SAP HANA

Lesson Name: Database Independent Code-to-Data

Lesson Objectives

After completing this lesson, participants will be able to –

- Know about basics of OPEN SQL
- Features of OPEN SQL
- New syntaxes and statements of OPEN SQL in SAP ABAP
- Performance rules and limitations of OPEN SQL
- Basics of Core Data Services (CDS)
- Demo on CDS
- CDS view Definition Features

Contents

Introduction to OPEN SQL

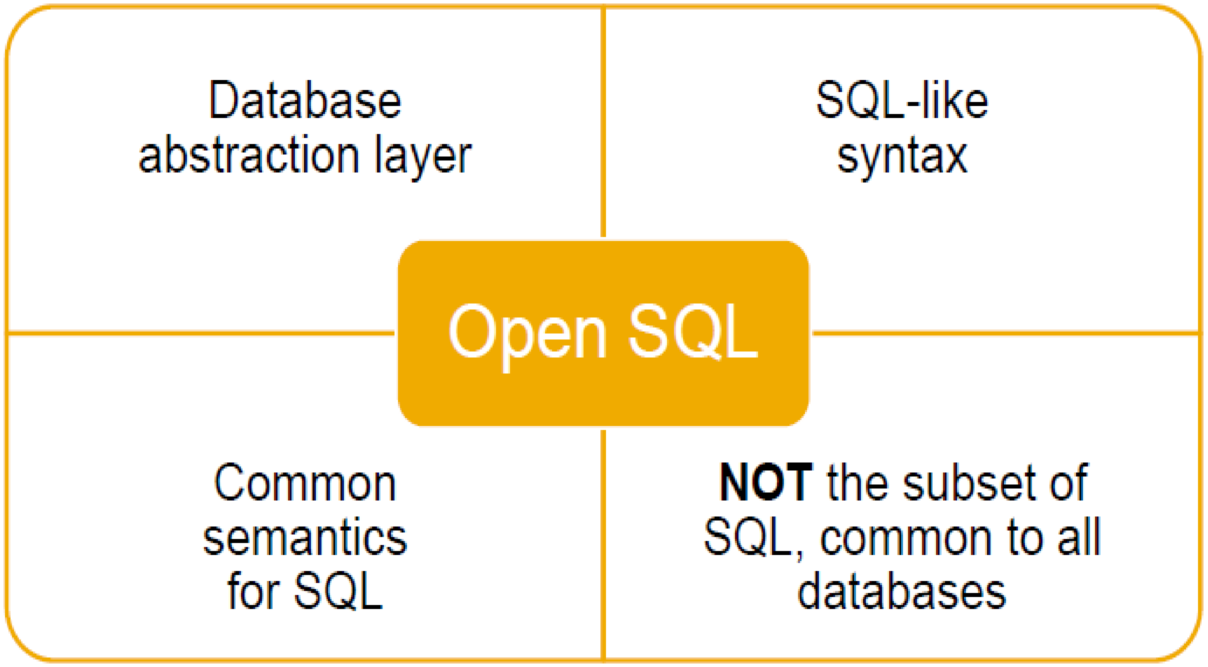
- Open SQL in our ABAP application server is the database abstraction layer calling an SQL like syntax.

- It is the database abstraction layer and actually the only database abstraction layer that has a common semantic for all of SAP supported databases.

- This is important, if we are talking about migration to SAP HANA/Database migration in general.

- No problem in migrating Open SQL from one database to another because it has the same semantics on all databases & you can use it as before.

Introduction to OPEN SQL


Database abstraction layer
SQL-like syntax
Open SQL
Common semantics for SQL
NOT the subset of SQL, common to all databases

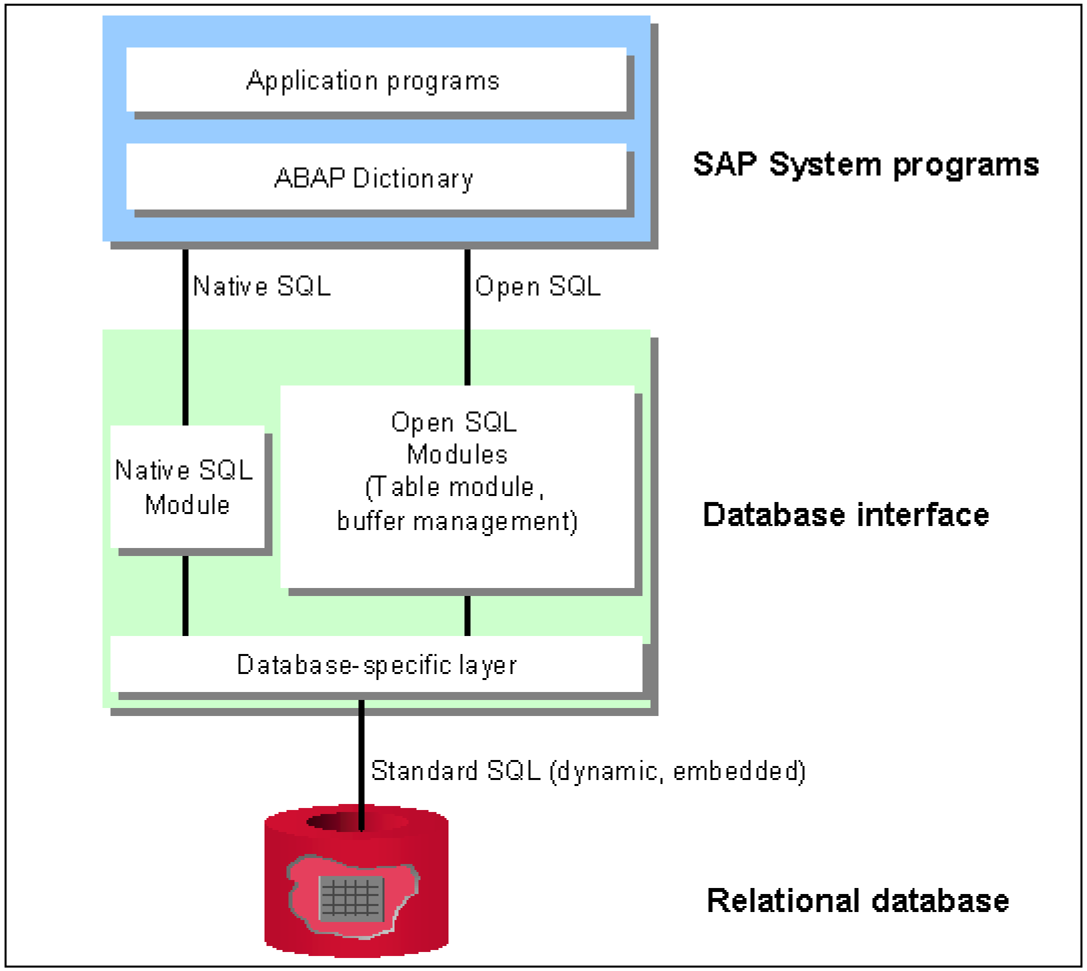
Open SQL is the only DB **abstraction layer** that defines a **common semantic** for all SAP-supported databases!

Introduction to OPEN SQL

Open SQL aspires to:

- Enable the application of the Code-to-Data paradigm
- Provide more standard SQL features
- Enable the consumption of SAP HANA-specific features
- With ABAP 7.4 and above, what's important is that Open SQL will really help you doing the code pushdown with a very easy way.

Introduction to OPEN SQL



Features of Open SQL in ABAP 7.4 SP2 and beyond.

Syntax enhancements:

Escaping of host variables

Comma-separated select list

SELECT list enhancements:

Aggregation functions

Literals

Arithmetical expressions

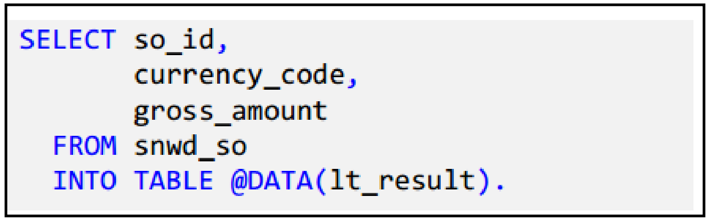
New Open SQL Syntax

- Comma separated element list
- Escaping of host variables
- Target type inference

Note:

Very important for you to know that you don't need to change your whole report now to the new SQL statement.

The old style will stay intact so you will still be able to use it.



```
SELECT so_id,
       currency_code,
       gross_amount
  FROM snwd_so
  INTO TABLE @DATA(lt_result).
```

New features of OPEN SQL:

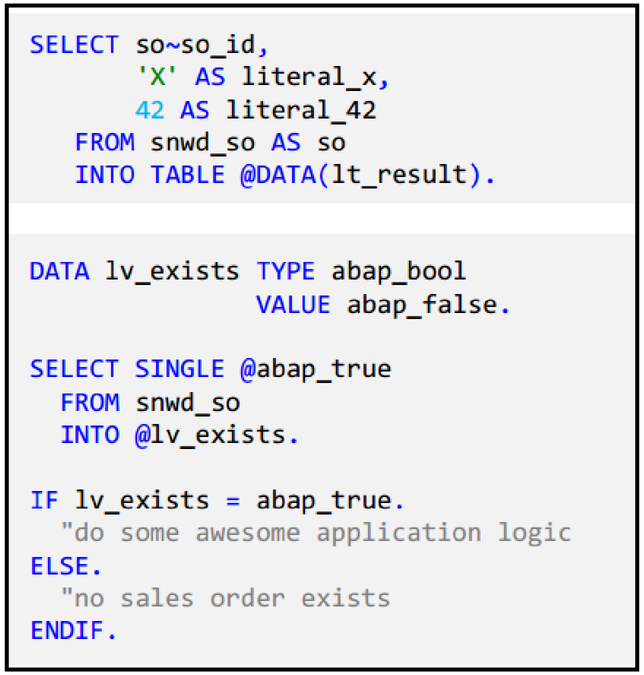
New SELECT List Features

- Aggregation functions
- Literal values
- Arithmetic expressions

New features of OPEN SQL:

Literal Values

- Can now be used in the SELECT list
- Allow for a generic implementation of an existence check



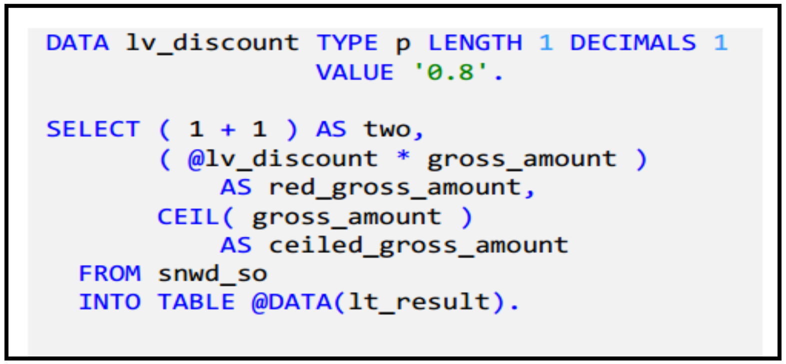
```
SELECT so~so_id,
       'X' AS literal_x,
       42 AS literal_42
   FROM snwd_so AS so
   INTO TABLE @DATA(lt_result).
```

```
DATA lv_exists TYPE abap_bool
               VALUE abap_false.

SELECT SINGLE @abap_true
  FROM snwd_so
  INTO @lv_exists.

IF lv_exists = abap_true.
  "do some awesome application logic
ELSE.
  "no sales order exists
ENDIF.
```

New features of OPEN SQL:

Arithmetic Expressions

- +, -, *, DIV, MOD, ABS, FLOOR, CEIL
- Remember: Open SQL defines a semantic for these expressions common to all supported databases
- Refer to the ABAP documentation to see which expression is valid for which types

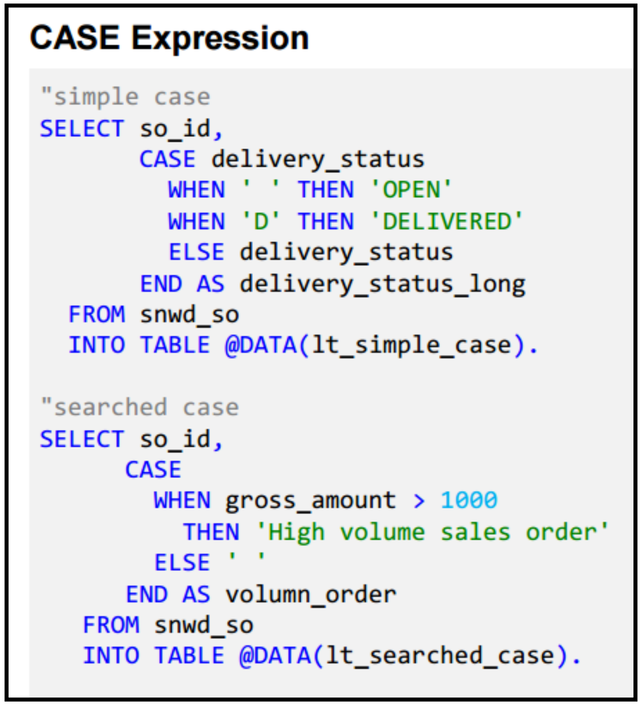


```
DATA lv_discount TYPE p LENGTH 1 DECIMALS 1
                 VALUE '0.8'.

SELECT ( 1 + 1 ) AS two,
       ( @lv_discount * gross_amount )
           AS red_gross_amount,
       CEIL( gross_amount )
           AS ceiled_gross_amount
  FROM snwd_so
  INTO TABLE @DATA(lt_result).
```

New features of OPEN SQL:

Open SQL enhancements

- SELECT list enhancements:
  - Conditional expressions



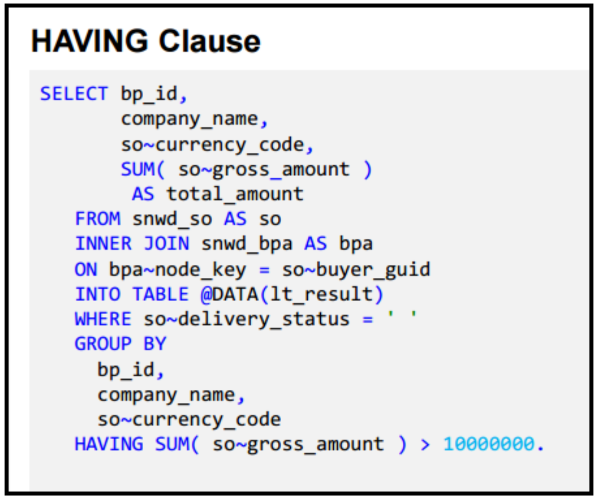
CASE Expression

```
"simple case
SELECT so_id,
       CASE delivery_status
         WHEN ' ' THEN 'OPEN'
         WHEN 'D' THEN 'DELIVERED'
         ELSE delivery_status
       END AS delivery_status_long
  FROM snwd_so
  INTO TABLE @DATA(lt_simple_case).

"searched case
SELECT so_id,
      CASE
        WHEN gross_amount > 1000
          THEN 'High volume sales order'
        ELSE ' '
      END AS volumn_order
   FROM snwd_so
   INTO TABLE @DATA(lt_searched_case).
```

New features of OPEN SQL:

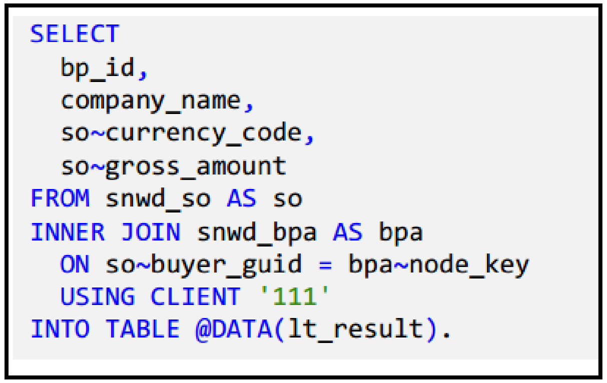
Open SQL enhancements

- Expressions in
  - HAVING clause
  - JOIN statements
  - Client handling



HAVING Clause

```
SELECT bp_id,
       company_name,
       so~currency_code,
       SUM( so~gross_amount )
        AS total_amount
   FROM snwd_so AS so
   INNER JOIN snwd_bpa AS bpa
   ON bpa~node_key = so~buyer_guid
   INTO TABLE @DATA(lt_result)
   WHERE so~delivery_status = ' '
   GROUP BY
     bp_id,
     company_name,
     so~currency_code
   HAVING SUM( so~gross_amount ) > 10000000.
```



```
SELECT
  bp_id,
  company_name,
  so~currency_code,
  so~gross_amount
FROM snwd_so AS so
INNER JOIN snwd_bpa AS bpa
  ON so~buyer_guid = bpa~node_key
  USING CLIENT '111'
INTO TABLE @DATA(lt_result).
```

List Of Open SQL Statements in SAP ABAP

The open SQL statements are:

INSERT

Insert record from internal table

Syntax:

INSERT <DB TABLE> FROM TABLE <INTERNAL TABLE>.

Insert record from work area

Syntax: INSERT <DB TABLE> FROM <WA>.

List Open SQL Statements in SAP ABAP

The open SQL statements are:

UPDATE

Update record from internal table

Syntax:

UPDATE <DB TABLE> FROM TABLE <INTERNAL TABLE>.

Update record from work area

Syntax: UPDATE <DB TABLE> FROM <WA>.

List Open SQL Statements in SAP ABAP

The open SQL statements are:

MODIFY

Update record from internal table

Syntax:

MODIFY <DB TABLE> FROM TABLE <INTERNAL TABLE>.

Update record from work area

Syntax: MODIFY <DB TABLE> FROM <WA>.

List Of Open SQL Statements in SAP ABAP

The open SQL statements are:

DELETE

Update record from internal table

Syntax:

DELETE <DB TABLE> FROM TABLE <INTERNAL TABLE>.

Update record from work area

Syntax: DELETE <DB TABLE> FROM <WA>.

Open SQL - Performance Rules:

To improve the performance of the SQL and in turn of the ABAP program, one should take care of the following rules-

Keep the Result Set Small

- Using the where clause
- If only one record is required from the database, use SELECT SINGLE whenever possible .

Open SQL - Performance Rules:

Minimize the Amount of Data Transferred

- Restrict the number of lines
- If only certain fields are required from a table, use the SELECT <field1> <field2> INTO ... statement
- Restrict no of columns
- Use aggregate functions

Using Internal Tables to Buffer Records

- To avoid executing the same SELECT multiple times (and therefore have duplicate selects), an internal table of type HASHED can be used to improve performance.

Open SQL - Performance Rules:

Minimize the Number of Data Transfers

- Avoid nested select loops
- An alternative option is to use the SELECT .. FOR ALL ENTRIES statement. This statement can often be a lot more efficient than performing a large number of SELECT or SELECT SINGLE statements during a LOOP of an internal table.
- Use dictionary views
- Use Joins in the FROM clause
- Use subqueries in the where clause

Open SQL - Performance Rules:

Minimize the Search Overhead

- Use index fields in the where clause
- When accessing databases, always ensure that the correct index is being used .

Reduce the Database Load

- Buffering
- Logical databases
- Avoid repeated database access

Limitations of OPEN SQL:

In OPEN SQL we can only use reference tables that are managed by/in the “ABAP dictionary”.

OPEN SQL does not support DML statements like create table.

OPEN SQL does not support “advanced” SQL statements like TRUNCATE, MERGE, ROLLUP.

In OPEN SQL you can’t use aggregate functions like sum,avg.

Limitations of OPEN SQL:

OPEN SQL does not support most column functions like SUBSTR, CONCAT (||) and “case expression” in both the select and where clauses.

OPEN SQL does not allow you to write predicates (where conditions) between more than one colum … so you can’t write the following condition:
where t1.col1 <> t1.col2

Limitations of OPEN SQL:

OPEN SQL does not allow to write predicates

(where conditions) on columns of a table joined with left (or right) join. So we can’t write the following SQL:

```
select … from t1 left join t2
where t2.col = ‘x’
```

Introduction to Core Data Services (CDS)

With the availability of the SAP HANA platform there has been a paradigm shift in the way business applications are developed at SAP. The rule-of-thumb is simple: ***Do as much as you can in the database to get the best performance***.

CDS is a data modeling infrastructure for defining and consuming semantic and reusable data models on the database, rather than on the ABAP server, regardless of the database system used.

Introduction to Core Data Services (CDS)

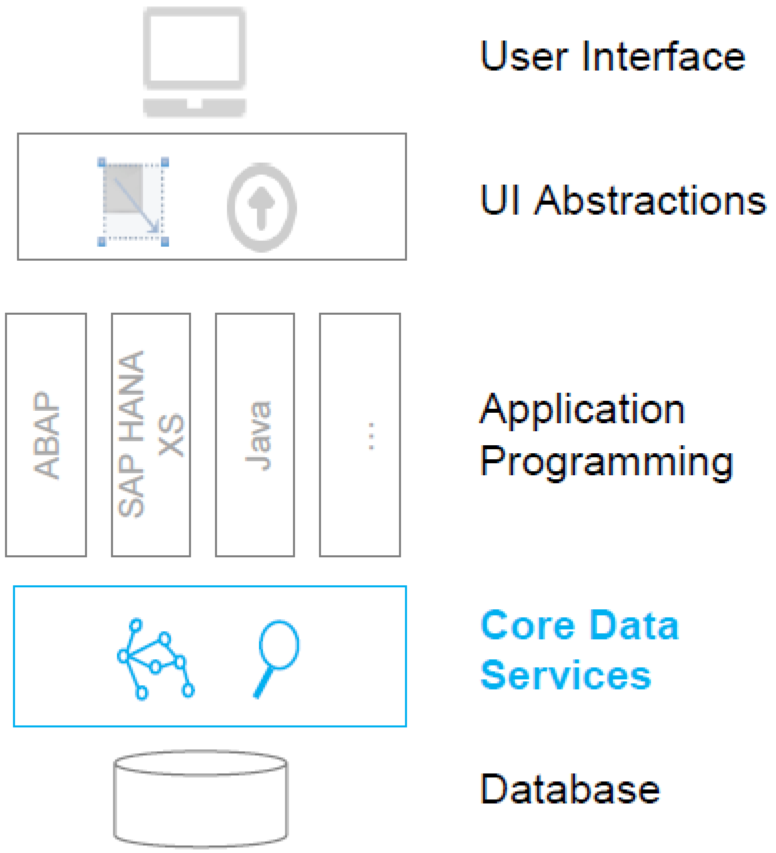
Technically, it is an enhancement of SQL which provides you with a data definition language (DDL) for defining semantically rich database tables/views (CDS entities) and user-defined types in the database.

CDS entities and their metadata are extensible and optimally integrated into the ABAP Data Dictionary and the ABAP language.

Introduction to Core Data Services (CDS)

Core Data Services

- Next generation of data definition and access for database-centric applications
- Optimized application programming model for all domains (transactional, analytical,…)
- Technically an extension to SQL:
  - Expressions
  - Domain-specific metadata
  - Associations
- CDS includes
  - Data Definition Language (**DDL**)
  - Query language (**QL**)
  - Data Manipulation Language (**DML**)
  - Data control language (**DCL**)



Introduction to Core Data Services (CDS)

Code-to-Data paradigm

- Supported through extended view functionality

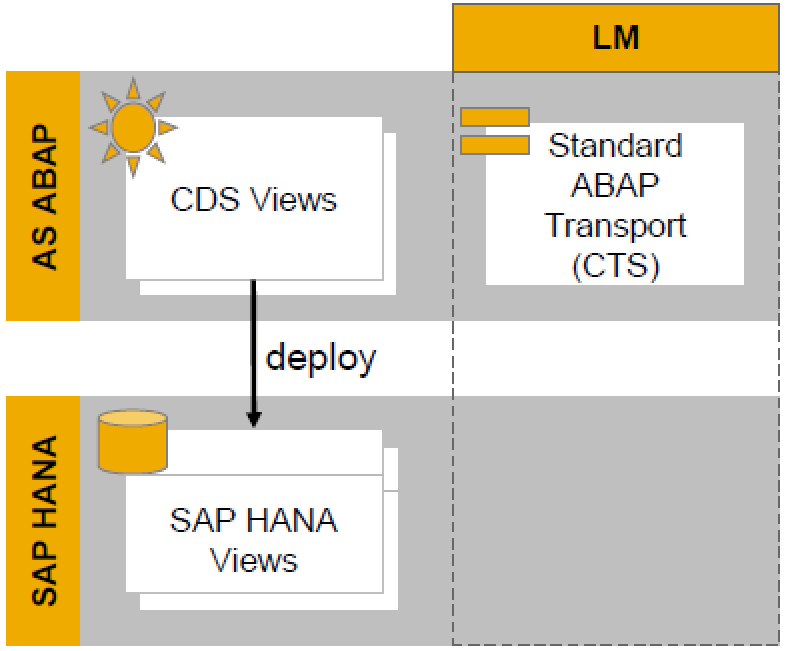
Definition of semantically rich data models in the ABAP Dictionary

- ABAP 'view entities' in DDL source objects (`R3TR DDLS`)

Fully integrated into the ABAP infrastructure

- Consistent lifecycle management with all other ABAP artifacts

Consumption via Open SQL on view entities



CDS in ABAP

<ins>Advantages</ins>

Semantically rich data models, i.e. CDS builds on the well-known entity relationship model and is declarative in nature, very close to conceptual thinking.

Compatibility across any database platform, i.e. CDS is generated into managed Open SQL views and is natively integrated into the SAP HANA layer.

CDS in ABAP

Advantages

Efficiency, i.e. CDS offers a variety of highly efficient built-in functions — such as SQL operators, aggregations, and expressions — for creating views.

Extensibility, i.e. Customers can extend SAP-defined CDS views with fields that can be automatically added to the CDS view.

Demo on CDS

ABAP CDS View Demo

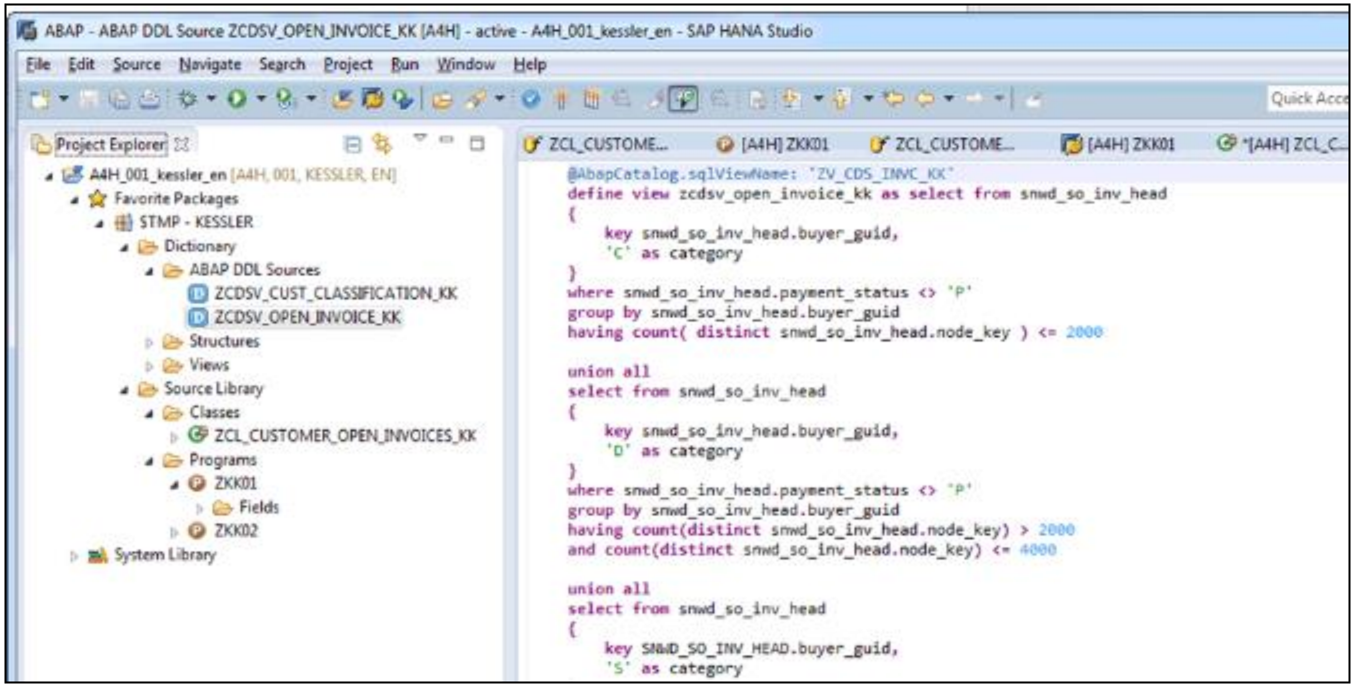
Advanced View Definition in ABAP

Data Preview

Open SQL Consumption

Demo on CDS

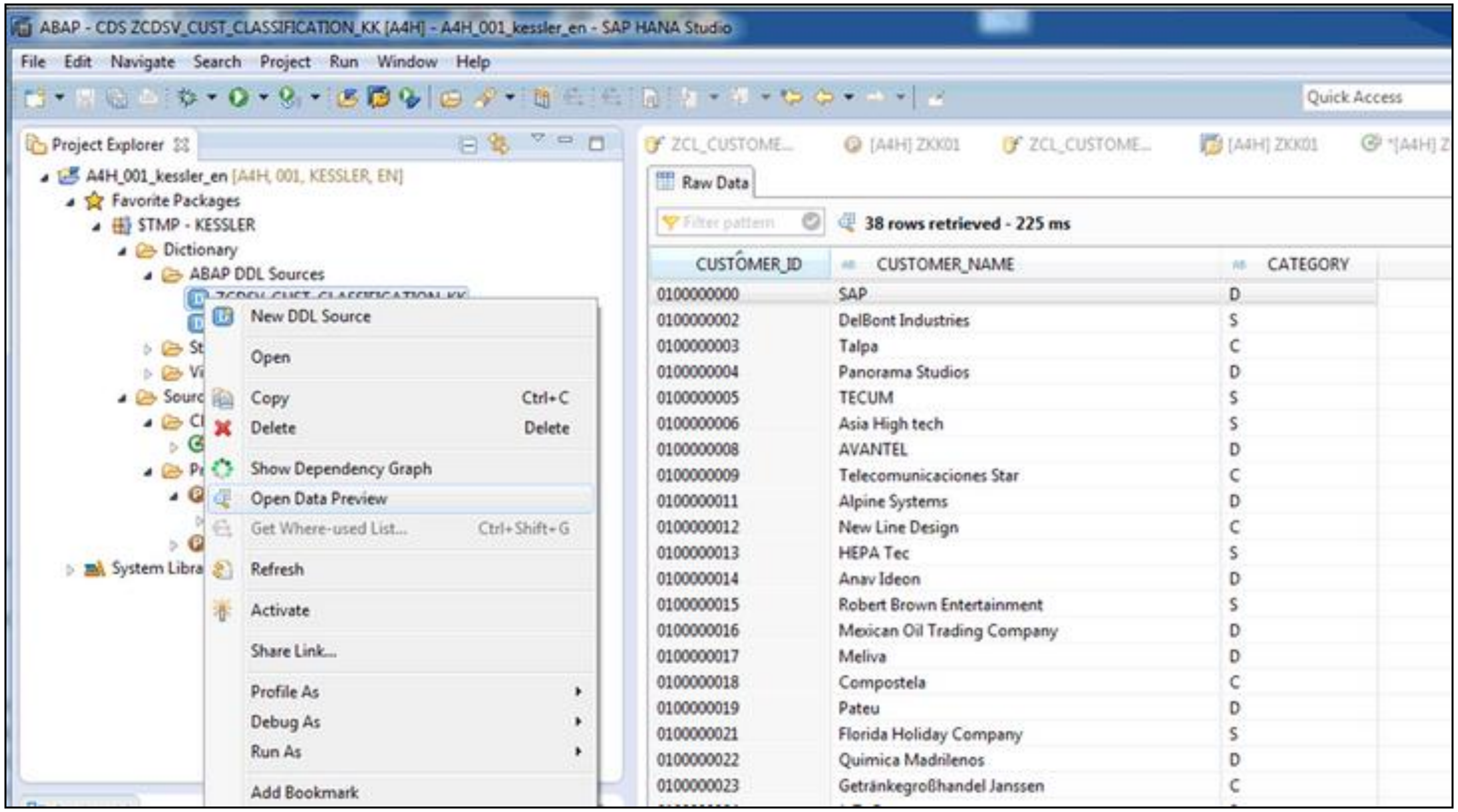
Advanced View Definition in ABAP


ABAP - ABAP DDL Source ZCDSV_OPEN_INVOICE_KK [A4H] - active - A4H_001_kessler_en - SAP HANA Studio
File Edit Source Navigate Search Project Run Window Help
Quick Acce
Project Explorer
A4H_001_kessler_en [A4H, 001, KESSLER, EN]
Favorite Packages
STMP - KESSLER
Dictionary
ABAP DDL Sources
ZCDSV_CUST_CLASSIFICATION_KK
ZCDSV_OPEN_INVOICE_KK
Structures
Views
Source Library
Classes
ZCL_CUSTOMER_OPEN_INVOICES_KK
Programs
ZKK01
Fields
ZKK02
System Library
ZCL_CUSTOME...
[A4H] ZKK01
ZCL_CUSTOME...
[A4H] ZKK01
*[A4H] ZCL_C...

```
@AbapCatalog.sqlViewName: 'ZV_CDS_INVC_KK'
define view zcdsv_open_invoice_kk as select from snwd_so_inv_head
{
    key snwd_so_inv_head.buyer_guid,
    'C' as category
}
where snwd_so_inv_head.payment_status <> 'P'
group by snwd_so_inv_head.buyer_guid
having count( distinct snwd_so_inv_head.node_key ) <= 2000

union all
select from snwd_so_inv_head
{
    key snwd_so_inv_head.buyer_guid,
    'D' as category
}
where snwd_so_inv_head.payment_status <> 'P'
group by snwd_so_inv_head.buyer_guid
having count(distinct snwd_so_inv_head.node_key) > 2000
and count(distinct snwd_so_inv_head.node_key) <= 4000

union all
select from snwd_so_inv_head
{
    key SNWD_SO_INV_HEAD.buyer_guid,
    'S' as category
```

Demo on CDS

Data Preview

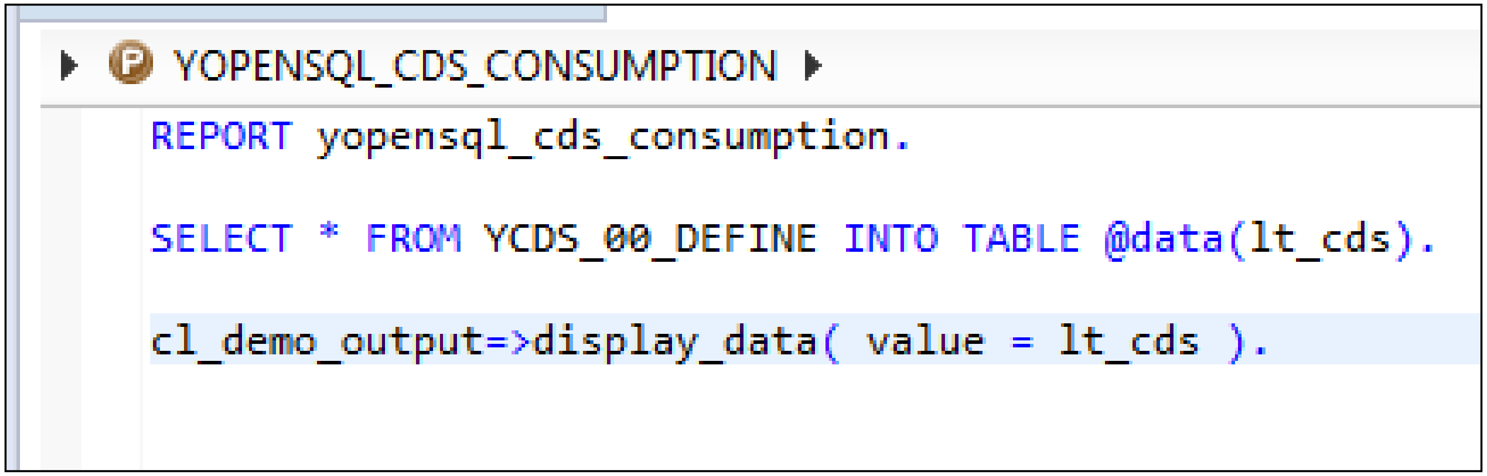


38 rows retrieved - 225 ms

CUSTOMER_ID	CUSTOMER_NAME	CATEGORY
0100000000	SAP	D
0100000002	DelBont Industries	S
0100000003	Talpa	C
0100000004	Panorama Studios	D
0100000005	TECUM	S
0100000006	Asia High tech	S
0100000008	AVANTEL	D
0100000009	Telecomunicaciones Star	C
0100000011	Alpine Systems	D
0100000012	New Line Design	C
0100000013	HEPA Tec	S
0100000014	Anav Ideon	D
0100000015	Robert Brown Entertainment	S
0100000016	Mexican Oil Trading Company	D
0100000017	Meliva	D
0100000018	Compostela	C
0100000019	Pateu	D
0100000021	Florida Holiday Company	S
0100000022	Quimica Madrilenos	D
0100000023	Getränkegroßhandel Janssen	C

Demo on CDS

Consumption of CDS View



YOPENSQL_CDS_CONSUMPTION

```
REPORT yopensql_cds_consumption.

SELECT * FROM YCDS_00_DEFINE INTO TABLE @data(lt_cds).

cl_demo_output=>display_data( value = lt_cds ).
```

Demo on CDS

Definition & Consumption of an ABAP CDS View

Definition in an ABAP DDL Source (R3TR DDLS)

Definition only possible with ABAP Development Tools in Eclipse/HANA Studio (not via transaction SE11)

Consumption via

Open SQL

Data Preview (context menu in ADT)

SAP List Viewer

SAP NetWeaver Gateway (OData Model)

CDS View Definition Features

CDS View Definition Features

Projection List:

- Client Dependency
- Semantic Information (Key)
- Aliases
- Aggregation
- Literals
- Arithmetic Expressions
- Conditional Expressions

View-on-View

CDS View Extensions

CDS View with Input Parameters

CDS View Definition Features

ABAP CDS View: Projection List

Client-dependent view; no explicit client field necessary

Semantic information (key field)

Aliases

Literal values:

- C-sequence literals (Max length: 1333)
- Signed integer literals (4-Byte)

Aggregation functions:

- MIN, MAX, COUNT, AVG, SUM
- Alias required for function results

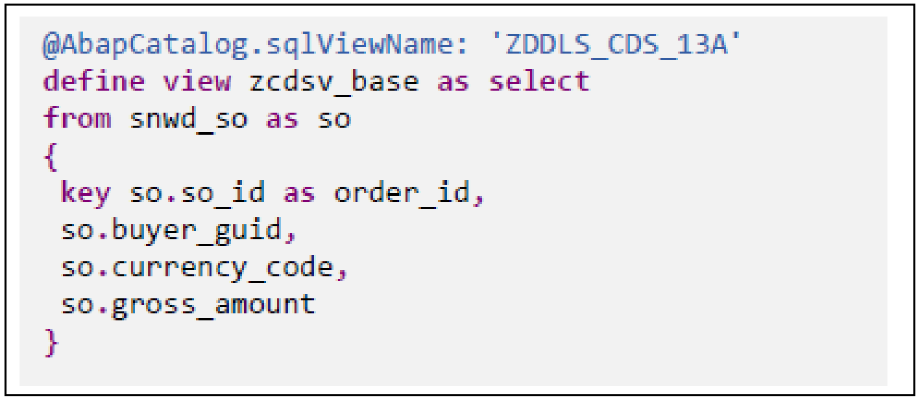
String functions:

- LPAD,SCORE,LEFT,LTRIM,SUBSTRING
- Alias required for function results

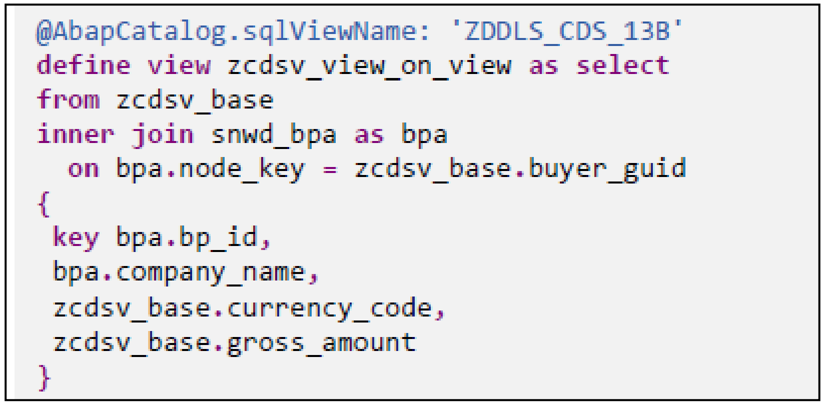
CDS View Definition Features

View-on-View

- View can have other views as data basis
- No restriction on the number of layers



```
@AbapCatalog.sqlViewName: 'ZDDLS_CDS_13A'
define view zcdsv_base as select
from snwd_so as so
{
 key so.so_id as order_id,
 so.buyer_guid,
 so.currency_code,
 so.gross_amount
}
```

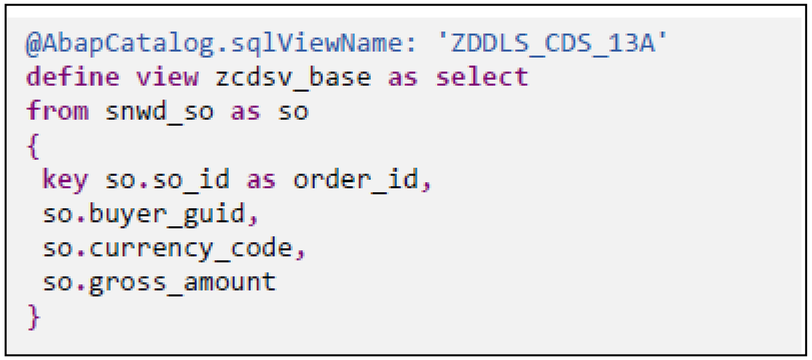


```
@AbapCatalog.sqlViewName: 'ZDDLS_CDS_13B'
define view zcdsv_view_on_view as select
from zcdsv_base
inner join snwd_bpa as bpa
  on bpa.node_key = zcdsv_base.buyer_guid
{
 key bpa.bp_id,
 bpa.company_name,
 zcdsv_base.currency_code,
 zcdsv_base.gross_amount
}
```

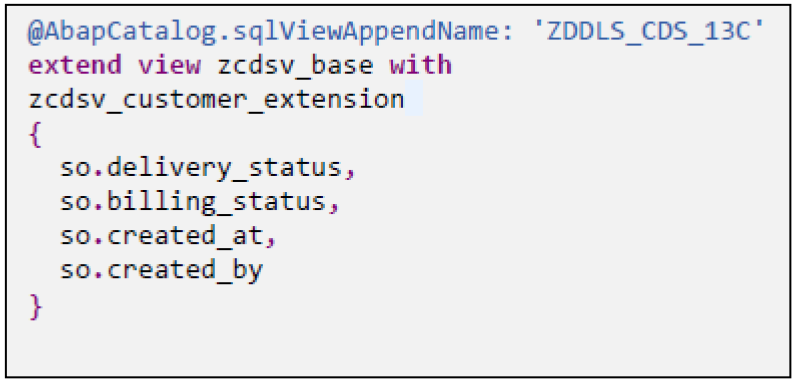
CDS View Definition Features

CDS View Extensions

- Extend base views with new fields



```
@AbapCatalog.sqlViewName: 'ZDDLS_CDS_13A'
define view zcdsv_base as select
from snwd_so as so
{
 key so.so_id as order_id,
 so.buyer_guid,
 so.currency_code,
 so.gross_amount
}
```

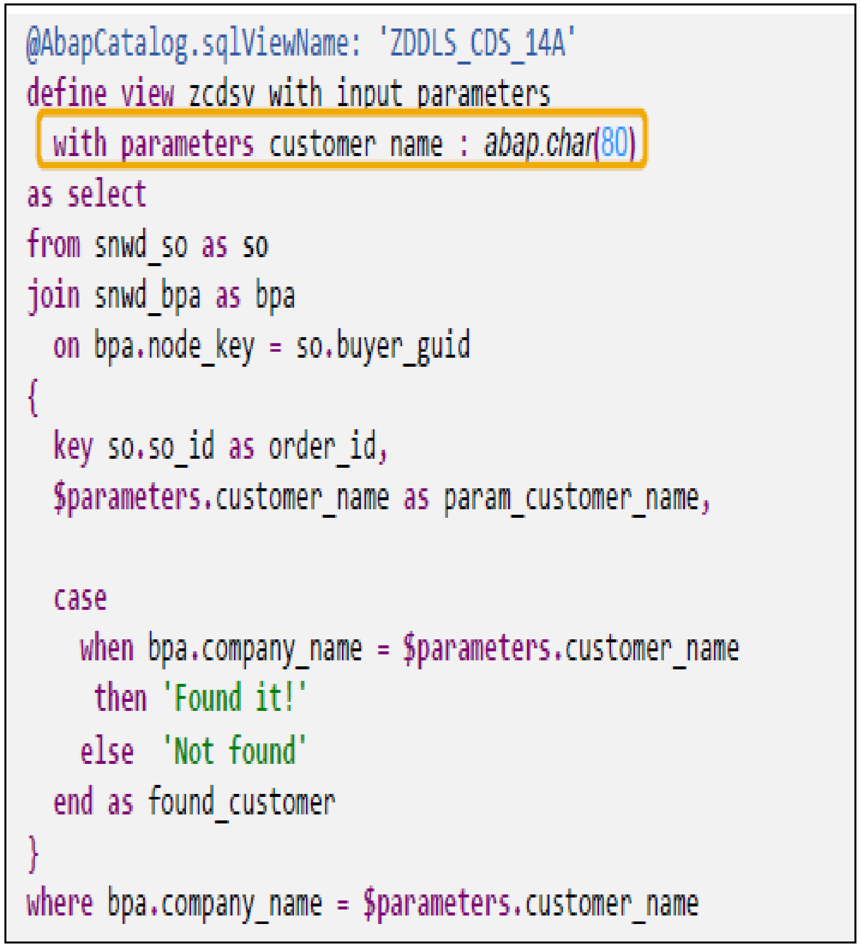


```
@AbapCatalog.sqlViewAppendName: 'ZDDLS_CDS_13C'
extend view zcdsv_base with
zcdsv_customer_extension
{
   so.delivery_status,
   so.billing_status,
   so.created_at,
   so.created_by
}
```

CDS View Definition Features

CDS View with input parameters

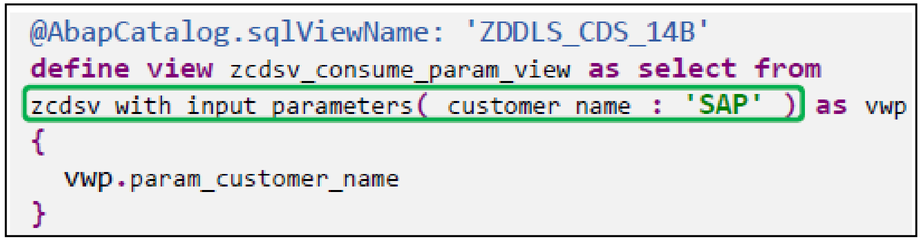
- Comma-separated list of scalar input parameters and corresponding type
- Supported parameter types:
    - Predefined data type like abap.char(char_len)
    - Name of a data element



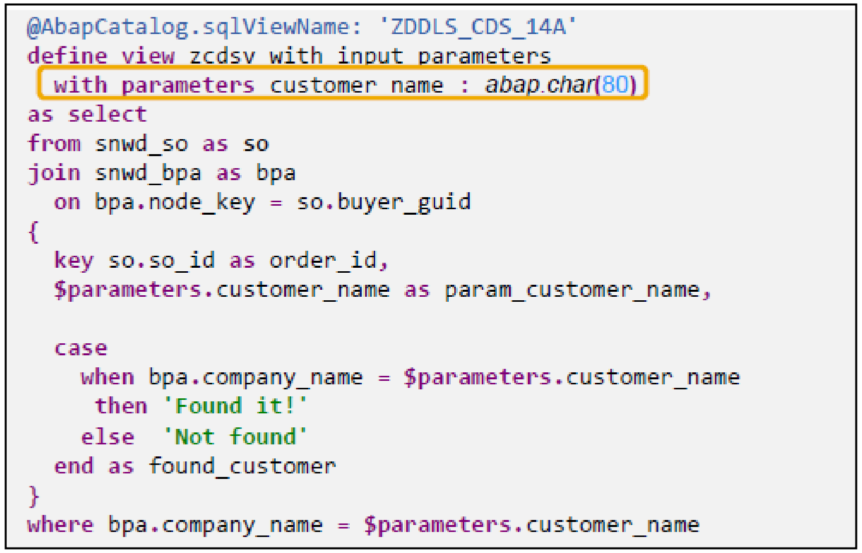
```
@AbapCatalog.sqlViewName: 'ZDDLS_CDS_14A'
define view zcdsv_with_input_parameters
  with parameters customer_name : abap.char(80)
as select
from snwd_so as so
join snwd_bpa as bpa
  on bpa.node_key = so.buyer_guid
{
  key so.so_id as order_id,
  $parameters.customer_name as param_customer_name,

  case
    when bpa.company_name = $parameters.customer_name
     then 'Found it!'
    else  'Not found'
  end as found_customer
}
where bpa.company_name = $parameters.customer_name
```

CDS View Definition Features

- Consumption in a CDS View



```
@AbapCatalog.sqlViewName: 'ZDDLS_CDS_14B'
define view zcdsv_consume_param_view as select from
zcdsv_with_input_parameters( customer_name : 'SAP' ) as vwp
{
  vwp.param_customer_name
}
```



```
@AbapCatalog.sqlViewName: 'ZDDLS_CDS_14A'
define view zcdsv_with_input_parameters
  with parameters customer_name : abap.char(80)
as select
from snwd_so as so
join snwd_bpa as bpa
  on bpa.node_key = so.buyer_guid
{
  key so.so_id as order_id,
  $parameters.customer_name as param_customer_name,

  case
    when bpa.company_name = $parameters.customer_name
     then 'Found it!'
    else  'Not found'
  end as found_customer
}
where bpa.company_name = $parameters.customer_name
```

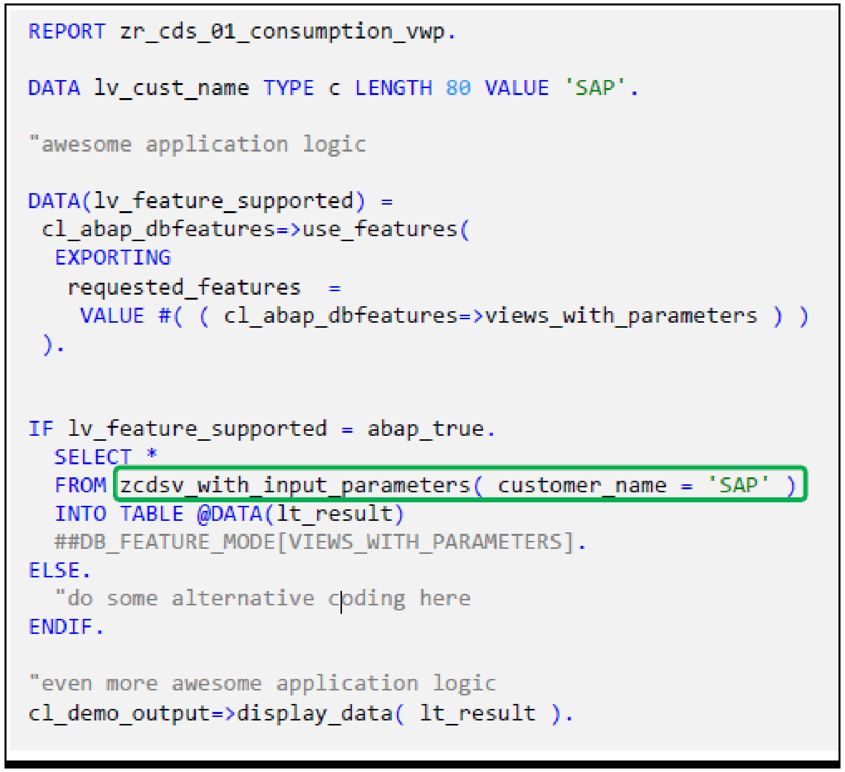
CDS View Definition Features

Consumption via Open SQL

- Check if the feature is supported
- Provide (mandatory) input parameter(s)
- Suppress syntax warning using the pragma
- Provide a “fallback‟ implementation / some error handling

CDS View Definition Features

Consumption via OpenSQL



```
REPORT zr_cds_01_consumption_vwp.

DATA lv_cust_name TYPE c LENGTH 80 VALUE 'SAP'.

"awesome application logic

DATA(lv_feature_supported) =
 cl_abap_dbfeatures=>use_features(
  EXPORTING
   requested_features  =
    VALUE #( ( cl_abap_dbfeatures=>views_with_parameters ) )
 ).


IF lv_feature_supported = abap_true.
  SELECT *
  FROM zcdsv_with_input_parameters( customer_name = 'SAP' )
  INTO TABLE @DATA(lt_result)
  ##DB_FEATURE_MODE[VIEWS_WITH_PARAMETERS].
ELSE.
  "do some alternative coding here
ENDIF.

"even more awesome application logic
cl_demo_output=>display_data( lt_result ).
```

Summary

In this lesson, you have learnt:

- Basic Concepts of Open SQL
- Features of Open SQL
- Open SQL Syntaxes and Statements
- Performance Rules and Limitations of Open SQL
- About Core Data Services
- CDS in ABAP
- Demos on CDS
- CDS View Definition Features

Review Questions

OPEN SQL Statements are those statements which are used to -------- or --------- database table data.

For OPEN SQL statements insertion in database table is possible in --------- way/ways.

Open SQL in ABAP application server is the ------- ---------layer calling an SQL like syntax.