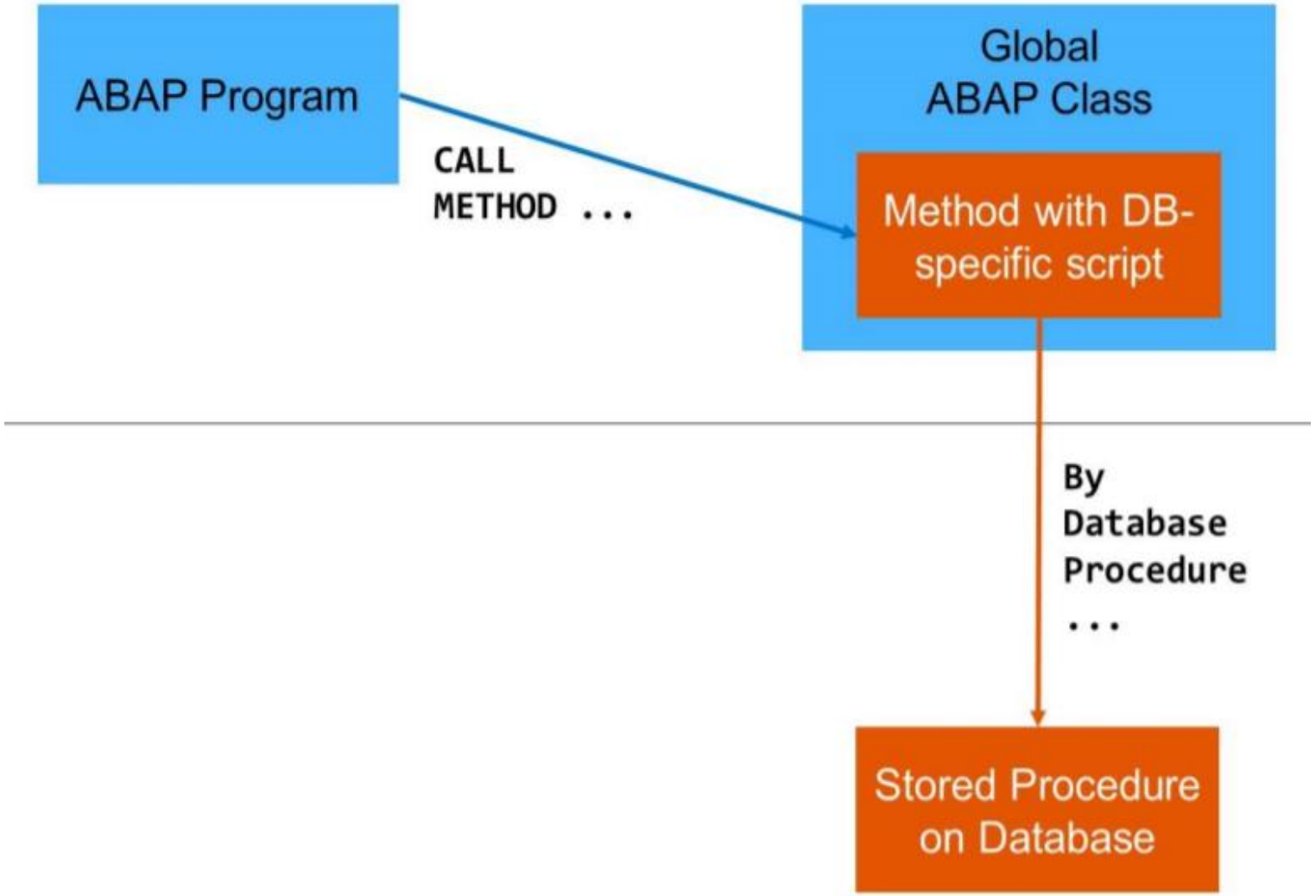
SAP Development Learning

Unit 3 – HANA Native features in CDS

Lesson 1 – ABAP Managed Database Procedure

Basic idea of AMDP

SAP Development Learning



Definition of an AMDP Method

SAP Development Learning

```
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

SAP Development Learning

```
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)

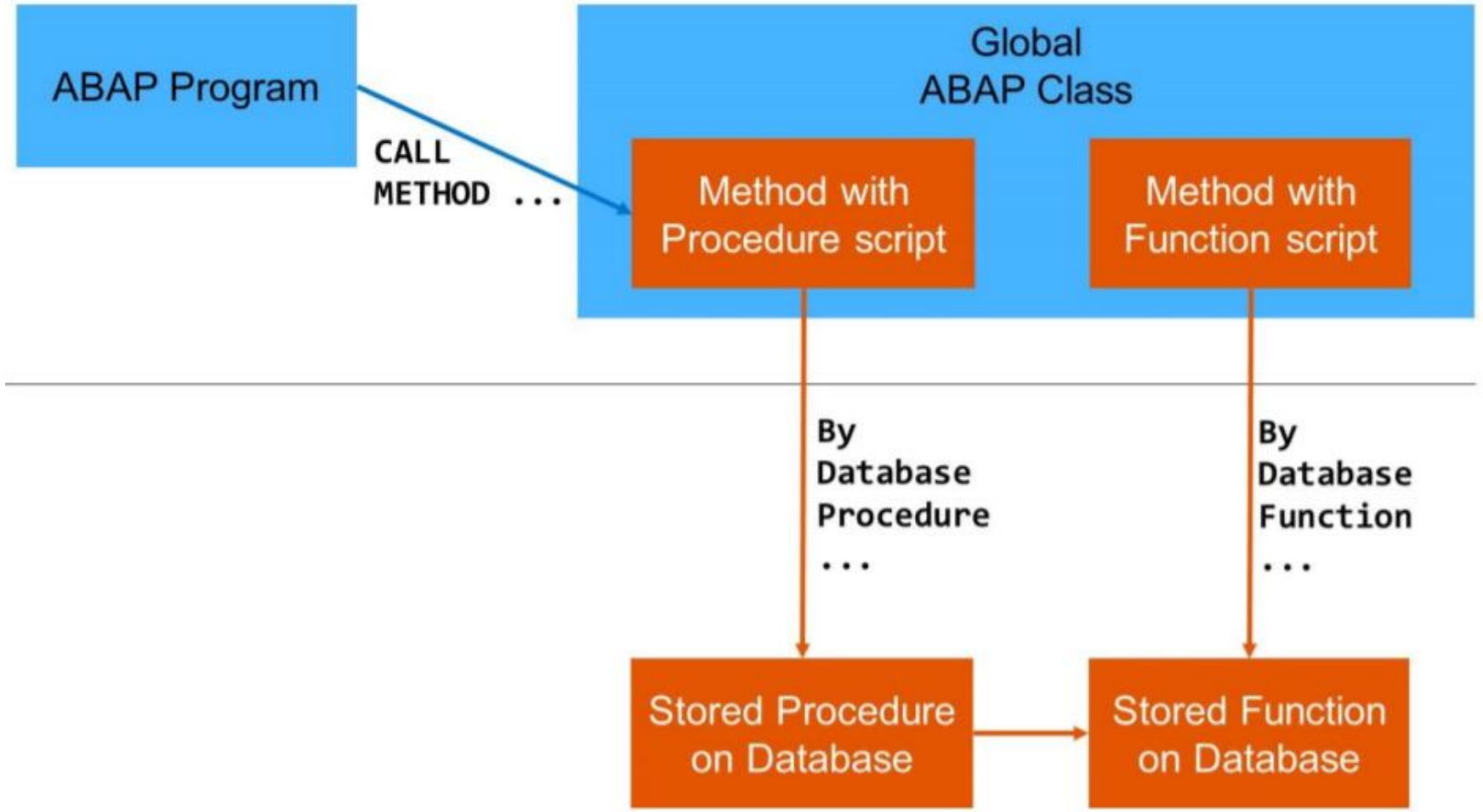
Prerequisites and Limitations

SAP Development Learning

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

AMDP Functions

SAP Development Learning



Definition of an AMDP Function

SAP Development Learning

```
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

SAP Development Learning

```
CLASS cl_s4d430_amdp_demo IMPLEMENTATION.

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

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

  ENDMETHOD.

ENDCLASS.
```

Return value is a table with columns ID and NAME

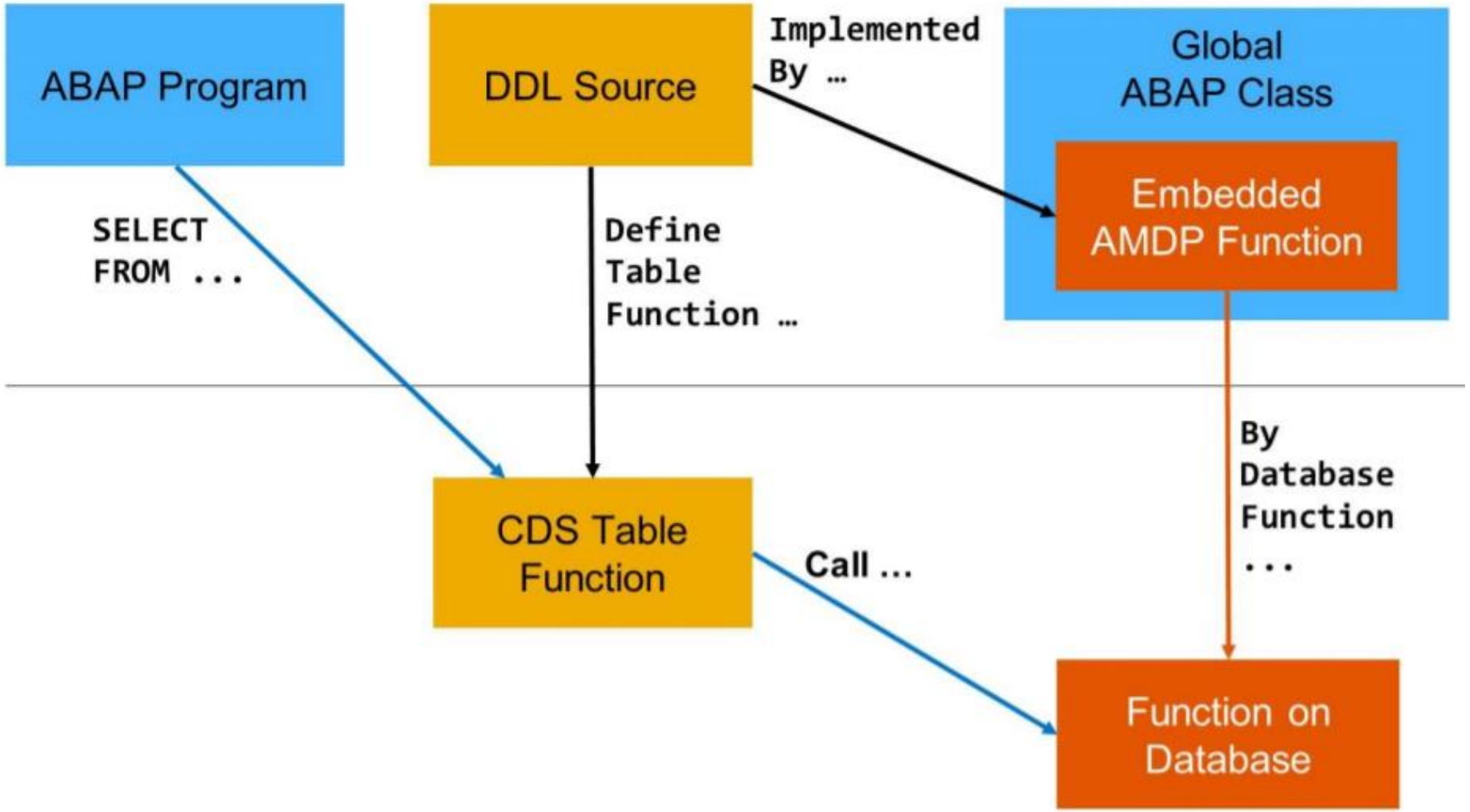
SAP Development Learning

Unit 3 – HANA Native features in CDS

Lesson 2 – CDS Table Function

The Basic Idea of CDS Table Functions

SAP Development Learning



Definition of a CDS Table Function

SAP Development Learning

```
@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

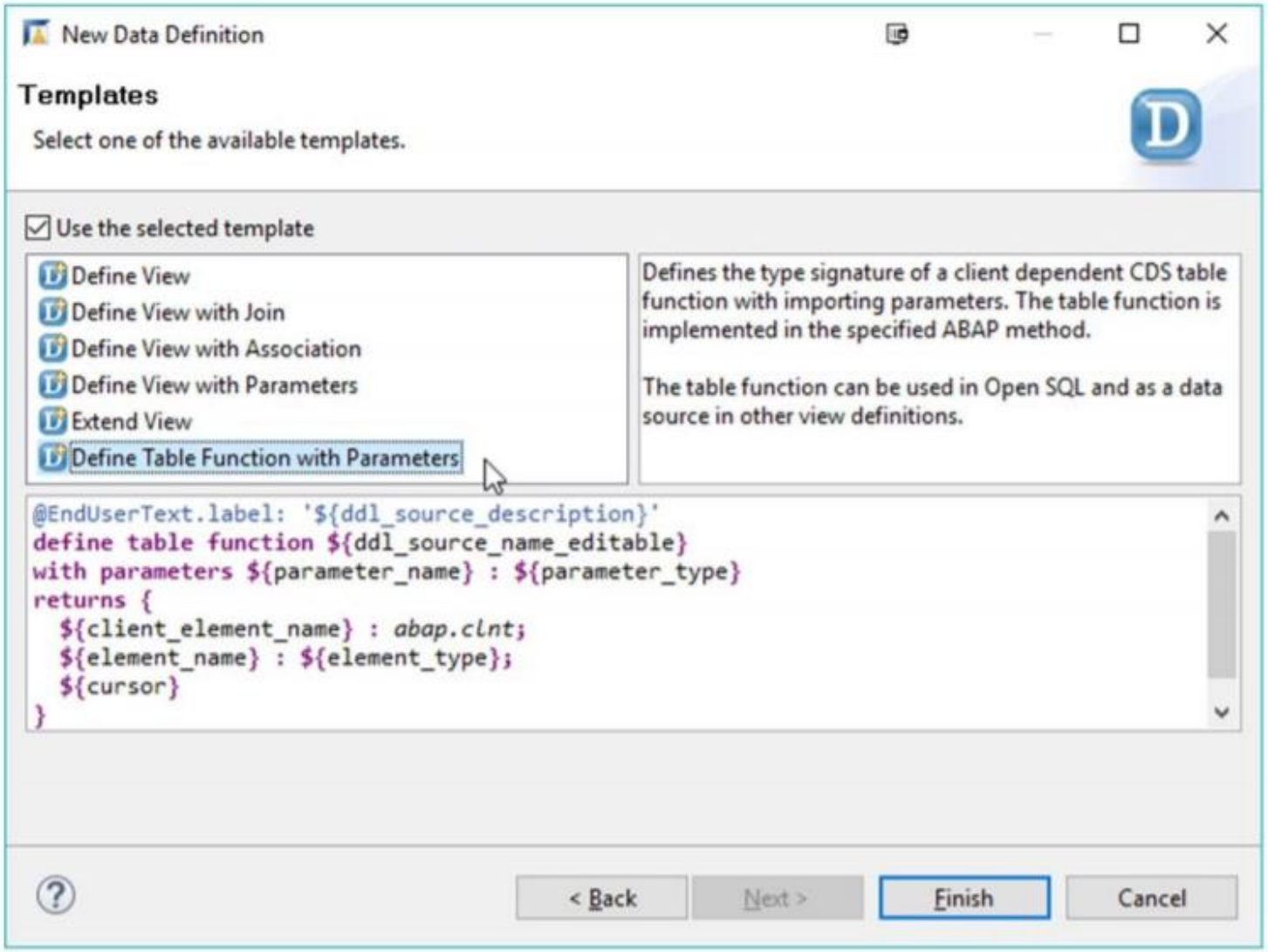
SAP Development Learning

```
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

SAP Development Learning



CDS Table Function: Recommended Sequence

SAP Development Learning

- **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

SAP Development Learning

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

SAP Development Learning

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

Check system capability and provide alternative coding

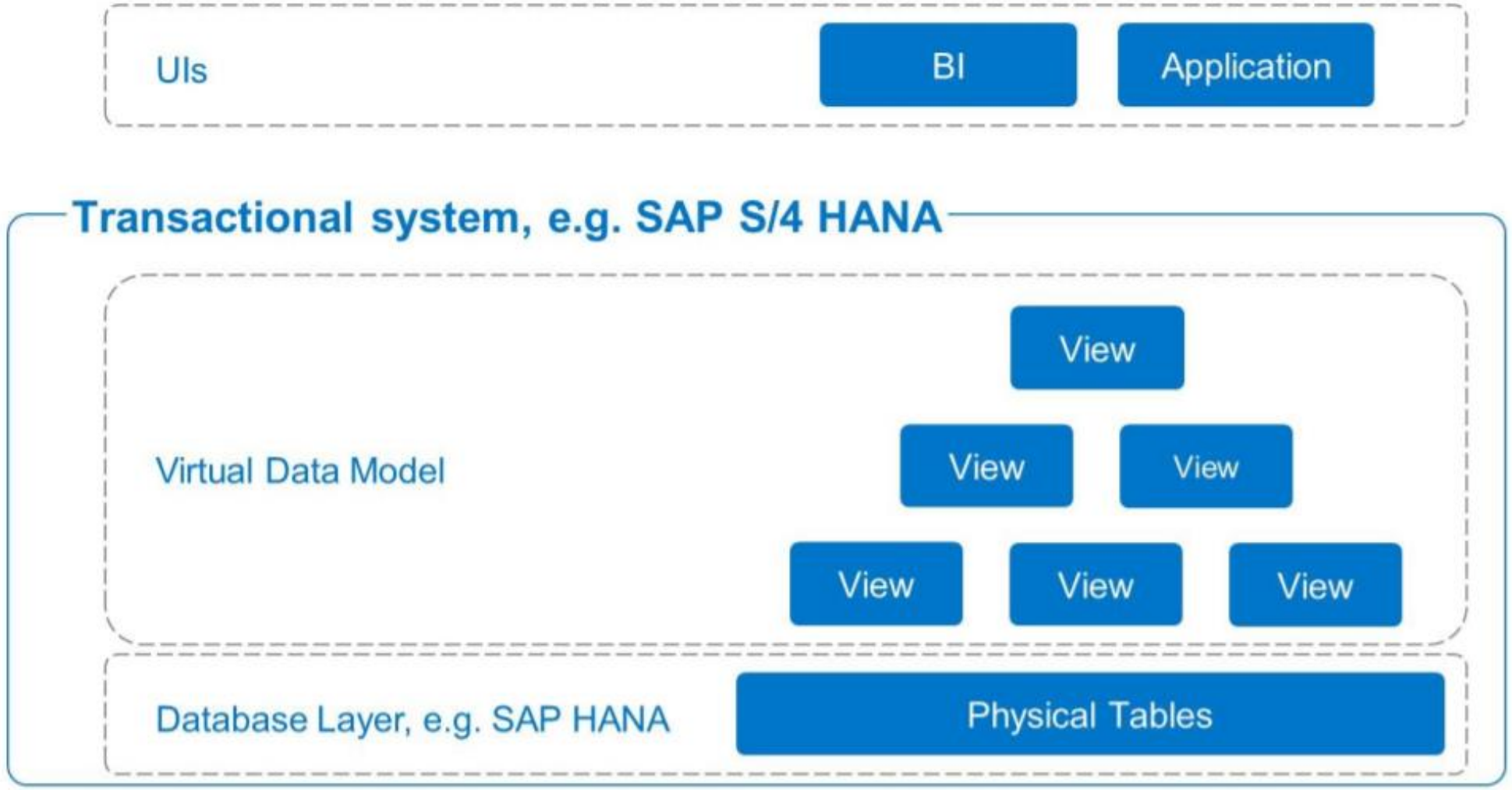
```
DATA gt_feature_set TYPE cl_abap_dbfeatures=>features_set_t.

INSERT cl_abap_dbfeatures=>amdp_table_functions
       INTO TABLE gt_feature_set.

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

SAP Development Learning

Unit 4 – Virtual Data Model

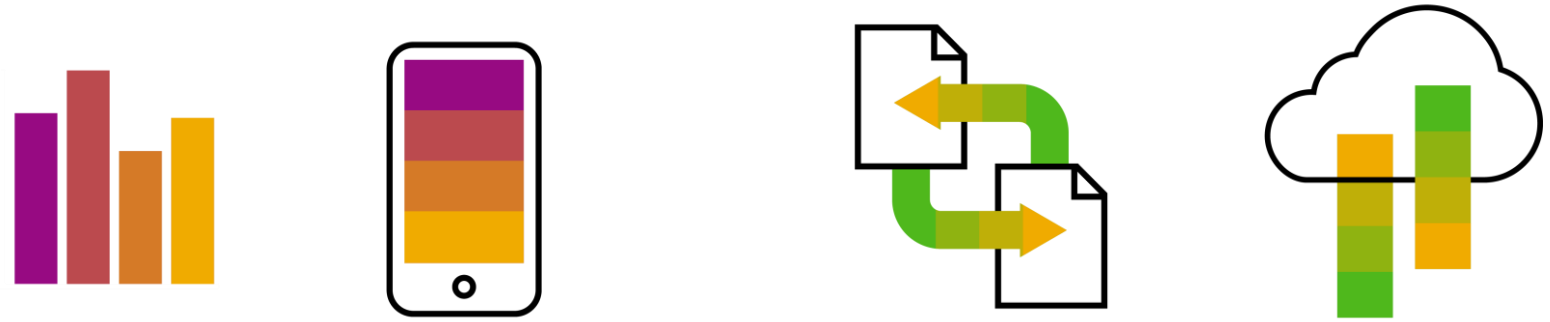
Virtual Data Models

SAP Development Learning



Stack of VDM Views

SAP Development Learning



All types of consumers

VDM Consumption Views C_*

VDM External API Views A_*

Purpose-built for apps and APIs

VDM Composite Interface Views I_*

Convenience and special views for reuse

VDM Basic Interface Views I_*

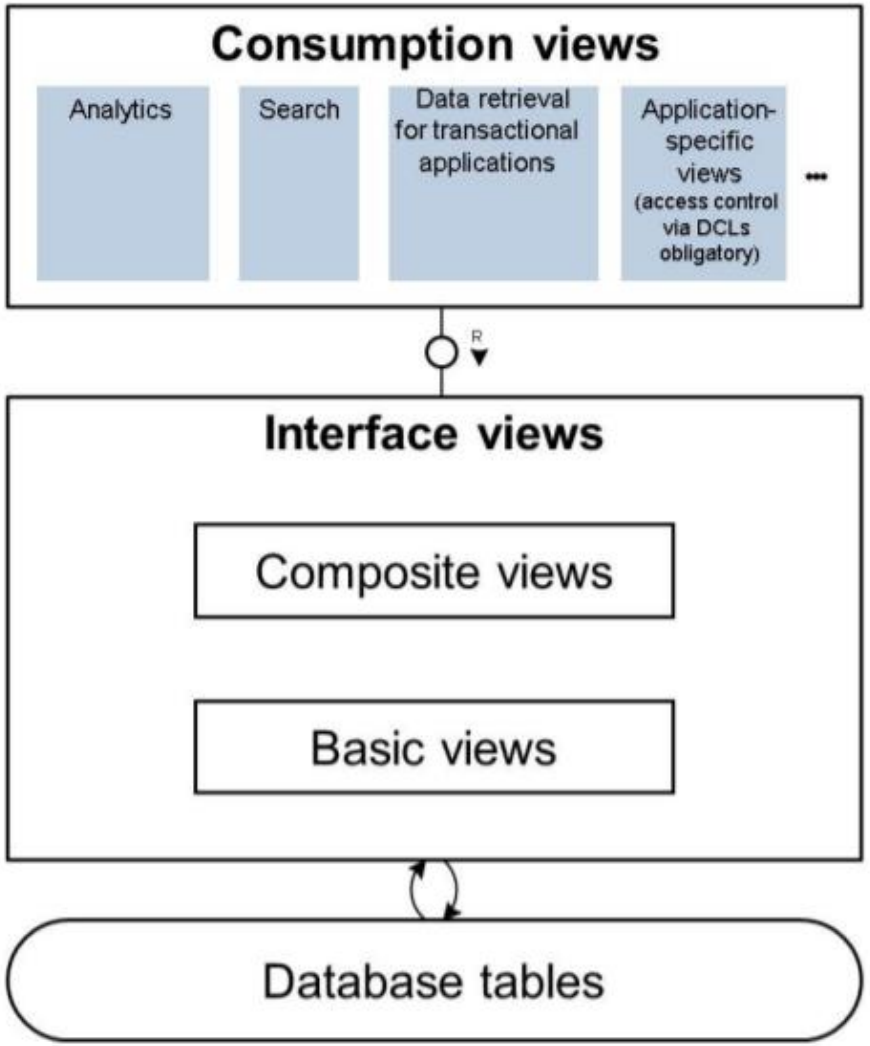
Basic VDM for reuse

Application Data (Tables)

Cryptic, invisible

The Layers in the Virtual Data Model

SAP Development Learning



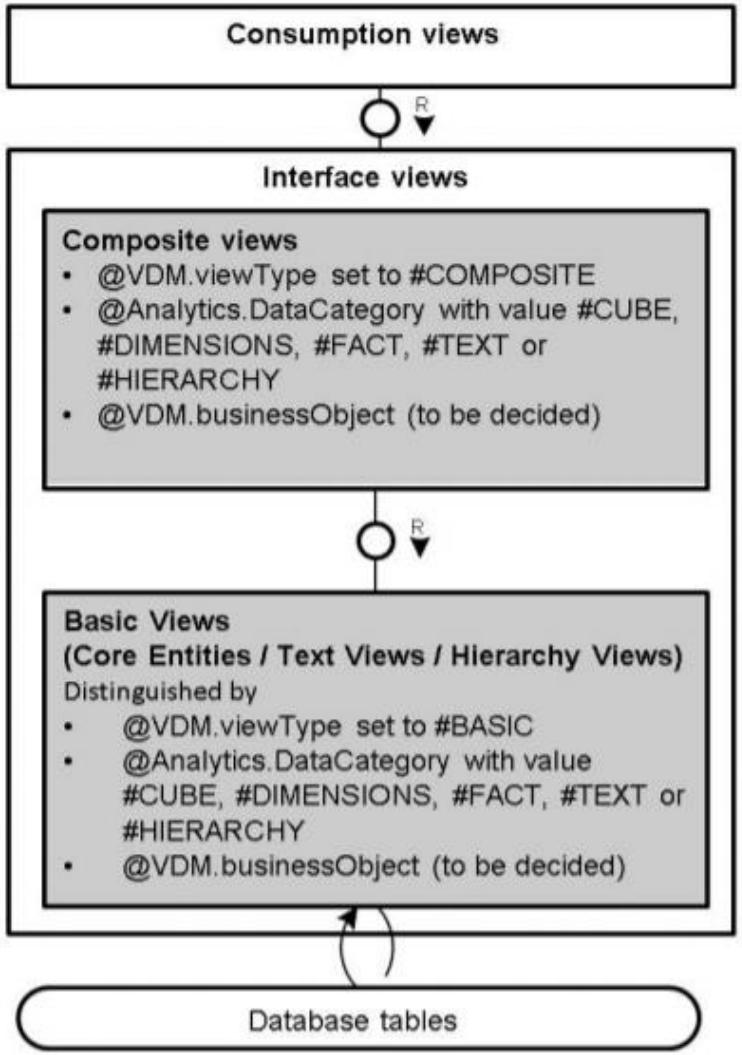
- Views from various consumption domains (e.g. Analytics, Search or …) which consume interface views and create queries on top to consume interface views

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

- Suite database model „as it is“

Interface Views Divided into Basic and Composite Views

SAP Development Learning

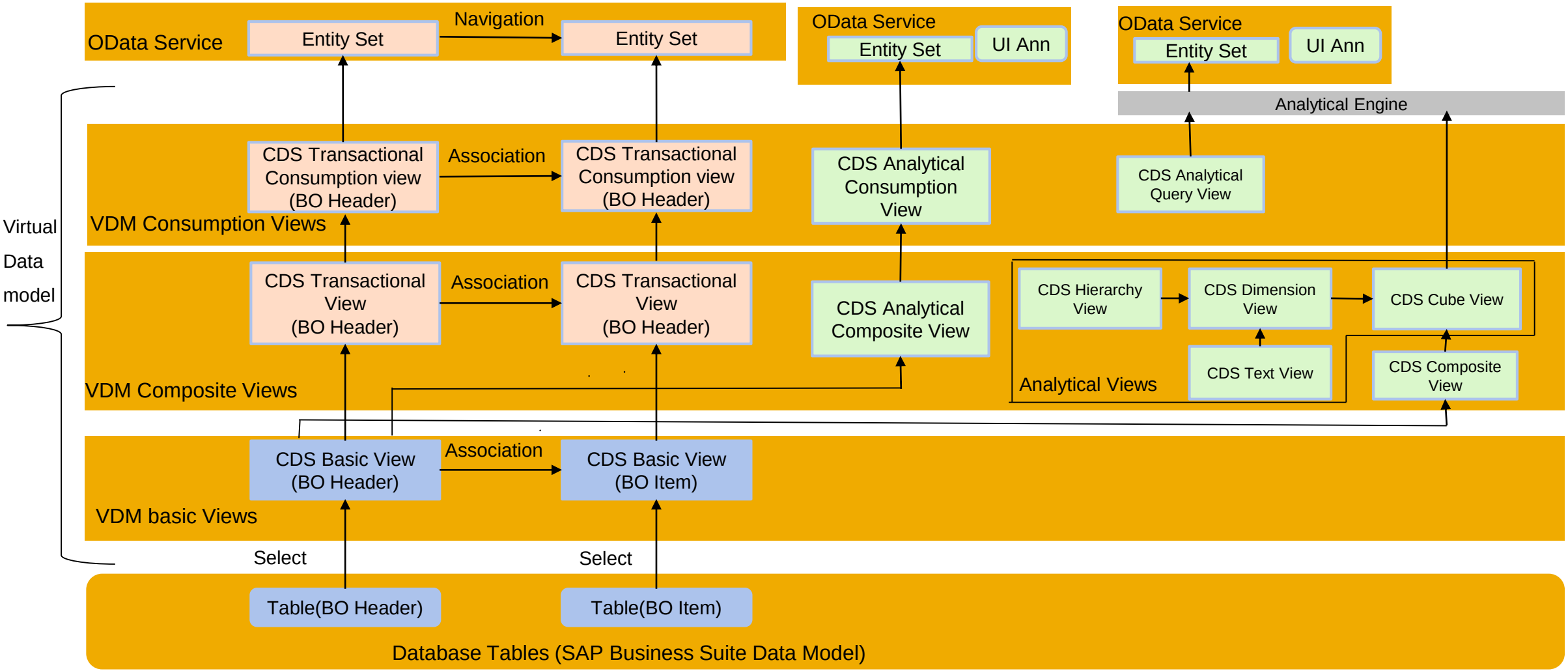


Composite views are built selecting from the basic views and exploring the associations between basic views. They can be specific to a consumption domain or reusable in many.

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

SAP Development Learning



VDM CDS View Layering

Naming Conventions

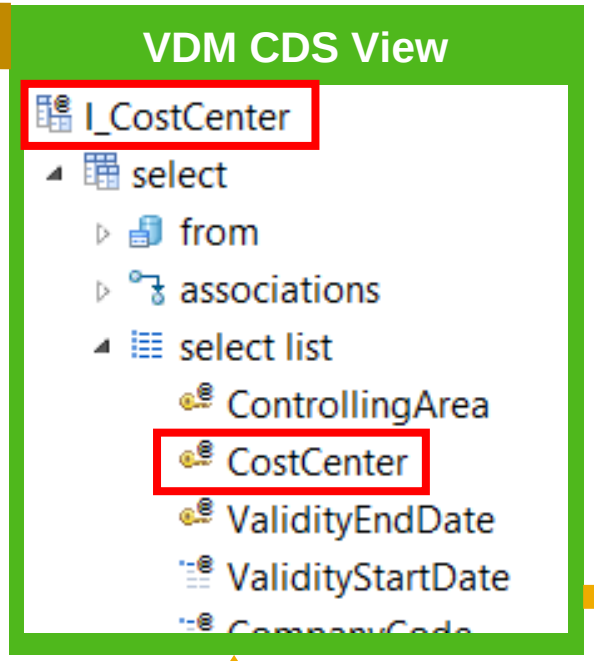
SAP Development Learning

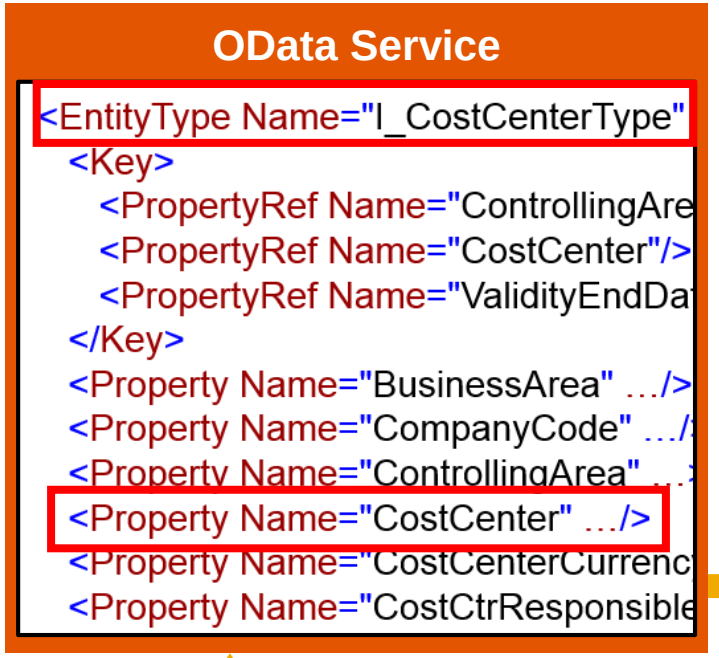
VDM View Type	Private	CDS Name Prefix	SQL Name Prefix	Global Field Name Checks
#CONSUMPTION		C_	C	X
#CONSUMPTION	X	P_	P	
#COMPOSITE		I_	I	X
#COMPOSITE	X	P_	P	
#BASIC		I_	I	X
#BASIC	X	P_	P	
#EXTENSION		E_	E	X

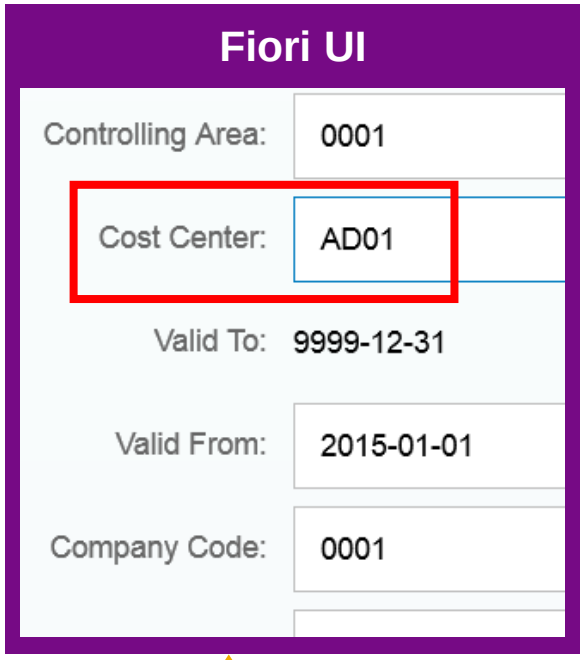
Names and meta data are propagated from CDS via OData to Fiori UIs

SAP Development Learning

Consistent, self-explanatory, business oriented naming of views and fields







Data Model-Related Annotations

SAP Development Learning

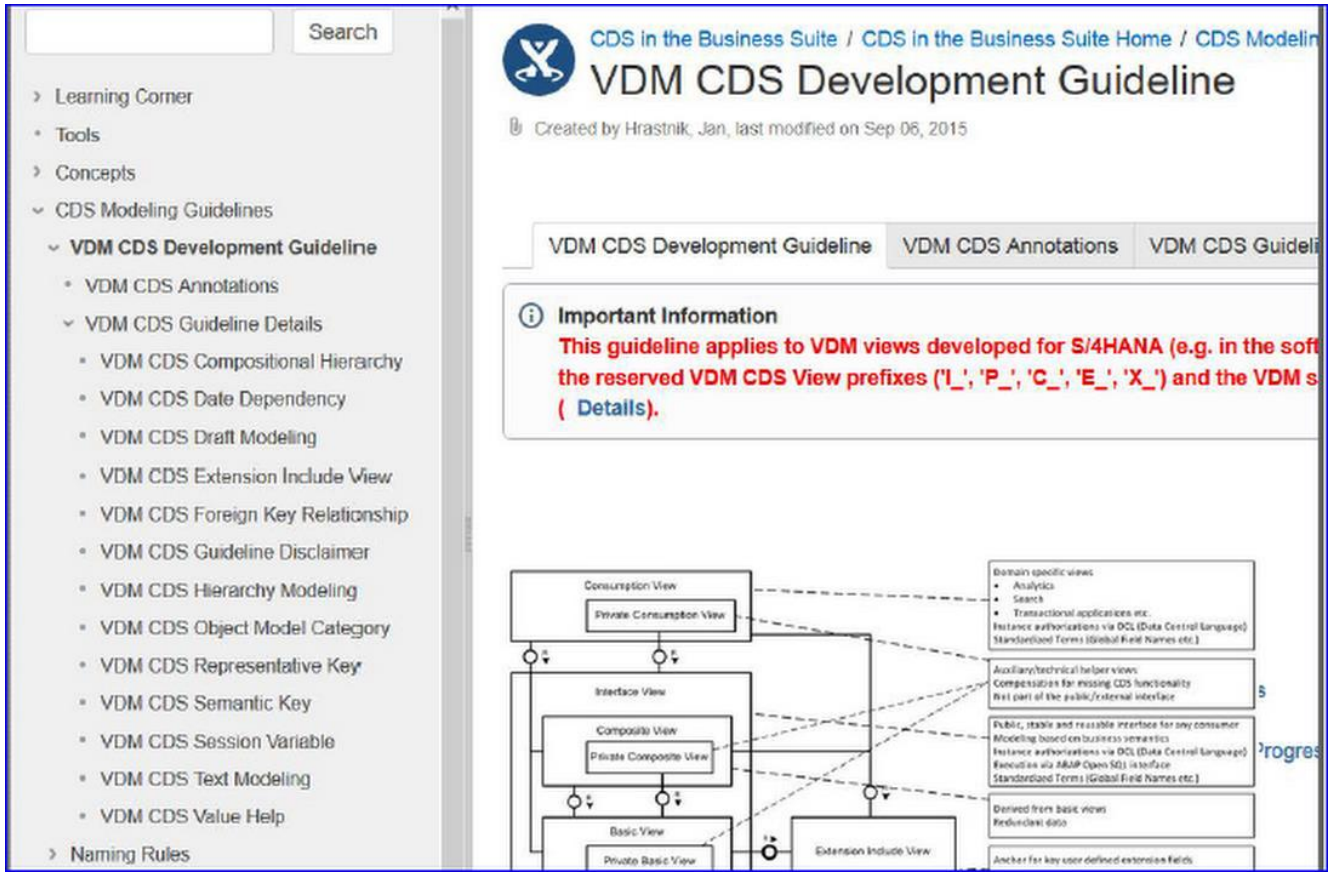
- ***@VDM.viewType***
  - Admissible Values: *#BASIC, #COMPOSITE, #CONSUMPTION, #EXTENSION*
  - In VDM, all CDS views shall be annotated explicitly with a VDM view type
- ***@VDM.private***
  - Admissible Values: #blank (=true), true or false
  - Only private views need to be annotated explicitly
- **@ObjectModel.dataCategory**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

VDM CDS Development Guideline

SAP Development Learning

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



VDM CDS ATC Checks

SAP Development Learning

- 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 correcly

More details can be found here:

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

ABAP CDS vs HANA CDS

SAP Development Learning

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

SAP Development Learning

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 Development Learning

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